The Dissertation Committee for Erkin Bahceci

certifies that this is the approved version of the following dissertation:

Competitive Multi-Agent Search

Committee:

Risto Miikkulainen, Supervisor

Bruce W. Porter

Gordon S. Novak Jr.

Vladimir Lifschitz

Riitta Katila

Competitive Multi-Agent Search

by

Erkin Bahceci, B.S., M.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

December 2014

To my family

Acknowledgments

I would like to thank my research supervisor, Risto Miikkulainen, for his invaluable research advice and support throughout my time at UT. He provided me encouragement and new ideas to try in every meeting. This dissertation would not be possible without his guidance and patience.

I would also like to thank my committee, Bruce W. Porter, Gordon S. Novak Jr., Vladimir Lifschitz, and Riitta Katila for their insightful comments and constructive criticism.

I would like to especially thank Thomas N. Wisdom and Robert L. Goldstone at Indiana University for providing data and support for the human game domain, which was crucial for the second half of this dissertation.

I have benefitted greatly from the discussions with current and former members of the Neural Networks Research Group at UT, who helped shape my ideas and present them more effectively.

I am also grateful to my friends İpek Neşe Şener, Sezgin Küçükçoban, and Onur Soysal for their help and support.

Additionally, I would like to thank Hsin Tsao, Vivek Haldar, and my teammates at Google for their support and understanding.

Finally, I would like to thank my mom and brother for their support and patience.

This research was supported in part by NSF under grants SBE-0914796, IIS-0915038, and DBI-0939454.

ERKIN BAHCECI

The University of Texas at Austin

December 2014

Competitive Multi-Agent Search

Publication No. _____

Erkin Bahceci, Ph.D.
The University of Texas at Austin, 2014

Supervisor: Risto Miikkulainen

While evolutionary computation is well suited for automatic discovery in engineering, it can also be used to gain insight into how humans and organizations could perform more effectively. Using a real-world problem of innovation search in organizations as the motivating example, this dissertation formalizes human creative problem solving as competitive multi-agent search. It differs from existing single-agent and team-search problems in that the agents interact through knowledge of other agents' searches and through the dynamic changes in the search landscape caused by these searches. The main hypothesis is that evolutionary computation can be used to discover effective strategies for competitive multi-agent search. This hypothesis is verified in experiments using an abstract domain based on the NK model, i.e. partially correlated and tunably rugged fitness landscapes,

and a concrete domain in the form of a social innovation game. In both domains, different specialized strategies are evolved for each different competitive environment, and also strategies that generalize across environments. Strategies evolved in the abstract domain are more effective and more complex than hand-designed strategies and one based on traditional tree search. Using a novel spherical visualization of the fitness landscapes of the abstract domain, insight is gained about how successful strategies work, e.g. by tracking positive changes in the landscape. In the concrete game domain, human players were modeled using backpropagation, and used as opponents to create environments for evolution. Evolved strategies scored significantly higher than the human models by using a different proportion of actions, providing insights into how performance could be improved in social innovation domains. The work thus provides a possible framework for studying various human creative activities as competitive multi-agent search in the future.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Competition is common both in nature and in human societies. Individuals and groups compete for territory, market share, and resources when they operate in the same domain, such as technological innovation, scientific discovery, and art and design. For instance, companies in high-technology industries compete when they are trying to develop the best products. They utilize various strategies such as exploration of new product ideas, exploitation of their current products and knowledge, imitating other companies' products, and sharing or hiding their knowledge. This problem-solving activity can be seen as a new kind of search process, one where multiple agents compete in searching for solutions in the same space. Once this process is formalized, it will be possible to study it systematically, identifying strategies that work best in different environments. With such knowledge, companies should be able to innovate better, and entire industries may become more productive. Laying such a groundwork for *Competitive Multi-Agent Search* (CMAS) is the goal of this dissertation.

1.1 Motivation

Single-agent search has been a successful approach to formalizing many types of human problem solving. For instance, it has been used in searching for solutions in puzzles, game playing, and planning. However, in the real world, agents often do not search for solutions in isolation. There are many agents who simultaneously look for solutions in the same search space, and therefore compete in finding them. They search in a non-stationary environment because both the knowledge of opponents' searches and the fitness landscape itself changes as the search progresses. As a result, agents need to be reactive in order to respond to those changes. This requirement makes both the problem and the solution method different from standard search; the traditional formalizations of single-agent search do not adequately describe competitive multi-agent search.

A clear example of such competitive multi-agent search, and the main motivation for this dissertation, is *innovation search*, a problem-solving activity where companies search for new products by recombining and manipulating existing knowledge (Katila, 2002). Unlike in single-agent search, the agents' behavior is affected by other agents' search in two ways: knowledge about the solutions that the other agents have found so far, and the dynamic changes to the search landscape that result from their searches (such as boosting the value of an emerging area or reducing the value of a crowded area). Neither of these mechanisms are taken into account in single-agent search, but they are central to competitive multi-agent search, as will be shown in this dissertation.

Once a proper formalization for competitive multi-agent search has been found, it will be possible to determine how it can be done well. In terms of innovation search, methods that would allow companies to find the best ways to innovate in various circumstances would be beneficial to all companies, as well as to potential users of their products. It would be useful for them to find the most effective balance of search strategies, such as how much exploration vs. exploitation to do, and how much and how quickly to share knowledge. For instance, too much exploration (e.g. looking for completely new product areas) would be

too time-consuming and costly. On the other hand, too much exploitation (e.g. creating products that are iterations of the company's existing products) would make it less likely for the company to generate innovative technologies (Katila and Chen, 2008). Therefore, not only would such insights reduce costs of product development, but it would also help bring more innovative products to market and do so more frequently. Furthermore, fitting CMAS and such insights to archival data of patents and products in a particular industry can help make such data more useful for that industry.

Benefits of effective competitive multi-agent search are not limited to companies searching for products, but apply to many other human creative problem-solving activities. For example, finding effective ways to balance exploration of new scientific ideas and iterating on earlier discoveries may help researchers be more productive, and the public will benefit from scientific advancements sooner. Similarly for designers and artists, finding out how much and how frequently to exchange ideas may be important. Sharing too much knowledge or doing it too frequently may cause designers and artists to influence each other too much, which might reduce the diversity of produced ideas. However, keeping all ideas private would prevent productive collaborations and bouncing off of ideas, which could potentially improve them. Moreover, CMAS could be a useful way to model games where the goal is to put together a team of players that contribute to the score, such as fantasy sports games and economic strategy games. Identifying the effects on the score of different levels of player changes and imitation of other game participants would help create better artificial agents for such games, improving player experience and perhaps extending them to training applications. Thus, understanding what makes CMAS effective could be helpful in several domains of creative problem solving.

1.2 Challenge

Formalizing and optimizing CMAS is challenging for four reasons. First, it is not sufficient to keep track of just the agent's own search progress; it is necessary to take into account

what the other agents are doing as well. For instance, in innovation search, patent filings and product announcements make such information known to other agents. Therefore, agents' actions depend on other agents' actions, making CMAS more complex than other search settings studied so far.

Second, the fitness landscape in CMAS is dynamic: When many agents converge in the same region of the fitness landscape, the landscape changes. Such changes take two forms: *boosting*, which is an increase in search points' fitness due to that region becoming more attractive (e.g. expanding demand in new markets in innovation search), and *crowding*, which is a subsequent decrease in the fitness (e.g. saturation of existing markets). Such dynamic changes mean that the agent must be continuously searching for new solutions.

Third, CMAS environments can vary in several dimensions. They can have varying sizes of search spaces and densities of agents. Different environments can have sets of opponents with a wide range of opposite strategies, from those that only exploit the points they find to those that do only exploration in the search space. It is unlikely that a single strategy works well in all environments. Instead, it is necessary to characterize different environments systematically and determining the best strategies for each of them separately.

Fourth, due to all these factors, it is difficult to manually create optimal strategies. As a result, methods that automatically discover optimal agent strategies for CMAS are needed. Tackling these challenges, this dissertation presents a formalization of CMAS and methods to characterize, model, and optimize agent strategies through evolutionary discovery.

1.3 Approach

The approach taken in this dissertation is to develop a formalization of multiple agents searching for solutions simultaneously in the same search space, while addressing the challenges stated in the previous section. First, the formalization for CMAS will allow agents to make their knowledge (i.e. points they have discovered) known to other agents, through

a *public memory*. They will also be able to store their knowledge without disclosing it to others through a *private memory*.

Second, the fitness landscape will change when agents visit points. These changes will be called *flocking* effects in this dissertation, and both boosting and crowding will be modeled. Whenever agents visit a point in the search space, the region around that point will initially interact by a multiplicative flocking factor that is greater than 1.0. After a while this factor becomes less than 1.0, and the region decreases linearly on subsequent agent visits.

Third, a standard set of environments will be used as the experimental setup for comparing different strategies' performance. The various environments will include those with large and small search spaces. Opponent agents in the environments will vary in their use of private and public memory. Moreover, they will be doing either exploitation or exploration using the best point in that memory as the starting point.

Fourth, in order to determine which strategies work well in specific environments and as general strategies, they will be encoded as neural networks, and optimized using NEAT, a popular neuroevolution method (Stanley and Miikkulainen, 2002). They will be optimized in homogeneous and heterogeneous environments with the different opponents above. Furthermore, multiple homogeneous environments will be used to evolve general strategies that can perform well in multiple environments.

Two domains will be used in this dissertation: an abstract domain and a concrete domain. In the abstract domain, competing agents will be simulated to search for high-fitness points on NK fitness landscapes (Kauffman, 1993), which will be modified to incorporate landscape changes. This domain will allow setting up different environments systematically, and investigating and understanding the results in detail.

The insights from the abstract domain will then be verified in a concrete human activity domain. An existing laboratory setting from Wisdom et al. (2013) that matches CMAS well will be utilized: a social innovation game that has thematic elements of fantasy

sports leagues. Using gameplay data of human subjects, a model for each subject will be trained via supervised learning. These human models will then be used as opponents during evolution of optimal strategies in the concrete domain.

The concrete domain will demonstrate that the CMAS approach works in the real world; in that sense it is general, and can likely be applied to characterize and optimize behavior in many other domains of human creative activity.

1.4 Outline

This dissertation is organized into four parts: background (Chapters 1-2), characterization and optimization of a competitive multi-agent search problem in an abstract domain (Chapters 3-5), in a concrete domain (Chapters 6-8), and discussion and future work (Chapters 9-10).

Chapter 2 discusses the motivation from the perspective of organizational theory, reviews single-agent and team-based search methods, agent-based modeling approaches, and the evolutionary algorithm employed.

Chapter 3 describes the first of the two domains utilized in this dissertation, i.e. a tunably-rugged NK fitness landscape. The chapter presents a visualization for the fitness landscape, details the agents of the domain, how they interact and search, and how their strategies are implemented.

Chapter 4 provides a characterization of the abstract domain via experiments on the effects of search diversity, choice of search method, exploration focus, environment size, and opponents' strategies. Detailed experimental results are provided, specifying what kind of strategies work in virtual environments.

Chapter 5 presents an encoding for agent strategies to facilitate evolution. It describes experiments that evolve strategies tailored for specific environments, as well as strategies that are intended to be more general-purpose. The results demonstrate that evolution can discover customized strategies for specific environments, as well as general ones

that can work well in multiple environments.

Chapter 6 describes the second of the two domains, i.e. a social innovation game. It summarizes the results of Wisdom et al. (2013) on that domain, and describes the gameplay data collected in that study.

Chapter 7 details how that data was interpreted as training data for supervised learning of a model for each human subject, in the form of a two-tiered neural network. Several distance objectives are described, which are used to evaluate how close those models are to the corresponding human subjects. Different modeling approaches are compared using multiple objectives to measure distance to human subjects. Two-tiered neural network models are shown to be better in imitation objectives than one-tiered neural network models, and better in both imitation and score objectives than simpler models. Effects of the length of training are also investigated, finding that performance in a subset of the objectives improves with increased training, whereas others get worse.

Chapter 8 utilizes those human subject models as opponents against which to optimize strategies in the form of a combined two-tiered neural network. Strategies are successfully evolved for specific homogeneous and complex environments, and they perform better in those environments than strategies evolved in other environments. General strategies are also evolved using multiple homogeneous environments, and they perform better overall than those evolved in a single environment, but worse than the ones evolved for each particular environment.

Chapter 9 discusses the similarities and differences between the abstract and concrete domains, and presents ideas for future work. These include extending the representation of agent strategies and the optimization method, applications for human subject models, theoretical characterization of CMAS, and analysis of real-world archival data.

Chapter 10 summarizes the contributions, and concludes the dissertation.

Chapter 2

Background and Related Work

This dissertation is motivated by the real-world problem of innovation search in organizations. The formalization of this problem, CMAS, builds on single-agent and team-search methods, but extends them with competitive and cooperative dynamic interactions between agents. It also builds on agent-based modeling, but applies that general approach to doing innovation search in a landscape that changes dynamically because of agent actions. The main contribution is to show that evolutionary computation is a good way to discover effective solution strategies for CMAS. While many different evolutionary approaches could be used, the particular one tested in this dissertation is based on NeuroEvolution of Augmenting Topologies (NEAT; Stanley and Miikkulainen, 2002) as one representative approach. With NEAT, strategies can be represented naturally with neural networks whose complexity is matched with the task.

2.1 Organizational Theory

Even though the main focus of this dissertation is on characterization and evolutionary optimization of CMAS strategies, it is useful to review the motivation from the perspective of a real-world example of CMAS, that of organizational theory in management science.

The example serves to make the issues and the motivation concrete.

Search (i.e. institutional problem solving) in the organizational theory literature is typically thought to take place in a *knowledge space*, conceptualized as a landscape. In innovation search, for example, firms generate, recombine, and manipulate knowledge within a pool of technological possibilities (Levinthal and March, 1981), and such activity can be tracked, for example, by using patents (e.g. Katila, 2002). Such a search can be represented in an NK landscape (Section 3.1) where N corresponds to dimensions of knowledge and K determines how complex the relationships between them are (Katila et al., 2014; Levinthal, 1997; Gavetti and Levinthal, 2000). This work has led to several insights. One is that firms that search more frequently and further away from their current knowledge bases (i.e. *explore*) are more likely to succeed (Greve, 2003a; Katila, 2002). Another is that firms typically search in exactly the opposite way: too little and too close (i.e. *exploit*), and therefore need to find effective strategies to resist such local tendencies (Helfat, 1994).

Despite these insights, the focus of organizational search research has been relatively narrow. Prior efforts typically assessed a firm's innovation activities only relative to its own behavior, i.e. as *single-agent search*. Only recently, researchers have started to conceptualize search beyond its single-agent roots and to incorporate competition. There is emerging research on situations in which firms learn from their competitors and on why such learning is sometimes difficult (Greve and Taylor, 2000; Rivkin, 2000), as well as research on how competitors interact dynamically (Katila et al., 2008). However, to date these studies have been conceptual and statistical only; competition has not been integrated in any formal models of organizational search.

The first contribution of this dissertation is to do so, i.e. to create a formalization that can be used to study such competitive processes with computational techniques. It thus contributes to management science, but it also contributes to Artificial Intelligence (AI), by defining a new and interesting class of search problems relevant to the real world. Although the motivation comes from the specific example of organizational search, the

same formalization should be useful in understanding a range of creative problem-solving properties in human societies, such as scientific discovery, creativity in engineering, and art and design.

The second, main contribution of this dissertation is then to show that new computational techniques are useful in this new domain. The next section reviews traditional search techniques in artificial intelligence and evolutionary computation, pointing out why they are not a good fit with CMAS problems.

2.2 Search Algorithms

Traditionally, research on search algorithms has focused on two types of search methods: search performed by a single agent and search performed by a team of similar cooperating agents. Single-agent search methods, such as A* (Hart et al., 1968) and iterative-deepening A* (IDA*; Korf, 1985), have been used in well-defined search domains, including path finding and scheduling problems. Such methods are understood well theoretically and guarantee optimal solutions to a problem, but they are impractical to utilize if the search space of the problem is too large.

On the other hand, with team-search methods the individual team members search for peaks in a fitness landscape in parallel; every point in the search space has a certain height corresponding to its fitness value, and the knowledge of all agents is collected into a single pool. These search methods are appropriate in problems that are inherently large or not well-defined, such as antenna design (Lohn et al., 2004) and robot control (Valsalam et al., 2007). The theory of these methods is less well developed and they do not usually guarantee optimal solutions.

Team-search methods are typically inspired by various types of biological and natural systems, such as evolution (Mitchell, 1996), swarm behavior (Kennedy et al., 2001), water drop modeling (Hosseini, 2009), and gravitational particle interactions (Rashedi et al., 2009). For instance, Particle Swarm Optimization (Kennedy et al., 1995) and Ant Colony

Optimization (Dorigo and Stützle, 2004) are inspired by social swarm behaviors in nature: They make use of multiple agents that represent solutions to optimization problems.

Competition has been incorporated into single-agent search methods by extending them to two-player games (Pearl, 1984) and multi-agent adversarial games (Zuckerman and Felner, 2011). The search for the best move for a player proceeds by first considering all moves of that player and then a subset of the moves of the opponents, utilizing techniques such as alpha-beta pruning (Knuth and Moore, 1975; Korf, 1991). However, since such search methods rely on enumerating all possible moves of at least one player, they are not practical with a large number of moves.

Competitive elements exist in team search as well. For instance, in evolutionary search, population members compete to propagate their genes, although the population as a whole cooperates to produce a single good solution. Inter-population competition has been incorporated into evolutionary search in a coevolutionary arrangement, where different populations try to outdo each other in the task (Pollack et al., 1996; Juille and Pollack, 1996; Werfel et al., 2000; Stanley and Miikkulainen, 2004). However, there is no absolute fitness; a team is considered successful simply if it does better than the other team. Moreover, the teams do not alter the fitness landscape and therefore do not influence each other's search.

Therefore, neither single-agent nor team-based search is a good fit with CMAS problems. In CMAS, interactions among agents and between the agents and the environment need to be taken into account explicitly. Agent-based modeling provides a framework for doing that, as will be described next.

2.3 Agent-Based Modeling

Agent-based modeling has been used extensively in various fields including search and optimization in computer science (Dorigo and Stützle, 2004; Kennedy et al., 1995; Knight, 1993), as well as real-world social and economical interactions in political science (Axelrod, 1997) and economics (Epstein, 1999; Holland and Miller, 1991). The idea is to model

each agent explicitly, with the goal that global patterns emerge from this process. The agent-based approach is therefore an appropriate formalization for competitive multi-agent search.

In the multi-agent systems literature, there are many examples of domains where the agents compete to solve problems (Hoen et al., 2005). CMAS can be seen as a special case of such problems, characterized by four special properties: (1) Competitive multi-agent search is modeled as search for the same highest peaks in a common landscape. (2) The agents may not necessarily know about the other agents' searches, and may or may not inform them about their own searches. (3) Their search actions have an effect on the landscape that is visible to all agents. (4) The agents' search strategies are stochastic, representing the bounded rationality of real-world agents such as human decision makers and organizations (March and Simon, 1958).

While some of these properties have been addressed in prior research, together they define a new and interesting problem class. It shares with prior work the idea of agents and their interactions as the appropriate level of modeling. However, building on these specific properties, it may be possible to develop a specific formalization and approach for CMAS problems that makes it easier to understand and solve such problems. This is the goal of this dissertation.

The main hypothesis tested is that while it is possible to formulate search strategies by hand, and adapt traditional single-agent search strategies to CMAS, better strategies can be discovered automatically by evolutionary optimization. In order to verify this hypothesis, a particular approach is developed using evolution of neural networks with the NEAT method (Stanley and Miikkulainen, 2002). Representing CMAS strategies as neural networks that generate patterns, such as Compositional Pattern Producing Networks (CPPNs; Stanley, 2007) or Central Pattern Generators (CPGs; Chiel et al., 1999), is a potentially powerful approach, as will be described in detail in Section 5.1. While such networks could be evolved through different methods, NEAT has been previously applied to them exten-

sively, and will therefore be used as the default implementation in this dissertation as well.

2.4 NeuroEvolution of Augmenting Topologies (NEAT)

NeuroEvolution of Augmenting Topologies (NEAT; Stanley and Miikkulainen, 2002) is a method for evolving neural networks that adjusts both the topology and the weights as part of the learning process (for other such methods, see Harvey et al., 1997; Hutt and Warwick, 2003; Moriguchi and Honiden, 2012). It is based on three synergetic ideas (for details, see e.g. Stanley and Miikkulainen, 2002):

First, the initial population of neural networks consists of minimally connected individuals with no hidden nodes (i.e. nodes other than input and output nodes). The networks gradually become more complex through mutations that add nodes and connections. Only those additions to the topology that improve performance are kept, which helps find small solutions to the problem. Starting with minimal topology also speeds up learning since the number of connection weights to be optimized during evolution, i.e. the size of the search space for connection weights, is minimal (Stanley and Miikkulainen, 2002).

Second, crossover between individuals with different topologies is made possible by keeping an innovation number for each gene in the genome of a neural network. Innovation numbers are used to match genes that have similar historical origin. They are an abstraction of homology in biological evolution, which is the mechanism for aligning similar genes during crossover (Sigal and Alberts, 1972). Keeping these numbers for each gene circumvents the expensive task of matching topologies of networks for crossover (see (Stanley and Miikkulainen, 2002) for details on innovation numbers).

The third component of NEAT is that innovation in population members is protected by separating the population into species depending on similarity. When a structural mutation alters an individual considerably, the individual may not initially perform as well as the other population members. If this happens, that individual will not survive even though this mutation might have led to a better-performing individual after some optimiza-

tion. Speciation protects such individuals by creating a separate species for networks that are different from others, allowing them to be optimized within the species first. To prevent the whole population from being reduced to a single species, explicit fitness sharing (Goldberg and Richardson, 1987) is used. This principle means that the fitness within the species is shared among its members, dividing the fitness of each individual by the size of its species, preventing species from becoming too large.

NEAT has been shown to be successful in several open-ended design domains, such as vehicle control and collision warning (Kohl et al., 2006) and controlling video game agents (Stanley et al., 2005). Most importantly, it has been particularly effective in evolving CPPNs, i.e. networks that produce spatial patterns (Stanley, 2007). The approach developed in Chapters 4 and 5 will make use of this idea in an abstract NK-landscape domain: Agent strategies will be encoded as discrete 4D and 2D patterns, as will be described in Section 5.1. On the other hand, Chapters 7 and 8 focus on a concrete domain of a social innovation game, where it is not as practical to generate and store discrete, pattern-based strategies. Therefore, CPPNs with fine-grained state inputs will be used to represent agent strategies in those chapters.

2.5 Conclusion

This chapter provided background on organizational theory in management science, which was the motivating real-world example of CMAS, and on search algorithms, on which this dissertation builds, as well as on agent-based modeling, of which CMAS is a special case. Furthermore, the NEAT method of neuroevolution was summarized; this method will be used to optimize agent strategies in an abstract domain and a concrete domain in later chapters. The first of those two domains will be described next.

Chapter 3

Abstract Domain: *NK* Model

In CMAS, multiple agents search for the highest peaks in the same landscape simultaneously. The agents either share or hide their knowledge about the landscape from other agents, and their searches change the landscape dynamically. They select search actions (either exploit locally or explore globally) based on a strategy that is encoded as CPPNs, and evolved through the NEAT method. This section describes the landscapes, the agent models, the memory types and search actions they use, how the strategies are represented and generated from CPPNs, and how multiple agents are simulated.

3.1 Fitness Landscape

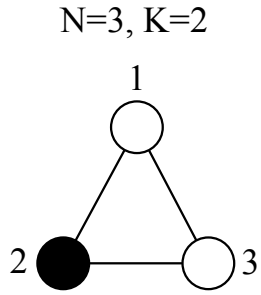
A popular way to evaluate search methods is to do it in an abstract NK fitness landscape (Kauffman, 1993). NK landscapes are N-dimensional hypercubes that assign fitness values to the points of the space such that the *ruggedness* of the landscape can be adjusted (using the K parameter). Such landscapes have been used extensively to model human problem solving, such as innovation search (Levinthal, 1997; Anderson, 1999; Gavetti and Levinthal, 2000). Compared to alternatives such as game theoretical models, the characteristics of the environment can be readily incorporated into an NK simulation, a large number of agents

can be included, and they can be boundedly rational, making it easier to draw relevant insights (Levinthal, 1997).

The K parameter specifies the level of interaction among the N dimensions and can take on values between 0 and $N - 1$. When K is 0, the fitness landscape is single-peaked and smooth, as in Figure 3.2 (a). One can go from the lowest-fitness point to the single peak by simply following the fitness gradient (i.e. separately flipping each bit that causes the fitness to increase). With a small K, the fitness landscape becomes a little rugged, but highest peaks are concentrated in a specific region. When K is $N - 1$, the landscape becomes fully random with many peaks distributed all over the space, as in Figure 3.2 (b).

More specifically, each point in the search space is encoded as a bit string of length N. The fitness of a point is calculated by taking the average of the fitness contributions of each bit in the bit string for that point. Each bit's contribution depends on the value of $K+1$ bits: that bit and the K bits that interact with that bit. As suggested by Kauffman (1993), for bit i the interacting bits are the K bits that follow it, i.e., bits $i + 1$, $i + 2$, ..., $i + K$ (mod N). These $K + 1$ bit values are used as a key to look up fitness-contribution values from a table, generated randomly from a uniform distribution. This table has N random values for each $(K + 1)$-bit key, of which there are 2^{K+1}. Figure 3.1 shows how the fitness is calculated for an NK fitness landscape with $N = 3$ and $K = 2$. In this case, the table consists of 3×8 fitness contribution values.

Dynamic fitness landscapes are modeled through *flocking*. That is, whenever an agent visits a point, the fitness of that point and those nearby change, depending on the *flocking intensity* and *flocking radius* parameters. The fitness of the area around a point defined by the flocking radius is multiplied by the flocking intensity. Two types of flocking are used. With *boosting*, flocking intensity is greater than 1.0 and the region rises, whereas with *crowding*, flocking intensity is less than 1.0 and the region sinks. There is no limit on the number of times a point can be visited. Therefore, when used in isolation, boosting and crowding will cause a point's fitness value to approach 1.0 and 0.0, respectively, with



Element			Fitness contribution			Overall fitness
1	2	3	F_1	F_2	F_3	$F = \frac{1}{N}\sum_{i=1}^{N} F_i$
0	0	0	0.6	0.7	0.2	0.50
0	0	1	0.8	0.3	0.8	0.63
0	1	0	0.7	0.4	0.7	0.60
0	1	1	0.4	0.6	0.7	0.57
1	0	0	0.3	0.8	0.2	0.43
1	0	1	0.5	0.9	0.8	0.73
1	1	0	0.5	0.7	0.8	0.67
1	1	1	0.5	0.6	0.5	0.53

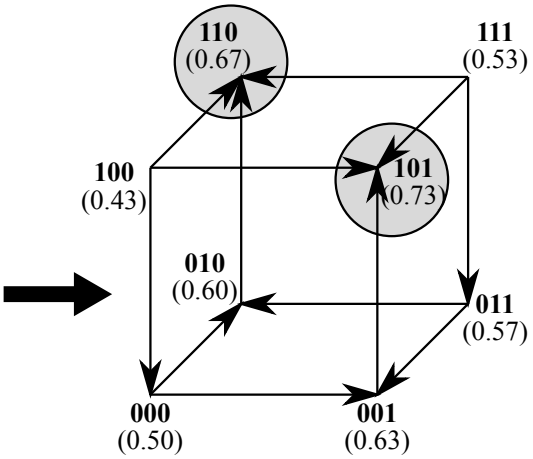
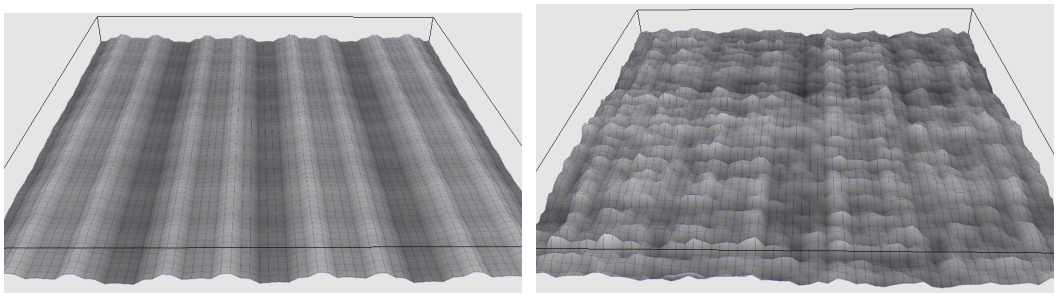


Figure 3.1: An example NK fitness landscape, where $N = 3$ is the number of dimensions or bits of each point and $K = 2$ is the number of other dimensions that interact with each dimension. For example, dimension 2 interacts with dimensions 1 and 3 (i.e. the two bits that follow it). To obtain the fitness of a point, e.g., 010, the fitness contribution values for its element values are averaged (i.e. the third row in the table). The arrows on the hypercube represent the direction of increasing fitness. Also shown on the hypercube are the peaks in the NK fitness landscape, indicated with shaded circles. The fitness values of all points constitute the complete fitness landscape. NK landscapes are useful because they are general and the difficulty (i.e. ruggedness) can be adjusted. Such abstract landscapes can be used as a platform to study search methods, as will be done in Chapters 3-5 in this dissertation.



