An Investigation of Continuous Learning in Incomplete Environments

by Yan Su, BSc

Thesis submitted to the University of Nottingham
for the degree of Doctor of Philosophy
September 2005

Dedicated to the loving memory of my Father

Table of Contents

Abstract

Blondie24 and *Chinook* are both examples of automated, world class checkers players. Blondie24 used an evolutionary approach, where a population of initially random players were evolved to the stage where it was able to play at an expert level. Blondie24 did not incorporate any domain knowledge within its learning processes. Chinook, on the other hand, utilised domain knowledge (in the form of grandmasters), along with opening and end game databases and sophisticated search techniques, which included parallel processing. The differences between the two approaches are stark. What are often not considered however are the similarities between the two systems, which could also be considered as their shortcomings. These similarities are: **1**) Ultimately, we have a static player in that once the final version of the player has been produced it no longer changes or adapts to its changing environment (e.g. by playing stronger players and learning from that experience. **2**) Blondie24 and Chinook operate within a perfect world in that the rules of the game are known and all players within that world have perfect knowledge about the world. **3**) Both approaches have the concept of a *current* best player. In the case of Chinook, this is the player currently under development. In the case of Blondie24 it is the best player from the current generation. This thesis addresses these shortcomings by investigating an evolutionary learning mechanism that: **1**) Continues to evolve during the lifetime of the system. **2**) Operates within an incomplete environment and does not have complete knowledge of its environment. **3**) Uses the concept of individual and social learning so that promising individuals are stored in a central pool and can be called upon in later generations.

We suggest that the move towards continuous evolution, not having perfect knowledge of the world and being able to learn from previous generations more closely mimics how we as humans learn. Of course, we are still far away from having

machines that could be considered as intelligent as humans, or even having *human-like* intelligence. However, this thesis demonstrates that agents are able to continuously learn within an incomplete environment and adapt to the environment that changes minute by minute.

In this thesis, we propose *imperfect evolutionary systems* as a new perspective on the study of continuous learning in incomplete environments. In order to demonstrate our approaches we use the stock market as an example of an imperfect system, which is also an example of a constantly changing environment. Following the literature review and a description of the stock market, we define what we mean by an imperfect evolutionary system (chapter 4). Following this, in chapter 5, a multi-agent simulated stock market is developed in which artificial stock traders, under partial understanding of the market, learn to trade profitably by utilising individual and social learning. We initially test this hypothesis on one share, which has a general upward trend, before showing that the same learning mechanisms are able to operate on various shares, with different trading profiles. In chapter 6, we extend the experiments so that the trading environment becomes incomplete and constantly evolving. This is achieved by introducing new market indicators during the evolutionary period. For example, new technical indicators are introduced to the artificial stock market and we monitor how the traders utilise this new information and how it is disseminated to other traders.

This thesis demonstrates that agents are able to operate within an incomplete environment, without full knowledge of their domain and still perform at a high level through continuous learning. This is closer to the environment in which humans operate. Of course, we are still a long way from fully understanding, and being able to emulate, human intelligence, but this thesis does provide some insights that could be used as the basis for future research.

List of Figures

List of Tables

Publications Produced

During my PhD research programme, the following publications have been produced, representing the research work conducted for this thesis:

Kendall, G. and Yan, S. (2006)"Imperfect Evolutionary Systems." Submitted to *IEEE Transactions on Evolutionary Computation* (2nd revision).

Kendall, G. and Yan, S. (2004) "Learning with Imperfections - A Multi-Agent Neural-Genetic Trading Systems with Differing Levels of Social Learning." in *Proceedings of the 2004 IEEE Conference on Cybernetic and Intelligent* Systems, pp. 47-52.

Kendall, G. and Yan, S. (2003) "A Multi-agent Based Simulated Stock Market - Testing on Different Types of Stocks." in *Proceedings of the Congress on Evolutionary Computation*. Special Session on "Evolutionary Computation in Economics". pp. 2298-2305.

Kendall, G. and Yan, S. (2003a) "Co-evolution of Successful Trading Strategies in A Simulated Stock Market." in *Proceedings of International Conference on Machine Learning and Applications*, pp. 200-206.

Acknowledgements

I would like to thank my academic supervisor, Dr. Graham Kendall for his kind support and valuable advice on my research.

I also want to thank Dr. Sonia Schulenberg (Napier University) for her valuable advice at the starting period of my PhD study.

I thank my mother for her generous love and care. Thank my family and friends for their consistent support. Thank God for always being there.

Chapter 1

Introduction

The research in this thesis investigates continuous learning in incomplete environments in pursuing a rational approach to the problem of how machine learning respond to new challenges from its environment. Today's machine intelligence still differs from human intelligence in many aspects. Using game playing as an example, a human checker player constantly learns from his opponents while a game-playing computer program, such as *Blondie24* (Fogel 2002), does not. A human also learns to play different kind of games while as a computer game player is in general designed just for a specific game and has no capability to learn other games. The problem here is that machine intelligence fails to react to new challenges from its environment and fails to learn new knowledge. Through the development of imperfect evolutionary systems, this thesis proposes a change from the conventional perfect paradigm in machine learning and artificial intelligence to an imperfect paradigm, in which intelligent artificial entities are capable to respond to new challenges from their incomplete and changing environments. In this chapter, a general background on the research of continuous learning in incomplete environments is presented in the context of artificial intelligence, evolutionary computation and information economics,

followed by an overview on the structural organisation of the thesis, and a list of contributions from this thesis.

1.1 Background

The motivation behind the development of imperfect evolutionary systems comes from the advancement in the field of artificial intelligence research, particularly, in the field of computer game playing. An important feature of human intelligence is our ability to learn new things, i.e., the ability to absorb new information from the environment and transfer it into our knowledge. There have been many efforts to create such intelligence. One of the most notable works in recent years is *Blondie24*, developed by Fogel and Chellapilla (Chellapilla and Fogel 2001; Fogel 2002). *Blondie24* is an evolutionary computer program that taught itself to play checkers from scratch, without the input of human expert knowledge. *Blondie24* eventually developed its own game playing strategies to the level of a human expert. The key advance in *Blondie24* is that the system did not use any pre-programmed human expertise, which significantly distinguishes *Blondie24* from other games such as *Chinook* (Schaeffer 1996), *Poki* (Billings *et al.* 2002, Billings *et al.* 2003), and *Deep Blue* (Newborn 1996; King 1997; Campbell *et al.* 2002), which were mainly fast, sophisticated search engines with the addition of large amounts of human expertise along with opening and end game databases. *Blondie24* was a major step forward in answering Arthur Samuel's challenge about machine learning without pre-injected expertise (Samuel 1959, 1967). However, *Blondie24* has still not answered the question of whether learning new challenges from the environment is possible. As

stated above, an important feature of human intelligence is our ability to learn new things from our environment. A human checker player's skill and knowledge about checker evolves when he plays with different opponents. In such situation we can observe the growth and adaptation of human intelligence as it adapts to its every changing environments. On the contrary, *Blondie24*, and *Chinook*, *Poki*, *Deep Blue*, are the same in the sense that they are static end products with no adaptability. These so-called intelligent computer programs, or systems, are like works of art, where their creators can make a little change here or there to make them look more perfect, but the artwork itself is not creative. Similarly, the developers of *Blondie24* can retrain the neural networks used by *Blondie24* to make the program even more "clever". However, *Blondie24* itself, as a "perfect" end product, is not intelligent in the sense that it cannot learn from its real world environment. Note that, in this thesis, the following two terms, "perfect" and "complete", are used interchangeably, same as to "imperfect" and "incomplete". By saying *Blondie24* is a perfect end product, we mean that *Blondie24* is regarded as a complete product by its developers and has no capability for continuous learning in changing environments.

Pioneering work by Palmer *et al.* (1994) in agent-based computational economics pointed out that the stock market can be used as a test bed for the study of complex and adaptive artificial lives. In Palmer *et al.* (1994), an artificial stock market (ASM), called the Santa Fe artificial stock market, was developed where independent artificial agents, modelled using learning classifier systems (LCS), buy and sell stock on a central market and co-evolve by means of genetic algorithms (GA). Inspired by Santa Fe ASM, various other types of artificial stock markets were developed in order to

study market dynamics/behaviours, such as the stock pricing or market phenomena such as bubbles and crashes, that cannot be modelled by conventional representative approaches under rational expectations (Arthur *et al.* 1997; LeBaron *et al.* 1999; Yang 1999; Chen and Liao 2000; Chen and Yeh 2002; Lavigne 2004 etc.). This bottom-up approach for modelling financial markets has given economists deeper insights into the working of financial theories and hypothesises. Modelling financial markets as evolutionary multi-agent systems have also provided a test bed for the exploration of many important issues related to artificial intelligence, for example, the modelling of individual artificial agents' rationality and co-evolution among heterogeneous artificial agents in social environments.

Besides the study of financial markets as evolutionary and adaptive systems, the research on *information economics* (Stiglitz 2001, 2003) over the last three decades has also proposed strong challenges to the prevailing competitive paradigm within economics. Many phenomena that have emerged from financial markets cannot be explained by the conventional competitive equilibrium model of economics, such as the persistent and large-scale unemployment in the labour market and the excessive volatility of asset prices in financial and property markets. Information economics intends to explain these market failures by questioning one of the assumptions that sustain the standard competitive paradigm, i.e., the assumption of the existence of perfect information in markets. Under this assumption, markets are fully informationally efficient. That is information is disseminated efficiently and perfectly throughout the economy. All participants of the market have the same information, i.e., *I know what you know and you know what I know*. Obviously, it is different in the

real world: Different people know different things. For example, in the insurance market, the insured knows more about his health than the insurance company; or in the stock market, millions of investors make their investment decisions based on what they know and what they think about the future market movement. Information economics stresses a change from the conventional perfect information paradigm to an imperfect information paradigm in understanding economic problems. Studies from information economics (Arnott *et al.* 2003) demonstrated the profound effects of imperfect information on the existence and characterisation of market equilibriums and how the change of view from a perfect world to an imperfect world affected economic and political policies.

This thesis takes its inspiration from the study of information economics and the change of perspective from economics being situated in a world of perfect knowledge to economics actually existing in an imperfect world. The same question can be asked about intelligence: Is human intelligence a perfect end product with the complete knowledge of the world and consistent all the time with no changes and adaptation? The answer is obviously no. Artificial intelligence is about the creation of human-like intelligence; this is what makes artificial intelligence different from computational intelligence. The success of *Blondie24* compared with other "intelligent" machines lies in the fact that its "intelligence" was evolved from scratch. However, *Bloind24* is still a perfect end product in which there is no adaptation of intelligence or growth of knowledge. The change in the paradigms of economic studies inspired us to rethink about artificial intelligence research from a different perspective: artificial intelligence for an imperfect world. Therefore, we propose *imperfect evolutionary systems* as an

alternative research paradigm for AI in the study of continuous learning and adaptation in incomplete environments.

1.2 Overview

Chapter 2 describes the problem that this thesis focuses on. Starting with a literature review on artificial intelligence research and a discussion on the definition of AI, attention is particularly given to machine learning techniques, paradigms and the development of evolutionary artificial neural networks. The chapter then focuses on one of the promising research domain of machine learning, computer game playing. With an in-depth analysis of the evolutionary computer checkers program, *Blondie24,* and a comprehensive survey on existing techniques and paradigms in the field of evolutionary computation, including genetic algorithms, evolutionary strategies and evolutionary programming, parallel genetic algorithms, learning classifier systems and genetic programming, memetic algorithms, ant colony optimisation, particle swarm optimisation and cultural algorithms, chapter 2 concludes by proposing a new paradigm for artificial intelligence research and evolutionary computation.

Chapter 3 provides a general introduction to the stock market and the study of stock markets as evolutionary systems from two aspects: evolutionary computation and financial engineering and evolutionary computation and artificial stock markets. Various types of multi-agent based artificial stock markets are described including learning classifier systems based artificial stock markets, genetic programming based artificial stock markets, and neural network based artificial stock markets. Individual

and social learning for modelling complex evolutionary systems are also discussed. The chapter then discusses the stock market as an imperfect information system, and the inspiration it brings us for the change from a perfect paradigm to an imperfect paradigm in the context of evolutionary computation.

Chapter 4 follows the inspiration from chapter 3 and presents an imperfect evolutionary system as a solution to the problem as to how machine learning can respond and adapt to new challenges from incomplete environments. This chapter formally defines what an imperfect evolutionary system is. It also analyses the characteristics of an imperfect evolutionary system. An individual and social learning paradigm is proposed as a potential framework for implementing imperfect evolutionary systems.

Chapter 5 includes the first part of our experimental studies on imperfect evolutionary systems. In this chapter, a perfect simulated stock market model is developed. The model is discussed in detail with regards to the modelling of artificial stock traders and the modelling of individual and social learning of artificial intelligent agents. Experiments on a single stock and different types of stocks of the simulated stock market demonstrates the robustness of the individual and social learning paradigm, which lays at the foundation of the transformation from a perfect evolutionary market to an imperfect evolutionary market in the next chapter.

Chapter 6 presents the second part of our experimental studies on imperfect evolutionary systems, which transforms the perfect market model in chapter 5 to an

imperfect evolutionary system through the modelling of an incomplete market environment and the modelling of imperfect artificial stock traders. Experiments demonstrate how intelligent individuals in an imperfect market react to the new challenges from their environments. More importantly, the formation of new knowledge and the dissemination and absorption of new knowledge within the imperfect evolutionary society by artificial stock traders with imperfect intelligence is investigated. The role of social learning in the individual and social learning paradigm is also analysed. Chapter 6 concludes with a discussion on the significance of studying imperfect evolutionary systems and potential future research directions.

Chapter 7 summarises the contributions from this thesis into two subjects: imperfect evolutionary systems and evolutionary modelling of financial markets, together with recommendations for future research.

1.3 Contributions

This thesis has the following contributions:

1. A survey and analysis of the machine learning techniques and artificial intelligence research which reveals that artificial intelligence often fails to respond to new challenges from the environment as it assumes a perfect learning paradigm.
2. A survey and analysis of information economics and stock markets as imperfect information systems indicating a change from a perfect information

paradigm to an imperfect information paradigm provides us a new perspective on AI research.

3. Imperfect evolutionary system is proposed as a new evolutionary learning paradigm for continuous learning in incomplete environments. Experimental studies demonstrate the continuity and adaptability of intelligence in an imperfect evolutionary system with the presence of new challenges from its evolving environments.
4. An individual and social learning paradigm is introduced as a general framework of imperfect evolutionary systems. Experimental studies demonstrate the robustness of the learning paradigm.

Chapter 2

Artificial Intelligence: Literature Review

This chapter presents an overview of artificial intelligence research. Section 2.1 to 2.5 discusses different areas of AI with an emphasis on machine learning, artificial neural networks and computer game playing. Section 2.6 describes a co-evolutionary computer checkers program, *Blondie24*, which learns to play checkers without pre-injected human knowledge. Section 2.7 considers various existing evolutionary paradigms for AI research. The chapter concludes with proposing a change from a perfect learning paradigm to an imperfect learning paradigm for solving the problem of how artificial intelligence is able to respond to new challenges from the environment.

2.1 Definition

Since the word, artificial intelligence, was first coined by John MaCarthy in the Darmouth summer conference in 1956, there is still not a unified definition of the term. Various definitions of AI have been given by different researchers depending on their own point of view. Russell and Norvig (2003) classifies the definitions of AI into four categories:

Systems that think like humans: For example, Bellman (1978) defines AI as the automation of activities that we associate with human thinking, activities such as decision-making, problem solving, learning, etc.

Systems that act like humans: For example, Rich and Knight (1991) define AI as the study of how to make computers to do things at which, at the moment, people are better.

Systems that think rationally: For example, Winston (1992) defines AI as the study of the computations that make it possible to perceive, reason and act.

