4-2013

Artificial Immune Systems and Particle Swarm Optimization for Solutions to the General Adversarial Agents Problem

Jeremy Mange
Western Michigan University, jeremy.mange@gmail.com

ARTIFICIAL IMMUNE SYSTEMS AND PARTICLE SWARM OPTIMIZATION FOR SOLUTIONS TO THE GENERAL ADVERSARIAL AGENTS PROBLEM

by

Jeremy Mange

A dissertation submitted to the Graduate College
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
Computer Science
Western Michigan University
April 2013

Doctoral Committee:

Dionysios Kountanis, Ph.D., Chair
Ala Al-Fuqaha, Ph.D.
Matthew Castanier, Ph.D.
Wuwei Shen, Ph.D.

ARTIFICIAL IMMUNE SYSTEMS AND PARTICLE SWARM OPTIMIZATION FOR SOLUTIONS TO THE GENERAL ADVERSARIAL AGENTS PROBLEM

Jeremy Mange, Ph.D.

Western Michigan University, 2013

The general adversarial agents problem is an abstract problem description touching on the fields of Artificial Intelligence, machine learning, decision theory, and game theory. The goal of the problem is, given one or more mobile agents, each identified as either "friendly" or "enemy", along with a specified environment state, to choose an action or series of actions from all possible valid choices for the next "timestep" or series thereof, in order to lead toward a specified outcome or set of outcomes. This dissertation explores approaches to this problem utilizing Artificial Immune Systems, Particle Swarm Optimization, and hybrid approaches, along with related theoretical and analytic issues. A non-linear integer programming formulation of the problem is provided, several novel approaches are explored and compared, and two original algorithms are presented and demonstrated to be more useful than established algorithms for certain classes of problems. As part of the research effort, a software system to solve instances of the general problem is presented, centered on a novel hybrid Artificial Immune Systems / Particle Swarm Optimization algorithm.

ACKNOWLEDGMENTS

The people I most would like to thank are the members of my committee – Dr. Ala Al-Fuqaha, Dr. Matt Castanier, Dr. Wuwei Shen, and particularly my advisor, Dr. Dionysios Kountanis. These men offered help and direction, from the proposal stage of this research through the revisions of the fully written dissertation. I am greatly indebted to them for their support, ideas, and corrections, and I could not have successfully completed this research without their help.

In addition, I would like to thank the administrators of the Science, Mathematics, and Research for Transformation (SMART) scholarship program, who provided funding and mentorship for the last several years of my graduate schooling.

Lastly, I would like to thank my parents, who have provided love and support throughout my education, and indeed throughout my entire life. My gratitude to them cannot be overstated.

Jeremy Mange

Contents

List of Tables

List of Figures

1 Introduction

1.1 Problem Description

The general adversarial agents problem is an abstract problem description touching on the fields of Artificial Intelligence, machine learning, decision theory, and game theory. The problem is that of, given one or more mobile agents, each identified as either "friendly" or "enemy", along with a specified environment state, to choose an action or series of actions from all possible valid choices for the next "timestep" or series thereof, in order to lead toward a specified goal. Combat environments, military planning situations (such as partially autonomous drones), as well as most traditional adversarial games within the game-theory domain can all be expressed in this manner, as can a wide variety of other strategic problems.

The concept of an arbitrarily small "timestep" was introduced to effectively eliminate the distinction between turn-based games or environments (those in which players alternate in selecting actions) and real-time games or environments (those in which any player may select an action at any time). By alternating action selection at an arbitrarily small interval, the general adversarial agents problem can capture both traditional adversarial games like chess and more complex real-world scenarios such as war simulations.

The important components of a general adversarial agents problem instance, then, are descriptions of the agents and of the environment. In particular, this includes a definition of the context space (equivalent to the game-theoretic concept of a game board), the mobility of each agent within that context space, the capability of each agent against all adversarial agent types, and a definition of the methods and results of encounters between opposing

agents. Together, these descriptions define the problem and how the environment progresses over time. In addition, a definition of the goal is necessary for action selection and planning, which may introduce other concepts such as a "value" for each agent.

Since each of these components of the problem can be expressed mathematically, it is possible to formulate the problem as an optimization problem, where the goal is expressed as a function either to be minimized or maximized. In this way, if a problem instance is constructed properly, a general-purpose optimizing solver can be used to provide an optimal action selection for the specified goal. However, it is infeasible to use this type of approach for anything beyond very small problem instances. For general instances, some sort of heuristic algorithm is necessary.

Within this dissertation, the problem will be formally defined, each of these claims will be proven, a novel heuristic algorithm will be developed combining elements of Artificial Immune Systems and Particle Swarm Optimization, and a complete software system to solve general instances of this problem will be described, with verification and examples.

1.2 Original Contributions

A summary of the major original contributions of this dissertation is as follows:

- Well-defined non-linear integer programming formulation of the general adversarial agents problem
- Development of the use of mathematical solvers to automatically generate strategy solutions for the general adversarial agents problem
- Proof that the general adversarial agents problem is NP-Hard

- Original enhanced Artificial Immune Systems Negative Selection algorithm, with experimental results and demonstration of improvement
- Development of the use of Particle Swarm Optimization methods to automatically generate strategy solutions for the general adversarial agents problem
- Original hybrid Artificial Immune Systems / Particle Swarm Optimization algorithm, with experimental results and demonstration of improvement
- Complete modular software system to solve arbitrary instances of the general adversarial agents problem

1.3 Organization of Dissertation

The dissertation is organized by chapters, with each chapter representing a major research component. The chapter organization is as follows:

1.3.1 Literature Review

In this chapter, literature relevant to this research effort are reviewed, in order to provide a context for the current research and its novel contributions. Research papers and books are summarized that touch on the adversarial agents problem in general or specific instances thereof, Artificial Immune Systems and enhancements, Particle Swarm Optimization and modern variants, and previous attempts to hybridize AIS and PSO algorithms.

1.3.2 Non-Linear Integer Programming Formulation

In this chapter, the general adversarial agents problem is formally defined using a non-linear integer programming formulation. Each portion of this formulation is developed mathematically, and the use of mathematical solvers for instances of the problem is discussed, with examples to illustrate their use. The limitations of these solvers is demonstrated, and the necessity of a heuristic algorithm for general instances of the problem is demonstrated.

1.3.3 Generalized Assignment Sub-Problem

In this chapter, the Generalized Assignment Problem is addressed. It is first proved that this problem is NP-Hard, then it is proved that this problem is a special case of the general adversarial agents problem. Thus is it demonstrated that a system to provide exact solutions to the general adversarial agents problem is computationally infeasible, providing theoretical support to the need for a heuristic algorithm.

1.3.4 Artificial Immune Systems Approach and Enhancement

In this chapter, Artificial Immune Systems, a class of algorithms from which parts of the heuristic algorithm will be drawn, are defined. A novel enhancement of the Negative Selection AIS method is presented, with experimental verification. A direct approach to the adversarial agents problem using AIS methods is demonstrated.

1.3.5 Particle Swarm Optimization and Hybrid Algorithm

In this chapter, Particle Swarm Optimization, the second method from which parts of the heuristic algorithm will be drawn, is defined. A direct approach to the adversarial agents problem using PSO is first given. Finally, the novel heuristic algorithm which is a hybrid of elements of AIS and PSO with some other enhancements, is presented. This new algorithm is tested against several benchmark functions in terms of both accuracy and efficiency, and is shown to outperform the standard PSO algorithm for problems of the type examined in this research.

1.3.6 Complete Software System

Finally, in this chapter, a complete software system incorporating the heuristic hybrid algorithm is shown. The modules of this system and their interactions are defined, extensions of the software framework to a variety of problem contexts are demonstrated, and examples from widely separated application areas are presented with results.

2 Literature Review

This chapter provides an overview of current research related to the adversarial agents problem and the approaches used for the final system of this dissertation research. Further details and references are provided where appropriate within later chapters, but this chapter serves to provide an overview and to establish the context and original contributions of this dissertation research.

2.1 Adversarial Agents Problem

Due to the generality of the adversarial agents problem, much research has been done that is in some way related to algorithms for the problem. Most of the proposed and implemented approaches have taken the form of heuristic algorithms, because of the complexity of the problem and the need for real-time systems in many of the contexts in which it appears. Heuristic algorithms are those which are not guaranteed to find optimal solutions, but aim to find "good enough" solutions in a more reasonable time frame than would be required to find absolutely optimal ones. In the case of the general adversarial agents problem, this equates to discovering strategies and action selections that tend to lead to desired outcomes, even if more optimal outcomes might be possible with different actions.

A number of approaches to limited versions of the adversarial agents problem have been successful within specific contexts. Some of the most notable of these are equilibrium-based strategy models ([Manshaei et al., 2011], [Sailer et al., 2007]) which calculate a balance of action choices based on the estimated likelihood of enemy actions, probability-based simulation ([Auer et al., 1995], [Wang and Gelly, 2007]) which builds strategy through repeated semi-random

simulations with learned weights, and evolutionary learning techniques ([Dalvi et al., 2004], [Villacorta and Pelta, 2010]), a field which encompasses a variety of genetic and biologically-inspired techniques in order to perform learning for goal-based action selection. These approaches are explored in more detail in the next sections.

While many of these techniques have been successful in finding solutions to instances of the general adversarial agents problem within specific contexts, there are few objective, external methods for verifying and evaluating these sorts of heuristic algorithms. A major goal of the this research is to provide such a method. As will be shown, the use of a non-linear integer programming solver to find solutions to instances of the problem can provide a benchmark against which to measure the accuracy and speed of a heuristic algorithm designed for the same purpose, both of which are presented in detail in later chapters.

Additionally, although some of the methods presented in this chapter have demonstrated success in specific domains, that is, for specific subsets of the general adversarial agents problem, all of them are either completely inapplicable or highly inefficient for other instances within this general problem domain. Another major contribution of this dissertation research is to provide a method that will be shown to be able to efficiently solve *any* instance of the general adversarial agents problem, and thus defines a framework that can be applied across a wide variety of problem domains.

2.1.1 Equilibrium-Based Strategy Algorithms

One successful approach to certain subsets of the general adversarial agents problem has been through the use of Nash equilibrium analysis. This type of

analysis takes into account the strategic interaction of several players in a game theory context, rather than focusing on strategy generation for a single player. This is useful both for problems that can be modeled as cooperative games, such as network traffic routing (see [Manshaei et al., 2011] for an example of such), and problems that can be modeled as adversarial games, such as those closer to the context of this research (see [Sailer et al., 2007] for an example of such).

A Nash equilibrium in a game theory context is a set of strategies selected by all players in a game such that no change in a strategy by any player yields a better outcome if all other players keep their strategies fixed.

More formally, given a game with n players, and a set of strategies:

$$S = S_1 \times S_2 \times ... \times S_n$$

where S_i is the strategy set for player i, and a set of payoff functions:

$$F = \bigcup f; f(x) = [f_1(x), f_2(x), ..., f_n(x)]$$

where $f(x)$ is the payoff function for $x \in S$; let x_i be a strategy profile for player i, and x_{-i} by the strategy profile of all players except player i.

Then a strategy profile x is a Nash equilibrium if:

$$\forall i, y_i \in S_i : f_i(x_i, x_{-i}) \geq f_i(y_i, x_{-i})$$

That is, no change in strategy by any player is of higher value for that player.

Analyses of problems in game theory using the Nash equilibrium have yielded useful results in several different contexts, as cited above. For coop-

erative games, this type of analysis can yield optimum cooperative strategies for all players. For adversarial games, this type of analysis can yield the frequency with which a player should choose among strategic options. A classic and common simple example of this (see, for example, [Sailer et al., 2007]) is the game Rock-Paper-Scissors. The payoff matrix (defining the values for F above) for that game is shown in Table 4, with the rows representing the choices of one player, the columns representing the choices of the other.

	R	**P**	**S**
R	0	-1	1
P	1	0	-1
S	-1	1	0

Table 1: Rock-Paper-Scissors payoff matrix

Nash equilibrium analysis of this table reveals that each strategy option (rock, paper, or scissors) should be selected with 33.3...% frequency. This same analysis and approach can be applied to more complex instances of the adversarial agents problem. [Sailer et al., 2007] provide an application of this strategy to a Real-Time Strategy computer game, for computer player strategy selection. They demonstrate that this approach is a promising one even for these large, complex instances of the adversarial agents problem.

2.1.2 Probability-Based Strategy Algorithms

One of the most important concepts for automated strategic selection in a game theory context is that of the trade-off between exploration and exploitation, that is, how much computational time to spend evaluating new strategy options, and how much to spend estimating and refining the value (or payoff) of already explored strategy options. Probability-based strategy algorithms

focus directly on this trade-off, and several algorithms have been proposed and evaluated that make automated strategy selections for particular types of game theory problems using this approach.

These algorithms are most commonly presented in the context of the multi-armed bandit problem ([Robbins, 1952]). This is the problem of a gambler, with a row of slot machines, who must decide in what order and with what frequency to play each machine in order to maximize the payout. Since the properties of each machine are initially unknown, this problem illustrates the trade-off between exploration and exploitation, as clearly the successful strategy is to play the machine with the highest average payout most often, but estimates of what the average payout of each machine is become more accurate only with repeated trials.

Many variations of the multi-armed bandit problem have been developed, and algorithms proposed to solve instances of the varied problems. Perhaps most directly related to this research, [Auer et al., 1995] provide an adversarial version of the problem in which the payouts of the machines are controlled by an adversary. They define an algorithm which they show to be optimal for this scenario that uses a probability-based strategy approach.

[Wang and Gelly, 2007] go one step further by explicitly applying a multi-armed bandit algorithm developed by the same authors from the previous paragraph, UCB1 ([Auer et al., 2002]; details below), to an adversarial agents game theory context, the game of Go. This application uses the exploration / exploitation balancing formula of UCB1 to construct a dynamic Go game tree, using Monte-Carlo simulations for estimates of the value of move choices within this tree. The details of this approach are very interesting, and the approach was incredibly successful. The author of this dissertation was working in the

area of automated strategy generation for the game of Go at the time this paper was published, and witnessed what a revolution the method caused in AI approaches to this notoriously difficult game. Although this approach is not directly applicable to most general instances of the adversarial agents problem, the general concepts of the exploration / exploitation tradeoff and the use of Monte-Carlo simulations for estimates of strategic choice values are used within the final system presented in later chapters. For that reason, some details of this algorithm are pertinent here.

The UCB1 algorithm ([Auer et al., 2002]) aims to ensure that the multi-armed bandit problem machine with the highest payout is played exponentially more often than any other machine, regardless of the time necessary to converge on learning the actual machine payouts. This is accomplished by first playing each machine once, then playing the machine j that maximizes the formula:

$$X_j + \sqrt{\frac{2logn}{T_j(n)}}$$

where n is the overall number of plays so far, and $T_j(n)$ is the number of times machine j has been played after the first n overall plays. This formula implicitly balances exploration and exploitation in a manner that has proved remarkably effective in a variety of contexts (see [Radlinski et al., 2008], [Pandey et al., 2007], [Kleinberg et al., 2008], [Da Cost et al., 2008]). A modification of this algorithm that appears within [Auer et al., 2002] and [Wang and Gelly, 2007] is as follows:

$$X_j + \sqrt{\frac{log(n)}{T_j(n)} min\left[\frac{1}{4}, V_j[T_j(n)]\right]}$$

This modification proved more effective in practice, according to both. The UCT algorithm, then, is simply the UCB1 algorithm with the additional use of a tree structure to dynamically track which sub-moves – that is, moves one or levels below the move under consideration in the game tree – have already been explored.

2.2 Artificial Immune Systems

Artificial Immune Systems (AISs) form a relatively new class of artificial intelligence algorithms modeled on some characteristics of the biological immune system. These algorithms are most commonly applied to classification problems, but also have been applied to optimization problems, as well as problems in a handful of other contexts.

AISs were first introduced in 1986 in [Farmer et al., 1986]. This paper presented a computer model of aspects of the biological immune system such as lymphocytes, antibodies, and antigens, and presented a classifier algorithm based on these aspects. It also included the definition of genetic operators to update and improve the antibodies involved in the system, which resemble the later Clonal Selection AIS method, which will be explored in detail in a later chapter. This paper included extensive discussion of the biological immune system which, although it formed the initial inspiration for the class of algorithms, would come to be less specifically emphasized over time.

Some initial research was done in this area shortly after the introduction of AISs, and in 1994, the Negative Selection (sometimes now referred to as "Negative Detection") algorithm was introduced in [Forrest et al., 1994]. This is often presented as the "classic" AIS algorithm, and is one of the most

straightforward. As might be guessed from the paper's title, Negative Selection focuses on the discrimination between two classes in a classification problem, one corresponding to "self", or healthy cells within the body, and the other corresponding to "non-self" or potentially dangerous cells, to which the immune system would respond; thus the algorithm is most easily applied to binary classification tasks. The details of the method by which this is carried out are presented in a later chapter. Although in [Forrest et al., 1994] the detectors involved were described as strings with a matching measure, it is commonly to more generally describe the detectors as objects within a search space, with a distance measure for points and other objects.

Although this algorithm was promising and eventually would be adopted in a variety of contexts, the generation of the detectors in the Negative Selection algorithm as described in [Forrest et al., 1994] was inefficient. [Dhaeseleer et al., 1996] introduced a more efficient and scalable generation strategy. An extensive evaluation and analysis of the Negative Selection algorithm and detector generation portions in particular was provided by [Wierzchon, 1999]. It was shown that the use of exclusively spheres for detectors within the search space was a severe limitation, and [Hofmeyr and Forrest, 2000] described a new strategy to allow the detectors to take on more arbitrary shapes.

Shortly thereafter, Clonal Selection was introduced in [De Castro and Timmis, 2002] and [Ayara et al., 2002], first as part of a Negative Selection algorithm, and later as a separate approach in its own right. Clonal Selection has much in common with Genetic Algorithms, and allows for the improvement of detectors (either within or apart from a Negative Selection system) through genetic operators.

Negative Selection and Clonal Selection are the two most common AIS

methods, and the two that will be most extensively explored and utilized in this research. "Danger theory" is a newer area of study within AIS (see, for example, [Aickelin and Cayzer, 2002] and [Williamson, 2002]) that has much in common with Negative Selection, but with some modifications.

Artificial Immune Systems have been applied to a variety of computational tasks, often with promising results. [de Castro and Von Zuben, 1999] provides an overview of many of these applications and successes. [Garrett, 2005] provides an excellent evaluation of the different AIS methods, and proposes a way of measuring their overall usefulness. He concludes that Negative Selection and Clonal Selection are both useful AIS methods, and while others such as Danger Theory may ultimately prove useful, they are less so at the moment.

2.3 Particle Swarm Optimization

Like Artificial Immune Systems, Particle Swarm Optimization (PSO) is a fairly recent biologically-inspired machine learning method. PSO was originally inspired by swarming behavior of groups of animals such as fish or birds. PSO is a pure optimization method, and as such is most naturally suited to unconstrained numerical optimization problems.

PSO was first introduced in 1995 in [Kennedy and Eberhart, 1995]. In this paper, the authors extended a simulation of a flock of birds searching for a cornfield, and applied a new, similar algorithm to numerical optimization problems. They extended the algorithm to an arbitrary number of dimensions and introduced the key concepts for PSO algorithms. In this algorithm, a group of particles are randomly initialized within a search space, and then move about the search space according to formulas that take into account the

best-known-position of the particle itself and the swarm as a whole. There is also an element of randomness introduced into the movement, to increase the exploration portion of the exploration / exploitation trade-off previously discussed, as well as to help avoid the problem of getting stuck at local minima or maxima. Interestingly, the PSO algorithm was originally introduced to optimize weights within a neural network, but it would quickly be used as a pure optimization algorithm in its own right, in many ways a competitor to the neural network method.

This algorithm met with some immediate popularity, and much more so once an improved version was introduced in [Shi and Eberhart, 1998]. It has several advantages – it is both conceptually and computationally simple to implement, as opposed to, for example, neural networks, and with good parameter selection, it performs very well at optimization tasks as compared to other general-purpose optimization algorithms. A major drawback, however, is PSO's sensitivity to parameters, and the difficulty in assigning those parameters without a priori knowledge of the problem domain.

The original PSO algorithm was described with two weight parameters. In 1998 in [Shi and Eberhart, 1998], a third "inertia" parameter was introduced, and the modified algorithm shown to perform better with good parameter selection than the original algorithm. This three-parameter version of PSO has proven immensely popular and is widely used in largely unchanged form today.

[Eberhart and Shi, 2001] provide an excellent overview of some developments in the PSO algorithm up to that point, and a wide variety of applications in which PSO is used. These include "fuzzy controller design, job shop scheduling, real time robot path planning, image segmentation, EEG

signal simulation, speaker verification, time-frequency analysis, modeling of the spread of antibiotic resistance, burn diagnosing, gesture recognition, and automatic target detection" ([Eberhart and Shi, 2001]).

2.4 Hybrid AIS / PSO Algorithms

A number of algorithms have been proposed that are hybrids of PSOs and Genetic Algorithms (GAs) (e.g. [Pedersen, 2010], [Robinson et al., 2002], [Premalatha and Natarajan, 2009]). Those algorithms that have incorporated elements of AISs and PSOs have almost exclusively focused on the Clonal Selection AIS method ([Yap et al., 2011], [Afshinmanesh et al., 2005], [Wang, 2006]), which is very conceptually similar to the GA algorithm, and thus have yielded much the same results as hybrid GA and PSO algorithms. These hybrid methods generally have the form of first using a genetic approach to derive an initial placement of members of a population within a search space, and then using these members as the particles within a PSO.

Yap, et. al. [Yap et al., 2011] present a fairly typical example of such a hybrid construction. In their algorithm, a standard PSO is run, and the positions in search space of the best PSO particles are used to define half of the initial population for the clonal selection AIS method, with the other half of the population random. Clonal Selection is then performed, which follows the procedure of a GA without the crossover operation. This process continues until a stopping condition is reached. A complete description of the Clonal Selection AIS method and its applicability to the system built as part of the current research is provided in later chapters.

