

# REORGANIZATION IN MULTIAGENT ORGANIZATIONS

by

**SHAM KASHYAP**

B. E., Mangalore University, India, 2002

---

A THESIS

submitted in partial fulfillment of the  
requirements for the degree

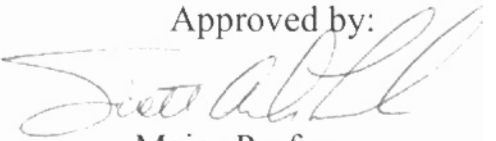
MASTER OF SCIENCE

Department of Computing and Information Sciences  
College of Engineering

KANSAS STATE UNIVERSITY  
Manhattan, Kansas

2006

Approved by:



Major Professor  
Scott A. DeLoach, Ph.D.

## **ABSTRACT**

In this thesis, I use the search and rescue problem to illustrate the design and implementation of a multiagent organization. I examine different reorganization strategies for this organization and evaluate their performance. I discuss the shortcomings of hand-coded reorganization strategies and present a genetic programming based approach for devising reorganization strategies. This approach allows us to utilize non-intuitive environmental knowledge about the problem in arriving at effective reorganization strategies. The results suggest that while there are significant costs involved in using this method, there is a definite boost in the performance of the organization.

# TABLE OF CONTENTS

LIST OF FIGURES .....	iv
LIST OF TABLES .....	v
ACKNOWLEDGEMENTS .....	vi
Chapter 1 Introduction .....	7
1.1 Problem.....	7
1.1.1 Hypothesis .....	8
1.2 Background.....	8
1.2.1 Multiagent Systems .....	8
1.2.2 Multiagent Organizations .....	9
1.2.3 Genetic Programming.....	11
1.3 Scope and Objectives.....	13
1.4 Document Overview .....	14
Chapter 2 Literature Review .....	15
2.1 Multiagent Organizations .....	15
2.2 Learning in Multiagent Systems .....	16
2.3 Summary.....	18
Chapter 3 Methodology and Design .....	19
3.1 Search and Rescue Simulator .....	19
3.1.1 Objects.....	19
3.1.2 Agents.....	19
3.1.3 Information sharing and triggers .....	20
3.2 Design of the Search and Rescue Organization.....	21
3.2.1 Goals.....	21
3.2.2 Sequence Diagram.....	23
3.2.3 Roles .....	23
3.2.4 Agents.....	26
3.2.5 Scoring.....	27
3.3 The Search and Rescue Organization Problem.....	29
3.3.1 Solution Strategies .....	29
3.3.2 Hand coded solution strategy .....	34
3.3.3 Evolutionary reorganization strategy.....	34
3.4 Summary of Methodology and Design.....	38
Chapter 4 Implementation.....	39

4.1 Simulator .....	39
4.1.1 Agent Manager .....	39
4.1.2 Black Board .....	39
4.1.3 Rescue Agent.....	40
4.1.4 Searcher Role.....	41
4.1.5 Rescuer Role.....	41
4.1.6 Organizational Master .....	41
4.2 Simulator Runs .....	42
4.3 Implementation of reorganization strategies.....	42
4.3.1 Hand coded strategies.....	43
4.3.2 Evolutionary reorganization strategies .....	44
4.4 Summary of Implementation .....	45
Chapter 5 Results .....	46
5.1 Introduction .....	46
5.2 Setup .....	46
5.2.1 Time constraints of experiments.....	47
5.3 Hand-coded strategies.....	47
5.3.1 Real-life based scenarios .....	49
5.4 Evolutionary reorganization strategies .....	51
5.4.1 Scenario 1 .....	51
5.4.2 Scenario 2 .....	53
5.4.3 Scenario 3 .....	54
5.4.4 Scenario 4 .....	56
5.4.5 Scenario 5 .....	57
5.4.6 Scenario 6 .....	58
5.4.7 Scenario 7 .....	59
5.4.8 Scenario 8 .....	60
5.4.9 All scenarios .....	62
5.5 Summary of results.....	64
Chapter 6 Conclusions and Future Work.....	66
6.1 Conclusions .....	66
6.2 Future Work.....	67
REFERENCES .....	68
Appendix A Users Manual.....	71
A.1 Overview.....	71

A.2 Running the software .....	71
A.2.1 Scores .....	71
Appendix B Reorganization Strategies .....	72
B.1 Scores .....	72
B.2 S-expressions of evolutionary reorganization strategies .....	72
B.2.1 Best Individual of Scenario 1 .....	72
B.2.2 Best Individual of Scenario 2 .....	73
B.2.3 Best Individual of Scenario 3 .....	73
B.2.4 Best Individual of Scenario 4 .....	73
B.2.5 Best Individual of Scenario 5 .....	74
B.2.6 Best Individual of Scenario 6 .....	74
B.2.7 Best Individual of Scenario 7 .....	74
B.2.8 Best Individual of Scenario 8 .....	75
B.2.9 Best Individual of all Scenarios .....	75
B.3 Best Fitness and Average Graphs .....	75
B.3.1 Scenario 1 .....	75
B.3.2 Scenario 2 .....	76
B.3.3 Scenario 3 .....	76
B.3.4 Scenario 4 .....	76
B.3.5 Scenario 5 .....	77
B.3.6 Scenario 6 .....	77
B.3.7 Scenario 7 .....	77
B.3.8 Scenario 8 .....	78
B.3.9 All Scenarios .....	78

## LIST OF FIGURES

Figure 1. An individual program represented as a tree .....	12
Figure 2. Search And Rescue Simulator .....	20
Figure 3 Simplified Artificial Organization Model .....	22
Figure 4. Goal Diagram .....	22
Figure 5. Sequence Diagram.....	24
Figure 6. Role Diagram.....	25
Figure 7. Scenario 1 .....	30
Figure 8. Scenario 2 .....	30
Figure 9. Scenario 3 .....	31
Figure 10. Scenario 4 .....	31
Figure 11. Scenario 5 .....	32
Figure 12. Scenario 6 .....	32
Figure 13. Scenario 7 .....	33
Figure 14. Scenario 8 .....	33
Figure 15. An evolutionary reorganization strategy .....	38
Figure 16. Static Structure .....	40
Figure 17. Extended hand coded strategy .....	44
Figure 18. An individual reorganization strategy .....	45
Figure 19. Best Individual for Scenario 1 .....	53
Figure 20. Best Individual for Scenario 2 .....	54
Figure 21. Best Individual for Scenario 3 .....	55
Figure 22. Best Individual for Scenario 4 .....	56
Figure 23. Best Individual for Scenario 5 .....	57
Figure 24. Best Individual of Scenario 6 .....	59
Figure 25. Best Individual for Scenario 7 .....	60
Figure 26. Best Individual for Scenario 8 .....	61
Figure 27. Best Individual for all scenarios .....	62

## LIST OF TABLES

Table 1. Terminal Set.....	35
Table 2. Function Set .....	36
Table 3. Control Parameters .....	37
Table 4. Hand-coded Strategies - Scores .....	48
Table 5. Hand-coded Strategies - Scores .....	49
Table 6. Hand-coded Strategies – Average Scores over all scenarios .....	50
Table 7. Hand-coded Strategies - extended cycles .....	51
Table 8. Best scores for individual scenarios.....	52
Table 9. Score of overall best strategy .....	63
Table 10. Average score of all strategies over all scenarios .....	64
Table 11. Performance of best individuals in all scenarios.....	72

## **ACKNOWLEDGEMENTS**

I would like to thank my major professor Dr. Scott A. DeLoach for his patient guidance throughout this research. Thanks to Dr. David A. Gustafson for serving on my committee. Special thanks to Dr. William Hsu for serving on my committee and introducing me to Artificial Intelligence. I also thank the members of the MACR lab and the GECKIES group at the KDD lab for their constructive comments and inputs to this research.

I am immensely grateful to my mentor, Dr. Bhadriraju Subramanyam at the department of Grain Science and Industry at K-State for his support and guidance throughout my stay at Manhattan. Many thanks to my friends for being there when I needed them the most.

Above all, without the encouragement and support of my family, I would not have finished this work.



# Chapter 1 Introduction

Agents are seen as the necessary software paradigm for realizing massive open distributed systems. But, as the complexity of software has increased, it has become harder to handle the complexity with single agent frameworks. Since the mid nineties, multiagent systems have received widespread attention in many fields of science and engineering. A multiagent system can be characterized by a group of interacting, self-directed agents having varied sensory and motor abilities. The idea of many agents working simultaneously opens up a wide array of natural, social and technical problems that can be understood by looking at them as artificial social organizations [39]. As the traditional ways of software design have given way to multiagent systems, the conventional fields of Distributed Artificial Intelligence (DAI) and Machine Learning (ML) have increasingly merged together and Learning in multiagent environments is considered as an important area of research and application in Artificial Intelligence [12].

An artificial organization by its nature demands adaptable and collective decision making capability from its agents. In this study, I look at how a machine learning approach can be incorporated into the design of such an organization to enhance its adaptability to the varying states of the organization.

The rest of this chapter is organized as follows. In Section 1.1, I state the problem statement and my hypothesis in this study. In the next Section, I briefly summarize the different topics in multiagent systems and machine learning that are related to this study. This is followed by the scope and objectives statements and the document overview for the rest of this thesis.

## 1.1 Problem

In any realistic setting, agents within the system can perceive the state of the environment only locally and only to the best of their ability. The information about the environment is hence incomplete and often incorrect leading to individual failures. In such cases, during the course of its existence, a multiagent system has to reorganize itself depending on the successes and failures of its agents if it has to be able to achieve its goals. Hence, the performance of the system is directly dependent on its ability to reorganization effectively. To reorganize, the multiagent system should be capable of

- detecting changes within itself and the environment and
- making necessary changes within its composition.

This requires that the multiagent system have a design that allows the programmer to specify the structure of the organization of agents, the goals that the organizations has to accomplish and the roles that agents have to play to achieve them, so that agents can recognize and distribute the necessary tasks within themselves efficiently. It also has to know of a way to adapt; by way of reorganizing itself when a change occurs in the environment or within the agent system.

### **1.1.1 Hypothesis**

Changes in the composition and performance of agents within an organization and changes in the environment in which the organization operates have complex effects on the performance of the organization. Understanding these effects is vital to the organization's adaptability to these changing circumstances. Hence an organization that uses a reorganization strategy that uses both agent specific information and environment specific information performs better than other organizations that do not use such a strategy.

Since the effect of such information can not be determined *a priori*, an evolutionary approach that over time produces strategies that make use of relevant information performs better than other hand-coded strategies that make use of predetermined information.

In this study, I attempt to design an organization of agents following [8] and develop reorganization strategies that use different information criterion and show that the above hypothesis holds good for a search and rescue application. This application lends itself well for testing different multiagent designs [1,4,10]. It involves a group of agents whose goal is to search for victims that are scattered in a bounded area and bring them to a designated safe area. The design and implementation of the search and rescue simulator is discussed in detail in Chapter 3.

## **1.2 Background**

In this Section, I explain the concepts and terminologies I have used in the design and implementation chapters. I briefly summarize multiagent systems, multiagent organizations, the OMACS and the GMoDS models. A summary of genetic programming and its terminologies is presented in Section 1.2.3. A detailed literature review of related topics follows in Chapter 2.

### **1.2.1 Multiagent Systems**

Multiagent systems can be defined as a group of agents that coexist and potentially interact with each other [36]. The design of multiagent systems necessarily involves the design of individual agents and the design of a framework in which these agents interact and realize their collective goals. Many development approaches like MaSE [9], GAIA [40], MESSAGE [25] and

Cassiopeia [13] have been developed to capture the relationships between agents and the how they coordinate with each other in the design of multiagent systems. As systems become increasingly complex, multiagent systems are not only expected to work together as a group to accomplish their goals but are also expected to exhibit intelligent social behavior that is seen in any organization.

## 1.2.2 Multiagent Organizations

Social behavior can be incorporated into multiagent systems by way of viewing the group of agents as organizations. Multiagent organizations include a set of *agents* that play *roles* within a structure that defines the *relationships* between the various roles [2]. In the following sections, I draw from the work of [8] for defining multiagent organizations. It defines the Organizational Model for Adaptive Computational Systems (OMACS) to define multiagent organizations.

### 1.2.2.1 OMACS

Using OMACS, an organization can be defined by the tuple

$$\{G, R, A, C, \Phi, P, \Sigma, \text{oaf}, \text{achieves}, \text{capable}, \text{requires}, \text{possesses}, \text{potential}\}$$

where  $G$  is the set of goals the organization pursues,  $R$  is the set of roles the agents in the organization play to accomplish the goals,  $A$  is the set of agents that are part of the organization,  $C$  is the set of capabilities the agents possess and determines whether an agent has the capability to play a role and if so, how well it can play it.  $\Phi$  is the assignment set describing what role a particular agent plays and what goal a role pursues.  $P$  defines the policies of the organization with respect to assignments, behavior of agents and reorganization.  $\Sigma$  is the domain model that allows the definition of object types in the organization environment and the relationships between these types. *oaf* is a function defining the quality of the assignments in  $\Phi$ . *achieves* is a function that defines how well a role achieves a goal. *capable* is a function defining how well an agent plays a role. *requires* is a function that specifies the set of capabilities required by an agent to play a role. *possesses* is a function that defines the quality of an agent's capability and *potential* is a function that defines how well an agent can play a role to achieve a goal.

In Section 1.2.2.2, I summarize GModS, which defines the goal model of the organization. Roles in multiagent organizations are analogous to the different roles humans play in a typical organizational structure. Roles are geared towards achieving the goals of the organization. In OMACS, a role is described by its name and a metric *rcf* that explains how well an agent can play

the role. Agents in an organization are the main components of the organization. They exhibit autonomous behavior, perceive the environment they inhabit with a set of sensors, react to the stimuli from the environment and take initiative to achieve goals. How well an agent does these actions are described by its capabilities. Capabilities in an organization are the set of all capabilities the agents in the organization possess and those that are required by the different roles within the organization.

Reorganization is a process of changing the assignment of agents to roles to goals as specified in  $\Phi$ . It involves the change in the state of the organization in response to reorganization triggers. Reorganization triggers are controlled or uncontrolled events that occur in the lifetime of the organization that may force the organization to reorganize. In the absence of any information or policies regarding reorganization, time complexity of total reorganization is of the order of  $2^{AxRxG}$  [8].

The organizational master is a specialized agent in the organization that handles all the organization-related tasks. Hence, it controls the behavior of the organization. Using the policies of the organization the organizational master changes  $\Phi$  in response to events in the environment and the organization. The organization model is explained in detail in Section 3.2.

#### 1.2.2.2 GMoDS

Goals are the set of objectives the organization as a whole tries to achieve. I use the Goal Model for Dynamic Systems (GMoDS) [8] to describe the goals of the organization. In GMoDS, the goals of the organization are represented using the goal specification  $G_{Spec}$  which includes the static representation of the organization goals and  $G_{Instance}$  which includes the representation of the goals of the organization that are dynamically instantiated and pursued. The relationships between the different goals of the organization are specified by the goal specification tree. The goal specification tree specifies relationships like a goal being sub goal of a larger goal, a goal being preceded by another goal, or a goal being triggered by an event. If achieving a goal involves accomplishment of sub goals, then using a set of predicates like conjunction, disjunction and precede, we can capture the necessary goal requirements of the organization. Goal instances are unique specializations of a goal. For example, if *search* is the goal, then *search(area1)* and *search(area2)* are two instances of the *search* goal. Here, *area1* and *area2* are the parameters for the *search* goal.

The lifecycle of a goal includes the following stages.

- Triggered (when an event triggers the creation of the goal instance),
- Active (when the goal instance is ready to be pursued),
- Obviated (when it is no longer necessary to pursue this goal) or
- Failed (when the organization decides that the goal can never be achieved) or
- Achieved (when the goal is completed)

### 1.2.2.3 Optimality of Organizations

In this study, the performance of an organization is measured by how well it accomplishes its goals. Specifically, in the search and rescue application, this refers to the ratio of victims rescued by the organization versus the actual number of victims present in the beginning. Hence, an optimal organization is one that rescues all the victims present in the search area.

As mentioned in the problem statement in Section 1.1, the reorganization strategy has a direct effect on the performance of the organization. Hence, to compare two reorganization strategies, I compare the performances of organizations that use these reorganization strategies in their reorganization module.

### 1.2.3 Genetic Programming

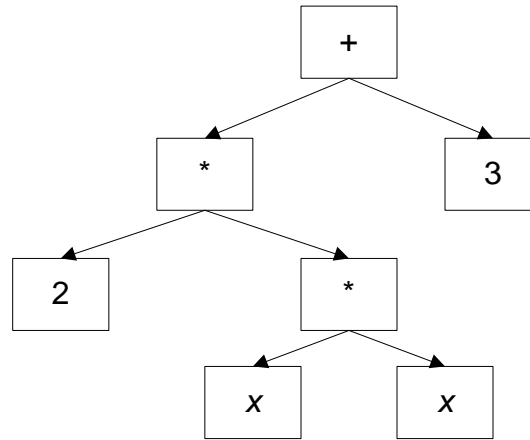
Genetic Programming is a way of program induction using evolutionary computation. Evolutionary computation is inspired from biological processes like natural selection and uses “directed selection” for search of problem solutions. It is increasingly being used for solving optimization problems for which no ideal solutions are known to exist.

#### 1.2.3.1 Terminology

A single computer program in genetic programming is known as an *individual*. Generally, individuals are represented as trees. In this representation, individuals are made up of *terminal nodes* and *functional nodes*. Terminal nodes are leaves of the tree while functional nodes are inside the tree having at least one child.

In Figure 1, 2,  $x$  and 3 are terminal nodes and the functional nodes are  $*$  and  $+$ . As depicted in the figure, the functions require two numerical arguments that are supplied by their child trees. Hence, this individual genetic program represents the mathematical equation  $2x^2 + 3$ . The

individual can also be represented by S-expressions using the prefix notation. The individual in Figure 1 represented as an S-expression:  $+ (* (2 * (x \ x) ) 3)$



**Figure 1. An individual program represented as a tree**

The *fitness* of an individual is a measure of how well it solves the problem. Depending on the problem we are trying to solve, the fitness criteria varies. For example, if the goal of the genetic program is to come up with the right equation, then the fitness of the individual in Figure 1 is the difference between the values the correct equation generates and the values the individual generates for various values of  $x$ . The lesser this difference, the higher the fitness of the individual. In cases where we do not know the right solution to the problem, we have to modify the problem so that a fitness value can be fixed to the individual.

A *population* is a set of individuals that have varied fitness. A population at any time in evolution is called a *generation*. The population of individuals can be created randomly or can be derived from a previous generation using *genetic operations*. Genetic operators are biologically inspired processes that produce a new population of individuals from a previous generation. The most common genetic operators include *crossover*, *reproduction* and *mutation*.

The crossover operator takes two individuals in a population, chooses a crossover node in both the individuals and swaps the node and the subtree below the node between the individuals. The reproduction operator copies a predetermined number of individuals from the previous population to the new population without changing them. The individuals are selected probabilistically based on their fitness. The mutation operator is similar to the crossover operator, but instead of swapping nodes, it replaces the selected node and its subtree with a randomly generated tree.

### **1.2.3.2 The five steps of genetic programming**

In [18], Koza specifies the five steps required to specify a genetic programming solution to a problem. These involve

1. Identify terminal set
2. Identify function set
3. Determine fitness criteria
4. Determine termination condition
5. Result designation

After determining the possible sets terminal and the non terminal (function) nodes that can appear in the tree, depending on the nature of the problem, a fitness criteria has to be determined. The termination condition specifies when the evolution process has to stop. The result designation specifies what we choose as the result of the process. It can be the best individual over all the generations or the best individual of the last generation of a set of highly fit individuals. This five step process helps us to design the genetic programming solution that suits the problem we are trying to answer.

Using the terminal and the function set, the initial population of individuals is created randomly. The fitness of the individuals within the population is measured to see if the individuals meet the termination condition. Until the termination condition is met, we continue to create newer generation of individuals using the genetic operators described in Section 1.2.3.1 and measure their fitness. In Section 3.3.3, I explain in detail how this procedure is applied to come up with evolutionary reorganization strategies.