(a) (b)

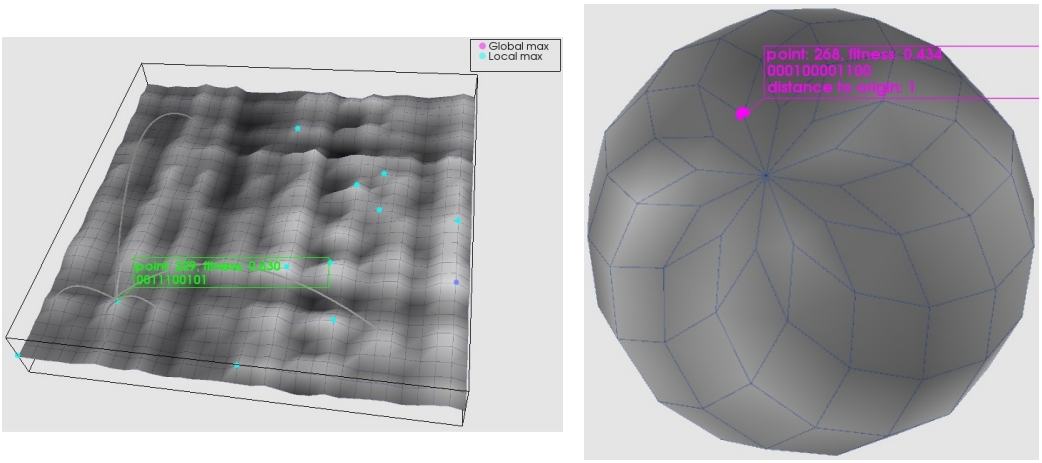
Figure 3.2: Rectangular visualization of two NK fitness landscapes with $N = 12$: (a) a smooth landscape (where $K = 0$) and (b) a rugged landscape (where $K = 11$).

every visit to that point. These changes make it possible to model the dynamics of fitness landscapes in applications such as innovation search, where boosting corresponds to expanding demand in new markets (such as tablet computers) and crowding to the saturation of existing markets (such as desktops; Katila et al., 2012).

3.2 Visualizing *NK* Fitness Landscapes

Although high-dimensional NK landscapes are useful in testing ideas about search and optimization in complex domains, it is not possible to visualize them accurately, and it is therefore difficult to develop an intuitive understanding of what happens in such spaces. In a typical visualization, two dimensions are chosen to be represented accurately along each axis, and the visualization is repeated for the different combinations of values for the other dimensions, as seen in Figure 3.2.

The main problem with such a visualization is that the continuity of the space is lost, i.e. nearby points can end up very far apart on the visualization, disallowing natural intuitions about space (Figure 3.3 (a)). Another issue with this kind of visualization is that it shows all 2^N points in the space; as N grows, this number becomes prohibitively large, making it impossible to visualize NK landscapes with sufficiently large N.



(a) (b)

Figure 3.3: (a) Rectangular vs. (b) spherical visualization of NK fitness landscapes. The rectangular visualization shows all points in the search space, but is not continuous, and the spherical one retains the sense of continuous space, but has less resolution further away from the poles.

A different, novel approach is taken in Figure 3.3 (b). In this *spherical* approach, the continuity of the space is preserved, and the space is represented with variable resolution. One point is chosen as the focus (e.g. 11111 in the five-dimensional case), and all of its neighbors in the original space (e.g. 01111, 10111, 11011, 11101, and 11110) are shown as its neighbors on the sphere, around a circle at distance 1. At distance 2, points with two bits away from the original are shown, by combining bit flips of the adjacent neighbors (to 00111, 10011, 11001, 11100, 01110), and so on until the complement (00000) of the focus point is reached at the other side of the sphere. In this manner, continuity of the original space is maintained in the grid that results on the sphere: nearby points in the grid are indeed neighbors in the original space, differing by one bit. The elevation and brightness of the spherical surface represents the fitness of the points on it: The higher the point and the lighter it is, the higher the fitness.

The resolution of this representation is full near the poles (i.e. at distance 1 and 4 in the five-dimensional case), and it decreases towards the equator (at distances 2 and 3, only

five of the 10 points are located on the grid). For visualization (as in Figures 4.2 and 5.5), other points of interest can be shown in the quadrilateral regions between the actual grid points, although their specific locations within those regions are undetermined. They are placed so that the distance between each point and the north pole point corresponds to the Hamming distance between them. Among the quadrilateral regions along the meridian at that distance, each point is in the one whose corners are closest to the point.

The main purpose of this visualization is to represent the local neighborhood of a specific point intuitively as a continuous space, with gradually less resolution towards the horizon. It thus gives a concrete snapshot of the current state of the search. The focus point can also be moved as the search progresses, resulting in a detailed track of the process. Such a visualization tool is implemented in a software package *NKVis*, i.e. a visualization tool for NK fitness landscapes, which is freely available at http://nn.cs.utexas.edu/?nkvis. This visualization will be used in this dissertation to demonstrate diversity behavior by the agents (e.g. Figures 4.2 and 5.5). The details of those agents are described next.

3.3 Agents

Each search agent is a software entity that looks for high-fitness points in the given fitness landscape. The behavior of an agent depends on the current state of the fitness landscape, the agent's strategy, and the current points in its memory.

Formally, the search agent's knowledge $\mathcal{X}(t)$ of the landscape and its topography at time t consists of the points $\mathbf{x}_i$ with fitness values $z(\mathbf{x}_i)$ $(1 \leq i \leq t)$, where z is the fitness function:

$$\mathcal{X}(t) = \{[\mathbf{x}_1, z(\mathbf{x}_1)], [\mathbf{x}_2, z(\mathbf{x}_2)], ...[\mathbf{x}_t, z(\mathbf{x}_t)]\}. \tag{3.1}$$

The agent moves to the next (i.e. $(t+1)^{\text{th}}$) point using a search strategy S based on what the agent already knows about the landscape (i.e. points visited by that agent and

other agents):

$$\mathbf{x}_{t+1} = S[\mathcal{X}(t)]. \tag{3.2}$$

An agent's strategy S consists of two components that determine how the agent will use its current knowledge. The first strategy component, S_1, specifies which type of memory (i.e. public or private, Section 3.4) and which search method (i.e. explore globally or exploit locally, Section 3.6) the agent will employ. The second strategy component, S_2, specifies in which type of memory the agent will place the last point it found.

The simulation runs in discrete time steps, where every time step each agent is allowed to move according to its strategy and based on its current knowledge (Algorithm 1). At the beginning of a time step, each agent probabilistically selects a search method and a source memory for the search starting point. This point is given as input to the search method chosen by the agent to find a new high-fitness point. At the end of each time step, if the agent has discovered a point that is better than the previous one, it schedules that point to be placed into the destination memory.

Algorithm 1 Agent's algorithm to complete one time step

1: Pick a search method and source memory probabilistically using S_1 strategy.
2: Perform one search step starting with the best point in the source memory.
3: **if** found a better point than the last one **then**
4: Pick a destination memory probabilistically using S_2 strategy.
5: Schedule placement of the new point in the destination memory.
6: **end if**

Even though the simulation advances the agents sequentially (Algorithm 2), the collective outcome of each time step in the simulation is independent of the agent execution order, due to the delayed execution of the side effects of agent actions. That is, every time step the simulator records landscape visits performed and public memory updates scheduled by agents, without actually making any changes immediately (line 5 in the algorithm). The recorded agent visits are performed at the end of each time step after all the agents complete their search steps, and the resulting landscape changes are carried out (line 7). Similarly,

public memory updates scheduled during a time step are applied at the end of that step (line 8). The simulation is then carried out until the agents have searched a significant part of the space (100 points in the experiments in Section 4.5 and Chapter 5, 500 in Section 4.4, and 200 in the rest of Chapter 4). The memory types will be described next.

Algorithm 2 Simulation algorithm

```
1: Initialize simulation and agents.
2: for each time step until the maximum number of steps is reached do
3:    for each agent a_i do
4:       Advance a_i one time step (Algorithm 1).
5:       Record any landscape visits and public memory updates by a_i.
6:    end for
7:    Apply recorded landscape visits by updating fitness landscape.
8:    Apply recorded public memory updates.
9: end for
```

3.4 Memory

Agents can place points in two types of memory: public and private. In terms of the innovation search example, public memory corresponds to public knowledge through patents, and private memory to trade secrets. Because memory contents change at each time step, public and private memory are denoted as $\mathcal{X}_{\mathrm{pub}}(t)$ and $\mathcal{X}_{\mathrm{priv}}(t)$, respectively.

Public memory can be accessed by all agents, and it serves as a common knowledge base among agents, thereby enabling cooperation. That is, through the use of public memory, agents can communicate the discovery of high-fitness points to other agents. This information in turn attracts the other agents to those good points, potentially benefiting all agents. However, use of public memory also makes it likely for all agents to spend their time in the same region of the search space, and through dynamic landscape changes, may lead to decreasing fitness.

On the other hand, private memory is unique to each agent. Each agent can place points in its own private memory, where they are hidden from other agents. When agents

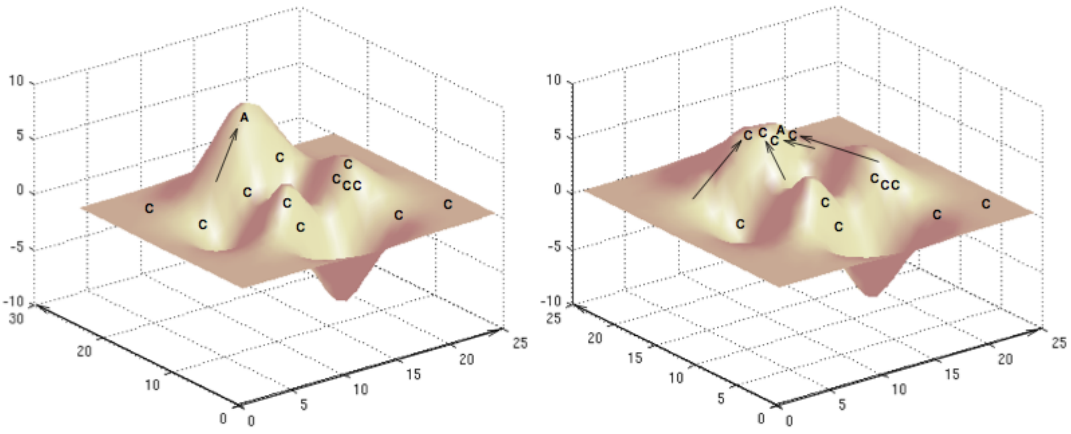


Figure 3.4: A conceptual illustration of crowding. (a) Agent A is first to discover a high-fitness point. (b) Soon after, multiple agents (competitors, labeled C) flock to the same peak. As a result, the peak region sinks. This crowding effect and the opposite effect of boosting model how agents interact indirectly through landscape changes.

make use of their private memory, they are more likely to spread out and explore different regions of the fitness landscape, leading to increased coverage of the search space.

Knowledge, i.e. agents' point memory, and the changing landscape, i.e. flocking, result in competitive multi-agent search. They capture interactions between agents, which are described next.

3.5 Agent Interactions

There are two types of interactions between agents: cooperation and competition, of which the two memory types are a crucial part. Private memory models the knowledge hidden from other agents, thereby providing a competitive environment. On the other hand, public memory models the knowledge shared among all agents. Therefore, public memory provides a direct way to cooperate.

On a fitness landscape with crowding, i.e. flocking with intensity less than 1.0,

the fitness landscape region around an agent sinks, which is akin to harvesting or depleting limited resources. Therefore, in such landscapes, to maximize their performance, the agents need not only find high-fitness regions in the landscape, but also do so before others do, thereby creating a competitive environment. Figure 3.4 visualizes this idea. Competition is also facilitated by the use of private memory, since it lets agents hide their discoveries from others, preventing them from accessing those points.

Another type of agent interaction is cooperation, which happens when the fitness landscape has boosting, i.e. flocking with intensity greater than 1.0. In that case, when multiple agents are in the same fitness landscape region, they benefit from each other's existence: The fitness landscape rises more quickly than it would with a single agent in the same region, which might happen, for example, in innovation search when excitement is created about a new product. Furthermore, cooperation is also obtained when agents use public memory since that allows each agent to be informed about high-fitness point discoveries of other agents.

A third type of interaction involves both cooperation and competition. Real-world competitive search domains often have this type of interaction. For instance, in an innovation search domain where firms search for products in product feature space, an agent finds a useful product, i.e. a useful combination of product features. When informed about this discovery through cooperative means, such as via public memory, other firms flock to the same region and look for ways to improve upon the initial point, i.e. exploit that region in the search space. The market for these products grows as consumers get excited about them and the firms find more value for them. That region in the landscape becomes fruitful for a limited time because of the excitement and possibly because firms add to its value by discovering new uses for the product, benefiting themselves and others, i.e. resulting in cooperation. However, as the region loses its novelty and the improved innovation benefit period wears out, competition becomes more dominant and the region gradually becomes depleted. Eventually, firms can no longer benefit from spending time in that region and

move elsewhere in the space.

These cooperative and competitive interactions are key to the modeling of competitive multi-agent search in this dissertation. The next section describes how agents search for new points.

3.6 Search Methods

To find a new point, an agent takes the best point currently in the memory (either public or private, depending on the strategy), and performs a search using this point as the starting location. Agents employ two search methods probabilistically depending on their S_1 strategy: the exploit search method, i.e. taking a local step, and the explore search method, i.e. making a long jump in the search space. These two methods are motivated by how agents, such as innovating firms, search in the real world (Section 2.1; March, 1991; note that thus these terms in this dissertation do not refer to deterministic and stochastic actions like they do in the reinforcement learning literature, but instead to the length of the search step, as they do in the management science literature).

The *exploit* search method starts with a given point in the search space. It then tries to discover new high-fitness points that are immediate neighbors of that point, i.e. are at 1-bit distance from it. Each new point is generated by flipping one bit in the point's N-bit representation. If the new point has a better fitness than the starting point, that point is placed in memory. Otherwise, the search continues by flipping another bit of the starting point, and so on until all bits are tried. The order of the flipped bits is a random permutation of numbers 1 through N.

The *explore* search method also starts with a given point, but it obtains new points in a different way. From the starting point, it jumps to a new point that is *not* an immediate neighbor. More specifically, it generates a new point by flipping multiple random bits of the starting point simultaneously and continues to do so until the new point has higher fitness than the starting point, or the maximum number of jump attempts is reached. The number

of bits to be flipped is also chosen randomly within a range given in simulation parameters. The range used in most of the experiments in this dissertation is [0.5, 1.0], which means that a new point in an exploration step is obtained by flipping between 50-100% of the bits of the starting point. The exploration action is thus a relatively long jump, and therefore distinctly different from the exploitation action.

These search methods provide two actions for agents to perform on the starting point. They are selected stochastically based on the agent's strategy, as described next.

3.7 Agent Strategy

Agents select among the two search methods using a search strategy. To represent this strategy, the state of each agent is converted to a discrete form based on whether the considered points' fitness values are less than 0.5 (i.e. *low fitness*) or more than 0.5 (i.e. *high fitness*). The two components of an agent's strategy, S_1 and S_2, consist of a set of probability values for each discrete state of the agent. Tables 3.1 and 3.2 show example S_1 and S_2 strategies, respectively. Each row represents a discrete state of an agent, and each column represents an action. The agents choose their actions probabilistically depending on their current state, according to the probabilities in these tables.

The S_1 strategy component is employed by agents to visit a new point. Using S_1, an agent selects both a search method and a starting point, i.e. the memory's best point from which the search begins, depending on the discrete-valued fitness of the best points of public and private memory. The state consists of the discrete-valued fitness of the two memories and the action is a combination of the search method and the starting point chosen. Thus, there are four possible states (low or high fitness for public memory's best point and low or high fitness for private memory's best point) and four possible actions (exploit or explore with public or private memory). The search yields a new point for the agent to visit. Thus,

State: \ Action:	Exploit with public mem.	Exploit with private mem.	Explore with public mem.	Explore with private mem.
Public: low fit. Private: low fit.	0.1	0.2	0.3	0.4
Public: low fit. Private: high fit.	0.0	1.0	0	0
Public: high fit. Private: low fit.	0.005	0.995	0	0
Public: high fit. Private: high fit.	0	0	0.9	0.1

Table 3.1: An example S_1 strategy component, where each row represents a state, and each column represents an action. S_1 consists of 4×4 probability values, one per action-state combination, which are used for selecting an action (i.e. a search method and a starting point) given the state (i.e. the binary-valued fitness of the best points in public and private memory). Each row of probability values adds up to 1.0, and determines what the agent will do in the corresponding state. For instance, the first row of this particular strategy specifies that when both the best public point and the best private point have low fitness, the agent will exploit that public point with 0.1 probability, exploit that private point with 0.2 probability, explore starting with that public point with 0.3 probability, or explore starting with that private point with 0.4 probability.

State: \ Action:	Place in public memory	Place in private memory
New point: low fitness	0.25	0.75
New point: high fitness	0.7	0.3

Table 3.2: An example S_2 strategy component. S_2 consists of 2×2 probability values (for two actions and two states) for determining to which memory to place the new point given the discrete fitness of that point. Each row of probabilities adds up to 1.0, and determines what the agent will do in the corresponding state. In this example, the agent will place each new low-fitness point into public memory with 0.25 probability and into private memory with 0.75 probability. The agent strategies can be visualized graphically as shown in Figure 3.5.

S_1 can be formalized as

$$\mathbf{x}_{t+1} \quad = \quad S_1(\mathcal{X}_{\text{pub}}(t), \mathcal{X}_{\text{priv}}(t)). \tag{3.3}$$

On the other hand, using the S_2 strategy component, agents determine where to put

a newly discovered point (i.e. the action) depending on the fitness of the new point $\mathbf{x}_{t+1}$ (i.e. the state). Thus, there are two states (low fitness or high fitness) and two actions (placing the point in public or private memory). Formally, S_2 is used to update knowledge, i.e.

$$\begin{aligned} \mathcal{X}(t+1) &= \{\mathcal{X}_{\text{pub}}(t+1), \mathcal{X}_{\text{priv}}(t+1)\} \\ &= S_2(z(\mathbf{x}_{t+1}), \mathcal{X}_{\text{pub}}(t), \mathcal{X}_{\text{priv}}(t)). \end{aligned} \tag{3.4}$$

where z is the fitness function.

The S_1 and S_2 strategies determine how agents behave and how knowledge gets updated in the simulation. They can be represented directly as vectors of real numbers such as those in Tables 3.1 and 3.2, and visualized graphically in Figure 3.5, and this representation is sufficient for the hand-coded strategies in this dissertation. In Section 5.1, strategies will also be encoded as CPPNs so that they can be evolved via neuroevolution methods.

3.8 Conclusion

This chapter described a formalization for CMAS in an abstract domain. This formalization includes a fitness landscape based on the NK model, modified so that fitness of regions around points that are visited by agents change. They are first to be boosted, then be reduced. The agents that search for points on this fitness landscape were also described. Specifically, the strategy they used to find new points in the search space, the public and private memory where they store those new points, which allow them to cooperate and compete, were detailed. Furthermore, since N-dimensional binary spaces get prohibitively large with high N, a spherical visualization for NK fitness landscape was introduced. It allows representing the local neighborhood of a chosen point with high resolution and points farther away with lower resolution. The next chapter will focus on investigation of the effects of various strategy parameters, and characterization of performance of various hand-coded agent strategies in different environments.

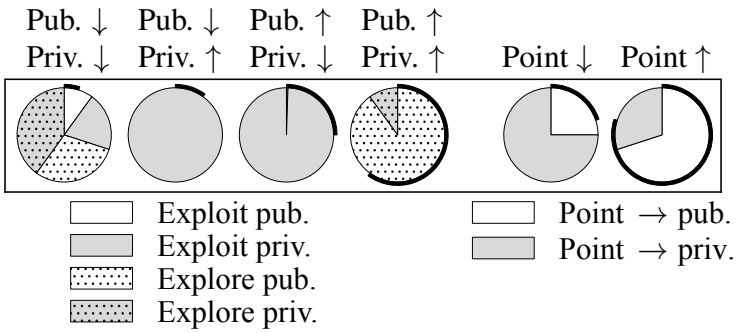


Figure 3.5: A pie chart depiction of the example agent strategy shown in Tables 3.1 and 3.2. Each of the circles corresponds to a row of one of those two tables (i.e. a state): four circles on the left for S_1 and two on the right for S_2, where $\downarrow$ and $\uparrow$ represent low and high fitness, respectively. Each shading pattern corresponds to a column of one of the two tables (i.e. an action). The size of each slice represents the probability of the action in the corresponding table row and column. This chart format allows visualization of the 20 probability values of a strategy in a compact way. Additionally, the black band around each circle indicates the average percentage of time the agent spent in the state that corresponds to the circle (i.e. a row in Tables 3.1 and 3.2). The example percentages shown from left to right are 5%, 10%, 25%, and 60% for S_1, and 20% and 80% for S_2. The total area of gray slices overall indicates how much private memory is used, whereas the amount of dotted shading in the first four circles tells the ratio of exploration. For instance, the first circle shows a row with non-zero probabilities for all actions, whereas the second circle specifies that only one action is possible in the corresponding state (i.e. with 1.0 probability). In certain cases, the probabilities for all but one action are close but not equal to 1.0 (e.g. third row in Table 3.1), which is seen in the visualization as a sliver (e.g. the third circle). Such small but nonzero probabilities were often discovered in the evolutionary experiments (Tables 5.1 and 5.2), and they turned out to make a significant difference in performance compared to similar fixed strategies where those probabilities are 0.0 (Figure 5.4).

Chapter 4

Characterization of *NK* Strategies

This chapter includes experimental analysis of the choice of memory type (i.e. private vs. public memory) and search method (i.e. exploitation vs. exploration search), benefits of a specific intuitive strategy, and effects of exploration distance (i.e. short, long, and gradually decreasing exploration) and agent density (i.e. sparse vs. dense). A number of different environments where the agents search according to fixed predetermined strategies are evaluated, with the goal of characterizing the effects of search diversity, search method balance, exploration focus, environment size, and opponent strategies in CMAS in general.

4.1 Effect of Public vs. Private Memory

The first experiment investigates how different mixtures of the two memory types affect how diverse the search is. The two types of memory are used together at levels varying from using only private memory to using only public memory, with three intermediate levels. The goal of the experiment is to understand how knowledge sharing and diversity affect search.

In one extreme (i.e. when agents use only private memory) each agent searches without any information sharing, resulting in maximal diversity. In competitive environ-

ments diversity delays crowding and improves the performance of agents. On the other hand, public memory allows agents to take advantage of the discoveries of other agents, and hence may be beneficial to the agent as well. Therefore, there is a trade-off in the use of public versus private memory.

This experiment was set up to evaluate the effects of using these two types of memory. The simulations had 10 agents with identical strategies on an NK landscape with $N = 30$ and $K = 4$. Landscape changes with initial boosting followed by crowding, with the flocking intensity decaying from 1.2 to 0.9 linearly. This way of changing flocking intensity creates interactions with both cooperation and competition as described in Section 3.5. Two jump ranges were used for exploration, [0.5, 1.0] (a long range) and [0, 0.2] (a short range). A jump range of [0.5, 1.0] means that a new point in an exploration step is obtained by flipping between 50-100% of all bits of the starting point, resulting in a relatively long jump.

In this experiment and the experiments through Section 4.4, all agents had the same strategy, and were allowed to use S_1 (i.e. select a new search method and source memory) only once every five time steps, and keep using the same ones for the following four steps in order to have the agents behave more consistently. Also, the multi-agent simulation ran for 200 time steps (except in Section 4.4, where it ran for 500 time steps), and was repeated 200 times. The average performance across all simulation runs and all agents was calculated for each agent strategy.

Five agent strategies were compared, each with a different S_1 strategy: (1) use only public memory, (2) use mostly public memory (i.e. public with 75% probability, private with 25% probability), (3) use public and private memory equally, (4) use mostly private memory (i.e. private with 75% probability, public with 25% probability), and (5) use only private memory. Each of the five strategies explored or exploited with 50% probability. S_2 was identical for each strategy, and used public and private memory equally.