This is a similar approach to all such efforts to combine PSO with the

Clonal Selection AIS method. In some cases the Clonal Selection is performed before the PSO step, in which case, as previously stated, the Clonal Selection step is used to generate candidate solutions which are used as the initial placement of the particles within a PSO. Under both approaches the results are similar, and again much the same as hybrid PSO-GA algorithms.

No research seems to have been published attempting to construct a hybrid algorithm using any other AIS method with the PSO algorithm. In a later chapter, an original hybrid algorithm will be presented which incorporates elements of the Negative Selection AIS method and a normal three-parameter PSO algorithm. This algorithm will be shown to out-perform the standard PSO algorithm for certain types of optimization tasks, and will form a critical part of the final system build as part of this research.

3 Non-Linear Integer Programming Formulation

3.1 Introduction

In this chapter, the general adversarial agents problem is examined from a non-linear integer programming point of view. Non-linear integer programming is first defined and placed within the context of other mathematical programming approaches. Then solvers for this type of problem are introduced and the methods used by these solvers are briefly discussed. The bulk of the chapter, then, is devoted to a formal non-linear integer programming formulation of the general adversarial agents problem. The general form of the problem is defined in terms of constants, variables, initial values, constraints, and an objective function, with discussion of how each of these elements define instances of the problem.

Once this formulation has been presented, examples are provided of specific instances of the general adversarial agents problem corresponding to this formulation, with the results of a mathematical solver used for the instances shown. Although a formal definition of the problem is of value in defining the exact parameters of what is being studied, these examples further demonstrate the usefulness of this formulation for solving actual instances. Finally, the limitations of this approach and its use within the larger context of this research and the final software system are explored.

The non-linear integer programming formulation of the general adversarial agents problem, as well as the use of mathematical solvers to automatically generate strategy solutions for instances of this problem, are the major original contributions of this chapter.

3.2 Non-Linear Integer Programming

Non-linear integer programming is a type of mathematical programming problem which involves optimizing an "objective function" subject to a set of equality and/or inequality constraints. These problems are expressed by the definition of a series of variables and a series of constraints on those variables, along with an objective function to be maximized or minimized. Linear programming problems are those in which all the constraints are linear, that is, all constraints can be expressed in the form $Ax \leq b$. For non-linear programming problems, the constraints can be more general. Integer programming problems are those in which all the variables are integers, whereas "mixed" or "mixed integer" programming problems can include variables of other types. The formulation in this section is a non-linear integer formulation.

One of the more common ways to express mathematical programming problems is through the use of AMPL, a computer language specifically designed for such problems. The following sections will use AMPL-like notation to define the constants, variables, and constraints, along with explanations of each.

For simplicity and for the sake of readable examples, the formulation will be presented in terms of two dimensions; however, extensions to an arbitrary number of dimensions are straightforward. Additionally, only the constants, variables, and constraints for the friendly agents (indicated by a superscript "f") will be shown; in problem instances the same are be defined for enemy agents as well.

3.2.1 Non-Linear Integer Solvers

A great deal of research has been done in the area of mathematical programming, and at this point, a variety of both general-purpose and special-purpose solvers exist for linear and non-linear integer programming problems. Linear programming dates back to World War II, and the famous simplex algorithm for linear programming problems was developed in 1947, with many extensions and further methods developed since. Linear programming has been extensively studied in part because of its applicability to a wide variety of problems within the field of operations research. Many operations research problems can be formulated using linear constraints and an objective function, and thus lend themselves easily to attack through automated solvers.

For linear integer programming problems, that is, problems where all constraints are of the form $Ax \leq b$, one can construct a polytope within the dimensions defined by the variables which represents the feasible region for the problem, or the set of all feasible solutions. It is straightforward to demonstrate that the extreme values for any objective function involving the same variables will lie along the border of this polytope. The simplex algorithm for linear programming problems, then, exploits this fact by starting at a vertex of this polytope and examining values for the objective function at other vertices until a maximum or minimum is reached. The simplex algorithm formed the basis of automated mathematical programming solvers, and variants of it are still used today.

This algorithm performs well in the average case, but has exponential worst-case behavior. In fact, although general linear programming (where all variables are real numbers rather than integers) has a polynomial-time algo-

rithm, integer linear programming is an NP-complete problem. Since integer non-linear programming is a generalization of integer linear programming, it too is an NP-hard problem. The full implications of this will be discussed in the following sections of this chapter, but it is immediately apparent that this approach cannot be used exclusively for general instances of the adversarial agent problem.

All general-purpose solvers, then, must either find global optimal values but have exponential worst-case performance, or use some sort of approximation strategy. Both of these approaches are common in various software packages for both linear and non-linear integer programming solvers. For the non-linear case of a minimization problem (which is how the problem is formulated in this chapter), many solvers, including the one used for instances of this formulation, utilize an approach known as convex optimization, which is more fully explained in the next section. If the objective function and each of the constraints is convex, this approach can be used directly. If not, the program can be divided into convex subsets, which effectively form lower bounds for the cost of that subset. The combination of these costs will eventually lead to a global solution, although often time constraints are put in place to prevent the solver from running infinitely. Once the solver is stopped, an estimate can be calculated of the likelihood that the current optimum is the global optimum, based on the division of the subsets.

3.2.2 Convex Optimization

As stated, convex optimization is a branch of optimization in which both the constraints and the objective function are convex, and the task is to minimize

the objective function; that is, to find $x \in X$ such that

$$f(x) = min\{f(x') : x' \in X\}$$

where X is the vector space for the problem. The important exploitable fact about a convex optimization problem is that any local minimum is also a global minimum, because of the convexity of the constraints and objective function. Because of this fact, solvers can utilize Lagrange multipliers for these problems. Lagrange multipliers are new variables that can be introduced to a optimization problem in order to more quickly find local minima (which for convex optimization problems, translate to global minima).

To define this approach, assume that all the constraints for a convex optimization problem are expressed in "standard form", that is, they are all of the form:

$$g_i(x) \leq 0$$

with an objective function:

$$f(x) : \mathbb{R}^n \to \mathbb{R}$$

to be minimized over x. The Lagrange multipliers, then, are a set of real numbers $\lambda_0, ..., \lambda_n$ which satisfy the following conditions:

(1) $\lambda_i \geq 0, \forall i$

(2) $\lambda_i g_i(x) = 0, \forall i$

These multipliers are introduced to form the associated Lagrangian func-

tion for the optimization problem, which is defined as:

$$L(x, \lambda_0, ..., \lambda_m) = \lambda_0 f(x) + \lambda_1 g_1(x) + ... + \lambda_n g_n(x)$$

For any point x that minimizes f, x also minimizes $L(y, \lambda_0, ...\lambda_n)$ for all $y \in X$. Fuller mathematical details and proofs can be found in [Bertsekas, 1999]. Of particular importance to the context of this research is the fact that the associated Lagrangian function can be efficiently solved, and thus automatic solvers exist that exploit this method to provide general-purpose non-linear integer programming solutions. The non-linear integer formulation of the general adversarial agents problems, then, is given in the following sections.

3.3 Constants (or Parameters)

In each of the following sections, the constants, variables, and constraints are expressed in the notation of the AMPL language, which was used to express them within the solvers used; further details on this language and solvers will be presented in a later section. This notation is in some ways similar to Python notation, and can be converted to more familiar mathematical function notation by considering each element with the brackets as an element of the cross product of the domain of the function. So, for example, the constraint $C^f\{1..F, 1..E\}$ is equivalent to the function $C^f : 1..F \times 1..E \to \mathbb{R}$. The default range of each is the set of real numbers, $\mathbb{R}$. It is noted in the description when the range (or "type", in AMPL terms) differs from this default.

The constraints, in the AMPL-style notation with brief descriptions, are as follows.

F: the number of friendly agents

T: the number of timesteps within the context time (or maximum game length, in game-theoretic terms)

D^x, D^y: the dimensions of the context space

$V^f\{1..F\}$: the value of each friendly agent

$C^f\{1..F, 1..E\}$: the capability of each friendly agent against each enemy agent (so agent i has capability $C^f[i, j]$ against enemy agent j)

$S^f\{1..F\}$: the maximum speed of each friendly agent (that is, the distance each agent can travel in one timestep)

$R\{1..D^x, 1..D^y, 1..D^x, 1..D^y\}$: defines the "reachability" of locations within the context space (so $R[a, b, c, d] = 1$ indicates that location (c, d) is reachable from location (a, b))

3.4 Variables

$X^f\{1..F, 0..T\}$: the x-coordinate for each friendly agent at each timestep

$Y^f\{1..F, 0..T\}$: the y-coordinate for each friendly agent at each timestep

$Lx^f\{1..F, 1..D^x, 0..T\}$: Boolean value indicating whether each friendly agent

is in the x-coordinate of a location at a timestep (so $Lx^f[i,x,t] = 1$ indicates that friendly agent i is in a location with an x-coordinate of x at timestep t – this is a helper variable in addition to X^f and Y^f)

$Ly^f\{1..F, 1..D^y, 0..T\}$: Boolean value indicating whether each friendly agent is in the y-coordinate of a location at a timestep

$H^f\{1..F, 0..T\}$: the health of each friendly agent at each timestep

$A^f\{1..F, 0..T\}$: Boolean value indicating whether each friendly agent is alive at each timestep (this is a helper variable in addition to H^f)

3.5 Initial Values

$X^f[i,0], Y^f[i,0]$: initial position of each friendly agent

$H^f[i,0]$: initial health of each friendly agent

3.6 Constraints

To ensure that each agent is at exactly one location at each timestep within the context space:

$$\sum_{x=1}^{D^x} Lx^f[i,x,t] = 1 \quad (\forall 1 \leq i \leq F, 0 \leq t \leq T)$$

$$\sum_{y=1}^{D^y} Ly^f[i,y,t] = 1 \quad (\forall 1 \leq i \leq F, 0 \leq t \leq T)$$

To ensure that Lx^f is consistent with X^f and Ly^f is consistent with Y^f:

$$X^f[i,t] = \sum_{x=1}^{D^x} x \cdot Lx^f[i,x,t] \quad (\forall 1 \leq i \leq F, 0 \leq t \leq T)$$

$$Y^f[i,t] = \sum_{y=1}^{D^y} y \cdot Ly^f[i,y,t] \quad (\forall 1 \leq i \leq F, 0 \leq t \leq T)$$

To ensure that each agent moves according to the reachability rules defined by R:

$$\sum_{x_1=1}^{D^x} \sum_{y_1=1}^{D^y} Lx^f[i,x,t] \sum_{x_2=1}^{D^x} \sum_{y_2=1}^{D^y} Lx^f[i,x_1,t] \cdot Ly^f[i,y_1,t]$$

$$\cdot Lx^f[i,x_2,t+1] \cdot Ly^f[i,y_2,t+1] \cdot R[x_1,y_1,x_2,y_2] = 1$$

$$(\forall 1 \leq i \leq F, 0 \leq t \leq T-1)$$

To ensure that each agent moves only as fast as it maximum speed allows, and does not move if it is destroyed:

$$\sqrt{\chi^2 + \delta^2} \leq S^f[i] \cdot A^f[i,t]$$

where

$$\chi = X^f[i,t] - X^f[i,t+1],$$

$$\delta = Y^f[i,t] - Y^f[i,t+1]$$

$$(\forall 1 \leq i \leq F, 0 \leq t \leq T-1)$$

To ensure that the health of each agent remains constant except in the case

of "combat", and to define the updated health of each agent in the case of combat (note: the superscript ^{e} indicates a variable for an "enemy"):

$$H^f[i,t+1] = H^f[i,t] - \sum_{x=1}^{D^x}\sum_{y=1}^{D^y} Lx^f[i,x,t]$$

$$\cdot Ly^f[i,y,t] \cdot \left(\sum_{j=1}^{E} Lx^e[j,x,t] \cdot Ly^e[j,y,t] \cdot C^e[j,i]\right)$$

$$(\forall 1 \leq i \leq F, 0 \leq t \leq T-1)$$

Note that this constraint incorporates a straightforward definition of "combat" in which the health of each agent is decreased by the combined capabilities of all opposing agents in the same location. This constraint could be modified to incorporate other definitions of encounters between adversarial agents as appropriate to the context of the problem.

To ensure that A^f is consistent with H^f (that is, $(H^f \leq 0) \Rightarrow (A^f = 0)$, otherwise $A^f = 1$):

$$A^f[i,t] \geq 0, \quad A^f[i,t] \cdot H^f[i,t] \geq 0$$

$$(\forall 1 \leq i \leq F, 0 \leq t \leq T)$$

3.7 Objective Function

The choice of objective functions to minimize (or maximize) depends on the context of the problem. A common approach to assign a weight α to the value of destroying enemy agents (weighted by context-appropriate values defined by V), and a weight β to the value of preserving friendly agents (again weighted

by value). The objective function, then, would be to minimize:

$$\alpha \cdot \sum_{i=1}^{E} \left(V^e[i] \cdot A^e[i,T]\right) - \beta \cdot \sum_{i=1}^{F} \left(V^f[i] \cdot A^f[i,T]\right)$$

The summation of the value of each agent multiplied by the "alive" boolean variable (0 or 1) for that agent, provides a total value for all agents that have not been destroyed.

Both this objective function and some of the constraints (most notably the reachability constraints) are non-linear. It is possible to construct a linear programming formulation of the almost identical problem, but since non-linear integer programming solvers are readily available, there is no need to move away from this more general formulation.

3.8 Examples

As previously mentioned, a variety of general-purpose solvers exist utilizing various algorithms for non-linear integer programming problems. Using such solvers and the problem formulation from the previous section, it is possible to automatically generate a strategy that is optimal with respect to the objective function. This is accomplished by solving for optimized values of X^f and Y^f in future timesteps, which represent the choice of movement actions for each mobile agent.

In this section, the results are shown of using such solvers on two relatively small instances of the general adversarial agent problem, in order to demonstrate the validity of such an approach for automated strategy planning and action selection. For the demonstration problems, examples are shown that are

simple enough to be easily understood visually, yet require a complex enough solution to demonstrate the accuracy of the problem formulation and the validity of the approach. In both of these examples, the enemy agents remain stationary for the sake of simplicity; in a more real-world system simulation (explored in later chapters) the enemy agents would also be controlled either manually or by another strategy algorithm.

Figure 1 illustrates the first example problem, with the circles representing friendly agents with their associated capabilities, and the square representing an enemy agent. The combat-defining constraint ($H^f[i, t+1]$) was altered to provide a health advantage to survivors of adversarial encounters, such that in this instance, if the friendly agents attacked the enemy agents one at a time, both would be destroyed, whereas if they attacked together, both would survive. Thus an accurate solution would require a synchronized attack, which would demonstrate a level of strategic planning arising purely from the mathematical description of the problem along with a non-linear solver.

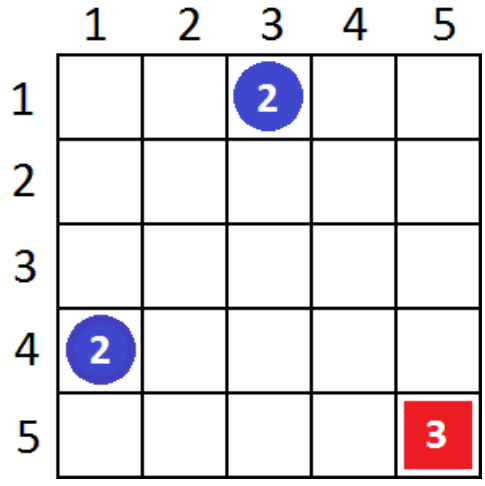

Figure 1: Example instance 1

The problem instance was defined using the previously described formula-

t	$X^f[1,t]$	$Y^f[1,t]$	$X^f[2,t]$	$Y^f[2,t]$
0	1	4	3	1
1	1	4	3	2
2	1	5	3	3
3	2	5	3	4
4	3	5	3	5
5	4	5	4	5
6	5	5	5	5

Table 2: Example instance 1 output

tion in the AMPL language and run on a general-purpose non-linear integer programming solver, with a value of 6 for T, reachability defined to describe a grid layout as shown, 1 for all S^f and V^f, and holding X^e and Y^e constant (representing an unmoving enemy agent). The results in terms of the optimized values for X^f and Y^f are displayed in table 2.

This table shows the X,Y coordinates of the friendly agents at each timestep of the problem instance. These results are shown visually in Figure 2. As can be seen, the two friendly agents do indeed demonstrate synchronized behavior for attacking the enemy agent, in order to minimize the objective function.

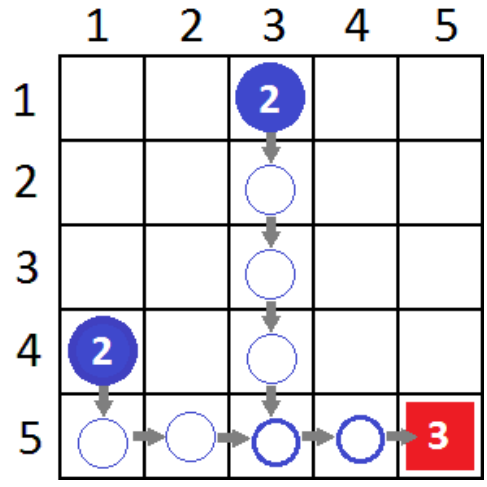

Figure 2: Example instance 1 solution

Figure 3 illustrates a second, slightly more complex example problem, again with the circles representing friendly agents and the squares representing enemy agents, and again with the modified combat ($H^f[i, t+1]$) rules. In addition, for this problem instance, the health constraint ($H^f[i, t+1]$) was further modified such that for non-combat timesteps, the health of each agent would regenerate. Thus, strategically for this problem, in order to survive and destroy all enemy agents, all three friendly agents would need to synchronize an attack on one enemy agent, regenerate their health, and then synchronize an attack on the second enemy agent. This more complex series of steps starts to reveal the power of using a non-linear integer programming solver to generate strategic and planning decisions.

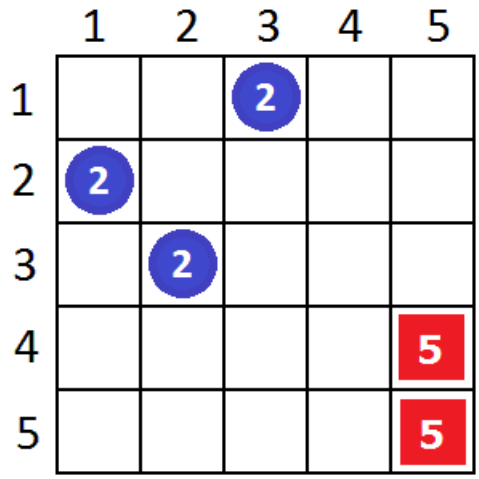

Figure 3: Example instance 2

This problem was solved using the same method as for the first problem. The results in terms of the optimized values for X^f and Y^f are displayed in table 3. Through inspection of the table, one can see that the three friendly agents did indeed synchronize an attack on the first enemy agent (at location $(5, 5)$), then remain in that location for three additional timesteps in order to

t	$X[1,t]$	$Y[1,t]$	$X[2,t]$	$Y[2,t]$	$X[3,t]$	$Y[3,t]$
0	1	2	2	3	3	1
1	1	3	2	4	3	1
2	1	4	2	5	3	2
3	1	5	2	5	3	3
4	2	5	2	5	3	4
5	3	5	3	5	3	5
6	4	5	4	5	4	5
7	5	5	5	5	5	5
8	5	5	5	5	5	5
9	5	5	5	5	5	5
10	5	4	5	4	5	4

Table 3: Example instance 2 output

regenerate sufficient health to successfully attack the second enemy agent.

Again, this table shows the X,Y coordinates of each of the friendly agents at each timestep of the problem instance. These same results are shown visually in Figure 4.

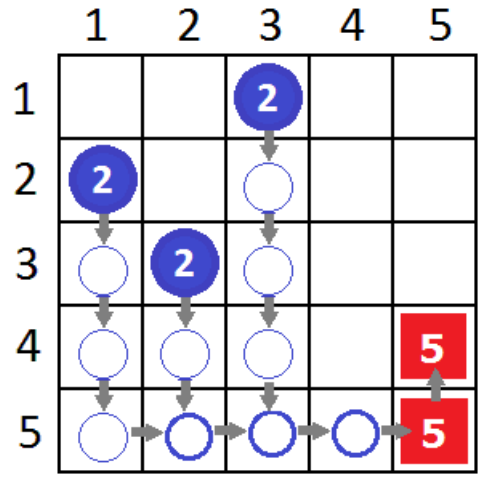

Figure 4: Example instance 2 solution

This example begins to illustrate the power of this approach both to model combat situations of varying levels of complexity and to produce meaningful

and accurate strategic decisions within adversarial system simulations. The authors ran solvers on a number of further problem instances requiring various types of strategic planning for an optimal solution, and in each case, given sufficient execution time, the results solved the instance in an optimal way. Together, the successful solving of such problem instances, including these examples, demonstrate the validity of the approach. The following section will expand on the possibilities of the approach and its applicability within the broader context of this research.

3.9 Applications and Extensions

A number of variations of this formulation are possible to apply to different problem domains. For instance, although the problem is formulated as a perfect-information game (one in which each side knows all information about the other), it could easily be modified to incorporate partial-information and confidence intervals for variables describing information about the adversary, as will be discussed in further chapters. Simulations of this type would more closely model, for example, real-world combat scenarios in which information about enemy forces is limited. As another extension example, the formulation above involves encounters between agents only when they occupy the same location within the context space. Another straightforward modification, then, would be to incorporate a variable to describe the "range" of each agent, and allow for adversarial encounters ("combat") within this range, again allowing for more accurate modeling of military forces of various types.