Systems that act rationally: For example, Poole *et al.* (1998) define AI as the study of the design of intelligent agents.

The existence of the diverse understanding of artificial intelligence is also often referred to as *weak AI* and *strong AI*. Weak AI asserts that machines could possibly act intelligently (or perhaps better, act as if they were intelligent), whereas strong AI asserts that machines that do so are actually thinking, i.e., machines would count as having minds like human (Russell and Norvig 2003). Strong AI is more concerned with the simulation between machine and man, machine intelligence and human intelligence. Weak AI is more concerned with adding "thinking-like" features to computers to make machines become more useful tools. This thesis adheres to the belief of strong AI. We argue that many AI techniques, especially when considering their underlying mechanisms, are far different from the way human intelligence works. However, the ultimate goal of artificial intelligence research is the recreation of human-like intelligence in machines. As commented by George Luger in his book (Luger 2005):

> *"... Although artificial intelligence, like most engineering disciplines, must justify itself to the world of commerce by providing solutions to practical problems, we entered the field of AI for the same reasons as many of our colleagues and students: we want to understand and explore the mechanisms of mind that enable intelligent thought and action ..."*

In this thesis, we draw upon a definition given by the American Association of Artificial Intelligence (www.aaai.org) as our working definition:

> *Artificial intelligence research is the scientific understanding of the mechanisms underlying thought and intelligent behaviour and their embodiment in machines.*

2.2 Overview

In the early years of AI, Alan Turning, in his well-known *Turing Test* (Turning 1950), tried to provide an operational definition for machine intelligence. The Turing test says a computer is considered as possessing intelligence if it passes the Turing test, where a human interrogator, after posing some written questions to both a human being and the computer, cannot tell whether the written responses come from a person or a computer. In order to pass the Turing test, the machine has to understand the question from the interrogator in natural language form, and have the capability of reasoning out an appropriate reply for the interrogator. Therefore, artificial intelligence is a diverse field where researchers address a wide range of problems that relate to intelligence. Following are a few major disciplines commonly found in the field of AI:

Natural Language Processing (NLP). Jurafsky and Martin (2000) describe natural language processing research as the study of building systems that are able to process and manipulate natural language used by human beings, which include studies such as speech recognition, natural language understanding, information retrieval and extraction etc. Early research in NLP mainly adopted a rationalist approach in the sense that it used rule-based representations of grammars and knowledge that were hand coded by the NLP system developer. Recent research trends in NLP focus on empirical methods where learning techniques are employed so that computers extract information from a natural language rather than the system developer explicitly encoding the computer with information of the language. Allen (1995) provides an extensive coverage of language processing from an AI perspective. Brill and Mooney (1997) give an overview on recent empirical NLP studies.

Knowledge representation and reasoning. Reasoning problems are concerned with starting from one or more given initial states and eventually reaching some pre-defined goal state. Therefore, the efficiency of a reasoning system is dependent on the number of transitions from one state to another before reaching the goal state. In order to increase the efficiency of a reasoning system, we need to minimise the number of intermediate states, which indirectly requires the knowledge used by the reasoning system to be well organised/represented. Accordingly, automated reasoning procedures are needed to make the appropriate knowledge available at a given problem state (Brachman and Levesque 2004). One of the important applications of knowledge representation and reasoning techniques is in the development of *Expert*

Systems (Giarratano and Riley 1998; Jackson 1999), which combine a knowledge base containing the problem solving knowledge for a specific domain from human experts, and an inference engine which interprets inquiries and processes the rules which are stored in the knowledge base.

Machine learning. If knowledge representation is about how to inject knowledge into a computer, machine learning is more concerned with how a machine can learn, or accumulate knowledge from experience so that machines can adapt themselves in a changing environment. The thesis investigates the problem of how machines can respond and adapt to new challenges from its environment through learning. Therefore, we discuss machine learning techniques, in particular evolutionary learning, in details in the following sections of this chapter.

Soft computing. As described in Konar (2000), soft computing differs from conventional (hard) computing in that it is tolerant of imprecision, uncertainty, partial truth, and approximation. The role model for soft computing is the human mind. The guiding principle of soft computing is: Exploit the tolerance for imprecision, uncertainty, partial truth, and approximation to achieve tractability, robustness and low solution cost. The major constituents of soft computing are Fuzzy Logic (Zadeh 1965, 1973, 1975; Bezdek 1993), Neural Computing (Fausett 1994), Evolutionary Computation (Yao 1999a; Fogel 2000), Machine Learning (Mitchell 1997) and Probabilistic Reasoning (Russell and Norvig 2003, pp. 462-613). In this thesis, special

attention is given to artificial neural networks in section 2.4 and evolutionary computation in section 2.7.

Computer vision. Computer vision research focuses on an artificial intelligent entity's visual perception of its physical environment and how useful information is extracted from the perceived image in order to fulfil tasks such as manipulation, navigation and object recognition (Forsyth and Ponce 2002). Computer vision consists of two stages: image formation and image processing. During the stage of image formation, geometry and photometry process provide a model as to how objects in a three-dimensional scene will map onto a two-dimensional array of pixels. The stage of image processing is then divided into three phases. In low-level vision the raw image is smoothed to eliminate noise allowing features of the two-dimensional image to be extracted. In the mid-level phase, these features are grouped together to form two-dimensional regions. In the high-level phase, the two-dimensional regions are recognised as actual objects in a scene through various cues such as motion, texture, and shading etc. One of the early studies on computer vision was presented in Marr (1982), which attempted to seek the computational insight into the working of human visual system. Forsyth and Ponce (2002) provide extensive coverage on algorithms and techniques for image formation and processing.

Robotics. The study of robotics not only makes use of every single aspect of artificial intelligence, from natural language processing to machine learning, to machine vision, but also draws on many aspects from other fields such mechanical engineering and

electrical engineering. Traditional approaches for robotics believe that in order to formulate intelligent behaviours in a robot, the robot must build a representation of its environment from the inputs of its sensor data and then plan actions based on the model. This generally involves robotic mapping and localisation, and robotic reasoning and planning. However, there are a number of problems concerning traditional AI approaches for robotics such as the complexity of the real world to be modelled, the computational bottlenecks, and noisy sensors. Therefore, researchers looked for alternative approaches for formulating robot intelligence. One of them is the subsumption architecture proposed by Jones and Flynn (1993). A subsumption architecture assumes the resource for modelling an environment is limited or unavailable. A robot is organised by means of layering task-achieving behaviours, where intelligent behaviour emerges from interactions of simple rules of behaviour in a complex environment. This approach is also called the behaviour-based robotics. Please refer to Murphy (2000) for a general coverage on robotics.

2.3 Machine Learning

Mitchell (1997) defines machine learning as the study of algorithms that improve a machine's performance at some task through experience. This section covers a variety of machine learning techniques in AI including decision tree learning, instance-based learning, Bayesian learning, heuristic learning, reinforcement learning and genetic learning.

2.3.1 Decision tree learning

Decision tree learning problem considers how a machine learns to build a decision tree that can be used for some decision-making process. Figure 2.1 illustrates a typical learned decision tree, adapted from Quinlan (1986). This decision tree classifies Saturday mornings according to whether they are suitable for playing tennis. For example, the instance, <*Outlook = Sunny, Humidity = High, Wind = Strong*>, would be sorted down the leftmost branch of the tree and would therefore give a negative response, i.e., the tree predicts that PlayTennis = No.

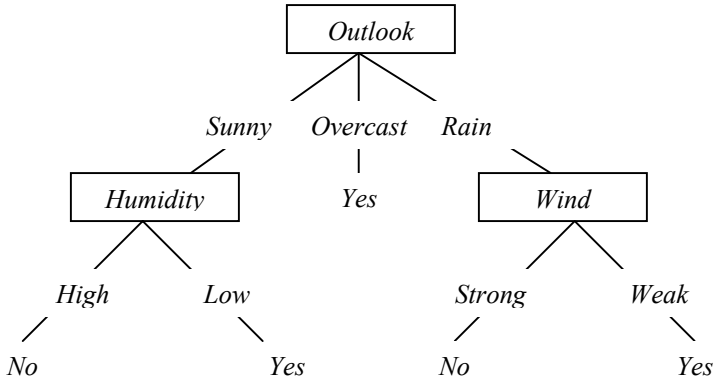


FIGURE 2.1 A learned decision tree for the concept *PlayTennis.*

In order to build a decision tree for a specific task, a decision tree learning algorithm must select which attribute to be tested at each node of each level of the tree, e.g., should *Humidity* be checked before or after *Outlook* in Figure 2.1. The ID3 algorithm (Quinlan 1986) is a classical learning algorithm that learns to build a decision tree by means of a top-down, greedy search approach through a complete hypothesis space of all possible decision trees based on the attributes set given by the problem as shown in Table 2.1, taken from Mitchell (1997).

ID3 (*Examples, Target_attribute, Attributes*)
{*Examples* are the training examples. *Target_attribute* is the attribute whose value is to be predicted by the tree. *Attributes* is a list other attributes that may be tested by the learned decision tree. The algorithm returns a decision tree that correctly classifies the given examples.}

Create a *Root* node for the tree
If all *Examples* are positive, return the single-node tree *Root* with label = "+"
If all *Examples* are negative, return the single-node tree *Root* with label = "-"
If *Attributes* is empty, return the single-node tree *Root*, with label = "most common value of *Target_attribute* in *Examples*".
Otherwise begin
  $A \leftarrow$ the attribute from *Attributes* that best classifies *Examples*, which has the highest information gain
  The decision attribute for *Root* $\leftarrow A$
  For each possible value, v_i, of A,
    Add a new tree branch below *Root*
    Let $Examples_{vi}$ be the subset of *Examples* that have value v_i for A
    If $Examples_{vi}$ is empty
      Then below this new branch add a leaf node with label = most common value of *Target_attribute* in *Examples*
    Else below this new branch add the subtree
      ID3($Examples_{vi}$, *Target_attributes, Attributes-{A}*)
End
Return *Root*

TABLE 2.1 An ID3 algorithm for decision tree learning.

As shown in Table 2.1, the ID3 algorithm builds a decision tree in a top-down manner. At each node the algorithm selects an attribute that best classifies the training examples provided for learning. The process continues until the tree perfectly classifies the training examples, or until all attributes have been used. ID3 uses a statistical property of attributes, called information gain, to select among candidate attributes.

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_V|}{|S|} Entropy(S_v). \quad (2.1)$$

Information gain, *Gain(S, A)*, measures how well a given attribute separates the training examples according to their target classification, as defined in equation 2.1. *Entropy(S)* is a measure of the impurity in collection *S*, which is equal to the minimum

number of bits of information needed to encode the classification of an arbitrary member of *S*. *Values(A)* is the set of all possible values of an attribute *A*, e.g., in Figure 2.1, we have possible values of "Strong" and "Weak" for attribute "Rain". S_v is the subset of *S* for which attribute *A* has value *V*. The higher a *Gain(S, A)* is, the greater information that the attribute *A* provides, hence greater chance attribute *A* is selected as a node in the tree.

Practical issues in decision tree learning include determining how deeply to grow a decision tree, handling continuous attributes, choosing an appropriate attribute selection measure, and handling training data with missing attribute values. Minger (1989) provides some experimental studies on different attribute selection measures. Quinlan (1993) developed a C4.5 algorithm, which extends from ID3 using post-pruning to find high accuracy hypotheses trees. Quinlan (1993) also provides discussions covering most of the practical issues mentioned above.

2.3.2 Instance based learning

Instance-based learning approaches store the past-learned instances in memory. When a new query instance is encountered, a set of similar related instances is retrieved from the memory and used to evaluate/classify the new query. Table 2.2 describes the standard K-NEAREST NEIGHBOR algorithm (Cover and Hart 1967) for instance-based learning, taken from Mitchell (1997).

K-NEAREST NEIGHBOR algorithm
(This algorithm approximates a discrete-valued function $f : R^n \rightarrow V$)
Training algorithm:
For each training example (*x*, *f*(*x*)), add the example to the list *training_examples*
Classification algorithm:

Given a new query instance x_q to be classified

Let $x_1 \ldots x_k$ denote the *k* instances from *traing_examples* that are nearest to x_q

Return

$$\hat{f}(x_q) \leftarrow \arg\max_{v \in V} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where δ(a, b) = 1 if a = b and δ(a, b) = 0 otherwise.

TABLE 2.2 A K-NEAREST NEIGHBOR algorithm for instance-based learning.

As we can see from Table 2.2, an instance-based learning method does not try to build a global function that can be used to evaluate any possible instances. Instead, it searches for the nearest related past instances, and builds an approximated local function $\hat{f}$ for the new instance based on the selected past instances. One of the most important practical issues in instance-based learning is to decide what it meant by *nearest* in Table 2.2, or in other words, how to decide if a past instance is within the neighbourhood of the new instance, i.e., closely related to the new query.

In the K-NEAREST NEIGHBOR algorithm, the nearest neighbours of an instance are defined in terms of the standard *Euclidean distance* as defined in equation 2.2,

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}. \qquad (2.2)$$

Where x_i is the *i*th instance. $a_r(x_i)$ is the *r*th attribute of instance x_i. *n* is the number of attributes of an instance. There are also other methods for selecting past instances for a new query. Distance-weighted NEAREST NEIGHBOR algorithms (Dudani 1975) weight the contribution of each of the *k* neighbours according to their distance to the new instance, giving greater weight to the closer neighbours. Locally weighted regression methods (Atkeson *et al.* 1997) construct an approximation function that fits all the past instances in the neighbourhood surrounding the new instance and then

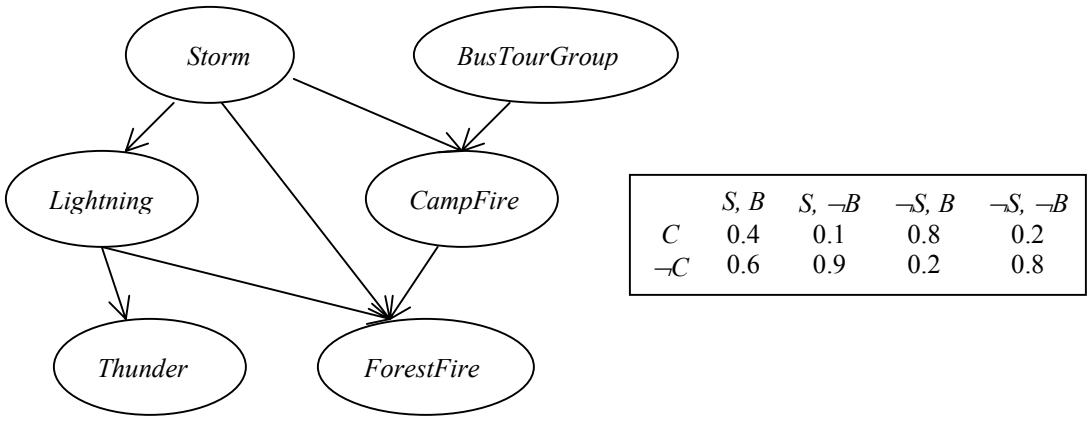
apply it to the new instance. Aha *et al.* (1991) provide a general discussion on instance-base learning algorithms.

Case-based Reasoning (CBR) is also a type of instance-based learning method. It also defers the decision of how to generalise beyond the training data until a new query instance is observed, and the classification of a new instance is done by means of analysing past related examples. The major difference between CBR and the classic K-NEAREST NEIGHBOR algorithm is how instances are represented. For K-NEAREST NEIGHBOR algorithms, instances are represented as real-valued points in a *n*-dimensional Euclidean space and Euclidean distances are used to decide how closely a past instance is related to a new query. In CBR, instances are represented using much more complex symbolic descriptions, and subsequently, the methods used to retrieve similar past instances are more elaborate. For example, in Burke *et al.* (2000), a CBR system is developed for course timetabling problems in which instances are represented as attribute graphs. In an attribute graph, nodes are used to represent courses while solid edges between nodes indicate hard constraints, and dotted edges indicate soft constraints for the timetabling problem. Labels on the edges and inside the nodes represent different restrictions. The authors, therefore, argue that their graph and attribute representation not only represents the components of an instance, but also the relationship between these components. When using graphs to represent instances in CBR, one of the practical issues is how *graph isomorphism* (i.e. how alike a graph is to another) is used to retrieve past instances. Aamodt and Plazas (1994)

provides a survey on case-based reasoning. Kolodner (1993) also presents an extensive examination of CBR issues.