## **1.3 Scope and Objectives**

The objectives of this study are

- To design and implement a multiagent organization to solve a search and rescue problem.
- To design a search and rescue simulator where in the multiagent organization can be simulated.
- To design reorganization strategies using the information about agents within the organization and the environment specific details of the search and rescue problem.
- To design reorganization strategies using an evolutionary approach

- To evaluate the performance of different reorganization strategies of the multiagent organization.

In this study, agents are completely independent in how they execute their tasks; but the organizational master decides the goals and roles for an agent. Hence the reorganization strategy for the organization specifically refers to the strategies of the organizational master and not of the entire organization itself. This reduces the learning problem in this study to a single agent learning problem in which, the organizational master's decisions indirectly affect the performance of the organization. Hence, in theory, any machine learning model should produce a successful reorganization strategy. In Section 2.2, I summarize the various machine learning techniques used in conjuncture with multiagent systems and explain why I use genetic programming in this study.

## **1.4 Document Overview**

This study is organized as follows. In the next chapter, I survey literature related to multiagent organizations and multiagent learning techniques. In Chapter 3, I discuss the design of the search and rescue simulator and the organizational model of agents. I restate the problem mentioned in Section 1.1 to suit the search and rescue simulator and describe the expected solution strategies. I also describe the five steps [18] required in the design of a genetic programming solution strategy. Chapter 4 discusses the implementation details for the simulator and the organization. In Chapter 5, I show the results of experiments conducted to test the hypothesis of this study. Based on these results, I conclude this study in Chapter 6 together with topics that could be researched in future.



## Chapter 2 Literature Review

Both multiagent organizations and distributed artificial intelligence (DAI) have vast bodies of literature associated with them. In this chapter, I survey relevant literature in both the fields and connect this study with them. This chapter is organized as follows. In the first Section, I review literature on design of multiagent organizations. In the next Section, I review relevant literature on machine learning adapted to multiagents.

### 2.1 Multiagent Organizations

Sycara in [30] documents the growth of multiagent systems from its beginnings to its current state. She notes the wide range of fields from which the area of multiagent systems has drawn inputs from and points out to the impediments in widespread applicability of multiagent systems like the lack of standard design methodologies and industry strength MAS toolkits. Recent advances in the field have tackled these issues well and we see that multiagent system design has generally moved from agent based approaches [9, 25, 40] to organization based approaches [7, 16, 41]. A survey of different paradigms used for multiagent organizations is presented in [18].

In [11], Dignum develops the OperA framework which is a three tier model that includes the organization model, the social model and the interaction model capturing both dynamic and predetermined interactions. With the help of formal logic, OperA guides the designer in conceptualizing the agent organization. In [22], Lematre and Excelente use roles, missions and organizational links as the basic entities of the organization. Social exchanges between the entities are decided based on the type of organizational link; communication, authority or acquaintance between the roles. In [41], Zambonelli, Jennings and Wooldridge introduce organizational rules as abstractions for the analysis and design of multiagent systems and use temporal logic to devise organizational rules and organizational structure. This work takes the requirements statements of the problem and splits it into the traditional agent-role model, an organizational rule model and an interaction model. Using first order temporal logic, these models are analyzed.

Horling and Lesser propose a new domain independent language called the organizational design modeling language (ODML) in [16]. While this model does not directly allow the designer to capture all the organization concepts, it provides low level primitives to describe each entity within the organization to capture the essential concepts. A specific element describing each entity is a *constant* which describes the quantitative information about the entity. Using this

information from all the active entities, the model predicts the different metrics of the organization without having to know any information of the environment.

In [8], DeLoach and Oyen develop the OMACS model. OMACS builds on the previous agent oriented approach of [9] by adding the formal analysis of the organizational behavior expected from the multiagent system. This is done by adding organizational capabilities, assignment set, organizational policies and a set of metrics that define the state of the organization at any given time. To represent a hierarchy of agents, OMACS introduces organizational agents. These are organizations that act as individual agents in a higher level of organization. The OMACS model extends the temporal logic used to define the organization to describe the process of transition from one state to another. The transition is triggered by reorganization triggers that are special events that change the goals of the organization or trigger a policy that requires a transition change.

The models summarized in this section have good ability to describe organizational behavior both qualitatively and formally. Other than [8] and [16] none of them specifically target the performance measures of the organization. [16] gives a unified metric that is derived from the different entities of the organization to predict the performance of the organization. This might not be as useful to the programmer because, in realistic environments, to decide what part of the organization needs to be intact and what needs to be changed, one has to know the individual abilities of agents and roles and a thorough knowledge of the local environment which might not be accurately visible to the organization. [8] gives rich information about the different metrics associated with the state of an organization enabling the designer to conceptualize reorganization. But, the OMACS model does not offer a concrete reorganization strategy. An extension of the OMACS model using a centralized architecture gives an organizational master all the necessary information about the OMACS model underneath it to carry out reorganization. The task of reorganization is left to the programmer, implemented depending on the environment.

In this study, I use the OMACS model to implement the multiagent organization to solve the search and rescue problem. The details of this model are provided in Section 3.2.

## **2.2 Learning in Multiagent Systems**

Very often, the problems one is trying to solve using machine learning systems are complex and ill understood. This has lead techniques to adapt to the task they face. Learning Classifier system is one of the important techniques to produce adaptive systems by combining reinforcement learning, evolutionary computation and other heuristic approaches [3].

Learning Classifier Systems (LCS) introduced by Holland [16] are rule-based systems, where the rules are usually in the form of IF *state* THEN *action*. Evolutionary computing techniques and heuristics are used to search the space of possible rules and reinforcement learning techniques are used to assign utility to existing rules, guiding the search for better rules. Initially the evolutionary computation was carried out by a genetic algorithm and the fitness of the rule was the payoff received from the reinforcement algorithm. Wilson [38] introduced the XCS, a simpler version of the LCS to increase understandability and performance. He also changed the way the Genetic algorithm allocated the fitness to a rule set from previous method. Fitness was assigned based on the accuracy of predictions in the payoff. This allowed an accurate mapping of the problem space rather than concentrating on higher payoff niches in the environment [3].

The LCS and XCS have been applied to many problems in machine learning including domains with multiagents. [19] provides a large bibliography of various resources on learning classifier systems and their application in various AI related problems including multiagent systems.

In [32], the authors explain a form of organizational learning using Organizational learning Oriented Classifier System (OCS). This system adapts classifier systems in multiagent scenario wherein the learning tasks are similar to the organizational learning requirements of this thesis. OCS is based on how human organizations behave and react to mismatches in the environment through *distinctions* and *practices* [14]. Distinctions in OCS can be related to reorganization of agents in this study, practices translate to the process of learning when to reorganize and what agents have to be used for reorganization. While the goals of [32] and my thesis are similar, the main differences are that this study uses local objective functions for learning reorganization while my problem statement is framed to maximize a global objective function. In [32], there is no specific distinction between what constitutes an organization and its components and thereby, organizational learning can mean different things like structural change in the organization or reassignment of agents.

As seen in [32], organizational learning in multiagents draws considerably from organization theory. In [5] the authors concentrate on modeling the performance of an organization exhibiting distributive decision making when information is distorted using the radar sensor network domain. This study explains the interdependence between information distortion (due to physical failure and design), organizational design and the task environment.

A large amount of literature exists in other areas in multiagent learning including evolutionary computing in multiagents. The predator prey problem [29, 37], robosoccer[29, 35], search and rescue [1, 4, 10] and resource allocation problems [29] are a few but popular domains in which

different aspects of multiagent learning problems are explored. As noted in [29], in many cases agents are assumed to be homogeneous, the system does not scale up to many agents or the actions learnt by the agents are at a low level in the organization. In other cases, the behavior of the multiagent system is static. In many cases, as commented in [27], either the agents are too specialized or too homogeneous.

As discussed in this section, there are a variety of learning techniques available to design a machine learning based reorganization strategy for the problem discussed in Section 1.1. Based on the setup of the search and rescue simulator and the organizational model of multiagents, it is seen that the reorganization strategy has to be learnt in a non-supervisory environment. The credit assignment problem is difficult to solve since the search space of the reorganization strategies is deceptive. Hence, one can not be sure about the importance of the different attributes of the environment and the agents. Further, the problem definition requires the agent system to show social behavior. Considering the advantages genetic programming has over other machine learning techniques [20], I use it to come up with the reorganization strategies for the search and rescue organization.

## **2.3 Summary**

In this chapter, I reviewed the relevant literature that guided the design choices for this study. We find that while vast literature exists in designing of multiagent organizations and in learning methods in multiagent systems, more work needs to be done in using the both in synergy to create adaptive agent organizations that display social behavior. In the next chapters, I use the techniques chosen from this chapter to design and implement the multiagent organization for the search and rescue application.

## Chapter 3 Methodology and Design

In this chapter, I explain the methodology used to design the search and rescue simulator and its solution strategies. This chapter is arranged as follows. First, I explain the design of the search and rescue simulator. Then, I formulate the organization problem mentioned in Section 1.1 for this application. This is followed by a description of the proposed solution strategies and the chapter summary.

### 3.1 Search and Rescue Simulator

In this study, the search and rescue problem is represented in a two dimensional grid world as shown in Figure 2. The scenario is a grid world ( $n * n$  matrix) where the agents have to rescue the victims that are spread across the grid at unknown locations.

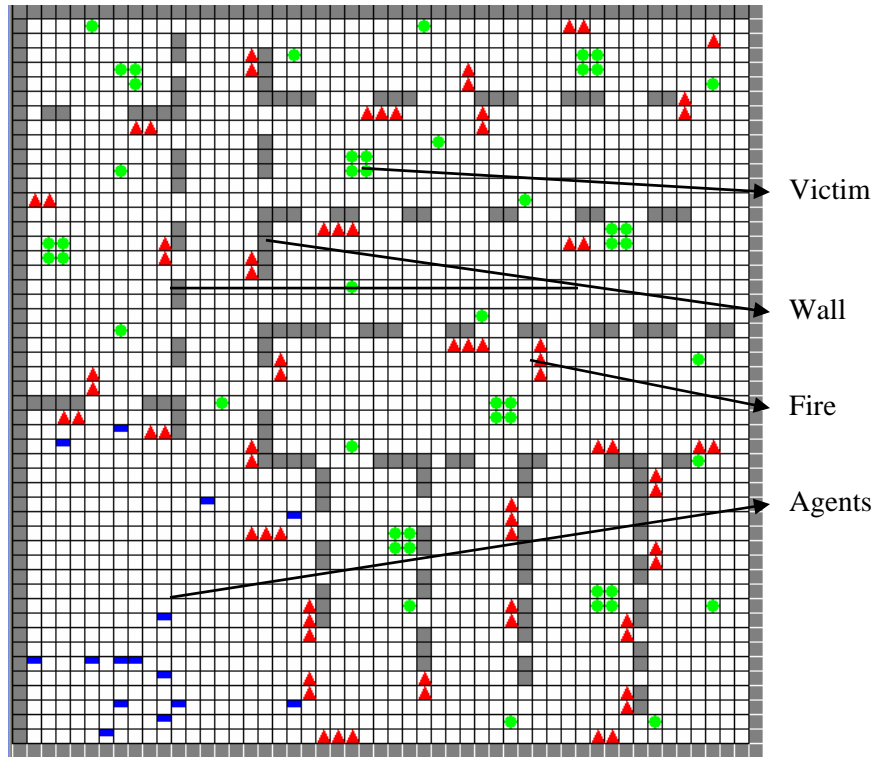
#### 3.1.1 Objects

The objects present in the simulator are shown in Figure 2. The grid world has walls and fires as obstructions in an agents' path. Walls are depicted by squares, the fire cells are triangles and victims are represented by circles. The agents are represented by the half filled squares. Only the agents have mobility. The smoke in the neighborhoods of the fire cells weakens the agent's sensors in those cells. Agents can not go through wall cells.

#### 3.1.2 Agents

As described by Russell and Norvig in [26], agents within the search and rescue simulator have a set of sensors through which they perceive the environment and act through a set of effectors. An agent decides on what sequence of actions it takes based on the limited reasoning capability it has. The sensors and responders of an agent make up its physical capabilities. Agents have different capabilities including the kind of roles they can play: Searcher and Rescuer roles. These roles are described in Section 3.2.3. The capabilities of an agent are read from a capability file which is loaded at the beginning of the simulation. The capability file includes numerical values ranging from 0 to 5, describing the quality of the physical capabilities of the agent and the list of roles it can play. Higher numerical values mean better particular physical capability. A zero value means that the agent does not possess the particular physical capability. Capabilities of an agent refer to the agent's maximum speed  $H$  (the number of cells traversed in each step of the simulation), the distance it can ping its sonar  $S$ , the weight  $W$  that the agent can carry, and the accuracy of the object detection sensor  $V$  within the agent. The sonar works only in the direction the agent is facing and helps the agent find the distance of the object present in its way. The

object detection sensor tells the kind of object the agent is facing with the accuracy of the detection depending on value of  $V$ . An agent dies if it tries to enter fire cells. The agents can explore, lift and carry victims. The physical capabilities of agents do not improve or degrade over time. No new objects are introduced during a run of the search and rescue simulator. The south-east corner of the grid world is designated as the “safe area”. If victims are brought to the safe area, they are rescued.



**Figure 2. Search And Rescue Simulator**

### 3.1.3 Information sharing and triggers

The state of the grid world is continuously updated as each agent explores the grid world. All agents can share this information through a shared black board. This helps the organization to decide whether an area is fully explored, or whether it is difficult to explore a region of the search area or if a victim is newly found or if it has been already identified.

The simulator allows triggers within the system to inform the agents about special changes that have occurred within the simulator. The triggers can be used to update information on the black board or to initiate response actions by the agents. Anybody including the system can raise a trigger.

## 3.2 Design of the Search and Rescue Organization

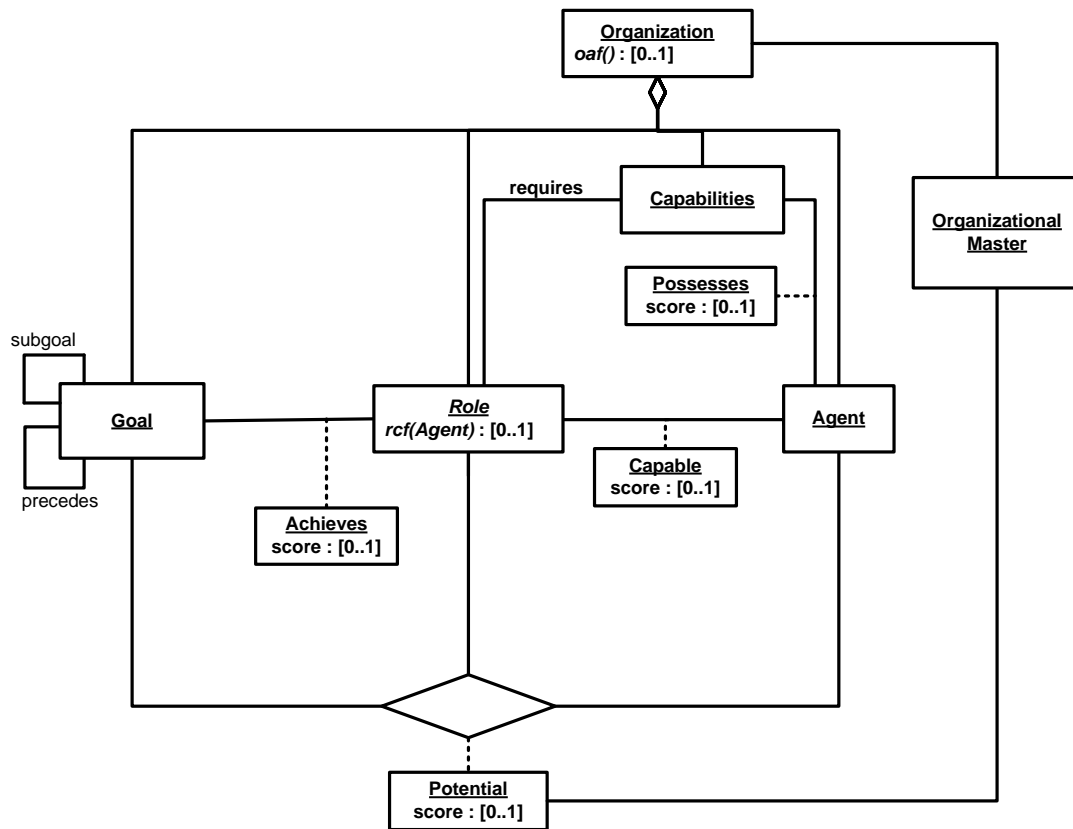
I use a simplified version of the organizational model developed in [8] to design the multiagent organization for the simulator. This model allows us to handle dynamic reorganization with the help of the organizational master. For ease of implementation, I use a single organizational master that co-ordinates between the agents in the system by assigning them roles and goals dynamically. The organizational master centrally controls the reorganization strategy of the whole multiagent system. This simplifies the problem of the reorganization in the sense that now, changing the reorganization strategy in a single organizational master; we can see its effect in the performance of the whole multiagent system. Based on the performance of the organization, the reorganization strategy of the organizational master is scored. Hence, this method allows us to compare two reorganization strategies in the search and rescue problem. The UML diagram of this model is presented in Figure 3. The diagram shows three important scores that describe the state of the organization at any given time in the simulator. The `oaf()` describes the organizational capability score based on current assignment of agents to roles and roles to goals. The `r.rcf(a)` describes the ability of an agent `a` to play role `r`. The `potential(a, r, g)` describes the potential of agent `a` in playing role `r` to achieve goal `g`. In the following sections, I explain each of the entities in the diagram.

### 3.2.1 Goals

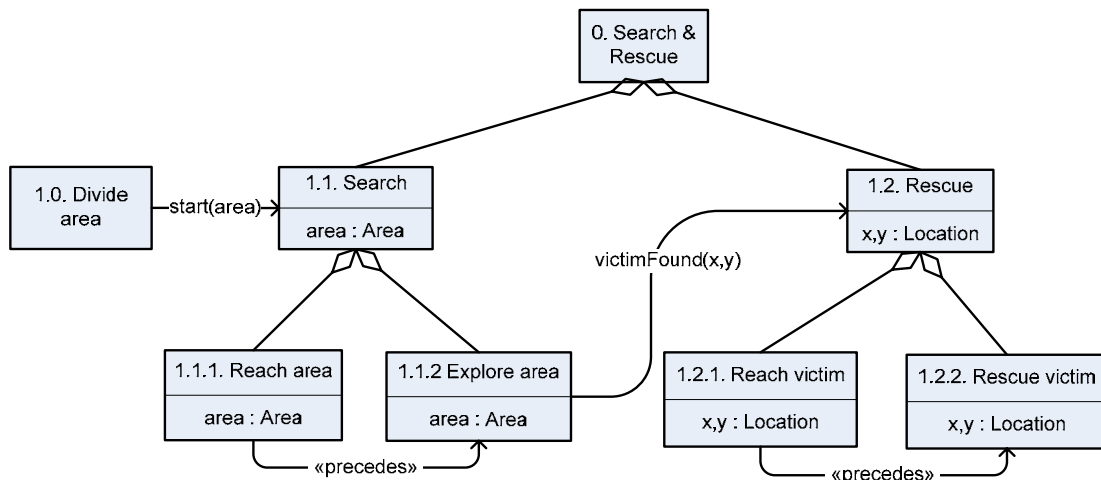
The goal diagram in Figure 4 is derived from the problem definition of the search and rescue application. All goal nodes except the root node and *Divide Area* are parameterized goals. That is, each of the goals requires area or target coordinate parameters to be initialized. The *search* goal is intended to search for victims in the target area. The *rescue* goal is assigned to rescue the victims detected during the search. Its parameters include the coordinates of the victim's location. The *Divide area* goal instantiates *search* goals by dividing the total area in the grid world into search regions. Using the framework depicted in [8], we can formulate the following relations between these goals.