Due to the number of constraints necessary for even small problem instances along with the complexity of many of the algorithms involved in math-

ematical programming optimization solvers, the non-linear integer programming approach is not feasible for most real-time simulation systems for larger problem instances, and thus ultimately is not used directly within the final system presented as part of this research. The advantage of this process is the ability to find globally optimal solutions to problem instances. For this reason, this approach is used for validating and benchmarking heuristic-based strategy planning algorithms, which are far more common and more feasible for real-world modeling and simulation systems, and indeed will ultimately be used for the final strategy-generating system presented in later chapters. A proposed heuristic system, such as the AIS and PSO-based ones later described, is used to generate strategy and select actions for small problem instances, with the non-linear integer programming solver run on the same instances, and the solutions compared. This exact approach is used to test and validate the heuristic approaches defined in later chapters. Since the non-linear integer programming solver finds a globally optimal solution, and since the general adversarial agent problem can be used to express problems in such a wide variety of domains, this provides a valuable means of objectively evaluating the strength of new algorithms and approaches to this important and widely applicable problem.

4 NP-Hardness of the General Adversarial Agents Problem

4.1 Introduction

In this chapter, the Generalized Assignment problem will be examined. It will be shown that the Generalized Assignment problem is a sub-problem of the general adversarial agents problem by showing that, when formally defined, the former is a special case of the latter. It will then be proven that the Generalized Assignment problem is NP-Hard. These two facts, taken together, prove that the general adversarial agents problem is also NP-Hard. This motivates the heuristic algorithm for the problem developed in later chapters by showing that a heuristic or approximation algorithm is not only helpful, but in fact necessary for a computationally feasible approach to the general adversarial agents problem.

In addition to this main goal of the chapter, a new concept of reliability within the Generalized Assignment problem is introduced, and two new computation problems based on this concept are defined and explored: the Generalized Assignment with Reliability problem, and the Improvement of Reliability problem. Each of these new problems is formally defined and examined from a theoretical and computational perspective, and a flexible approximation algorithm is offered to provide approximate solutions to instances of the Generalized Assignment with Reliability problem.

The mainly theoretical aspects of this chapter, particularly the proof that the general adversarial agents problem is NP-Hard, provide a bridge between the formal definition of the problem and the limitations of the non-linear integer programming approach seen in the previous chapter, and the use of

heuristic algorithms to solve instances of the problem on which the following chapters focus.

4.2 Assignment Problem

The Assignment Problem is a basic problem in computer science and mathematics. There are several equivalent ways to view the assignment problem – one of the simplest to state and understand is the task of finding a maximum weight matching for a weighted bipartite graph. The problem can also be viewed in terms of a knapsack with items, or a set of agents and a set of tasks. Viewed from the latter perspective, any agent may perform any task, but there is a cost function associated with every pairing of agents to tasks. The goal of the problem under this perspective is to find a one-to-one assignment of agents to tasks such that the total cost is minimized (or, equivalently, maximized).

More formally, given sets A and T with equal cardinality, along with a cost function $C : A \times T \to \mathbb{R}$, find a bijective function $F : A \to T$ that minimizes:

$$\sum_{a \in A} C\left(a, f\left(a\right)\right)$$

The Assignment Problem is known to be in the **P** complexity class, and several polynomial-time algorithms exist that solve this problem (see, for example, [Kuhn, 1955]).

4.3 Generalized Assignment Problem

The Generalized Assignment Problem (sometimes abbreviated GAP), as the name might indicate, is a generalization of the Assignment Problem presented

in the previous section. Under the agents / tasks view of the Assignment Problem, the Generalized Assignment Problem adds a "budget" for each agent that defines a maximum total cost of tasks they can perform, and the restriction of a one-to-one matching is eliminated.

An equivalent perspective from which the problem is often introduced is that of bins and items. If there are a (not necessarily equal) number of bins and items, each bin has a maximum weight, and each bin-item pairing has an associated weight and profit, the problem is to assign each item to a bin such that the total profit is maximized.

More formally, that is, given n items $X = \{x_1, x_2, ..., x_n\}$ and m bins $B = \{b_1, b_2, ..., b_m\}$ with associated maximum capacities $c_1, c_2, ..., c_m$, and each bin-item pairing has an associated profit p_{ij} and weight w_{ij}, find a function $f : X \to B$ such that:

$$\sum_{i}^{m} w_{ix} < c_i, \forall x : f(x) = b_i$$

that maximizes:

$$\sum_{i}^{m} \sum_{j}^{n} w_{ij}$$

The Generalized Assignment Problem is NP-Hard. To demonstrate this, the famous Knapsack problem, which is well known to be NP-Complete, will be defined, and then reduced to the Generalized Assignment Problem. First, however, a modification of the Generalized Assignment Problem will be presented which both is of theoretical interest and has certain interesting real-world applications.

4.4 Generalized Assignment Problem with Reliability

In this section, a novel modification of the Generalized Assignment Problem is presented, along with motivation for the problem, analysis of the computational complexity, and potential approximation algorithms to approach the problem.

As a modification of the Generalized Assignment Problem, the concept of "reliability" is introduced, or equivalently, the probability of failure of any of the items in the previous presentation of the problem. For this modification, the analogy of items and bins breaks down somewhat. The motivation for this modification is, instead, another common application of the assignment problem, what is commonly referred to as the "weapon / target assignment problem". Using this analogy, there are a number of targets (commonly conceptualized as incoming enemy missiles) and a number of weapons with which to attack those targets, with an associated probability of destroying the target. The problem is to optimally assign the weapons to the targets in order to destroy the maximum number of targets.

An extension of this problem, closely related to the general adversarial agents problem, assigns a reliability to each weapon, or equivalently, a probability of failure for the weapon as a whole. This complicates the problem, since an optimal solution to the original problem might not be optimal with this new factor. The goal changes to finding the optimal assignment of weapons such that there is the highest probability of the targets being destroyed.

More formally, then, given n weapons $X = \{x_1, x_2, ..., x_n\}$, with associated probabilities of failure $F = \{f_1, f_2, ..., f_n\}$, and m targets $B = \{b_1, b_2, ..., b_m\}$ with associated values $c_1, c_2, ..., c_m$, with each weapon / target pairing having a

probability of destruction of p_{ij}, to find a function $f : X \rightarrow B$ that maximizes:

$$\sum_{i=1}^{m}(1 - f_x) \cdot p_{ix} \cdot c_i, \forall x : f(x) = b_i$$

It is trivial to note that this extended problem is NP-Hard. The original weapon / target assignment problem has been shown to be NP-Hard ([Ahuja et al., 2007]), and since this extension is a generalization of the original problem, it is also NP-Hard. For this reason, approximation or heuristic algorithms are necessary to efficiently approach instances of the problem.

An approximation proposed for the original weapon / target assignment problem that can be easily adapted to the extended problem involves branch-and-bound techniques. The problem is first formulated as an nonlinear integer programming problem, in much the same fashion as the translation of the general adversarial agents problem to the nonlinear integer programming domain given in the previous chapter. From there, a mathematical programming solver is used, using relaxation techniques common to approximation algorithms for NP-Hard problems. Although this extended problem is closely related to the general adversarial agents problem, there are some key differences, and for that reason a complete presentation and analysis of approximation algorithms for the problem is outside the scope of this research. A presentation of some such algorithms for the original weapon / target assignment problem is given in [Ahuja et al., 2007].

4.5 Knapsack Problem

The knapsack problem is, informally, to find the best packing within a knapsack of a subset of a set of items, each of which has an associated weight and

value. More formally, given n items, $x_1, x_2, ..., x_n$, each of which has a weight w_i and a value v_i, along with a maximum weight W, maximize:

$$\sum_{i=1}^{n} v_i x_i, x_i \in \{0, 1\}$$

subject to:

$$\sum_{i=1}^{n} w_i x_i \leq W$$

where $x_i = 0$ indicates that item i is not in the knapsack, and $x_i = 1$ indicating that it is. The Knapsack Problem is one of the most famous problems in Computer Science, and the decision version of the problem is commonly introduced as a fundamental NP-Complete problem ([Garey and Johnson, 1979]). The optimization version as presented in this section is, naturally, also NP-Hard.

4.6 Reduction of Knapsack Problem to Generalized Assignment Problem

In order to demonstrate that the Generalized Assignment Problem is NP-Hard, the Knapsack Problem will be reduced to it.

Given a generic instance of the Knapsack Problem as presented in the previous section, create an instance of the Generalized Assignment problem by setting $m = 1$, setting $w_{1i} = w_i, \forall i$, and setting $p_{1i} = v_i, \forall i$. This reduction is very straightforward because the Knapsack Problem is, in fact, a special case of the Generalized Assignment problem with only a single bin.

Thus, because the Knapsack Problem is NP-Hard, the Generalized Assignment Problem is also NP-Hard.

4.7 Improvement of Reliability Problem

Given the concept of reliability as previously defined, a natural transformation of the Generalized Assignment Problem with Reliability is to focus on the reliability itself. If the generic concept of "resources" is introduced, which increase the baseline reliability of the weapons in the problem (or, equivalently, decrease the probability of failure), a natural new problem is to determine the minimum number of resources necessary in order to achieve a desired reliability. This problem is referred to as the "Improvement of Reliability problem".

Formally, then, let a function be denoted as g which defines the decrease in the probability of failure of a weapon for a given number of resources. That is, the new probability of failure of a weapon with original probability f_i for an addition of r resources is: $f_i - g(r)$. The Improvement of Reliability problem is to find minimum values $r_1, r_2, ..., r_n$ such that:

$$\sum_{i=1}^{n} f_i - g(r_i) \leq (1 - R)$$

where R is the desired reliability of the overall system.

Alternatively, the problem could be defined as that of how to optimally re-assign the weapon / target assignment to improve reliability. Even given a perfectly ideal assignment for the Generalized Assignment Problem, this assignment will not necessarily produce the greatest overall system reliability, again defined as the average likelihood of target destruction across all targets, weighted by value, or:

$$\sum_{i=1}^{m} (1 - f_x) \cdot v_i, \forall x : f(x) = b_i$$

The following example will illustrate this principle, that increases of system reliability can result from re-assignment of weapon / target assignments.

4.7.1 Example

Suppose that, using the weapon / target analogy, there are 5 weapons $(x_1, x_2, ..., x_5)$ and 3 targets (b_1, b_2, b_3), and suppose that the profit matrix p_{ij} was defined according to Table 4:

	b_1	b_2	b_3
x_1	5	1	1
x_2	5	1	1
x_3	1	5	1
x_4	1	5	1
x_5	1	1	5

Table 4: Example p_{ij} matrix

Furthermore, suppose c_i and w_{ij} are defined in such a way that no assignment violates this constraint, and that the values associated with destroying each target is $v_1 = 5, v_2 = 1, v_3 = 1$. It is straightforward to see from inspection that according to the Generalized Assignment Problem, the ideal assignment would be to assign weapons 1 and 2 to target 1, weapons 3 and 4 to target 2, and weapon 5 to target 3, as shown in Figure 5.

However, if this problem is extended to include the concept of reliability, this changes. Now suppose the probability of failure of the weapons is defined as: $f_1 = .5, f_2 = .5, f_3 = .1, f_4 = .1, f_5 = .1$. The overall value of the assignment, then, is:

$$\sum_{i=1}^{m}(1-f_x)\cdot v_i = (1-(.5)(.5))\cdot 5+(1-(.1)(.1))\cdot 1+(1-.1)\cdot 1 = 3.75+.99+.9 = 5.64$$

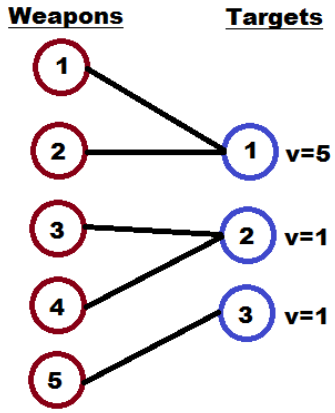

Figure 5: Example initial assignment

However, a simple change in the assignment would be to assign weapon 3 to target 1, as shown in figure 6. This assigns an additional weapon with much more reliability to the highest-valued target, target 1.

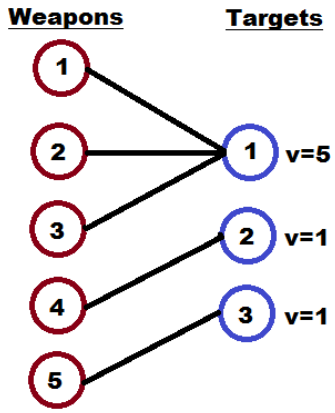

Figure 6: Modified assignment with higher reliability

The overall value of this modified assignment, then, is:

$$\sum_{i=1}^{m}(1-f_x)\cdot v_i = (1-(.5)(.5)(.1))\cdot 5+(1-(.1))\cdot 1+(1-.1)\cdot 1 = 4.875+.9+.9 = 6.675$$

This illustrates the fact that when overall system reliability is under consideration, a re-assignment from a previously ideal assignment based on the Generalized Assignment Problem can produce a higher overall reliability. To illustrate this effect further, a series of larger random assignment problem instances were created, with between 100 and 200 weapons, between 100 and 200 targets, with random values assigned to the targets. For each instance, the best assignment for the generalized assignment problem was first calculated. Then, with increasing limits for the number of allowed weapon-target reassignments, a new assignment was calculated to maximize reliability. The percentage improvement of those assignments were then recorded. Figure 7 shows the averaged results for 100 such randomized instances.

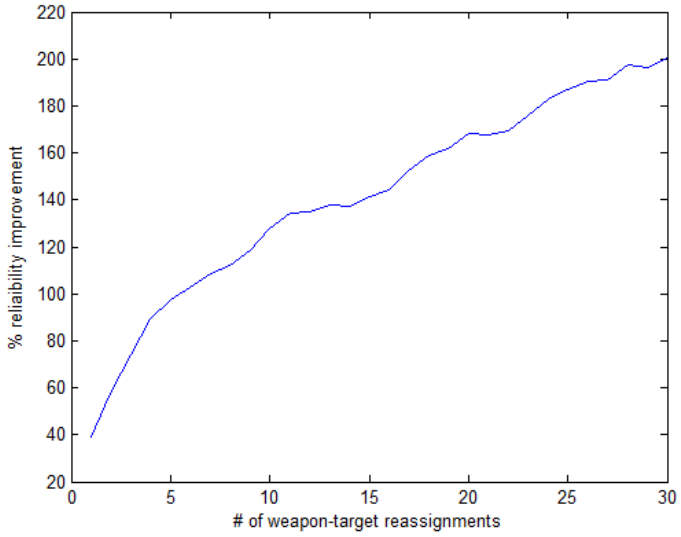

Figure 7: Reliability increase with re-assignments

As can be seen, initial re-assignments have the largest impact on increasing reliability. This fits with intuition, as choosing the single best reassignment will have a much greater impact on reliability than choose one reassignment af-

ter many others have already occurred. After many reassignments, the average improvement in reliability drops off dramatically, but with random problem instances of this size, even the 30th reassignment produces on average a measurable increase in reliability.

These examples show how system reliability can often be improved through weapon-target reassignment, and for specific small instances, these reassignments can be calculated and applied. However, as was proved earlier, this problem in general is NP-Hard, and therefore prompts the need for an approximation algorithm.

4.8 Multi-Site Reliability Problem

A further common extension of this set of problems involves the introductions of "sites", locations from which mobile weapons can be launched. This extension can be defined by the introduction of a new variable, s_i for each $i \leq n$. This variable defines the site for each weapon, so, for instance, a value of $s_1 = 2$ would indicate that weapon 1 is located at site 2. The number of sites is denoted by S.

For the previous example, let $s_1 = s_2 = 1, s_3 = s_4 = 2, s_5 = 3$. The original multi-site assignment, then, would be as shown in figure 8.

The calculated re-assignment from that example for the multi-site version would be as shown in figure 9.

This problem extension allows the concept of "distance" to be introduced as a new parameter of the problem. This parameter associates a distance for each site with each target. This can be represented as a new variable, d_{ij}, for all $i \leq S, j \leq m$. With this parameter introduced, a natural new problem is how

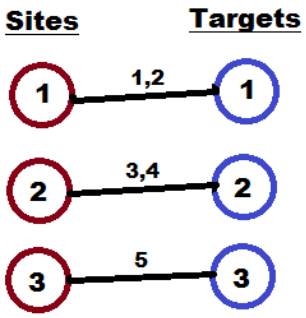

Figure 8: Multi-site original assignment

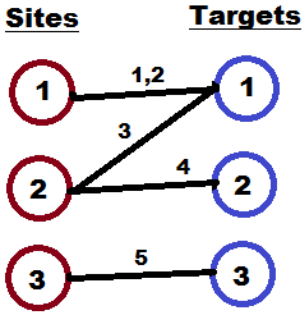

Figure 9: Multi-site re-assignment

to optimally re-assign weapons to targets such that the distance from weapons to targets is minimized. That is, find the re-assignment that minimizes:

$$\sum_{i=1}^{S} d_{ix}, \forall x : s_{f(x)} = i$$

This minimization only takes distance into account, however, which does not fit well with the overall goal of the problem. A more realistic real-world problem involves a balance of the two parameters; that is, finding a re-assignment that both minimizes the distance between sites and their asso-

ciated targets and maximizes reliability. Similar to the balance of goals in the non-linear integer programming formulation of the adversarial agents problem, weights α and β can be introduced to allow flexibility in the relative importance placed on the two goals. Using these weights, then, the problem is to minimize:

$$\alpha \sum_{i=1}^{m} (1 - f_x) \cdot v_i + \beta \sum_{j=1}^{S} d_{jx}, \forall x : f(x) = b_i, s_{f(x)} = i$$

4.9 Approximation Algorithm

Since it was previously demonstrated that the Generalized Assignment Problem with Reliability is NP-Hard, the use of exact algorithms to find solutions to instances of these problems may be infeasible for large program instances. For that reason, a heuristic approximation algorithm is defined for use with this problem. The algorithm is a greedy approximation algorithm loosely based on the algorithm in [Cohen et al., 2006].

The algorithm is adaptive to any α-approximation algorithm to the Knapsack problem, many of which are well-known. Say such an algorithm is defined, which is denoted as a, and which returns a set of Knapsack selections.

The approximation algorithm makes use of a vector of assignments which is updated through the run of the algorithm to arrive at the final assignment. This vector is referred to as T. $T[i] = j$ indicates that weapon x_i is assigned to target b_j, and $T[i] = -1$ indicates that weapon x_i has no target. The vector C holds the current values of the assignments, with $C_j[i] = c_{ij}$ if weapon i has no target, and $C_j[i] = c_{ij} - c_{ik}$ if weapon x_i is assigned to target b_k.

The algorithm then, is as follows:

Algorithm 1 Generalized Assignment Problem Approximation Algorithm

for $i = 1...n$
    $T[i] = -1$
for $j = 1...m$
    $S_j = a(b_j, P_j)$
    for $i \in S_j$
        $T[i] = j$

As can be seen, this algorithm repeatedly updates a set of assignments by repeated calls to the Knapsack α-approximation algorithm, in effect simplifying the more general problem by making use of approximate solutions to the simpler one.

4.10 Generalized Assignment Problem as Special Case of General Adversarial Agents Problem

In order to see that the Generalized Assignment Problem is, in fact, a special case of the broader general adversarial agents problem, much as with the reduction of the Knapsack Problem to the Generalized Assignment Problem, the concepts and variables from the Generalized Assignment Problem must be mapped to concepts within the adversarial agents problem. For this mapping, refer to the non-linear integer programming formulation of the adversarial agents problem.

If the "bins" of the Generalized Assignment Problem are considered as the friendly agents of the adversarial agents problem, and the "items" as enemy agents, the idea of the mapping begins to become clear. The capacities of each bin (c_i) map to the health of each friendly agent, with the profits (p_{ij}) mapping to the difference in values of the friendly and enemy agents, and the weights (w_{ij}) mapping to the capabilities of the associated friendly and enemy

agents. This portion of the mapping is shown more explicitly in Table 5.

Generalized Assignment	**Adversarial Agents**
B	F
X	E
c_i	H_i^f
p_{ij}	$V_i^f - V_j^e$
w_i	$C_i^f - C_j^e$

Table 5: Concept mapping

With this mapping defined, all that remains is to set up the general adversarial agents problem instance in such a way that allows for a direct mapping of friendly to enemy agents to be modeled with no other interactions (for instance, no additional attacks beyond the ones involved in the mapping). This is easily achieved by defining 1-timestep reachability for each pair of locations on the game board, and setting $T = 1$. In this way, each agent can reach each other agent in one timestep, and since the entire context time consists of only 1 timestep, each agent will move to a maximum of one new location. In this way, the assignment from the Generalized Assignment problem is captured by the general adversarial agents problem.

A reduction of the Generalized Assignment Problem to the general adversarial agents problem, therefore, is defined by taking an arbitrary instance of the Generalized Assignment Problem, and constructing an instance of the general adversarial agents problem. This is accomplished by first setting the parameters of the adversarial agents problem shown in Table 5 to the corresponding values of the given Generalized Assignment Problem instance. T is set to 1, and $R[x_1, y_1, x_2, y_2]$ is set to 1 for all values of x_1, y_1, x_2, y_2. This constructs the corresponding instance of the general adversarial agents problem and, as explained in this section, this new instance captures all of the fea-

tures of the Generalized Assignment Problem. If the generalized assignment problem instance has a solution, the Generalized Assignment Problem has the same solution, and vice versa. This is straightforward because the Generalized Assignment Problem is, in fact, a proper subproblem of the general adversarial agents problem.

Because the Generalized Assignment Problem is NP-Hard and this reduction exists, the general adversarial agents problem is also NP-Hard.