2.3.3 Bayesian learning

Before discussing Bayesian learning, we need to understand Bayesian reasoning and Bayesian networks. Bayesian reasoning is a probabilistic approach for inference reasoning under uncertainties. In Bayesian learning, Bayesian networks are used as simplified probabilistic representations of the world based on independence and conditional independence over a set of variables. For instance, Figure 2.2 demonstrates a Bayesian network that gives the probability that a forest fire will happen, adapted from Mitchell (1997).



	S, B	*S, ¬B*	*¬S, B*	*¬S, ¬B*
C	0.4	0.1	0.8	0.2
¬C	0.6	0.9	0.2	0.8

FIGURE 2.2 A Bayesian network. The network on the left represents a set of conditional independence assumptions. The table on the right is the conditional probability table for *Campfire* node where *Campfire* is abbreviated to *C*, *Storm* abbreviated to *S*, and *BusTourGroup* abbreviated to *B*.

To understand Figure 2.2, we need define conditional independency. Let X, Y, and Z be three random variables. We say that X is conditionally independent of Y given Z if the probability distribution governing X is independent of the value of Y given a value

for *Z*. In Figure 2.2, the Bayesian network represents the joint probability distribution over the Boolean variables *Storm*, *Lighting*, *Thunder*, *BusTourGroup*, *CampFire*, and *ForestFire*. Each node in this network is conditionally independent of its non-descendants, given its immediate parents. The directed arcs in the network imply the relationship between nodes. For example, Figure 2.2 gives the assertion that *CampFire* is conditionally independent of its non-descendants *Lighting* and *Thunder*, given its immediate parents *Storm* and *BusTourGroup*. In other words, once we know the values of *Storm* and *BusTourGroup*, the variables *Lightning* and *Thunder* provide no additional information about *CampFire*. Each node in a Bayesian network is also associated with its conditional probability table, such as the conditional probability table for *CampFire* shown in Figure 2.2. For example, the top left entry in the table expresses the assertion that given *Storm* as true and *BusTourGroup* as true, the probability of *CampFire* being true is 0.4.

Bayesian reasoning studies the methods of how to use a Bayesian network to infer the values of some target variable given the observed values of other variables. The inference can be straightforward if the values of all the other variables in the network are known exactly. More generally, there may be only a subset of variables' values that are observable. A variety of methods have been developed for probabilistic inference in Bayesian networks. Exact inference methods, such as clustering algorithms (Pearl 1986) and variable elimination algorithms (Zhang and Poole 1994), evaluate sums of products of conditional probabilities as efficiently as possible. Approximate inference methods sacrifice precision to gain efficiency so that these

algorithms are able to cope with much larger Bayesian networks than is possible using exact inference methods. For example, Monte Carlo algorithms (Henrion 1988) provide approximate solutions by randomly sampling the distributions of unobserved variables from the known probabilities. Russell and Norvig (2003, pp. 504-516) provide discussions on various inference methods of Bayesian networks.

Bayesian learning deals with the issue of how to build a Bayesian network. In other words, developing effective algorithms for learning Bayesian networks from given training data. There can be several different situations in Bayesian learning, e.g., the network structure might be known in advance, or it might have to be inferred from the training data; all the network variables might be directly observable in each training example, or some might be unobservable.

In the case where the network structure is given, Russell *et al.* (1995) describe a gradient ascent procedure that searches through a space of hypothesis corresponding to the set all possible entries for the conditional probability tables. An alternative to the gradient ascent approach is the expectation maximization or EM algorithm (McLachlan and Krishnan 1997), which is particularly useful in the presence of unobserved variables. The EM algorithm starts with an arbitrary initial hypothesis. It then repeatedly calculates the expected values of the hidden variables assuming the current hypothesis is correct, and then recalculates the maximum likelihood hypothesis, along with estimated values for the hidden variables. In the case where the network structure is not known, most of the algorithms for learning the structures of

Bayesian networks involve searching in a hypothesis space of possible structures. For example, Cooper and Herskovits (1992) developed a Bayesian scoring metric for choosing among alternative network structures. Heckerman (1998) provides a survey on different approaches for learning Bayesian networks.

2.3.4 Heuristic Learning

Reeves (1995) describe heuristic methods as search techniques that find good (i.e. near-optimal) solutions at a reasonable computational cost. In this section, we discuss two heuristic learning algorithms: tabu search and simulated annealing.

Tabu search (TS) (Glover 1986, Glover and Laguna 1995) is a local search method that starts from a random solution, and keeps on searching in its neighbours for better solutions. In tabu search, the algorithm maintains a "tabu list" of previously visited solutions in order to avoid revisiting them. Table 2.3 outlines a tabu search algorithm, adapted from Glover and Laguna (1995).

Tabu Search

{ S^{now} is the current solution. S^{next} is the next solution. $N(S^{now})$ is the neighbourhood of current solution. M^T is the tabu list which is a memory of past visited solutions. f is the fitness of a solution.}

1. Random generate S^{now}.
2. Initialise tabu list M^T to empty.
3. Generate neighbourhood $N(S^{now})$
4. Select the best solution S^{next} among all (or a subset of) possible solutions in the neighbourhood $N(S^{now})$.

   If (S^{next} is not in M^T and $f(S^{next})$ is improved) Then

   $S^{now} \leftarrow S^{best}$.

   $M^T = M^T \cup S^{best}$.

   Repeat step 3 and 4.

   Else

   Terminate and return S^{now} as the local optima.

TABLE 2.3 A tabu search algorithm.

In general, there are two types of tabu lists, a *long-term memory* maintains the history through all the exploration process as a whole and a *short-term memory* only keeps the most recently visited tabu movements. In other words, some moves are freed after a certain period so that the algorithm has a better chance of escaping from local optima.

Simulated annealing (SA) (Kirkpatrick *et al.* 1983; Laarhoven and Aarts 1988; Eglese 1990, Aarts *et al.* 1997) is also a local neighbourhood search algorithm, which emulates the metallurgical annealing process. In a simulated annealing algorithm, moves to worse solutions are accepted with a specified probability, controlled by a parameter called "temperature", which decreases over the course of the search as shown in Table 2.4, adapted from Dowsland (1995).

Simulated Annealing (for a minimization problem)
{S is the solution space. N is the neighbourhood structure. f is the objective funcion }
Select an initial solution s_0.
Select an initial temperature $t_0 > 0$.
Select a temperature reduction function α.
Repeat
  Repeat
    Randomly select $s \in N(s_0)$.
    $\delta = f(s) - f(s_0)$.
    If $\delta < 0$ Then
      $s_0 = s$.
    Else
      Generate random x uniformly in the range (0, 1).
      If $x < \exp(-\delta / t)$ Then $s_0 = s$.
  Until interation_count = maximum.
  Set $t = \alpha(t)$. //decrease the temprature.
Until stopping condition = true.
Return s_0.

TABLE 2.4 A simulated annealing algorithm.

The major issue concerning simulated annealing is the sensitivity of parameters in the algorithm, such as the initial temperature t_0 or the reduction factor α. Careful selection of parameters and setting in the algorithm is critical to the success of SA.

Both tabu search and simulated annealing are local search algorithms. Both algorithms suffer from the problem of becoming trapped in local optima. In the next section, we will discuss a global heuristic learning method, *genetic algorithms*. Glover and Laguna (1995) and Dowsland (1995) provide extensive coverage on applications of tabu search and simulated annealing in various domains.

2.3.5 Genetic learning

2.3.5.1 GENETIC ALGORITHMS

Genetic algorithms (De Jong 1988; Goldberg 1989; Mitchell 1996; Fogel 2000) differ from other machine learning techniques in the sense that they are population-based evolutionary approach, which emulates the evolutionary process in nature. Genetic learning first creates a population of potential solutions to a particular problem, and then evolves this population of solutions by means of genetic operations such as crossover and mutation. Here we discuss genetic algorithms with an emphasis on their representations, selection schemes, and common reproduction operators. Genetic algorithms are generally implemented in a canonical form as shown in Table 2.5, adapted from Fogel (2000):

1. ***Fitness Function:*** The problem to be solved is defined and captured in an objective function called a fitness function that indicates the fitness of any potential solution.
2. ***Representation:*** A population of candidate solutions is initialised subject to certain constraints. Typically, a candidate solution is represented using a binary string, termed a *chromosome.*
3. ***Evaluation:*** Each chromosome in the population is evaluated and assigned a fitness score.
4. ***Selection:*** A selection scheme is applied to the population, such as *roulette wheel selection*¸ so that fitter solutions are selected as parents to generate new solutions.
5. ***Reproduction:*** The selected chromosomes generate new offsprings via the use of specific genetic operators such as *crossover* and *mutation.*
6. ***Elimination:*** A replacement strategy is applied to chromosomes from the current population and new generated chromosomes through reproduction, so that less fit

solutions are eliminated and a new population is ready for the next iteration in the evolution.
7. ***Termination:*** The process is halted if a suitable solution has been found or if the available computing time has expired. Otherwise the process proceeds to step (3), where the new chromosomes are scored and the procedure continues.

TABLE 2.5 A genetic algorithm.

As shown in Table 2.5, an individual in the population of potential solutions is termed a *chromosome*. Genetic algorithms emphasise the genetic encoding of a problem into chromosomes. The chromosomes/potential solutions are evolved by means of genetic operations, such as crossover and mutation, where useful "genes" are persevered through selection and crossover and mutation adds random variations into the search process. In genetic algorithms, chromosomes are typically represented as binary strings.

Before a genetic operation, such as crossover or mutation, occurs in order to generate new solutions, a GA has to select from the population of potential solutions so that individuals reproduce with a probability proportionate to their fitness. Various selection schemes exist including *roulette wheel selection*, *rank-based selection*, *tournament selection*, and *elitist selection* (Mitchell 1996):

- ***Roulette Wheel Selection***

Let $f_1, f_2, \ldots, f_n$ be fitness values of individuals 1, 2, …, n. The probability an individual i is selected for reproduction under roulette wheel selection is

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}. \tag{2.3}$$

It is obvious from equation 2.3 that individuals with higher fitness will have a higher probability of being selected. Roulette wheel selection may cause problems when the initial population contains one or two very fit individuals and the rest of the population are a lot less fit. In this case, the fitter individuals will quickly dominate the population and lead to premature convergence, i.e., the population converges to a very fit individual in relation to the rest of the population but one, which is still poor with respect to the global optima. In the case where individuals have very similar fitness values, roulette wheel selection also causes problems because it will be difficult for the population to move forward. One of the solutions to these problems is to scale the fitness values before they are used in calculating selection probabilities (Goldberg 1989).

- ***Rank-based Selection***

Rank-based selection first sorts all the individuals according to their fitness values, and then computes selection probabilities according to their ranks rather than their fitness values. Yao (1993) described a non-linear ranking-based selection as shown in equation 2.4:

$$p_i = \frac{i}{\sum_{j=1}^{n} j}. \tag{2.4}$$

where n is the number of individuals in a population. These n individuals are ordered in a non-descending order, and denoted as $0, 1, \cdots, i, \cdots, n$.

- ***Tournament Selection***

Both roulette wheel selection and rank-based selection need global information from the whole population as shown in equations 2.3 and 2.4. Tournament selection, on the other hand, only considers a subset of the whole population. In general, to run a tournament selection, a tournament size N is chosen. The procedure then is to randomly choose N individuals from the population and let them compete with each other. The fittest individuals are selected as parents to produce new offsprings.

- ***Elitist Selection***

Elitism is usually used in addition to other selection schemes. Elitist selection always preserves the fittest individuals (normally expressed as a percentage of fittest individuals from the population) to the next generation. Therefore, elitist selection increases a GA's speed of convergence, but elitist selection has similar problems to roulette wheel selection. For example, elitist selection has the risk of resulting in the GA search process being trapped in local optima since the inherited best individuals might not be close to the global optima.

After individuals are selected from the current population of potential solutions as parents, a GA then produces new offspring/solutions by means of crossover and mutation. Table 2.6 illustrates the common reproduction operators for genetic algorithms, adapted from Mitchell (1997). *Single-point crossover* cuts the parent chromosomes at a randomly chosen point and swaps the resulting sub-chromosomes. In *two-point crossover*, offspring are created by substituting intermediate segments of one parent into the middle of the second parent string. *Uniform crossover*, combines bits sampled uniformly from two parent chromosomes. The uniform crossover shown

in Table 2.6 uses a crossover mask as 10011010011, i.e., for the first offspring, 1 from crossover mask means sample a bit from first parent at this position whereas 0 means sample the bit from the second parent instead; the second offspring is created in the opposite way. *Point mutation* produces small random changes to the chromosome by choosing a single bit at random and changing its value. Mutation usually happens after the crossover operation.

	Initial strings	Offspring
Single-point crossover:		
	11101001000	11101010101
	00001010101	00001001000
Two-point crossover:		
	11101001000	11001011000
	00001010101	00101000101
Uniform crossover:		
	11101001000	10001000100
	00001010101	01101011001
Point mutation:		
	11101001000	11101011000

TABLE 2.6 Common reproduction operators for genetic algorithms.

Early work in simulated genetic systems can be traced back to Fraser (1957, 1960), Bremermann (1958) and Holland (1962). Holland (1975) is generally recognised as one of the seminal works on genetic algorithms. Major issues concerning genetic algorithms include their convergence rate to an optimal solution and the avoidance of premature convergence to local optima. Yao (1993) discusses the impact of different genetic operators on the performance of GAs. Fogel (2000) provides discussions on the theory of convergence for GAs. Genetic algorithms have been successfully applied to various machine learning problems such as function approximation (De Jong *et al.*

1993) and learning network topology and connection weights for artificial neural networks (Whitley *et al.* 1990; Billings and Zhang 1995). GAs have also proven to be successful in solving high-dimensional optimisation problems, for example, the NP-hard TSP problem (Whitley *et al.* 1991), control system design problems (Kristinsson and Dumont 1992), job-shop scheduling problems (Petrovic and Fayad 2004) etc. More applications of GAs can be found in Goldberg (1989), Davis (1991), Mitchell (1996) and Coley (1999).

2.3.5.2 EVOLUTIONARY STRATEGIES AND EVOLUTIONARY PROGAMMING

Together with genetic algorithms, evolutionary strategies and evolutionary programming all belong to a class of population-based stochastic search algorithms, termed *evolutionary algorithms* (EAs). The major differences among them lie in the representation of problems and the reproduction operators used. Evolutionary Strategies (ES) (Rechenberg 1965; Schwefel (1965, 1981)) were developed for numerical optimisation problems, where potential solutions are represented as vectors of real numbers rather than binary strings. Evolutionary strategies are generally characterised with a deterministic selection scheme, Gaussian mutation, and discrete or intermediate recombination. There are two major deterministic selection schemes in evolutionary strategies (Yao 1999a).

- $(\mu + \lambda)$ - ES: μ parents are used to create λ offspring. All individuals, i.e. the $(\mu + \lambda)$ solutions, compete and the best μ solutions are selected as parents for next generation.

- (μ,λ) - ES: μ parents are used to create λ offspring, but only the λ offspring compete for survival and the μ parents are completely replaced each generation.

Table 2.7 presents an implementation of (μ,λ) evolutionary strategies, adapted from Yao (1999a) and Fogel (2000).