Figure 4.1 (a) and (b) show the diversity advantage with flocking, with jump ranges

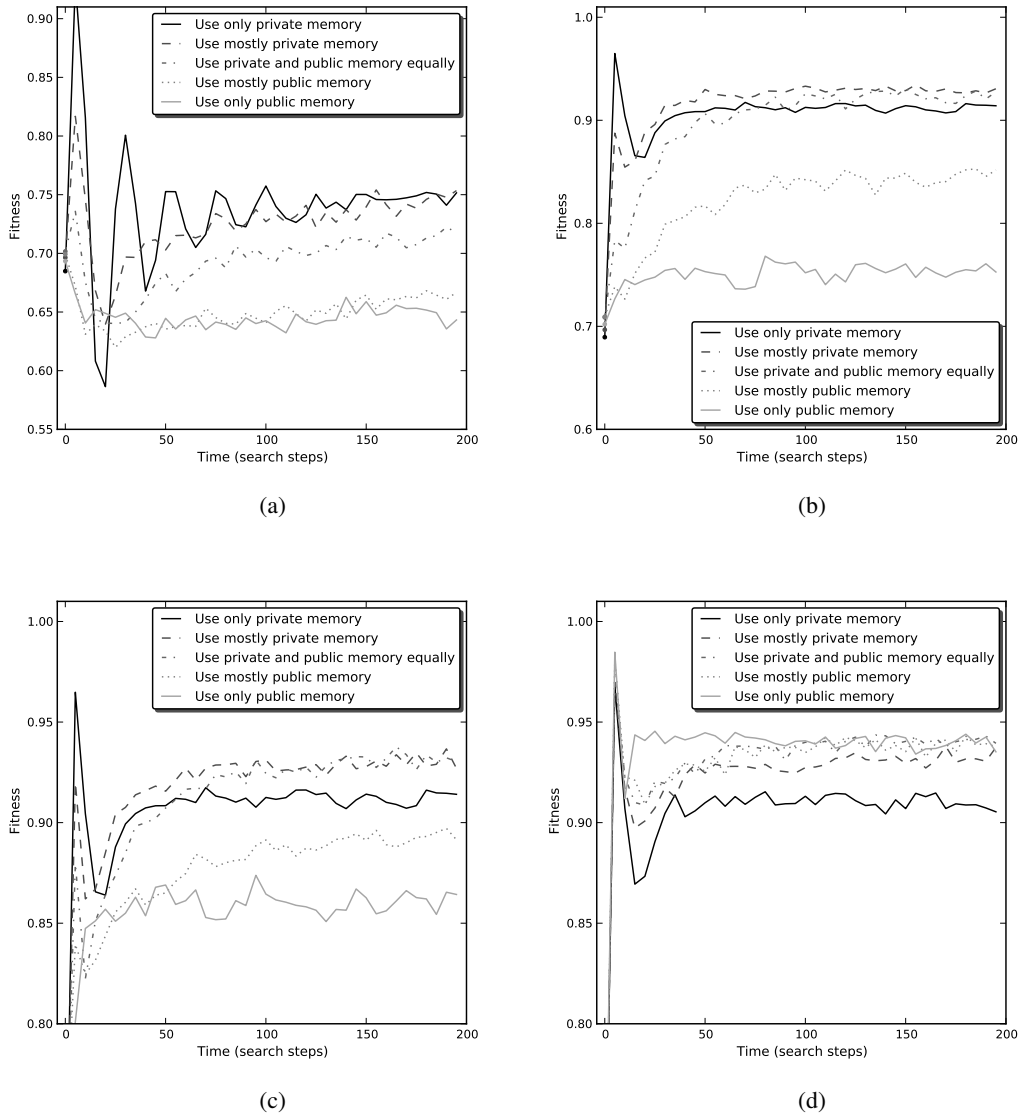
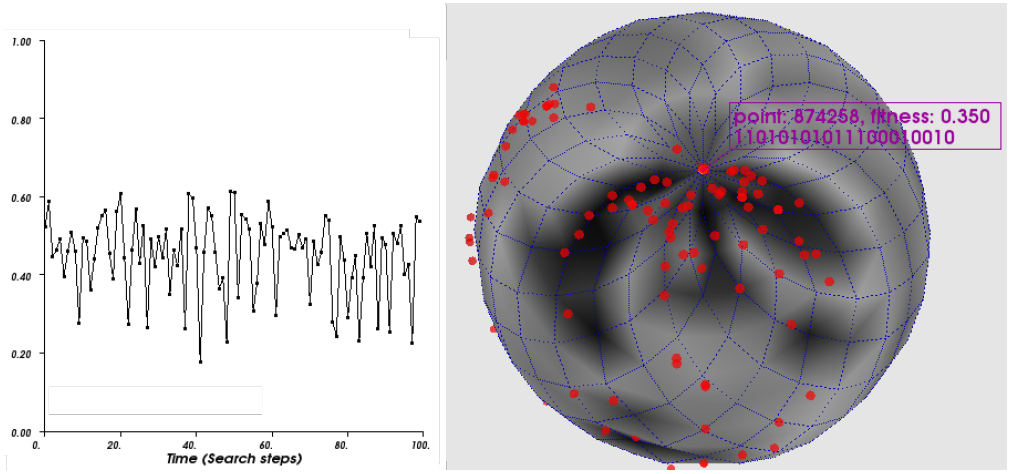


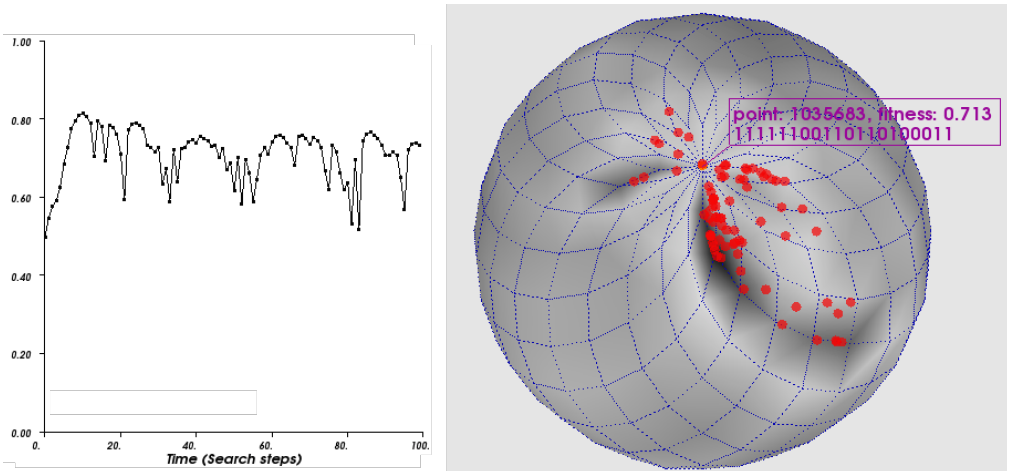
Figure 4.1: Effect of diversity with flocking, i.e. with a dynamic landscape. Each plot represents the average fitness of 10 agents searching with an identical strategy in the same landscape. Advantage of higher diversity (i.e. using more private memory) is seen (a) with jump range [0.5, 1.0] and (b) with jump range [0, 0.2]. When the agents utilize private memory, their searches are more diverse, and they find points with higher fitness. (c) The speed of crowding can be reduced by decreasing the limit for the number of visits at the same time step. Decreasing the limit to two reduces the advantage of diversity. (d) Reducing the limit further to one eliminates the advantage entirely. This result shows that the disadvantage of the public-only strategy is indeed caused by quick crowding (i.e. the *Twitter effect*). When all agents use only public memory, diversity is lost, and the agents perform poorly on average. Therefore, diversity is crucial for good performance overall.

[0.5, 1.0] and [0, 0.2], respectively. When agents use primarily private memory, i.e. visit a diverse set of points at a given time step, performance improves, with differing lower and upper boundaries depending on the choice of the jump range parameter. Simultaneous visits by multiple agents to the same point more quickly deplete the region around a point, taking that region into crowding faster than with a single visit per step. If such quick depletion is prevented by reducing the allowed maximum visits per step, the advantage of diversity is lost. This result can be seen in Figure 4.1 (c), where the maximum visits per step parameter is two, reduced from five in Figure 4.1 (b). When this parameter is further reduced to one, which allows only one flocking change to each point per step, the most diverse strategy, i.e. the one using only private memory, starts doing worse than the other strategies, as seen in Figure 4.1 (d). By limiting the rate of depletion in this manner, the reason why the use only public memory strategy performed worst among other strategies is verified: the quick depletion due to the lack of diversity. That is, when only public memory is used, all agents take the same best point in public memory as a starting point for their search, and therefore are more likely to be in the same region. Thus a *Twitter effect* results: all agents follow the same lead, and much of the space remains unexplored (Figure 4.2 (a)). As more private memory is used, less information is shared, and the points will become more diverse.

Interestingly, when the landscape does not change (i.e. when flocking is disabled), Figure 4.3 shows that using *both* memory types in the simulation is better than using only one type of memory. So, using only private memory performs worse than using any mixture of both memory types, unlike in the flocking case in Figure 4.1 (a). Flocking effects do not exist in this case, but there may be another advantage to having higher diversity in the no-flocking case: Searching with a more diverse set of points may lead to a higher probability of finding higher-fitness regions in the search space. However, employing public memory also helps in improving fitness by letting agents know about other agents' discoveries. So, there is a trade-off between public and private memory, as in the flocking case. The reason why there is relatively little performance difference between private-only and public-only



(a) The *Twitter* effect



(b) Wave-riding

Figure 4.2: The *Twitter* effect (a) and *wave-riding* behavior (b). The fitness of an agent's last visited point over 100 time steps is shown as a time series at left, and the corresponding spherical visualization of these points is shown at right. The fitness and binary coordinates of the agent's position at time step 100 are displayed in the purple box, and the agent's past visited points identified with red dots (see Section 3.2 for details of this visualization). (a) The fitness of an agent that always exploits with public memory where other agents also do the same stays low throughout the simulation due to low diversity and overcrowding (i.e. the Twitter effect). (b) The fitness of an an agent that always exploits with private memory initially rises with the landscape. As its current point starts sinking, the agent jumps to a nearby point that has been partially boosted, and spends a few time steps there. It then repeats this behavior. This behavior is clearest in sparse environments, demonstrating how agents can compete well by being constantly on the move. It is also similar to incremental improvement in many high-technology industries, giving them a computational interpretation. Animations of the Twitter effect and wave-riding behavior can be seen at http://nn.cs.utexas.edu/?bahceci:phd14.

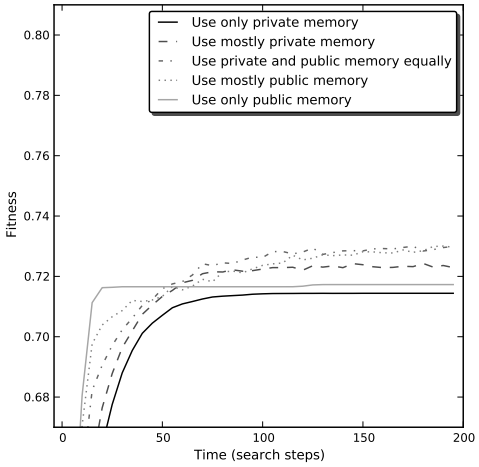


Figure 4.3: Effect of diversity without flocking, i.e. when the landscape is fixed. Using both memory types turns out to be better than using either only public or only private memory: Private memory helps find new high-fitness points, and public memory utilize those that have already been found.

strategies may be that both extremes in this trade-off are equally bad, simply due to the different landscape dynamics caused by the lack of flocking and a lower advantage to using private memory over public memory compared to the flocking case. So, whether keeping *all* information private is good or not depends on the amount of flocking in the environment.

The results of this experiment generally show that minimal diversity (i.e. Twitter effect) is bad for performance and the strategies that have higher diversity (i.e. ones that make use of private memory) perform better. In terms of the application of modeling innovation, this result means that it makes sense to keep diversity among agents high by keeping some information private (i.e. hidden from other agents) so that all agents do not crowd the same innovation neighborhood.

4.2 Effect of the Choice of Search Method

To see how the choice of search method (i.e. exploration vs. exploitation) affects performance, various combinations were tested. Changing the search method involves modifying the S_1 strategy, while keeping the S_2 strategy fixed.

Since the focus of this experiment is the search method, the strategies selected for the experiment are specified to use the same memory type, which is selected as private memory to prevent any interfering effects of communication between agents. So, the agents put all new points into private memory and pick their starting points only from private memory. The S_1 strategy is variable with different levels of exploitation and exploration depending on whether the best point in private memory has high or low fitness. Five S_1 strategies were compared: *always exploit*, *exploit on low fitness, explore on high fitness*, *exploit and explore equally* (i.e. with 50% probability), *explore on low fitness, exploit on high fitness*, and *always explore*.

Furthermore, to see how robust the observed effects are, two values for two simulation parameters, namely exploration jump range and boosting level, were considered, resulting in four sets of results. The jump range used for exploration was either [0.5, 1.0] or [0, 0.2]. As in the previous experiment, flocking consists of initial boosting and subsequent crowding. The flocking intensity gradually decays either from 1.2 to 0.9 or from 1.1 to 0.9.

Figure 4.4 shows the results for these four parameter settings. With high jump range (Figure 4.4 (b) and (d)), the *always exploit* strategy is significantly better than the other four strategies. However, when the jump range is low (Figure 4.4 (a) and (c)), there is not much difference, presumably because when the agent makes short jumps, the explore search method behaves similarly to the exploit search method.

Short jumps (which happen during exploitation or exploration) allow the agent to *ride a boosting wave*, i.e. move with the area that is being boosted (and maybe a little crowded) as it moves through the space. A single agent's fitness curve in a single simulation shows this *wave-riding* behavior in detail (Figure 4.2 (b)). An agent first stays on a point

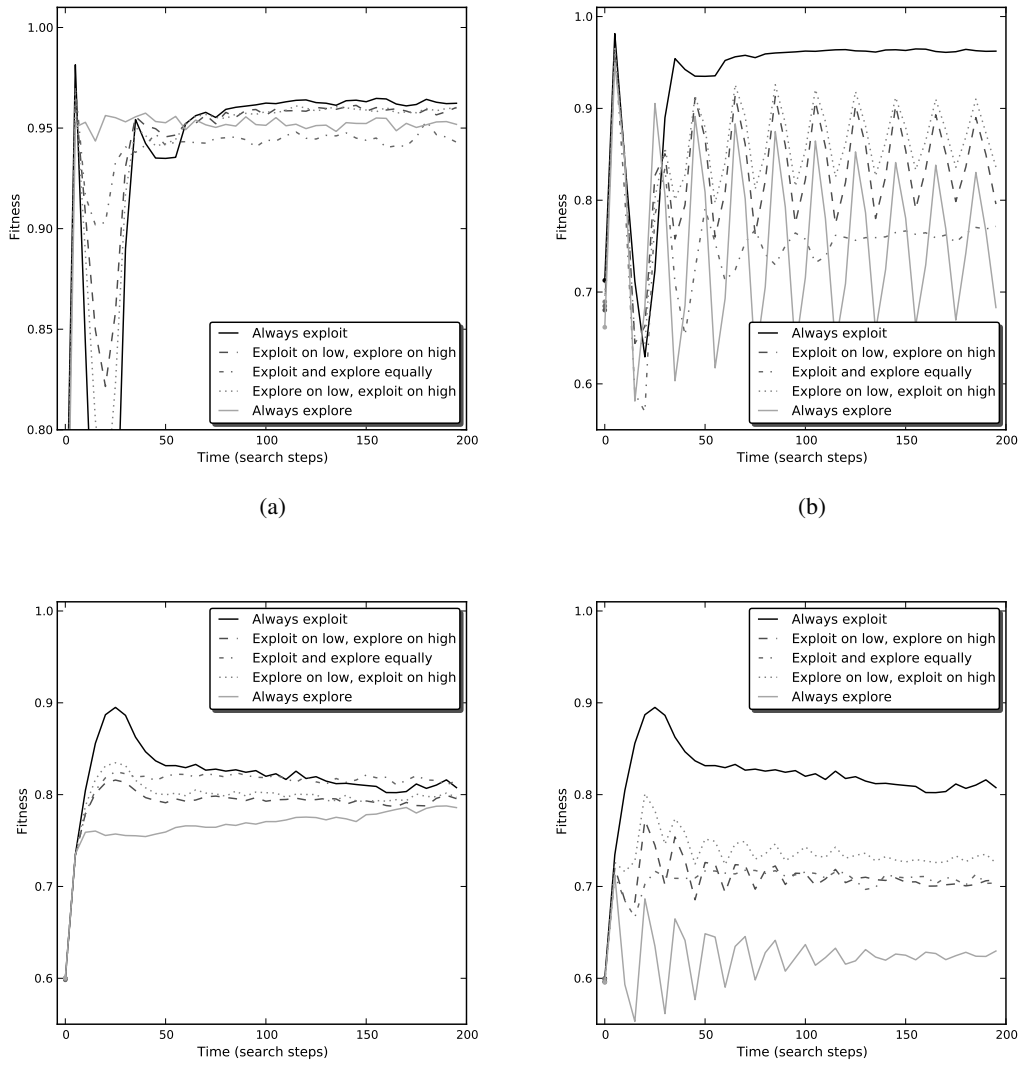


Figure 4.4: Effect of mixing exploration and exploitation at various levels in agents' strategies. Only private memory is used for both S_1 and S_2 in all five strategies. All 10 agents share the same strategy. Initial boosting intensity is high (1.2) in the top row and low (1.1) in the bottom row. When exploration jumps are long [0.5-1.0] as in the right column, strategies that do exploration show an oscillation behavior, unlike when exploration jumps are short [0-0.2] as in the left column. The *always exploit* strategy is significantly better than the others when exploration jumps are long, but not when they are short. Note that the y-axes have different scales to make these observations more visible.

that gets initially boosted and subsequently crowded, and then moves to different points a few times. The agent's timing is such that when it moves to a new point, the new point has been partially boosted previously, but it can still be boosted some more (i.e. has not reached the fitness ceiling of 1.0 yet). This behavior is possible because when an agent visits a point, it boosts all nearby points in the neighborhood (defined by the flocking radius); therefore, as the point starts losing fitness due to crowding, the agent can find a similar nearby point that is partially boosted. Hence, the agent moves on the top of a series of boosting curves similarly to a surfer riding a wave. When averaged across simulation runs and agents, the most effective wave-riding behavior is seen in Figure 4.4 (a), with the highest average fitness values.

Another interesting observation is an oscillation behavior with some strategies. A typical such behavior of an individual agent in a single run can be seen in Figure 4.5. Such oscillations happen when an agent is repeatedly unable to find a point that is better than their current point, which prevents the agent from riding a wave, and causes it to stay at its current point too long (i.e. until the point's fitness sinks below the fitness of points that it can reach in one time step). This type of regular oscillation is common with strategies that explore with long jumps, as can be seen in Figure 4.4 (b) and (d). When exploration jumps are no larger than the flocking radius, which is 2.0, as in Figure 4.4 (a) and (c), such oscillations do not emerge because explored points already get boosted by the agent when it moves to its current point.

The wave-riding behavior result is significant because it provides a straightforward and powerful way for agents to collect high-fitness points without changing their strategy. This result may be important as an insight in innovation search in real-world industries. If firms choose their next area of investment and product research wisely, such as choosing markets and interest areas that are becoming popular, they could ride a similar high-fitness wave.

On the other hand, the oscillations in some results prevent the agents from main-

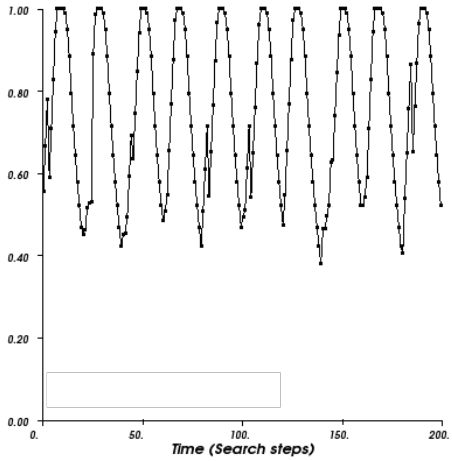


Figure 4.5: A typical oscillation behavior of an individual agent. This agent always explores using private memory. Such oscillations indicate that the agent is not able to ride a wave because it cannot find a point on which to move that has a higher fitness than its current point. Instead, it stays at the same point until that point's fitness becomes lower than fitness of points it can reach in one step.

taining high performance. So, it would be beneficial to avoid using strategies that lead to such high fluctuations. This observation also translates to innovation search: the innovators should move quickly to ride the wave instead.

4.3 Evaluation of *Explore on Low Fitness, Exploit on High Fitness* Strategy

This experiment investigates under which parameter settings the *explore on low fitness, exploit on high fitness* strategy (Greve, 2003b; Cyert and March, 1963) is better than other strategies. The reason this strategy is of interest is that it makes sense intuitively: If an agent encounters a high-fitness point, it should search its nearby neighborhood. If the landscape is not too rugged, there may be other good points in this neighborhood. If the agent comes to a low-fitness point, it should jump to a random far away region, which should, on average, be no worse than its current location.

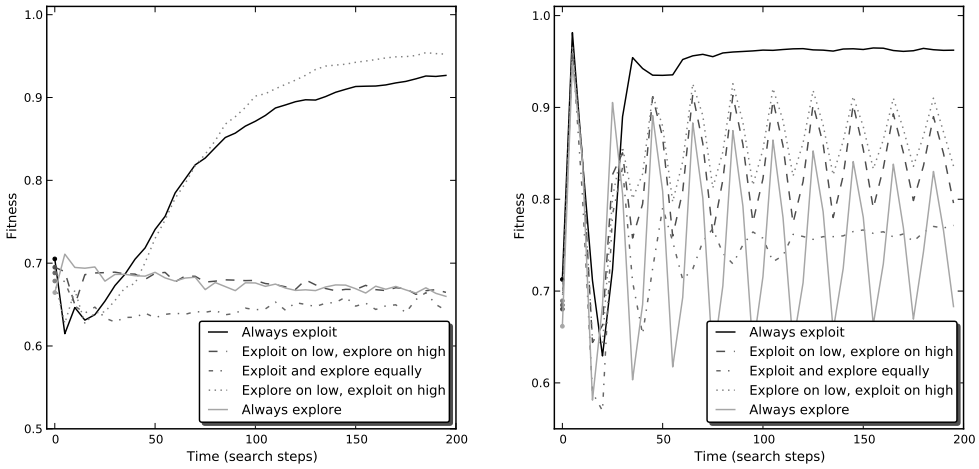


Figure 4.6: (a) The *explore on low fitness, exploit on high fitness* strategy is effective when only public memory is used, compared to other combinations of exploration or exploitation on high or low fitness. (b) However, that strategy offers no advantage when only private memory is used. In fact, always exploit works best in this case. The reason is that it establishes an effective wave-riding behavior. Thus, although intuitive, explore on low, exploit on high is thus not the best strategy in all environments.

Again, flocking consisted of initial boosting and subsequent crowding, with flocking intensity gradually decaying from 1.2 to 0.9 over 10 agent visits, and the exploration jump range was [0.5, 1.0]. Two settings were investigated: one where only public memory is used and one where only private memory is used (because these two settings gave characteristically different performance in earlier experiments).

As can be seen in Figure 4.6 (a), when only public memory is used, there is a statistically significant advantage to the *explore on low fitness, exploit on high fitness* strategy. On the other hand, when only private memory is used, that strategy is surpassed by the *always exploit* strategy, which benefits from the wave-riding behavior described in Section 4.2. The reason why wave-riding does not happen with the *always exploit* strategy with only public memory might be that the required fitness level and timing are different.

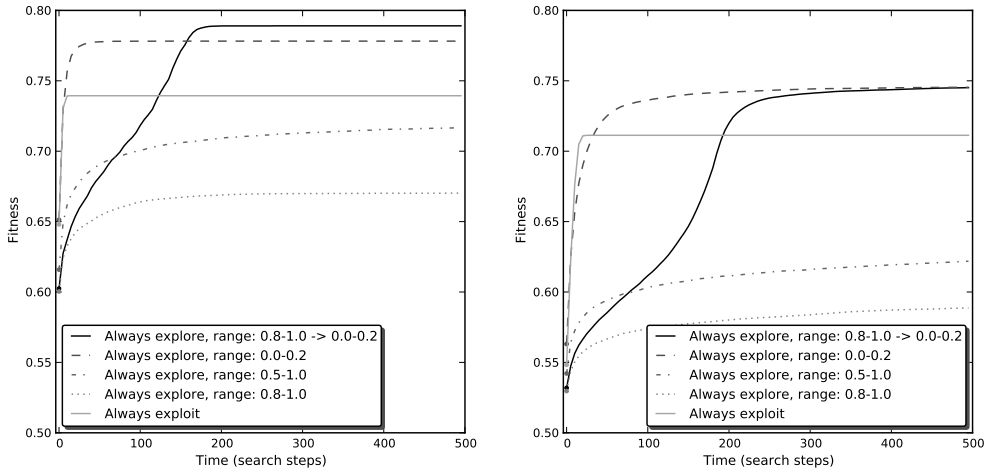
Thus, the initial hypothesis that the intuitive *explore on low fitness, exploit on high fitness* strategy would be advantageous holds only in some cases. Although it is possible to see where the intuitive idea comes from, the reality is much more complicated and interesting, and further study is required.

4.4 Effect of Exploration Focus

A fourth experiment investigates what happens when a different jump ranges are used in the explore strategy. The motivation was to determine the optimal exploration range, i.e. how far one should jump while exploring, to see whether gradually decreasing exploration jump range would be beneficial, and to evaluate how exploration with different jump ranges compared to an *always exploit* strategy. Gradually decreasing jump range is similar to simulated annealing (Černỳ, 1985), where the solution candidates can initially jump far away, but they are gradually *cooled* and are allowed gradually smaller jump freedom. This process improves the performance of optimization.

Performance of strategies that only explore with different constant jump ranges (namely [0, 0.2], [0.5, 1.0], and [0.8, 1.0]) were compared with each other and with another one whose jump range gradually reduces from [0.8, 1.0] to [0, 0.2], as well as with the *always exploit* strategy. This experiment was done with no flocking, i.e. with fixed fitness landscape, to isolate the effect of exploration. Two different number of dimensions (i.e. N) were used: a low value of $N = 12$ and a high value of $N = 30$, with $K = 4$ in both spaces. Because such a large increase in dimensionality results in a vastly larger landscape, individual strategies may not be equally effective for both small and large spaces. Therefore, a difference in the relative ability of the individual strategies was expected across the two spaces.

In Figure 4.7 (a) and (b), with both $N = 12$ and $N = 30$, using a short jump length resulted in better performance than using long jumps. The *always exploit* strategy, on the other hand, falls between the *always explore* strategies with jump ranges [0, 0.2] and [0.5,



(a) (b)

Figure 4.7: Effect of exploration jump range when there is no flocking with (a) $N = 12$ and (b) $N = 30$. Gradually decreasing exploration jump range turns out to be advantageous for $N = 12$, but not for $N = 30$. Thus, this strategy is helpful, but only if the search space is sufficiently small.

1.0]. The performance ranks among the constant *always explore* strategies was the same.

Figure 4.7 (a) also shows that when $N = 12$ there is an advantage to gradually decreasing the jump length, whereas in (b) with $N = 30$, short jumps and gradually shortening them both perform equally well. The reason may be that it is more difficult to find the highest region in the search space when N is 30 as opposed to 12. Gradually reducing jump length may make it possible to find such a region when N is small, but may not offer a sufficient boost when N is large, i.e. when the search space is much larger.

Thus, the simulated-annealing-like reduction in jump length was not always advantageous. The different relative performance of strategies between small and large spaces may be important as a general rule thumb. It is thus important to evaluate a strategy in environments with different characteristics, such as fitness landscape size and strategies of opponent agents, which the next experiment will investigate in depth.

4.5 Effect of Environment

The fifth experiment focuses on how a single agent performs in environments with different sizes and opponent strategies (i.e. strategies of agents other than the evaluated one). Unlike in the earlier experiments in this chapter, the simulation environments all had eight agents in this experiment, and the simulations ran for 100 time steps. Moreover, only the first agent's strategy was evaluated, and the remaining seven agents were given the role of *opponents*, all with a common manually specified and fixed strategy, which was not necessarily the same as the first agent's strategy. All agents were allowed to use S_1 (i.e. select a new search method and source memory) every time step. The S_1 and S_2 components of the opponents' strategies were set to constant probabilities, resulting in a CMAS environment with specific characteristics. Six distinct opponent strategies (i.e. six different environments) were implemented: always exploit or always explore, and always using the best point from the public memory, always using the best point from the private memory, or using either of those two points with a 50% probability. These hand-coded strategies were selected because they are intuitive, clear, and represent common strategies in innovation search (Katila, 2002; Greve, 2003a; Helfat, 1994).

The environments also varied in search space dimensions (N), resulting in two different densities of agents (i.e. number of agents divided by search space size): (1) sparse environments with $N = 20$ and (2) dense ones with $N = 10$. The K parameter of the NK landscapes was set to three in all environments to provide a moderate level of interaction among the N dimensions. Such moderate interaction is motivated from the organizational search perspective (Knudsen and Levinthal, 2007). The fitness landscape changed via initial boosting and subsequent crowding as agents moved in the landscape, achieved by decaying the flocking intensity from 1.05 to 0.9 linearly for each point over 10 agent visits to the same point, with a flocking radius constant at 2.0. Due to this dynamic nature of the fitness landscape, which is directly related to how many agents there are on average per search space point, the environments are more naturally characterized by agent density rather than

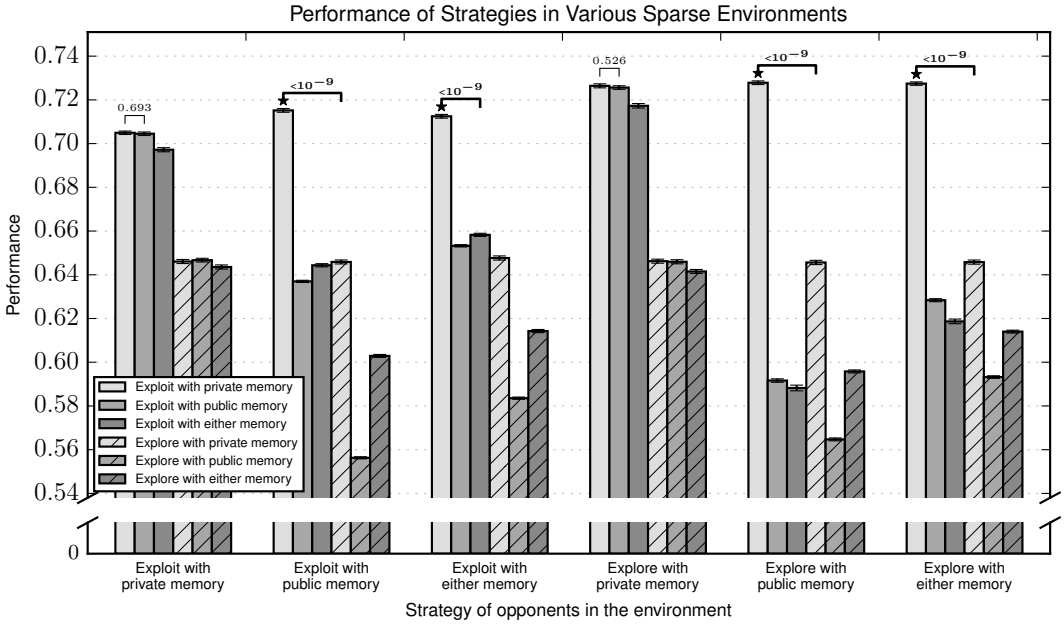
search space size. These environment parameter values were chosen because they resulted in meaningful innovation search behavior in preliminary simulations.

Performance with several distinct strategies in such environments is compared in Figure 4.8. In each environment and for each evaluated strategy, the simulation was repeated 200 times with randomized agent starting points in each run. The performance of the strategy was then calculated by averaging the performance across those runs, which in turn was defined as the average fitness of the points that the agent with that strategy visited during the 100 time steps (i.e the duration of the run). Since fitness of points in the search space varies between 0 and 1, so does each strategy's performance.