In Figure 4, goals *Reach Area* and *Explore Area* are conjunctive subgoals of the *Search & Rescue* goal. Goals *Reach Victim* and *Rescue Victim* are conjunctive subgoals of the *Rescue* goal. Also, *Reach Area* precedes *Explore Area* and *Reach Victim* precedes *Rescue Victim*. All the leaf nodes in the search and rescue problem denote achievement goals. There are no maintenance goals in this problem. The *start(area)* event triggers the instantiation of a *search* goals with *area* of the search as its parameter. The *victim found* event occurs when an agent pursuing the *search* goal

finds a new victim. This event triggers the instantiation of a *rescue* goal with the coordinates of the victim's location as its parameters.



**Figure 3 Simplified Artificial Organization Model**



**Figure 4. Goal Diagram**



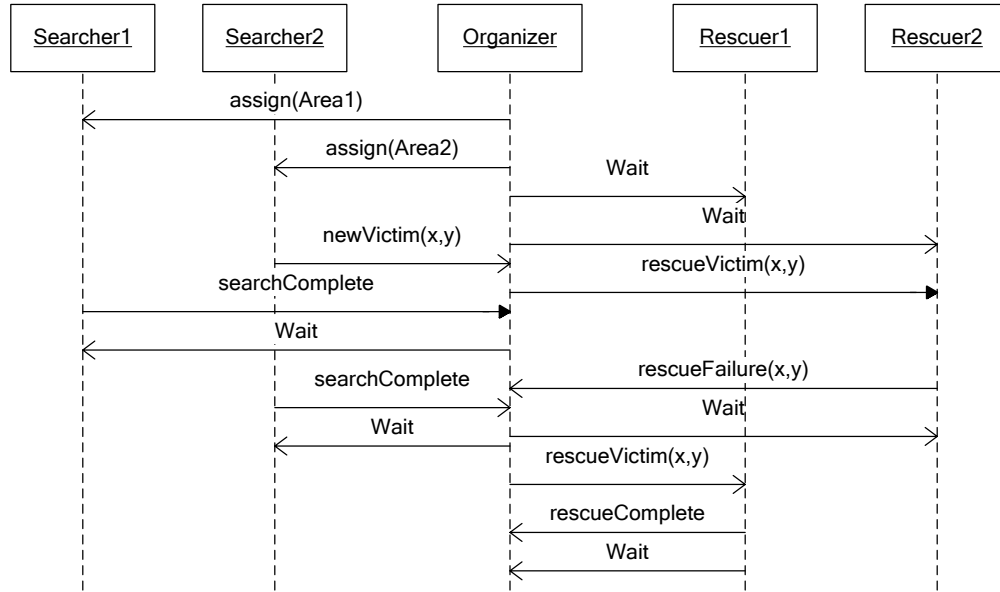
Active *search* goals that fail to complete are re-assigned when new agents capable of achieving *search* goal become available. The *rescue* goals are instantiated in response to the triggers raised for the *victims found* event. At any time, the number of active *search* goals is equal to the number of search areas that are still not fully explored. The number of active *rescue* goals at any time is equal to the number of victims found but not rescued. Active goal instances may not be pursued if the organizer decides not to, based upon the available resources and the reorganization strategy of the organization.

### 3.2.2 Sequence Diagram

The sequence diagram in Figure 5 shows an example of the functionality expected from the multiagent system. From the sequence diagram, we get three roles: organizer, searcher and rescuer. The organizer initially assigns particular areas of the search space to the available searchers. Available rescuers are noted by the organizer. When a new victim is found by a searcher, the organizer assigns a rescuer to rescue the new victim. If the rescuer fails in accomplishing its goal, the organizer assigns the goal to a different rescuer. The failed rescuer is added back to the free rescuer pool. Similarly, searchers completing the search goals are added to the free searcher pool.

### 3.2.3 Roles

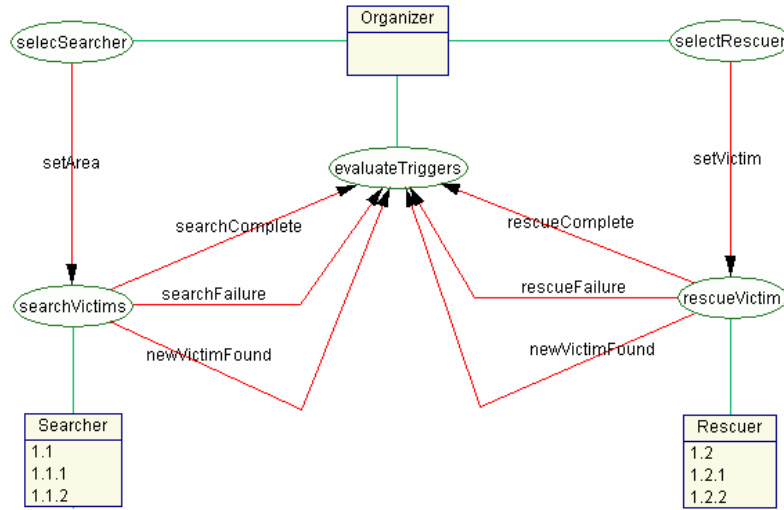
Based on the sequence diagram in Figure 5, the role diagram for the system was derived. The role diagram is shown in Figure 6. There are two roles in the system. Within the search and rescue simulator, each agent can either act as a searcher, rescuer or can perform both of these roles. The organizational master is the organizer. The searcher and rescuer roles each have one task to accomplish. The *searchVictims* task of the searcher is invoked by the organizer by using the *setArea*. Similarly, the *rescueVictim* task of the rescuer is invoked by the organizer using the *setVictim* protocol. Both these protocols provide the necessary coordinates as parameters to accomplish each task. Based on the events that occur during the execution of the *searchVictims* and *rescueVictim* tasks, triggers are raised both by the agents and the system. The system allows for five types of triggers. These triggers can be classified based on the events that trigger them. The *searchComplete*, *searchFailure*, *rescueComplete* and *rescueFailure* triggers are results of goal completion and goal failures. These are classified as reorganization triggers. The *newVictimFound* trigger is raised when a new victim found event occurs. This triggers the instantiation of a *rescue* goal as described in Section 0. Hence this trigger is classified as a goal trigger. Triggers are stored on the black board since it is visible to all the agents. The organizer responds to these triggers as and when they are raised.



**Figure 5. Sequence Diagram**

As shown in Figure 3, the `requires` relation captures the notion of a role *requiring* a specific capability to achieve its goals. In the search and rescue simulator, this relation is implemented using the capability file that specifies whether a particular type of agent has the physical capability to play a particular role. This is similar to reading the capabilities of agents described in Section 3.1.2. Based on this implementation of the `requires` relation, five agents have the capability to play the search role, five agents have the capability to play rescue role and five agents have the capability to play both the search and rescue roles. Agents differ from each other in terms of their physical capabilities. These physical capabilities determine how well an agent  $a$ , plays the role  $r$ . This notion is captured by the  $r.rcf(a)$  function. The function  $r.rcf(a)$  is a value between 0 to 1 representing how well agent  $a$  plays role  $r$ . The `achieves( $r,g$ )` is a relation between a role  $r$  and a goal  $g$  that denotes how well the role *achieves* the goal. A 0 value for `achieves` means that the role can not achieve the particular role. For example, a searcher will have a 0 `achieves` score for a *rescue* goal. The `achieves( $r,g$ )` value is predetermined in this system. Roles are hand coded (the algorithm an agent chooses to execute its role is hand-written and does not change over time). The organizer role is performed by the organizational master and is explained in detail in Section **Error! Reference source not found.**

### 3.2.3.1 Role Diagram



**Figure 6. Role Diagram**

### 3.2.3.2 Searcher

The searcher pursues the goal of exploring a given area by following its exploration algorithm. Its strategy is to move to its assigned exploration area and find victims. When it finds a new victim, it raises the `newVictimFound` trigger. When it has finished exploring the assigned area, it raises the `searchComplete` trigger. If the searcher fails to explore its assigned area fully, it raises the `searchFailure` trigger. All searcher roles follow the same exploration algorithm. Agents require sonar, speed and object detection capabilities to perform the search role. The exploration algorithm is described in Section 4.1.4.

### 3.2.3.3 Rescuer

The rescuer's goal is to reach the victim that has been identified and carry it to the safe area. Once a rescuer is assigned a victim, it tries to reach the victim following its *reach* algorithm. After bringing the victim to the safe area, it raises the `rescueComplete` trigger. If it fails, the `rescueFailure` trigger is raised. If the rescuer finds new victims in its way, it raises the `newVictimFound` triggers. Agents require sonar, speed, weight carrying and object detection capabilities of the agent performing the search role. The exploration algorithm is described in Section 4.1.5.

### 3.2.3.4 Organizer

The organizer creates the initial search and rescue organization and to re-organize the organization based on the triggers raised by the agents. As mentioned in Section 3.2, these tasks are controlled by the organizational master because they are all organization-related. Hence, this role is the organizational master itself. The organizational master does not appear in the grid world and does not have any physical capabilities. It instantiates *search* goals in response to the *start(Area)* event shown in section Figure 4 and assigns them to the available searchers initially. This active goal set for the *search* goals are instantiated only once in response to the event and hence, is treated as a simple assignment policy. Hence this event does not raise a reorganization trigger. The reorganization strategy of the organizational master is activated when the agents within the system raise reorganization and goal triggers. Therefore, the *oaf* is directly related to the actions of the organizational master. The organizational master also awards points to each agent based on its goal and role accomplishments within each simulation. This scoring is discussed in Section 3.2.5. The re-organization policy of the organizational master can be either hand-coded or expressed by an s-expression tree, derived from a learning algorithm.

### 3.2.4 Agents

As described in Section 3.1.2, the agents within the search and rescue system are heterogeneous in their physical capabilities. This relationship between an agent and its individual capabilities is captured by the *possesses* function. A value of 0 specifies that an agent does not *possess* the particular capability and a score of 5 represents that the agent has full capability in the particular physical attribute. We noted in Section 3.2.3 that role *r*'s *rcf* depends on the capabilities of the agent *a*. Hence, how well an agent performs a particular role is dependent on the agent's *possesses* function for the physical capabilities required by the particular role. This notion is captured by the *capable* relation. An agent's *capability score* for a particular role is defined as the role's *rcf* described in Section 3.2.3. Now, with all the relationships between goals, roles and agents specified, we can derive the *potential score* of an agent *a* playing role *r* achieving goal *g*. A *potential score* greater than zero indicates that the agent *a* has at least some ability to play role *r* in order to achieve goal *g*. In the search and rescue organization, when actual assignments are made for the agents we get a dynamic assignment set consisting of active agent-role-goal tuples  $\langle a, r, g \rangle$  represented by  $\Phi$ . Hence, the contents of  $\Phi$  can be described by

$$\Phi \subseteq \{ \langle a, r, g \rangle \mid \text{potential}(a, r, g) > 0 \}$$

### 3.2.5 Scoring

There are two types of scores in the simulator. The first type is the *expected scores* of the organization and the agents. These scores are calculated based on the physical capabilities and the role playing potential of the agents. These scores are expectations on how well an agent plays a role. The scores described in the model correspond to this type of score. The second type of score is the *actual* score achieved by the organization and the agents in each simulation. These scores are calculated based on how well the organization and the agents performed in each simulation. While the individual potential scores for the agents are determined *a priori*, the actual scores are calculated and updated after each simulation.

#### 3.2.5.1 Expected Scores

In this Section, I explain how the expected scores for this system are calculated. Based on the model, there are four expected scores to be calculated. They are, *rcf*, the *potential score*, the *achieves score* and the *oaf*.

The  $rcf(a)$  of a role  $r$  depends on the capabilities of agent  $a$ . In the search and rescue simulator, it is not easy to figure out what capabilities should be considered for calculating the *rcf*. Hence, in this model, I assume that all capabilities that are required to perform the role have equal importance in determining how well an agent performs a particular role. Thus, to determine the *rcf* for an agent  $a$  to perform the role  $r$ , I use the average of the sum of physical capabilities required by the agent to play the role. For other agents, *rcf* is 0. This is expressed by the equation

$$r.rcf(a) = \begin{cases} \frac{\sum_{c \in C_r} c/5}{m} & \text{if } a \text{ can play role } r \\ 0 & \text{otherwise} \end{cases}$$

where  $C_r$  is the subset of the capability set of agent  $a$  required to play the role  $r$ , the  $c$ 's are individual *posses* scores in  $C_r$  and  $m$  is the number of individual capabilities in  $C_r$ . The *rcf* varies from 0 to 1. As discussed in Section 3.2.4, this score is also equal to the agent's *capability score*.

In the search and rescue simulator, all agents that are capable of performing the search role use the same search algorithm and all agents that are capable of performing the rescue role use the same rescue algorithm. Hence,

$$achieves(g,r) = \begin{cases} 1 & \text{if role } r \text{ can accomplish goal } g \\ 0 & \text{otherwise} \end{cases}$$

From [8] we have,

$$potential(a,r,g) = achieves(r,g) * capable(a,r)$$

In this system, if an agent  $a$  has to accomplish a goal  $g$  it can only do so following only a particular role  $r$ . Further, a role  $r$  can only achieve one goal  $g$ . Hence, how well an agent accomplishes a goal depends only on how well it plays the role it is assigned.

$$potential(a,r,g) = r.rcf(a)$$

This score varies between 0 and 1. At any time during the simulation, the organizational assignment function  $oaf$  is the average of the *potential scores* of all the active agents pursuing respective goals.

$$oaf = \frac{\sum_{\langle a,r,g \rangle \in \Phi} potential(a,r,g)}{n}$$

where  $n$  is the number of active goal sets in the simulation. The *potential scores* and  $oaf$  allow us to choose agents during reorganization, based on their physical capabilities.

### 3.2.5.2 Actual Scores

The organizer awards points to each agent is when it completes its assigned goal. For instance, when an agent completes searching its assigned area or rescues a victim, points are awarded to it on the basis of the area explored and the time taken to explore the area. Hence these points represent the *actual score* of the agent in achieving its goals, playing either the search or the rescue role. The scores are averaged by the number of times the agent has been assigned the role. This allows us to compare agents based on their scores irrespective of the number of times each agent has been assigned a particular role. These scores help us to choose agents during reorganization, based on their prior performance on the scenarios.

The ‘raw score’ of each simulation is the ratio of the number of victims rescued to the number of victims initially present in the simulation. Hence, the raw score varies from 0 to 1. An ideal search and rescue organization would have a raw score of 1. This also acts as a score for the reorganization strategy used in the simulation. The raw score averaged over multiple simulations

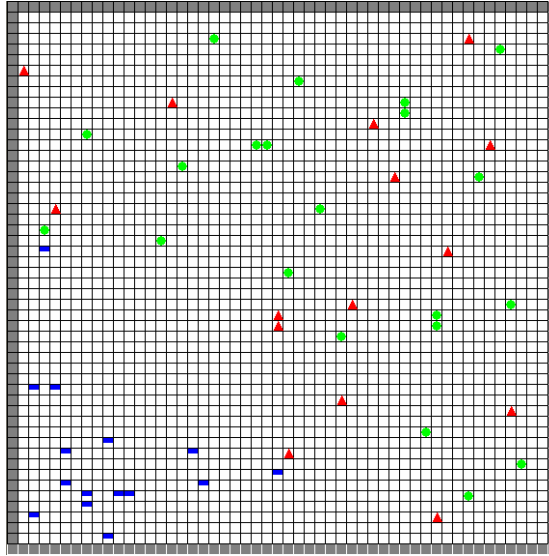
carried over varied scenarios gives a rough idea of how well the reorganization strategy works on an average among these scenarios.

### **3.3 The Search and Rescue Organization Problem**

Consider a sample scenario in the simulator that includes 15 agents rescuing 15 victims in a 50 \* 50 grid. The goal of the system is to find as many of the victims and bring them to the safe area. A group of agents with various capabilities are introduced in the grid world. The agents have to be assigned roles dynamically depending on ‘how well’ the agents can perform the role in this scenario. It is not possible to perfectly predict the performance of this multiagent system *a priori*. Agents may die in the grid world because of limited physical capabilities. The system may have to reorganize in order to satisfy the current goal since the loss of an agent may result in a sub-optimal organization. When an agent accomplishes the goal it has been assigned, or when an agent fails to accomplish its assigned goals, the agents may have to re-organize in order to reach an optimal organization as described in Section 1.2.2. Finding an alternative organization is not trivial since the number of valid organizations is large and the performance of these organizations may not be optimal in all cases. Also, there is no trivial way of measuring the optimality of an organization. There is no ‘one ideal solution’ for the re-organization of this agent organization, because based on the state of the simulation, the best reorganization strategy changes. Hence the search and rescue simulator allows us to test the hypothesis proposed in Section 1.1 and validate its claim.

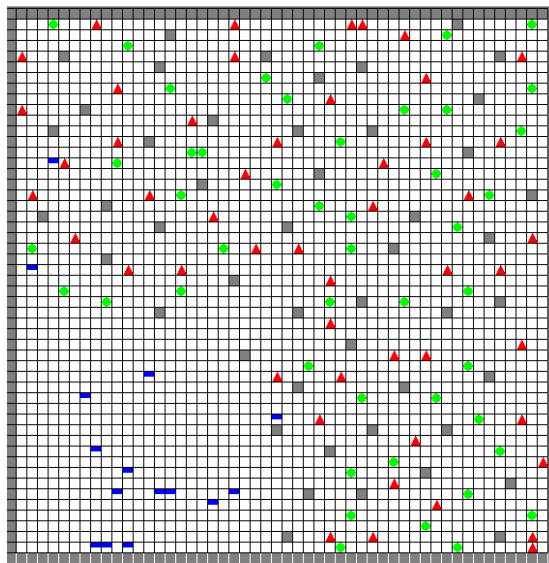
#### **3.3.1 Solution Strategies**

All the re-organization strategies implemented in this work are tested on eight scenarios. Intuitively, the scenarios are designed to exploit specific capabilities of the agents in different situations as well as depict situations similar to real world search and rescue operations. Each of the scenario and its description are given below. Figure 7 shows the simplest scenario in the simulator with few scattered victims. There are no walls very few fire cells. It tests for basic reorganization skills.



**Figure 7. Scenario 1**

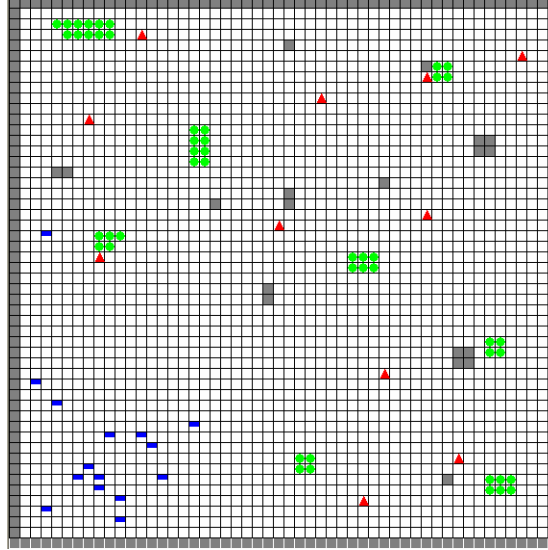
The objects in Figure 8 were randomly generated. There are no blocks of walls or fire and all objects are scattered throughout the grid.



**Figure 8. Scenario 2**

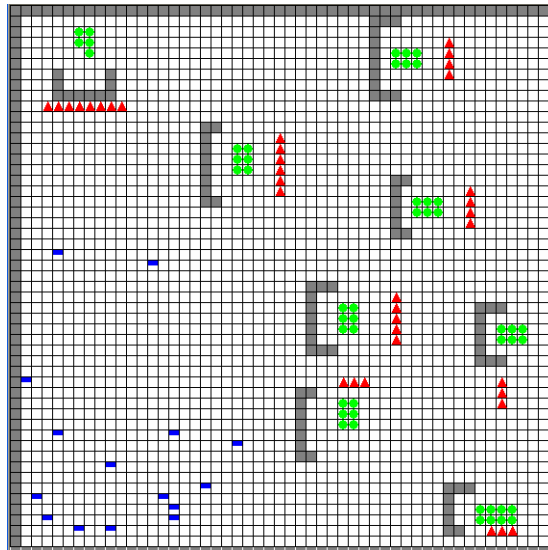
In Figure 9, groups of victims are placed at different locations. There are very few walls and fires. Figure 10 has blocks of walls and fire blocking groups of victims. But it is easy to get around the obstacles.