1. Generate the initial population of μ individuals. Each individual is a real-valued *n*-dimensional vector, where *n* is the number of parameters to be optimised.
2. Evaluate the fitness value for each individual of the population.
3. Generate λ offspring by adding a Gaussian random variable with zero mean and preselected standard deviation to each dimension of an individual.
4. Evaluate the fitness of each offspring.
5. Sort the λ offspring into a non-descending order according to their fitness values, and select the μ best offspring out of λ to be parents of the next generation.
6. Stop if the stopping criterion is satisfied; otherwise go to step 3.

TABLE 2.7 An implementation of (μ,λ) evolutionary strategies.

Table 2.7 describes mutation-based evolutionary strategies, i.e. offspring are generated by applying Gaussian mutations on parents. Equations 2.5 and 2.6 describe a Gaussian mutation operation used in Yao (1999a).

$$\eta_k'(j) = \eta_i(j)\exp(\tau' N(0,1) + \tau N_j(0,1)) \quad (2.5)$$

$$x_k'(j) = x_i(j) + \eta_k'(j) N_J(0,1) \quad (2.6)$$

At each generation, the standard deviation $\eta_i(j)$ of each individual *i* on the *j*th dimension is updated using equation 2.5 and generates the new standard deviation $(\eta_k'(j))$ for offspring *k*. The new offspring $x_k'(j)$ is then generated using equation 2.6. $N(0,1)$ is a normally distributed one-dimensional random number with mean 0 and standard deviation 1. $N_J(0,1)$ indicates that the random number is generated anew for

each dimension. The factors τ and τ' are usually set to $(\sqrt{2\sqrt{n}})^{-1}$ and $(\sqrt{2n})^{-1}$ (n is the total number of dimensions in the problem). Evolutionary strategies also employ recombination operators for the generation of new offspring. Two major recombination operators are commonly used: discrete recombination and intermediate recombination. Discrete recombination is similar to uniform crossover in genetic algorithms (see Table 2.6), where new offspring are generated by randomly mixing components from two parents. Intermediate recombination combines the components from two parents through some polynomial functions to produce new offspring.

Evolutionary Programming (EP) (Fogel *et al.* 1966; Bäck and Schwefel 1993; Fogel 1994) also uses vectors of real numbers as its representations of potential solutions for a problem to be solved. The most noticeable difference between evolutionary strategies and evolutionary programming is that evolutionary programming does not use any recombination or crossover, but uses a kind of tournament selection as its selection scheme and Gaussian mutation as its only reproduction operator (Yao 1999a). Table 2.8 below describes a typical implementation of evolutionary programming algorithms (Bäck and Schwefel 1993; Yao 1999a; Fogel 2000).

1. Generate the initial population of μ individuals.
2. Evaluate the fitness value for each individual of the population.
3. Each parent creates a single offspring by means of Gaussian mutation (see equation 2.5 and 2.6).
4. Evaluate the fitness of each offspring.
5. Select μ individuals form the union of both the parents and the offspring generated by Gaussian mutation through a stochastic tournament selection process (see section 2.3.5.1). The selected μ individuals form the population for next generation.
6. Stop if the stopping criterion is satisfied; otherwise go to step 3.

TABLE 2.8 An implementation of evolutionary programming algorithms.

Evolutionary strategies and evolutionary programming with real-value representations and Gaussian mutation are particularly useful in optimising real-valued numerical parameters. Michalewicz (1992) points out that for real-valued numerical optimisation problems, floating-point representations outperform binary representations because they are more precise and lead to faster execution. In section 2.4.3, we will look at mutation-based evolutionary algorithms concentrating on their applications in evolutionary artificial neural networks.

2.3.6 Reinforcement learning

Sutton and Barto (1998) define reinforcement learning as a computational approach to learning whereby an agent tries to maximise the total amount of reward it receives when the machine is interacting with a complex, uncertain environment. What makes reinforcement learning problems different from other machine learning problems is the problem reinforcement learning faces is a sequence of actions rather than single isolated function approximations. Each of the actions in the sequence will account for the final outcome. Each of the successive actions will result in an immediate, or local, reward for the learning agent. The aim of reinforcement learning is to maximise the final accumulative reward when the learning agent reaches the goal state by learning an optimal control strategy that selects an optimal sequence of actions. The concept of reinforcement learning is illustrated in Figure 2.3 below, adapted from Mitchell (1997).

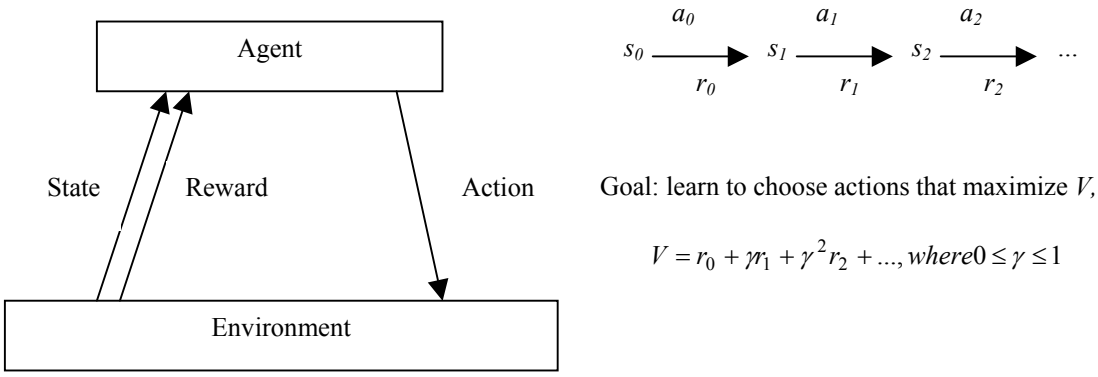


FIGURE 2.3 Reinforcement learning with an agent interacting with its environment. Each time the agent performs an action a_t in some state s_t, the agent receives a real-valued reward r_t. This produces a sequence of states s_i, action a_t and immediate rewards r_t. the learning agent's task is to learn a control policy that maximises the expected sum of these rewards V, with future rewards discounted exponentially by their delay.

Figure 2.3 in fact describes a Markov decision process (MDP). In a Markov decision process a learning agent perceives a set S of distinct states of its environment and has a set A of actions that it can perform. At each discrete time step t, the agent senses the current state s_t, chooses a current action a_t, and performs it. The environment responds by giving the agent a reward $r_t = r(s_t , a_t)$, and by producing the succeeding state $s_{t+1} = \delta(s_t , a_t)$. In a MDP process, $\delta(s_t, a_t)$ (next state given s_t and a_t) and $r(s_t, a_t)$ (reward given s_t and a_t) depend only on the current state and action, and not on any earlier states or actions. The task of a learning agent is to learn a policy $\pi: S \rightarrow A$ for selecting actions based on current observed states, i.e., $\pi(s_t) = a_t$. The aim of the agent is to learn the optimal policy π^* that produces the greatest possible cumulative reward over time as defined in equation 2.7 below,

$$\pi^*(s) = \underset{a}{\text{argmax}} \; [r(s, a) + \gamma V^*(\delta(s, a))]. \qquad (2.7)$$

which says the optimal action in state s is the action a that maximizes the sum of the immediate reward $r(s, a)$ and the value (discounted by γ) of following the optimal

policy thereafter. V^* is the maximum discounted cumulative reward that the agent can obtain starting from state $\delta(s, a)$ (see Figure 2.3).

Q learning, one of the reinforcement learning algorithms, defines the term $[r(s, a) + \gamma V^*(\delta(s, a))]$ in equation 2.5 as a Q function,

$$Q(s, a) \equiv r(s, a) + \gamma V^*(\delta(s, a))$$

So that the learning problem becomes:

$$\pi^*(s) = \underset{a}{\text{argmax}} \; Q(s, a)$$

Table 2.9 presents the Q learning algorithm, taken from Mitchell (1997), which iteratively approximates the Q function. Symbol $\hat{Q}$ refers to the learning agent's estimation of the actual Q function. The algorithm represents hypothesis $\hat{Q}$ by using a large table with a separate entry for each state-action pair, $\hat{Q}(s, a)$. The algorithm repeatedly observes its current state s, chooses some action a, executes the action, and then observes the resulting reward r and the new state s'. The algorithm then updates the table entry for $\hat{Q}(s, a)$.

Q learning algorithm
For each s, initialise the tale entry $\hat{Q}(s, a)$ to zero.
Observe the current state s
Do forever:
  Select an action a and execute it
  Receive immediate reward r
  Observe the new state s'
  Update the table entry for $\hat{Q}(s, a)$ as follows:

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

  $s \leftarrow s'$

TABLE 2.9 Q learning algorithm.

Q learning algorithms represent *Q*-functions in tabular form with one output value for each input value. As the state spaces get larger, the time to convergence increases rapidly for *Q* learning. One way to solve this problem is through function approximation (Russell and Norvig 2003, pp. 777-781). Other reinforcement learning methods include utility function learning where a model of the environment is needed and policy learning which maps directly from states to actions. Reinforcement learning is particularly useful in complex domains. For example, in computer game playing, chess and backgammon have state spaces that contain in the order of 10^{50} to 10^{120} states (Russell and Norvig 2003, p. 777). It is very hard for a human to provide accurate and consistent evaluations of such a large number of positions, which would be needed if a learning algorithm tries to learn an optimal evaluation function for playing chess or backgammon. Another complex domain where reinforcement learning is particularly well suited is robot control where successful behaviours may consist of thousands or even millions of primitive actions. Dean *et al.* (1993), Kaelbling *et al.* (1996) and Sutton and Barto (1998) provide extensive discussions on reinforcement learning and its applications.

2.4 Artificial Neural Networks

The study on Artificial Neural Networks (ANNs) has been inspired by the study of biological neural systems, which consist of very complex webs of interconnected neurons and react to external stimuli from the environment. In this section, we look at artificial neural networks with regards to network architectures, learning methods, and in particular, evolutionary artificial neural networks.

2.4.1 Perceptrons and multi-layer perceptrons

The *perceptron* (Rosenblatt 1959) was one of early artificial neural network models developed to classify patterns through supervised learning. Figure 2.4 illustrates the basic processing unit of a perceptron. A processing unit receives a set of inputs, $a_1,\cdots,a_n$. A special input, a_0, termed a *bias*, is always fixed at value of +1. Each connection between an input *i* and the processing unit *j* has an associated weight, w_{ji}. *f* is called as activation function, which scales the output from the processing unit into a certain range.

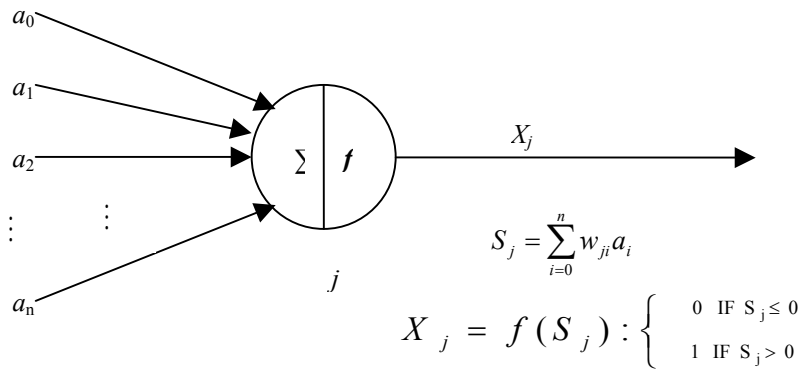


FIGURE 2.4 Perceptron basic processing unit

The perceptron processing unit performs a weighted sum of all input values as shown in equation 2.8,

$$S_j = \sum_{i=0}^{n} w_{ji} a_i \qquad (2.8)$$

where w_{ji} is the weight associated with the connection from input a_i to unit *j*. The perceptron then gives the output X_j by using an activation function *f* to scale the weighted sum to a certain range as shown in equation 2.9,

$$X_j = f(S_j) : \begin{cases} 0 \text{ IF } S_j \leq 0 \\ 1 \text{ IF } S_j > 0 \end{cases} \quad (2.9)$$

The activation function used in equation 2.9 is also called the *step function*, which tests if the weighted sum is above or below a threshold value that, in fact, equals to the value of w_{oj}, i.e., the connection weight between the bias a_0 and the unit j.

Figure 2.5 demonstrates a two-layer perceptron network that has an input layer and an output layer of processing units. Each unit in the input layer simply uses the input value as its output. The second layer of units then carries out the computations described in equations 2.8 and 2.9.

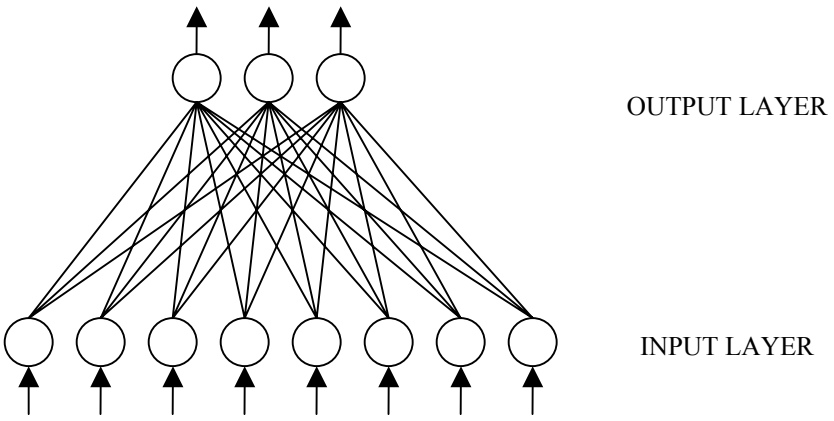


FIGURE 2.5 A two-layer perceptron.

A learning task for a perceptron network is to automatically tune its weights so that the network produces the desired output for given inputs. The two-layered perceptron described in Figure 2.5 can be trained for function approximation problems using a simple perceptron learning rule, which adjusts the network weights based on the differences between the target output and the network output after the training data is presented to the network, as shown in equation 2.10. Usually the training data set is

repeatedly presented to the network until the output from the network is same as, or close enough to the target output.

$$w_{ji}^{new} = w_{ji}^{old} + c(t_j - x_j)a_i \tag{2.10}$$

w_{ji}^{new} is the updated weight from the current weight w_{ji}^{old}, which is the weight associated with the connection between input a_i and output node *j*. t_j is the target output. x_j is the actual output from the network. *c* is a constant called the *learning rate*, which controls the scale of the updates, and thus the speed of learning.

The two-layer perceptrons can be successfully trained for solving a number of function approximation and pattern classification problems. Rosenblatt (1962) demonstrated the convergence of the perceptron learning rule. However, Minsky and Papert (1969) proved that two-layered perceptron (or named as single-layer perceptron in some textbooks since the input layer does not do any computation) cannot learn to represent simple functions such as XOR due to its limited representational capabilities in representing non-linearly separable functions, where linearly separable means that a pattern can be separated into two classes by drawing a single line (or a plane in higher dimension). In order to enhance the capability of single-layer perceptrons, perceptrons with multiple layers and non-linear activation functions were used as activation functions to bring nonlinearity into perceptrons. Figure 2.6 shows a Multi-layer Perceptron (MLP) with one hidden layer between the input layer and the output layer. The MLP is fully connected and feedforward, i.e. a neuron/processing-unit is connected to all the neurons in the previous layer and signals flow through the network in a forward direction, from left to right on a layer-by-layer basis.