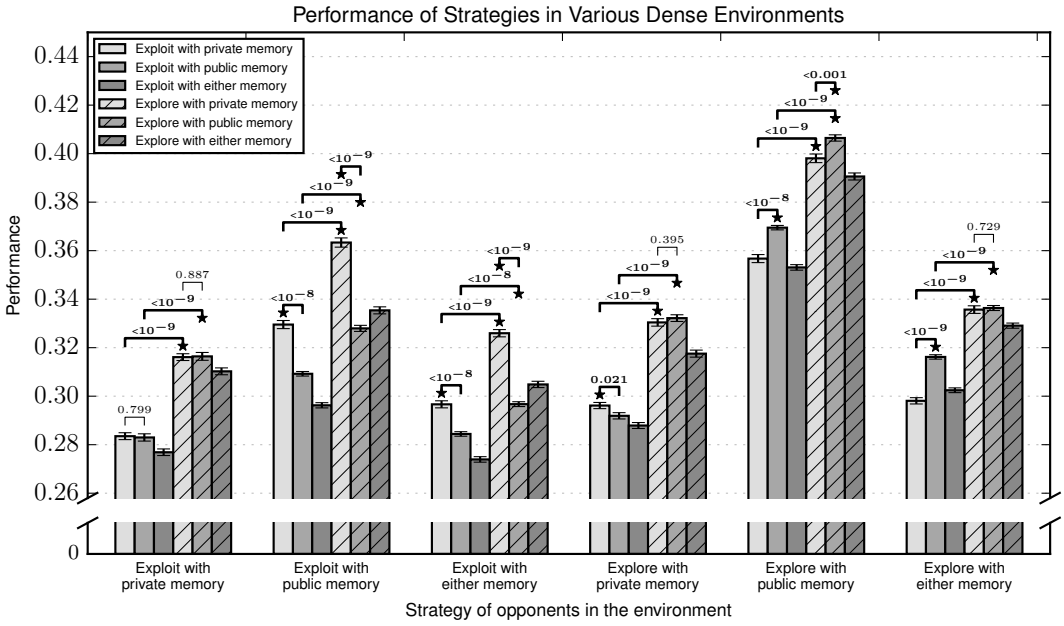
In the sparse environments (Figure 4.8a), the *exploit with private memory* strategy performed the best among all evaluated strategies (with p-value $< 10^{-9}$ compared to the one with the second highest mean performance), except in the environments where the opponents used only private memory. In those environments it tied with the *exploit with public memory* strategy. Indeed, when the opponents never access the public memory, that memory becomes equivalent to the private memory for the evaluated agent. Thus, the successful strategy in the sparse environments can be described as *exploit with a memory that is not accessible by the opponents*.

On the other hand, when the environment was dense (Figure 4.8b), exploring performed better than exploiting in general. However, the best memory type to use while exploring depended on the opponent strategy. For instance, when the opponents *explored with public memory*, it was better for the evaluated agent to use public memory (with p-value $< 10^{-3}$), whereas when the opponents *exploited with public memory*, it was better to use private memory (with p-value $< 10^{-8}$). Indeed, in this case the regions around good public memory points would generally become crowded and therefore have low fitness, making private memory search more effective.

These results suggest that different environments require different search strategies. Further, finding an effective strategy for a given environment is possible and constitutes an



(a) Sparse environments



(b) Dense environments

Figure 4.8: Performance comparison among manually specified strategies in sparse ($N = 20$) and dense ($N = 10$) environments. Statistical significance is estimated between averages over 200 starting locations and shown for the fixed strategies in each environment. Error bars denote one unit of standard error of the mean and statistically significant differences are indicated by stars. Note that because there is more interaction in the dense environment and therefore more crowding, the y-axes have different scales. Each strategy (represented by a bar of different color) was evaluated in six different homogeneous environments with seven identical opponents whose strategy is identified along the x-axis. Different strategies perform best in different environments; finding an effective strategy for a given environment is possible and an important and interesting challenge.

interesting and important challenge that will be taken on in the next chapter.

4.6 Conclusion

The experiments in this chapter investigated various effects of utilized memory type, including the Twitter effect that was observed when agents crowded the same region. Evaluating the effect of different search methods yielded observations such as wave-riding and oscillating performance curves, indicating that timing and how far an agent jumps when picking new points matters in obtaining consistently good performance. Also, a specific intuitive strategy (exploit on high, explore on low) was found to be effective only in specific circumstances, as was different exploration ranges. Finally, performance of a single agent was evaluated in environments with differing size and opponent strategies demonstrating that a choice of strategy matters. Obviously, such a hand-coded comparison can ever only include a small subset of all possible strategies. The ones included are prototypical and cover the space well, but there is no reason to believe that they are the best for the given environments. The next chapter will focus on optimization to find strategies that are better than these prototypical ones.

Chapter 5

Optimization of *NK* Strategies

In order to determine experimentally whether better strategies exist, this chapter first describes a strategy encoding suitable for optimization, then employs a machine discovery method to do the optimization. First, a strategy is optimized for each environment separately; second, the same method is used to create a general strategy by optimizing the average performance of a strategy across multiple environments; and third, they are compared with a real-time tree search strategy.

5.1 Encoding Strategy Patterns for Optimization

Preliminary experiments on learning strategies showed that representing them as continuous patterns through Compositional Pattern Producing Networks (CPPNs), which are neural networks with nodes that have various activation functions, and are useful for creating patterns such as symmetric ones (Stanley, 2007), as opposed to a set of distinct probability values corresponding to strategy tables, leads to better results. The S_1 and S_2 strategy components can be seen as functions that output the probability values in Tables 3.1 and 3.2. For instance in the case of S_2, the function takes two binary inputs: whether the new point has low or high fitness, and whether the destination memory being considered is the public or

private memory. Each combination of these binary inputs generates one of the probability values in Table 3.2. This function can be implemented as a CPPN with two inputs and one output, and with nodes that have either a sigmoid or a Gaussian activation function, which can be evolved with a neuroevolution algorithm.

On the other hand, in the case of S_1 there are four discrete inputs: whether the best point in public memory has low or high fitness, whether the best point in private memory has low or high fitness, whether exploit or explore action is being considered, and whether the source memory for that action is public or private memory. This function can also be represented as a CPPN with one output, but with four inputs instead of only two, to produce probability values as in Table 3.1.

To simplify the approach further, the functions for S_1 and S_2 can be represented together by a single CPPN (Figure 5.1) with four inputs and two outputs instead of by two separate CPPNs with one output each. Utilizing such a combined CPPN for both S_1 and S_2 allows a single population of networks to be evolved. The components that belong together to evolve together, sharing common structure.

To generate the 16 probability values in S_1 (as in Table 3.1), all four inputs of the CPPN network are used. The inputs are set to each combination of -1 or 1 in turn, representing the different cells in the strategy table (where -1 corresponds to the 0 index). The first output of the network is then entered into the corresponding S_1 table cell. After all cells have been filled in this manner, the rows of S_1 are normalized to add up to 1. If all values in a row are very small, then all cells in that row are set to equal probability. The process for S_2 is similar to S_1, except that only the first two inputs of the CPPN are used, and the second output unit of the network is used to calculate the values for the S_2 cells. This approach allows agent strategies to be evolved conveniently as CPPNs, which will be described in the next section.

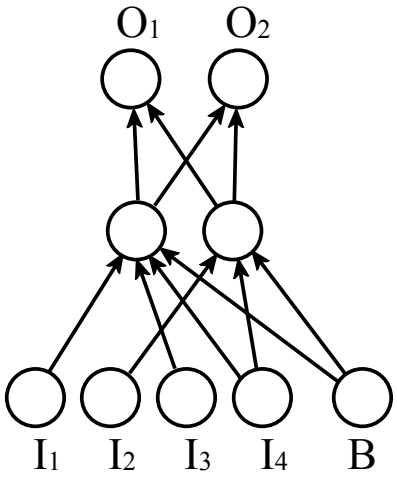


Figure 5.1: An example CPPN with four inputs, a bias input, and two outputs. The number of links and hidden nodes as well as weights of existing links are not fixed, and can change during evolution. Combinations of -1 and 1 values supplied to input nodes I_1, I_2, I_3, and I_4 act as indices into the rows and columns of Tables 3.1 and 3.2 (with -1 value representing index 0). For S_1, I_1 specifies whether public memory has low or high fitness, I_2 specifies whether private memory has low or high fitness, I_3 specifies whether the action is exploit or explore, and I_4 specifies whether the action is carried out using public or private memory; the output value of O_1 for each combination of the four inputs determines the action probability for the state and action specified by that combination, which is the value shown in the corresponding cell of Table 3.1. Similarly, for S_2, I_1 specifies whether the new point has low or high fitness, and I_4 specifies whether the action is to place that point in public or private memory; the output value of O_2 for each combination of those two inputs determines the action probability for the state and action specified by that combination, which is the value shown in the corresponding cell of Table 3.2. While calculating probabilities for S_2, I_2 and I_3 are set to 0.0. In this manner, strategies can be represented continuously as CPPNs, with S_1 and S_2 sharing the same network structure.

5.2 Evolving Strategies for a Particular Environment

CPPNs that represent strategies were evolved using NEAT with a population of size 100 in the same homogeneous environments as before. NEAT was allowed to run for 500 generations, which was the point by when performance plateaued in preliminary runs (Figure 5.2). Evolutionary runs were repeated 64 times. Other parameters that were used during evolu-

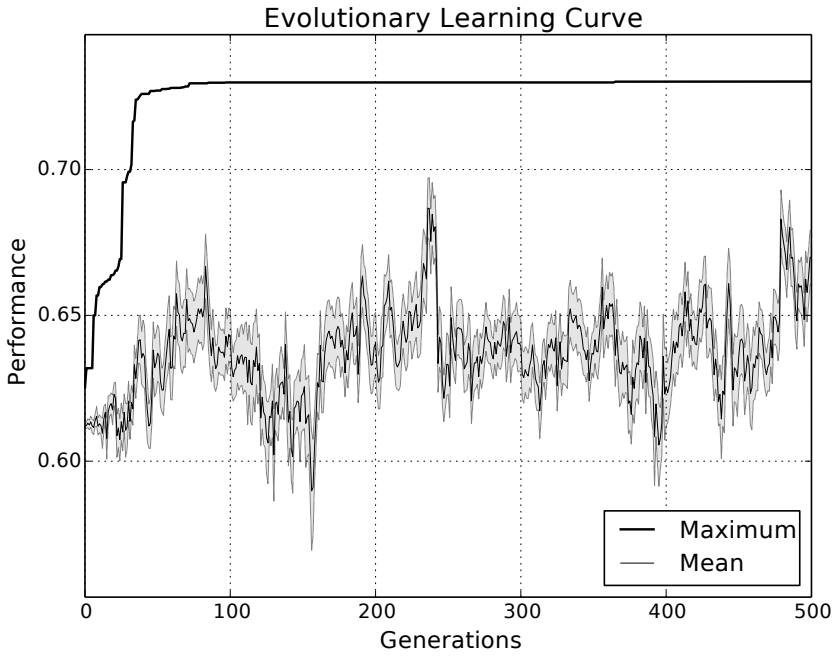
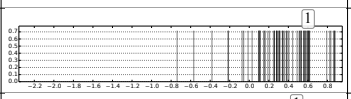
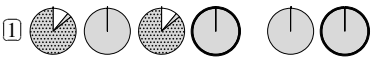
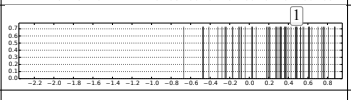
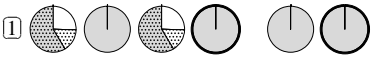
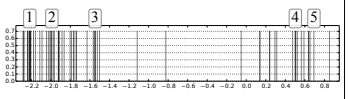
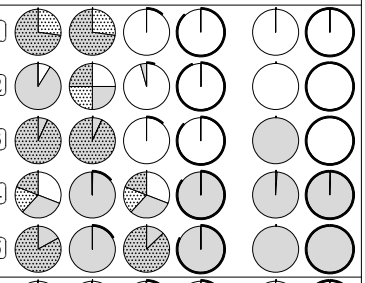
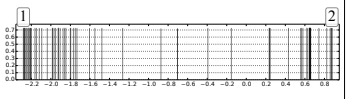
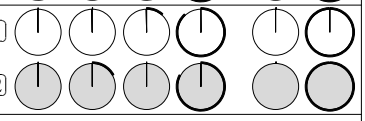
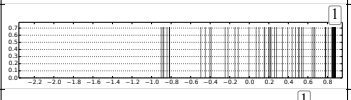
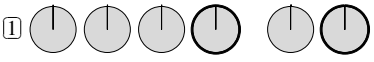
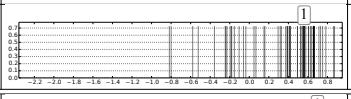
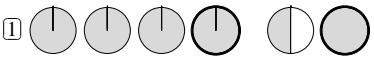
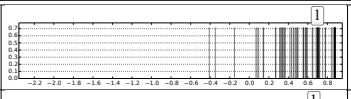
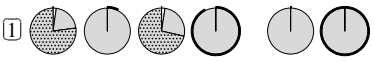
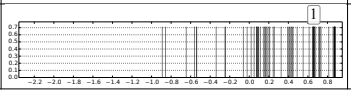
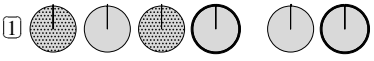


Figure 5.2: Learning curve for one of the evolutionary runs, showing the maximum and mean performance of the evolved population. The shaded region indicates standard error of the mean. Maximum performance usually plateaued well before 500 generations.

tion are listed in Tables A.2, A.3, and A.4. At each generation of an evolutionary run, the fitness of each population individual, i.e. each CPPN, was calculated by first generating the S_1 and S_2 probability tables from the CPPN, and then using those tables to control the single evaluated agent in the environment. Sizes of evolved CPPNs vary, but average 26.6 and 26.9 nodes, and 75.4 and 75.6 links for sparse and dense environments, respectively.

Tables 5.1 and 5.2 show distributions of strategies evolved on sparse and dense environments, with example evolved strategies depicted in pie charts as in Figure 3.5. To visualize the distribution of the 64 evolved strategies, they were first represented as 20-dimensional vectors and their dimensionality was then reduced to one using PCA (separately for sparse and dense environments). In the dense environments, the first principal component captured 83% of the variance, and in the sparse environments, 39%, whereas the second principal component captured 8% and 16%, respectively, suggesting that one-

Sparse Environment Id	Opponents	PCA for All Evolved Strategies	Sample Strategies
		⟵ Public Private ⟶	Pub. ↓ Priv. ↓ · Pub. ↓ Priv. ↑ · Pub. ↑ Priv. ↓ · Pub. ↑ Priv. ↑ · Point ↓ · Point ↑
1	**Exploit** with **public** memory		
2	**Explore** with **public** memory		
3	**Exploit** with **private** memory		
4	**Explore** with **private** memory		
5	**Exploit** with **either** memory		
6	**Explore** with **either** memory		
7	Environments 1-6 above and one with RTTS opponents (Avg. over all environments)		
8	One opponent from env. 1-6 and one RTTS opponent (Heterogeneous environment)		

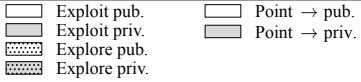
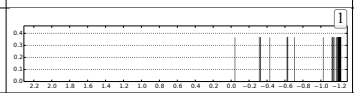
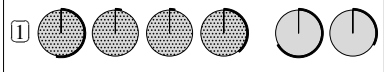
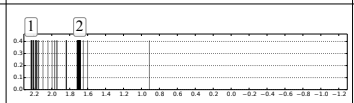
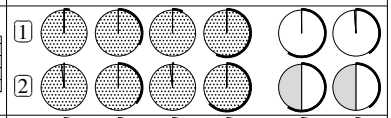
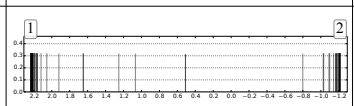
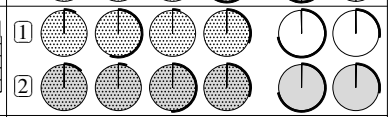
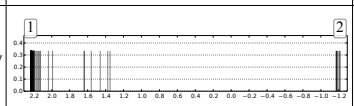
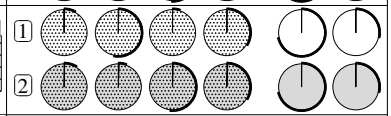
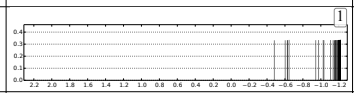
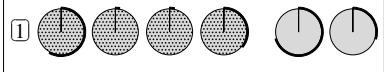
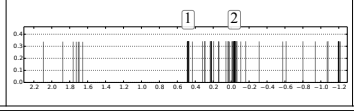
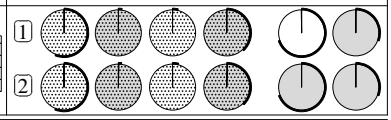
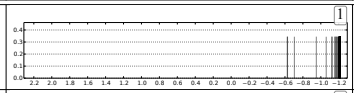
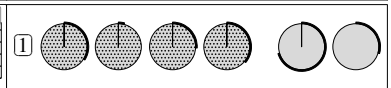
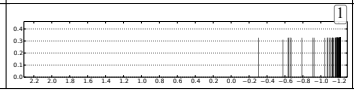
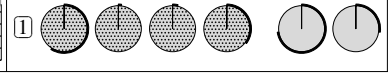


Table 5.1: Strategies evolved in sparse environments ($N = 20$). The third column shows all 64 strategies evolved in the environment with the opponents specified in the second column. The x-axis is the first principal component (from PCA across all strategies in all sparse environments) and the y-axis is the fitness of the strategy. The x-axis represents primarily public vs. private memory preference, with public memory use increasing toward the left and private memory use toward the right. Using the pie chart format of Figure 3.5, the fourth column displays sample strategies for each environment, numbered to indicate their positions on the PCA plot. Pie charts for all evolved strategies can be seen at http://nn.cs.utexas.edu/?bahceci:phd14. Most strategies cluster to the right, indicating that private memory search works well, except in Environments 3 and 4, where the distribution is bimodal (because the opponents do not use public memory). In some cases, parts of the strategy do not matter and there is considerable diversity (e.g. in Environment 3). The best strategies are slightly less than extreme, which allows them to perform better than fixed strategies.

Dense Environment		PCA for All Evolved Strategies	Sample Strategies
Id	Opponents		
		⟵ Public Private ⟶	Pub. ↓ Priv. ↓ / Pub. ↓ Priv. ↑ / Pub. ↑ Priv. ↓ / Pub. ↑ Priv. ↑ / Point ↓ / Point ↑
1	**Exploit** with **public** memory		
2	**Explore** with **public** memory		
3	**Exploit** with **private** memory		
4	**Explore** with **private** memory		
5	**Exploit** with **either** memory		
6	**Explore** with **either** memory		
7	Environments 1-6 above and one with RTTS opponents (Avg. over all environments)		
8	One opponent from env. 1-6 and one RTTS opponent (Heterogeneous environment)		

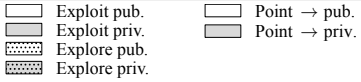


Table 5.2: Strategies evolved in dense environments ($N = 10$). The x-axis scale in the PCA plots is inverted so that the shift from public to private memory use is in left-to-right direction as in Table 5.1; the y-axes have a different scale from Table 5.1. In the dense environments, there is less variability due to stronger evolutionary pressure. Memory preference in different environments is similar to that in Table 5.1, with the exception of Environment 2, where an opposite memory preference emerged. This effect is investigated further in Figure 5.3. Pie charts for all evolved strategies can be seen at http://nn.cs.utexas.edu/?bahceci:phd14.

dimensional visualization is indeed meaningful. The reason for the lower variance captured by the first principal component in the sparse environments (39%) compared to that in the dense environments (83%) is that sparse environments lead to more diverse evolved strategies, which is investigated in the fifth observation below.

The first PCA dimension is the x-axis of the plots in the third column of Tables 5.1 and 5.2. Note that the x-axis scale is the same for all sparse environments, but different from the axis for the dense environments. The y-axis of the PCA plots indicates the fitness of the strategies. Interestingly, in both sparse and dense environments the x-axis corresponds to a preference for using public or private memory: The leftmost strategies across all plots have the highest public memory use, whereas the rightmost ones use private memory the most. This tendency can be clearly seen e.g. in the two sample strategies for Environment 4 in Table 5.1.

Several interesting observations can be made based on Tables 5.1 and 5.2. First, most strategies cluster on one side of the PCA plot, except for Environments 3 and 4, where a bimodal distribution is observed. These two environments are the only ones where the opponents uses only private memory. Since the opponents never use public points, public memory effectively becomes just another private memory for the evolved agent, and a bimodal distribution results. This result is very similar to the observation in Section 4.5 where public and private memory use resulted in similar performance when the opponents used only private memory.

Second, the distribution of strategies evolved in most of the remaining environments is biased toward strategies that prefer private memory over public memory. Thus, evolution discovered that it is good to be different from competitors. This result also parallels the performance comparison of manual strategies in Figure 4.8. Furthermore, private memory preference in sparse environments often results in wave-riding behavior as observed in Section 4.2. Hence, the evolutionary simulations rediscovered this strategy, thus demonstrating that it is indeed a good one.

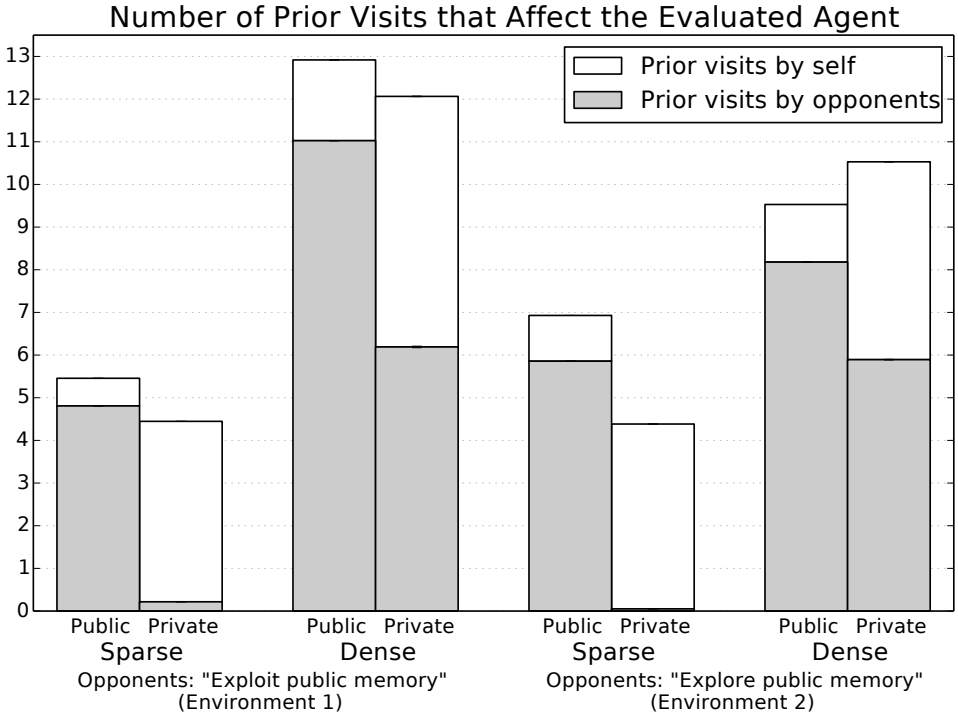


Figure 5.3: The number of prior agent visits within the flocking radius of the evaluated agent in Environments 1 and 2 in Tables 5.1 and 5.2. Agent visits cause the landscape to sink due to crowding, and lead to lower fitness. The bars labeled Public represent the average number of prior visits when the evaluated agent uses one of the 64 evolved strategies that prefer public memory the most (i.e. the leftmost 64 strategies in the PCA plots of all environments in Tables 5.1, for the bars labeled Sparse, and 5.2, for the bars labeled Dense. Similarly, the bars labeled Private show the average number of visits with the rightmost 64 strategies in the same tables. The bars' colors, white and gray, indicate prior visits by the evaluated agent (i.e. "self") or by its opponents, respectively. Switching from private strategies to public ones increases the total number of prior visits in three out of the four environments, but not in dense environment 2. As a result, evolution selects for a public memory strategy in this case, demonstrating that it can find effective strategies that would be difficult to discover by hand.

Third, in Environment 2 in Table 5.2, where opponents explore with public memory, a surprising opposite effect is seen: public memory is preferred over private memory. Not only is this result counterintuitive, it is also the opposite of that in its sparse counterpart.

This effect can be explained by measuring the number of prior visits to a given

area by the agent and its opponents. As Figure 5.3 shows, the number of prior visits that affect the evaluated agent is usually higher when the agent uses a public-memory strategy than when it uses a private-memory strategy, and therefore evolution usually favors private memory. However, the opposite is true for dense environment 2: The public-memory strategy actually results in fewer prior visits. The reason becomes clear when the source of prior visits is considered. When the evaluated agent uses private memory, it is affected by many of its own prior visits, but manages to avoid opponent's prior visits relatively well. When it switches to using public memory, it reduces its own prior visits, but also increases the opponent's visits. Such a trade is usually detrimental, but not so in the dense environment 2. Because the environment is dense and the opponents are exploring, even when the agent is using private memory, the opponents make many prior visits already by chance. Switching to public strategy therefore does not increase the opponent visits much, but it does reduce self-visits significantly. The net effect is therefore beneficial, and evolution will select for public-memory strategy for dense environment 2.

Fourth, the agents encounter different states in dense and sparse environments. This result can be seen by observing the black bands around the circles of sample strategies in Tables 5.1 and 5.2, which represent the average percentage of time spent in each S_1 and S_2 state. Across all evolved strategies in dense environments, the S_1 state where public and private memory have low fitness (i.e. the first circle) is encountered 34% of the time on average, whereas in sparse environments only 0.011%. Similarly, in dense environments the evolved agents spent 69% of their time in the S_2 state where the new point has low fitness, but only 0.593% in sparse environments. The reason is that the fitness landscape sinks more in dense environments, and by the end of the 100-step simulation, all points in the search space have low fitness. In contrast, in sparse environments, there are often points with high fitness that the agents can visit, and the agents evolve to do so. Therefore, evolved agents spend most of their time in high-fitness states in sparse environments and low-fitness states in dense environments.

Fifth, a comparison of the PCA plots across the two tables indicates that the strategies evolved in sparse environments vary more than those evolved in dense ones. For example, in Environment 3 in Table 5.1 the evolved strategies are dispersed much more than their counterparts in Table 5.2. It is interesting to analyze why. First, note that strategies located on the left side (i.e. those that use public memory more often than private memory) mostly differ in which actions they take when fitness of public memory is low (i.e. the first two circles of sample strategies 1, 2, and 3), and when the points they visit have low fitness (i.e. the fifth circle). Note further that such states are visited only rarely (i.e. less than 1% of the time), as indicated by very little black band around the circles. The reason is that only the agent itself changes public memory (not its opponents, which exploit with private memory), and it places only high-fitness points in it. Therefore, public memory always has a high fitness, and as a result, the first two states are rarely encountered and the corresponding parts of the strategy (represented by the first two circles) are rarely used. Also, since low-fitness points are usually not visited in sparse environments, the fifth circle is rarely used. Similar observations can be made for the strategies on the right side (i.e. those that use private memory more often, such as sample strategies 4 and 5): They vary mostly in states where private memory has low fitness (i.e. the first and third circle) as well as when visited points have a low fitness (i.e. the fifth circle), and these are indeed states they rarely visit. Thus, changes to those parts of the strategy do not affect the evolved agent's performance, and evolution results in diverse solutions for them. In contrast, in dense environments all states are encountered at least 9% of the time on average: All parts of the strategy are therefore useful, and there is less variance.