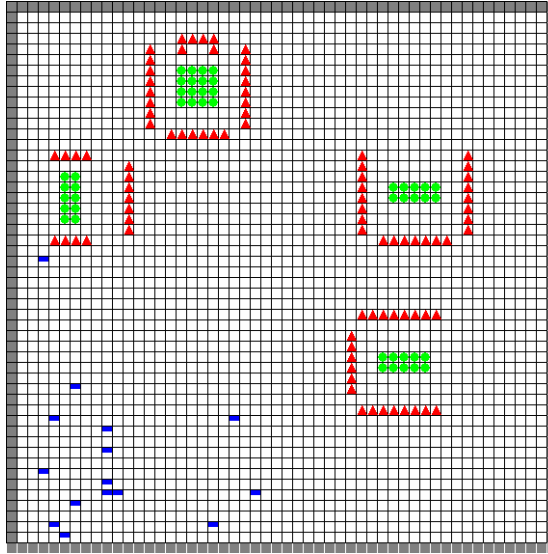
**Figure 9. Scenario 3**

The scenario allows the organizer to distinguish difficult areas, easy targets and far away victims and use agents with different capabilities for rescuing victims located in different situations.



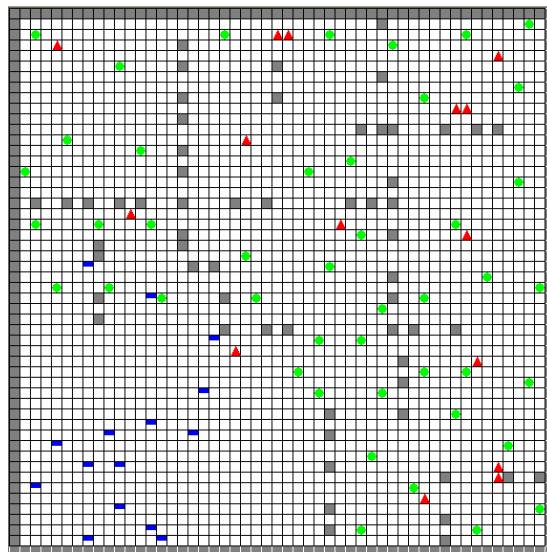
**Figure 10. Scenario 4**

In Figure 11, there are large groups of victims blocked by fire cells. Three groups have openings in three different directions and one group that is covered in all sides. This scenario allows the organizer to distinguish rescue failures based on difficulty of reaching targets and use different capabilities of the agents effectively.

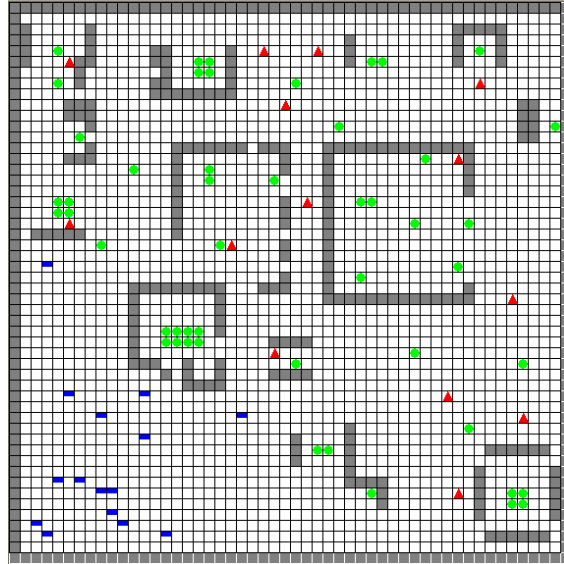


**Figure 11. Scenario 5**

Figure 12 – Figure 14 depict real life scenarios with increasing difficulty. The number of obstacles and their placement within the grid make it progressively difficult for agents to accomplish their goals and the chance of agent failures increases. This requires agents with different capabilities to be assigned to different areas for search and rescue grid based on the requirements of the particular area and the specific physical capability that will help each agent to accomplish its goal in the specific area.

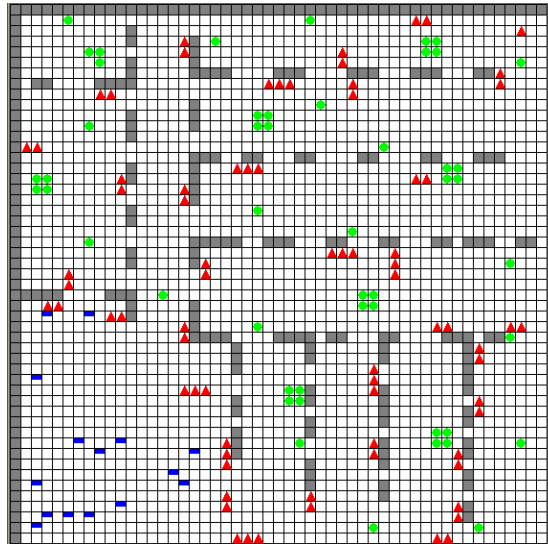


**Figure 12. Scenario 6**



**Figure 13. Scenario 7**

The raw score mentioned in Section 3.2.5.2 is used to differentiate the performance of reorganization strategies. A good solution to the organization problem stated in Section 3.3 should be able to perform reasonably well on average in all the scenarios shown in Figure 7 – Figure 14. Since these scenarios allow the organizer to extract useful information necessary to make rational decisions, any reorganization strategy that performs well in these scenarios should be able to perform well in any scenario in the search and rescue simulator.



**Figure 14. Scenario 8**

The performance of a reorganization strategy not only depends on the strategy followed by the organizer but also the search and rescue agents participating in the simulation. If the physical

capabilities of the agents participating in a particular simulation are not good, then no reorganization strategy will be able to get good results. A good reorganization strategy should make the best use of the available agents and their physical capabilities (thereby scoring consistently higher than other strategies for any given set of search and rescue agents). On an average over many simulations, it should also perform better than other strategy when the search and rescue agents are randomly picked from the agent pool.

### **3.3.2 Hand coded solution strategy**

As illustrated in Section 3.3, the programmer can not fully utilize scenario specific information since extracting useful information for better re-organization from a scenario is not trivial. Hence, a hand coded reorganization strategy is one in which the programmer selects a few parameters (*a priori*) upon which reorganization decisions are made. For example, the ‘least distant’ reorganization strategy selects the free agent nearest to the place of the trigger to replace a failed agent or chooses the nearest available rescuer to rescue a newly found victim. The ‘*oaf maximization*’ reorganization strategy selects the (agent, role) pair with the highest potential score to replace a failed agent or chooses the (agent, rescue) pair with the highest potential score to rescue a newly found victim. This maximizes the *oaf* of the organization at the time of the reorganization. The implementation of these strategies will be discussed in detail in Section 4.3.1.

### **3.3.3 Evolutionary reorganization strategy**

The inspiration for evolving reorganization strategies comes from the problem definition. The organization problem in the search and rescue application can be better answered if we are able to extract useful scenario specific details of each scenario and select searchers and rescuers that are most suited for the situation. For example, it might be useful not to assign a rescuer to a victim when it is known that previous rescue attempts of that victim have repeatedly failed. But, this might not work well in all scenarios especially if a majority of victims are located in areas where it is difficult to rescue. An evolutionary reorganization strategy might emerge over time that extracts the different scenario specific details and makes use of them in selecting an agent best suited for a particular job.

Evolutionary reorganization strategies are realized by allowing s-expression trees representing reorganization strategies to evolve using genetic programming. Each individual reorganization strategy is represented by three s-expression trees. Depending upon the type of trigger raised, one of these trees is evaluated. The evaluation of the tree returns an agent that can be assigned to the task for which the trigger was raised. The terminal and non terminal nodes of the s-expression

trees are chosen in such a way that, the application specific, non-intuitive strategies might be able to emerge through evolution.

To specify the reorganization strategy using genetic programming, I first specify the five steps in the problem specification of a genetic program [18].

### 3.3.3.1 Terminal Set

The terminal nodes represent leaf nodes in the s-expression tree. Each terminal node returns a particular agent as its return value. This result is passed on to its parent node. The description of the terminal set used here is given in Table 1.

To select the agents specified by the terminal set, during each run of the organizer, the available agents (agents that are in ‘wait’ mode) with the highest and the lowest rating for each physical attribute were determined. Hence, if the s-expression tree returned Best Agent, the organizer would choose the agent with the highest potential score as calculated in Section 4.1.6.1.

**Table 1. Terminal Set**

Terminal Node	Description
Best Potential (BP)	agent with best potential score
Best Agent (BA)	agent with the best actual score
Fastest Agent (FA)	agent with maximum H
Most Accurate Agent (MA)	agent with maximum V
Longest Vision (LA)	agent with maximum S
Nearest Agent (NA)	agent nearest to a specified location
Nil (Nil)	returns a null agent

### 3.3.3.2 Non Terminal Sets

The non terminal nodes represent operators and functions that evaluate a specific condition and choose one of their children as a result of that evaluation. All the non terminal nodes except the Not function are binary (they have two child nodes) in nature. After evaluation, a non terminal node passes the evaluations of a child node to its parent node. Hence, each non terminal node

also returns an agent as its result. The functions used in the non terminal nodes are described Table 2.

**Table 2. Function Set**

<b>Function(args)</b>	<b>Description</b>
IfDifficult(target) (IFDIFF)	If the target area or cell is difficult to reach and failures have been known to have occurred, choose the left child, else choose the right child.
IfObstacles(target) (IFOBS)	If the target area or cell is explored and it has been found that there are obstacles in and surrounding the target area, choose the left child, else choose the right child.
IfFar(target) (IFTF)	If the target area or cell is far from the “safe area, choose the left child, else choose the right child.
IfNear(target) (IFTN)	If the target area or cell is near to the “safe area, choose the left child, else choose the right child.
Not() (NOT)	Choose the complement of the child node.

The information used by each function varies in the context in which the s-expression tree is being evaluated. For example, when the new victim found trigger and the rescuer failure trigger begin the evaluation of the s-expression tree, all the operators use the location of the victim as their target cells. But, when a searcher fails, the operators use the failed searcher’s target area in their decisions. Similarly, the Not operator operates differently depending on whether its child is a leaf node or a non terminal node. For example, the expression (NOT BP) is evaluated to choose the agent with the lowest potential score. The expression (NOT (IFTF BP NA)) is evaluated to choose the nearest available agent instead of the agent with the lowest potential score.

### 3.3.3.3 Fitness function

I use the raw score of the organization described in Section 3.2.5.2 to determine the fitness of an individual. To see whether a reorganization strategy performs well even when the participating search and rescue agents have limited physical capabilities, the simulations are repeated over

varying set of fixed number of agents picked from the agent pool. The fitness of the reorganization strategy is the average performance of the organization over these repetitions.

#### 3.3.3.4 Control Parameters

The control parameters are summarized in Table 3. This setting is based on standard genetic program settings used in ECJ.

**Table 3. Control Parameters**

Population size	500
Maximum generations	50
Crossover probability	90%
Reproduction probability	9%
Mutation probability	1%
Maximum size of programs	17
Maximum size of initial programs	6
Method for initiating individuals	Ramped half and half
Selection method	Tournament, group size = 7

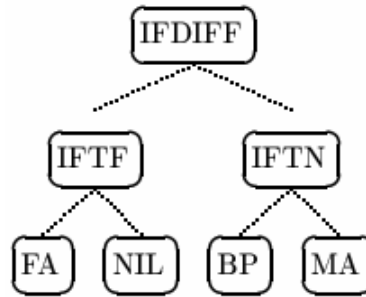
#### 3.3.3.5 Stopping criterion

When a set of agents with limited physical capabilities are chosen, it might not be possible for the organization to rescue all the victims in a given scenario. It is not possible to attain the ideal fitness of 1. Since we do not know best solution *a priori*, the stopping criterion is maximum generations. That is, we run the simulations for the maximum number of generations and choose the individual with the best average raw score.

#### 3.3.3.6 An example reorganization strategy

Figure 15 is an example of a tree representation of a reorganization strategy for a single trigger. The corresponding s-expression is (IFDIFF (IFTF FA NIL) (IFTN BP MA)). The meaning of this strategy is that, if the target area is difficult to reach and is far from the safe area, then the fastest available agent has to be chosen. If the area is difficult to reach but is not far from the safe area, then no assignment is made. If the target area is not difficult to reach and if the area is near

to the safe area, the agent with the highest potential score is chosen. If the target is not difficult to reach and is not near the safe area, then the most accurate agent is chosen.



**Figure 15. An evolutionary reorganization strategy**

### **3.4 Summary of Methodology and Design**

In this chapter, I described the design of the Search and Rescue simulator and the structure of the organizational model that will be used in the simulator. The organizational problem was defined and the possible solution strategies were described in Section 3.3. The evolutionary reorganization strategy was represented as a genetic program and its description was given in detail. In the next chapter I discuss the implementation of this design.



## Chapter 4 Implementation

In this chapter, I explain the implementation of the Search and Rescue Simulator and the reorganization strategies explained in Section 3.3. I begin by explaining the class structure of the simulator followed by description of how the simulator runs. Then, I discuss the implementation of the reorganization strategies.

### 4.1 Simulator

The simulator was developed based on the Wumpus world described in [26]. The important classes are shown in the class structure in Figure 16. The agent manger instantiates the search and rescue world. The search and rescue world includes the search and rescue agents and other objects within the simulator. It also includes the blackboard that is visible to the search and rescue agents and the organizational master.

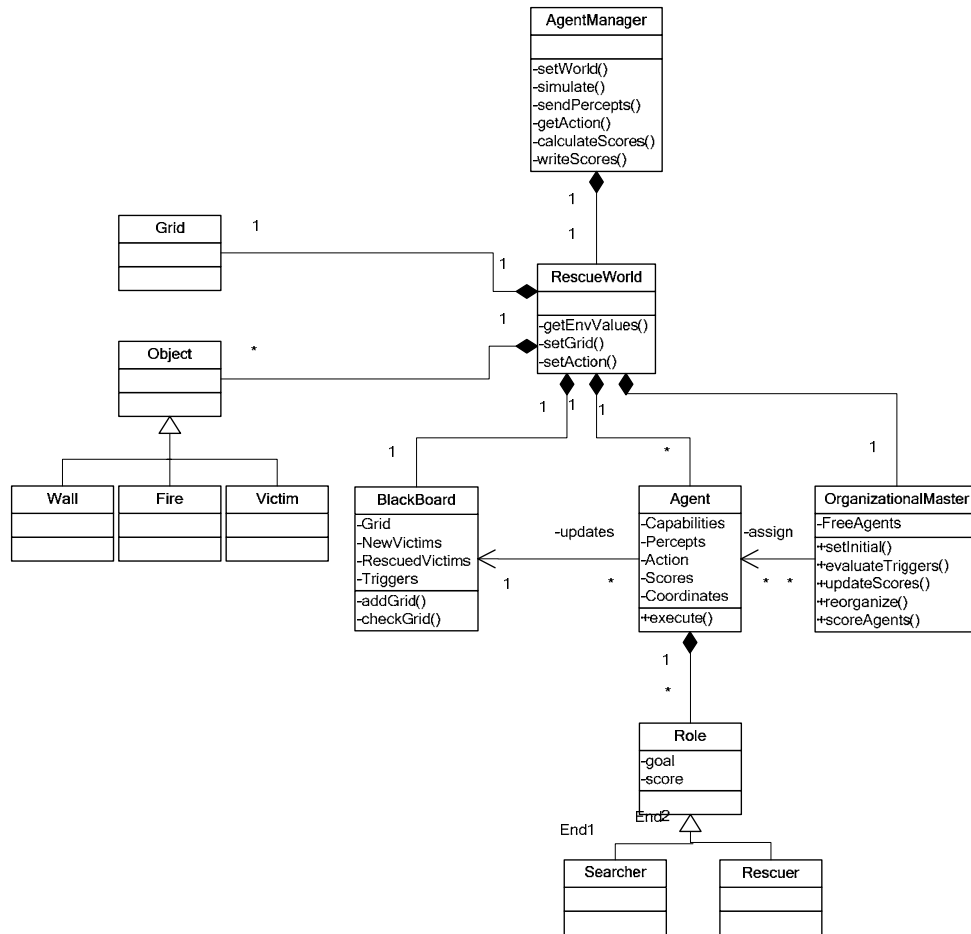
The organizational master sets up the initial configuration of goal assignments to the search and rescue agents. The classes shown in Figure 16 are explained below.

#### 4.1.1 Agent Manager

This class oversees all the activities of the simulator. Its members include the Rescue world that houses all the objects, the file readers and writers that fill the capabilities of the agents and scores and has access to all the information in the simulator. A single cycle of the simulator is completed by the `simulate()` function. In each cycle of the simulation, the agent manager sends the percepts to each of the search and rescue agents and simulates their action on the search and rescue world. It also invokes the organizational master to evaluate any possible triggers that might rise during the simulation. At the end of the simulation, it calculates the raw score of the organization based on the number of victims saved and writes the agent scores and organization scores in a file.

#### 4.1.2 Black Board

This class implements the communication system for the search and rescue agents. They update the blackboard with the information about the cells they visit. This way, the blackboard stores the information gathered by the agents in the simulation in a common grid. It also stores information about new victims found during the simulation.



**Figure 16. Static Structure**

### 4.1.3 Rescue Agent

This class implements the search and rescue agents. Depending on their physical capabilities, the agents can be divided into 15 types. Of the 15 agents, 5 agents can play only searcher roles, 5 agents can play only rescuer roles and the remaining 5 agents can play both searcher and rescuer roles. The physical and role playing abilities of the agents are loaded from a file that specifies the ratings for each agent's sensor and physical abilities. An agent that does not have an assigned goal is in 'wait' mode and wanders within the simulation boundary randomly until it is assigned with a search or rescue goal. An agent that enters a fire cell dies and the system raises a goal failure trigger. This trigger corresponds to the goal the agent was trying to achieve before entering the fire cell. If the agent is carrying a victim, both the agent and the victim are terminated in the simulation run.

#### **4.1.4 Searcher Role**

This class implements the search algorithm. The role can achieve two goals; reach target area and explore that area. Once the agent reaches its assigned area, it plans the exploration by setting up target cells and scans the region between these target cells. If a searcher finds a victim in its path, it updates the blackboard with the co-ordinates of the new victim. At the end of the exploration, it checks with the blackboard and confirms whether all the cells in its target grid have been explored. If not, it makes finite attempts to explore unidentified cells. If it fails, it raises the search failure trigger. If it completes the exploration successfully, the searcher raises a search complete trigger. In both situations, it adds itself to the free searchers array and puts itself into 'wait' mode.

#### **4.1.5 Rescuer Role**

This class implements the rescuer algorithm. A free rescuer is activated when a new victim is assigned to it to be rescued. The two goals this role can achieve are 'reach victim' and 'carry victim' to the safe area. The role makes finite attempts to reach the victim and if it fails, it raises a rescue failure trigger. If the rescuer succeeds in reaching the victim, it lifts the victim and attempts to bring it to the safe area. If it fails after repeated attempts, it keeps the victim at its current location and reports its failure in the blackboard. This raises a rescue failure trigger. In both situations, the rescuer adds itself to the free rescuers array and puts itself into 'wait' mode.

#### **4.1.6 Organizational Master**

The organizational master instantiates the initial configuration of the agents. Initially, it assigns search and rescue roles to the agents based on their role playing abilities. Half of the agents that have both search and rescue capabilities are assigned as searchers and the rest are assigned as rescuers. The simulation grid is divided into search areas among the initially available searchers. This triggers the instantiation of a finite number of *search* goals depending on the initially available searchers. All rescuers are put in 'wait' mode in the beginning.

##### **4.1.6.1 Scoring**

The expected scores are calculated when the agents are loaded during a simulation as described in Section 3.2.5. In each step of the simulation, the organizational master checks whether any triggers were raised in the system. For goal completion triggers, points are awarded to agents completing the goal. The ratio of the number of time steps taken by the agent to complete the goal and the actual area to be covered by the agent to complete the goal is awarded to it. The same score is awarded when an agent completes rescuing a victim and bringing it to the safe area. This

way, if two or more agents are involved in completing a particular instance of a goal, each of them is only awarded points for their part in completing the goal. Since the time counter starts when the agent is assigned the search and rescue goals, intermediate goals like reach and explore are not awarded points separately. There is no negative penalty for goal failures and hence, the scores remain unchanged when goal failure triggers occur.