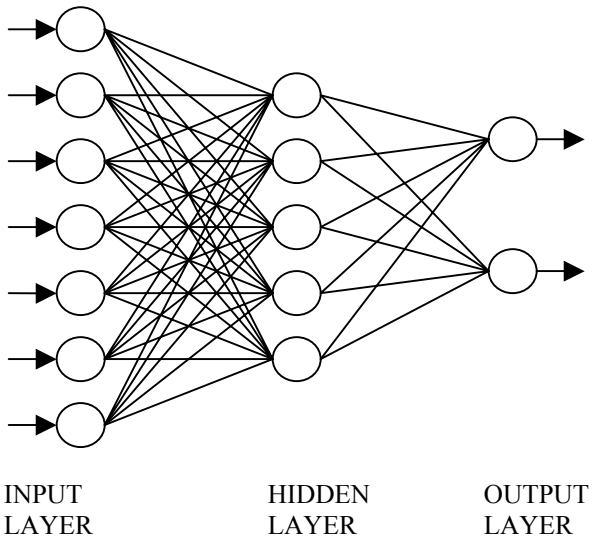


FIGURE 2.6 A three-layer percepton

Table 2.10 lists some of commonly used linear and non-linear activation functions used in artificial neural networks.

Name	*Formula*	*Range of Output*
Step function	Step(x) = 1 if x $\geq$ 0, else 0.	0 or 1
Sign function	Sign(x) = +1 if x $\geq$ 0, else -1	$\pm$ 1
Sigmoid function	Sigmoid(x) = $1(1+e^{-x})$	(0,1)
Hyperbolic function	Tanh(x) = $(e^{x}-e^{-x})/(e^{x}+e^{-x})$	(-1,1)

TABLE 2.10 Some commonly used non-linear activation functions in multi-layer perceptrons

Sigmoid and hyperbolic functions are the two commonly used non-linear activation functions for MLPs. MLPs cannot be trained by using the simple learning rules shown in equation 2.8 that are used for single-layered perceptrons. The problem is that we now have one or more hidden layers between the input layer and the output layer. How errors, i.e. the difference between the target output and the actual output ($t_j - x_j$), are propagated through each layer becomes crucial during the adjustment of connection weights. Paker (1985) and Rumelhart *et al.* (1986) developed the well-known *Backpropagation* algorithm for the training of multi-layer perceptrons.

2.4.2 Backpropagation learning and other neural network models

The backpropagation algorithm learns the connection weights for a multi-layer perceptron given a network with a fixed set of units and interconnections. In other words, the architecture of the neural network is known in advance. To use a backpropagation algorithm for training, the MLP must have at least one hidden layer, but not necessarily be fully connected. Table 2.11 presents a backpropagation algorithm for learning a three-layer feedforward MLP network using the sigmoid function as an activation function, adapted from Mitchell (1997).

BACKPROPAGATION **Algorithm** (*training_examples*, η, n_{in}, n_{out}, n_{hidden})

{*Each training example is a pair of the form* $\langle \vec{x}, \vec{t} \rangle$, *where* $\vec{x}$ *is the vector of network input values, and* $\vec{t}$ *is the vector of target network output values.* η *is the learning rate.* n_{in} *is the number of network inputs.* n_{out} *is the number of output units.* n_{hidden} *is the number of units in the hidden layer. The input from unit i into unit j is denoted* x_{ji}. *The weight from unit i into unit j is denoted as* w_{ji}.}

Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} outputs.
Initialise all connection weights to small random numbers (e.g., between -.05 and .05).
Until the termination condition is met, Do

For each $\langle \vec{x}, \vec{t} \rangle$ in *training_examples*, Do

Propagate the input forward through the network:

1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

Propagate the errors backward through the network and update weights:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1-o_k)(t_k-o_k)$$

3. For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1-o_h)\sum_{k \in outputs} w_{kh}\delta_k$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

$$\Delta w_{ji} = \eta \delta_j x_{ji}$$

TABLE 2.11 Backpropagation algorithm for multi-layer perceptron learning. The trained MLP network is a three-layer feedforward neural network as shown in Figure 2.6, with a sigmoid function used as activation functions for processing units.

As shown in table 2.11, the backpropagation algorithm updates the weights of output layer units using the error between the target output and the actual output from the network as shown in step 2. This error is then propagated backwards to a hidden layer unit in the form of a weighted sum of all connections between this hidden layer unit and all output layer units. Backpropagation is essentially a gradient descent method that minimises the error between the target output and actual output from the network. For mathematical analysis of the backpropagation algorithm, please refer to Fausett (1994) and Russell and Norvig (2003, pp. 744-748).

Besides multi-layer perceptrons with backpropagation learning, many other neural network topologies have also been explored by different researchers. Recurrent networks are neural networks that possess at least one feedback connection. This is different from feedforward networks where signals only flow through the network in a forward direction. Some signals in a recurrent network flows in a backward direction. To illustrate this concept, Figure 2.7 shows a *Simple Recurrent Network* (SPN) (Elman 1990). The simple recurrent network is largely similar to a three-layer feedforward multi-layer perceptron except that there is a set of "context units" in the input layer. However, there are also connections from the hidden layer back to the context units fixed with a weight of 1 as shown as curved lines in Figure 2.7.

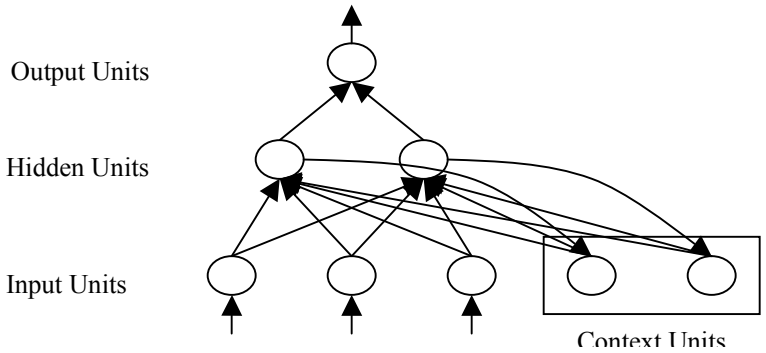


FIGURE 2.7 A simple recurrent neural network.

At each time step, the inputs from training data are propagated in a standard feedforward fashion, and then a learning rule (usually backpropagation) is applied. The fixed feedback connections between the hidden units and context units result in the context units always maintaining a copy of the previous values of the hidden units (since they propagate over the connections before the learning rule is applied). In other words, these context units keep a memory of information in previous time steps. This makes particular sense when the network is trained to predict a time series. For example, when a recurrent neural network is used to predict stock prices, the feedback connections will enable the network to look back further in the price history. Recurrent networks are powerful for learning complex sequences, but obviously, recurrent networks have more complex network structures, and hence, can be difficult to design and train. Several training methods have been proposed for learning recurrent networks, such as the gradient-based method proposed in Williams and Zipser (1995). Hopfield networks (Hopfield 1982) are recurrent neural networks in which all connections are symmetric, i.e., if there is a connection from unit *i* to unit *j*, there must be a connection from unit *j* feedback to unit *i*. Other popular neural network models include Radial basis function (RBF) networks (Park and Sandberg 1991), probabilistic neural networks (PNN) (Specht 1990), and Kohonen self-organizing maps (SOM) (Kohonen 1997). Good introductions to neural netowrks can be found in Anderson and Rosenfield (1988), Fausett (1994) and Callan (1999).

Multi-layer perceptrons with backpropagation learning have proven successful in many practical problems such as financial time series predictions (Zirilli 1996), computer game playing (Tesauro and Sejnowski 1989), and industrial applications

(Schlang *et al.* 1996). However, backpropagation training on multi-layer perceptrons has a few drawbacks. First, backpropagation learning is a gradient descent method that minimises the mean square error between target and actual outputs over all trainning examples. It tends to become trapped in a local minimum and is incapable of finding global minima if its error function is multimodal and/or indifferentiable (Sutton 1986). Secondly, using backpropagation for neural network training requires the network archituecure to be known in advance. There are no generally accepted protocols in ANN research that specifies how many layers and hidden units and what activation functions should be used for a particular application. Developers usually design a network either based on their experiences or through tedious trials on different number of layers and hidden units. With attempts to solve these problems, evolutioanry approaches for learning artificial neural networks have been explored.

2.4.3 Evolutionary artificial neural networks

Yao (1999) describes an evolutionary learning approach that can be introduced into ANN's at three different levels: *connection weights*, *network architectures*, and *learning rules*. Using evolutionary algorithms, such as genetic algorithms, to evolve connection weights provides a global search method for network weight training, and avoids the problem of being trapped in a local minimum caused by gradient descent learning. Evolving network architectures without human intervention provides an approach to automatic ANN design as both ANN connection weights and structures can be evolved. The evolution of learning rules can regarded as a process of "learning how to learn" in ANN's where the adaptation of learning rules is achieved through

evolution. (Please refer to Moriarty and Mikkulainen (1997) and Yao (1999) for comprehensive surveys on evolutionary artificial neural networks.)

2.4.3.1 EVOLVING CONNECTION WEIGHTS

The evolutionary approach to weight training in ANN's has two major phases. The first phase is to decide on the representation of the connection weights, i.e., whether to use binary strings or real-valued vectors. The second phase is to decide the genetic operators to be used for the evolutionary process in conjunction with the representation scheme. Table 2.12 describes the typical cycle of the evolution of connection weights, taken from Yao (1999). The evolution stops when the fitness is greater than a predefined value or the population has converged.

1. Decode each individual (a chromosome represents all connection weights) in the current generation into a set of connection weights and construct a corresponding ANN with the weights.
2. Evaluate each ANN by computing its total mean square error between actual and target outputs. Other error functions can also be used and problem-dependent. The higher the error, the lower the fitness. A regularization term may be included in the fitness function to penalize large weights.
3. Select parents with higher fitness for reproduction.
4. Apply search operators, such as crossover and/or mutation, to parents to generate offspring, which form the next generation of potential connection weights.

TABLE 2.12 A typical cycle for evolving connection weights in EANN's.

Although connection weights in a neural network are usually real numbers, early works in evolutionary ANN's (Caudell and Dolan 1989; Garis 1991) used binary strings as their representation scheme. In a binary representation, each connection weight is represented by a number of bits with a certain length. For example, the chromosome below consists of six connection weights:

0100	1010	0010	0000	0111	0011

where 0000 simply indicates no connection between two nodes. There are several encoding methods, such as the uniform method or the exponential method that can be used to encode real valued weights into binary bits using different ranges and precisions. One of the issues concerning binary representation is the trade off between the precision of binary representation and the length of the chromosome. If too few bits are used to represent a real-number connection, there may be problems of insufficient accuracy in representing real numbers with discrete values. If too many bits are used, the chromosomes become extremely long which leads to a loss of efficiency in the evolutionary algorithm (Whitley *et al.* 1990).

Representing all the connection weights of a neural network using a vector of real values provides direct manipulation of the connection weights and avoids the loss of precision in representation. For example, a representation of nine connection weights using a vector of real values could be:

0.005	0.123	0.569	0.000	-0.584	0.002	-0.900	0.325	-0.001

When connection weights are represented by real numbers, a better way to evolve a population of real-valued vectors is to use evolutionary strategies or evolutionary programming since ES and EP are suited to optimisation problems with continuous values compared to discrete binary representations. This is easy to understand because if the representations are vectors of real numbers, a crossover operation only creates new combinations of current connection weights, but mutation actually creates new values of connection weights that differ from the initial set of connection weights.

Mutation also avoids some other problems caused by crossover operators. For example, assume we have two different chromosomes as *(-0.1, 0.5, 0.6, 0.4, 0.5)* and *(0.02, -0.02, 0.6, 0.4, 0.5)*, with a single-point crossover at the second weight, we will have offsprings that are exactly same as their parents. Crossover operators are also inefficient in evolving neural network architectures, which we will discuss in the next section. A number of successful applications using evolutionary programming or evolutionary strategies evolving connection weights with real-valued representations can be found in Porto *et al.* (1995), Fogel *et al.* (1995), Yao *et al.* (1996), Greenwood (1997), Sarkar and Yegnanarayana (1997), and Chellapilla and Fogel (2001).

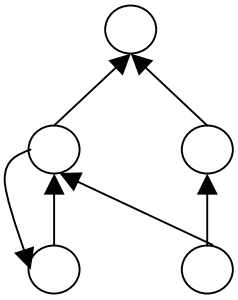
2.4.3.2 EVOLVING NETWORK ARCHITECTURES

The architecture of an ANN includes its topological structure, i.e. connectivity, and the activation function of each node in the network. Evolving artificial neural network architectures can be viewed as a search through a space of all possible network structures. Table 2.13 shows a typical cycle for evolving network architectures, adapted from Yao (1999). The process stops when a satisfactory ANN is found.

1. Each hypothesis of network architecture in the current generation is encoded into chromosomes for genetic operations, by means of a direct encoding scheme or an indirect encoding scheme.
2. Evaluation of fitness. Decode each individual in the current generation into an architecture, and build the corresponding ANNs with different sets of random initial connection weights. Train the ANNs with a predefined learning rule, such as the Backpropagation algorithm. Compute the fitness of each individual (encoded architecture) according to the training results, for example, mean-square-error, and other performance criteria such as the complexity of the architecture, e.g., less number of nodes and connections preferred.
3. Select parents from the population based on their fitness.
4. Apply search operators to the parents and generate offspring, which form the next generation.

TABLE 2.13 A typical cycle for evolving network architectures in EANN's.

The direct encoding scheme (Whitley *et al.* 1990; Fogel 1993; McDonell *et al.* 1994a) for encoding a network architecture specifies all the details of an architecture in a chromosome, i.e. every connection and node of the architecture. In general, a $N \times N$ matrix $C = (c_{ij})_{N \times N}$ is used to represent an ANN architecture with N nodes, where c_{ij} indicates the presence or absence of the connection from node i and node j, as shown in Figure 2.8.



$$\begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

00100 00110 10001 00001 00000

(a) (b) (c)

Figure 2.8 Direct encoding of neural network architecture.

The 5×5 matrix in Figure 2.8(b) can be easily transferred into a binary string as the one in Figure 2.8(c). Note that there is a feedback connection between the first node and the third node in Figure 2.8(a), which shows the evolutionary learning of an ANN architecture is restrained by the type of ANN to be learned. After network architectures are encoded into binary strings, crossover and mutation operators are used to evolve the population of encoded architectures. As mentioned in the previous section, crossover operations may cause inefficiency in the evolution of network architectures. This appears in several aspects. First, artificial neural networks, as we described in section 2.4.1, are a distributed representation form of knowledge, i.e. knowledge of solving a problem is stored in each node and each connection weight of the network. One single node or connection in general does not explain very much

about a problem. Instead, a group of hidden nodes with particular connection weights detect and extract certain features from the inputs, just as the brain can be divided into different regions that specialise in different functions. Crossover operators are more likely to destroy these useful feature detectors during the evolutionary process than the mutation process. Secondly, crossover operators also suffer from the negative impact caused by the permutation problem. A permutation problem (Hancock 1992; Igel and Stagge 2002) occurs when two ANN's that order their hidden nodes differently but still have the equivalent functionality. For example, in the diagram below:

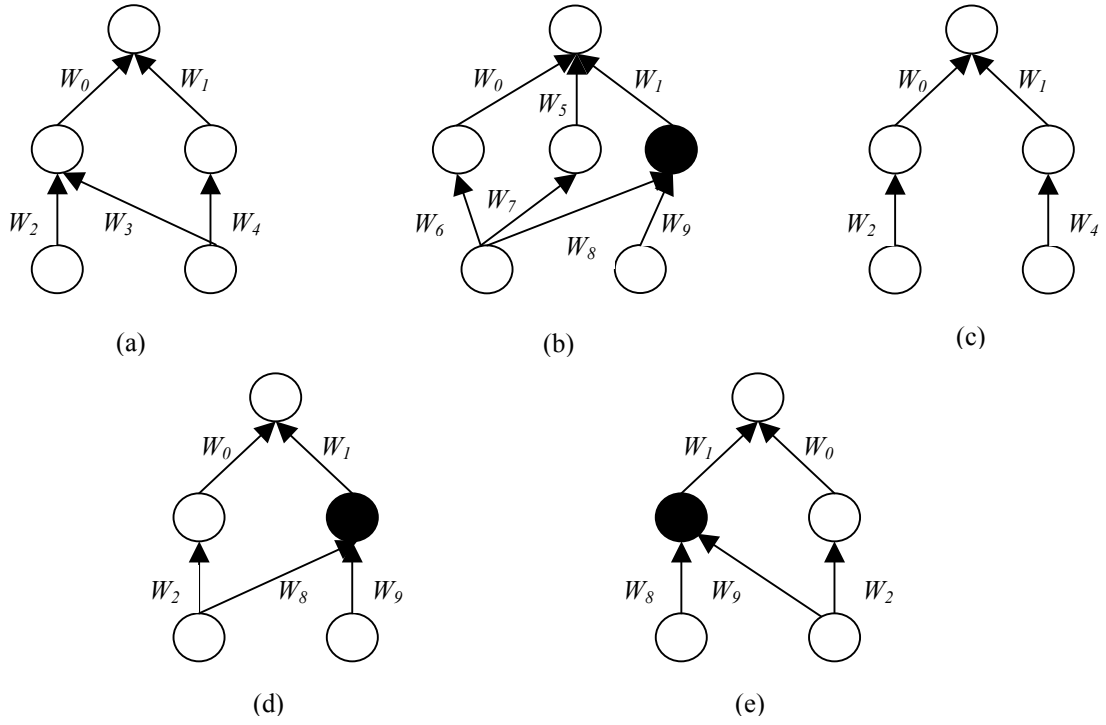


FIGURE 2.9 The permutation problem with crossover operation in EANN's.