The implications of this are similar to those discussed at the end of the non-linear integer programming chapter, namely, that a complete analytical solution will not be computationally feasible for general instances of this problem. Because of this fact, heuristic algorithms are necessary for the general instance solver that is one of the main goals of this research. Attention will now shift to the inspiration, development, enhancement, and verification of those algorithms.

5 Artificial Immune Systems Approach and Enhancement

5.1 Introduction

Given the computationally difficult nature of the generalized adversarial agents problem and the infeasibility of using the non-linear integer programming solver approach for larger problem instances, heuristic algorithms are necessary for real-time solutions to these instances. In this and following chapters, such heuristic algorithms will be presented based on two biologically-inspired computational methods: Artificial Immune Systems and Particle Swarm Optimization. Novel variants of these algorithms to apply to the adversarial agents problem will be presented, and ultimately a hybrid algorithm incorporating elements of both will be given, with analytical and experimental verification.

In this chapter, the two major variants of the Artificial Immune Systems algorithm – the Negative Selection method and the Clonal Selection method – will be presented, along with an enhancement of the Negative Selection method, and a modified version of the Clonal Selection method to apply to instances of the adversarial agents problem. The novel enhancement of the Negative Selection method is an interesting and promising area of research in its own right, and the modified Clonal Selection method provides one of the fundamental building blocks of the final software system for this research.

The application of Artificial Immune Systems to the general adversarial agents problem, as well as the enhanced Negative Selection AIS algorithm are the major original contributions of this chapter.

5.2 Artificial Immune Systems Background

Artificial Immune Systems form a class of artificial intelligence algorithms designed to mimic portions of the classification and learning behaviors of biological immune systems. Biological immune systems display many desirable characteristics for computational classification and optimization algorithms, including complexity, robustness, and adaptivity [Aickelin and Dasgupta, 2005]. One of the major tasks of the immune system is to differentiate between "self" and "non-self" cells within the body. Non-self cells are treated as potentially hazardous and met with one of various immune responses. Artificial Immune Systems are inspired by this process, and perform the same type of classification using methods based on study of the workings of biological immune systems.

The AIS algorithm emulates the learning behavior of the immune system, as the training data set for a supervised learning problem is presented for the AIS to develop a series of "detectors", which correspond conceptually to lymphocytes. Lymphocytes are types of white blood cells that carry out the functions of the adaptive immune system, which allows for the learning of and adaptation to previously encountered pathogens over time. Similarly, within an AIS, the detectors are generated in such a way as to adapt to the training data in order to more accurately classify later data points.

The general process of an AIS for a binary classification task is to generate a series of detector "antibodies" – structures that have a shape and volume within the problem space – based on a training data set. These antibodies are then matched against "antigens" – new test data points within the problem space – which are classified based on their matching against the set of anti-

bodies, similar to how T-cells react with proteins within the human body to classify them as either self or non-self [Aickelin and Dasgupta, 2005].

These antibodies are generated using one of a variety of AIS methods, the most common of which are the clonal selection method and the negative selection method. Clonal selection is an evolutionary approach in which a population of antibodies is randomly created and then subjected to the Genetic Algorithm approach of reproduction and mutation (usually without the crossover operation). Negative selection for classification problems involves the random creation of a series of antibodies, generally represented as n-dimensional hyperspheres within a real-valued mapping of the problem space (see [Gonzalez and Dasgupta, 2003], [Gonzalez et al., 2002]), followed by a culling process in which those antibodies which react with any members of one of the binary classification classes (corresponding conceptually to the "self" cells) are eliminated. This is very similar to the biological process wherein the thymus rejects T-cells which react with any self cells [Aickelin and Dasgupta, 2005].

5.3 Negative Selection AIS Algorithm

The standard AIS algorithm is normally presented in terms of a binary supervised learning classification problem. That is, a set of training data $X = \{x_1, x_2, ..., x_n\}$ (where each x_i could be a vector of values, $x_i = [x_{i1}, x_{i2}, ..., x_{ik}]$) is given, with a corresponding set of class labels $Y = \{y_1, y_2, ..., y_n\}, y_i \in \{0, 1\}, \forall i$. The goal is to develop a function that accurately classifies not only the training data, but generalizes to be accurate in classifying new data points as well.

The standard AIS algorithm operates by generating a series of antibodies

within a feature space. Each dimension of the feature space can be any function of the input data. Some research has been done on the importance of feature selection within the AIS framework (e.g. [Mange et al., 2011]), but for simplicity, the space will here be described as the k-dimensional space defined by the k components of each input data point vector. Each input data point, then, can be represented as a point within this space. The AIS antibodies are spheres which occupy volume within the feature space and do not contain any "non-self" data points. Therefore the goal of an AIS is to fill as much of the "self" portion of the feature space as possible with antibodies.

As a simple example, consider Figure 10, showing a 2-dimensional feature space in which the '+' symbols represent "non-self" data points, and the dots represent "self" data points.

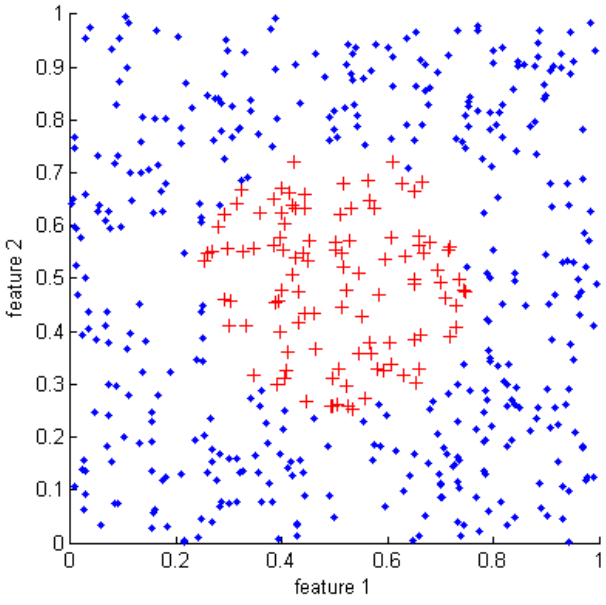

Figure 10: Example training data set for AIS

Given this set of training data, the series of AIS antibodies is then gen-

erated. Several methods have been proposed for deriving the antibody set, with the two most common being Negative Selection and Clonal Selection. In the standard Negative Selection method, circles (or spheres or hyper-spheres in higher dimension feature spaces) are randomly generated, and those that contain any "self" data point are rejected until a pre-determined number of valid antibodies has been established. Figure 11 shows an example of a series of antibodies generated in such a manner.

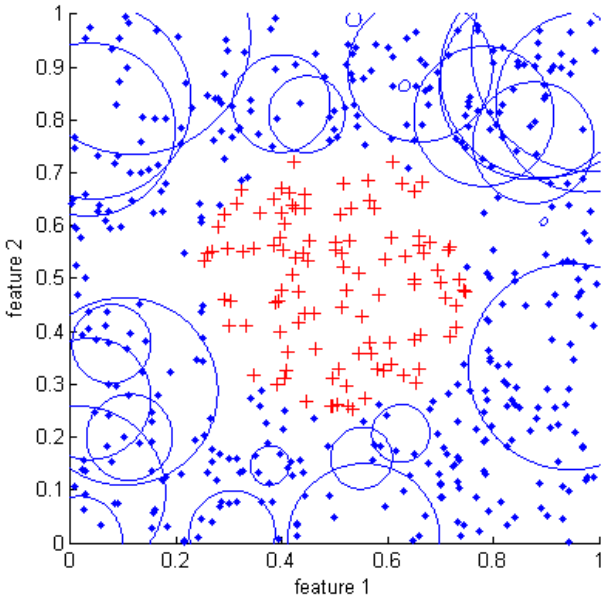

Figure 11: Example data with AIS antibodies

Several refinements to the basic Negative Selection algorithm have been proposed, including rejecting antibodies which overlap others, moving the center for antibodies which include any self data points, or increasing or decreasing the radius of a antibody under certain conditions ([Dhaeseleer et al., 1996]), but this illustrates the basic concepts involved. Once these antibodies have been generated using the training data set, the AIS system would be run on

the test data, with any test data point which fell within the radius of a antibody being labeled as "self", and all others as "non-self" (clearly, many more antibodies than shown in Figure 11 would be necessary for an accurate classification system). In this way, the AIS performs an adaptive learning-based binary classification task.

5.4 Enhanced Negative Selection AIS Algorithm

5.4.1 Algorithm Description

The enhanced Negative Selection AIS algorithm improves upon the standard version by generating antibodies in a more dynamic and adaptive manner, using a variant of the iterative generalization and specialization processes of the Current-best-hypothesis learning algorithm presented in [Russell and Norvig, 2010]. This enhanced algorithm not only is more accurate than the standard version at classification tasks, but also more closely models the actual variety and adaptability of lymphocytes in the biological immune system, which inspire the AIS algorithm in the first place.

In the enhanced AIS algorithm, instead of generating antibodies to fit all the training data at once, antibodies are generated iteratively, processing one test data point at a time. The algorithm makes use of two parameters, α, which controls the rate of expansion or contraction associated with the generalization and specialization processes, and β, which is a threshold value for when a new antibody is started.

The first self data point defines the initial antibody, which is formed as a square (or cube or hypercube depending on the feature space dimension) with each side having length 2α. From this point, each data point is processed

iteratively. If it is a self data point which already lies within an existing antibody, or a non-self data point lying outside all existing antibodies, nothing is changed. Otherwise, a process of generalization (for a self data point outside all existing antibodies) or specialization (for a non-self data point inside an existing antibody) occurs.

Figures 12 and 13 show these processes. In Figure 12, a non-self ('+' symbol) data point is processed and found to be within the antibody. This triggers the "specialization" process, in which the antibody changes shape to exclude this new data point, leaving a margin of α around the point (in this example, $\alpha = 0.1$).

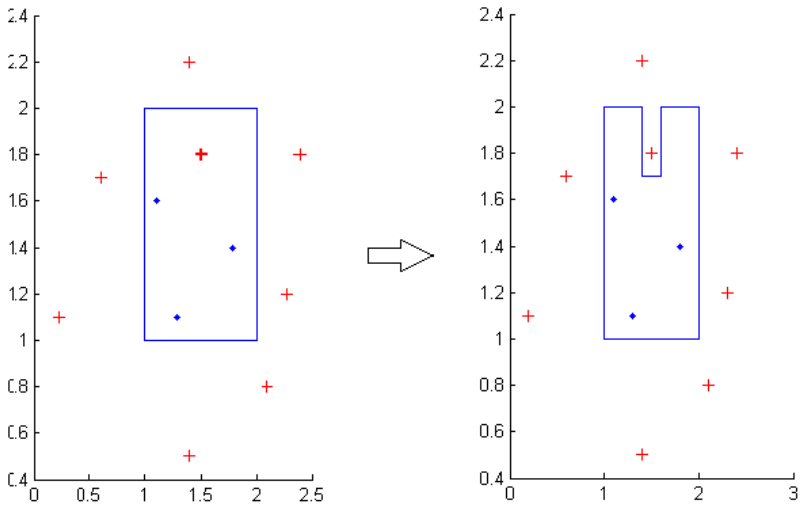

Figure 12: Enhanced AIS specialization

In Figure 13, a self data point is processed and found to be outside of all existing antibodies. This triggers the "generalization" processes, in which the antibody changes shape to include the new data point, again leaving a margin defined by α.

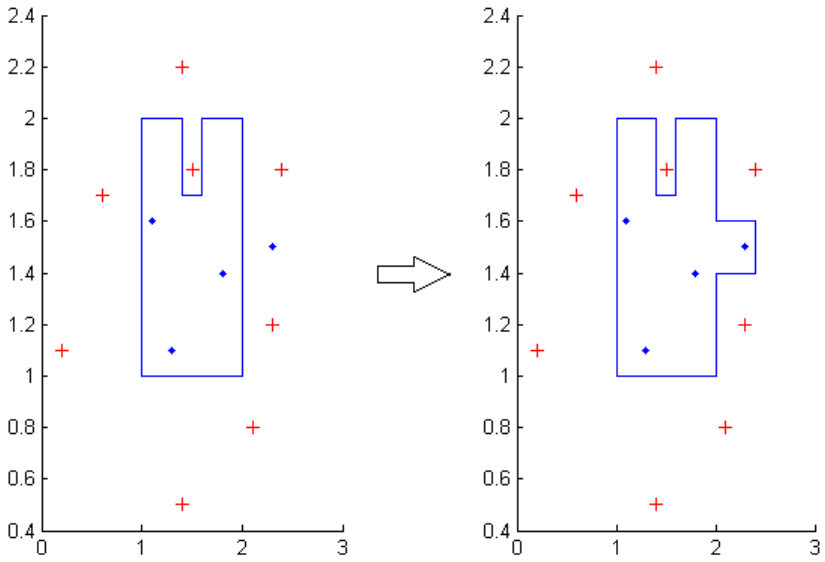

Figure 13: Enhanced AIS generalization

These are the two processes which iteratively change the shape of the antibodies while processing the training data set. For each data point, the nearest edge of any antibody is calculated. If the distance of the point from this edge is less than β, either a specialization or generalization process occurs. However, if a self point is at a distance greater than β from all antibodies, then a new antibody must be formed. This is done in the same manner as at the beginning of the antibody generation phase, where this single point defines the initial new antibody, as illustrated in Figure 14.

For each training data set point that is processed, either the antibody set remains unchanged, or one of these three processes takes places: specialization, generalization, or the creation of a new antibody. Some implementation details and special considerations have been omitted in this section for the sake of clarity; these will become clear in the pseudo-code for the enhanced Negative

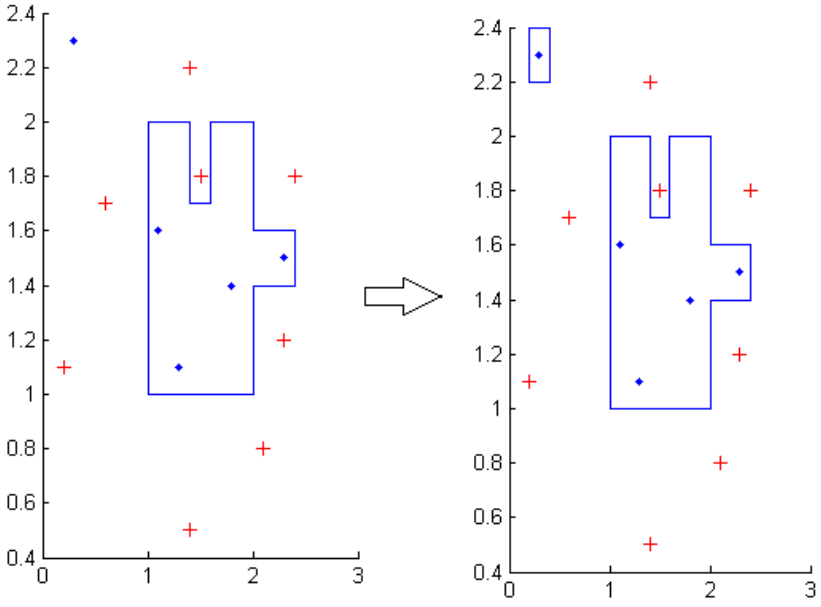

Figure 14: Enhanced AIS new antibody

Selection-based AIS algorithm.

5.4.1.1 Pseudo-code This section assumes the givens defined in the introduction to the standard AIS algorithm; X is a vector of training data inputs, with Y the corresponding class labels. α and β are the two parameters that are defined based on knowledge of the data set, or given default values.

Algorithm 2 contains the pseudo-code definition of the antibody set generation algorithm for the enhanced version of Negative Selection-based AIS.

Algorithm 2 Antibody set generation for the enhanced AIS algorithm

$D \leftarrow \{\}$
for all $x_i \in X$ **do**
  if $y_i = 1$ **then**
    if (x_i, y_i) not contained in any $d_j \in D$ **then**
      $dist \leftarrow$ distance to nearest edge of closest $d_j \in D$
      if $dist > \beta$ **then**
        $d \leftarrow$ square: $(x_i - \alpha, y_i - \alpha), (x_i - \alpha, y_i + \alpha), (x_i + \alpha, y_i + \alpha), (x_i + \alpha, y_i - \alpha)$
        $D \leftarrow D \cup d$
      else
        $d_j \leftarrow$ generalization of d_j by margin $min(\alpha$, maximum margin that does not include a non-self data point)
      end if
    end if
  else
    if (x_i, y_i) contained in any $d_j \in D$ **then**
      $dist \leftarrow$ distance to nearest edge of closest $d_j \in D$
      $d_j \leftarrow$ specialization of d_j by margin $min(\alpha$, maximum margin that does not include a self data point)
    end if
  end if
end for

5.4.2 Results

Due to the design of the enhanced algorithm, the most significant gain is in terms of run time for the detector generation portion of the AIS algorithm. Figure 15 shows the result of running both the standard AIS algorithm with Negative Selection antibody generation and the enhanced Negative Selection-based AIS algorithm with generalization / specialization based antibody generation on the same sample datasets of various sizes. These datasets are from publicly available classification algorithm verification repositories, from the UCI Machine Learning Repository ([University of California, 2012]).

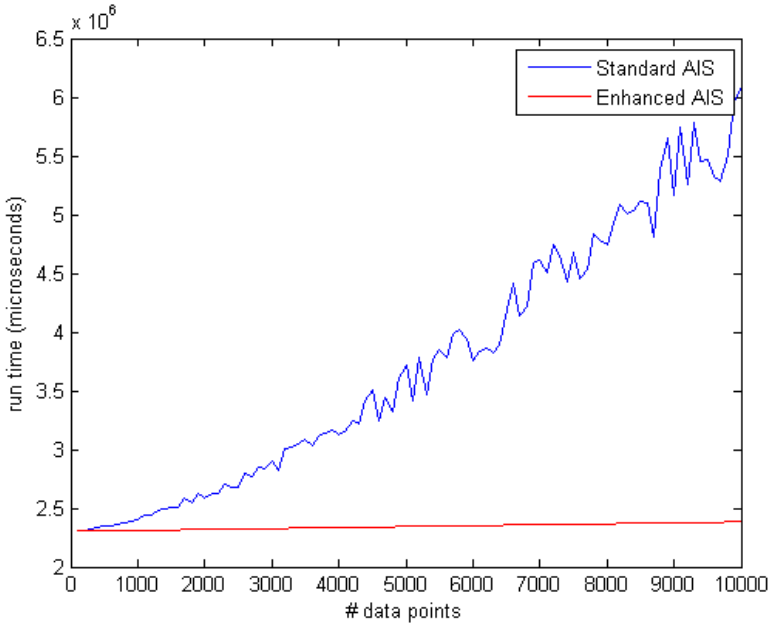

Figure 15: Run time comparison

The reason for the dramatic speed gain for the enhanced AIS algorithm, particularly with a large number of training data points, is immediately apparent when comparing the two detector generation strategies. Using the standard Negative Selection algorithm, each generated detector must be checked

against every "self" data point in order to ensure that none lie within the detector area, whereas by using the generalization / specialization method of the enhanced AIS algorithm, processing each data point requires only constant time. Thus as the number of training data points increases, the difference in running time between the two algorithms becomes increasingly pronounced, and the advantage of the enhanced AIS algorithm more apparent.

In testing, the enhanced Negative Selection AIS algorithm also showed increased classification accuracy over the standard Negative Selection AIS algorithm in all four of the tested data sets, with a significant gain in two of them. Figure 16 shows the classification accuracy results for the tested data sets.

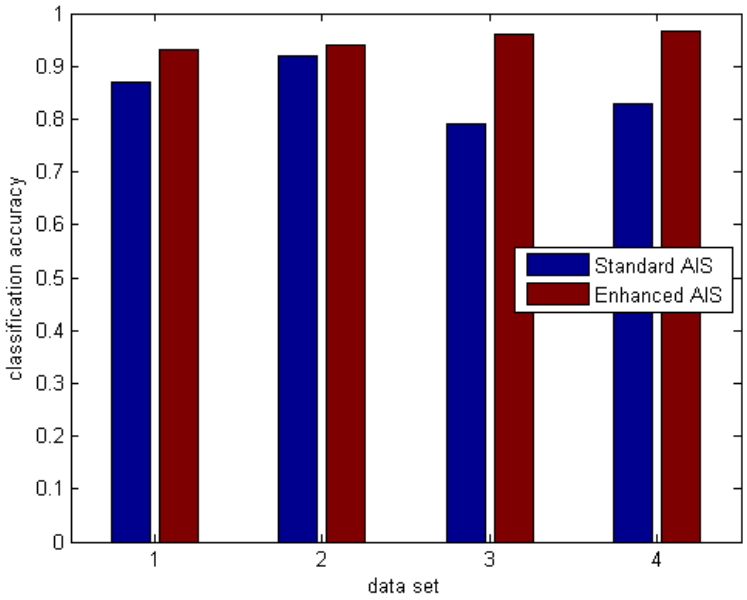

Figure 16: Classification accuracy

These results are very promising, particularly when considered alongside the speed gains previously discussed. Since the goal of the enhanced AIS al-

gorithm was to provide a method for producing more complex detector shapes which would more accurately fit the problem space, it was expected that, if properly implemented, it would show improved classification accuracy over the standard algorithm, and experimentally this expectation was verified. Using data sets 3 and 4, a particularly dramatic improvement was observed. The reason for this likely lies in the data itself, that these data sets contain more complex data that is not as easily divisible within the feature space, thus limiting the effectiveness of the standard AIS Negative Selection approach.

Of course, because there are several variants of even the standard AIS algorithm, and because the effectiveness of each can be influenced by the selection of algorithmic parameters, it is difficult to strongly state that the enhanced AIS algorithm presented in this paper is an unqualified improvement over the standard AIS approach for general data sets. However, given the dramatic speed gains of the enhanced algorithm, along with the improvement in classification accuracy for all of the tested data sets, it appears that the enhanced AIS algorithm does provide a significant advantage over the standard algorithm. In addition, the complexity and mutability of the detectors within the enhanced AIS algorithm more closely resembles those same aspects of the lymphocytes within the biological systems that provide the inspiration for the entire AIS approach.