## 4.2 Simulator Runs

A run of the simulator repeatedly runs the simulator for the selected number of cycles. A simulation cycle refers to one invocation of the `simulate()` function of the agent manager. The simulator can run both in graphics and text mode. The `SimGui` class implements the GUI for the simulator. The user can select the number of agents, the number of cycles for each run of the simulation, the number of runs on each scenario and the number of scenarios that have to be simulated. The Sim GUI also has a map editor with which the user can design different scenarios by placing walls, fire and victims at selected locations.

The front end of the simulator instantiates the agent manager and invokes the agent manager's `simulate()` function as many times as the number of cycles was specified by the user. When a run of the simulation ends, the scores of the agents and the roles are written on to the score files and this updated information is used in subsequent runs of the simulator.

## 4.3 Implementation of reorganization strategies

The reorganization strategy module of the organizer gets activated when new triggers are raised by within the search and rescue system. Hence reorganization strategies refer to the actions of the organizational master in response to the triggers raised in the simulator. The three specific triggers that invoke reorganization are new victim found trigger, search failure trigger and rescue failure trigger. To make the implementation simple, I use only the agents that are in 'wait' mode for reorganization. An agent can be in the wait mode either because it has completed its goal and is currently unassigned to any goal or it might have failed to accomplish its goal and is waiting to be reassigned. This is not a full reorganization because not all the agents are considered for reorganization. Hence one may argue that the reorganization might not lead to an enhancement of the raw score of the organization. To tackle this problem, I made the agents to 'give up' after finite attempts to complete their goals. Even though this does not solve the problem completely, inferior assignments are detected early on and agents in those assignments raise failure triggers and are considered for reorganization. In the following sections, I use the term *available* agent to an agent that is in the wait mode.

### 4.3.1 Hand coded strategies

#### 4.3.1.1 Best Potential strategy

This is a simple implementation of the *oaf* maximization strategy discussed in Section 3.3.2. In this strategy, the goal of the reorganization is to maximize the *oaf* of the organization at the time of the reorganization under the constraints that only the agents in wait mode are chosen. The intuition behind this strategy is that by choosing an agent that has the maximum *potential score* we maximize the chance of the goal being accomplished by the agent. For example, when a search failure trigger occurs, all the agents that can not achieve the search goal are eliminated from the free agents array. In the remaining agents, the agent with maximum potential  $(a, r, g)$  is chosen.

#### 4.3.1.2 Least Distance strategy

In this strategy, the goal of the reorganization is to choose the nearest available agent during reorganization. For example, when a new victim is found, the nearest available rescuer is assigned to rescue the victim. When a searcher fails, the nearest available searcher is assigned the search role.

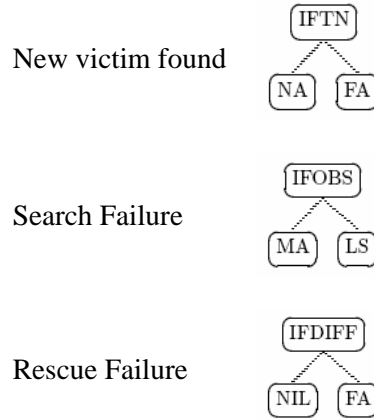
#### 4.3.1.3 Individual physical capability based strategies

Here, the goal of the reorganization is to choose the agent that has the best values for a particular physical capability as specified in Section 3.1.2. For example, the Fastest Agent reorganization strategy chooses the agent with the highest hop speed value within the available set of agents in response to the trigger. Similarly, Longest Sonar strategy chooses an available agent with the highest sonar value and the Most Accurate Agent strategy chooses an available agent that has the highest object detection sensor capability.

#### 4.3.1.4 Extended hand coded strategies

As mentioned in Section 3.3.3, even though the use of scenario specific information might be useful, one can not guarantee that such information shall lead to better performance in all situations. To test this hypothesis, I develop a simple hand coded strategy that uses predetermined s-expression trees to choose agents during reorganization. Specifically, in this strategy, when a new victim is found close to the safe area, I assign the nearest available rescuer. Else, I select an available agent with the highest hop speed. When a searcher fails, I replace it with an available searcher with highest sonar value if the search area is known not to have obstacles; else, I replace it with an available searcher with the highest object detection sensor capability. When a rescuer fails, if the victim it is trying to rescue is difficult to rescue, I will not replace the rescuer; else, I

replace it with an available rescuer with the highest hop speed. Based on the type of trigger raised, this strategy can be represented by three s-expression trees as shown in Figure 17.

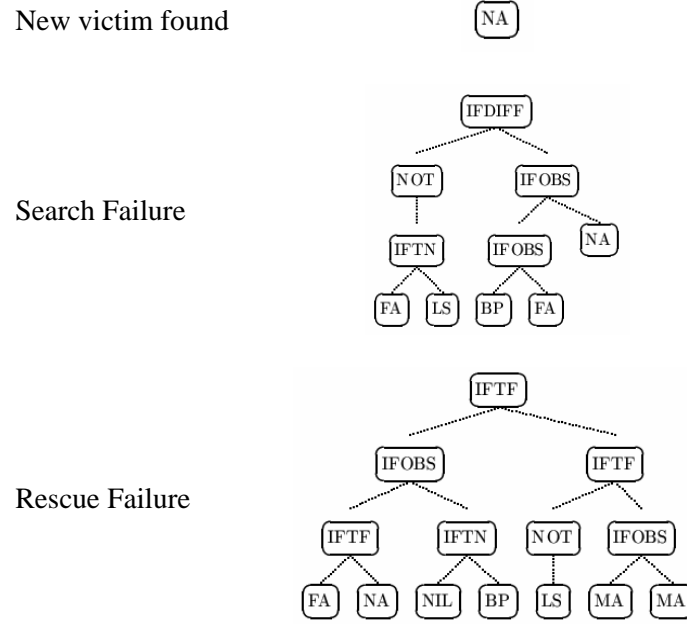


**Figure 17. Extended hand coded strategy**

### 4.3.2 Evolutionary reorganization strategies

I used Evolutionary Computation in Java (ECJ) [23] to implement the evolutionary reorganization strategies. ECJ is an evolutionary computational system that allows for easy representation of genetic programs in Java. The terminal and non terminal sets mentioned in Section 3.3.3 are Java programs using which ECJ creates genetic programs that are evolved using the control parameters given in Table 3. A reorganization strategy is represented by three s-expression trees. The trees correspond to the three types of triggers raised within the simulator. Hence, an individual reorganization strategy contains a new victim tree, a search failure tree and a rescue failure tree. Based on the type of trigger raised, one of these trees is selected and evaluated.

The trees return a number that denotes one of the available agents. Based on the kind of the trigger raised, a rescue goal is activated (in case of a new victim found trigger) and the selected agent is set to pursue this goal or the goal of a failed agent is reassigned (in case of a goal failure) to the selected agent. Figure 18 shows a reorganization strategy including the triggers that evaluate the corresponding trees. When a new victim is found, the reorganization strategy chooses an available agent that is nearest to the victim. When a rescue agent fails to accomplish its goal, a rescue failure trigger is raised and the organizational master evaluates the rescue failure tree. Based on the conditions of the scenario, one of the available agents is selected as the replacement.



**Figure 18. An individual reorganization strategy**

Initially, a population of reorganization strategies is created randomly. Each individual is used as the reorganization strategy of the organizational master and is tested a given scenario. The fitness of each reorganization strategy is determined by the raw score averaged the number of simulations. Selection, sexual reproduction and mutation operators are applied on the s-expression population and newer individuals are generated. The process is repeated over many generations and the individual s-expression tree that has the highest raw score is selected as the best possible reorganization strategy found. The experiment is continued over all the scenarios to get a reorganization strategy that performs well on an average over all scenarios.

## 4.4 Summary of Implementation

In this chapter, I explained the implementation details of the Search and Rescue Simulator, the Organizational model and the hand coded solution strategies. The simulations carried out and the results are discussed in the next chapter.

# Chapter 5 Results

## 5.1 Introduction

In this chapter, I discuss the results of the simulation runs conducted on the search and rescue simulator. Both hand-coded strategies and evolutionary reorganization strategies were used for reorganization. In the first part of this chapter, I explain the common setup parameters for the experiments and in the following Sections; I describe the results of the hand-coded strategies and evolutionary reorganization strategies.

## 5.2 Setup

The search and rescue grid is fixed at 50 x 50. As mentioned in Section 4.1.3, based on the physical capabilities of the agents, the agent pool of the simulator consists of 15 types of agents and based on the role playing capabilities, the agent pool can be divided into searchers, rescuers and agents having the capability to play both the roles. For each scenario, 15 agents were randomly picked from this agent pool such that 5 agents were capable of playing only the searcher role, 5 agents were capable of playing only the rescue role and 5 agents had the capability to perform both the search and rescue roles. Initially, the organizational master alternatively assigns search and rescue roles to the 5 agents that have the capability to perform both the search and rescue roles. Hence, based on this experiment setup, for each scenario, initially 7 to 8 searchers were available for exploration and were assigned different regions in the scenario. The remaining agents were put in the “wait” mode and are assigned search and rescue goals and roles as the simulation progresses.

Since agents are picked randomly, for each hand-coded reorganization strategy, I repeated the simulation for each scenario 100 times and recorded their average raw score. This score is assigned as the score of the reorganization strategy. On an average, there are around 45 victims per scenario, spread out in an area of 2500 cells. On an average, the victims are 50 cells away from the safe area. If we assume that each agent moves one cell per move and takes three times this distance to reach the agent, pick it and bring it to the safe area, then for rescuing all victims it would take 6750 steps collectively. On an average if there are 7 rescuers available, then ideally, 1000 steps should be sufficient for the agents to rescue all victims. Hence, the experiments were repeated for 1000 cycles of the simulator. I also test the hand-coded strategies by doubling this to 2000 cycles of the simulator. The results are discussed in Section 5.3.



To get the best evolutionary reorganization strategy for each scenario, I simulated each reorganization strategy in the GP population on the particular scenario as described in Section 3.3.3 for 1000 cycles. For each individual in the population, the simulation was repeated 100 times and the average of the raw scores was awarded as the fitness of the individual. Based on the control parameters for the GP described in Section 3.3.3, I used ECJ to evolve reorganization strategies and selected the strategy that achieved the best fitness for each of the scenarios. Similarly, I also obtained the best evolutionary reorganization strategy for all the scenarios.

### 5.2.1 Time constraints of experiments

In setting up the experiments, a significant factor was the time constraints of the experiments. All the experiments were conducted on a single processor Pentium 4 3.2 GHz, 1 GB RAM machine.

The hand-coded strategies could be optimized since the decision variables were known *a priori*. For evolutionary strategies however, all the possible variables had to be calculated in every step. For example, each time a goal failure trigger or a new victim found trigger arose, the organizational master has to have the list of agents that have the best physical capability for all the capabilities, the scores of the available agents, the status of exploration of the target region etc. Hence, a single reorganization decision for hand-coded strategies on an average took 0.003 milliseconds while a single reorganization decision for an evolutionary strategy took about 0.017 milliseconds.

To evaluate an individual reorganization strategy for a single scenario with 100 turns and 1000 simulation cycles, it would take approximately 1.7 seconds. Using the control parameters listed in Table 3, to come up with the best reorganization strategy for each scenario, it could take up to almost 12 hours.

## 5.3 Hand-coded strategies

The performance of hand-coded reorganization strategies for 1000 cycle simulations is summarized in the tables below.

Table 4 summarizes the scores of the different hand-coded strategies on scenarios in Figure 7- Figure 11. The first three strategies are classified under physical capability based reorganization strategies. The Best Potential strategy is a simple implementation of the oaf maximization strategy and the Nearest Agent strategy follows the least distant strategy. These strategies are described in 4.3.1.

**Table 4. Hand-coded Strategies - Scores**

Strategy	Scenario1	Scenario2	Scenario3	Scenario4	Scenario5
Most Accurate Agent	0.62	0.37	0.54	0.18	0.11
Longest Sonar	0.72	0.41	0.44	0.24	0.02
Fastest Agent	0.77	0.50	0.67	0.04	0.17
Best Potential	0.72	0.35	0.50	0.16	0.02
Nearest Agent	0.86	0.33	0.50	0.12	0.02
Extended hand coded	0.75	0.52	0.58	0.10	0.19

In the first scenario, the nearest agent strategy yields the best result followed by the fastest agent strategy. This is expected since the scenario has very few obstacles other than fire cells. Hence, nearness to the target and speed of the agent are the primary factors deciding the performance of the agents. The extended hand coded strategy's performance is similar to the Fastest Agent strategy since in this scenario, victims are rarely near the target and are not difficult to rescue. Hence, the strategy reduces approximately to the Fastest Agent strategy.

In scenario two, all objects are randomly spaced and the obstacles appear in single cells. Hence, all physical capabilities play a significant part in the overall performance of the agents. But, since the objects are randomly dispersed, a collective measure like the potential of the agent overlooks variations within the different physical capabilities may have a lower score. Also, due to the dense distribution of objects within the scenario, nearness of the assigned agent to the target does not have a significant impact on the overall score of the system. In the second scenario, we also see that the spectrum of scores of various scenarios narrows down because the objects are placed randomly. The extended hand coded strategy performs better than other hand coded strategy since it uses agents with different physical capabilities for different situations.

The third scenario shows that when there are very few obstacles, all other physical capabilities other than the speed of the agent are of the same importance. Also, in this scenario, victims are found in groups. Hence, for a single victim, if we choose the nearest available agent, there is not going to be a significant performance increase because on an average, agents will be spread out and very few agents will be in the vicinity of the region to make use of the nearness factor.

The fourth and fifth scenarios are difficult and hence in general, all strategies have lesser scores. These scenarios are difficult because the obstacles in these scenarios have continuous shapes and without full knowledge of the region, it is difficult for an agent to successfully plan either a search or rescue. But, in both cases, the most accurate agent strategy performs reasonably well because this strategy maximizes the possibility of keeping the assigned agents alive. In the fourth scenario, the longest sonar strategy works best. This may be due to the fact that in this scenario, the obstacles are mostly walls and hence, exploring their structure is not harmful, even if object detection is inaccurate. This helps an agent to plan the exploration of the target region or rescue a victim effectively. In the fifth scenario, the extended hand coded strategy scored maximum points followed by the fastest agent strategy.

### 5.3.1 Real-life based scenarios

**Table 5. Hand-coded Strategies - Scores**

	Scenario6	Scenario7	Scenario8
Most Accurate Agent	0.69	0.18	0.10
Longest Sonar	0.60	0.22	0.10
Fastest Agent	0.67	0.31	0.04
Best Potential	0.60	0.27	0.14
Nearest Agent	0.69	0.41	0.10
Extended hand coded	0.64	0.44	0.16

Table 5 summarizes the performance of the hand-coded strategies on the three real-life based scenarios (6 – 8) depicted in Figure 12-Figure 14. All the strategies perform well in Scenario 6. This may be due to the fact that scenario is relatively easy due to sparsely spread obstacles. Scenario 7 has groups of victims within large wall blocks. Since there are few fire cells, accuracy might not be an important factor in deciding the performance of the agents. Because it is relatively harder for agents to reach victims and target areas, agents nearby the region where the trigger occurs have a better chance of completing the goals and hence the nearest agent strategy fairs well in this scenario. Scenario 8 shows that speed of the agent might not be the most important physical capability to be considered in reorganization. This scenario requires the organizational master to extract the different situations within the simulator to make decision on

assigning agents in response to triggers. For agents to accomplish their tasks in this scenario, they have to possess a good overall mix of individual physical capabilities. So, even though the difference between performances of the different strategies is small, a strategy that chooses agents with different physical capabilities for different situations performs well.

In Table 6, I present the average score of the hand-coded strategies over all scenarios. In individual scenarios, we saw that some strategies seem to work better than others but other than the extended hand coded strategy, Fastest Agent strategy and the Nearest Agent strategy, the rest seem to reach the same performance level if we average on all scenarios. The extended hand coded strategy performs marginally better than the fastest agent strategy since it allows for some flexibility in choosing different agents for different situations.

**Table 6. Hand-coded Strategies – Average Scores over all scenarios**

Strategy	Average
Most Accurate Agent	0.35
Longest Sonar	0.35
Fastest Agent	0.40
Best Potential	0.35
Nearest Agent	0.38
Extended hand coded	0.42

The hand-coded strategies used in these experiments either extracted specific information from the physical capabilities of the agents or used a dynamic condition like nearness to the target region to reorganize the agents. Even though these strategies use different information for making the reorganization decisions, on average, all of them scored approximately in the same range of overall average raw scores. This is confirmed when the same tests were repeated with 2000 simulation cycles. The result for the 2000 cycle runs for scenarios 6 – 8 and the average over all scenarios is reported in Table 7.

These results suggest that, while no hand-coded strategy was really bad, there is no one significant strategy that outperforms the others. In the next Section, I summarize the results of evolutionary reorganization strategies.

**Table 7. Hand-coded Strategies - extended cycles**

Strategy	Scenario6	Scenario7	Scenario8	Total
Most Accurate Agent	0.73	0.43	0.14	0.46
Longest Sonar	0.71	0.39	0.11	0.46
Fastest Agent	0.76	0.41	0.15	0.47
Best Potential	0.73	0.42	0.14	0.46
Nearest Agent	0.72	0.41	0.11	0.47
Extended hand coded	0.73	0.49	0.22	0.50

## 5.4 Evolutionary reorganization strategies

The standard GP experiments mentioned in Section 5.2 yielded better results than the hand-coded strategies discussed in the previous Section. In many cases, the best individual strategy for each scenario used the relevant variables as predicted for the scenario in Section 3.3.1. In Table 4 and Table 5, we see that the best individuals for each scenario perform better than any of the hand-coded strategies. In this Section, I describe the best evolutionary strategy found for each scenario and the evolutionary strategy that performed the best over all scenarios. The structure of an evolutionary strategy is described in 4.3.2. Table 8 summarizes the scores of the best individuals obtained for each strategy. Comparing these values with those in Table 4 and Table 5, we see that the best individuals for each scenario perform better than any of the hand-coded strategies. In this Section, I describe the best evolutionary strategy found for each scenario and the evolutionary strategy that performed the best over all scenarios. I also compare the performance of best strategies of each scenario on other scenarios. The structure of an evolutionary reorganization strategy is described in Section 4.3.2.