Figure 2.9(a) to (c) represent three parent networks. Figure 2.9(d) shows a child network produced when applying crossover operation on Figure 2.9(a) and (b). Figure 2.9(e) is another offspring produced by applying crossover operation on (b) and (c). Note that Figure 2.9(d) has the same functionality as the network in Figure 2.9(e), but if we represent them as chromosomes, they are different. Figure 2.9(d) will be

represented as $(w_0, w_1, w_2, w_8, w_9)$. Figure 2.9(e) will be represented as $(w_1, w_0, w_8, w_9, w_2)$. This shows that different chromosome representations could in fact represent networks that are functionally equivalent. Therefore, crossover operation may reduce the population diversity and make the evolution process less efficient in evolving artificial neural networks. In general, crossover is not used as the primary operator in most EANN's applications (McDonnell and Waagen 1994; Heimes *et al.* 1997; Fang and Xi 1997; Yao 1997; Yao and Liu 1997a). There are some studies (Hancock 1992; Likothanassis 1997) that argue that crossover might be important for some problems. Stanley and Miikkulainen (2002) also argued that their method of crossover of different network topologies increased efficiency on benchmark reinforcement learning tasks. More research is needed to understand the efficiency of crossover operators in evolving artificial neural networks.

One of the issues concerning the direct encoding of network architectures is the length of the chromosome. As the size of the network grows, the chromosome will become longer, and hence reduce the efficiency of the evolutionary algorithm. Indirect encoding scheme (Kitano 1990; Harp *et al.* 1990; Gruau 1994; Grönroos *et al.* 1999) tends to reduce the length of the genotype representation of architectures. In an indirect encoding scheme only some characteristics of an architecture are encoded in the chromosome. For example, a *parametric representation* (Harp *et al.* 1990) may only contain a set of parameters such as the number of hidden layers and the number of hidden nodes in each layer assuming the networks are all feedforward MLPs. Obviously, this greatly restricts the choice of potential network architectures. Another

popular indirect encoding scheme is the *development rule representation* (Kitano 1990) that encodes development rules in chromosomes. These development rules specify certain primary building blocks in a network. The evolution of architectures is transferred into the evolution of development rules. The development rule representation can lessen the destructive effect of crossover due to its capability in preserving promising building blocks that have been evolved, but apparently more work is needed during the encoding and decoding stage of chromosomes. Development rule representation is not good at evolving detailed connectivity patterns among individual nodes. Development rule representation also separates the evolution of architectures and the evolution of connection weights, which makes it unsuitable for simultaneous evolution of architecture and connection weights. For more discussions on indirect encoding of network architectures, please refer to Moriarty and Mikkulainen (1997) and Yao (1999).

2.4.3.3 Simultaneous Evolution Of Architectures And Weights

In table 2.13 we described the evolution of network architecture as a separate process from the evolution of connection weights. This separation could cause noise problems, as described below, in the fitness evaluation of individual architecture hypothesis (Yao and Liu 1997). The first source of the noise comes from the random initialisation of connection weights when the individual architectures are evaluated, due to the fact that different random initial weights may produce different training results. The second source of the noise is caused by the training algorithms used for the evaluation. Different training algorithms may generate different training results even with the same set of initial weights.

One way to solve the problems caused by the separation of the evolution of the architecture and the connection weights is to evolve them simultaneously. In fact, by looking at Figure 2.8(c), we can simply replace the discrete binary values in the chromosome, i.e. 0 and 1, with real values that represent the values of the weights on the presented connections, so that the chromosome encodes both the network architecture and its associated connection weights. There are a number of studies on evolving architectures and connection weights simultaneously in EANN's. Yao and Liu (1997) describe an evolutionary system called EPNet, which uses evolutionary programming for the simultaneous evolution of ANN architectures and connection weights. The principle of using mutation as the primary operator and avoiding crossover that we discussed in previous sections is maintained in the design of EPNet, which uses a number of mutation operators for adjusting architectures and weights. Figure 2.10 shows the main structure of an EPNet, taken from Yao and Liu (1997).

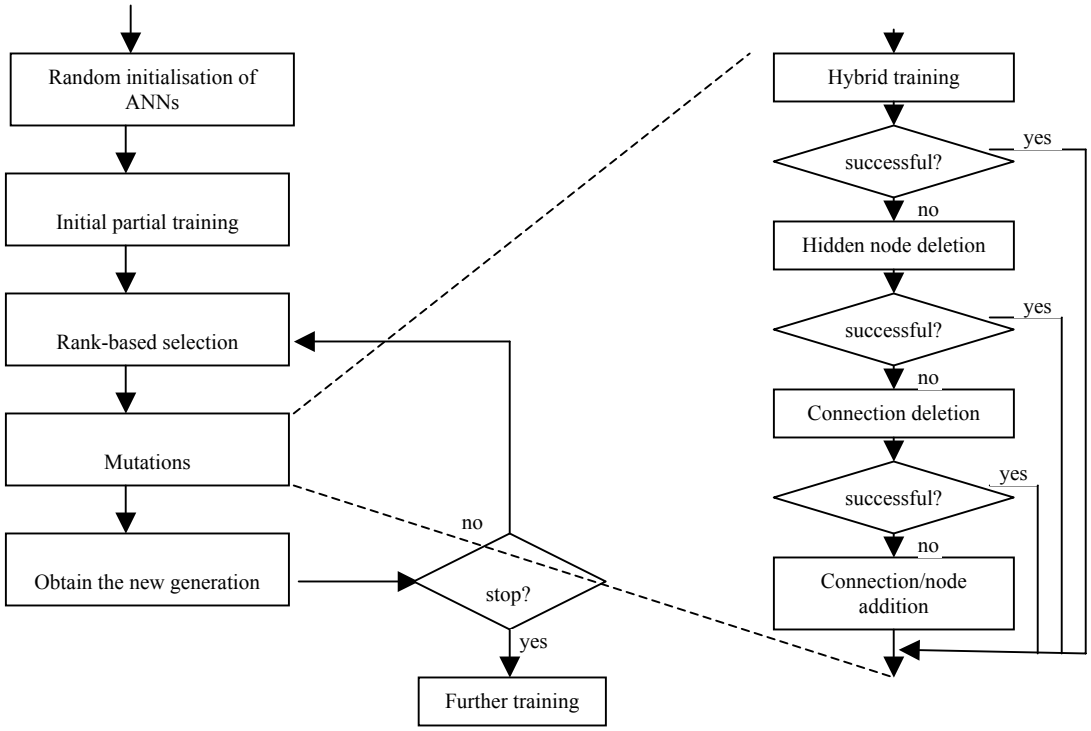


Figure 2.10 EPNet for simultaneous evolution of network architectures and connection

An EPNet uses initial partial training and hybrid training for adjusting connection weights through mutations by means of Backpropagation training and simulated annealing. During the initial partial training stage, as shown in Figure 2.10, each network in the population is trained repeatedly on the training data set for a certain number of epochs using a modified backpropagation algorithm (An epoch is a complete run of training on the whole training data set). The error value of each network on the test data set is checked after partial training. If the error has not been significantly reduced, the assumption is that network is trapped in a local minimum and the network is mark with "failure". Otherwise the network is marked with "success". The networks in the population are then ranked according to their error values in the partial training. Using rank-based selection, see section 2.3.4.1, one parent network is selected from the current population.

Two different types of training occur during the hybrid training stage: the modified Backpropagation training and simulated annealing. If the parent network is marked with "success", the modified Backpropagation training is applied to the parent network again to refine the network and generate a new offspring. The parent network is then replaced with the new offspring. A new parent is then selected using rank-based selection. If the parent network is marked with "failure", a simulated annealing algorithm (Kirkpatrick *et al.* 1983) is used to obtain offspring network. If the simulated annealing learning reduces the training error of the parent network significantly, mark the offspring with "success", and replace its parent in the current population. If the training error is not significantly reduced after applying simulated annealing, mutation occurs.

EPNet uses four mutation operators for evolving network architectures: node deletion, connection deletion, connection addition, and node addition. These four mutations are attempted sequentially. Note that it considers reducing the size of the network before it attempts to add new connections or nodes to the network, which ensures compact network architectures to be evolved. EPNet has been successfully applied to a number of practical problems (Yao and Liu 1995; Yao and Liu 1997; Yao and Liu 1997a). Other applications of evolving ANN architecture and connection weights simultaneously can be found in Koza and Rice (1991); Bornholdt and Graudenz (1992); White and Ligomenides (1993); Nikolopoulos and Fellrath (1994); and Maniezzo (1994).

The activation function is another important part of an ANN network architecture. In White and Ligomenides (1993), node activation functions are evolved by setting 80% of the nodes in the initial population using a sigmoid function and 20% of the nodes used a Gaussian function. The evolution decides the optimal mixture between these two activation functions automatically. In Liu and Yao (1996), EPNet randomly adds or deletes sigmoid nodes and Gaussian nodes in the current network architecture, and achieved good performance on some benchmark problems.

2.4.3.4 EVOLVING LEARNING RULES

We have introduced some well-designed learning rules for training artificial neural networks, such as the perceptron learning rule for single-layer perceptrons and the backpropagation algorithm for MLPs. Other types of learning rules for different type

of ANN's also exist, such as the Hebbian learning rule (Fausett 1994). In fact, we can assume any learning rules to be in a more general form as follows (Mitchell 1996):

$$W_{ji}(t+1) = W_{ji}(t) + \Delta W_{ji}$$

where

$$\Delta W_{ji} = f(a_i, o_j, t_j, w_{ji})$$

a_i is the input to unit *i*. o_j is the output from unit *j*. t_j is the targeted output from unit *j*. w_{ji} is the current weight on the connection from *i* to *j*. We can assume the learning rule f to be a linear combines of these variables as follows:

$$f(a_i, o_j, t_j, w_{ji}) = k_0(k_1 w_{ji} + k_2 a_i + k_3 o_j + k_4 t_j + k_5 a_i w_{ji} + k_6 o_j w_{ji} + k_7 t_j w_{ji} + k_8 a_i, o_j + k_9 a_i, t_j + k_{10} o_j t_j).$$

Thus a learning rule can be evolved through the evolution of a set of parameters $(k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10})$. Examples of evolving learning rules can be found in Chalmers (1990) and Baxter (1992).

2.5 Computer Game Playing

Game playing has been a very active research domain in artificial intelligence possibly due to the fact that people are more easily convinced that a machine can "think" if the machine can play games with them. Game playing also provides an excellent test bed for the study of artificial intelligence because game playing generally involves most of the important aspects of artificial intelligence such as knowledge representation, automated reasoning and machine learning.

Traditional game-playing computer programs employs a knowledge-based approach, where human expertise about a particular game is hand encoded into the computer by means of developer-designed evaluation functions or databases of well-developed opening and end games. In general, a legal-move generator is used to generate potential valid moves through the problem space. These possible moves are then organised in the form of a decision tree. In an ideal world the program would construct the game tree completely, i.e. until the endgame states (e.g. "win" or "lose") were achieved. The program could then search for a path through the tree that eventually led the program to win the game. However, it is often impossible to produce the entire game tree due to the limitations on computational time and memory space. For example, *chess* has an average branching factor of 35 and a complete game tree could have approximately 35^{100} positions to evaluate and search. Therefore, game-playing programs in general only build partial decision trees by looking ahead a limited number of moves. With the partial game tree constructed, an evaluation function is used to assess the relative worth of the potential outcomes from alternative moves. Rich and Knight (1991, pp. 310-314) provide discussions on the construction of game trees and their associated searching algorithms.

The seminal work of Arthur Samuel (1959, 1967) constructed computer programs that learnt to play checkers. In his program, Samuel used several parameters that he believed were crucial to win a checkers game, including piece advantage, capacity for advancement, control of the centre of the board, threat of a fork, etc. These parameters are then combined into an evaluation function by attaching an appropriate weight to

each parameter and adding the terms together. The complete evaluation function has the form:

$$c_1 \times pieceadvantage + c_2 \times advancement + c_3 \times centrecontrol + \cdots$$

Each terminal node in the game tree is then evaluated using the evaluation function above. Samuel's work was also one of the first that effectively used heuristic search methods in searching the game tree for next best move.

In general, for one-person games or puzzles, a simple A^* algorithm (Rich and Knight 1991, p. 76) can be used to search for the best move. For more complex two-player games, a *minimax* search algorithm is commonly used (Campbell and Marsland 1983; Kaindl 1990; Rich and Knight 1991; Fogel 2000). A minimax algorithm performs a complete depth-first exploration of the game tree. The algorithm first recurses down to each of the bottom left nodes in the tree, and uses the evaluation function to calculate the values of each left nodes/possible positions. It then chooses the best value and backs that value up to the root node/starting position, with the assumption that the opponent plays to minimise his utility/evaluation function. A best next move is chosen to maximise the machine's utility/evaluation function. Samuel (1959) used a minimax procedure to find the best next move from current position. Samuel (1967) improved the minimax search with alpha-beta pruning, which also incorporated a supervised learning mode that allowed the program to learn to select better parameters to be included in the evaluation function.

Samuel also acknowledged his program's reliance on human expertise. He wrote in his 1959 paper: "… *some effort might well be expended in an attempt to get the*

program to generate its own parameters for the evaluation polynomial." The challenge Samuel left open to other researchers was to design a program that could invent its own features and learn the game from scratch without human advice. In Samuel (1967), a supervised learning mode is incorporated so that the program learns to select better parameters for the evaluation function, which was one of the earliest efforts in developing computers that learn to play games without human prior knowledge. However, after Samuel's challenge, computer game researchers continued to pay more and more effort in injecting more knowledge into machines rather than in designing programs that learn by themselves. The other two pioneers in AI, Marvin Minsky and Allen Newell, questioned if it is possible for any learning to happen at all over available time scales when the only available information to a machine is "win, lose, or draw" (Minsky 1961).

Chinook (Schaeffer *et al.* 1996; Schaeffer 1997) was another successful checkers program. *Chinook* won the man-machine world championship in 1994. This was the first time a machine won a world championship. The success of *Chinook* is even more remarkable as it defeated the world checkers champion Marion Tinsley, coming 3 years before *Deep Blue* defeated Kasparov, the world chess champion at the time who had held the title for over 40 years. One of important characteristics of *Chinook* is that the program relied heavily on human expertise and brute search force to play the game. In *Chinook*, parameters used for the evaluation function were chosen by Schaeffer and his colleagues (including checkers grand masters) based on their personal knowledge about what makes a good checkers player. *Chinook* also used an

opening game database of forty thousand openings based on checker grand master's expertise, and a six-piece endgame database, which means that when the game enters the stage when there are only six pieces left, all *Chinook* has to do is to find a path from the endgame database that results in a guaranteed win, or at least a draw.