5.5 Clonal Selection AIS Algorithm

5.5.1 Biological Inspiration

The Clonal Selection AIS method draws biological inspiration from a slightly difference perspective from the Negative Selection method. While Negative

Selection is modeled on the maturation process of T-cells within the immune system, Clonal Selection is inspired by the process of B and T lymphocytes developing improved responses to antigens over time. This process is known as "affinity maturation". "Affinity" refers to the level to which an antibody recognizes and produces an appropriate response for a specific antigen. Within the biological immune system, as a set of antibodies are exposed to the same antigen repeatedly, its affinity for that antigen will increase. This happens by means of production of new antibodies with a greater natural affinity for the antigen in question. This process allows for the "memory" behavior of the immune system, whereby the body is better able to deal with diseases it has encountered in the past. This is also the process that allows for inoculations and immunizations – introducing just enough of an antigen to allow the immune system to develop a mature antibody set to deal with them, yet not enough to incur symptoms of the disease itself.

5.5.2 Algorithm Description

As might be inferred from the description of the biological inspiration for the Clonal Selection AIS method, the algorithm shares traits with Genetic Algorithms, or Evolutionary Algorithms. The Clonal Selection method is most naturally suited for optimization problems, as opposed to the Negative Selection method, which is most naturally suited for classification problems. In particular, this algorithm lends itself well to unconstrained numerical optimization problems, as does Particle Swarm Optimization. How to modify such algorithms to solve instances of heavily constrained optimization problems like the adversarial agents problem is more completely discussed when the hybrid AIS / PSO algorithm is introduced.

As with a Genetic Algorithm, within the Clonal Selection framework, an encoding must first be selected that allows candidate solutions to the optimization problem to be represented in such a way that "mutations" (much like those in a GA context) can be efficiently performed. A random initial population of these encoded solutions is then established. Then, until some stopping condition is reached – usually either time or affinity-based – the population undergoes a series of iterations of the familiar genetic operations – selection, cloning, and mutation. This process is more thoroughly laid out in Algorithm 3.

Algorithm 3 Clonal Selection AIS method

$Population \leftarrow RandomInitialPopulation()$
while $!StopCondition$
  for $p_i \in Population$
    $CalculateAffinity(p_i)$
  $P_{selected} = Select(Population, SelectionSize)$
  $P_{clones} = \{\}$
  for $p_i \in P_{select}$
    $P_{clones} \leftarrow P_{clones} \cup Clone(p_i, CloneRate)$
  for $p_i \in P_{clones}$
    $Mutate(p_i, MutationRate)$
    $CalculateAffinity(p_i)$
  $Population \leftarrow Select(Population, P_{clones}, PopulationSize)$
  $P_{rand} = RandomIndividuals(RandomRate)$
  $Replace(Population, P_{rand})$

5.6 Adversarial Agents Problem Approach

As shown, the Negative Selection AIS method is most naturally suited to classification problems, whereas the Clonal Selection AIS method is most naturally suited to optimization problems. This distinction is one of the interesting and largely unexamined areas that provided some of the initial inspiration for this

research. Particle Swarm Optimization, as the name suggests, is also most naturally suited to optimization problems. In order to explore some of the tradeoffs and crossover between optimization algorithms and classification algorithms, the hybrid algorithm presented in the next chapter incorporates the Negative Selection AIS method – mainly a classification method – with Particle Swarm Optimization – an optimization method. The result is an algorithm that outperforms both for certain tasks, and will form the basis of the system to solve instances of the adversarial agents problem.

6 Particle Swarm Optimization Approach, Hybrid Algorithm

6.1 Introduction

This chapter focuses on Particle Swarm Optimization (PSO). The PSO algorithm is first defined, then its application to the general adversarial agents problem as formally defined within the non-linear integer programming chapter is discussed. Examples of this direct application of PSO are provided.

Next, a novel algorithm is presented which is a hybrid of PSO and the Artificial Immune Systems algorithm explored in the previous chapter. This hybrid algorithm exploits some of the desirable features of both algorithms in order to provide a new optimization algorithm that is useful in a variety of contexts, including the general adversarial agents problem under consideration. This hybrid algorithm is rigorously tested using a variety of standard PSO benchmark functions, and is shown to outperform the standard PSO algorithm for certain optimization tasks.

The hybrid algorithm defined in this chapter forms another of the key building blocks of the final software system for solving arbitrary instances of the general adversarial agents problem, and it, along with the Particle Swarm Optimization approach to the general adversarial agents problem, are the major original contributions of this chapter.

6.2 Particle Swarm Optimization

Particle Swarm Optimization algorithms simulate the behavior of certain groups of organisms. PSOs involve the use of a "swarm" of particles, each of which

is represented as a point within the search space for an optimization problem. These particles move through the search space according to rules based on the best-known position both of the individual particle and the swarm as a whole. PSOs have been shown to perform very well for optimizing complex functions [Kennedy et al., 2001], and have become popular optimization algorithms.

The basic operation of a PSO is to first randomly place the swarm of particles in the n-dimensional search space, assigning each a random velocity. Then, the position (x) and velocity (v) of each particle – both of which are represented as n-element vectors – are updated with an element of randomness, in a series of iterations according to the following equations [Kennedy et al., 2001]:

$$v_{i+1} = \omega v_i + \alpha \cdot rand() \cdot (l_i - x_i) + \beta \cdot rand() \cdot (g_i - x_i) \tag{1}$$

$$x_{i+1} = x_i + v_i \tag{2}$$

x_i denotes the position of the particle at iteration i, v_i the velocity at iteration i, l_i the local best-known position (best-known to the individual particle) at iteration i, and g_i the global best-known position of the swarm at iteration i. ω, α, β are parameters of the PSO algorithm.

Although in the standard PSO algorithm the particles are initially placed randomly within the problem search space, it has been shown that the initial placement of the particles can greatly impact both the speed and likelihood of convergence to a global minimum (or maximum) [Kennedy et al., 2001]. The hybrid algorithm presented later in this chapter takes advantage of that fact.

6.3 PSO Approach to the Adversarial Agents Problem

6.3.1 General Approach

The particle swarm optimization algorithm as described in the previous section applies most naturally to unconstrained numerical optimization problems. However, the adversarial agents problem as presented in the non-linear integer programming formulation is heavily constrained. Indeed this problem gives rise to a wide variety of natural constraints for values of any variables to be found through machine learning, and thus does not easily translate to an unconstrained numerical optimization context. In this section it is explained how the PSO algorithm is adapted to solve instances of the generalized adversarial agents problem.

The most direct way to move from the non-linear integer programming formulation of the problem to a PSO-based approach is to incorporate "feasibility space checking" into the PSO algorithm. [Hu and Eberhart, 2002] describe one such approach, which has much in common with many artificial intelligence algorithms within game theory. Each particle, both when it is first initialized and when its position is updated during each iteration of the PSO run, has its new position checked by each system constraint to ensure that the new position lies within the feasible region of the search space. In this way, only feasible solutions – that is, solutions which conform to all of the constraints defined in the non-linear integer programming formulation of the problem – are learned by the swarm and stored in local or global swarm memory. This is very similar to move-legality checking within many game theory algorithms – generated moves are examined for legality according to the game rules, which correspond conceptually to context constraints for the adversarial

agents problem.

The similar, although computationally much faster approach used within this algorithm involves constraining the positions of the particles within the movement step itself. That is, instead of allowing for purely random movement and then checking the feasibility of the new position, movement is done randomly only within the set of feasible surrounding space. This avoids the need for repeated rejected random movements, particularly when the number of feasible movements from a particular position is small.

It is important to note that considering the movement variables of the non-linear integer programming formulation of the adversarial agents problem as the dimensions of a search space yields a very highly dimensional ($F \cdot T \cdot D$ PSO dimensions, where D is the number of dimensions in the adversarial agents problem itself), yet highly constrained search space. This is exactly the type of problem a PSO-based algorithm applies well to – because the updating rules for the particles are straightforward and computationally inexpensive to implement, particles can move quickly even through very high-dimension search spaces, and because the search space itself is highly constrained, there are not in fact as many extra feasible solutions as the famous "curse of dimensionality" would normally provide for.

6.3.2 Examples

To demonstrate the particle swarm optimization approach to the central problem involved in this research, as well as to verify its accuracy and usefulness, the modified PSO algorithm described in the preceding section was applied to the same examples given in the non-linear integer programming chapter. The results are shown and explained below.

Once again, the first example problem is shown in Figure 17, with the circles representing friendly agents with their associated capabilities, and the square representing an enemy agent. The originally defined health constraint was again modified as described in the examples section of the non-linear integer programming chapter, and this example was implemented as described in the previous section, with the movement variables (X^f and Y^f) defining the dimensions of the PSO search space. Thus each particle in the PSO swarm represents a strategy selection for the problem instance, and because of the feasibility space modification of the PSO algorithm, each of these strategies will conform to the constraints defined. Thus the final solution of the modified PSO algorithm will define a high-value (based on the definition of the objective function), legal (based on the definition of the constraints) movement strategy.

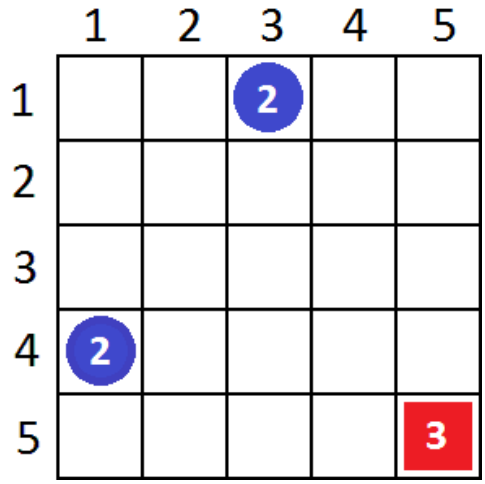

Figure 17: Example instance 1

The number of dimensions for the PSO search space for this example is $F \cdot T \cdot D = 2 \cdot 6 \cdot 2 = 24$, although again a value for each dimension can be only a small number of possibilities, unlike a traditional real-valued search space. Each PSO particle represents a point within this 24-dimensional space,

conceptually corresponding to a series of movements for each of the friendly agents through all the timesteps of the system.

Near the beginning of the PSO run when the particles are spread randomly, the interpretation of these points is somewhat ridiculous, as the friendly agents move in random patterns, often within the same area as their starting location. However, as the PSO runs, more of the search space is explored, and better values for the objective function are found. This is why carefully expressing the objective function for a particular problem type is so important – as the algorithm finds new and better values for the objective function, the corresponding generated strategy becomes stronger. For instance, in a problem of this type, including within the objective function a calculation of the ending distance of the friendly agents from the enemy agents allows for the PSO to converge toward solutions which increase this value, that is, strategies where the friendly agents get closer and closer to the enemy agents.

To demonstrate this more concretely, an example is shown in table 6 of an interpretation of the initial value of one of the PSO particles for this example instance. As can be seen from inspection, both friendly agents simply move randomly without approaching the enemy agent.

t	$X^f[1,t]$	$Y^f[1,t]$	$X^f[2,t]$	$Y^f[2,t]$
0	1	4	3	1
1	1	3	2	1
2	2	3	1	1
3	2	4	2	1
4	1	4	2	2
5	1	5	2	2
6	2	5	2	3

Table 6: Example instance 1 initial particle interpretation

This can be compared with table 7, which shows an interpretation of the

next improved (in terms of the objective function value) location for the particle. As can be seen, although the friendly agents still do not reach the enemy agent, this particle location and corresponding generated strategy is much closer to a reasonable strategy for the instance.

t	$X^f[1,t]$	$Y^f[1,t]$	$X^f[2,t]$	$Y^f[2,t]$
0	1	4	3	1
1	1	3	3	1
2	2	3	3	2
3	2	4	3	3
4	3	4	4	3
5	3	4	4	2
6	4	4	4	3

Table 7: Example instance 1 next step particle interpretation

The final results at the end of the PSO run, for the generated X^f and Y^f values, are shown in table 8.

t	$X^f[1,t]$	$Y^f[1,t]$	$X^f[2,t]$	$Y^f[2,t]$
0	1	4	3	1
1	1	4	3	2
2	2	4	4	2
3	3	4	4	3
4	3	5	4	4
5	4	5	4	5
6	5	5	5	5

Table 8: Example instance 1 output

The same results are shown visually in Figure 18. As can be seen, although the path for each agent is a bit more erratic than the non-linear integer programming solution, they do still show the synchronized behavior required for a successful strategy for this problem instance. Thus the value of the objective function for the strategy generated by the PSO-based algorithm is exactly the same as the value for the strategy generated by the non-linear integer pro-

gramming solver. Thus the two strategies are equivalent, although different. This strongly indicates the viability of the PSO-based approach for solving these problem instances.

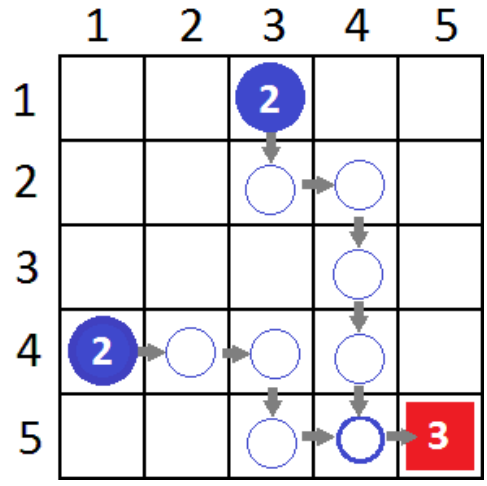

Figure 18: Example instance 1 solution

The second example again is the same as the second example in the non-linear integer programming chapter, and is displayed in Figure 19. Once again, the constraints are modified as described in that section, and again this example requires a more complex strategic plan for success.

The results of running the modified PSO-based algorithm for this problem instance are shown in table 9, showing the generated values of X^f and Y^f, which again represent the generated strategy.

The same results are shown visually in Figure 20. Once again, an equivalent although different strategy was generated by the PSO-based algorithm, which again produced success for the problem instance, with again the same value for the objective function as found by the non-linear integer programming solver. This provides further confirmation of the accuracy and usefulness of this approach for these problem instances, and provides a baseline which is

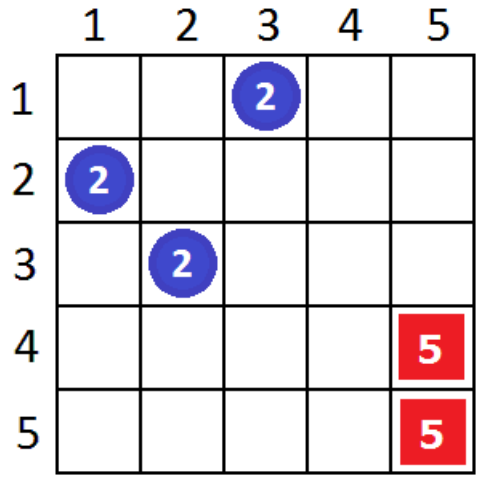

Figure 19: Example instance 2

extended by the hybrid AIS / PSO algorithm described in the following section, which is used in the final system to solve instances of this problem.

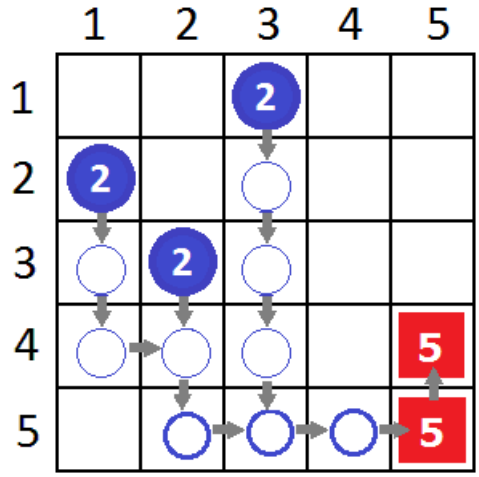

Figure 20: Example instance 2 solution

t	$X[1,t]$	$Y[1,t]$	$X[2,t]$	$Y[2,t]$	$X[3,t]$	$Y[3,t]$
0	1	2	2	3	3	1
1	1	3	2	4	3	2
2	1	4	2	5	3	2
3	2	4	2	5	3	3
4	2	5	2	5	3	4
5	3	5	3	5	3	5
6	4	5	4	5	4	5
7	5	5	5	5	5	5
8	5	5	5	5	5	4
9	5	5	5	5	5	5
10	5	4	5	4	5	4

Table 9: Example instance 2 output

6.4 Hybrid AIS / PSO Algorithm

6.4.1 Introduction

Thus far, there have been few efforts to produce hybrid algorithms incorporating elements of both Artificial Immune Systems and Particle Swarm Optimization. Part of the reason for this may be that AIS algorithms are most naturally suited to classification problems (though some efforts have been made to apply these algorithms to the context of optimization, see for instance [Coello and Cortes, 2005]), while PSO algorithms are most naturally suited to optimization problems.

In this section, a new algorithm is presented which is a hybrid of the negative selection AIS method and a standard PSO. The new algorithm uses a modified negative selection algorithm to restrict the initial placement of the particles in the PSO to areas of the search space that are likely to yield the global minimum (or maximum). Through the use of common PSO benchmark functions, it is shown that this new hybrid algorithm outperforms the standard PSO algorithm for optimization tasks.

6.4.2 Hybrid Algorithm

The hybrid AIS / PSO algorithm first makes use of a modified AIS negative selection method to generate a set a antibodies, then places the initial PSO particles in such a way that they do not "react" (in the AIS sense) with any of the antibodies.

The standard negative selection method randomly generates a series of antibodies, represented as spheres in the n-dimensional problem space, with both the location and radius of the sphere initially random [Gonzalez and Dasgupta, 2003]. This fits well with the context of a binary classification problem, as for each antibody, every point within the problem space is either inside or outside the antibody. However, for optimization tasks, the objective is not to classify data points, but rather to find the global minimum (or maximum). For this reason, for this algorithm, the negative selection algorithm is modified by first randomly generating only the *position* of the antibodies, then assigning each radius proportionally based on the value of the objective function at the antibody's center. Thus, for a minimization problem, those antibodies with the highest objective function value would also have the largest diameter. Since no particle can initially "react" with any antibody (that is, lie within its n-dimensional hyper-sphere), this makes it unlikely that any particle will be placed in the vicinity of this high-function-value point in the search space, and thus more likely that all particles will have initial values closer to the global minimum than if they were placed randomly. This is defined explicitly in the pseudo-code for the antibody generation and particle positioning phase of the hybrid algorithm, in Algorithm 1 below.

Two additional parameters are needed for the hybrid algorithm, which

are referred to as k and γ in the pseudo-code below. k defines the number of antibodies to generate, and γ defines the maximum size for the radius of an antibody, assuming a normalized search space in the range $[0, 1]$ for each dimension. The algorithm is presented in a simplified, readable form that shows the generation of the antibodies and the placement and movement of the particle swarm within the search space. To apply to an instance of the general adversarial agents problem, the number of dimensions of the search space would first be calculated based on the parameters of the problem instance, and during the PSO run, constraints would be enforced to ensure that particles were examining only feasible solutions, as previously discussed.

S defines the number of particles in the PSO swarm, n refers to the number of dimensions in the problem search space, and f is the objective function to be minimized.

Once this initial AIS-based portion of the algorithm is complete, the PSO algorithm is run as usual, terminated either by a number of iterations or a target value for the objective function.

Figure 21 shows an example of how the antibody generation and PSO particle placement work. On the left is a graph of the 2-dimensional Rastrigin's function (see "Experimental Design"), and on the right is a graph of a small number of generated antibodies (represented by circles) and initial placement of a small number of PSO particles (represented by Xs). The Figure shows the relationship of the antibody size to the function value. Antibodies having centers at positions with large function values are proportionally larger than those with centers at positions with small function values. One can also observe from comparing the function graph with the placement of the particles that

Algorithm 4 Hybrid algorithm: antibody generation and particle positioning

```
max = −∞
min = ∞
// random generation of antibody centers
for i = 1..k
    for d = 1..n
        A[i, d] = rand()
    max = MAX(max, f(A[i]))
    min = MIN(min, f(A[i]))

// calculation of antibody radii
for i = 1..k
    R[i] = γ · (f(A[i]) − min)/(max − min)

// generation of PSO particles
for i = 1..S
    antibodyReaction = true
    while antibodyReaction == true
        antibodyReaction = false
        for d = 1..n
            P[i, d] = rand()    // particle position
            V[i, d] = rand()    // particle velocity
        for j = 1..k
            delta=0
            for d = 1..n
                delta+ = (A[j, d] − P[i, d])^2
            if √delta < R[j]
                antibodyReaction = true
```

even with a small number of antibodies, the swarm as a whole is much more optimally placed to find the global minimum (which for this function occurs at the origin) than if the particles were placed randomly, as with the standard PSO algorithm.

The effect of these antibodies for general adversarial agents problem instances is to effectively eliminate portions of the search space for the initial placement of the PSO particles. Since each point within the search space rep-

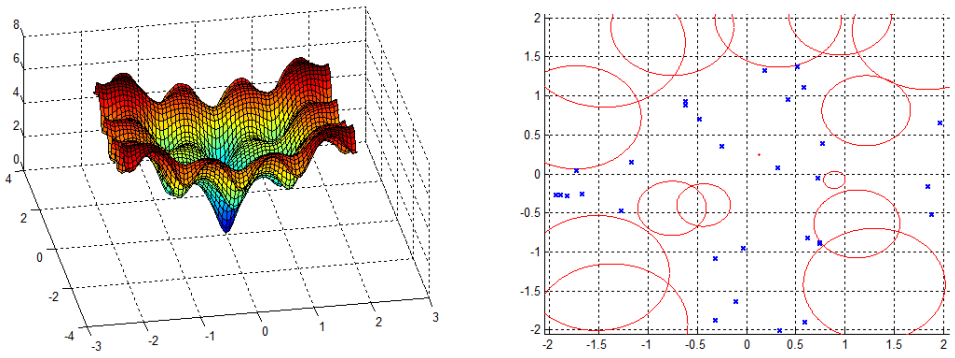

Figure 21: Hybrid algorithm example – Rastrigin's function and generated antibodies

resents a set of strategy decisions that form a potential solution to the problem, this means that each antibody eliminates a set of these potential solutions.