### 5.4.1 Scenario 1

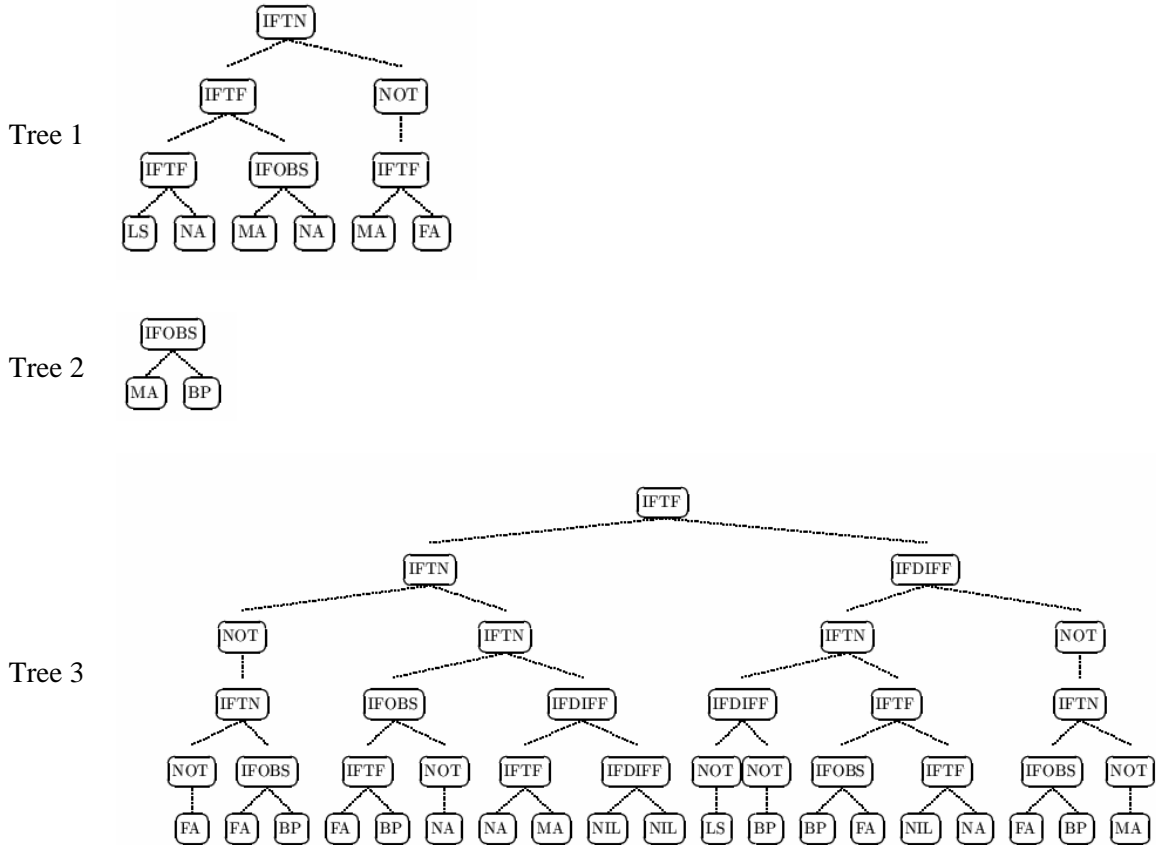
As noted in Section 3.3.1, scenario 1 is the simplest of the scenarios. In Figure 19, Tree 1 corresponds to the new victim found trigger, Tree 2 corresponds to search failure trigger and Tree 3 corresponds to rescue failure. I follow the same format in all subsequent sections. According to the reorganization strategy in Figure 19, when a new victim is found, if it is far from the safe area, then the fastest available agent is assigned to rescue the victim. If the victim is not far, but also not near the safe area, the most accurate agent is chosen for rescuing the victim.

**Table 8. Best scores for individual scenarios**

Scenario	Score of Best Individual in the scenario
1	0.97
2	0.70
3	0.80
4	0.36
5	0.30
6	0.81
7	0.45
8	0.31

Since the obstacles in this scenario are minimal,  $IFOBS$  condition is always negative. Hence for search failures, the organizational master always chooses the searcher with the best potential. The left sub tree in the rescue failure tree in most cases evaluated to a nil. This means that if a rescuer fails to rescue a victim far from the safe area, most probably, the rescue task will not be reassigned. The right sub tree in almost all cases evaluate to the agent with the best potential score. In this scenario, we see that even with such a complicated strategy, the reorganization decisions are mostly fixed. This does not affect the performance of the strategy because in the scenario, there are no victims placed near the safe area to begin with. Since the scenario is very simple, the possibility of a rescuer reaching an area nearer to the safe area and still failing to complete its task is also less. Hence, the  $IFDIFF$  condition is mostly evaluated negatively. Since the scenario does not exhibit complexity, most of this reorganization strategy is never evaluated.

The strategy achieved an average of 0.97 on the scenario. In all the hand-coded strategies, the Nearest Agent strategy got the best values for this scenario. This strategy makes use of only one piece of information about the scenario. Compared to it, the best individual strategy for this scenario made use of many other aspects of the scenario and so, it might have reached a better score than the best hand-coded strategy.

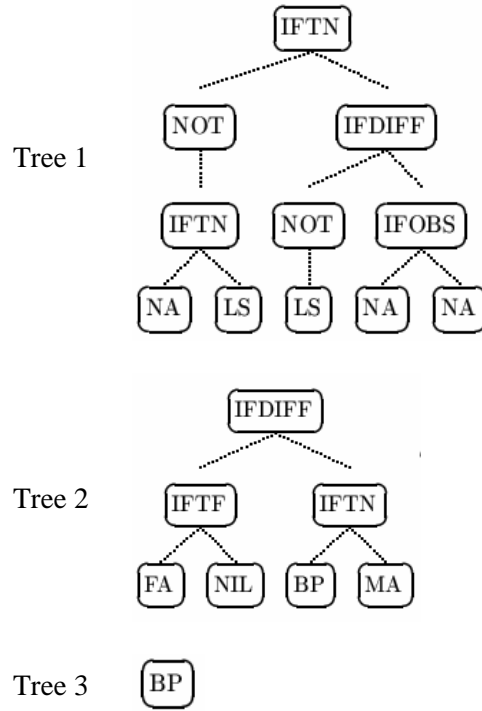


**Figure 19. Best Individual for Scenario 1**

### 5.4.2 Scenario 2

Scenario 2 consists of randomly distributed obstacles and victims. The best individual reorganization strategy obtained is displayed in Figure 20. It scored an average of 0.70 on scenario 2. Based on this strategy, if a victim is found near the safe area, the organizational master picks an available rescuer that has the best sonar capability. If the victim is not located near the safe area and is known to be difficult to reach it, the organizational master selects a rescuer with the least sonar capability. But, if the rescue of the victim is known not to be difficult, then the nearest rescuer is assigned to rescue the victim.

If a searcher fails to accomplish its role, then, the strategy suggests that the organizational master should consider the difficulty of exploration of the region and the distance of the region from the safe area to decide which searcher to assign the role of the failed searcher.



**Figure 20. Best Individual for Scenario 2**

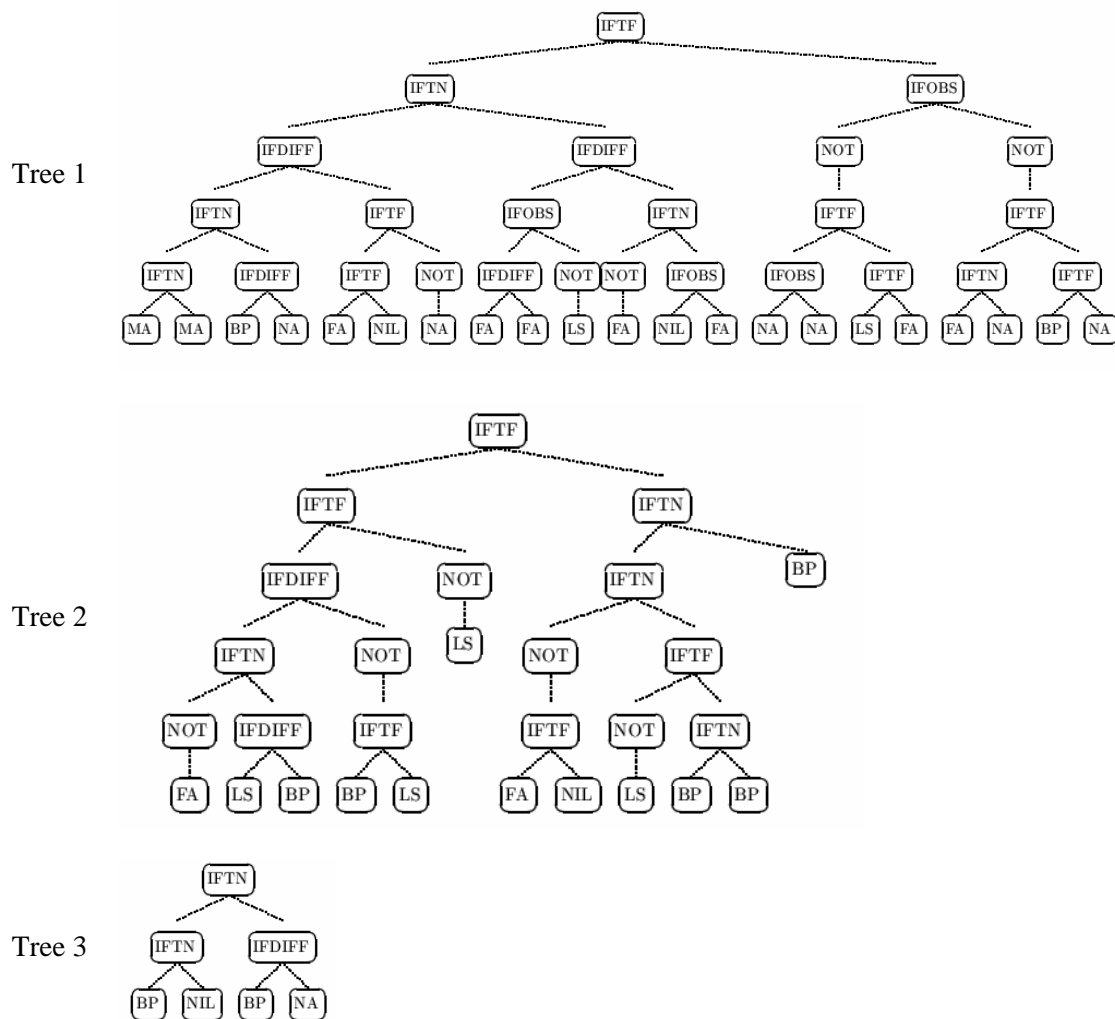
For a rescue failure, the strategy always chooses an available agent with the best potential score for rescue. As mentioned in Section 5.3, in scenario 2, none of the hand-coded strategies are effective because all physical capabilities were important in the scenario. The best individual here further illustrates this by choosing agents with different physical capabilities for different situations to yield better results.

### 5.4.3 Scenario 3

The best individual for scenario 3 achieved an average score of 0.80 on this scenario. This scenario consists of groups of victims placed at different locations in the region with very few obstacles. Hence, `IFOBS` condition will always be negative. Hence, when a new victim is found, the strategy looks for the following conditions. If the victim is far from the safe area and if it is known that there haven't been any failed attempts to rescue the victim, then, the fastest available rescuer is assigned to rescue the victim. The fastest available rescuer is assigned to rescue the victim if the victim is near to the safe area. If it is known that repeated attempts to rescue the victim have failed and it is difficult to rescue the victim, then an available rescuer with the least



sonar capabilities is assigned to rescue the victim. If the victim is neither far from the safe area nor near the safe area, then the nearest available rescuer is assigned to rescue it.

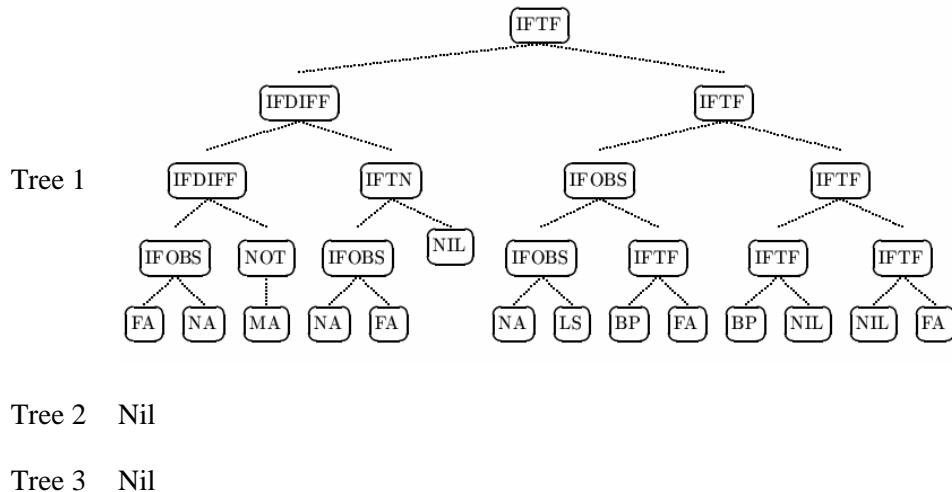


A failed rescuer near the safe area or a failed rescuer attempting to rescue a victim that is known to be difficult to rescue, is replaced by a rescuer with the best potential. Else, it is replaced by the nearest available rescuer.

By looking at the scenario, it was expected that the reorganization strategy should make use of the relative location of the victims to boost its performance. But the best hand-coded strategy for this scenario was the Fastest Agent strategy. The best individual evolutionary strategy further refined this hand-coded strategy by making use of the fast agents in situations where it was safe to use them and in other situations, it selected different agents. This might be the reason for the boost in performance of this strategy.

#### 5.4.4 Scenario 4

Scenario 4 is characterized by groups of victims between blocks of obstacles. Hence, in many cases, the `IFOBS` condition returns positive. The `IFOBS` and `IFDIFF` conditions in this case are independent of each other because a victim that is not surrounded by obstacles evaluates `IFOBS` as negative. But, to reach the victim, an agent might have to overcome difficulties on its path and hence `IFDIFF` can be positive. The same might be true vice versa. The best hand-coded strategy for this scenario was the Longest Sonar strategy. The best individual strategy for this scenario is displayed in Figure 22. It scored 0.36 on the scenario. In this strategy, when a new victim is found, if it is far from the safe area and is known to be difficult to rescue and has a lot of obstacles in the way, the fastest rescuer is assigned to do the job.



**Figure 22. Best Individual for Scenario 4**

But, if there are fewer obstacles, then the strategy picks the nearest rescuer to rescue the victim. If the new victim is far and not difficult to rescue, then the strategy will not assign any rescuer to

rescue that victim. If the victim is not far from the safe area, then the strategy picks the fastest available rescuer to rescue the victim.

### 5.4.5 Scenario 5

**Figure 23. Best Individual for Scenario 5**

around it and is far from the safe area, then the rescuer with least potential score is assigned to rescue it, but if it is not far and not near the safe area, then the rescuer with the best sonar capability is assigned for the job. If the victim does not have obstacles around it and is far from the safe area, the nearest available rescuer is assigned to rescue it. If there are no obstacles and the victim is not known to be difficult to rescue and is not far from the safe area, the most accurate rescuer available is assigned to rescue it. In all other cases, no agent is assigned to rescue the victim.

When a searcher fails to complete its goal, then if it is near the safe area, then the searcher with the best sonar capability is assigned to replace the failed searcher. If the failed searcher is not near the safe area, then the searcher that is farthest from the searcher is assigned to replace it.

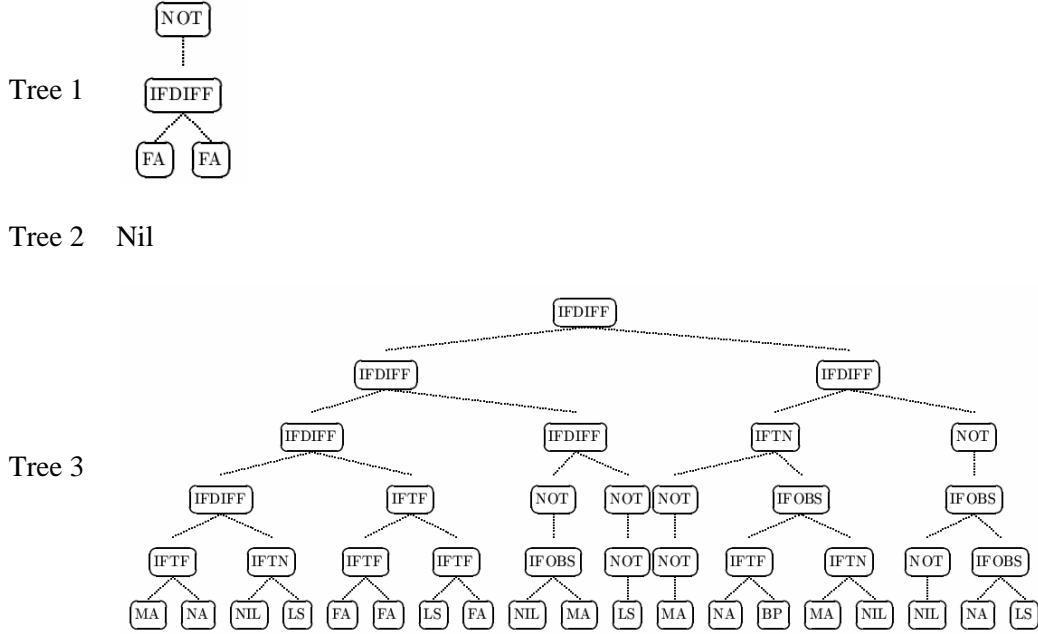
When a rescue goal fails, if the victim that was being rescued is known to be difficult to rescue, then the most accurate rescuer is assigned to rescue it.

In this strategy, the combination of assignments when a new victim is found and when the rescue fails is quite complex. The strategy seems to first rescue the victims in the border, those that do not have obstacles around them. The farthest block of victims in this scenario are surrounded by obstacles to begin with, the strategy assigns the rescuers with the least potential to rescue those victims. When a few of these victims are partially rescued or if the low potential score rescuers fail to complete the goal, then the victims' difficulty of rescue increases and more accurate agents are assigned to the rescue. For search failures, the strategy takes a different approach. When a searcher that is not near the safe area fails (the chances of a searcher to fail if it is near the safe area is low since the scenario does not have any objects in that region), the strategy assigns the farthest available searcher. This might ensure that, when a searcher fails, it is better to start the search process from a different path.

This strategy scored 0.30 while the best hand-coded strategy scored 0.17 for the same scenario. The best strategy for this scenario scored less than what the best strategy for previous scenario scored on that scenario. Since the previous scenario had more wall obstacles, it might have been easy for the best strategy there to take risk and target performance rather than survival as this strategy does.

#### **5.4.6 Scenario 6**

This scenario is part of the three real life scenarios. The best individual strategy for scenario 6 is shown in Figure 24. It scored 0.81 on an average on this scenario.



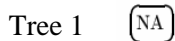
**Figure 24. Best Individual of Scenario 6**

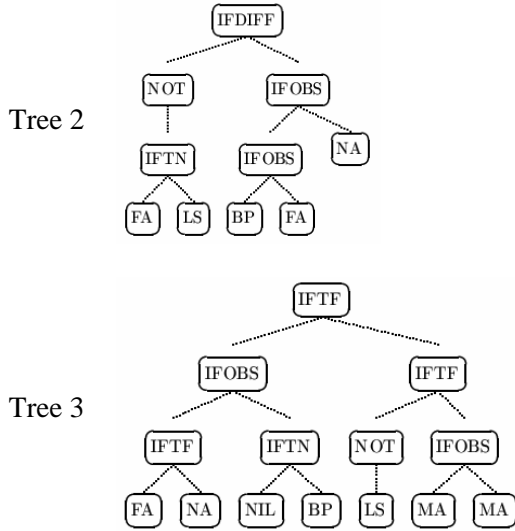
The best hand-coded strategies scored 0.69 on this scenario. The strategy assigns the fastest available rescuer to rescue a newly found victim. There is no replacement for a failed searcher and when a rescuer fails, if the victim the failed rescuer was pursuing is known to be difficult to rescue and is far from the safe area, then the most accurate rescuer is assigned as the replacement. If it is not far from the safe area, the nearest rescuer is given the job. If it is not known to be difficult to rescue the victim that the failed rescuer was pursuing and if there are obstacles around the victim then the nearest rescuer is assigned to rescue the victim.

This strategy uses a combination of the best hand-coded strategies for this scenario. As mentioned in Section 5.3, this scenario is relatively easy and hence, the absence of a search failure tree does not affect the performance.

#### 5.4.7 Scenario 7

The best individual strategy for scenario 7 is shown in Figure 25. It scored a score of 0.45 The Nearest Agent strategy was the best hand-coded strategy for this scenario scoring 0.41. As seen in the previous scenario, this strategy uses the best hand-coded strategy but with slight modifications. The performance improvement is also less.