In Environment 3, opponents exploit private memory, and place the new points they find in private memory, which makes the evolved agent the only one (among the eight agents in the simulation) that changes public memory. Since each one of sample strategies 1, 2, and 3 places new high-fitness points in public memory (i.e. the sixth circle), public memory always has high fitness. Therefore, the first two states (which together represent public

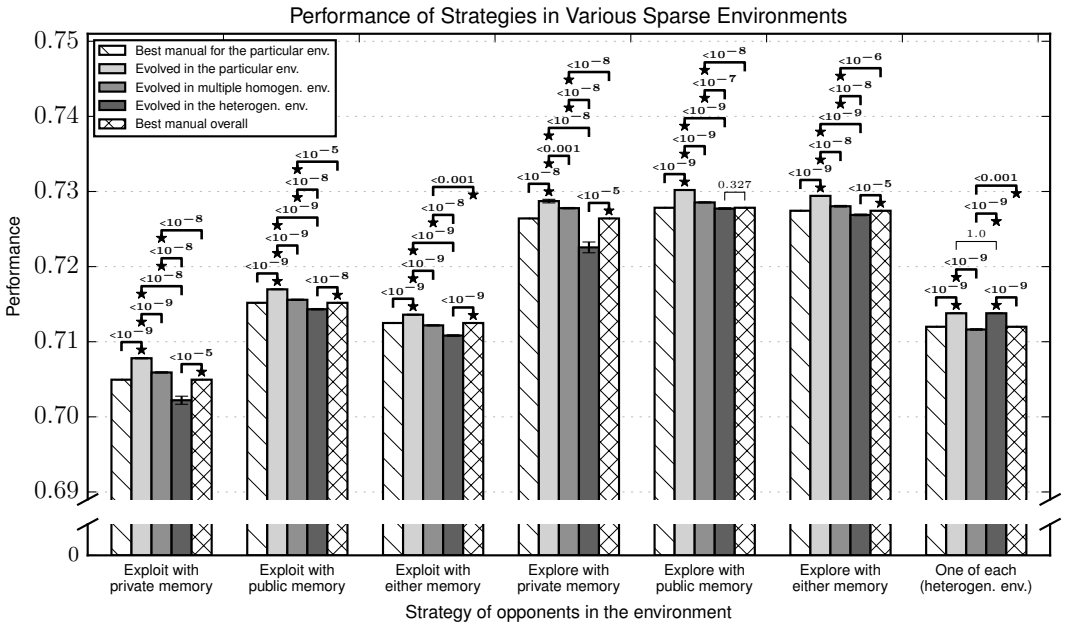
memory having low fitness) are not observed with those strategies. Thus, the corresponding parts of the strategy (i.e. the first two circles and rows in the S_1 table) are not used during simulation. In contrast, if the evolved agent uses private memory instead of public, as in sample strategy numbered 4 and 5, the unused states are different: The first and third circles, which together represent the state of private memory having low fitness, do not get used. Since low fitness points are usually not visited in sparse environments, the parts of the strategy for new low-fitness points (i.e. the fifth circle and the first row of the S_2 table) are not used either. Thus, changes to those parts of strategy do not affect the evolved agent's performance. During evolution, strategy variants with differences in those unused parts do arise, which leads to higher diversity in the evolved strategies in sparse environments. On the other hand, since in dense environments all states are encountered at least 9% of the time on average, most states are useful in each environment for the evolved strategies. Since most parts of the strategies are used in dense environments, there is relatively lower variance in evolved strategies, compared to sparse environments.

Sixth, the best strategies are not perfectly extreme, unlike the fixed hand-coded ones, but often contain small slivers of probability for alternative actions. Such small differences allow them to perform better. As can be seen in Figure 5.4, in each environment the evolutionary optimization resulted in a strategy that performs at least as well as the best manual strategy.

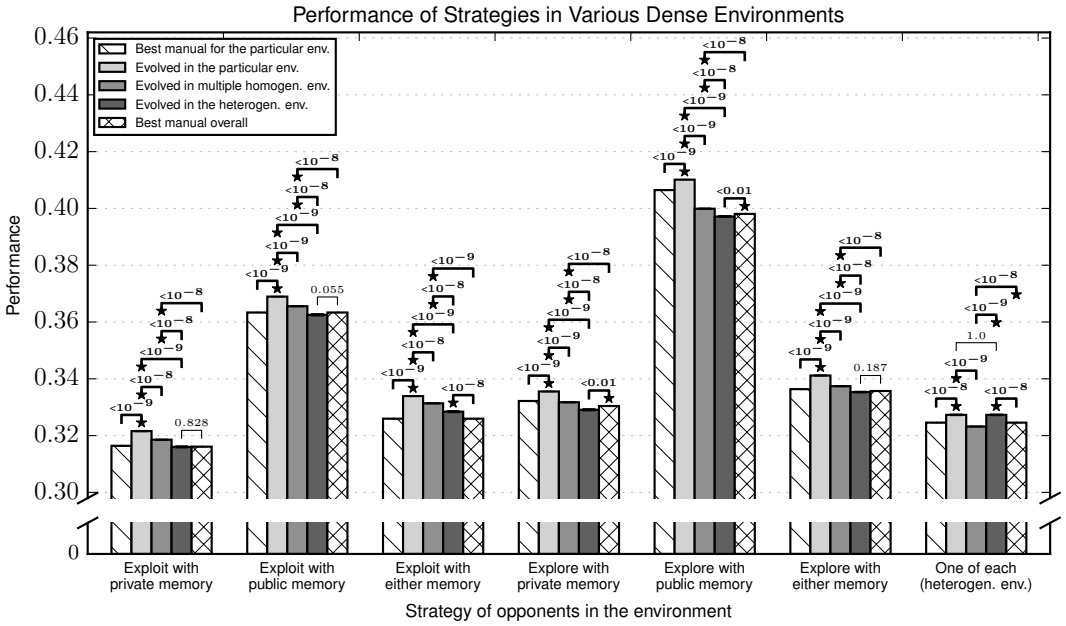
Thus, the results verify the hypothesis that custom-designed strategies are usually more successful than generic ones. An interesting question is: Are there general strategies that work well on all environments?

5.3 Evolving General Strategies

Two ways of evolving general strategies were tested: evolving in multiple homogeneous environments and in a single heterogeneous environment. In the first approach, seven homogeneous environments were used, consisting of the six environments above, and an envi-



(a) Sparse environments



(b) Dense environments

Figure 5.4: Performance comparison among evolved strategies as well as the best manual strategy from Figure 4.8 in sparse ($N = 20$) and dense ($N = 10$) environments. Statistical significance is estimated between averages over 64 evolution runs. The same six environments are included as in Figure 4.8, as well as a seventh one where each opponent had a different hand-coded strategy (the second and fourth strategies in this plot are identical, hence the 1.0 p-value between those two bars). Strategies evolved for each environment separately perform the best in all cases, while the general strategies evolved in multiple environments are better than the best manual overall strategy in almost all cases. Machine discovery is therefore a powerful approach to develop CMAS strategies.

ronment with seven opponents that use an adapted RTTS strategy (which will be described in the next section). During evolution, the fitness of each evaluated strategy was calculated by averaging the fitness score across those environments. The main disadvantage of this approach is that it takes a very long time: Each strategy must be evaluated in seven environments, using a total of 700 simulation steps.

The approach that uses a single heterogeneous environment avoids this problem. There is one opponent of each type in this environment, allowing the agent to interact with various types of opponents at once. Therefore, evaluation only requires one environment and 100 simulation steps. As in previous experiments, there are seven opponents but now each of them comes from a different homogeneous environment.

The strategies evolved using these two approaches can be seen in Tables 5.1 and 5.2 as Environments 7 and 8. Private memory was mostly preferred over public memory in both sparse and dense environments. With the homogeneous approach, strategies evolved in the sparse environment exploited private memory when private memory had high fitness (Environment 7 in Table 5.1); otherwise, their behavior varied, including exploring private memory while still sometimes exploiting it, as shown in the sample strategy. When the environments were dense (Environment 7 in Table 5.2), this approach evolved strategies that mostly explored, but also rarely exploited with private memory.

Similarly, in the heterogeneous approach (Environment 8), the evolved strategies in the sparse environment exploited with private memory when private memory had high fitness, but also explored when private memory had low fitness. On the other hand, the strategies evolved in the dense heterogeneous environment always explored with private memory. The likely reason is the same as in Section 5.2: There are opponents that use public memory, making it less beneficial to use.

Performance of the strategies evolved using the two general approaches was compared to that of the strategies evolved specifically for that particular environment, as well as to the manual strategy that performed best in that environment. The results can be seen

in Figure 5.4. In both sparse and dense environments, the performance of the strategy that was evolved for that particular environment was always the best of the evolved strategies (with p-value $< 10^{-9}$ compared to the strategy with the second highest mean in most environments, and $< 10^{-3}$ in the rest), and that was always followed by the single strategy that was evolved in multiple homogeneous environments, which in turn always performed better than the one evolved in the single heterogeneous environment (with p-value $< 10^{-7}$ in all environments). Interestingly, the performance of the single heterogeneous environment was on average within 1% of that of multiple homogeneous environments, even though it required one-seventh of the evolution time. Thus, evolution in a heterogeneous environment is an elegant and effective approach to finding general strategies.

Overall, the similarity in performance between the different learning approaches suggests that it may be possible to evolve a single strategy that is effective in various environments, although the very best results are obtained by customizing the strategy to each particular environment separately.

5.4 Comparison with Real-Time Tree Search

In order to highlight how different CMAS problems are from conventional search problems, a real-time tree search (RTTS) algorithm was devised for the NK fitness landscape. RTTS is real-time in the sense that it does not perform the whole search offline like A* does, but instead alternates between planning and execution phases by performing a limited look-ahead search at each state before selecting an action and moving to a new state. In this respect, this algorithm is similar to e.g. Real-Time A* (RTA*; Korf, 1990), a well-known search method in the single-agent tradition. Further, at each step RTTS, like RTA*, performs a full exploitation search from the current point, i.e. considers all successor states reachable by exploitative search actions. In contrast, CMAS also includes exploratory search actions, which can potentially reach any point in the search space.

The reason for defining and employing RTTS instead of simply using RTA* is that

the search domain of NK fitness landscapes differs in several ways from those for which RTA* was designed: (1) The goal is not to reach a certain point in the search space as in RTA*, but rather to follow a path that yields as much fitness as possible; (2) Since there is no goal point, there cannot be a heuristic to calculate the cost of reaching a goal point; (3) Avoiding loops is not a concern as long as the revisited points have high fitness; (4) The search space is dynamic due to flocking of agents, which makes it less useful to keep a hash table of observed states and their estimated costs for returning to those states. Thus, RTTS can be seen as an adaptation of RTA* to CMAS problems.

The RTTS algorithm works as follows. Given an agent's state s (in this case, the last point the agent has visited), a score is calculated for each successor state s'. Since only exploit actions are considered, a successor of a state is equivalent to a neighbor of a point in the space. Each point has one neighbor s'_i per dimension i ($1 \leq i \leq N$), obtained by flipping the bit of that dimension in point s. For each neighbor s'_i, RTTS carries out a look-ahead search starting from that state, and calculates the score of s'_i by summing the fitness of that point and the maximum fitness among those of the successors of s'_i. The agent's next move is chosen as the exploitation action that results in the state with the maximum score among all s'_i.

In fact, what the RTTS agent does for each s'_i is identical to what the agent does for s itself. Thus, the agent's search can be described as fixed-depth tree search. When the depth of this search tree is set to two, the points that RTTS evaluates consist of point s itself, all neighbors s'_i of s ($1 \leq i \leq N$), and all neighbors of all s'_i. The number of these points is $1 + N(N+1)/2$, which amounts to 56 and 211 points for $N = 10$ and $N = 20$, respectively. Therefore, to keep the number of evaluations at a reasonable level, the RTTS search depth was limited to two (Figure 5.5).

The results, compared to best evolved CMAS strategies, are shown in Figure 5.6. Note that RTTS with search of depth one would only reach the nearest neighbors, amounting to the *always exploit* strategy, with the small difference that all neighbors are considered

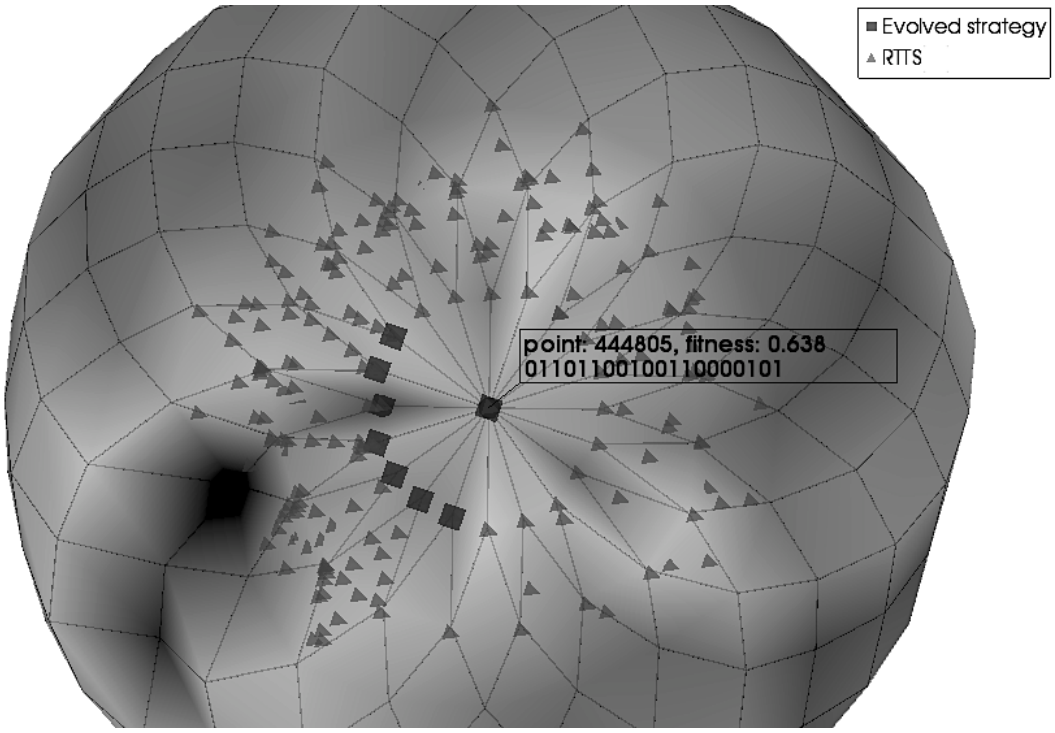


Figure 5.5: Point evaluations with RTTS and an evolved strategy. The environment is 20-dimensional and shown in a spherical visualization where elevation and brightness represent fitness, and distance from the center point approximates the Hamming distance from it. The 211 search points that the RTTS agent evaluates in a single time step in order to determine which action to take are shown as triangles. All of them are within two steps of the starting point for the search, shown at the center. In contrast, evolved strategies perform eight point evaluations on average (shown as squares), underscoring how different the CMAS strategies are from classical single-agent search methods.

instead of stopping at the first neighbor that improves over the current point. At depth two, however, RTTS is a distinctly different strategy from those considered so far; it is a traditional single-agent search method adapted to the CMAS setting.

The differences between RTTS and CMAS methods are clear in the results. The evolved agents as well as the manual CMAS strategies evaluate only eight points per step on average (Figure 5.5). Thus, they are significantly more economical than RTTS in a high-dimensional landscape (i.e. 26 times more in the sparse environments and seven times more

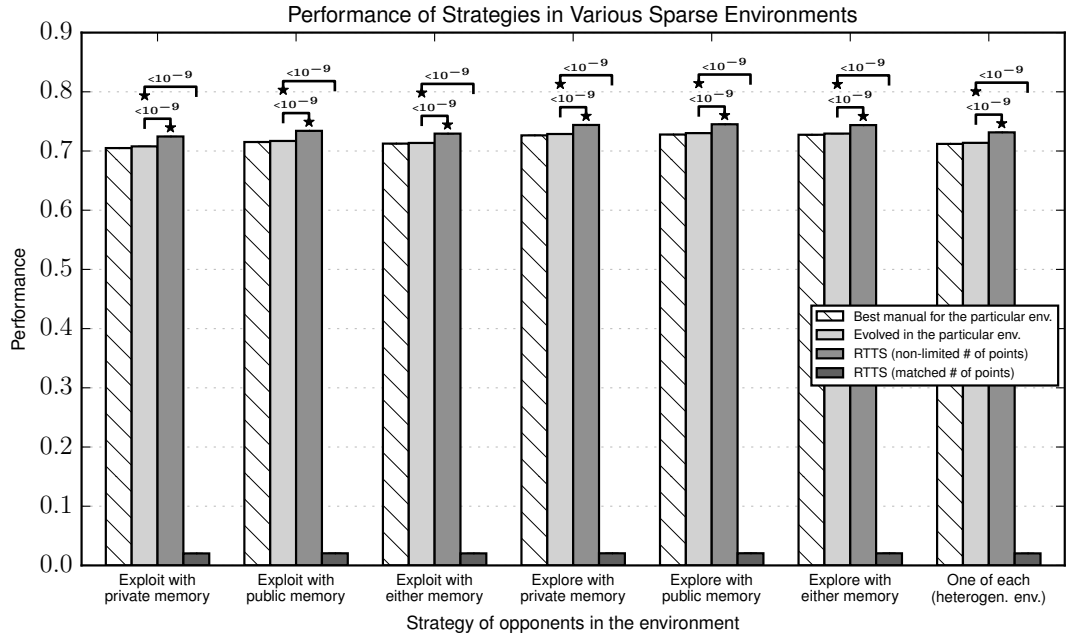
in the dense ones). To make RTTS more comparable with the CMAS methods, it is possible to match the number of points it considers with that of the CMAS methods, by limiting it to one action every 26 time steps in the sparse environments and every seven time steps in the dense ones. As can be seen in Figure 5.6, under such limited resources, RTTS makes very little progress. Whereas the CMAS methods are designed to proceed with the information gained from only a few points, RTTS expects to see the entire depth-2 search tree before making a decision.

Interestingly, even without the resource limitation, RTTS is still not better than the CMAS methods (Figure 5.6). The reason is that it is constantly mislead by the dynamic landscape: The fitness value of a point that looked promising during the look-ahead may diminish once the agent gets there, and it may miss points whose value increased.

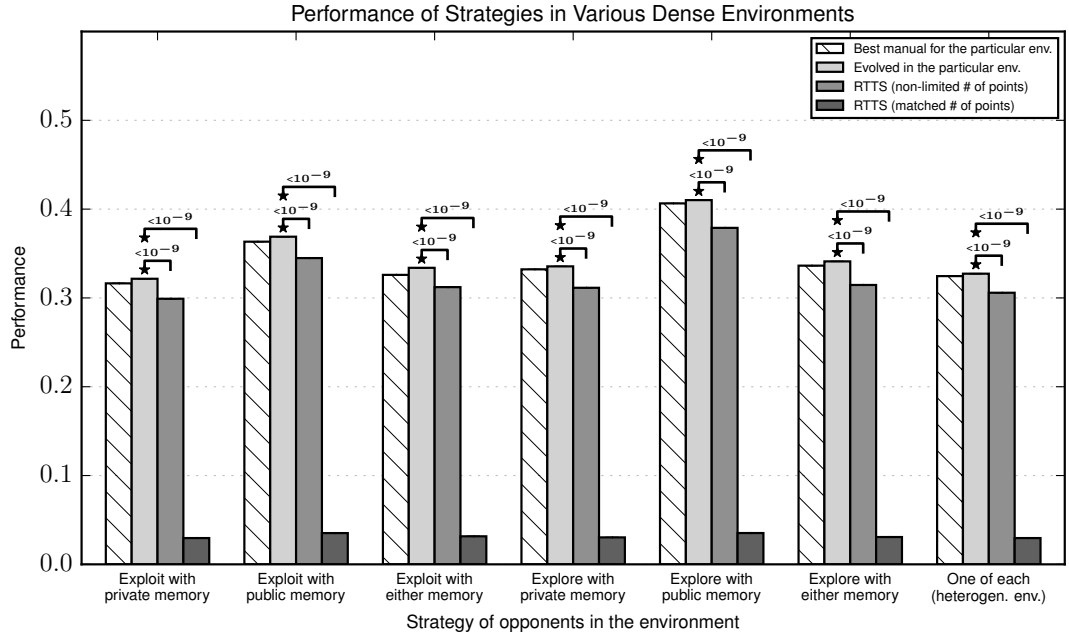
Thus, CMAS problems are different from classical single-agent search problems, and can be solved better by methods designed for such problems in mind, such as those described in this dissertation.

5.5 Conclusion

This chapter focused on the optimization of strategies for the abstract NK domain. First, a CPPN was described to encode strategies in a way that is easy and scalable to evolve. Then, customized strategies were evolved for homogeneous environments that have the hand-coded strategies from the previous chapter as opponents. Those evolved strategies performed better in their environment than strategies evolved in other environments. Furthermore, general strategies were evolved, which perform better in each environment than other strategies evolved in homogeneous environments, except for the strategy evolved specifically for that environment. Finally, evolved strategies were compared with a real-time tree-search algorithm, and were shown to perform better when the tree-search algorithm was limited to consider the same number of points as the evolved strategies. Moreover, in the dense environments, the evolved strategies were better than even the unlimited version of



(a) Sparse environments



(b) Dense environments

Figure 5.6: Performance of the RTSS strategy compared to the best manual strategy (Section 4.5) and the best evolved strategy (Section 5.2), in sparse ($N = 20$) and dense ($N = 10$) environments. The non-limited version was based on a complete 2-ply lookahead at each step, whereas the matched version evaluated the same number of points as the other methods. The non-limited version is comparable to the other methods, but the matched version is much worse. The assumptions of RTSS do not hold in CMAS problems, which thereby require a different approach.

the tree-search algorithm. Thus, this chapter showed that evolutionary computation is useful in an abstract CMAS domain and provided a number of insights on what works and when. A more concrete, real-world domain with human search agents will be described in the next chapter.

Chapter 6

A Concrete Domain: Social Innovation

To study competitive multi-agent search in a domain that is more concrete than the abstract NK domain of the previous chapters, a dataset of human behavior in a competitive multi-agent search task was employed: a social innovation game created by Wisdom et al. (2013). While the competition dynamics are not exactly the same as in the abstract simulation on NK fitness landscapes, the human study serves as a real-world example and application of competitive multi-agent search. This chapter describes the characteristics of this domain, as well as results of human subject experiments conducted by Wisdom et al. The discussion serves as a foundation for the next two chapters, which focus on modeling human behavior and optimizing strategies in this domain (Chapters 7 and 8, respectively).

6.1 The Game Domain

The human dataset was collected under laboratory conditions at the Percepts and Concepts Laboratory of Robert L. Goldstone at Indiana University[1]. The task that human subjects

[1] http://cognitrn.psych.indiana.edu/

performed was a multi-player problem-solving game similar to a fantasy sports league: The players tried to build a collection (or team) of icons that would score higher than the team of other players (Wisdom et al., 2013). The subjects, who were undergraduate students, were assigned to groups of size one through nine, in which they each played eight games.

A screenshot of the game's graphical user interface can be seen in Figure 6.1. During games that consisted of 24 rounds, each player built a team of five or six members (shown as *icons* on the interface) in each round. The players could add icons to their teams by dragging an icon from a source, and dropping it onto one of the player's own icons, replacing it. Icons could be copied in this manner from four sources: (1) player's last team (a *retaining* action), (2) player's best scoring team up until that point in the game (a *retrieval* action), (3) another player's last team (an *imitation* action), or (4) the league of all available icons (an *innovation* action). Depending on the game configuration, there were either 24 or 48 available icons from which the players could choose.

Besides individual icons, players were allowed to copy whole teams as well. To make such a copy, the player needed to drag the score label above the source team, and drop it onto their own team, replacing all icons in the team with those in the source team. This type of action constituted a higher level action compared to copying a single icon. If the player decided to copy multiple icons from a source, the team copy action saved them time, which is important, since players only have 10 seconds to act in each game round.

In each game, a fixed number of points was assigned to each icon in the league. Moreover, bonuses and penalties were assigned to a subset of distinct pairs of icons. These point assignments, which were not known by the human subjects and were determined by Wisdom et al. (2013) to make the game challenging by giving high bonuses to least valuable icons and high penalties to most valuable icons, are shown in Figure 6.2 (a) and (b) for two different game configurations. Each human subject played the first half of their game with the 24-icon configuration, and the second half with the 48-icon configuration. The score of a given team was calculated by adding the points for individual icons and any bonuses or

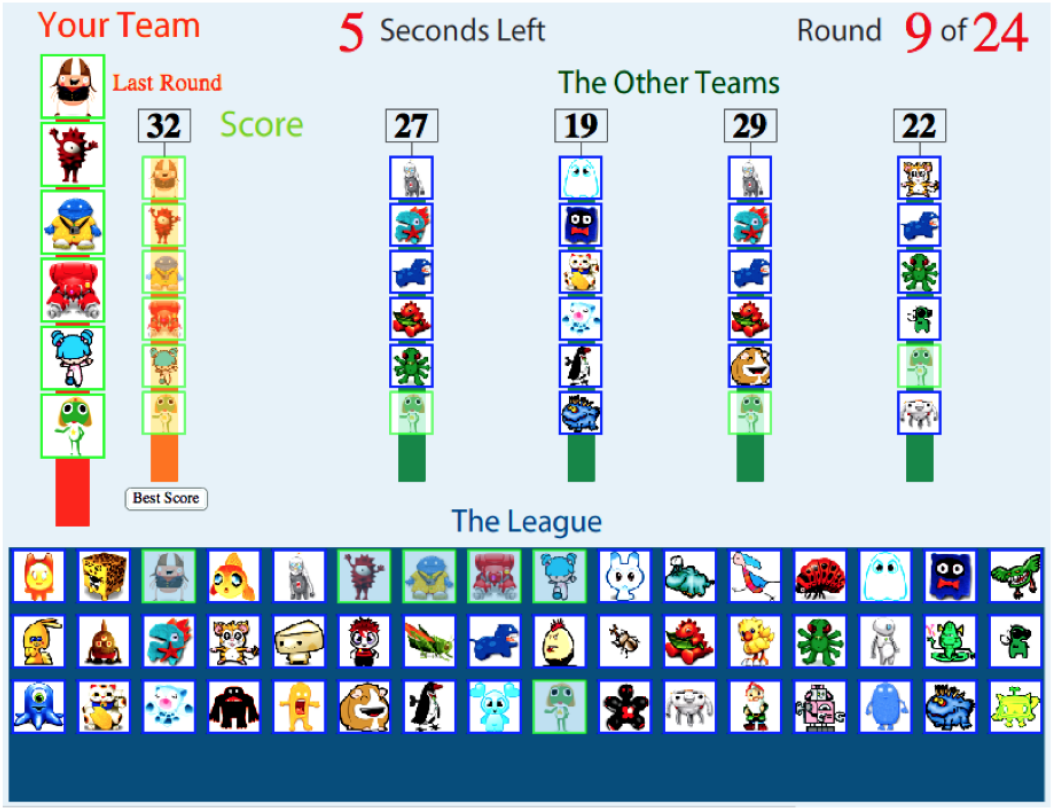
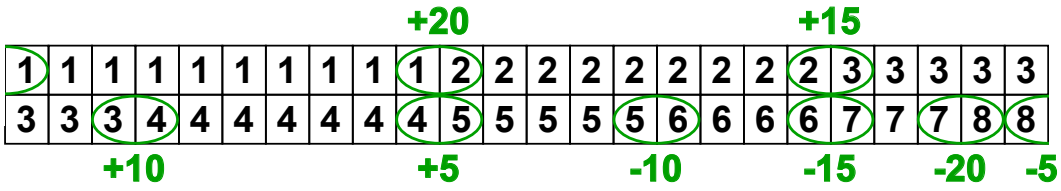


Figure 6.1: The graphical user interface for the social innovation game (Wisdom et al., 2013). The player's current team is shown at the left hand side (labeled "Your Team") and the opponents teams at top right (labeled "The Other Teams"). The objective is to build a team of icons that has a better score than the other players' teams. To build a team, a player can copy (i.e. drag and drop) icons from four sources: (1) player's own team in the last round, which is labeled "Last Round" and has its score shown above it, (2) the player's best team so far, which replaces the "Last Round" team on the user interface when the "Best Score" label is clicked, (3) any of the other players' teams, which are displayed to the right with their scores above them, and (4) the *league* (i.e. pool) of all available icons, shown at the bottom. The players can also copy any of the other teams in its entirety by dragging the score label above the team, and dropping onto their own current team. The user interface also displays the current round and the total number of rounds at the top right corner, and the remaining time in the current round at the top center. In this dissertation, it serves as a concrete domain on which to study how humans perform competitive multi-agent search. (Used with permission from Robert L. Goldstone.)

(a) League size of 24



(b) League size of 48

Figure 6.2: Points assigned to individual icons (boxes) and bonuses and penalties assigned to a subset of icon pairs (ovals) for the two game configurations with leagues of (a) 24 and (b) 48 icons.

penalties for icon pairs in the team. At the end of each round, subjects learned the score of their own team, as well as of others' teams, which they were able to use when deciding their teams for the next round. However, in the first round, where their teams were randomly assigned, they did not have the score information to make decisions. While the fixed point assignments for individual icons and icon pairs did not change from game to game, icon picture assignments were randomly shuffled before each game to prevent memorization of those point assignments from affecting subjects' strategies.