As an illustration, consider the first example problem instance in figure 17. One of the generated antibodies for this problem instance might be centered at the point defined by the values in table 10. If this antibody had a radius of 1, then it would eliminate from the initial placement of the PSO particles both the strategy illustrated by this table, which is shown visually in figure 22, and any strategy paths that differed from this path by only one move. Although this is only one antibody and its individual effect is limited, it is straightforward to observe that a large set of these antibodies could be very helpful in eliminating from consideration strategy choices that are unlikely to be good; that is, that are unlikely to produce good values for the objective function associated with the problem instance. This, indeed, is exactly the effect of the AIS Negative Selection portion of the hybrid algorithm.

6.4.3 Experimental Design

In order to test the usefulness of the hybrid algorithm, it was tested against the standard PSO algorithm on a number of common benchmark optimiza-

t	$X^f[1,t]$	$Y^f[1,t]$	$X^f[2,t]$	$Y^f[2,t]$
0	1	4	3	1
1	1	5	4	1
2	2	5	5	1
3	3	5	5	2
4	3	4	4	2
5	2	4	3	2
6	2	3	2	2

Table 10: Center of generated antibody

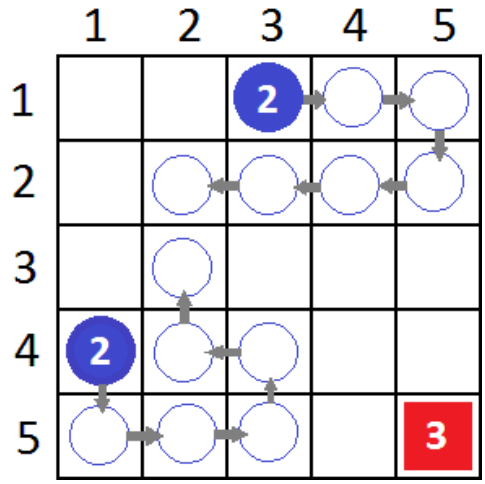

Figure 22: Visualized center of generated antibody

tion functions. Since the hybrid algorithm takes the form of an optimization algorithm as opposed to a classification algorithm (as with the standard AIS Negative Selection algorithm), comparing it to the standard PSO algorithm is the most effective way to establish its advantages for those types of problems to which it most naturally applies. Several complex functions are common in benchmarking PSOs and other optimization algorithms; for the purposes of testing four of the most common of these functions ([Shi and Eberhart, 1999], [Vesterstrom and Thomsen, 2004], [Yap et al., 2011], [Xie et al., 2002]) were chosen, defined as follows:

Rastrigin's function ($-5.12 \leq x_i \leq 5.12$):

$$f(x) = \sum_{i=1}^{n} \left(x_i^2 - 10 \cdot cos(2\pi x_i) + 10\right) \tag{3}$$

Griewank's function ($-600 \leq x_i \leq 600$):

$$f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \tag{4}$$

Rosenbrock's function ($-2.048 \leq x_i \leq 2.048$):

$$f(x) = \sum_{i=1}^{n-1} \left(100\left(x_{i+1} - x_i^2\right)^2\right) + \left(x_i - 1\right)^2 \tag{5}$$

Ackley's function ($-32.668 \leq x_i \leq 32.668$):

$$f(x) = -20 \cdot exp\left(-0.2\sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}\right) - exp\left(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)\right) + 20 + e \tag{6}$$

The global minimum for each of these functions is 0.

The same PSO parameters were used for both algorithms, taken from the PSO parameter chart given by [Pedersen, 2010]. The additional parameters used for the hybrid algorithm were $\gamma = 0.2, k = 100$. In testing, the dimensionality of the benchmark functions had little effect on the results, therefore the most familiar 2-dimensional versions were used.

For each of these functions, both the new hybrid algorithm and the standard PSO algorithm were run 1000 times for each of several values for the number of iterations (see "Results"), and averaged the objective function value

found by each algorithm over all runs. In addition, each algorithm was run 1000 times with a target of the actual global minimum for each function, to compare the average number of iterations required to find the global minimum. The results are presented in the following section.

6.4.4 Results

The results of the comparison between the hybrid AIS/PSO algorithm and the standard PSO algorithm are displayed in the Figures below – Rastrigin's function in Figure 23, Griewank's function in Figure 24, Rosenbrock's function in Figure 25, and Ackley's function in Figure 26. Rosenbrock's function is particularly interesting; although the value of the minimized objective function drops very quickly with a relatively small number of PSO iterations, the actual global minimum required more iterations to find than any of the other functions for both algorithms (see Figure 6). Overall, the speed at which the algorithms converged toward the global minimum varied by function, but in each case the hybrid function outperformed the standard PSO function for every PSO iteration limit, with an average function value improvement of 13.58%.

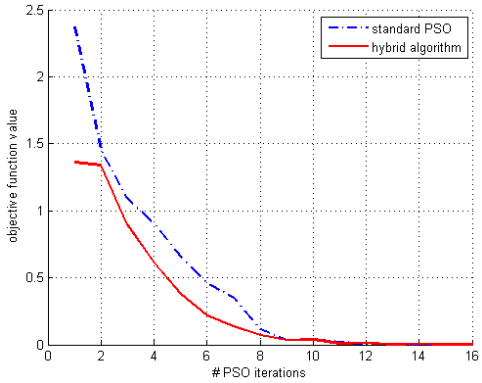

Figure 23: Rastrigin's function algorithm comparison

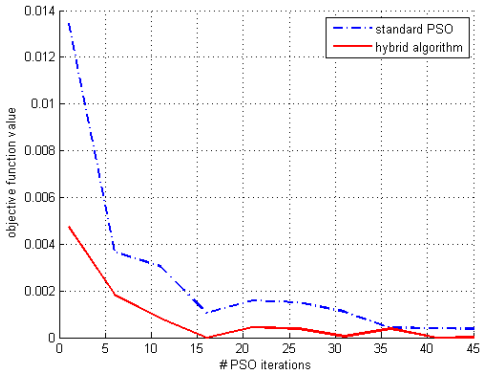

Figure 24: Griewank's function algorithm comparison

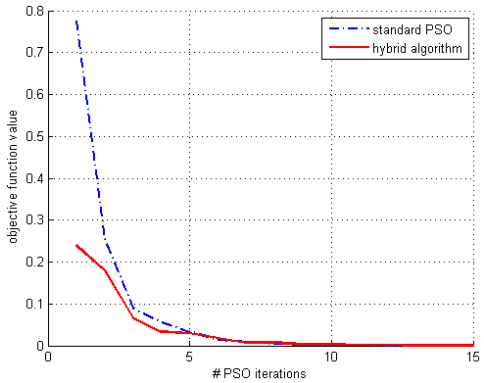

Figure 25: Rosenbrock's function algorithm comparison

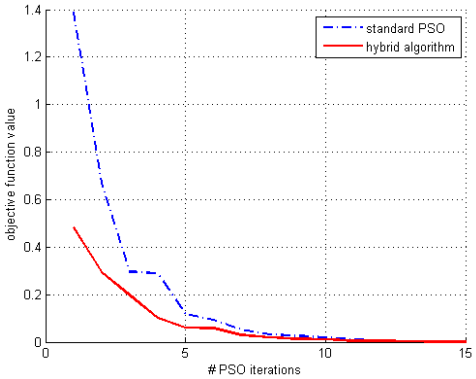

Figure 26: Ackley's function algorithm comparison

Finally, the results for the comparison of the number of iterations necessary to find the global minimum for all of the test functions is displayed in Figure 27. In every case, the hybrid function found the global minimum in

significantly fewer iterations than the standard PSO, on average with 26.89% fewer iterations.

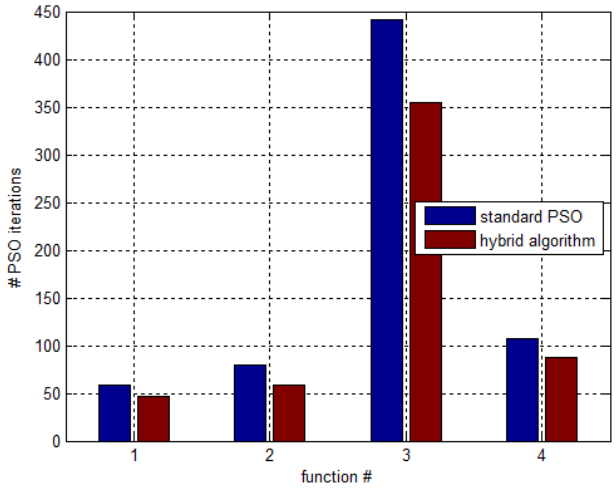

Figure 27: All functions PSO iterations comparison

6.4.5 Comparison with Non-Linear Solver

Since the hybrid AIS / PSO algorithm is a heuristic algorithm which provides approximate solutions for problem instances, of particular interest is a performance comparison between this hybrid algorithm and the non-linear integer programming solver used in a previous chapter. In that chapter, the claim was made that an exact non-linear integer programming solver was not a viable option for larger problem instances, because the non-linear integer formulation creates an NP-Hard problem (which the generalized adversarial agents problem is in general, as proved in a previous chapter). That infeasibility is demonstrated in this section.

A number of random problem instances were created with increasing numbers of variables and constraints, and the time required by the two algorithms to terminate was recorded. An averaged version of the results is shown in

Figure 28, with the average time plotted against the total number of variables and constraints. This figure shows the problematic phenomenon, as even with relatively low numbers of variables and constraints, the time required for the exact non-linear integer programming solver to terminate starts to increase very sharply as compared to the hybrid algorithm, much as expected.

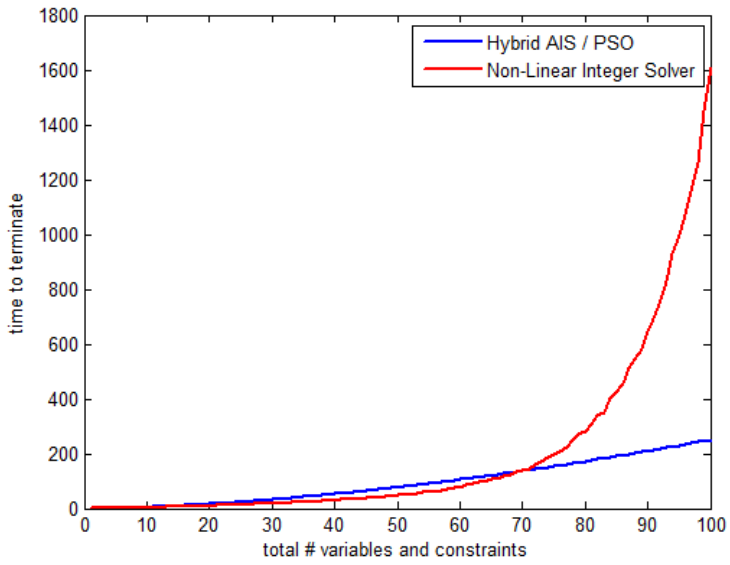

Figure 28: Hybrid algorithm vs. non-linear integer solver time

Since these were randomly generated problem instances, the exact details of the instances, such as time scale and problem representation, are far less important than the overall comparison of the two algorithms, and the demonstration of the need for a heuristic approximation algorithm.

The phenomenon is, in fact, somewhat more interesting than the averaged graph indicates. The number of constraints affects the termination time of the non-linear integer programming solver more overall than the number of variables. Figure 29 shows a mesh surface plot of this phenomenon, with the termination time for the non-linear integer programming solver plotted against

both the number of constraints and the number of variables in the problem instance.

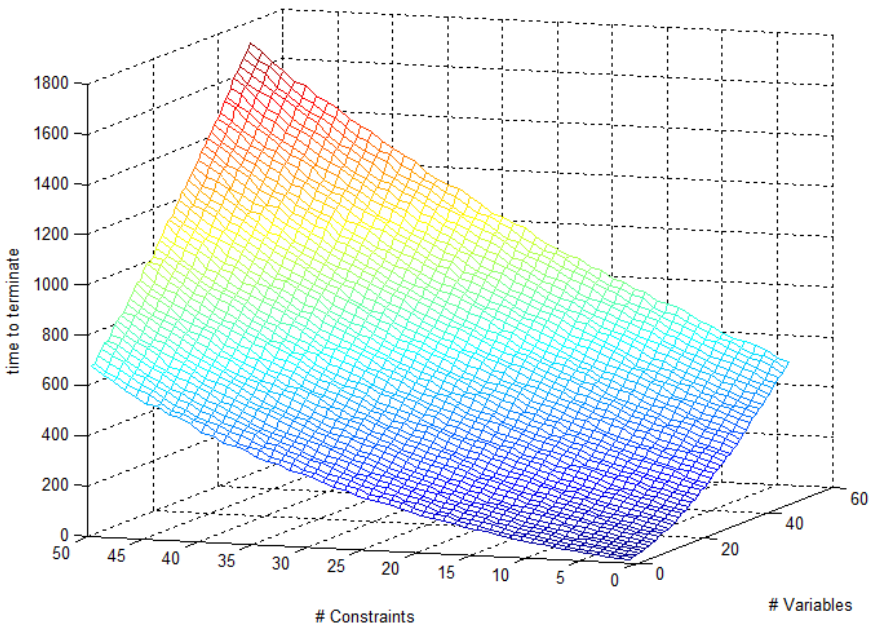

Figure 29: Constraints vs. variables for termination time

This same general phenomenon was observed with each of the main algorithms presented in this paper, including the non-linear integer programming solver, the PSO-based algorithm, the modified Clonal Selection AIS algorithm, and the hybrid algorithm – that constraints affect the performance of the algorithm more than variables. This fact will be important when considering the constraint module of the main software system presented in the following chapter.

The hybrid algorithm presented in this section combining the AIS negative selection method with the PSO algorithm clearly outperformed the standard PSO algorithm in benchmark optimization tasks. Thus by exploiting some of the desirable features of the immune-inspired AIS system, by establishing

antibodies to restrict the placement of the swarm particles, the standard PSO algorithm was improved upon by over 25%, based on the average number of iterations required to find a global minimum. This demonstrates the feasibility not only of this hybrid algorithm in particular, but of the hybrid approach in general – the possibilities of improving upon existing computational methods by intelligently combining them with other compatible methods. This hybrid algorithm forms one of the core components of the overall system to provide instances of the generalized adversarial agents problem.

7 Complete Software System

7.1 Introduction

In this chapter, the final software system is presented, based on the theoretical work and algorithmic building blocks defined in the previous chapters. The modular design of the system is first presented, with definitions of each of the important modules and their interactions within the software. The core heuristic algorithm, based on the work of the previous chapters, is then defined. Next, learning and knowledge representation within the system is discussed.

Finally, the results of the system are presented in several ways. Carefully selected example problem instances are defined and the results of running them within the software system are shown. These instances demonstrate the flexibility of the software system and its applicability to a variety of contexts. Since the general adversarial agents problem is itself so general and can capture so many different scenarios, these example instances demonstrate some of that variation, in addition to providing further experimental evidence of the correct function of the system, in addition to the theoretical evidence provided by earlier chapters.

7.2 Overall Design

The final software product of this research is a modular system to solve general instances of the adversarial agents problem. The final system was written in the C# language, using a modular, object-oriented approach. As presented in earlier chapters, the novel hybrid PSO / AIS optimization algorithm forms the heart of this software system, and performs the constrained numerical

optimization. This algorithm follows the modular approach of having self-contained units with well-defined inputs and outputs, which is very important for this particular application, precisely because the general adversarial agents problem is such a generalized problem, and can be used to model a wide variety of contexts (some of which are demonstrated in the Examples in a later section).

Given these inputs and outputs, the first task in running the software system on a problem instance is to implement an input module, which presents the problem to the heuristic algorithm in a way that corresponds to its interface. It is possible to create a generic input module based purely on the definition of the variables and constants in the non-linear integer programming formulation of the problem – and indeed, such a module was designed and implemented – but ultimately this strategy was discarded in favor of the designable input module approach. The reason for this is that not all of the variables and constants which comprise a full definition of the problem are necessary for many problem contexts. Allowing the users of the software system to create their own input modules allows for more flexibility in expressing the problem data in a way that is sensible to its context.

Recall that the only input to the hybrid PSO / AIS algorithm for the general adversarial agents problem that forms the search space in which the algorithm operates are the X^f and Y^f variables, as these are the variables which define the position of each agent at each timestep, and thus model the strategic choices involved in the problem. The obvious next concern, then, is how constraints are handled within the software system.

Because of the way the hybrid heuristic algorithm is defined and implemented, the non-linear integer programming formulation constraints serve to

constrain potential strategic choice selections within the problem, or, equivalently, to define the feasible region within the search space. The modular design of the software system then allows for a constraint module with a straightforward interface: it is a boolean module that accepts as input potential solutions to the problem instance (or, equivalently, points within the search space), and provides a determination of whether the solution is a feasible one. This allows the hybrid PSO / AIS algorithm to run efficiently with repeated calls to the constraint module in order to perform the constrained optimization.

Finally, there is the question of the objective function. For maximum flexibility, this is also designed as a separate module for the system, again with well-defined inputs and outputs. This setup allows for not only a straightforward mathematical objective function, but also more complex programmatic functions based on the context of the problem instance. This is helpful for allowing various types of problems to be expressed in a way that can be handled efficiently by the software system.

Thus, once the input, constraint, and evaluation modules are defined, the constrained numerical optimization is performed by the hybrid heuristic algorithm. From there, the results can be written directly to a data file, or an output module can be defined, to allow for visualization or post-processing of the output data.

This description covers the case of single-use optimization. The software system, however, can also be used in an "online" fashion, whereby the optimization steps are performed repeatedly, and input and output data are updated at various iterations. This allows for more complex scenarios such as an Artificial Intelligence combat system, where enemy forces are being moved and updated as the problem instance is carried out. This situation starts to

reveal the power of the software system – it can be used as an actual learning machine, where generated strategy decisions are responded to by the adversary, and the context space is continually updated to create a new situation which must then be dealt with by new strategy decisions, and so on.

For this kind of online usage, a control module must be defined which controls the use of input and output information for repeated invocations of the heuristic algorithm. The control module also controls the parameters of the heuristic algorithm, for instance, dynamic control over the number of iterations or optimization limit for each run. For example, in the combat system context, a "player" might only be interested in strategy generation for the immediate future, as anything beyond that is likely to change in response to adversarial actions, and thus the control module could limit the heuristic algorithm to run only within a search space relevant to that timeframe.

In addition, the control module defines the actions of the adversary within the problem instance. This can be as simple as no actions (as in the very simple instances shown in previous chapters) or as complex as an entire separate Artificial Intelligence system to control strategy generation for the adversary. In this way, for example, the software system could be used to test the strength of possible game theory AI players, by using the AI algorithm in the control module to generate strategy decisions for the adversary.

The software system also allows for an explicit learning or knowledge module. The adversarial agents problem up to this point has largely been dealt with in terms of a perfect-information game theory game, but this is not always the case. For instance, again in the context of a combat situation, one side might not know the location and capabilities of all enemy forces initially. This learning module allows for a dynamic representation of knowledge, along

with confidence intervals that can be updated through the run of the system, as new facts are learned, and old knowledge estimates are strengthened or weakened. This aspect of the software system will be discussed in more detail in a later section.

Therefore, a high-level view of the interaction of the main system modules is shown in Figure 30.

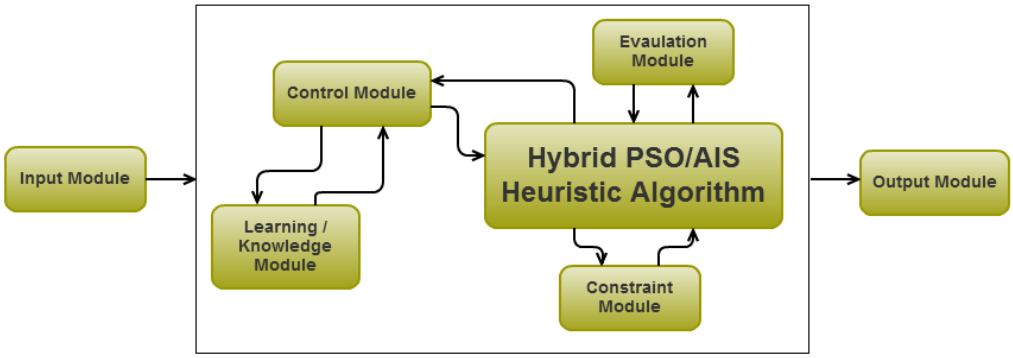

Figure 30: System module diagram

7.3 Heuristic Algorithm

The hybrid PSO / AIS algorithm that forms the heart of the optimization portion of the software system has already been described in some detail in a previous section. This algorithm was implemented both in Matlab for initial testing and verification, and in C# for later use within the software system. The commented code for the basic Matlab version is included as an Appendix, as it is likely most understandable for the general reader. This algorithm is both a basic research contribution in its own right, and also is one of the major components of this software system.

7.4 Learning and Knowledge Representation

As stated previously, the general adversarial agents problem has largely been presented as a perfect-information game theory game, that is, one in which both sides know all facts about the agents involved. However, to allow for maximum flexibility and applicability of the software system, this does not need to be the case. The software system also allows for the modeling of partial-information games, situations in which one side does not have certainty about those facts.