Deep Blue, a chess program developed by IBM, defeated world chess champion Garry Kasparov in 1997, which is seen as a triumph for artificial intelligence. Newborn (1996) describes the history of chess computer programs and the early works of *Deep Blue*. King (1997) provides more insights to the 1997 match. Campbell *et al.* (2002) attribute the success of *Deep Blue* to a number of factors: a single-chip chess search engine, a massively parallel system with multiple levels of parallelism, a strong emphasis on search extensions, a complex evaluation function, and effective use of a Grandmaster game database. *Deep Blue* used around 120 parameters for its evaluation function that are pre-programmed by the *Deep Blue* team based on knowledge accumulated from chess grand masters. Coefficients of parameters in the evaluation function were mainly hand tuned. Even during the match between *Deep Blue* and Garry Kasparov, developers of *Deep Blue* actually hand tuned the parameters and coefficients in the evaluation function in order to reflect the special characteristics of Kasparov's play. *Deep Blue* also attempted to automatically tune coefficients of parameters by means of supervised learning. *Deep Blue* gathered nine hundred games from chess masters, a supervised learning algorithm then adjust the parameter weights so that the evaluation from their function would produce the greatest match between the moves that the machine thought were best and those actually played by the

masters. A large database of end games was also used by *Deep Blue*. Another crucial factor in Deep Blue's success was its application-specific hardware that was specifically developed for *Deep Blue*, e.g. *Deep Blue* used 32 parallel processors and 512 custom designed chess ASICs which allowed a search of 200 million chess positions per second (Clark 1997). Therefore, *Deep Blue* is also largely dependent on pre-injected human knowledge and the parallel computer power at its disposal. In some sense, *Deep Blue* is more a triumph in computer design rather than any improvement in developing artificial intelligence.

The most severe criticism of traditional knowledge-based approaches for developing game-playing machine intelligence is centred on the large amount of pre-injected human expertise into the computer program and the lack of learning capabilities in these machines (Fogel 2000; Fogel 2002). The evaluation functions and opening and end game databases we described above are designed by program developers depending on either their own knowledge about a game or the expertise from game masters. In this sense, a computer game program's intelligence to play a game is not gained by actually playing a game, but rather comes from the pre-designed evaluation formulas and databases of moves. Moreover, this intelligence is not adaptive. It could be argued that human read books and watch other people playing a game before they actually start playing themselves. However, humans improve their skills through trial-and-error. New features and strategies for playing a game can be discovered by new players rather than grand masters, while old features could be viewed as useless and old strategies are discarded. Human also adapt their skills or strategies when they meet

different types of players under different circumstances in order to accommodate their special characteristics. We do not see such adaptations and characteristics in the knowledge-based computer game programs. Fogel (2002) commented on this phenomena in computer game-playing as follows:

> "*… To date, artificial intelligence has focused mainly on creating machines that emulate us. We capture what we already know and inscribed that knowledge in a computer program. We program computers to do things – and they do those things, such as play chess, but they only do what they are programmed to do. They are inherently "brittle". … We'll need computer programs that can teach themselves how to solve problems, perhaps without our help. …"*

Chellapilla and Fogel (2001) proposed an alternative approach in developing computer game programs. In their study, a new type of checkers program, called *Blondie24*, is developed by means of evolving artificial neural networks without pre-injected human knowledge. The study of applying evolutionary artificial neural networks in computer game playing points out a new direction for machine intelligence.

2.6 *Blondie24*

Blondie24 is a checkers program that learns to play the game of checker to a level that is competitive with human experts (Fogel 2000; Chellapilla and Fogel 2001; Fogel 2002). A significant difference between *Blondie24* and other traditional checkers

programs is in the evaluation function used. In traditional game-playing programs, evaluation functions usually take the form of weighted polynomials comprising features that were believed to be important for generating good moves based on game masters' expertise. The weighting of these features in the evaluation function also generally rely on hand tuning. In *Blondie24*, its evaluation function is an artificial neural network consisting of an input layer, three internal processing (hidden) layers, and an output layer. The inputs to the artificial neural network/the evaluation function only contain information about the positions of the pieces on the board and the piece differential. In other words, the neural network only knows the number of pieces on the board and their positions; no other human expertise on how to play checkers is pre-programmed into the neural network.

Initially, *Blondie24* was designed using only raw position and piece information. Therefore, each checkerboard can be represented by a vector of length 32, with each component corresponding to an available position on the board, and hence 32 input nodes in the input layer of the neural network. Components in the vector are values from $\{-K,-1,0,+1,+K\}$, where 0 indicates an empty position, 1 indicates one player's piece, -1 indicates one opponent's piece is on that position, and K is a random value assigned for a king. There were 40 nodes in the first hidden layer, and 10 nodes on the second hidden layer. During the later stages of *Blondie24*, when it was tested on an internet game site (http://www.zone.com), a spatial preprocessing layer with 91 nodes was added to the network as the first hidden layer with the aim being to extract the features of how pieces are associated with each other on the board by dividing the

board into subsections, as shown in Figure 2.11 (taken from Fogel 2002), together with a second hidden layer of 40 hidden nodes and a third hidden layer of 10 hidden nodes.

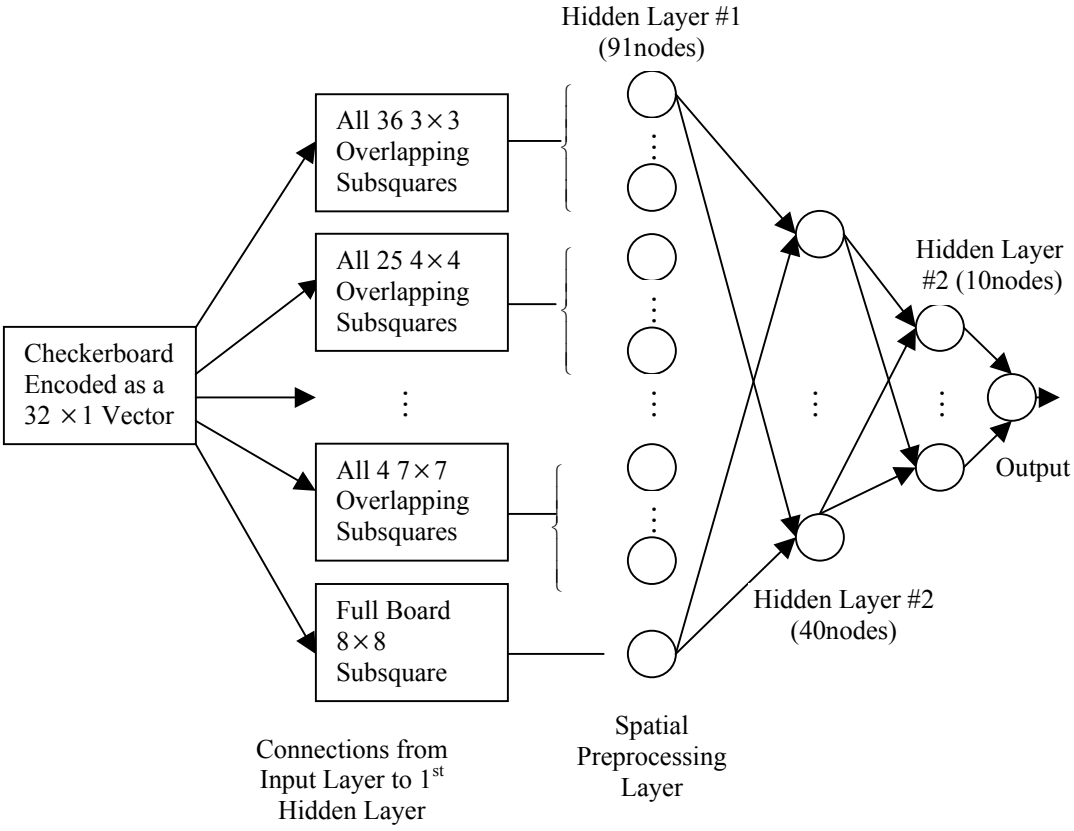


Figure 2.11 Using artificial neural network as evaluation function for *Blondie 24*

With no pre-injected human knowledge, the artificial neural network needs to learn to extract useful features from the given board information. This is done by means of evolution. A population of 30 artificial neural networks with the same structure as shown in Figure 2.11 is randomly created. Each neural network, in turn, plays games against five randomly selected opponents from the population. Each game was played using a minimax alpha-beta search (Kaindl 1990) on the associated game tree with 4 or 6-ply (a ply is a move in a game) look ahead. Each player scored -2, 0, or +1 points

for a loss, draw, or win. In total, there were 150 games per generation, with each network participating in an average of 10 games. After all games were complete, the 15 networks that received the greatest total points were retained as parents for the next generation.

Each parent in the current generation generated an offspring by varying all of the associated weights and biases, and possibly the king-value *K* as well. This is done in an evolutionary programming fashion as shown in section 2.3.5.2:

$$\sigma_i'(j) = \sigma_i(j)\exp(\tau N_J(0,1)),\ J = 1,\ldots,N_w$$

$$w_j' = w_j + \sigma_i'(j)N_j(0,1),\ j = 1,\ldots,N_w$$

and

$$K_i' = K_i + \delta$$

where N_w is the total number of weights and bias terms in the network. τ is of the value of 0.0839, and $N_j(0,1)$ is a standard Gaussian Random number. The mutation factor $\sigma_i'(j)$ is first mutated, and then the connection weights w_j' and bias terms are mutated with the updated $\sigma_i'(j)$. The value of a king piece, K_i' is also evolved where δ is chosen uniformly at random from {-0.1, 0, 0.1}.

This process of having the evolving neural networks play against each other was iterated for 840 generations, which took about 6 months. The best-evolved neural network was used as the final evaluation function in *Blondie24* to play against human opponents on an Internet game site (http://www.zone.com/). *Blondie24* played 165

games against human opponents over six weeks, and achieved a rating of 2045.85 with a standard deviation of 33.94 on the zone.com web site, which is rated at the human expert level.

A few reasons make *Blondie24* a significant advance in computer game playing. Firstly, *Blondie24* learns to play checkers to a human expert level without relying on any pre-injected human knowledge about the game such as hand-encoded evaluation functions or databases of opening and end games. The evolutionary artificial neural network learning mechanism in *Blondie24* more closely mimics human learning processes with trial-and-error. In this sense, *Blondie24* answers Samuel's challenge on designing programs that would invent their own features and learn how to play a game of checkers without pre-injected human knowledge as we discussed in last section. Secondly, *Blondie24* also answers the Newell's challenge on making a program learn just by playing games against itself and receiving feedback, not on each game, but only after a series of games (quoted in Minsky 1961, pp. 20). In *Blondie24*, a network plays games against another 5 networks from the current population, and received a total score for its play so far. Only the final total score counts towards the network's final fitness. The EANN's learning mechanism in *Blondie 24* can be viewed as a type of reinforcement learning where the evolution of artificial neural networks consists of a sequence of steps with a reward at each generation. Thirdly, the successful application of evolutionary algorithms in evolving artificial neural networks for game playing shows that evolution is an alternative way for the creation of artificial intelligence. Fogel (2000, 2002) argues that evolution accounts for the intelligent

behaviours in living organisms. By simulating the evolutionary learning processes on a computer, the machine can become intelligent. Indeed, we can find many other studies and applications in game playing on the recognition of intelligence as an evolutionary process, such as Turing (1950), Fogel *et al.* (1966), Fogel (1992), Axelrod (1987), Fogel (1993), Kendall and Hingston (2004) for Iterated Prisoner Dilemma problems, Moriarty and Miikkulainen (1995) for the game of Othello, Pollack and Blair (1998) for the game of Backgammon, Richards *et al.* (1998) and Kendall *et al.* (2004) for the game of Go, and Kendall and Whitwell (2001) for Chess.

However, strictly speaking, there is still a defect in the design of *Blondie24*, i.e. the lack of adaptability. After 840 generations of evolution, *Blondie24* kept the best-evolved neural network as its evaluation function, and played 165 games against human opponents. Note that the evaluation function, i.e. the best-evolved neural network, was not adaptable during the 165 games played. In this sense, *Blondie24* is more like an end product rather than an intelligent entity that is capable of adapting itself during its interactions with its environment. As commented by Eric Harley in his book review (Harley 2002):

> *"... An interesting point is that the end product which looks intelligent is Blondie, yet she is not in fact the intelligence. Like the individual wasp, Blondie is fixed in her responses. If she played a million games, she would not be iota smarter. In this sense, she is like Deep Blue. ... Perhaps a better example of intelligence would be ... a human, who can adapt her behaviour to any number of new challenges..."*

Although Fogel (2000, 2002) argued that the rating of *Blondie24* could be higher under further trainings on *Blondie24* after the 165 games with the EANN's learning paradigm, or even trained to play other games. However, this still does not change the fact that *Blondie24* does not learn during its interaction with its human opponents. If the developers of *Blondie24* do not provide further training on *Blondie24*, *Blondie24* will always be the same *Blondie24*. The creation of *Blondie24* is a learning process, but *Blondie24* itself does not have the capability to learn. No doubt, evolution is the fundamental resource for intelligence. However, it seems something, or some mechanism, is missing from the EANN's learning paradigm used in *Blondie24.* Before we attempt to look for this missing mechanism in the creation of human-like intelligence, we will look at other existing different types of learning techniques and paradigms in evolutionary computation in next section, followed by our proposal on a new evolutionary computation paradigm for artificial intelligence.

2.7 Evolutionary Computation

Yao (1999) defines Evolutionary Computation (EC) as the study of computational systems that use ideas and inspirations from natural evolution and adaptation. Besides the classic evolutionary algorithms, such as genetic algorithms, evolutionary strategies, and evolutionary programming that we introduced in section 2.3.5., there also exist various other techniques and paradigms in evolutionary computation, including parallel genetic algorithms, memetic algorithms, learning classifier systems,

genetic programming, ant colony optimisation algorithms, particle swarm optimisation algorithms, and cultural algorithms.

2.7.1 Parallel genetic algorithms

The major motivations behind parallel implementations of genetic algorithms (Stender 1993; Cantu Paz 1998; Cantu Paz 2000) include:

- For some problems, the populations in a GA need to be very large and the memory required to store each individual might be excessive. In some cases this makes it impossible to run an application efficiently using a single sequential implementation, hence some parallelisation form of GA is desirable.
- Fitness evaluation is usually very time-consuming. Parallel processing of fitness evaluations could greatly reduce computational time.
- Parallel implementations of GAs can search in parallel different subspaces of the hypothesis space, thus making it easier for the GA to escape from low-quality sub-regions of the search space.

Cantu Paz (1998) proposed four classifications of parallel GAs: global single-population master-slave GAs, single-population fine-grained GAs, multiple-population coarse-grained GAs, and hierarchical parallel GAs which are the combinations of the first three parallel GAs. Figure 2.12 below schematically describes the first three types of parallel GAs, adapted from Cantu Paz (1998).