The next section gives a summary of the results from experiments done by Wisdom et al. in this domain.

6.2 Summary of Experimental Results with Human Subjects

Wisdom et al. (2013) conducted two experiments in the social innovation game. The first experiment explored which social learning strategies were employed by human subjects, and how the difficulty or complexity of the problem space (i.e. league and team size) affected those strategies. The second experiment evaluated how the visibility of the opponents' scores affected their strategies.

In the first experiment, Wisdom et al. identified several biases in human subjects' strategies with respect to solution payoff, icon frequency among players, and solution similarity. A bias toward greater potential payoff was apparent when imitation was employed. Of all teams that had imitated icons, 94.3% of the time the imitation source was a single player, and of those single-source imitations, 82% had the player with the highest score as the imitation source and 10.7% the player with the second highest score. Moreover, the score of the player who was the imitation source was higher than that of the imitating player in 89.6% of the cases.

Another bias the subjects had was toward popular icons. The probability of imitation or innovation for an icon was higher than chance levels only when the icon's choice frequency was above 0.5 (i.e. when a majority of other players possessed the icon). Moreover, subjects had a significant preference for icons whose choice frequency increased in the previous round (i.e. positive choice momentum). Subjects also had a small but significant bias toward imitating players whose teams were more similar to their own previous team than those of other players.

Other results demonstrated effects of game round on the subjects' behavior. The proportion of actions (i.e. icon sources) that the players employed changed somewhat with the round (Figure 6.3). As the game progressed, retention and retrieval increased slightly, while imitation and innovation decreased but overall the proportion of actions remained relatively stable. On average, subjects imitated 9.8%, innovated 13.7%, retained 73.9%, and retrieved 2.6% of the time. That is, subjects were in general conservative in their gameplay, but became even more so over the course of 24 rounds.

Scores of players also changed with round. They kept increasing until the end of the game across all groups sizes (i.e. one through nine; Figure 6.4 top), which means that they were able to adopt their strategies to the hidden point values. On the other hand, game round had the opposite effect on guess diversity, i.e. the proportion of icons that at least one player had in a group of a given size in a given round, normalized by an expected value

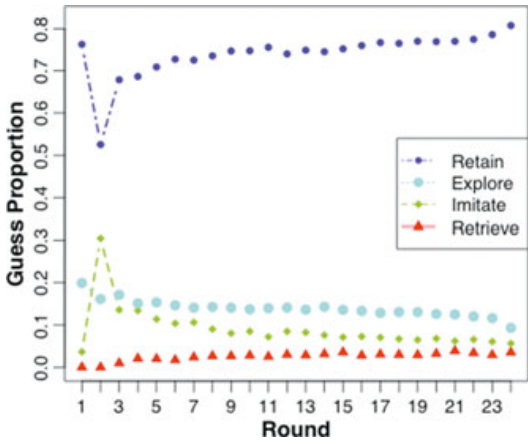


Figure 6.3: Changes in proportion of actions taken by human subjects over the course of 24 rounds. Retention was by far the most popular action, and the proportion of actions remained rather constant. Retention increased slightly with round, together with retrieval, whereas imitation and exploration (i.e. innovation) dropped. These observations suggest that the strategies they employed were rather stable, conservative, and became even more so over time. (Reproduced from (Wisdom et al., 2013) with permission. Copyright © 2013 Cognitive Science Society, Inc.)

for this proportion for random players in a group of the same size. There were a consistent gradual drop for all group sizes (except size one, where guess diversity is always 1.0 by definition; Figure 6.4 bottom), suggesting that they were discovering increasingly similar solutions over time.

Similar effects were observed with increasing group size: Scores generally increased, while guess diversity decreased. Group size also had an effect on action (or icon source) proportions: With larger groups, retention and imitation was higher, whereas innovation and retrieval was lower. Just like game round and group size, game order (i.e. which game out of the eight games for each human subject is being played) increased score and reduced guess diversity.

To see how choice source strategy affected score, Wisdom et al. grouped human subjects by which action (i.e. icon source, if any) they chose with a probability that is at least one standard deviation above the mean action choice proportion across all sub-

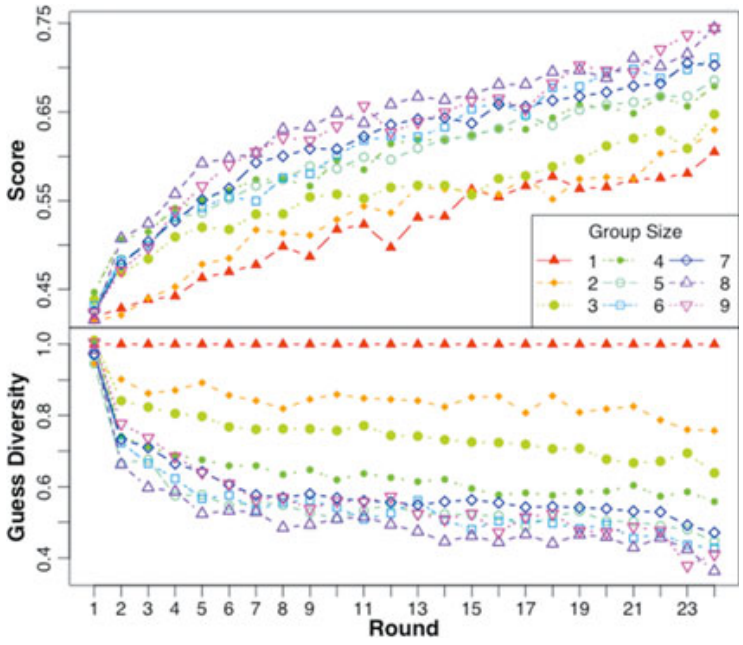


Figure 6.4: Over the course of 24 rounds, scores of human subjects increased for all group sizes, whereas the diversity of their icons decreased (except for groups of size one, for which guess diversity is always 1.0 by definition). These results suggest that the players were able to learn the underlying point values to some extent, and that they converged to similar solutions. (Reproduced from (Wisdom et al., 2013) with permission. Copyright © 2013 Cognitive Science Society, Inc.)

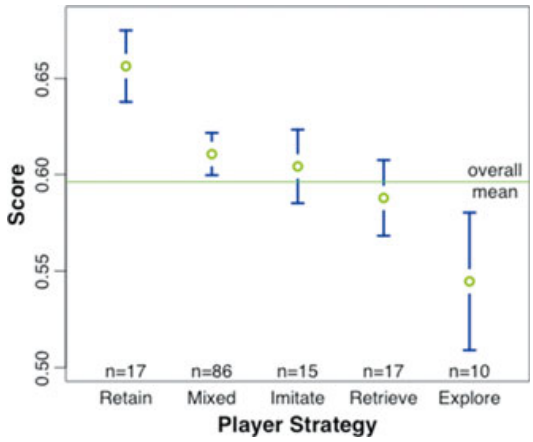


Figure 6.5: Scores of human subjects grouped by their choice of a dominant action. Subjects who chose to do imitation and retention dominantly scored better than those whose dominant action was retrieval or exploration (i.e. innovation). (Reproduced from (Wisdom et al., 2013) with permission. Copyright © 2013 Cognitive Science Society, Inc.)

jects. Subjects for which no such action existed were considered in a separate *mixed* group. Figure 6.5 shows that the retain-dominant group scored the highest, with the mixed and imitate-dominant groups above average, while dominantly retrieving subjects scored below average, and innovate-dominant subjects scored the lowest.

Wisdom et al. also investigated the effects of game difficulty by utilizing the two game configurations mentioned in Section 6.1: one with a 24-icon league and five-icon teams, and another with a 48-icon league and six-icon teams. With the more difficult configuration, scores, guess diversity, and proportion of innovation was lower, whereas retention was higher.

The second experiment of Wisdom et al. evaluated whether knowledge of the opponents scores affected the players' strategies. They found that when subjects could not see the scores of other players, their average scores and final scores were lower, and their guess diversity was higher, compared to when scores were visible. Moreover, when the scores were not visible, the bias for icon frequency was in the opposite direction to that with visible scores: There was a preference for less popular icons when subjects were imitating or

innovating. While these results are interesting and can be incorporated in the CMAS model, the experiments in this dissertation focused on the visible case. Therefore, only data from the first human subject experiment will be utilized, as is described in the next section.

6.3 Data Collected in Human Subject Experiments

The data contained information about 39 game sessions such as number of participants and date. For each of the eight games in each session, the data described the game configuration: the values of individual icons and bonus or penalty values for icon pairs for each game, the size of players' teams, and the size of the league of all available icons.

The rest of the data consisted of the details of the gameplay. For each player in each round of each game, the player's score, normalized score (based on the minimum and maximum possible score with the game's configuration), and the IDs of each icon in the player's team, as well as each icon's source were included. The icon source corresponded to the four icon actions, and was the source player's ID in the case of imitation. Even though the graphical user interface of the game allowed players to copy whole teams from another player or from their own past teams, the data did not contain such team-level actions.

This data is detailed enough so that neural network models can be trained to imitate each individual player in the dataset. These models can then be used to create computational environments where the strengths and weaknesses of the different strategies can be characterized systematically, their effectiveness measured quantitatively, and optimal strategies discovered.

6.4 Conclusion

This chapter provided details for a concrete domain for social innovation in the form of a multi-player game. Experimental results from a study involving human subjects playing this game were summarized, which identified several biases, related to solution payoff, icon

popularity, and solution similarity. Other results in the same study showed effects of group size, game order, and current game round on score, guess diversity, and action proportions, as well as effects of dominantly preferred actions over score. These characterizations of human gameplay will be used in the next chapter to evaluate how good the learned models of human players are. The data collected from the human subjects' games, which included the individual team choices of all players, were also described. This dataset will be utilized in the next chapter to train models of individual human players. These models will be in turn used in Chapter 8 to demonstrate that evolution can be used to discover optimal strategies for this domain.

Chapter 7

Characterization of a Concrete Domain: Social Innovation

The human subject data described in the previous chapter provide a foundation for studying competitive multi-agent search in the real world. The data will be used to build neural network models of individual subjects in the social innovation game in this chapter. They will then be simulated extensively to characterize what works in that game, including discovering optimal strategies automatically through neuroevolution.

Motivation for modeling human subjects in the social innovation domain is given in Section 7.1, while Section 7.2 introduces a suitable neural network architecture for the models, which is trained with the data described in Section 7.3. The trained models are evaluated using several distance objectives as summarized in Section 7.4. Two experiments are carried out to determine the best modeling approach (Section 7.5) and to find out the effects of training time on the trained models (Section 7.6). The result is a set of player models that will be used in the next chapter to characterize the human subjects and to determine optimal strategies in this domain.

7.1 Motivation

The social innovation domain described in the previous chapter, which is based on a human game, provides a laboratory setting in which competitive multi-agent search in humans can be characterized. Although the experiments of Wisdom et al. (2013) describe human behavior in this game well qualitatively, more data is necessary to characterize the strategies they use quantitatively. One way to do it is to run many subjects and repetitions. The time and cost of such experiments is prohibitive, however. An alternative taken in this chapter is to use the existing data to construct computational models of the human subjects, and then run the extensive experiments with the models.

There are two advantages to building human models. First, it is much more practical to place models of human subjects in hypothetical situations than doing the same with the subjects themselves. Time and cost become significant obstacles in experiments, which using models can easily overcome. It is also much easier and faster to optimize strategies against models of human subjects than against the human subjects.

Secondly, the dataset for human subjects' gameplay is limited. Using a model of human subjects, one can collect gameplay data from a larger number of games in order to analyze the behaviors of the corresponding human subjects. For instance, hypothetical situations that do not exist in the human subject groups in the dataset can be simulated using such models, which would allow experiments that cannot be done otherwise.

7.2 Two-tiered Neural Network Architecture

As described in Section 6.1, the user interface for the human subjects permitted two levels of action: replacing the player's whole team and replacing individual icons (i.e. team members). When a player performs a team-level action, changes due to the player's earlier actions in the same round are reverted, since replacing the whole team overwrites all team members. Therefore, any action taken before the last team-level action (if any) in a round

has no effect on the player's score for that round. Moreover, any action taken on an icon reverts the effects of other icon-level actions on the same icon, since it overwrites the corresponding team member. So, for each icon, only the last icon-level action performed on that icon in each round has an effect on the player's score for that round. Thus, for each player and round, the player's last team replacement (if any) and the last replacement of each team member following that team replacement (if any) is sufficient to calculate the player's score at the end of that round.

Therefore, a strategy that makes at most one team replacement, followed by at most one replacement for each team member is sufficient to model a player's strategy of picking a team in each round. Accordingly, a two-tiered artificial neural network model was created to model a player: The outputs of two separate neural networks were interpreted as high-level (i.e. team-level) and low-level (i.e. icon-level) actions (Figure 7.1).

For the team action network, two features were used to specify the team-level game state for a given player: (1) elapsed game time (i.e. current round's number), which allows for behavioral differences between beginning of the game, mid-game, and end-game, and (2) the player's score relative to the highest score among the opponents, which encodes how well the player is doing in the current game.

For the icon action network, two features were used to encode the icon-level state for a given player and one of its icons: (1) the icon's popularity across all players (i.e. how many players have that icon), and (2) the icon's age (i.e. how many rounds the icon has been kept by the player without being replaced). Since the team-level state is also useful in deciding which actions to take on individual icons, the two inputs for the team-level state were also provided to the icon action network. All team- and icon-level state feature values were normalized to the [-1,1] range before being provided to the networks as inputs.

There were three different sources from which a whole team can be copied: (1) the player's best team so far, (2) another player's team from the last round, and (3) the player's own team from the last round. Since each player starts each round with their team from the

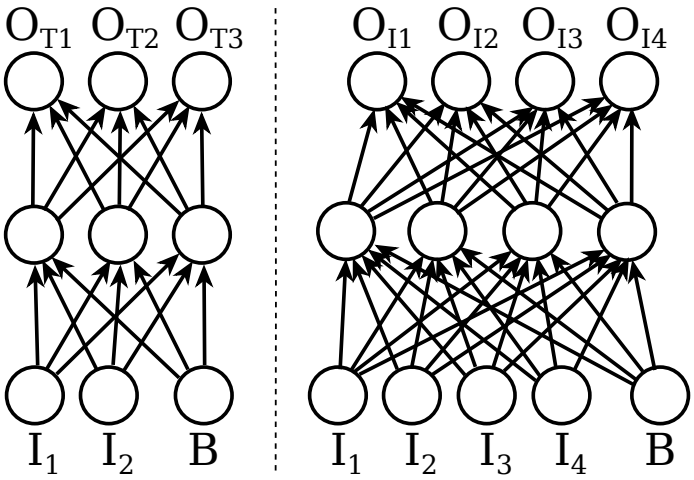


Figure 7.1: Human subject models consist of two separate neural networks: (1) a network with two inputs (i.e. current game round and player's score relative to the best opponent) and three outputs (i.e. imitation, retrieval, and no action) for team actions, and another network with four inputs (i.e. current game round, player's score relative to the best opponent, icon's popularity, and icon's age) and four outputs (i.e. imitation, innovation, retrieval, and retention or no action) for icon actions. Both networks have a bias input, and the same number of hidden nodes as their output nodes. The icon action network is run for each icon of the player's team in each round. The network outputs are normalized and used as action probabilities for that icon, while the team action network is run only once per round with the outputs used as team action probabilities after normalization. In this manner, the network for icons is shared among all icons of the player. This approach allows more training data to be used for the icon network than would be available with six separate networks for each icon position.

last round, picking the third source effectively cancels the changes made so far. Therefore, there are effectively three possibilities for team-level actions: copying own best team, copying an opponent's team, and not doing any team copying. These actions correspond to the three outputs of the team action network in Figure 7.1. Those network outputs are normalized, and then interpreted as the probability of performing the corresponding team actions. That is, the agent chooses the action in each round probabilistically among those actions, rather than deterministically picking the action with the maximum output activation.

At the icon level, players can copy icons from four different sources: (1) another

player's team from the last round, (2) the pool of all icons, (3) the player's own team from the last round, and (4) the player's best team so far. These four sources correspond to the four actions mentioned in Section 6.1: imitation, innovation, retrieval, and retention, and map to the four outputs in the icon action network in Figure 7.1. As with the team action network, the icon action network's outputs were normalized, and used as icon action probabilities. However, unlike the team action network, the icon action network needs to be run for each icon separately, utilizing its two icon-specific inputs. It will then generate an icon-specific decision: which action to take for that particular icon. On the other hand, in both team and icon action networks, the number of hidden nodes was chosen to be the same as the number of outputs (i.e. three and four, respectively).

The next section will describe the training data used in this task. The two networks are trained to model individual human subjects using the supervised backpropagation training method.

7.3 Training Data

The training data for constructing the models consisted of input-output pairs for the team and icon action networks. These pairs were collected from the dataset of gameplay among human subjects. Only the games with eight or nine human subjects were included, because games with fewer players had lower scores, and including them would make comparing scores with models less straightforward. For each human group that was included, only the last game was used for training models because the last game had the highest score on average (as mentioned in Section 6.2).

Team actions taken during the human subjects' games were not recorded in the gameplay data. Therefore, they had to be inferred from the data by considering individual icon actions' sources, which were included in the gameplay data. If a majority of a subject's icons in a round came from the same source, then the subject was assumed to have copied the whole team from that source, and changed the remaining minority of the icons (if any)

afterward. In that case, those majority icons were considered to be retained after the team action, which became their icon action in the training data for that round.

The values for these team- and icon-level state features were calculated for each round, and paired with the source choices for the player's team and icon actions in that round. Thus, the number of input-output pairs for the team action network was equal to the number of rounds, whereas the icon action network had six times as many pairs, since teams had six icons. The input-output pairs for team and icon actions were then used to train models, which were evaluated with the distance objectives described in the next section.

7.4 Distance Objectives

The more similar a model's behavior is to a given human subject, the more useful that model will be as a replacement for that subject. In order to quantify this similarity, several distance objectives were used to measure the difference between the behaviors of the model and the human subject on whose gameplay data the model was trained.

For each model, 200 games were simulated where one of the players was the model. To make comparisons of different sets of models fair, a consistent set of opponents was used in these games. They consisted of simplistic models with fixed action probabilities derived from the human gameplay data. Each of the following distance objectives was averaged over all games:

- **Normalized absolute score**: Score of the evaluated player normalized to the [0, 1] range, averaged across rounds.
- **Normalized relative score**: Difference in normalized absolute score between the evaluated player and its best opponent, averaged across rounds.
- **Team imitation ratio**: Ratio of the number of copying of an opponent's whole team over the number of rounds.

- **Icon imitation ratio**: Ratio of icons that were individually copied from an opponent.
- **Innovation ratio**: Ratio of icons that were copied from the pool of all available icons.
- **Team retrieval ratio**: Ratio of the number of copying of the player's own best past team over the number of rounds.
- **Icon retrieval ratio**: Ratio of icons that were individually copied from the player's own best past team.
- **Retention ratio**: Ratio of icons that were not changed.
- **Icon consistency**: Number of rounds icons were kept in a player's team, averaged across all uses of icons by the player.

In two experiments, models were trained for each player and evaluated according to these distance objectives.

7.5 Experiment 1: Comparison of Modeling Approaches

The input-output pairs generated from gameplay data for team and icon actions were used to train the neural networks described in Section 7.2 with the goal of modeling each human subjects' behavior as close as possible. Since the actual actions taken by the players are known, supervised learning techniques could be used. The standard technique is backpropagation, and it was found to be sufficiently powerful in this task.

The two-tiered neural network models were compared to the corresponding models created with two other modeling approaches based on human gameplay data, as well as a baseline model with uniform action choice. More specifically, four models were thus compared: (1) the baseline model where icon actions are picked based on fixed and uniform probabilities (i.e. 0.25 for each of the four icon actions), (2) another simplistic model with fixed probabilities, but with each action's probability set as the human subject's usage

ratio for that action, calculated from the data, (3) a one-tiered neural network model that performs icon actions but not team actions, and (4) the two-tiered neural network model, which performs both team and icon actions.

The training data format for the one-tiered network models was similar to that for the two-tiered ones, except that there were no team actions, and therefore no icons were retained due to an earlier team action in the current round. Both types of neural network models were trained with backpropagation using 500 epochs, using a learning rate of 0.1, momentum of 0.1, and weight decay of 0.01. All compared models chose actions probabilistically based on the relative activation of the output units.

The results are shown in Figure 7.2, which compares the four models on average over all 34 subjects, based on the distance objectives from the previous section (smaller values mean that the models are closer to human subjects). Since the baseline model is not based on the data, it was significantly outperformed by the three other model types in six out of nine objectives (p-value $< 10^{-5}$). Similarly, the second model type (i.e. the one with fixed action probabilities derived from the gameplay data) was significantly worse than the two-tiered neural network models (p-value = 0.029) and somewhat worse than the one-tiered neural network models (p-value = 0.075) in the absolute score objective, and somewhat worse than both types of neural network models in the relative score objective (p-value = 0.087 and 0.076 for one- and two-tiered models, respectively). However, the second model was the best one for the innovation and retention ratio objectives, which is not surprising since that model was based on the action ratios from the human subjects.

While there was no significant difference between the two neural networks in innovation ratio, icon consistency, or the two score objectives, they did differ in three other distance objectives, including team and icon imitation ratio (i.e. first and second on the second row in Figure 7.2). The two-tiered models were significantly closer to human subjects along these objectives than the other model types (p-value $< 10^{-4}$). The main reason for the difference in team imitation ratio is that the other models did not perform any team ac-

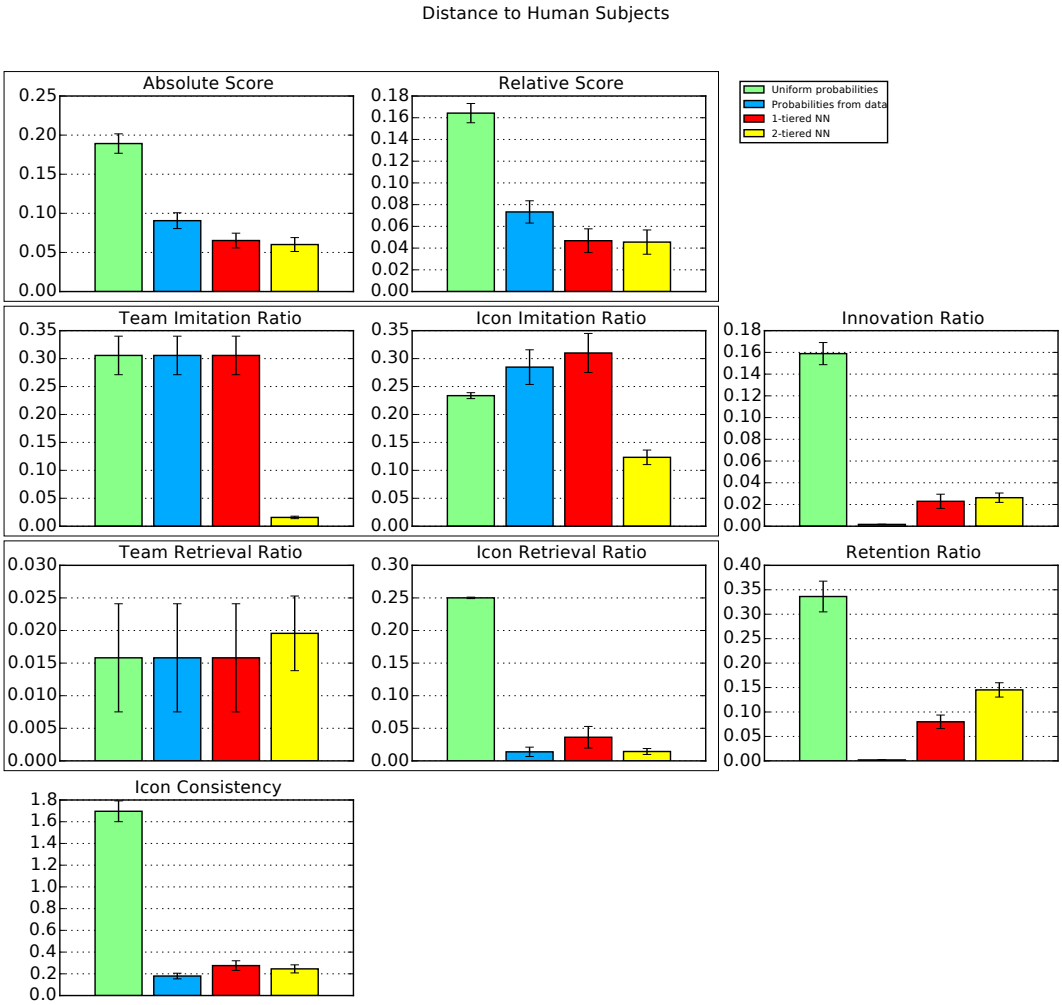


Figure 7.2: Different modeling approaches compared using several distance objectives. Values show average distance of the models to the corresponding human subject. Smaller values are better. Each distance objective is averaged across the human subjects. Error bars indicate one unit of standard error of the mean. Note that the y-axes have different scales. Related objectives are grouped in surrounding boxes. The two-tiered neural network models do significantly better than all others in team and icon imitation ratio objectives, whereas the difference with the one-tiered networks are not significant in other objectives. The reason why two-tiered network models do not do better in retrieval ratio objectives is that retrieval is done very rarely by humans, and therefore insufficient data for team and icon retrieval actions. The simpler models (i.e. second bar) that are based on probabilities from human gameplay data perform better in innovation and retention ratio objectives, but they cannot perform team and icon actions separately, unlike the two-tiered network models.

tions, and therefore got the same result on team action ratio objectives (i.e. "team imitation ratio" and "team retrieval ratio"). However, unlike in the team imitation ratio, two-tiered models did not perform significantly better in the team retrieval ratio. A likely reason is that human subjects rarely performed that action: 27 out of 34 subjects did not do team retrieval at all, while the others did very rarely. The two-tiered models learned to rarely perform that action, which was not significantly different from not doing it at all.

While the two-tiered models were significantly worse in the retention ratio (p-value < 0.01), the advantage of two-tiered network models in the two imitation action ratio objectives and in their support for team actions make the two-tiered network models a better choice for modeling human subjects than the one-tiered ones.

After choosing a neural network model architecture, there is still the question of how long the training should be, which is the focus of the next section.

7.6 Experiment 2: Effect of Number of Training Epochs

Training for too long may overfit the models to the training data, which reduces the quality of generalization to unseen situations. Techniques such as early stopping and cross-validation based on RMS error are commonly used in machine learning to avoid overfitting. However, they do not take into account other objectives that might be of importance, such as the distance objectives described above. Indeed, in preliminary experiments using cross-validation, the resulting sets of models did not show an advantage over the two-tiered network models in any of the distance objectives, and performed worse in some of them. Therefore, a comprehensive experiment was done to determine the effect of training duration on the various distance objectives.

Figure 7.2 shows the distance to human subjects using the same objectives as in the previous section, where the number of training epochs was varied from 50 to 1000 in increments of 50. Several objectives were not significantly affected by training duration, such as "absolute score", "relative score", "icon consistency", and "icon retrieval ratio".