It is for this purpose that the learning / knowledge module of the system is designed. Since the interface for this module is very generic, it allows for the knowledge representation within to vary quite a bit. For this reason, specifics of knowledge representation will be presented here only in a cursory fashion, as it is mostly outside the immediate context of this research.

At a high level, knowledge within this kind of software system is generally represented as a table of facts – information such as enemy agent positions, or capabilities, or estimates of future actions – with associated confidence intervals. Confidence intervals are a way of expressing the probability that a fact is true, thus high confidence is associated with a fact the system is largely sure of, and low confidence is associated with facts that might be simply rough estimates. These confidence intervals can be updated over the run of the system.

For a system like this, a common and powerful way to express knowledge is in terms of Bayesian confidence intervals ([Rasmussen, 2004], [Sebastiani, 2002], [Dietterich, 2000]). This knowledge formulation uses the Bayesian concept of prior and posterior distributions, in which facts are initially estimated

according to some distribution considered most likely, and then this estimation is repeatedly updated based on observations during the run of the system. Thus the facts within the knowledge fact table increasingly correspond to the actual situation as observed during the course of the problem instance.

There are several other possibilities for expressing knowledge and machine learning for the general adversarial agents problem. Again, because of the modular system design, the software system is able to accommodate a wide variety of these possibilities as long as the ultimate inputs and outputs conform to the interface specification for the module.

7.5 Results and Example Instances

The final software system has proved very powerful and has successfully handled even very large and complex instances of the general adversarial agents problem. In this section, some smaller, more understandable instances will be presented along with the system results in order to demonstrate its function. These example instances also will display some of the generality of both the problem as defined and the software system, in that they express problems from widely separated context areas.

7.5.1 Example Instance: Combat Scenario

Since the most common context in which this problem has been discussed is that of a combat scenario, the first example instance models that context. For this instance, a cooperative scenario of the type common in robotics Artificial Intelligence applications was created, in which three "friendly" units must work together in order to prevent a faster "enemy" unit from escaping. This

problem was expressed in two dimensions, and the initial problem instance setup is show visually in Figure 31.

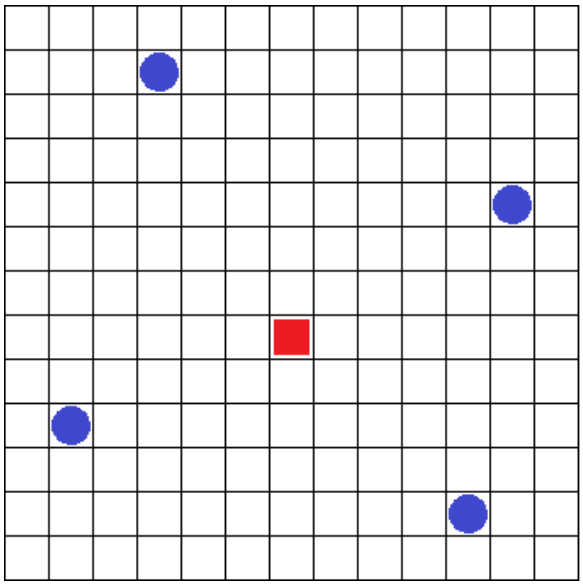

Figure 31: Combat instance: initial setup

In this problem instance, again the circles represent friendly agents and the square represents the enemy agent. The friendly agents can all move at a speed of 2 spaces per timestep (Manhattan distance), and the enemy agent can move at a speed of 3 spaces per timestep. These constraints are handled by the constraint module within the software system. The capabilities of the agents are defined such that any one of the friendly agents can destroy the enemy agent – this is to focus the strategic portion of the problem instance on preventing the enemy agent from escaping.

The adversary AI is defined within the control module of the system, and for this instance utilizes a basic alpha-beta tree search mechanism. The enemy agent attempts to maximize its average distance from all friendly agents 3 timesteps into the future, given the best estimated moves for the friendly agents. For this scenario, this AI algorithm creates a realistic strategy in

which the enemy agents "runs" from the friendly agents as they attempt to surround it.

The evaluation module for this instance, which again models the objective function for the instance, was written to limit the "escape vectors" of the enemy agent. That is, a position was evaluated based on the number of Manhattan-distance paths from the enemy agent's position to any edge of the context space, and this function was minimized by the heuristic algorithm. This again demonstrates the flexibility of the evaluation module approach within the software system – that sort of advanced processing would not be possible to express with a mathematical formula, but it is the behavior of most interest for this problem instance.

The results of this problem instance run on the software system are shown visually in Figure 32, with subsequent context spaces showing subsequent timesteps. As can be seen, the system solves the problem instance successfully, and the friendly agents display cooperative behavior as they surround and eventually defeat the enemy agent.

Clearly, the success or failure of the software system on specific instances of the problem depends heavily on well-designed and implemented control, constraint, and evaluation modules. With that caveat, however, the software system provides a powerful framework for both expressing and solving these program instances.

7.5.2 Example Instance: Game Theory

In addition to the combat scenarios which in large part motivated the definition of the problem and the design of the software system, the general adversarial agents problem as formulated can be used to describe more pure

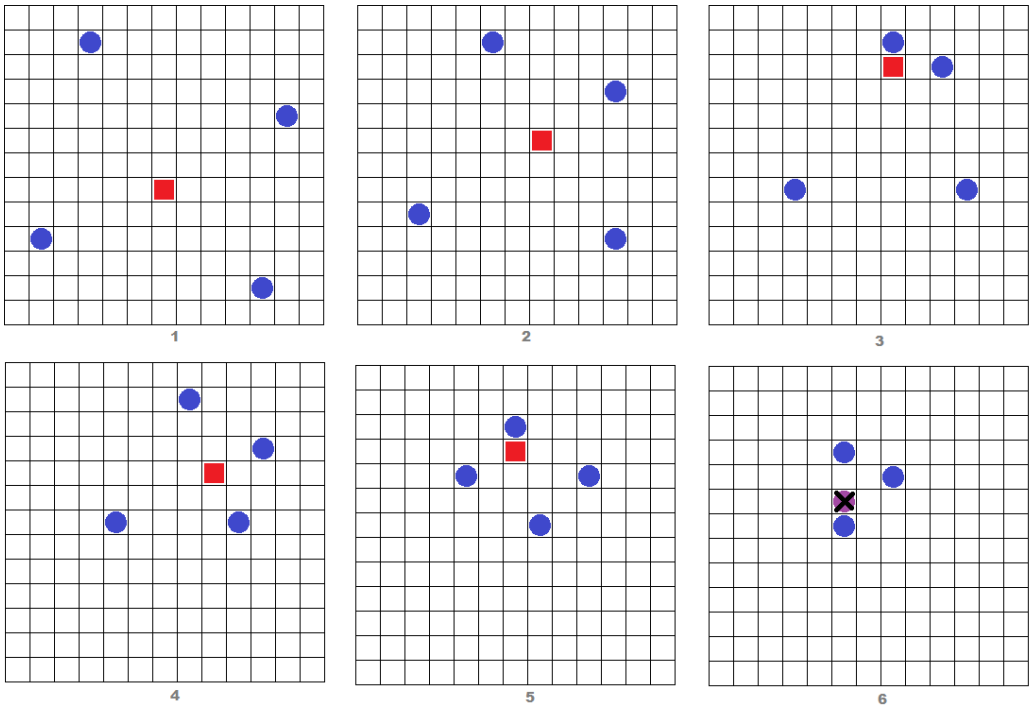

Figure 32: Combat instance: solution

game-theoretic problems. One of the most common games used to illustrate artificial intelligence algorithms for game theory problems is Tic-Tac-Toe (see [Fogel, 1993], [Matheus and Rendell, 1989], [Angeline and Pollack, 1993], etc.) because of its simplicity to show visually, and the fact that most people can understand almost completely the strategy involved for success.

In order to demonstrate the applicability of the software system to such game theory problems, an instance to play Tic-Tac-Toe was also created. The constraint module for this instance is very straightforward, since the rules of Tic-Tac-Toe are very simple. Two different versions of the control module were used: one in which a human player could input moves directly, and one in which moves were generated randomly.

The design of the evaluation function for this problem instance further illustrates the power of the software system's framework, specifically in its

ability to incorporate other "helper" algorithms within specific contexts. In this case, a Monte-Carlo evaluation function was created, whereby a board position is scored by semi-randomly playing out thousands of games from that position, and assigning a score based on the average outcome of those games. This is a similar approach to that used in [Wang and Gelly, 2007] and other Monte-Carlo-based AI approaches.

Incorporating this evaluation module into the software system produced a very strong AI player, the results of which are displayed in Table 11. Against the random-move generation strategy, the software system won 84% of the time, and never lost. Against the human player, the software system won 60% of the time, and also never lost. These are impressive results, but of course Tic-Tac-Toe is a very simple and computationally solvable game. Nonetheless, this again demonstrates the validity of the overall framework, and the ability to incorporate other helpful algorithms along with the heuristic algorithm for more powerful methods of solving specific instances or groups thereof.

	Wins	**Losses**	**Ties**	**Win %**
Random	837	0	163	84%
Human	18	0	12	60%

Table 11: Tic-Tac-Toe results

Because this Tic-Tac-Toe example is perhaps the simplest to understand, the C# code corresponding to a simplified (for readability) version of this example is included as an appendix. This code demonstrates the interaction of the various system modules in solving a simple problem instance.

7.5.3 Example Instance: Pathfinding

Finally, to show the power of the software system to solve instances of the general adversarial agent problem in completely different contexts, an example of using the problem to automatically create a maze pathfinding algorithm is shown.

First, the input module for this instance was created. In this case, the input module needed to read in a text file and create a maze object that would be shared by the other modules of the software system. The example maze text file, in tabular form, is shown in Table 12, with the 'S' designating the maze start, the 'F' designating the maze finish, and dots ('.') representing valid spaces within the maze.

#	#	#	#	#	#	#	#	#	#	#	#
#	.	.	.	#	.	.	.	.	.	.	#
S	.	#	.	#	.	#	#	#	#	.	#
#	#	#	.	.	.	.	.	.	#	.	#
#	.	.	.	.	#	#	#	.	#	.	#
#	#	#	#	.	#	F	#	.	#	.	#
#	.	.	#	.	#	.	#	.	#	.	#
#	#	.	#	.	#	.	#	.	#	.	#
#	.	.	.	.	.	.	.	.	#	.	#
#	#	#	#	#	#	.	#	#	#	.	#
#	.	.	.	.	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#	#	#	#	#

Table 12: Example maze text file

For this example, the search space in which the hybrid PSO / AIS algorithm was run had more dimensions than previous "online" instances of the software system, because for this instance the desired output is the entire maze solution, not simply the immediate next strategy decisions. Expressed in the same terms as earlier general adversarial agents game instances, the desired output

is values of locations for the "agent" (that is, the X^f and Y^f values from the non-linear integer programming formulation). Therefore the constraint module for this instance limits potential solutions to valid paths, where each step is 1 Manhattan-distance away from the previous step, and the path starts at the S and ends at the F locations marked in the maze text file.

The evaluation module is the most straightforward for this problem instance. Since validity of potential solutions is handle by the constraint module, the evaluation module merely returns a score based on the only factor of importance for this context – the length of the solution. Therefore, with a minimization by the hybrid algorithm, shorter valid solutions will be given preference over longer ones.

Once again, the software system performed admirably on this example instance. The generated solution is shown in Table 13, with the '*' characters representing the path found by the system. As one can see, the path found is, in fact, the shortest path through the maze.

#	#	#	#	#	#	#	#	#	#	#	#
#	*	*	*	#	.	.	.	.	.	.	#
S	*	#	*	#	.	#	#	#	#	.	#
#	#	#	*	*	.	.	.	.	#	.	#
#	.	.	.	*	#	#	#	.	#	.	#
#	#	#	#	*	#	F	#	.	#	.	#
#	.	.	#	*	#	*	#	.	#	.	#
#	#	.	#	*	#	*	#	.	#	.	#
#	.	.	.	*	*	*	.	.	#	.	#
#	#	#	#	#	#	.	#	#	#	.	#
#	.	.	.	.	.	.	#	.	.	.	#
#	#	#	#	#	#	#	#	#	#	#	#

Table 13: Example maze solution

The software system with these same modules was tested on a number of

other map text files available online with solutions. In all cases, the system found a valid path through the maze, and in all but a very small number, it found the shortest path. At times a slightly longer path was returned, most likely due the common PSO phenomenon of getting stuck and a local minimum rather than a global one.

Together, these examples reveal both the efficiency and the generality of the software system. Using a straightforward modular design, a user can construct modules appropriate to the context of the problem instance he is interested in, and use the software framework to solve those instances. Any valid instance of the general adversarial agents problem, which has been repeatedly shown to be an extremely general, wide-reaching, and computationally difficult problem, can be solved using this software system.

8 Extensions and Conclusion

The research in this dissertation provides several original contributions to the fields of artificial intelligence and machine learning. Several original algorithms were presented and demonstrated to be more useful than established algorithms for certain classes of problems. The general adversarial agents problem was defined formally and formulated as a non-linear integer programming problem, and a number of novel approaches to solving arbitrary instances of the problem were explored. Theoretical research was presented on the general adversarial agents problem itself, as well as a number of related computer science problems. Finally, a modular software system to solve instances of this problem was defined and demonstrated.

In addition to being useful and applicable in its own right, the research in this dissertation also provides a foundation for a number of potential extensions and further research within this area. Many of those extensions were explored at length in the individual chapters on the areas to which that new research would apply. In particular, perhaps the most immediate extension of the core software system would be to incorporate a more sophisticated learning and knowledge module. This would allow the software to be extended to further problem domains in which knowledge of the adversary or environment is only partially known and can change over time. A related extension would be to "fuzzify" either the variables or processing of the system. A fuzzy logic based approach would provide a new perspective on the problem, and has yielded promising results in similar types of systems.

The general adversarial agents problem is a very general problem. Many different problems, both from theoretically interesting fields like game theory,

and from real-world arenas like robotics control and military strategy generation, can be modeled as instances of this general problem. Although there are many software programs to deal with specific sub-problems in specialized areas, no software framework exists to solve all general instances of the general adversarial agents problem. Such a framework is valuable not only as a tool in its own right, but also as a standardized platform through which to evaluate other algorithms and approaches.

This dissertation provides that software system, and demonstrates its effectiveness in a variety of contexts. In addition, the research involved provides several other more fundamental results, including a proof that the general adversarial agents problem is NP-Hard, an accurate and usable non-linear integer programming formulation of the problem, an original enhancement of the Negative Selection Artificial Immune Systems method, and a novel Artificial Immune Systems and Particle Swarm Optimization hybrid algorithm, along with verification of improvements. Each of these contributions has value in its own right from somewhat disparate areas of Computer Science research, but they are drawn together in the software system presented, which is capable of solving any instance of the general adversarial agents problem in a flexible and extendable way.

Appendices

The appendices contain code to demonstrate the concepts presented in this dissertation.

Appendix A contains the Matlab version of the code for the hybrid PSO / AIS algorithm. This version of the code is easiest to read and shows the fundamental workings of the heuristic algorithm.

Appendix B contains a simplified version of the C# code for the Tic-Tac-Toe instance of the complete software system. This code is commented and is intended to demonstrate primarily the interaction of the various software modules that form the software system. Toward that end, some of the more complex details of the actual problem instance have been ommitted, such as the extended UCT algorithm and the hybrid PSO / AIS algorithm (as that is presented in Appendix A).

Appendix A

Hybrid PSO / AIS Matlab Code

```
function returnVal = hybridOptimization(doAIS)
    close all
    dimensions = 2; % number of dimensions of the search space
    numParticles = 153;         % these two parameter ...
        values based on
    params = [0.41 2.13 1.06];  % paper, referenced

    lowerBound = -32.668;
    upperBound = 32.668;

    % assign initial positions
    positions = (upperBound-lowerBound) * ...
        rand(numParticles,dimensions) + lowerBound;

    if doAIS==1
        % generate AIS Negative Selection antibodies
        numAntibodies = 100;
        alpha = 0.2;

        antibodies = (upperBound-lowerBound) * ...
            rand(numAntibodies,dimensions) + lowerBound;
        % optionally plot the antibody initial positions
        %plot(antibodies(:,1), antibodies(:,2),'rx')
        maxValue=-999999;
        minValue=9999999;
        for i=1:numAntibodies
            v = f(antibodies(i,:));
            antibodyValues(i)=v;
            maxValue = max(v, maxValue);
            minValue = min(v, minValue);
```

```
28          end
29
30          % plot the circles, calculate radii
31          for i=1:numAntibodies
32              % calculate radius of each antibody relative to ...
                   its objective
33              % function value
34              antibodyRadii(i)=alpha * ...
                    (upperBound-lowerBound) * ...
                    ((antibodyValues(i)-minValue) / ...
                    (maxValue-minValue));
35              % optionally plot antibodies as circles
36              %circle(antibodies(i,1), antibodies(i,2), ...
                   antibodyRadii(i));
37          end
38
39          % generate particle positions using antibody ...
               information
40          for i=1:numParticles
41              check=0;
42              while check==0
43                  check = 1;
44                  pos=(upperBound-lowerBound) * ...
                        rand(1,dimensions) + lowerBound;
45                  % check against all antibodies
46                  for j=1:numAntibodies
47                      dist=0;
48                      for d=1:dimensions
49                          dist = dist + ...
                                (pos(d)-antibodies(j,d))^2;
50                      end
```

```
51                      dist = sqrt(dist);
52                      if dist < antibodyRadii(j)
53                          check=0;
54                          break;
55                      end
56
57                  end
58              end
59              positions(i,:)=pos;
60          end
61      end
62
63      % set random velocities, find current local and global ...
            bests for the
64      % swarm
65      velocities = (upperBound-lowerBound) * ...
            rand(numParticles,dimensions) + lowerBound;
66      bestKnowns = positions;
67      globalBest = positions(1,:);
68      for i=2:numParticles
69          if f(positions(i,:)) < f(globalBest)
70              globalBest = positions(i,:);
71          end
72      end
73
74      iters=0;
75      while f(globalBest) > 0 && iters < 1000
76          for j=1:numParticles
77              for d=1:dimensions
78                  % main PSO velocity update function
79                  velocities(j,d) = params(1)*velocities(j,d) ...
```

```
                    + params(2) * rand() * ...
                    (bestKnowns(j,d)-positions(j,d)) + ...
                    params(3) * rand() * ...
                    (globalBest(d)-positions(j,d));
80              end
81              % update N-dimensional positions of each particle
82              positions(j,:) = positions(j,:) + velocities(j,:);
83
84              if f(positions(j,:)) < f(bestKnowns(j,:))
85                  bestKnowns(j,:) = positions(j,:);
86                  if f(positions(j,:)) < f(globalBest)
87                      globalBest = positions(j,:);
88                  end
89              end
90          end
91          % optionally plot the updated positions
92          %plot(positions(1,:),positions(2,:),'x')
93          iters=iters+1;
94      end
95
96      returnVal = iters;
97      %plot(positions(1,:),positions(2,:),'x')
98  end
```

Appendix B

C# Simplified Software System Code

TicTacToeForm.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.ComponentModel;
4  using System.Data;
5  using System.Drawing;
6  using System.Linq;
7  using System.Text;
8  using System.Windows.Forms;
9
10 namespace TicTacToe
11 {
12     public partial class TicTacToe : Form
13     {
14         private ControlModule controlModule = new ...
               ControlModule();
15
16         public char[,] board = new char[3, 3];
17         public char human = 'X';
18         public char computer = 'O';
19
20         public TicTacToe()
21         {
22             InitializeComponent();
23         }
24
25         private void resetButton_Click(object sender, ...
               EventArgs e)
26         {
```

```
27              resetBoard();
28          }
29
30          private void resetBoard()
31          {
32              for (int i = 0; i < 3; i++)
33                  for (int j = 0; j < 3; j++)
34                      board[i, j] = ' ';
35
36              syncBoard();
37          }
38
39          private void syncBoard()
40          {
41              for (int i = 1; i < 4; i++)
42                  for (int j = 1; j < 4; j++)
43                  {
44                      Control[] results = ...
                            this.Controls.Find("label" + i + j, ...
                            true);
45                      results[0].Text = board[i-1, ...
                            j-1].ToString();
46                  }
47
48          }
49
50          private void handleClick(int row, int col)
51          {
52              row--;
53              col--;
54              if (board[row, col] == ' ')
```

```
55              {
56                  board[row, col] = human;
57                  Control[] results = ...
                        this.Controls.Find("label" + (row+1) + ...
                        (col+1), true);
58                  results[0].Text = human.ToString();
59              }
60              else
61                  MessageBox.Show("Please choose an empty spot");
62          }
63
64          private void label11_Click(object sender, EventArgs e)
65          {
66              handleClick(1, 1);
67          }
68
69          private void label12_Click(object sender, EventArgs e)
70          {
71              handleClick(1, 2);
72          }
73
74          private void label13_Click(object sender, EventArgs e)
75          {
76              handleClick(1, 3);
77          }
78
79          private void label21_Click(object sender, EventArgs e)
80          {
81              handleClick(2, 1);
82          }
83
```

```
84          private void label22_Click(object sender, EventArgs e)
85          {
86              handleClick(2, 2);
87          }
88
89          private void label23_Click(object sender, EventArgs e)
90          {
91              handleClick(2, 3);
92          }
93
94          private void label31_Click(object sender, EventArgs e)
95          {
96              handleClick(3, 1);
97          }
98
99          private void label32_Click(object sender, EventArgs e)
100         {
101             handleClick(3, 2);
102         }
103
104         private void label33_Click(object sender, EventArgs e)
105         {
106             handleClick(3, 3);
107         }
108
109         private void TicTacToe_Load(object sender, ...
                EventArgs e)
110         {
111             resetBoard();
112         }
113
```