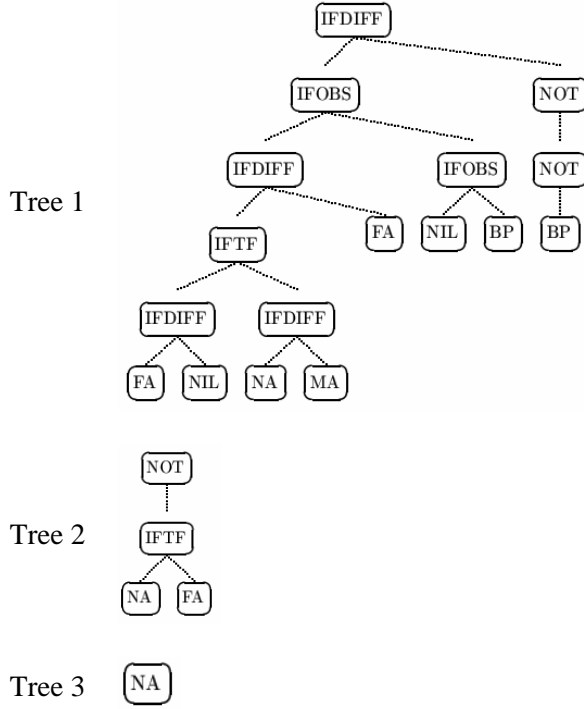
**Figure 25. Best Individual for Scenario 7**

The strategy chooses the nearest available rescuer to rescue a newly found victim. When a searcher fails, if it is exploring an area which is known to be difficult to explore and is near the safe area, the strategy picks a searcher with the best sonar capability. If the area of exploration is far, the fastest available searcher is chosen as the replacement. If the search area is not known to be difficult to explore but there are obstacles in it, then the searcher with the best potential score is chosen for the job. If there are no obstacles in the area, then, the nearest searcher is assigned to accomplish the search. When a rescuer fails, if the victim it is pursuing is far from the safe area and is known to be surrounded by obstacles, the fastest rescuer is chosen to continue the rescue. If there are no obstacles found near the location, a rescuer with the best potential is picked to continue the rescue. If the victim is not far from the safe area, then the most accurate rescuer available is chosen.

The performance of the best evolutionary strategy in the case of scenario 7 was marginally better than that of the best hand-coded strategy. As discussed in 5.3.1, the Nearest Agent strategy was the most intuitive for this scenario. I think the refinement in the hand-coded strategy lead to the improvement of the performance.

#### 5.4.8 Scenario 8

Figure 26 shows the best individual strategy for scenario 8. It scored 0.31 on the scenario. The best hand-coded strategy for this scenario was Best Potential Agent strategy with a score of 0.14.



**Figure 26. Best Individual for Scenario 8**

In this strategy, if the newly found victim is known to be difficult to rescue and is known to be surrounded by obstacles and is far from the safe area, the fastest available rescuer is assigned to rescue it. If it is not far from the safe area, the nearest rescuer is picked to rescue it. If the new victim is difficult to rescue and does not have any obstacles surrounding it or if rescuing it is not difficult, the rescuer with the best potential score is assigned to rescue it.

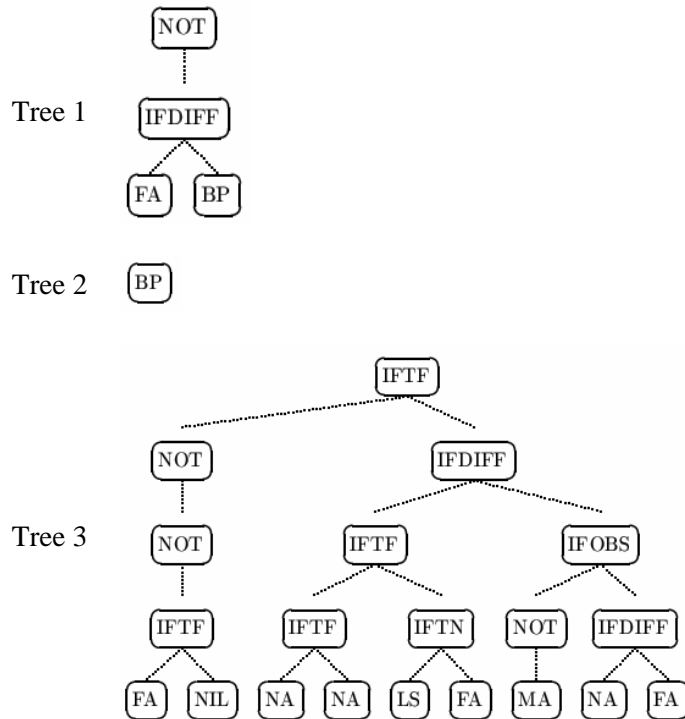
When a searcher fails, if its area of exploration is far from the safe area, the fastest searcher is picked to complete the exploration. If it is not far from the safe area, the nearest searcher is assigned the failed searcher's goal. When a rescuer fails, the strategy replaces it with the nearest available rescuer.

The best strategy for scenario 8 suggests that before a victim is difficult to rescue (after rescuing it fails multiple times), the hand-coded strategy still works. But, when it is recognized that the victim is difficult to rescue, then depending on the obstacles surrounding the victim and the distance from the safe area, the strategy looks for features of available rescuers like speed and nearness to the victim.

### 5.4.9 All scenarios

In this Section, I explain the evolutionary reorganization strategy with the highest average score over all scenarios. To obtain the best evolutionary reorganization strategy over all scenarios, I used the same setup as described in Section 5.2, except that instead of using the average raw score of one scenario, the fitness awarded to the individual is the average raw score over all scenarios. All the scenarios are given equal weight in arriving at the fitness of an individual strategy. The best individual strategy over all scenarios is shown in Figure 27.

The strategy assigns the fastest available agent to a newly found victim if it is found to be difficult to rescue. Otherwise, it assigns an agent with the best potential score to rescue the victim.



**Figure 27. Best Individual for all scenarios**

When a searcher fails to explore its target region, the strategy chooses the searcher with the best potential score to replace it. When a rescuer fails to complete its goal, if its victim is far from the safe area, the strategy chooses the fastest available rescuer to replace it. If the victim is near the safe area and is difficult to rescue, the rescuer with the best sonar capability is used to replace the failed rescuer. If the victim is neither far nor near the safe area and is difficult to rescue, the fastest available rescuer is assigned the failed rescuer's goal. If the victim is surrounded by obstacles but is not known to be difficult to rescue and is not far from the safe area, then the



rescuer with least accuracy is chosen to replace the failed rescuer. If the victim is neither far nor difficult and no obstacles are found near it, the fastest available rescuer is used to replace the failed searcher.

This strategy suggests that it is better to assign an agent with the best potential score to rescue a newly found victim when it is not known whether the victim is difficult to rescue or not. In the long run, it predicts that it is best to replace a failed searcher with an available searcher with the best potential score. For replacing a failed rescuer, the strategy suggests that the decision has to be based on the nearness of the victim to the safe area,

Table 9 compares the score of the best overall strategy on all scenarios with the highest scores on each scenario obtained by hand-coded strategies and evolutionary strategies. It shows that while the best strategy over all the scenarios is not the best strategy to any of the scenario, it performs better than the hand-coded strategy in all scenarios except scenario 7.

**Table 9. Score of overall best strategy**

Scenario	Score of the best strategy over all scenarios	Score of the best hand-coded strategy in each scenario	Score of best evolutionary strategy in each scenario
1	0.91	0.86	0.97
2	0.64	0.52	0.70
3	0.78	0.67	0.80
4	0.33	0.24	0.36
5	0.28	0.19	0.30
6	0.80	0.69	0.81
7	0.33	0.44	0.45
8	0.29	0.16	0.31

Table 10 summarizes the average performance of all the strategies on all scenarios we have considered so far. It shows that while the best strategies for each scenario need not be the best overall strategies. Another thing to note is that, when average performance on all scenarios are concerned, just as was the case with hand-coded strategies, the scores do not vary much. In Table

10, we see that the difference between the lowest score by an evolutionary strategy and the best evolutionary strategy is just 0.08. Among the hand-coded strategies, this difference is 0.07.

**Table 10. Average score of all strategies over all scenarios**

Strategy	Average	Strategy	Average
Most Accurate Agent	0.35	Best of scenario 3	0.49
Longest Sonar	0.35	Best of scenario 4	0.51
Fastest Agent	0.40	Best of scenario 5	0.52
Best Potential	0.35	Best of scenario 6	0.49
Nearest Agent	0.38	Best of scenario 7	0.50
Extended hand coded	0.42	Best of scenario 8	0.52
Best of scenario 1	0.46	Best of all scenarios	0.54
Best of scenario 2	0.45		

## 5.5 Summary of results

In this chapter I explained the experiments that were conducted to test the search and rescue simulator and the different strategies proposed for reorganization of the agent organization. The results show that there is a marked difference between the performance of hand-coded strategies and the performance of evolutionary strategies, both in individual scenarios and in average performance over all scenarios. A complete table of scores, the s-expressions of the best individuals of each scenario and the best over all scenarios is given in Appendix B. The best fitness plots show the learning curve for the best performance in each generation for each scenario. A generation to generation comparison of performance for all scenarios can be seen by plotting the average fitness in each generation for each scenario. The graphs of best fitness (with error bars) and average fitness in each generation of the evolution for all the scenarios are presented in Appendix B.3. The lines parallel to the x-axis represent the score of the best hand coded strategy for each scenario. These graphs show that improvement in performance of evolutionary strategies across generations occurs; although at a varied rate for each scenario. The error bars also show that the magnitude of variation from the mean point for the best scores is comparatively small. The average performance on each scenario also shows an improving trend

across generations. It can be seen from the graphs that more generations are required for the fitnesses to reach a threshold in each of the scenarios. By plotting the performance of the extended hand coded strategy and the best overall evolutionary strategy, we can see the improvement in the performance of the organization by using evolutionary strategies. This graph is presented in Appendix B.3.9. In the next chapter, I state the conclusions I draw from these results.

## Chapter 6 Conclusions and Future Work

I designed and implemented the search and rescue organization using the OMACS model. I compared reorganization strategies that were both hand-coded and evolved using genetic programming. The results show that evolutionary reorganization strategies perform better than hand-coded strategies both on individual scenarios and on average on all scenarios. Hence, we can affirm that the hypothesis stated in Section 1.1.1 is valid. While it is already known that environment and problem specific information can increase the efficiency and robustness of an organization [33], this study shows that such information might not be readily available and in such situations, an evolutionary reorganization framework provides an efficient solution.

### 6.1 Conclusions

The time constraints of training reorganization strategies using genetic programming are significant but as mentioned in Section 5.2.1, once we arrive at reorganization strategies, the time overhead during a simulation cycle is not that high. Moreover, once a strategy is evolved; we can optimize the performance by only calculating the required values. Hence, the problems mentioned in Section 5.2.1 can be minimized. Results show that the best individual strategies for difficult scenarios performed relatively well on other scenarios also suggesting that fewer training scenarios might have been sufficient for evolving reorganization strategies that perform well on an average on all scenarios.

The performances of the search and rescue roles were bound by the exploration and rescue algorithms used in the simulator. The physical capability of the agents also put an upper threshold on the performance of the organization as a whole. Hence, the gain to be had from reorganization although significant is limited.

The results also show that there are indirect effects of reorganization that affect further assignments late in the simulation or in responding to a later trigger that rose during the same simulation cycle. For example, it is not clear why the distance from the safe area should be considered for deciding on the assignment for a searcher. This may have an indirect effect on the performance of the strategy because this way, an agent may start over the search process from a different position or allow the reorganization policy in a particular scenario to extract ease of exploration information near the vicinity of the safe area.

## 6.2 Future Work

The results of this study open an interesting set of questions concerning reorganization in organizations. In this study, we deliberately chose to evolve reorganization strategies while the conditions for triggering reorganizations were kept static. Also, reorganization did not entail reassignment of all agents in the organization but only those that were available at the time of the organization. Hence, irrespective of the kind of reorganization strategy being used, a better way of deciding when to reorganize and a better way of choosing the agents for reorganization would only add to the performance of the organization as a whole. This would be an interesting topic for further research.

As mentioned in the previous section, we know that the performance of organization is also bound by the quality of the algorithms used to implement the different roles in the organization. The `achieves(r, g)` score assigns a value on the role during the design phase based on the expectation of the programmer on how well the role *r* achieves the goal *g*. While this captures the `potential` of the agent at any particular instant during the execution, it is not clear on how this score actually affects the average performance of the organization.

Within an evolutionary reorganization strategy there are competing conditions and hence, each reorganization decision has complex effects on the overall performance of the organization. But these effects were discussed and substantiated only qualitatively. It would be interesting to measure the individual affects of these competing conditions so that if a generalized policy exists, it can be incorporated in the design process itself. This can be done by tracking the number of times each of the leaf nodes in each strategy were chosen for reorganization. Based on this information, one can observe which individual affect actually dominates.

## REFERENCES

1. R.C. Arkin, T. Balch. "Cooperative multiagent robotic systems," AI based Mobile Robots: Case Studies of Successful Robot Systems. D. Kortenkamp, R. P. Bonasso, and R. Murphy, eds., MIT Press, 1998.
2. P.M. Blau, W.R. Scott, Formal Organizations, Chandler, San Francisco, CA, 1962, pp.194-221.
3. L. Bull ed., Applications of Learning Classifier Systems Series: Studies in Fuzziness and Soft Computing, Vol. 150, Springer-Verlag, 2004.
4. Y.U. Cao, A. Fukunaga, and A. Kahng, "Cooperative mobile robotics: Antecedents and directions," Autonomous Robots, vol. 4, Kluwer Academic Publishers, 1997 , pp. 1-23.
5. K.M. Carley, Z. Lin, "A theoretical study of organizational performance under information distortion," Management Science, vol. 43, no. 7, 1997, pp. 976-997.
6. K.M. Carley. "Adaptive organizations and emergent forms", Organization Science, 769(2-3), 1998.
7. S. DeLoach. "Multiagent Systems Engineering of Organization-based Multiagent Systems," in A. Garcia et al. (Eds.): SELMAS 2005, LNCS 3914, pp. 109 – 125, 2006.Springer-Verlag, Berlin Heidelberg, 2006.
8. S. DeLoach, W.H. Oyenon, An Organizational Model and Dynamic Goal Model for Autonomous, Adaptive Systems, tech. report MACR-TR-2006-01, Computing and Information Sciences dept. Kansas State University, 2006.
9. S. DeLoach. "The MaSE Methodology", Methodologies and Software Engineering for Agent Systems. The Agent-Oriented Software Engineering Handbook Series : Multiagent Systems, Artificial Societies, and Simulated Organizations , Vol. 11. Bergenti, Federico; Gleizes, Marie-Pierre; Zambonelli, Franco eds., Kluwer Academic Publishing, 2004.
10. S. DeLoach, E. Matson, Y. Li. "Exploiting Agent Oriented Software Engineering in the Design of a Cooperative Robotics Search and Rescue System," The International Journal of Pattern Recognition and Artificial Intelligence, 17 (5), August 2003, pp. 817-835.
11. V.A. Dignum, A Model for Organizational Interaction: Based on Agents, Founded in Logic. PhD thesis, Utrecht University, 2004.
12. P. Dillenbourg, ed., Collaborative learning. Cognitive and computational approaches, Pergamon Press. 1999. Ch 4, pp. 64-80.
13. A. Drogoul, J.D. Zucker. "Methodological issues for designing multi-agent systems with machine learning techniques: Capitalizing experiences from the robocup challenge," tech. report LIP6 1998/041, Laboratoire d'Informatique de Paris 6, October 1998.
14. R. Espejo, W. Schuhmann, M. Schwaninger, and U. Bilello, Organizational Transformation and Learning: A Cybernetic Approach to Management, John Wiley & Sons, 1996.
15. M. Hannoun, O. Boissier, J.S. Sichman, C. Sayettat, "MOISE: An Organizational Model for Multi-agent Systems," Proc. of Int'l Joint Conf. IBERAMIA-SBIA, Lecture Notes in Artificial Intelligence, Springer Verlag, vol. 1952:, 2000, pp 156-165.
16. J.H. Holland, Adaptation in Natural and Artificial Systems. University of Michigan Press. 1975.

17. B. Horling, V. Lesser, "Using ODML to Model and Design Organizations for Multi-Agent Systems". Proc. of Workshop on From Organizations to Organization Oriented Programming (OOP 05), July 2005, pp. 33-48.
18. B. Horling, V. Lesser, "A Survey of Multi-Agent Organizational Paradigms," The Knowledge Engineering Review, vol. 19, no. 4, Cambridge University Press, 2005, pp. 281-316.
19. T. Kovacs, "A Learning Classifier Systems Bibliography", Dept. of Computer Science, Univ. of Bristol, <http://www.cs.bris.ac.uk/~kovacs/lcs/search.html>, 2002.
20. J.R. Koza, D. Andre, F.H. Bennett III, and M. Keane. Genetic Programming 3: Darwinian Invention and Problem Solving. Morgan Kaufman, April 1999.
21. J.R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, 1994.
22. C. Lematre, C.B. Excelente, "Multi-agent organization approach", Proc. of the 2nd Iberoamerican Workshop on DAI and MAS, Toledo, Spain, 1998.
23. S. Luke, Issues in Scaling Genetic Programming: Breeding Strategies, Tree Generation, and Code Bloat, doctoral dissertation, Dept. of Computer Science, Univ. of Maryland, College Park, MD, 2000.
24. E. Matson, S. DeLoach, "Capability in Organization Based Multi-agent Systems," Proc. of the Intelligent and Computer Systems (IS '03) Conference, 2003.
25. MESSAGE: Methodology for Engineering Systems of Software Agents. Deliverable 1. Initial Methodology. July 2000. EURESCOM Project P907-GI.
26. A. Namatame, T. Tsukamoto, "Learning agents for cooperative hyperinformation systems," Proc. International Conference on Intelligent and Cooperative Information Systems, 1993, pp. 124-133.
27. L. Panait, "Learning Team Behaviors with Adaptive Heterogeneity," Proc. of the Fourth Int'l Joint Conf. on Autonomous Agents and Multi Agent Systems (AAMAS-2005), 2005, pp. 1382.
28. S.J. Russell and P. Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995.
29. P. Stone and M. Veloso, "Multiagent Systems: A survey from a machine learning perspective," Autonomous Robots, vol. 8 no. 3, July 2000, pp. 345-383.
30. K.P. Sycara: "Multiagent Systems," AI Magazine, vol. 19, no 2, summer 1998, pp. 79-92.
31. C. Tacquard-Choulet, P. Baptiste, "MUCO: a cognitive system for problem solving based on self-organization: a "multi-agents" implementation with blackboard," Systems, Man and Cybernetics, Proc., International Conference on Systems Engineering in the Service of Humans, no. vol.4, 17-20 Oct 1993, pp. 131-136.
32. K. Takadama, S. Nakasuka and T. Terano. "Printed Circuit Board Design via Organizational - Learning Agents", Applied Intelligence, Vol. 9, No. 1, 1998, pp. 25-37.
33. M. Tambe, and W. Zhang, "Towards flexible teamwork in persistent teams: extended report.," Journal of Autonomous Agents and Multi-agent Systems, Vol. 3, 2000, pp. 159-183.
34. E. Uchibe, M. Nakamura, and M. Asada, "Co-Evolution of Cooperative Behavior Acquisition in a Multiple Mobile Robot Environment," Proc. of the IEEE International Conference on Intelligent Robots and Systems, Vol. 1, 1999, pp. 4254-430.

35. M. Veloso, P. Stone, and K. Han. "The CMUnited-97 Robotic Soccer Team: Perception and Multi-agent Control. *Robotics and Autonomous Systems*," vol. 29, no. 2-3, November 2000, pp.133–143.
36. N. Vlassis, A concise introduction to multiagent systems and distributed AI. Informatics Institute, Univ of Amsterdam, Sept. 2003. <http://www.science.uva.nl/vlassis/cimasdai>.
37. G. Weiss, P. Dillenbourg. "What is 'multi' in multiagent learning?", *Collaborative Learning. Cognitive and Computational Approaches*, P. Dillenbourg, ed., Pergamon Press, 1999, pp. 64-80.
38. S.W. Wilson, Classifier Fitness Based on Accuracy. *Evolutionary Computation*, vol 3, no 2, 1995, pp. 149-76.
39. M.J. Wooldridge, *An Introduction to Multiagent Systems*, John Wiley and Sons, 2002
40. M.J. Wooldridge, N.R. Jennings, D. Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design". *Int'l. J. of Autonomous Agents and Multi-Agent Systems*, Vol. 3, No. 3, 2000, pp. 285 -312.
41. F. Zambonelli, N.R. Jennings, M.J. Wooldridge, "Organisational Rules as an Abstraction for the Analysis and Design of Multi-Agent Systems. *IJSEKE*. 11(3), June 2001, pp. 303-328.



# Appendix A Users Manual

## A.1 Overview

The software provided with this thesis includes the Search and Rescue Simulator and a modified version of the ECJ[23]. ECJ is licensed under the Academic Free License version 3.0.

## A.2 Running the software

All the commands below apply in the root directory of the software. The software runs in two modes: hand-coded strategies, evolutionary strategies.