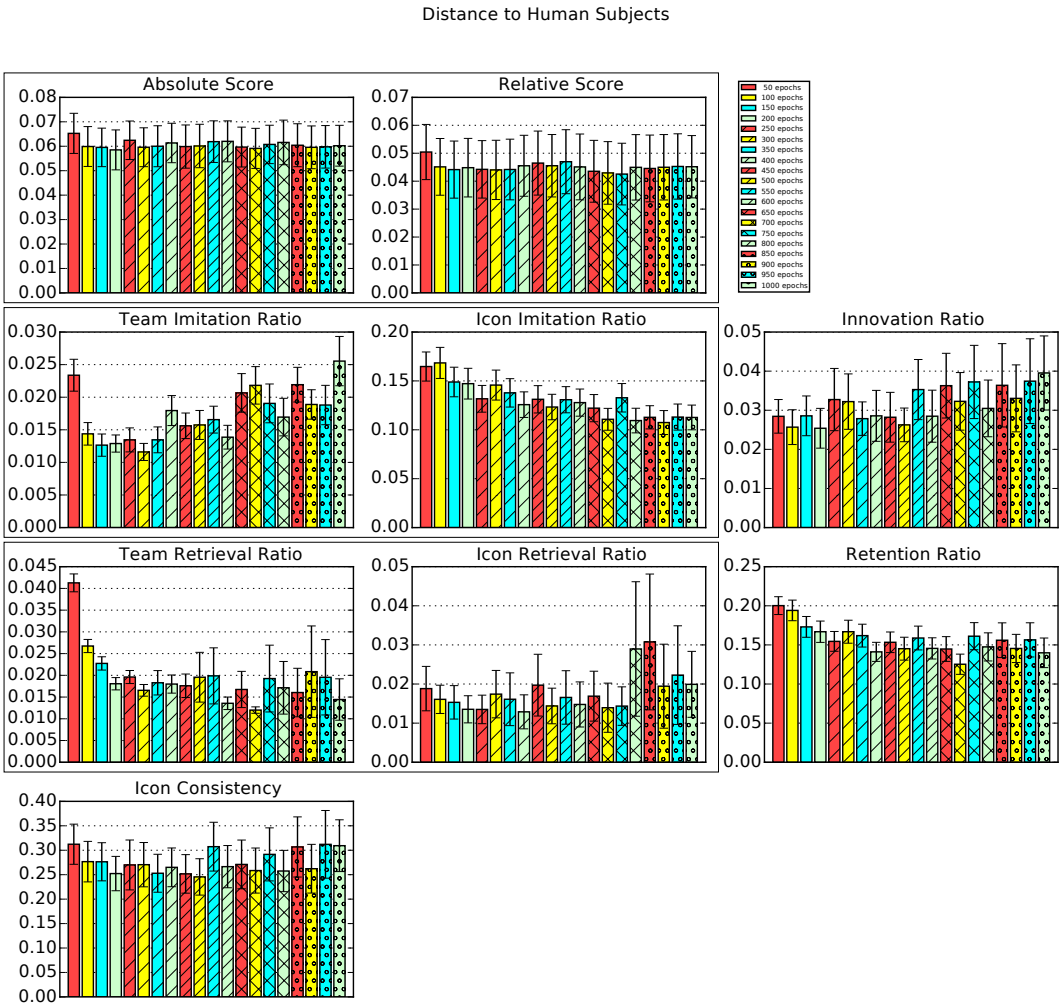


Figure 7.3: Effect of number of training epochs on the distance objectives. Smaller values are better. Each distance objective is averaged across the human subjects. Error bars indicate one unit of standard error of the mean. Note that the y-axes have different scales. Objectives such as "team imitation ratio" and "innovation ratio" showed an upward trend with increased training, whereas "icon imitation ratio", "team retrieval ratio" , and "retention ratio" went down. The remaining distance objectives were not affected significantly by the length of training. Therefore, a moderate number of training epochs is a good compromise given these distance objectives.

On the other hand, "team imitation ratio" and "innovation ratio" showed an overall increase with increasing number of training epochs, whereas "icon imitation ratio", "team retrieval ratio", and "retention ratio" objectives had the opposite trend.

While it is difficult to pick a best model due to the multiple objectives, based on the trade-off that stems from the opposite trends in the objective values, the set of models trained with a moderate number of epochs, such as 500, can be considered as a good compromise. Therefore, that set of models was used in the next chapter as opponents against which the strategies were optimized.

7.7 Conclusion

This chapter dealt with the question of how to best model human subjects in the domain of a social innovation game. It introduced several distance objectives to quantify the similarity of a given model to the human subject on which it was modeled. The first of two experiments investigated which modeling approach is best in terms of those objectives, and found that models with two-tiered neural network were better than one-tiered network models and models based on fixed action probabilities derived from the gameplay data. The second experiment examined the effect of training time (i.e. number of training epochs) on the resulting models, again, using the same distance objectives, and discovered a trade-off between two subsets of the considered objectives with opposite trends.

While modeling human subjects is useful in its own right, it does not provide a clear understanding of how good human subjects really are. What strategies perform best in this domain, and are they different from those that the humans employ? The next chapter aims to shed light on this question by developing strategies that are optimized for this domain.

Chapter 8

Optimization of Social Innovation

Whereas the goal of the previous chapter was to understand and model human behavior in the social innovation game, the goal of this chapter is to find out whether strategies exist that are better than those that humans use. Moreover, the goal is to identify what works best against different types of opponents. For instance, do we need different strategies against opponents that imitate a lot, and against those that do more innovation to perform as well as possible? Also, can we find strategies that generalize to some extent? This chapter attempts to answer such questions using evolution as the optimization method. The network architecture for Chapter 7 was first simplified so that it could be more easily evolved. Two experiments were then conducted: First, in order to characterize environments and successful strategies, strategies were evolved against a uniform set of opponents of a certain type. Second, in order to show that better strategies can be discovered in complex real-world environments, they were evolved in the game groups of the human subject study.

Section 8.1 describes the network architecture that is used to represent strategies. This architecture is optimized via evolution, as detailed in Section 8.2. The experimental results for evolving strategies against a single opponent type and evolving general strategies are analyzed in Sections 8.3 and 8.4, respectively. Section 8.5 describes the results from an experiment with an evolutionary setup that has more realistic set of opponents, taken from

the game groups of the subjects in the human study.

8.1 Two-tiered Combined Neural Network

In the previous chapter, two separate neural networks were used to model human subjects. However, if such a model was used for evolving strategies, hidden network nodes that represent useful states would have to evolve separately twice (i.e. once per neural network). For instance, a hidden node may represent the state where the game is about to end and the relative score of the agent is lower than the player with the highest score, and the output of this hidden node may be useful for both high-level and low-level actions. Sharing such hidden nodes between the two network tiers would allow neural network representations to be evolved once and used for both high-level and low-level actions. Therefore, employing a combined neural network can reduce the search time for a neural network solution.

Thus, for the goal of optimizing strategies, a combined two-tiered neural network architecture was chosen to represent strategies. As in the CPPNs evolved for the abstract domain in Sections 5.2 and 5.3, the neural network nodes had either a sigmoid or a Gaussian function as their activation function. However, unlike the CPPNs in the abstract domain, the networks were not used to generate a pattern. As seen in Figure 8.1, high-level (i.e. team-level) actions and low-level (i.e. icon-level) actions were generated by separate outputs of the same neural network, as opposed to outputs of two separate neural networks as in Section 7.2. The next section details how these networks were optimized for the social innovation game domain.

8.2 Experimental Setup

As was done in Chapter 5, the strategy for a single agent was optimized given an environment with a fixed set of opponents. As described in the previous section, a single neural network was used to represent a strategy, which made it straightforward to, again, use NEAT

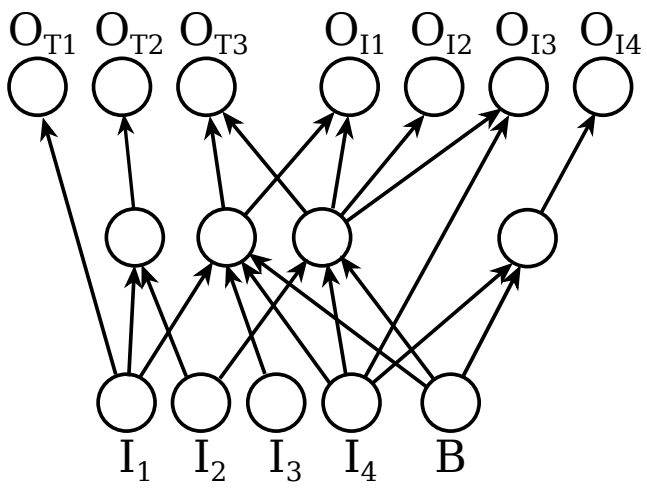


Figure 8.1: An example combined two-tiered neural network, with four inputs (i.e. current game round, player's score relative to the best opponent, icon's popularity, icon's age), three outputs for team actions (i.e. imitation, retrieval, and no action), and four outputs for icon actions (i.e. imitation, innovation, retrieval, and retention or no action). The outputs for icon actions are normalized and used as action probabilities for each icon of the player's team, while team action outputs are normalized separately and used as team action probabilities. In this design, the team and icon networks are combined, as opposed to the separate networks in Figure 7.1 Hidden nodes representing game states that are useful for both team and icon actions are shared, and do not have to be evolved twice.

to optimize it.

Strategies were evolved in environments with eight opponents, each of which was the model of a human subject, created using supervised learning as described in Chapter 7. Just as in Chapter 5, NEAT ran for 500 generations for each environment, with a population of size 100, and with 64 repetitions for each evolutionary setup.

The fitness of each evolved strategy was evaluated by running a simulation where the first agent used the evolved strategy and the opponents used a fixed strategy chosen to create a particular environment. The fitness was calculated as the normalized score of the first agent averaged across 200 games. As in human subjects' games, at the beginning of each game, each player had a team that was randomly selected from the league. Moreover, as in human games, icon memorization was effectively prevented, since agents did not

guess or keep scores for individual icons or pairs of icons. The next section describes the first evolutionary experiments conducted with this approach, where the environment consisted of uniform opponents of a certain type. Section 8.4 then extends to heterogeneous opponents.

8.3 Experiment 1: Evolving Specific Strategies

In order to see which strategies work well in different social innovation game environments, several evolutionary setups were created, each with a different set of opponent models. For each of the four icon actions (i.e. imitate, innovate, retain, and retrieve), the model for the subject that used that action the most was selected as the dominant model for that action, among the models for subjects who took part in games with at least five players. For each action, eight copies of the corresponding action-dominant model were used as the opponents in an environment, resulting in four homogeneous environments, similar to the homogeneous environments of Section 5.2.

To make it easier to examine evolved strategies, the corresponding networks were used as CPPNs to generate patterns to fill state-action probability tables, similar to the tables introduced in Section 3.7 for the abstract NK domain. The possible range of input values for the four CPPN inputs were discretized into three coarse values for the time input, and two for the remaining inputs: (1) beginning of game, mid-game, or end-game; (2) low or high score relative to the best opponent; (3) low or high icon popularity among players; and (4) old or new icon. Combinations of these input values correspond to 24 possible discrete game states for each icon. For each state, the agent had a choice of three team actions for the whole team, as well as four icon actions for each icon. As a result, from each evolved CPPN, a 24×7 state-action probability table was obtained.

To identify the differences between strategies evolved in different environments, action probabilities were compared using Student's T-test, separately for each action and for each of the 24 discrete states. This comparison resulted in 24 p-values for each pair

of strategies median p-value was then used to determine whether strategies evolved in an environment had significantly higher or lower probabilities than those evolved in another environment.

Strategies evolved in the environment with imitate-dominant opponents do significantly more innovation (with median p-value $< 10^{-3}$) and less imitation (with median p-value for icon and team imitation 0.0134 and 0.0454, respectively), as well as more retrieval (with median p-value 0.0287). In such an environment, the source of imitation is always the highest-scoring player. Therefore, lower imitation and higher innovation and retrieval lead to higher diversity, which increases the chance of finding icons with higher score, beating the opponents.

On the other hand, strategies evolved with innovation-dominant opponents had significantly higher imitation at the team level (median p-value $= 0.022$) compared to strategies from other homogeneous environments, while there was no significant difference in icon imitation. A possible explanation is that when all other teams are innovating a lot (i.e. replacing their icons with random ones from the pool), team imitation is more reliable than icon imitation. Strategies evolved against retention-dominant opponents have increased team imitation as well, but it is not as significant (median p-value $= 0.066$).

With retrieve-dominant opponents, evolved strategies innovated significantly less (median p-value $< 10^{-4}$) than in other environments, and imitated somewhat more at the icon level (median p-value $= 0.094$). Innovation is getting a completely random icon, whereas icon imitation copies a random icon from the best-scoring opponent's team, which, in this environment, opponents are likely to have icons from the opponent's best team so far. Therefore, in this environment innovation is less likely to contribute positively to a player's score than imitation.

The 24×7 action probability tables were also used to examine the distribution of strategies. Similar to the approach in Section 5.2, the probability tables of evolved strategies were treated as 168-dimensional vectors, and reduced to two-dimensional points via PCA.

The first two components captured 83% of the variance in the high-dimensional vectors, and therefore two-dimensional PCA plots were used to visualize the strategies. Table 8.1 shows the distribution of the best strategies from the 64 evolutionary runs for each environment.

The diagonal line at the left of the PCA plots represents maximum icon imitation (i.e. with probability 1.0), and the one at the right, corresponds to maximum team imitation; therefore there are no points below those two lines. The result that the strategies evolved in imitation-dominant opponents imitated less means that the points (i.e. strategies) are further away from the diagonals than in the other environments. The relatively high team imitation evolved against innovation- and retention-dominant opponents can also be observed in these plots: there is a strong cluster around the diagonal line at the right representing maximum team imitation in these environments. Similarly, for strategies evolved against retrieve-dominant opponents, relatively more numerous points can be seen near the diagonal at the left, which corresponds to higher icon imitation.

Performance of the evolved strategies across various environments is shown in Figure 8.2. For each homogeneous environment, the strategies evolved in the same particular environment significantly outperformed the ones evolved in other environments (p-value $< 10^{-19}$). This result is similar to that of Figure 5.4 and shows that there is indeed value in optimizing the strategies for a particular environment. Strategies deployed in foreign environments performed at similar levels with each other, except for the ones evolved against imitate-dominant opponents, which performed significantly worse than the others (p-value $< 10^{-4}$). A possible explanation is that the increased innovation is actually a handicap in those other environments.

While homogeneous environments are useful in evolving strategies to counter a specific type of opponent, a different approach is needed for evolving general-purpose strategies that can be used against different opponents, which will be the focus of the next section.

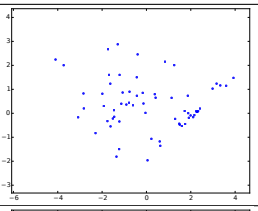
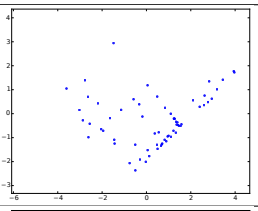
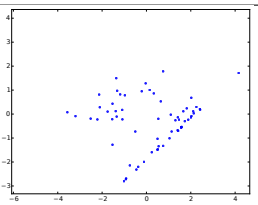
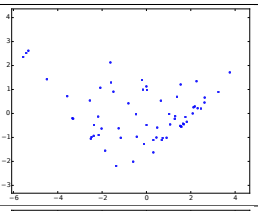
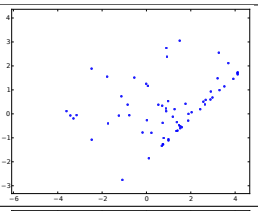
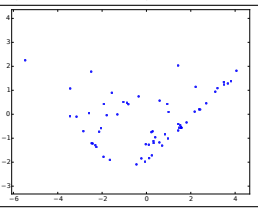
Environment		PCA for All Evolved Strategies
Id	**Opponents**	
1	Imitate-dominant (8)	
2	Innovate-dominant (8)	
3	Retain-dominant (8)	
4	Retrieve-dominant (8)	
5	Heterogeneous (2 opponents from each of Environments 1-4 above)	
6	Multiple homogeneous (Fitness averaged over Environments 1-4 above)	

Table 8.1: Strategies evolved in homogenous and heterogeneous environments. Each PCA plot shows 64 evolved strategies. The PCA dimensions are shared across all evolved strategies. The diagonal boundary lines at the bottom left and bottom right represent maximum icon imitation and maximum team imitation, respectively.

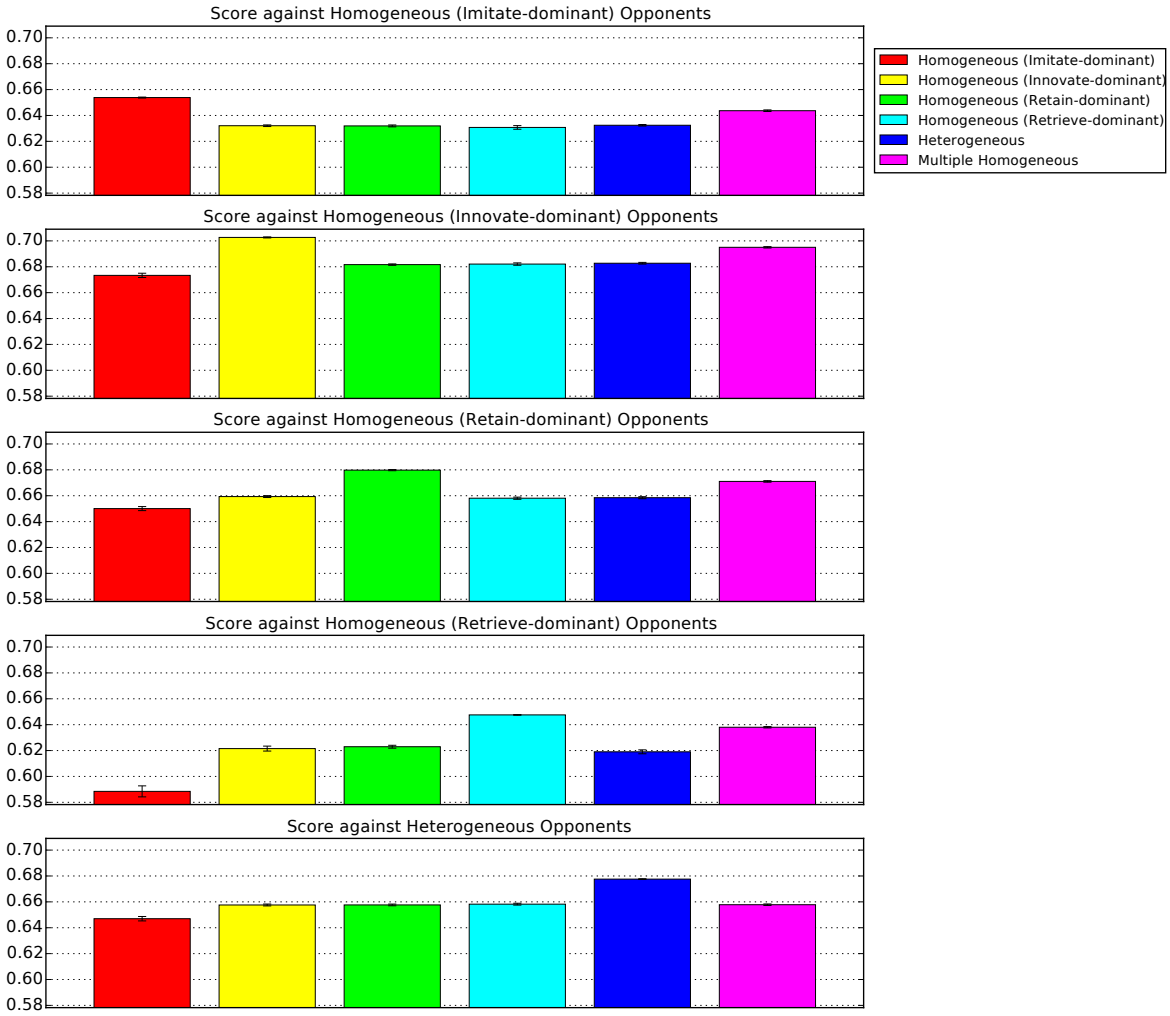


Figure 8.2: Performance comparison among strategies evolved in a single homogeneous, multiple homogeneous, and a heterogeneous environment. Each row shows the performance of strategies in one environment. Each column shows the average score of 64 strategies evolved in the environment indicated by the legend when they are evaluated in the environments that correspond to rows. Thus, the bars in the diagonal starting from top left show the performance of strategies that were evolved in the same environment as the one in which they are evaluated. Each group of evolved strategies on the diagonal perform significantly better than the other evolved strategies in the same row. Moreover, the strategies evolved in multiple homogeneous environments perform significantly better than others in all homogeneous environments (except the strategies evolved in the evaluation environment). Thus, evolution can produce customized strategies for given homogeneous and heterogenous environments, as well as general strategies that perform better than others in multiple environments.

8.4 Experiment 2: Evolving General Strategies

In order to evolve general strategies, two more evolutionary setups were employed, similar to those in Section 5.3: a setup where fitness of strategies are evaluated by averaging the score of the strategy across all four homogeneous environments, and a setup with a heterogeneous environment where two copies of each of those four action-dominant models were used as opponents.

Figure 8.2 shows that strategies evolved in heterogeneous environments also performed better than other strategies in the heterogeneous environment itself, but did not show any benefit over other strategies in homogeneous environments. On the other hand, while strategies evolved in multiple homogeneous environments performed worse in each homogeneous environment than the ones evolved in that particular environment, they significantly outperformed the rest of the strategies (p-value $< 10^{-11}$). In line with results in Section 5.3, these results suggest that evolving strategies in a set of diverse homogeneous environments may be a useful approach to create strategies that generalize well.

So far, evolutionary environments consisted of an artificial combination of models, in order to understand how the different strategies interact. The next section looks at evolutionary results with sets of opponents that are determined in a more realistic way, taking into consideration the groups in the human subject experiment data.

8.5 Experiment 3: Evolving in Complex Environments

To see how strategies evolve in environments that are more realistic than the ones employed so far, evolutionary environments were created to simulate three groups of eight or nine players from the human experiment data. For each player in each group, the player was replaced with the evolving strategy and the opponents with the corresponding human subject's models that were created in Chapter 7, resulting in nine evolutionary environments for Group 1, and eight for Groups 2 and 3, with a total of 25 environments. Each pair of en-

vironments within each group shared all but one opponent, which makes them more similar compared to environments from the two other groups.

As in the previous experiments, 64 evolutionary runs were performed for each environment, and the best strategy at the end of each run was selected as the resulting evolved strategy from that run. Table 8.2 shows the distribution of those strategies, organized by originating human group. Since there are many more strategies displayed in the PCA plots, the two diagonal boundary lines mentioned above at the left and right are much more prominent in Table 8.2 than in Table 8.1.

Furthermore, a few more lines parallel to those two are visible about half-way and one-third of the way from the bottom convergence point to the left and right edges. Just like the left and right boundary diagonals correspond to maximum icon and team imitation, respectively, these lines correspond to 50% or 67% icon or team imitation. This result is likely an artifact of the employed configuration of NEAT, where each member of the initial evolution population is a minimal network, often having a zero value for all except one output of the network, which gets interpreted as a fractional action probability such as 0.5, 0.33, or 0.67. With more strategies in the PCA plots, it is also clearer that the corresponding points lie in a triangle (i.e. the bottom half of a rotated square), which means most of them have at least 50% of team or icon imitation.

The performance of the evolved strategies, grouped by originating human groups can be seen in Figure 8.3. As in Experiments 1 and 2, strategies evolved with similar opponents (i.e. within each group) performed significantly better in environments from the same group compared to strategies evolved in the other two groups' environments (p-value $< 10^{-32}$). The performance difference was smaller than in the first two experiments, which is to be expected because the environments are more similar in Experiment 3 than in Experiments 1 and 2. It is also more difficult to characterize the strategy difference upon which these performance improvements are based. They are subtle and numerous, and in combination, allow the strategy to beat its opponents.

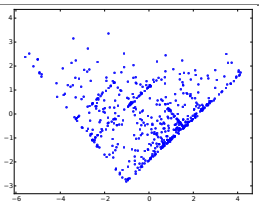
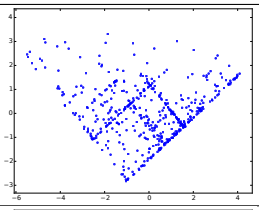
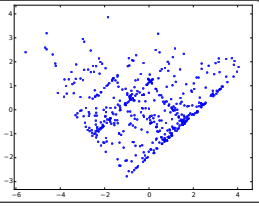
Environment		PCA for All Evolved Strategies
Id	**Opponents**	
1	Group 1 (8) $\times 9$	
2	Group 2 (7) $\times 8$	
3	Group 3 (7) $\times 8$	

Table 8.2: Strategies evolved in environments with model groups that correspond to three human subject groups. For each group, the same number of environments were created as the number of players in the group, where in each environment, one model was replaced with the evolved strategy during fitness evaluation. The first PCA plot shows $64 \times 9 = 576$ evolved strategies since there are nine players in the first group, whereas the second and third plots show $64 \times 8 = 512$ strategies since those groups had eight players. The PCA dimensions are shared with the evolved strategies in Table 8.1. The diagonal boundary lines at the bottom left and bottom right represent maximum icon imitation and maximum team imitation, respectively, and they are more prominent than in Table 8.1 here due to the increased number of displayed strategies.

Figure 8.3: Performance comparison among strategies evolved in environments that correspond to three human subject groups, which are also compared with the average performance of models for human subjects in the group. The layout is similar to Figure 8.2. Again, the groups of strategies on the diagonal perform significantly better than the others shown in the same row (i.e. those evolved in other environments), though the difference is smaller than in Figure 8.2. The results demonstrate that evolution can produce customized strategies for given environments in this domain.

Most interestingly, the evolved strategies significantly outperformed the human subject models that they replaced during evolution by a difference of 0.1 normalized score (p-value $< 10^{-5}$). The score advantage of the evolved strategies can be explained by the difference in action ratios. Overall, the evolved strategies imitated 54% more than the human subject models, with 52% more team imitation (p-value $< 10^{-10}$). On the other hand, the evolved strategies retained their icons 41% less (p-value $< 10^{-8}$), retrieved icons 4.4% less (p-value = 0.044), and innovated 4% less than the models (p-value = 0.014). This result shows that it is indeed possible for evolution to discover and utilize opportunities in realistic human competitive multi-agent environments, and improve behavior over humans. Moreover, this result also provides insights into how human performance could be improved.

8.6 Conclusion

This chapter focused on optimization of strategies in the social innovation game domain. Three experiments explored strategies evolved in various environments with opponents that are models of subjects from the human study: (1) single homogeneous environments with opponents that dominantly perform one type of action, (2) multiple homogeneous environments and a single heterogeneous environment, and (3) environments with sets of opponents representing three diverse subject groups from the human study. Both the first and third experiments showed that evolution was able to produce strategies that are tailored to the particular environment and that they significantly outperform strategies evolved on other environments. Though, the differences were subtle and harder to characterize in the third experiment. Moreover, the second experiment demonstrated that strategies that generalize across diverse sets of opponents can be evolved using multiple homogeneous environments for fitness evaluation. The third experiment in turn showed that it is possible to discover and utilize subtle opportunities in realistic environments, and perform better than the human models by imitating more, and retaining, retrieving, and innovating less. The next chapter will provide general discussion and potential future work.

Chapter 9

Discussion and Future Work

This chapter compares the results of the abstract domain that was the focus of Chapters 3, 4, and 5 with those of the concrete domain of Chapters 6, 7, and 8. It then discusses the limitations and main areas of future work to overcome them, including developing more versatile strategy representations, using alternative optimization methods, exploring further applications for human models, applying the framework to analyzing real-world archival data, characterizing CMAS theoretically, and extending the framework with opponent modeling and communication.

9.1 Domain Comparison

The abstract NK domain and the concrete game domain have several similarities. Most importantly, both domains have a search space with dimensions that correspond to potential elements of a solution. In the abstract NK domain, these elements are the bits of an N-bit binary number, which is a point in the search space. Similarly, a solution in the concrete game domain with six-icon teams and a 48-icon league can be considered to be a 48-bit binary number, but with the limitation that exactly six bits are set as 1 (i.e. included in the team) and 42 bits are set as 0 (i.e. excluded from the team).

In both domains, the elements of a solution contribute to the fitness of the solution in combination with other elements. In particular, in the abstract NK domain each bit makes a contribution to the fitness in combination with K other bits. On the other hand, in the concrete domain only a subset of all icons are involved in such interactions and only in pairs: If such a pair exists in a player's team (i.e. the corresponding bits are set to 1 in the solution) in a round, then bonus or penalty points are added to the player's score for that round.

As is typical in CMAS, both domains have multiple agents who simultaneously perform a search in the solution space for high-fitness solutions. The agents can copy other agents' solutions in whole to a varying extent. In the abstract domain, they can do that via the use of public memory if other agents use public memory, whereas in the concrete domain they can do that in any round using the team imitation action.

On the other hand, the two domains have certain differences as well, besides the ones already mentioned. First, icons in a player's team (i.e. the 1 bits in the solution) contribute to the fitness of the solution individually in the concrete domain, whereas there is no explicit fitness contribution for the 1 bits of the solution in the abstract domain. Second, team scores are not calculated using the NK model in the concrete domain, but by using points for individual icons and bonuses or penalties for a subset of icon pairs, which are fixed. Third, the fitness landscape in the abstract domain is dynamic (i.e. fitness of points first get boosted and then reduced due to crowding), whereas in the abstract NK domain, the landscape does not change as players discover solutions. Fourth, since in the abstract domain there are few agent states of interest (represented by combinations of network input values), the evolved neural networks are used as CPPNs to generate patterns that fill action probability tables; in contrast in the concrete domain, evolved neural networks are used directly as the agent's strategy to allow more fine-grained agent states than in the abstract domain.