```
114         private void generateMove_Click(object sender, ...
                EventArgs e)
115         {
116             int[] chosenMove = ...
                    controlModule.returnResult(board);
117             if (chosenMove[0] == -1)
118                 // invalid move
119                 MessageBox.Show("No valid moves found");
120             else
121             {
122                 board[chosenMove[0], chosenMove[1]] = 'O';
123                 syncBoard();
124             }
125         }
126
127     }
128 }
```

ControlModule.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace TicTacToe
7  {
8      class ControlModule
9      {
10         private HeuristicModule heuristicModule;
11
12         public ControlModule()
13             // constructor
14         {
15             heuristicModule = new HeuristicModule();
16         }
17
18         //———————————————————
19
20         public int[] returnResult(char[,] contextState)
21             // main interaction function
22             // returns a string representing the calculated ...
                   action choice
23             // given the context state represented by the ...
                   argument
24             // in this case, returns a 2—element array ...
                   representing the chosen move
25         {
```

```
26              // offload the "heavy lifting" to the Heuristic ...
                   Module
27
28              return heuristicModule.performSearch(contextState);
29          }
30      }
31  }
```

HeuristicModule.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace TicTacToe
7  {
8      class HeuristicModule
9      {
10         private ConstraintModule constraintModule;
11         private EvaluationModule evaluationModule;
12
13         public HeuristicModule()
14             // constructor
15         {
16             constraintModule = new ConstraintModule();
17             evaluationModule = new EvaluationModule();
18         }
19
20         //----------------------
21
22         public int[] performSearch(char[,] contextState)
23             // main interaction function
24             // performs a search using the hybrid PSO/AIS ...
                   heuristic function
25             // to find the maximal-valued action choice
26         {
27             // for the sake of code readability (and due to ...
```

```
                limited problem size for Tic-Tac-Toe)
28              // a simple search of all possible moves is ...
                    done here
29
30              // main hybrid PSO / AIS code is included as a ...
                    separate appendix
31              // to the dissertation, in a more readable ...
                    Matlab format
32
33              int[] bestMove = new int[] {-1, -1};
34              double bestMoveScore = double.MinValue;
35
36              double[,] debugString = new double[3,3];
37
38              for(int row=0; row<3; row++)
39                  for (int col = 0; col < 3; col++)
40                  {
41                      int[] actionSelection = new int[] { ...
                            row, col };
42                      if ...
                            (constraintModule.isLegalActionSelection(actionSelectio
                            contextState))
43                      {
44                          double score = ...
                                evaluationModule.evaluationResult(actionSelection,
                                contextState);
45                          debugString[row, col] = score;
46                          if (score > bestMoveScore)
47                          {
48                              bestMoveScore = score;
49                              bestMove = ...
```

```
                                    (int[])actionSelection.Clone();
50                              }
51                          }
52                      }
53
54                  return bestMove;
55              }
56          }
57  }
```

ConstraintModule.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace TicTacToe
7  {
8      class ConstraintModule
9      {
10         public ConstraintModule()
11             // constructor
12         {
13
14         }
15
16         //————————————————————
17
18         public bool isLegalActionSelection(int[] action, ...
               char[,] contextState)
19             // returns true/false value indicating whether ...
                   the passed
20             // action selection is legal in the current ...
                   context state
21         {
22             // simplest constraint checking for ...
                   Tic-Tac-Toe, merely
23             // see if the chosen space is currently occupied
24             if (contextState[action[0], action[1]] == ' ')
```

```
25                  return true;
26
27              return false;
28          }
29      }
30  }
```

EvaluationModule.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5
6  namespace TicTacToe
7  {
8      class EvaluationModule
9      {
10         private const int SIMULATION_LIMIT = 10000;
11         public Random rng;
12
13         public EvaluationModule()
14             // constructor
15         {
16             rng = new Random();
17         }
18
19         //-----------------------------------------------------------------
20
21         public double evaluationResult(int[] action, ...
               char[,] contextState)
22             // main interaction function
23             // returns the evaluated "score" of the action
24             // given the current context state
25         {
26             // demonstrates the flexibility of the ...
                   Evaluation Module design:
```

```
27              // a "helper" algorithm is used here to ...
                   evaluate action choices
28              // based on the Monte-Carlo simulation ...
                   algorithms described
29              // in the dissertation (see Wang and Gelly, 2007)
30              char[,] originalState = ...
                    (char[,])contextState.Clone();
31              originalState[action[0], action[1]] = 'O';
32
33
34              int wins = 0;
35              int games = 0;
36              for (int sims = 0; sims < SIMULATION_LIMIT; sims++)
37              {
38                  char[,] gameState = ...
                        (char[,])originalState.Clone();
39                  char result = playOutGame(gameState);
40                  if (result == 'O')
41                      wins++;
42                  games++;
43              }
44              return wins / (double)games;
45          }
46
47          //-------------------------------------------------------------------
48
49          public char playOutGame(char[,] state)
50              // using Monte-Carlo randomization, plays out ...
                   the Tic-Tac-Toe game
51              // returns winner, 'X' or 'O'
52
```

```
53              // once again, this is a simplification of the ...
                   actual UCT algorithm
54              // for the sake of readability
55              // this version merely plays out games ...
                   randomly; the actual UCT algorithm
56              // constructs a dynamic tree to track games and ...
                   choose future self and
57              // opponent moves more intelligently for a more ...
                   realistic simulation
58          {
59              bool myTurn=false;
60
61              List<int[]> legalMoves = new List<int[]>();
62              // find current legal moves
63              for (int row = 0; row < 3; row++)
64                  for (int col = 0; col < 3; col++)
65                      if (state[row, col] == ' ')
66                          legalMoves.Add(new int[] { row, col });
67
68              while (legalMoves.Count > 0)
69              {
70                  // first use the heuristic plugins to search
71                  // for an appropriate move given the ...
                       heuristics used
72                  int[] move = heuristicPlugin1(state, myTurn);
73
74                  if (move[0] == -1)
75                      move = heuristicPlugin2(state, myTurn);
76
77                  // if no heuristic move, randomly choose a ...
                       legal move
```

```
78                  if(move[0]==-1)
79                      move = legalMoves[rng.Next(0, ...
                            legalMoves.Count)];
80
81                  // play move
82                  if (myTurn)
83                      state[move[0], move[1]] = 'O';
84                  else
85                      state[move[0], move[1]] = 'X';
86
87                  // check if game is over
88                  if (isWinner('O', state))
89                      return 'O';
90                  if (isWinner('X', state))
91                      return 'X';
92
93                  // find current legal moves
94                  legalMoves.Clear();
95                  for (int row = 0; row < 3; row++)
96                      for (int col = 0; col < 3; col++)
97                          if (state[row, col] == ' ')
98                              legalMoves.Add(new int[] { row, ...
                                    col });
99
100                 myTurn = !myTurn;
101             }
102
103             // no one won, "cat's game"
104             return ' ';
105         }
106
```

```
107         //-------------------------------------------------------------------------
108
109         private int[, ,] lines = new int[8, 3, 2] { { { ...
                {0,0}, {0,1}, {0,2} },
110                                                   { {1,0}, ...
                                                        {1,1}, ...
                                                        {1,2} },
111                                                   { {2,0}, ...
                                                        {2,1}, ...
                                                        {2,2} },
112                                                   { {0,0}, ...
                                                        {1,0}, ...
                                                        {2,0} },
113                                                   { {0,1}, ...
                                                        {1,1}, ...
                                                        {2,1} },
114                                                   { {0,2}, ...
                                                        {1,2}, ...
                                                        {2,2} },
115                                                   { {0,0}, ...
                                                        {1,1}, ...
                                                        {2,2} },
116                                                   { {2,0}, ...
                                                        {1,1}, ...
                                                        {0,2} ...
                                                        } };
117
118         private int[] heuristicPlugin1(char[,] board, bool ...
                myTurn)
119             // searches for immediate wins available
120         {
```

```
121             char me = 'X';
122             if (myTurn)
123                 me = 'O';
124             for (int i = 0; i < 8; i++)
125             {
126                 int score = 0;
127                 int[] win = { -1, -1 };
128                 for (int j = 0; j < 3; j++)
129                     if (board[lines[i, j, 0], lines[i, j, ...
                            1]] == me)
130                         score += 5;
131                     else if (board[lines[i, j, 0], lines[i, ...
                            j, 1]] == ' ')
132                     {
133                         score -= 1;
134                         win[0] = lines[i, j, 0];
135                         win[1] = lines[i, j, 1];
136                     }
137                 if (score == 9)
138                     // can win
139                     return win;
140             }
141             return new int[] { -1, -1 };
142         }
143 
144         //-------------------------------------------------------------------------
145 
146         private int[] heuristicPlugin2(char[,] board, bool ...
                myTurn)
147         // searches for immediate blocks available
148         {
```

```
149             char opponent = 'X';
150             if (!myTurn)
151                 opponent = 'O';
152             for (int i = 0; i < 8; i++)
153             {
154                 int score = 0;
155                 int[] block = { -1, -1 };
156                 for (int j = 0; j < 3; j++)
157                     if (board[lines[i, j, 0], lines[i, j, ...
                            1]] == opponent)
158                         score += 5;
159                     else if (board[lines[i, j, 0], lines[i, ...
                            j, 1]] == ' ')
160                     {
161                         score -= 1;
162                         block[0] = lines[i, j, 0];
163                         block[1] = lines[i, j, 1];
164                     }
165                 if (score == 9)
166                     // can block
167                     return block;
168             }
169             return new int[] { -1, -1 };
170         }
171 
172         //-------------------------------------------------------------------------
173 
174         private bool isWinner(char player, char[,] board)
175         {
176             // brute force check to see if the player has won
177             for(int i=0; i<8; i++)
```

```
178                 if (board[lines[i, 0, 0], lines[i, 0, 1]] ...
                        == player &&
179                     board[lines[i, 1, 0], lines[i, 1, 1]] ...
                            == player &&
180                     board[lines[i, 2, 0], lines[i, 2, 1]] ...
                            == player)
181                     return true;
182
183             return false;
184         }
185     }
186 }
```

Bibliography

[Afshinmanesh et al., 2005] Afshinmanesh, F., Marandi, A., and Rahimi-Kian, A. (2005). A novel binary particle swarm optimization method using artificial immune system. *Computer as a Tool*, pages 217–220.

[Ahuja et al., 2007] Ahuja, R., Kumar, A., Jha, K., and Orlin, J. (2007). Exact and heuristic algorithms for the weapon-target assignment problem. *Operations Research*, 6:1136–1146.

[Aickelin and Cayzer, 2002] Aickelin, U. and Cayzer, S. (2002). The danger theory and its application to ais. *Proceedings of the First International Conference on Artificial Immune Systems*, pages 141–148.

[Aickelin and Dasgupta, 2005] Aickelin, U. and Dasgupta, D. (2005). Artificial immune systems. *Search methodologies introductory tutorials in optimization and decision support techniques*, pages 375–399.

[Angeline and Pollack, 1993] Angeline, P. J. and Pollack, J. B. (1993). Coevolving high-level representations. *Artificial Life III*.

[Auer et al., 2002] Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235–256.

[Auer et al., 1995] Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. E. (1995). Gambling in a rigged casino: The adversarial multi-armed bandit problem. *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pages 322–331.

[Ayara et al., 2002] Ayara, M., Timmis, J., de Lemos, R., de Castro, L., and Duncan, R. (2002). Negative selection: How to generate detectors. *Proceedings of the First International Conference on Artificial Immune Systems*, pages 89–98.

[Bertsekas, 1999] Bertsekas, D. P. (1999). *Nonlinear Programming*. Athena Scientific, 2nd edition.

[Coello and Cortes, 2005] Coello, C. A. C. and Cortes, N. C. (2005). Solving multiobjective optimization problems using an artificial immune system. *Genetic Programming and Evolvable Machines*, 2:163–190.

[Cohen et al., 2006] Cohen, R., Katzir, L., and Raz, D. (2006). An efficient approximation for the generalized assignment problem. *Information Processing Letters*, 100:162–166.

[Da Cost et al., 2008] Da Cost, L., Fialho, A., Schoenauer, M., and Sebag, M. (2008). Adaptive operator selection with dynamic multi-armed bandits. *Proceedings GECOO '08*.

[Dalvi et al., 2004] Dalvi, N., Domingos, P., Mausam, Sanghai, S., and Verma, D. (2004). Adversarial classification. *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[De Castro and Timmis, 2002] De Castro, L. N. and Timmis, J. (2002). *Artificial Immune Systems: A New Computational Intelligence Approach*. Springer-Verlag.

[de Castro and Von Zuben, 1999] de Castro, L. N. and Von Zuben, F. J. (1999). Artificial immune systems: Part ibasic theory and applications.

Technical Report DCA-RT 01/99, School of Computing and Electrical Engineering, State University of Campinas.

[Dhaeseleer et al., 1996] Dhaeseleer, P., Forrest, S., and Helman, P. (1996). An immunological approach to change detection: Algorithms, analysis and implications. *IEEE Symposium on Security and Privacy*.

[Dietterich, 2000] Dietterich, T. G. (2000). Ensemble methods in machine learning. *Lecture Notes in Computer Science*, 1857:1–15.

[Eberhart and Shi, 2001] Eberhart, R. and Shi, Y. (2001). Particle swarm optimization: Developments, applications and resources. *Proc. IEEE Int. Conf. Evolutionary Computation*, 1:81–86.

[Farmer et al., 1986] Farmer, J. D., Packard, N. H., and Perelson, A. S. (1986). The immune system, adaptation, and machine learning. *Evolution, games and learning*, pages 187–204.

[Fogel, 1993] Fogel, D. (1993). Using evolutionary programing to create neural networks that are capable of playing tic-tac-toe. *IEEE International Conference on Neural Networks*, 2:875–880.

[Forrest et al., 1994] Forrest, S., Perelson, A. S., Allen, L., and Cherukuri, R. (1994). Self-nonself discrimination in a computer. *IEEE Symposium on Research in Security and Privacy*.

[Garey and Johnson, 1979] Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.

[Garrett, 2005] Garrett, S. (2005). How do we evaluate artificial immune systems? *Evolutionary Computation*, 13:145–178.

[Gonzalez and Dasgupta, 2003] Gonzalez, F. and Dasgupta, D. (2003). Anomaly detection using real-valued negative selection. *Genetic Programming and Evolvable Machines 4*, pages 383–403.

[Gonzalez et al., 2002] Gonzalez, F., Dasgupta, D., and Kozma, R. (2002). Combining negative selection and classification techniques for anomaly detection. *Proc. of the 2002 Congress on Evolutionary Computation*, pages 705–710.

[Goodman et al., 2002] Goodman, D., Boggess, L., and Watkins, A. (2002). Artificial immune system classification of multiple-class problems. *Intelligent Engineering Systems Through Artificial Neural Networks*, 12.

[Hofmeyr and Forrest, 2000] Hofmeyr, S. A. and Forrest, S. (2000). Architecture for an artificial immune system. *Evolutionary Computation*.

[Hu and Eberhart, 2002] Hu, X. and Eberhart, R. (2002). Solving constrained nonlinear optimization problems with particle swarm optimization. *Proceedings of the Sixth World Multiconference on Systemics*, pages 203–206.

[Huaiping et al., 2006] Huaiping, C., Jingxu, L., Yingwu, C., and Wang, H. (2006). Survey of the research on dynamic weapon-target assignment problem. *Journal of Systems Engineering and Electronics*, pages 559–565.

[Juang, 2004] Juang, C. F. (2004). A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. *IEEE Transactions on Systems, Man, and Cybernetics*, 34:997–1006.

[Kennedy and Eberhart, 1995] Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. *Proc. IEEE Int. Conf. Neural Networks*, 4:1942–1948.

[Kennedy et al., 2001] Kennedy, J., Eberhart, R., and Shi, Y. (2001). *Swarm Intelligence*. Academic Press.

[Kleinberg et al., 2008] Kleinberg, R., Slivkins, A., and Upfal, E. (2008). Multi-armed bandits in metric spaces. *Proceedings of the 40th ACM Symposium on Theory of Computing*.

[Kountanis and Williams, 1994] Kountanis, D. and Williams, K. (1994). The target optimization problem has a polynomial solution. *Congressus numerantium*, pages 80–86.

[Kuhn, 1955] Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, pages 83–98.

[Mange et al., 2011] Mange, J., Daniszewski, D., and Dunn, A. (2011). Artificial immune systems for diagnostic classification problems. *Proceedings of the 22nd International Workshop on Principles of Diagnosis*.

[Mange and Dunn, 2010] Mange, J. and Dunn, A. (2010). Design of diagnostic algorithms to perform prognostics of an electrical system. *Proceedings of the 21st International Workshop on the Principles of Diagnostics*.

[Mange and Kountanis, 2012] Mange, J. and Kountanis, D. (2012). Non-linear programming approach to simulation of the general adversarial agents problem. *Proceedings of the 2012 Military Communications Conference*.

[Manshaei et al., 2011] Manshaei, M., Zhu, Q., Alpcan, T., Basar, T., and Hubaux, J.-P. (2011). Game theory meets network security and privacy. *EPFL Technical Report*.

[Matheus and Rendell, 1989] Matheus, C. and Rendell, L. (1989). Constructive induction on decision trees. *International Joint Conference on Artificial Intelligence*, pages 645–650.

[Pandey et al., 2007] Pandey, S., Chakrabarti, D., and Agarwal, D. (2007). Multi-armed bandit problems with dependent arms. *Proceedings of the 24th International Conference on Machine Learning*.

[Pedersen, 2010] Pedersen, M. E. H. (2010). Good parameters for particle swarm optimization. *Technical Report, Hvass Laboratories*.

[Pregenzer et al., 1996] Pregenzer, M., Pfurtscheller, G., and Flotzinger, D. (1996). Automated feature selection with a distinction sensitive learning vector quantizer. *Neurocomputer*, 11.

[Premalatha and Natarajan, 2009] Premalatha, K. and Natarajan, A. M. (2009). Hybrid pso and ga for global maximization. *International Journal Open Problems Compt. Math.*, 2:597–608.

[Pugh and Martinoli, 2009] Pugh, J. and Martinoli, A. (2009). Distributed scalable multi-robot learning using particle swarm optimization. *Swarm Intelligence*, 3:203–222.

[Radlinski et al., 2008] Radlinski, F., Kleinberg, R., and Joachims, T. (2008). Learning diverse rankings with multi-armed bandits. *Workshop on Machine Learning for Web Search*.

[Rasmussen, 2004] Rasmussen, C. E. (2004). Gaussian processes in machine learning. *Lecture Notes in Computer Science*, 3176:63–71.

[Rehak et al., 2005] Rehak, M., Pechoucek, M., and Tozicka, J. (2005). Adversarial behavior in multi-agent systems. *Multi-Agent Systems and Applications IV*, pages 470–479.

[Riley and Veloso, 2000] Riley, P. and Veloso, M. (2000). On behavior classification in adversarial environments. *Distributed Autonomous Robotic Systems*, 4:371–380.

[Robbins, 1952] Robbins, H. (1952). Some aspects of the sequential design of experiments. *Bulletin American Mathematical Society*, 55:527–535.

[Robinson et al., 2002] Robinson, J., Sinton, S., and Rahmat-Samii, Y. (2002). Particle swarm, genetic algorithm, and their hybrids: Optimization of a profiled corrugated horn antenna. *IEEE Antennas Propagation Society Inter. Symp. Dig.*, pages 314–317.

[Russell and Norvig, 2010] Russell, S. J. and Norvig, P. (2010). *Artificial Intelligence: A Modern Approach*. Pearson.

[Sailer et al., 2007] Sailer, F., Buro, M., and Lanctot, M. (2007). Adversarial planning through strategy simulation. *Computational Intelligence and Games*, pages 80–87.

[Sebastiani, 2002] Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM Computing Surveys*, 34:1–47.

[Shi and Eberhart, 1999] Shi, Y. and Eberhart, R. C. (1999). Empirical study of particle swarm optimization. *Proc. IEEE Int. Congr. Evolutionary Computation*, 3:101–106.

[Shi and Eberhart, 1998] Shi, Y. H. and Eberhart, R. C. (1998). A modified particle swarm optimizer. *IEEE International Conference on Evolutionary Computation*.

[University of California, 2012] University of California, I. (2012). Uc irvine machine learning repository.

[Vesterstrom and Thomsen, 2004] Vesterstrom, J. and Thomsen, R. (2004). A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. *Evolutionary Computation*, 2:1980–1987.

[Villacorta and Pelta, 2010] Villacorta, P. J. and Pelta, D. A. (2010). Evolutionary design and statistical assessment of strategies in an adversarial domain. *Evolutionary Computation*, pages 1–7.

[Wang, 2006] Wang, Q. (2006). A hybrid optimization algorithm based on clonal selection principle and particle swarm intelligence. *Intelligent Systems Design and Applications*, 2:975–979.

[Wang and Gelly, 2007] Wang, Y. and Gelly, S. (2007). Modifications of uct and sequence-like simulations for monte-carlo go. *IEEE Symposium on Computational Intelligence and Games*, pages 175–182.

[Wierzchon, 1999] Wierzchon, S. T. (1999). Generating antibody strings in an artificial immune system. *Technical Report ICS PAS Report No. 892*, Institute of Computer Science, Polish Academy of Sciences.

[Williamson, 2002] Williamson, M. M. (2002). Biologically-inspired approaches to computer security. *Technical Report HPL-2002-131*, HP Labs.

[Xie et al., 2002] Xie, X. F., J., Z. W., and Yang, Z. L. (2002). Adaptive particle swarm optimization on individual level. *Int. Conf. on Signal Processing*, pages 1215–1218.

[Yap et al., 2011] Yap, D. F. W., Koh, S. P., Tiong, S. K., and Prajindra, S. K. (2011). Particle swarm based artificial immune system for multimodal function optimization and engineering application problem. *Trends in Applied Sciences Research*, 6:282–293.