In the hand-coded strategy mode, the software runs independently of ECJ. The simulator can run both in visible and silent modes. The command type for this mode is

```
java ksu.cis.sim.gui.Simulator [visible] [Search Agents] [Rescue Agents] [Both] [Scenario Number] [Repetitions] [Max X] [Max Y] [Cycles]
```

In the evolutionary strategy mode, a known evolutionary reorganization strategy can be tested or a new strategy can be evolved. The reorganization strategy to be tested is written in prefix notation in the file `ec\app\organization\ReRun.txt`. To run the simulator in this mode, type

```
java ec.Evolve -file ec\app\organization\ReRun.params -p stat.gather-full=true
```

The `ReRun.params` file specifies that there is a single individual in the population and the number of generations is set to one.

To evolve an evolutionary strategy, we use the parameter file `organization.params` to specify the genetic program. To run the simulator in this mode, type

```
java ec.Evolve -file ec\app\organization\organization.params -p stat.gather-full=true
```

For both these modes, the output is reported in `out.stat` in the root directory. We can modify `ec\app\organization\Reorganizer.java` to change the scenario settings for the evolutionary strategy.

### A.2.1 Scores

The raw score of the organization and the individual agent scores are reported in the directory `ksu\cis\sim\agentSystem\environment\data\Scores`. The raw scores are reported in the file `Organizerscores.csv` and the agents' scores are reported in the file `agentScores.txt`.

## Appendix B Reorganization Strategies

### B.1 Scores

Table 11. Performance of best individuals in all scenarios

Scenario	1	2	3	4	5	6	7	8	Total
Best Strategy of scenario									
1	<b>0.97</b>	0.44	0.77	0.21	0.15	0.71	0.28	0.15	0.46
2	0.79	<b>0.70</b>	0.66	0.29	0.22	0.57	0.23	0.16	0.45
3	0.86	0.46	<b>0.80</b>	0.24	0.20	0.80	0.31	0.22	0.49
4	0.81	0.57	0.72	<b>0.36</b>	0.25	0.79	0.30	0.27	0.51
5	0.86	0.59	0.76	0.29	<b>0.30</b>	0.75	0.37	0.23	0.52
6	0.90	0.50	0.70	0.31	0.12	<b>0.81</b>	0.40	0.22	0.49
7	0.91	0.62	0.69	0.22	0.13	0.76	<b>0.45</b>	0.27	0.50
8	0.82	0.68	0.74	0.28	0.23	0.78	0.30	<b>0.31</b>	0.52
All	0.91	0.64	0.78	0.33	0.28	0.80	0.33	0.29	<b>0.54</b>

### B.2 S-expressions of evolutionary reorganization strategies

#### B.2.1 Best Individual of Scenario 1

New victim found trigger:

(IFTN (IFTF (IFTF LS NA) (IFOBS MA NA)) (NOT (IFTF MA FA)))

Search Failure Trigger:

(IFOBS MA BP)

Rescue Failure Trigger:

(IFTF (IFTN (NOT (IFTN (NOT FA) (IFOBS FA BP))) (IFTN (IFOBS (IFTF FA BP) (NOT NA)) (IFDIFF (IFTF NA MA) (IFDIFF NIL NIL)))) (IFDIFF (IFTN (IFDIFF (NOT LS) (NOT BP)) (IFTF (IFOBS BP FA) (IFTF NIL NA))) (NOT (IFTN (IFOBS MA NA) (NOT MA)))))

## B.2.2 Best Individual of Scenario 2

New Victim Found Trigger:

(IFTN (NOT (IFTN NA LS)) (IFDIFF (NOT LS) (IFOBS NA NA)))

Search Failure Trigger:

(IFDIFF (IFTF FA NIL) (IFTN BP MA))

Rescue Failure Trigger:

BP

## B.2.3 Best Individual of Scenario 3

New Victim Found Trigger:

(IFTF (IFTN (IFDIFF (IFTN (IFTN MA MA) (IFDIFF BP NA)) (IFTF (IFTF FA NIL) (NOT NA))) (IFDIFF (IFOBS (IFDIFF FA FA) (NOT LS)) (IFTN (NOT FA) (IFOBS NIL FA)))) (IFOBS (NOT (IFTF (IFOBS NA NA) (IFTF LS FA))) (NOT (IFTF (IFTN FA NA) (IFTF BP NA))))))

Search Failure Trigger:

(IFTF (IFTF (IFDIFF (IFTN (NOT FA) (IFDIFF LS BP)) (NOT (IFTF BP LS))) (NOT LS)) (IFTN (IFTN (NOT (IFTF FA NIL)) (IFTF (NOT LS) (IFTN BP BP)) BP))

Rescue Failure Trigger:

(IFTN (IFTN BP NIL) (IFDIFF BP NA))

## B.2.4 Best Individual of Scenario 4

New Victim Found Trigger:

(IFTF (IFDIFF (IFDIFF (IFOBS FA NA) (NOT MA)) (IFTN (IFOBS NA FA) NIL)) (IFTF (IFOBS (IFOBS NA LS) (IFTF BP FA)) (IFTF (IFTF BP NIL) (IFTF NIL FA))))

Search Failure Trigger:

NIL

Rescue Failure Trigger:

NIL

## B.2.5 Best Individual of Scenario 5

New Victim Found Trigger:

```
(IFOBS (IFTF (IFTF (IFOBS (NOT BP) (IFTN FA MA)) (IFOBS (IFOBS LS MA)
(NOT NA))) (IFTN (IFDIFF (IFDIFF LS MA) (IFTF MA FA)) (IFTN (NOT LS)
(IFOBS LS BP)))) (IFDIFF (IFDIFF (IFDIFF (IFTF NA NIL) (IFDIFF NIL FA))
(IFTN (NOT MA) (IFTF LS LS))) (IFTF (IFOBS (NOT BP) (IFTF NA BP)) (IFOBS
(IFTF BP NA) (IFDIFF LS MA))))
```

Search Failure Trigger:

```
(NOT (IFTN (NOT NA) LS))
```

Rescue Failure Trigger:

```
(IFDIFF MA NIL)
```

## B.2.6 Best Individual of Scenario 6

New Victim Found Trigger:

```
(NOT (IFDIFF FA FA))
```

Search Failure Trigger:

```
NIL
```

Rescue Failure Trigger:

```
(IFDIFF (IFDIFF (IFDIFF (IFDIFF (IFTF MA NA) (IFTN NIL LS)) (IFTF (IFTF
FA FA) (IFTF LS FA))) (IFDIFF (NOT (IFOBS NIL MA)) (NOT (NOT LS))))
(IFDIFF (IFTN (NOT (NOT MA)) (IFOBS (IFTF NA BP) (IFTN MA NIL))) (NOT
(IFOBS (NOT NIL) (IFOBS NA LS))))
```

## B.2.7 Best Individual of Scenario 7

New Victim Found Trigger:

```
NA
```

Search Failure Trigger:

```
(IFDIFF (NOT (IFTN FA LS)) (IFOBS (IFOBS BP FA) NA))
```

Rescue Failure Trigger:

```
(IFTF (IFOBS (IFTF FA NA) (IFTN NIL BP)) (IFTF (NOT LS) (IFOBS MA MA)))
```

## B.2.8 Best Individual of Scenario 8

New Victim Found Trigger:

```
(IFDIFF (IFOBS (IFDIFF (IFTF (IFDIFF FA NIL) (IFDIFF NA MA)) FA) (IFOBS  
NIL BP)) (NOT (NOT BP)))
```

Search Failure Trigger:

```
(NOT (IFTF NA FA))
```

Rescue Failure Trigger:

NA

## B.2.9 Best Individual of all Scenarios

New Victim Found Trigger:

```
(NOT (IFDIFF FA BP))
```

Search Failure Trigger:

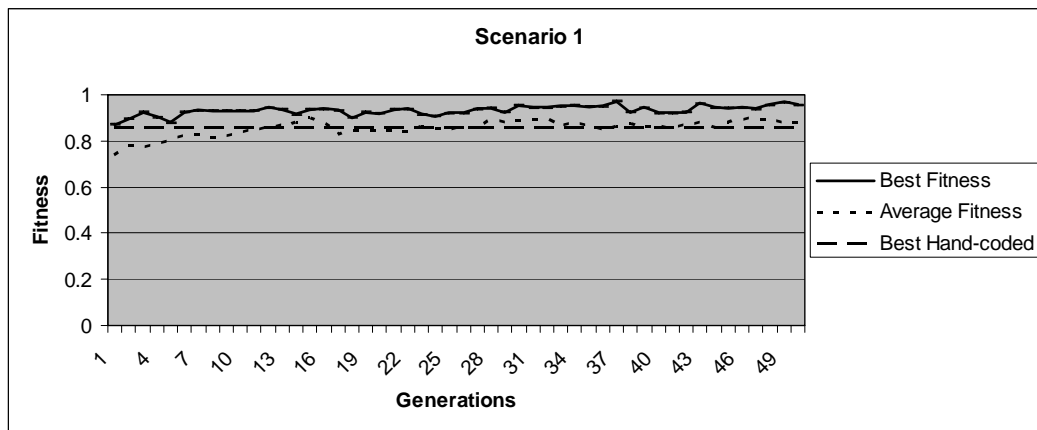
BP

Rescue Failure Trigger:

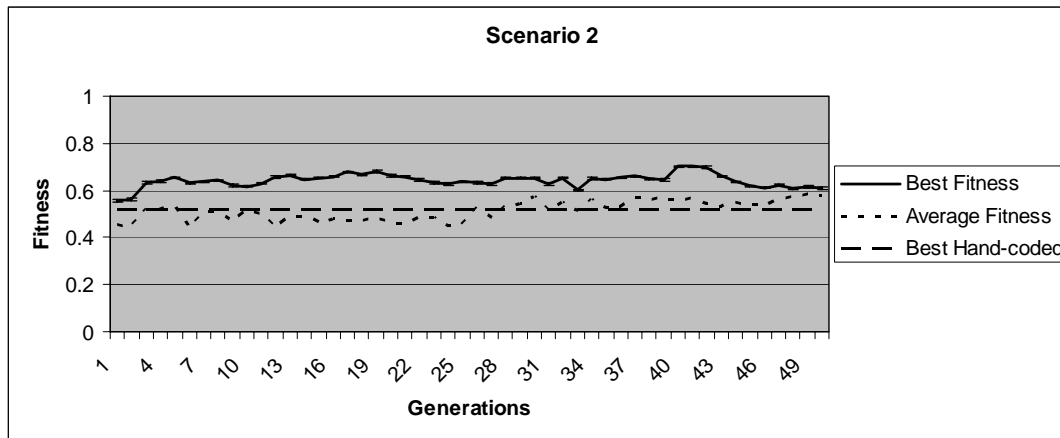
```
(IFTF (NOT (NOT (IFTF FA NIL))) (IFDIFF (IFTF (IFTF NA NA) (IFTN LS  
FA)) (IFOBS (NOT MA) (IFDIFF NA FA))))
```

## B.3 Best Fitness and Average Graphs

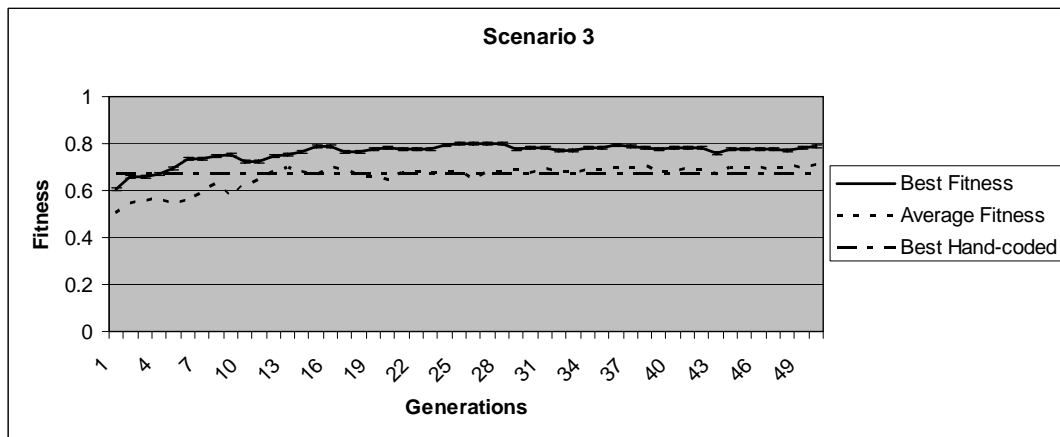
### B.3.1 Scenario 1



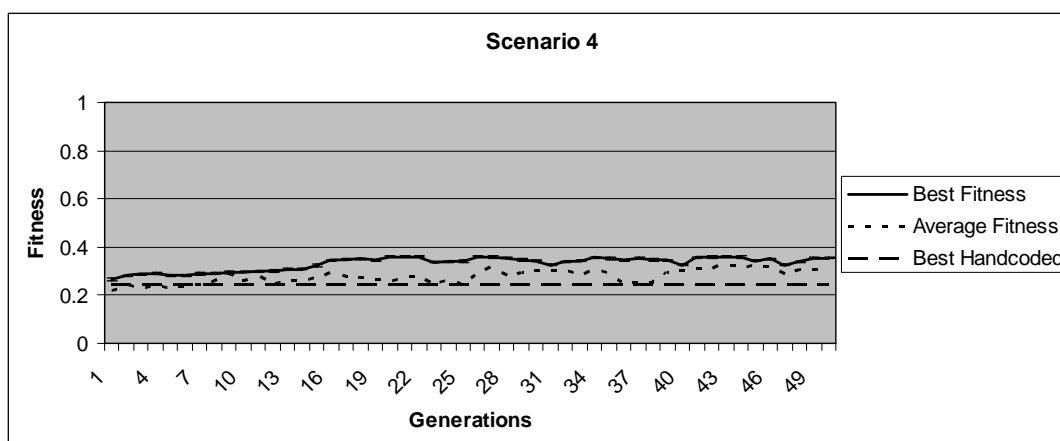
### B.3.2 Scenario 2



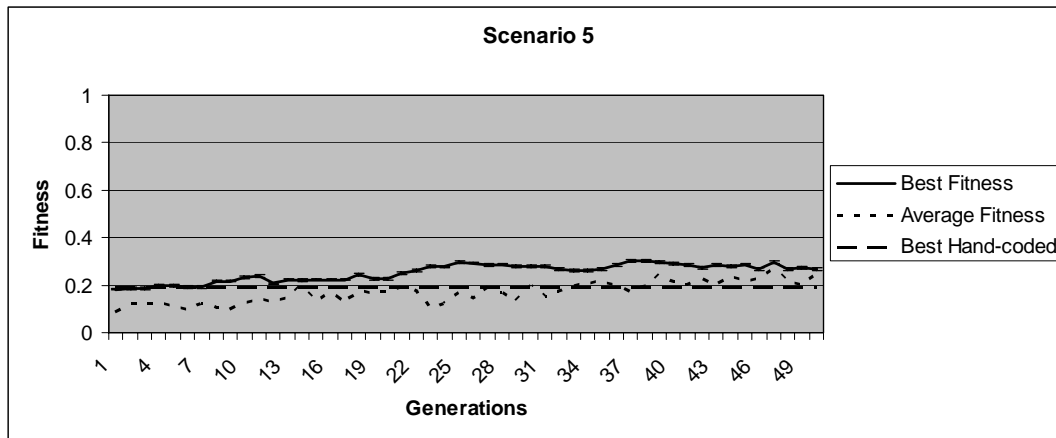
### B.3.3 Scenario 3



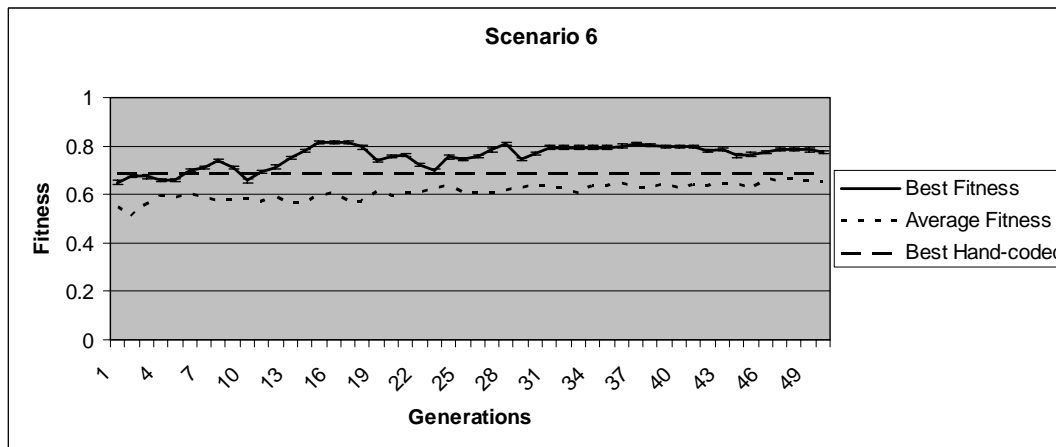
### B.3.4 Scenario 4



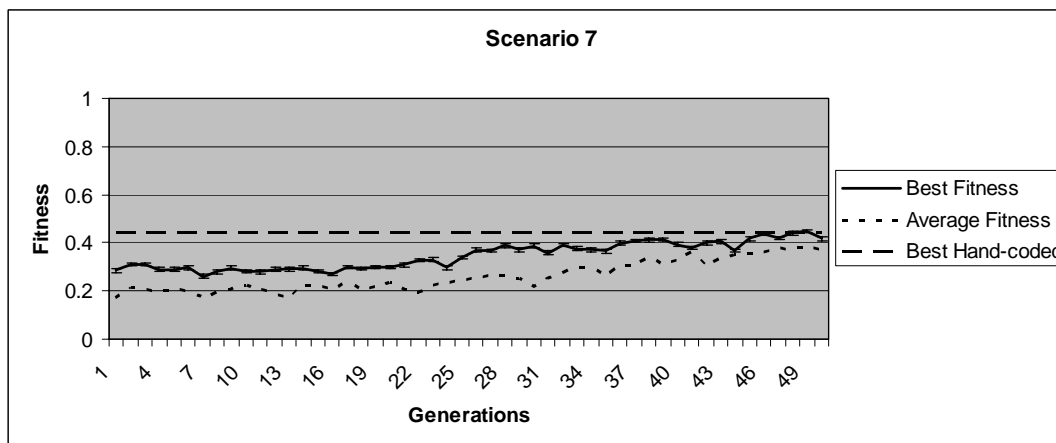
### B.3.5 Scenario 5



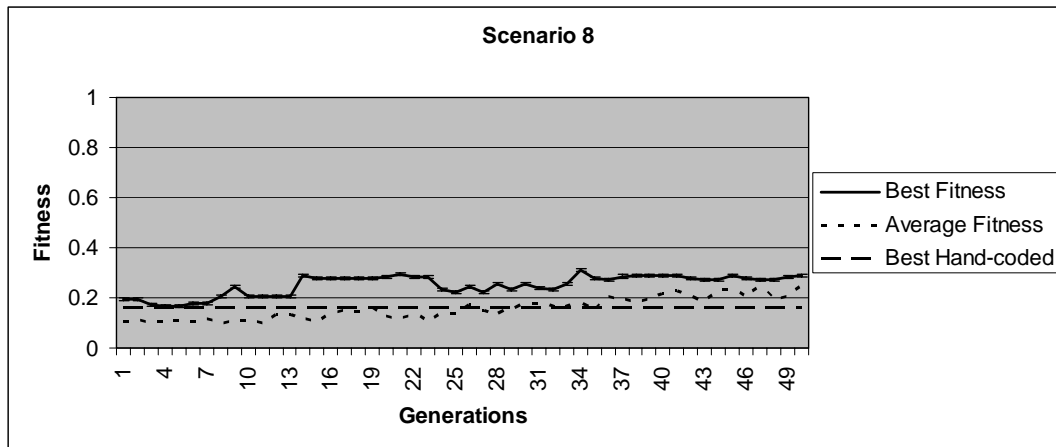
### B.3.6 Scenario 6



### B.3.7 Scenario 7



### B.3.8 Scenario 8



### B.3.9 All Scenarios