The experiments in Section 5.2 showed that distinctly different search strategies

evolve in different environments, and they are generally better than strategies evolved in other environments, evolved general strategies, and manual strategies. They are also significantly more complex than the manual strategies, and would be difficult to design without an automatic machine discovery method such as evolutionary computation. The same result applies to the concrete game domain as demonstrated in Sections 8.3 and 8.5. Moreover, Section 5.3 showed how a single general strategy can be evolved to perform well across multiple different environments, with only a small cost in performance. Section 8.4 showed this result too applies to the concrete domain. Therefore, even though the domains are different, i.e. one is abstract and mathematical, and the other concrete and based on human performance, the same CMAS formalization applies to both and leads to consistent conclusions. The main hypotheses of the dissertation were therefore verified: CMAS captures a category of real-world problem solving, and evolutionary computation is useful in discovering good strategies for CMAS problems.

9.2 Strategy Representation

There are several limitations to the strategy representations employed in this dissertation. In Section 3.7, agent strategies were only coarsely encoded for the abstract NK domain. For instance, agents were able to choose the best point of either the public memory or the private memory. A new action could be added to agents' strategies to let them pick the best of both memories based on their fitness. Such an extension would improve performance by allowing the agents to take advantage of the points with the highest fitness that are available to them, e.g. when both public and private memory points both have high or low points. It might also lead to more realistic agent behaviors.

Furthermore, the power of CPPN to represent strategies was not yet fully utilized for the abstract domain. Because their inputs and outputs are continuous, CPPNs can in principle represent strategies over a very large number of (even continuous) states and actions. Such an approach would make it possible to represent much more refined strategies,

which could in turn lead to more complex emergent behaviors, and to more accurate modeling of real-world search. In fact, a similar path was chosen for the concrete domain in Chapter 8: evolved CPPNs are used as neural networks that represent agent strategies directly: actions (i.e. network outputs) are then performed probabilistically based on agent's state (i.e. network inputs).

However, even that approach has limitations in the concrete game domain. For instance, in the experiments in Chapters 7 and 8, the imitation action was limited to copying the highest-scoring player. This limitation can be relaxed by allowing the imitation source to be one of the top two or three players, chosen randomly with probabilities based on proportions observed in experiments with human subjects (i.e. 82% from the player with first rank and 10.7% from the player with second rank; Wisdom et al., 2013). Alternatively, agents can be allowed to choose the imitation source freely, just like the human subjects, (e.g. via using model outputs to specify the exact imitation source). This approach might help mimic behaviors such as copying from a specific player, instead of always from the best-scoring one.

Another limitation concerns choosing icons to imitate, innovate, and retrieve. While the second tier neural network considers icon-specific state (i.e. icon's popularity and age), while deciding whether to e.g. imitate, the icon to copied from the source was chosen randomly. A natural extension would be to add another output to the evolved neural network architecture, and run the network with icon-specific inputs on each icon to decide which of those icons to copy. The icons that already exist in the player's own team would be skipped for that round. Such an extension could allow learning to copy more popular icons, and therefore make innovation, icon imitation, and icon retrieval actions more realistic and less random.

To obtain even more sophisticated and potentially more realistic strategies, agent strategies could remember how the replacement of only a single icon in any player's team changes its score. Such a memory is possible because the scores for all players' teams

are made known to all players at the end of each round. Players could even replace single icons in their teams to actively collect such observations. Such a memory could help them make more informed decisions about which icons to copy from other teams or the league and which icons to replace in their teams. For instance, if the score increases significantly when another player replaces icon X in their team with icon Y, and the first player happens to have X in their team in a following round, they could choose to replace that icon with Y. Note that observations made in one game would only be useful during that game, since point assignments for individual icons and icon pairs are shuffled at the beginning of each game. While the bonuses or penalties for icon pairs might complicate such an extension, it might still be worthwhile, especially for the game configuration with 48 icons in the league, where only a third of the icons are in such a pair: The probability of either the copied or replaced icon being in such a pair and of both that icon and its pair being in a team of six icons is relatively low. Also, if two conflicting observations are made for the same pair of icons, which would mean that one icon in that pair is part of a bonus or penalty pair, then that icon pair could be excluded from this extension.

In short, there are several ways to improve and extend strategy representations. Future work opportunities exist in optimizing them as well, as described in the next section.

9.3 Optimization

For general strategies like those evolved in Sections 5.3 and 8.4 to scale up to even more diverse environments, it might be beneficial to allow multimodal behaviors, so that the same agent strategy can have distinct behaviors in different contexts (Schrum and Miikkulainen, 2012). A further step in this direction would be to coevolve all or a subset of the opponents as well. In this manner, the environment could present more diverse challenges, and more interesting and perhaps realistic general strategies could evolve.

Another future direction is to extend the search with multiple objectives, i.e. fitness measures. Instead of having a single fitness objective, there could be more than one rel-

evant way of measuring the quality of a solution. For instance, for a product in industry, these multiple fitness objectives could include novelty, usefulness, and product desirability according to consumers, as well as market share and value from the perspective of each company. The fitness used in this dissertation corresponds to this value. When applied to innovation search, that value would represent the potential to make money. To implement multiple objectives agents can be made to utilize aspects of multi-objective optimization methods, such as NSGA-II (Deb et al., 2000), and consider multiple candidate solutions as a Pareto set. Such an extension would make it possible to model product evaluation more flexibly and more accurately, and to decouple novelty and other types of fitness to investigate their effects in isolation and therefore more effectively.

A limitation of the experiment in Section 8.5 was that strategies were optimized against opponent models for only three groups of human subjects. More subject groups can be used as opponents; such an extension would make it possible to investigate the differences between strategies optimized against different sized groups. Because it is already known that group size affects ratios of utilized actions and the score, such a comparison might yield insights into what strategies work best against such diverse groups.

One downside of the evolutionary approach utilized in this dissertation is that it requires a large population and therefore a significant number of fitness evaluations. Thus, alternative evolutionary methods such as CMA-ES (Hansen et al., 2003) and Estimation of Distribution Algorithms (Larrañaga and Lozano, 2002) may be considered. In those approaches, statistical models of the solutions and their fitness are maintained, making it possible to get by with fewer fitness evaluations in structured domains like CMAS.

The optimization for the concrete game domain used human models as the opponents in the environment. The next section will present future work in improving those models.

9.4 Human Models

Gameplay data used for training human models in Chapter 7 was limited to games with eight or nine players. With fewer players, scores are different, which complicates distance objectives. This restriction could be relaxed by normalizing the scores achieved by human subjects, thus giving access to more training data.

Moreover, only the last game for each human group was used for modeling. The average score of subjects increased across games; they probably learned to play better over the eight games. By training only on the last game, their best performance was thus captured. However, the average scores usually plateaued in the last three games. Including all three would help increase the coverage of training data to more game states for the subjects who are modeled.

Models of human subjects, such as those obtained in Sections 7.5 and 7.6, can be useful as replacements for human players, thereby making it possible to obtain much more data, e.g. through Mechanical Turk (MT). When an MT worker wants to play the game, and there is nobody else connected to the game server at the same time, the previously trained models can fill in so that the human player has a group of peers with which to play the game. Human models that show high levels of a specific action (e.g. imitation), can be used as peers for human subjects to create specific environments. Homogeneous sets of such model peers could help provide insight into how human subjects react to environments with, for example, high innovation, imitation, or retention. On the other hand, combinations of model peers with different dominant actions could be utilized to find which types of environments human subjects perform best.

Furthermore, by evolving general strategies that score as high as possible (as in Section 8.4), as low as possible, or at any desired level, one can create game environments for human subjects where all their peers are likely to score at a desired level. Such environments can be used to study how people's performance or their proportion of actions are affected by the competition. It would be interesting to determine, for example, whether

high-performing peers lead people to imitate more, or whether low-performing peers cause people to innovate more.

9.5 Other Future Work

More generally, the simulations in this dissertation suggest that the CMAS approach can be used to provide insight into what kinds of strategies work well in real-world competitive multi-agent search. One possibility is to set up the search space and the agent parameters based on real-world archival data, such as the historical record on patents and products in a particular industry (e.g. mobile electronics, robotics, and pharmaceuticals). Search can then be simulated on that landscape, explaining why certain strategies were effective, and potentially discovering new strategies that would have worked even better.

On the other hand, CMAS is a general and formally defined problem domain, which should make it possible to analyze it theoretically. Developing a theory for the interactions of agents in competitive multi-agent search may help derive optimality principles based on stochastic strategies. It might then be possible to formulate theoretical bounds on performance of agents in various environments such as ones where agents act extremely competitively or extremely cooperatively. For example, stochastic processes (Papoulis and Pillai, 2001) could be used to characterize the scope and power of the search methods, deriving convergence and dominance conditions, as has been done in prior work on coevolution and estimation of distribution algorithms (Stanley and Miikkulainen, 2004; Alden, 2007; Ficici and Pollack, 2001; Mühlenbein and Höns, 2005). Such an extension may provide theoretical insight into innovation search as a real-world application domain. Furthermore, since the speed and magnitude of boosting and crowding differs in various real-world CMAS domains, those parameters could be utilized in the theoretical analysis to make it more useful for particular domains.

Another potential application for CMAS is games where users (e.g. team owners) put together a solution (e.g. a team) that consists of distinct components (e.g. football play-

ers) to compete with other users' solutions. For instance, in (American) fantasy football[1], users perform actions such as drafting, trading, adding, and dropping real players in order to create their fantasy teams. Each week, a specified number of starting players are designated for each game position by the user to start that week's game, where the remaining players in the user's team are benched. Each team gets a score based on the performances of its starters in that week's NFL games. Similar issues come up in several economic strategy games, where the players try to maximize their portfolio of resources. It may be possible to create good artificial players for such games, making them more interesting and realistic. It may also be possible to extend them to training people in strategic thinking in economic domains.

Aspects of real-world competitive multi-agent search that were not addressed in this dissertation include opponent modeling and explicit interactions between agents via direct communication. Opponent models let agents adapt to the environment by altering their strategies depending on the opponents (Carmel and Markovitch, 1996). Communication allows agents to cooperate more effectively, form coalitions, and negotiate.

Another missing aspect is the concept of cost for discovering solutions (i.e. moving to a new point in the search space). For instance, in innovation search, the cost of coming up with a new product or prototype is different for each company. Moreover, this cost also depends on the particular features included in the product. To capture this level of detail, each included feature (i.e. 1 bit in the binary representation of the solution) would need to contribute a certain amount to the cost individually and possibly in combination with other features, similar to how icons contribute to the team's score in the concrete human game domain of Chapters 6-8.

Furthermore, in real-world innovation search domains, such as technological innovation or scientific discovery, the dimensionality of the search space is not fixed. New dimensions are constantly added to the search space, as in the case of invention of new

[1] http://en.wikipedia.org/wiki/Fantasy_football_(American)

classes of features or patents that can be used to build new products. However, the dissertation assumes that the search space has a fixed dimensionality. While such a restriction may not be a problem for modeling and studying past agent search behavior, it may make predicting and planning for the future harder. Therefore, an extension that would allow new dimensions to be added to the search space would make the methodology more powerful. Such extensions would make it possible to extend CMAS to more competitive problem-solving situations and make it more useful in the real world.

9.6 Conclusion

This chapter discussed the similarities and differences between the abstract NK and concrete social innovation game domains, as well as how the results in the two domains compare. Several ideas for future work were proposed to remove assumptions, simplifications, and limitations, and to try alternative evolutionary optimization methods. Extensions to obtain more representative human models in the concrete domain, and possible future applications for those models were also discussed. The extensions presented in this chapter would all be useful in modeling real-world innovation search in various future domains.

Chapter 10

Conclusion

Competitive multi-agent search takes place in a non-stationary environment, which requires the solution method to be reactive to changes in the fitness landscape and the knowledge of opponents' searches. Thus, the problem definition and the solution method for such search is different from standard search. Therefore, competitive multi-agent search has not been extensively studied in previous work neither computationally nor as a model for human problem solving. This dissertation aims to improve our understanding of such search in general and in innovation search in particular, using a series of experiments in an abstract and a concrete domain. This final chapter summarizes the contributions of the dissertation, and evaluates their potential impact.

10.1 Contributions

This dissertation makes three contributions. First, in Chapter 3 it develops *competitive multi-agent search* (CMAS) as a formalization of human creative problem-solving activities. CMAS was originally developed to understand how high-technology companies search for technological innovations, but the same formalization can potentially be applied to computational modeling of scientific discovery, engineering problem solving, and art and de-

sign. In CMAS, multiple agents search for the same peaks (i.e. innovations) on the common fitness landscape. They each try to find as many and as high peaks as possible over a given amount of time, representing the cumulative value of their innovations. While searching, they can choose to share information about what they find, or keep such information private. Furthermore, the landscape is dynamic in that the fitness of the points can increase or decrease when multiple agents discover them, representing the dynamic valuation of innovations in the real world. The CMAS formalization is useful because it makes it possible to characterize how humans perform creative problem solving activities, resulting in precise theories in management science, psychology, and social science. However, the formalization also makes it possible to use such models to determine how humans could perform better than they currently do, thus informing both the individuals who are trying to solve these problems, and the administrators that design policies to encourage innovation and creativity.

The second contribution focuses on this opportunity to do better. The dissertation demonstrates that *evolutionary computation is a particularly good way to solve CMAS problems*. Two domains were utilized as experimental platforms. The first one was an abstract, general CMAS domain, defined in terms of an NK fitness landscape (Kauffman, 1993), as described in Chapter 3. A comprehensive array of basic search strategies was created for this domain in Chapter 4, based on local (exploitative) search and long-range (exploratory) search using public and/or private information about the landscape, and compared in various environments with those strategies used as opponents. An advanced search strategy was also implemented based on tree search, representing a typical AI problem solving method (Korf, 1990). These strategies were then used to instantiate several different competitive environments, by including competitors with different strategies. In Chapter 5, new strategies were evolved for each environment in order to perform better than the existing ones. The results show that (1) evolution can discover customized strategies that perform well in homogeneous and heterogeneous environments, (2) it can discover general strategies that

perform well across many different environments, and (3) the good evolved strategies are more complex than the basic intuitive strategies. Such strategies exhibit optimal preference for acting publicly or privately depending on the particular environment, and result in overall principles such as riding a wave of dynamically increasing landscape. The simulations also demonstrate that some intuitive strategies such as *explore when low and exploit when high* are not always the best. Furthermore, an interesting "Twitter effect" was discovered: If the domain allows too much visibility, the agents tend to follow the same points, and the performance of the population as a whole suffers.

The second domain was a social innovation game domain, described in Chapter 6. Using human gameplay data from a study by Wisdom et al. (2013), in Chapter 7 models were trained for each human subject, comparing different modeling approaches. In Chapter 8 those models were then used as opponents to evolve optimal strategies. The results show that evolution can produce strategies customized for specific environments: Strategies evolved in single homogenous environments and in a heterogeneous environment performed better than strategies evolved in other environments. The same result also applies to strategies evolved against models of diverse human subject groups, but the performance difference was smaller, and the difference in strategies was subtle and harder to characterize. Furthermore, those evolved strategies performed significantly better than the human models, where the evolved strategies differed from the models in that on average they performed significantly higher imitation and lower retention, innovation, and retrieval. Evolution was also able to produce general strategies that perform better than strategies evolved in a single environment, but lower than those evolved for the particular evaluation environment. Thus, insights about CMAS were first developed in the abstract domain that made them easy to see, and then verified in the concrete, real-world human subject domain. These results suggest that evolutionary computation is a good way to design strategies for CMAS in general.

The third contribution is a technical one: a novel spherical visualization for NK

fitness landscapes, described in Section 3.2. This visualization maintains the continuity of the original high-dimensional landscape while reducing it to an intuitive 3D surface. A focal point is selected, and continuity is maintained by representing points further away with lesser resolution. This visualization is useful in illustrating the search strategies in the abstract domain: For instance, it makes it strikingly clear why the wave-riding behavior is so effective, and why the Twitter effect can be so devastating. The visualization is general, and could be useful for any study involving high-dimensional binary spaces.

10.2 Conclusion

This dissertation showed that CMAS is a potentially useful way to study problem solving in the real world, and that evolutionary computation is an effective way to gain insight into such problems. It is possible to formulate many real-world creative activities as CMAS, discover effective search strategies that might be hard to design by hand, and understand why they are effective. CMAS thus demonstrates an interesting role for evolutionary computation: it can be used not only as an automated method for engineering, but also as a way to understand how human behavior can be more effective. In the future, it may be possible to use CMAS simulations to make recommendations to human decision makers, as well as inform policy makers that aim at encouraging innovation and creativity.

Appendix A

Parameters

Simulation Parameter	**Value**
N (for the NK model)	12 and 30
K (for the NK model)	4
Number of agents	10
Number of time steps per run	200 in Sec. 4.1-4.3 and 500 in Sec. 4.4
Number of runs per strategy evaluation	200
Flocking intensity	$\{1.1, 1.2\} \rightarrow 0.9$ (over 10 visits)
Flocking radius	2
S_1 usage frequency	once every five time steps

Table A.1: Simulation parameters for the abstract domain in Sections 4.1-4.4.

Simulation Parameter	**Value**
N (for the NK model)	10 and 20
K (for the NK model)	3
Number of agents	8
Number of time steps per run	100
Number of runs per strategy evaluation	200
Flocking intensity	$1.05 \rightarrow 0.9$ (over 10 visits)
Flocking radius	2
S_1 usage frequency	every time step

Table A.2: Simulation parameters for the abstract domain in Section 4.5 and Chapter 5.

Evolution Parameter	**Value**
Number of evolutionary repeats per setup	64
Number of generations	500
Population size	100

Table A.3: Evolution parameters for both the abstract and concrete domains.

NEAT Parameter	Value
Add bias to hidden nodes	1
Adult link age	2
Age significance	1.2
Allow add node to recurrent connection	0
Allow recurrent connections	0
Allow self recurrent connections	0
Compatibility modifier	0
Compatibility threshold	20
Disjoint coefficient	1.0
Dropoff age	10
Excess coefficient	1.0
Extra activation functions	1
Extra activation updates	19
Fitness coefficient	1.0
Force copy generation champion	1
Link gene minimum weight for phenotype	0
Mutate add link probability	0.2
Mutate add node probability	0.2
Mutate demolish link probability	0.04
Mutate link probability	0.2
Mutate link weights probability	0.8
Mutate node probability	0.05
Mutate only probability	0.5
Mutate species champion probability	0
Mutation power	2
Only Gaussian hidden nodes	0
Signed activation	0
Smallest species size with elitism	1
Species size target	0
Survival threshold	0.2
Weight difference coefficient	0.8

Table A.4: NEAT parameters for both the abstract and concrete domains.

Backpropagation Parameter	Value
Learning rate	0.1
Momentum	0.1
Weight decay	0.01

Table A.5: Backpropagation parameters used when training neural network models for the concrete domain.

Bibliography

Alden, M. E. (2007). *MARLEDA: Effective Distribution Estimation Through Markov Random Fields*. Ph. D. thesis, Department of Computer Sciences, The University of Texas at Austin. Technical Report AI07-349.

Anderson, P. (1999). Complexity theory and organization science. *Organization Science 10*(3), 216–232.

Axelrod, R. (1997). *The complexity of cooperation: Agent-based models of competition and collaboration*. Princeton Univ Pr.

Carmel, D. and Markovitch, S. (1996). Opponent modeling in multi-agent systems. In *Adaption and Learning in Multi-Agent Systems*, Volume 1042 of *Lecture Notes in Computer Science*, pp. 40–52. Springer Berlin Heidelberg.

Černỳ, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications 45*(1), 41–51.

Chiel, H. J., Beer, R. D., and Gallagher, J. C. (1999). Evolution and analysis of model CPGs for walking: I. Dynamical modules. *Journal of Computational Neuroscience 7*(2), 99–118.

Cyert, R. and March, J. (1963). *A Behavioral Theory of the Firm*. Englewood Cliffs, NJ: Prentice-Hall.

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Parallel Problem Solving from Nature PPSN VI*, pp. 849–858. Springer.

Dorigo, M. and Stützle, T. (2004). *Ant Colony Optimization*. MIT Press.

Epstein, J. (1999, May). Agent-based computational models and generative social science. *Complexity 4*(5), 41–60.

Ficici, S. G. and Pollack, J. B. (2001). Pareto optimality in coevolutionary learning. In J. Kelemen (Ed.), *Sixth European Conference on Artificial Life*. Berlin: Springer.

Gavetti, G. and Levinthal, D. (2000). Looking forward and looking backward: Cognitive and experiential search. *Administrative Science Quarterly 45*, 113–137.

Goldberg, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. In J. J. Grefenstette (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 148–154. San Francisco: Morgan Kaufmann.

Greve, H. (2003a). A behavioral theory of R&D expenditures and innovations: Evidence from shipbuilding. *Academy of Management Journal 46*, 685–702.

Greve, H. (2003b). *Organizational learning from performance feedback: A behavioral perspective on innovation and change*. Cambridge Univ Pr.

Greve, H. and Taylor, A. (2000). Innovations as catalysts for organizational change: Shifts in organizational cognition and search. *Administrative Science Quarterly 45*, 54–80.

Hansen, N., Müller, S., and Koumoutsakos, P. (2003). Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation 11*(1), 1–18.

Hart, P., Nilsson, N., and Raphael, B. (1968, July). A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on 4*(2), 100 –107.

Harvey, I., Husbands, P., Cliff, D., Thompson, A., and Jakobi, N. (1997). Evolutionary robotics: the Sussex approach. *Robotics and autonomous systems 20*(2), 205–224.

Helfat, C. (1994). Evolutionary trajectories in petroleum firm R&D. *Management Science 40*, 1720–1747.

Hoen, P. J., Tuyls, K., Panait, L., Luke, S., and Poutr, J. A. L. (2005). An overview of cooperative and competitive multiagent learning. In *LAMAS*, pp. 1–46.

Holland, J. and Miller, J. (1991). Artificial adaptive agents in economic theory. *American Economic Review 81*(2), 365 – 71.

Hosseini, H. (2009). The intelligent water drops algorithm: A nature-inspired swarm-based optimization algorithm. *International Journal of Bio-Inspired Computation 1*(1/2), 71.

Hutt, B. and Warwick, K. (2003). Synapsing variable length crossover: Biologically inspired crossover for variable length genomes. In D. Pearson, N. Steele, and R. Albrecht (Eds.), *Artificial Neural Nets and Genetic Algorithms*, pp. 198–202. Springer Vienna.

Juille, H. and Pollack, J. B. (1996). Dynamics of co-evolutionary learning. In *In Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, pp. 526–534. MIT Press.

Katila, R. (2002). New product search over time: Past ideas in their prime? *Academy of Management Journal 45*(5), 995–1010.

Katila, R., Bahceci, E., and Miikkulainen, R. (2014). Organizing for innovation: Exploratory, ambidextrous and exploitative units in competitive environments. Working paper.

Katila, R. and Chen, E. (2008). Effects of search timing on product innovation: The value of not being in sync. *Administrative Science Quarterly 53*, 593–625.

Katila, R., Chen, E. L., and Piezunka, H. (2012). All the right moves: How entrepreneurial firms compete effectively. *Strategic Entrepreneurship Journal 6*(2), 116–132.

Katila, R., Rosenberger, J., and Eisenhardt, K. (2008). Swimming with sharks: Technology ventures, defense mechanisms, and corporate relationships. *Administrative Science Quarterly 53*, 295–332.

Kauffman, S. A. (1993). *The Origins of Order*. New York: Oxford University Press.

Kennedy, J., Eberhart, R., et al. (1995). Particle swarm optimization. In *Proceedings of IEEE international conference on neural networks*, Volume 4, pp. 1942–1948. Piscataway, NJ: IEEE.

Kennedy, J., Eberhart, R., and Shi, Y. (2001). *Swarm Intelligence*. Morgan Kaufmann Publishers.

Knight, K. (1993). Are many reactive agents better than a few deliberative ones? In *Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 1*, IJCAI'93, San Francisco, CA, USA, pp. 432–437. Morgan Kaufmann Publishers Inc.

Knudsen, T. and Levinthal, D. (2007). Two faces of search: Alternative generation and alternative evaluation. *Organization Science 18*, 39–54.

Knuth, D. E. and Moore, R. W. (1975). An analysis of alpha-beta pruning. *Artificial Intelligence 6*(4), 293 – 326.

Kohl, N., Stanley, K. O., Miikkulainen, R., Samples, M., and Sherony, R. (2006). Evolving a real-world vehicle warning system. In *Proceedings of the Genetic and Evolutionary Computation Conference*.

Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence 27*, 97–109.

Korf, R. E. (1990). Real-time heuristic search. *Artificial Intelligence 42*(2-3), 189 – 211.

Korf, R. E. (1991). Multi-player alpha-beta pruning. *Artificial Intelligence 48*(1), 99 – 111.

Larrañaga, P. and Lozano, J. A. (2002). *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*, Volume 2. Springer.

Levinthal, D. A. (1997). Adaptation on rugged landscapes. *Management Science 43*(7), pp. 934–950.

Levinthal, D. A. and March, J. G. (1981). A model of adaptive organizational search. *Journal of Economic Behavior and Organization 2*, 307–333.

Lohn, J., Linden, D., Hornby, G., Kraus, W., Rodriguez-Arroyo, A., and Seufert, S. (2004). Evolutionary design of an X-band antenna for NASA's Space Technology 5 mission. In *National radio science meeting*, pp. 2313–2316.

March, J. and Simon, H. (1958). *Organizations*. New York: Wiley.

March, J. G. (1991). Exploration and exploitation in organizational learning. *Organization Science 2*(1), pp. 71–87.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press.

Moriguchi, H. and Honiden, S. (2012). CMA-TWEANN: Efficient optimization of neural networks via self-adaptation and seamless augmentation. In *Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference*, pp. 903–910. ACM.

Mühlenbein, H. and Höns, R. (2005). The estimation of distributions and the minimum relative entropy principle. *Evolutionary Computation 13*(1), 1–27.

Papoulis, A. and Pillai, S. U. (2001). *Probability, Random Variables and Stochastic Processes*. McGraw-Hill Science/Engineering/Math.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Reading, MA: Addison-Wesley.

Pollack, J. B., Blair, A. D., and Land, M. (1996). Coevolution of a backgammon player. In C. G. Langton and K. Shimohara (Eds.), *Proceedings of the 5th International Workshop on Artificial Life: Synthesis and Simulation of Living Systems (ALIFE-96)*. Cambridge, MA: MIT Press.

Rashedi, E., Nezamabadi-pour, H., and Saryazdi, S. (2009, June). GSA: A gravitational search algorithm. *Information Sciences 179*(13), 2232–2248.

Rivkin, J. (2000). Imitation of complex strategies. *Management Science 46*, 824–845.

Schrum, J. and Miikkulainen, R. (2012, June). Evolving multimodal networks for multitask games. *IEEE Transactions on Computational Intelligence and AI in Games 4*(2), 94–111.

Sigal, N. and Alberts, B. (1972, November). Genetic recombination: The nature of a crossed strand-exchange between two homologous DNA molecules. *Journal of Molecular Biology 71*(3), 789–793.

Stanley, K. (2007, June). Compositional pattern producing networks: A novel abstraction of development. *Genetic Programming and Evolvable Machines 8*(2), 131–162.

Stanley, K. O., Bryant, B. D., and Miikkulainen, R. (2005). Real-time neuroevolution in the NERO video game. *IEEE Transactions on Evolutionary Computation 9*(6), 653–668.

Stanley, K. O. and Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evolutionary Computation 10*, 99–127.

Stanley, K. O. and Miikkulainen, R. (2004). Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research 21*, 63–100.

Valsalam, V. K., Bednar, J. A., and Miikkulainen, R. (2007). Developing complex systems using evolved pattern generators. *IEEE Transactions on Evolutionary Computation 11*(2), 181–198.

Werfel, J., Mitchell, M., and Crutchfield, J. (2000, November). Resource sharing and coevolution in evolving cellular automata. *Evolutionary Computation, IEEE Transactions on 4*(4), 388 – 393.

Wisdom, T. N., Song, X., and Goldstone, R. L. (2013). Social learning strategies in networked groups. *Cognitive Science 37*, 1383–1425.

Zuckerman, I. and Felner, A. (2011, December). The MP-MIX algorithm: Dynamic search strategy selection in multiplayer adversarial search. *Computational Intelligence and AI in Games, IEEE Transactions on 3*(4), 316 –331.