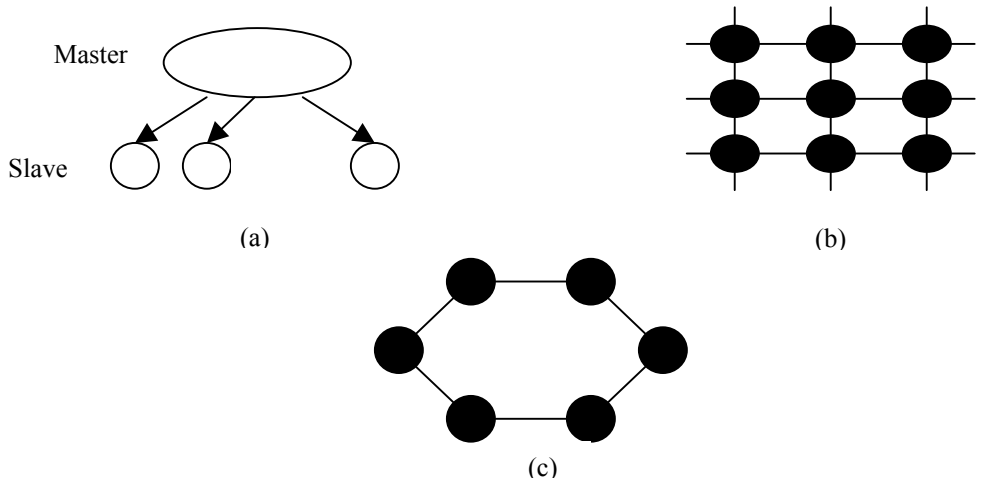


FIGURE 2.12 (a) master-slave parallel GA. (b) fine-grained parallel GA. (c) multiple-population parallel GA.

Figure 2.12(a) shows a master-slave parallel GA. The master machine stores the population to be evolved and executes GA operations. The master machine also distributes individuals to slave machines so that slave machines can evaluate the fitness of individuals in a parallel fashion. The slave machines only conduct fitness evaluations, not genetic operations such as crossover or mutation. Master-slave parallel GAs use global information, i.e. the entire population, for selection and genetic reproduction, because only a single population is saved in the master machine. In this sense, master-slave parallel GAs are the same as the classic simple GAs. One of the practical issues concerning master-slave parallel GAs is the communication overhead between the master and slave machines, which could greatly degrade the performance of the algorithm (Abramson *et al.* 1993; Hauser and Manner 1994). Nevertheless, master-slave parallel GAs are easy to implement and can increase the efficiency of GAs when the evaluation is computationally expensive.

Fine-grained parallel GAs, shown in Figure 2.12(b), also consists of one single population. Individuals from the single population are spatially structured, so that, ideally, there is only one individual for every processor. In Figure 2.12(b), each black dot represents an individual processing unit. The spatial structure limits the interactions between individuals. An individual can only compete and mate with its neighbours to produce new offspring. However, since the neighbourhoods overlap with each other, good solutions may disseminate across the entire population. Two problems need to be considered when designing a fine-grained parallel GA. First, what kind of spatial structure should be used for the population? A 2-Dimensional grid configuration, as shown in Figure 2.12(b), is very common in a fine-grained parallel GA since in most parallel computers their processing units are connected using this topology (Manderick and Spiessens 1989). However, because parallel computers also use global routers to send messages to any processor on the network, it is also possible to simulate other network topologies on the top of the grid. Anderson and Ferris (1990) experimented with the ring, hypercube, mesh and island structures and concluded that the ring and island structure were the best. Schwehm (1992) tested different population structures including ring, torus, cube and hypercube on a graph-partitioning problem, where the torus structure achieved the best performance. Another issue to be considered is the size and the shape of the neighbourhoods. Manderick and Spiessens (1989) observed that the performance of the algorithm degraded as the size of the neighbourhood increased. Sarma and Jong (1996) found that the ratio of the radius of the neighbourhood to the radius of the whole gird is a critical parameter in determining selection pressure for reproduction. Successful

applications of fine-grained parallel GAs can be found in job shop scheduling problems (Tamaki and Nishikawa 1992) and 2-dimensioanl bin-packing problems (Kroger *et al.* 1993).

Multiple-population parallel GAs, also called *distributed genetic algorithms* (DGAs) or *island GAs*, are the most popular forms of parallel genetic algorithms. Two important characteristics differentiate multiple-population from other types of parallel GAs: the use of several relatively isolated subpopulations and migration among subpopulations. As shown in Figure 2.12(c), each black node represents a subpopulation, or called a *deme*. By dividing a single population into subpopulations, the selection and mutation process of DGAs is restricted within a subset of individuals. For single population GAs, the selection and reproduction use global information over the entire population. For multiple-population parallel GAs, the selection and reproduction only uses local information within the subpopulation. This design of parallel genetic algorithms essentially mimics the spatial separation of conspecifics in nature. As in nature, the subpopulations in distributed GAs evolve separately with occasional migration between neighbouring populations. While we are trying to make genetic algorithms faster through parallelisation, we also need to make sure the performance of the genetic algorithm does not degrade due to the localisation of the search, i.e. search within subpopulations. In DGAs, the migration of good solutions among neighboured subpopulations plays a critical role. Problems such as how often the migration should happen (*the migration interval)*, how many individuals should be exchanged between subpopulations (*the migration rate)*, and

which individuals should be chosen for the migration determines the performance of distributed genetic algorithms. Pettey *et al.* (1987) developed a multi-population parallel GA where a copy of the best individual found in each deme is sent to all its neighbours after every generation. Their parallel GA was able to find solutions of the same quality as a sequential GA. Tanese (1989) conducted experimental studies on the frequency of migrations and the migration rate, and concluded that migration significantly increases the quality of solutions for distributed GAs. Migrations do not necessarily happen at predetermined constant frequencies. Braun (1990) described a DGA for the travelling salesman problem where migration occurred after the demes converged completely and the migration restores diversity into the demes to prevent premature convergence. Applications of multi-population parallel GAs can be found in function approximations (Muhlenbein *et al.* 1991), graph partitioning problems (Levine 1994), and synthesis of VLSI circuits (Davis *et al.* 1994) etc.

2.7.2 Memetic algorithms

While genetic algorithms emulate biological evolution, memetic algorithms (MAs) (Moscato 1989) attempt to mimic culture evolution where units of culture, called *memes*, evolve with local refinement. Memetic algorithms can be described as a combination between a population-based global search and a heuristic local search made by each of the individuals. The main difference between genetic algorithms and memetic algorithms is that in GAs individuals do not make local searches. Table 2.14 describes the basic form of memetic algorithms, adapted from Krasnogor (2002).

```
Memetic_Algorithm():
Begin
    t = 0;
    Generate an initial population;
    Repeat Until (Termination criteria fulfilled) Do
        Selection;
        Crossover;
        Mutate;
        Improve_by_local_search;
        Select_for_next_generation;
        t = t + 1;
    Od
    Return best solution(s);
End
```

TABLE 2.14 A memetic algorithm.

In most memetic algorithms, the local search used to refine an individual solution is a simple hill climbing algorithm, or more sophisticated heuristic methods such as simulated annealing, tabu search, greedy randomised adaptive search procedures (GRASP) and fuzzy adaptive neighbourhood search (FANS). Please refer to section 2.3.4 for discussions on simulated annealing and tabu search algorithms. GRASP combines the advantages of random construction and greedy construction of neighbouring solution i' (Resende and Ribeiro 2002). FANS treats the neighbourhood of an individual as a fuzzy set and evaluates solutions through a fuzzy evaluation (Krasnogor and Pelta 2002). Hart (1994) regarded local searches in memetic algorithms as individual learning, i.e. refining individuals with local search, improves the efficiency of genetic algorithms in two ways: the local searches may generate better solutions more efficiently than the GA's competitive selection, and the fitness of the refined solutions may reflect the domain-wide characteristics of the object function more accurately, especially when complete local searches are performed.

In Table 2.13, we place the local search after selection and reproduction. In practice, local searches could happen at any stages of the evolution. Figure 2.13 demonstrates this phenomenon, adapted from Krasnogor (2002).

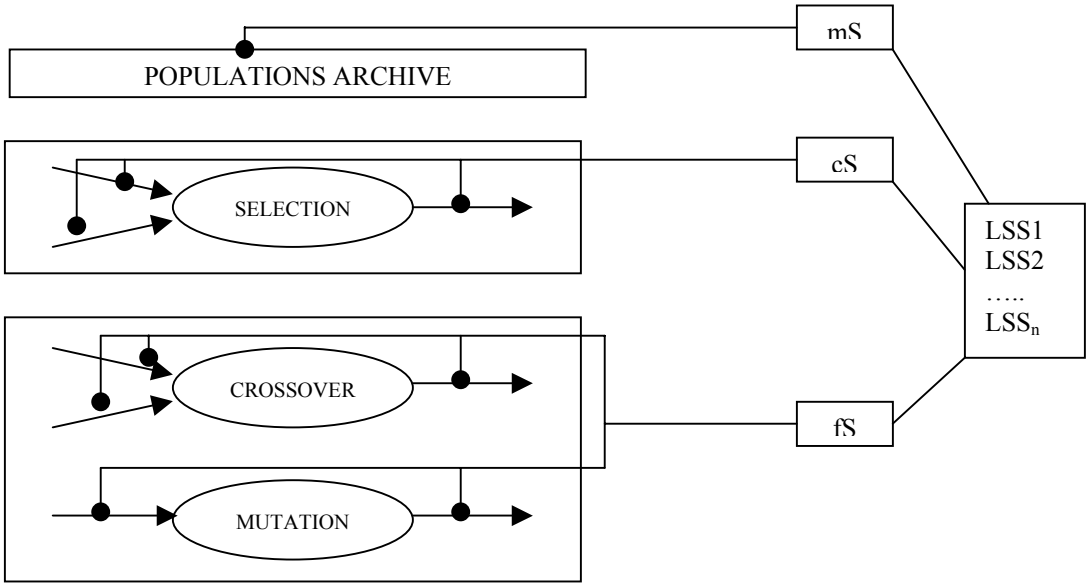


FIGURE 2.13 Local search in memetic algorithms

Local searches can occur before/after the selection, or before/after the crossover and mutation. *mS*, *cS* and *fS* are schedulers that coordinate memes/individuals' activities with various genetic operations, such as crossover and mutation. LSS_i are local search strategies, such as simulated annealing or GRASP that we described above. Figure 2.13 indicates there could be multiple local search methods available for memes. Applications of memetic algorithms can be found in network topology design problems (Dengiz *et al.* 1997), protein folding problems (Krasnogor and Smith 2000), multiobjective optimisation problems (Knowles and Corne 2000) etc.

2.7.3 Learning classifier systems

The following two evolutionary paradigms: learning classifier systems (LCS) and genetic programming (GP), are both based on genetic algorithms. Compared to the binary strings used by traditional genetic algorithms, LCS and GP employ complex, symbolic representations. Learning classifier systems (Holland (1988, 1995)) uses

production rules, called *classifiers*, as their representations for potential solutions. An example of a classifier can be described as:

Condition	Action	Strength
100100##	1	56

Each classifier contains three parts: <*condition*> : <*action*> : <*strength*>. *conditions* encode the environment using a binary string where 1 means that for some condition is true, and 0 means no such condition is present in the external environment. For example, for a medical diagnosis problem, the first bit in the condition represents fever; the second bit represents coughing. The example above says the patient has fever symptoms but not coughing. *Action* specifies the actions to be taken when a *condition* is matched. *Strength* specifies the usefulness of a production rule, which also guides the evolution of production rules in the rule set of the LCS system. Figure 2.14 illustrates the architecture of a learning classifier system (Holland 1988).

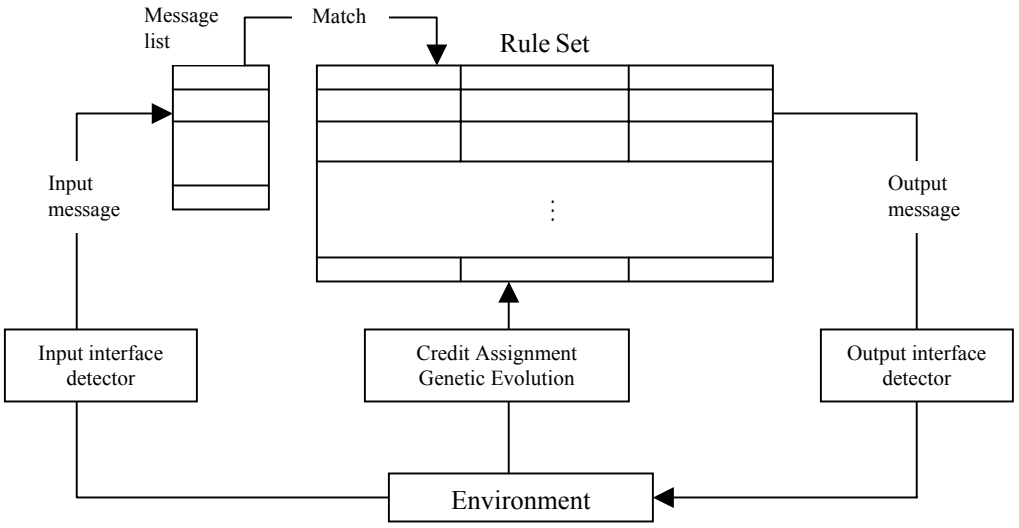


Figure 2.14 Learning classifier systems

The classifier system repeatedly reads messages from its environment, searches for matching conditions in the rule set, and then the corresponding action is taken. The environment evaluates the response from the classifier system and feeds the evaluation

back to the LCS by means of credit assignment. A genetic algorithm is then used to evolve the rules of the classifier system. Note that a # symbol in the condition of a rule means "don't care" which matches either 1 or 0. for example, 1### matches any messages with length of 4 starting with 1. Therefore, during the search for a matching rule, the LCS might find multiple rules matching the incoming message. Generally, more specific classifiers, i.e. containing less #, are favoured during the search.

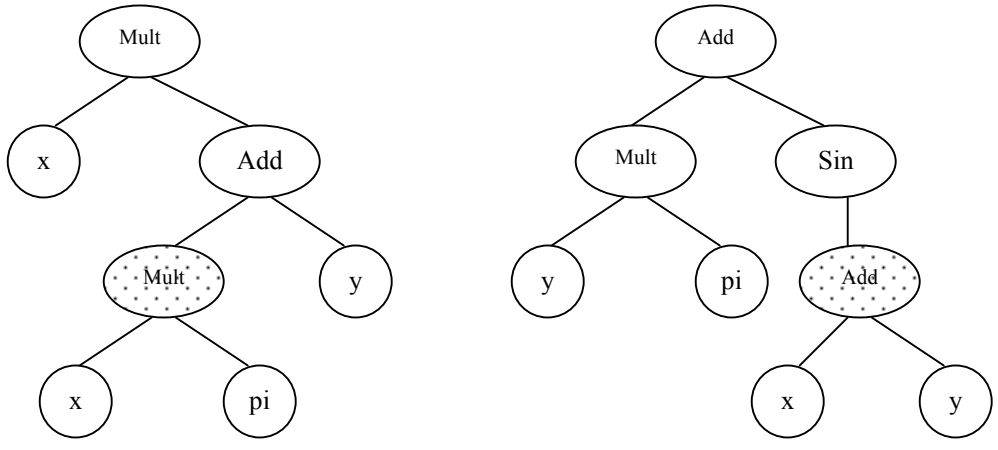
Credit assignment in LCS might be difficult because in some cases the system is only able to assign credits at certain intervals. In other words, we need to assign credits to early-acting classifiers/rules that set the stage of a sequence of actions leading to a favourable situation. One of the common credit assignment methods is the bucket brigade algorithm where an auction takes place to select and award a winning classifier from the set of matched classifiers (see Holland 1988; Goldberg 1989). Grefenstette (1988) also describes an epochal credit allocation scheme, called the profit sharing plan, where a constant fraction of the current reward is paid to every classifier that becomes active since the last receipt of reward.

The use of a genetic algorithm to discover new classifiers by crossover and mutation in LCS also happens at intervals. The strength of a classifier is used as its fitness during the selection for reproduction. There are two popular approaches in evolving LCSs: the *Michigan approach* and the *Pitts approach* (De Jong 1988). For the Michigan approach (Booker 1982), each individual in a population is a classifier/rule. The whole population represents a complete LCS. For the Pitts approach (Smith

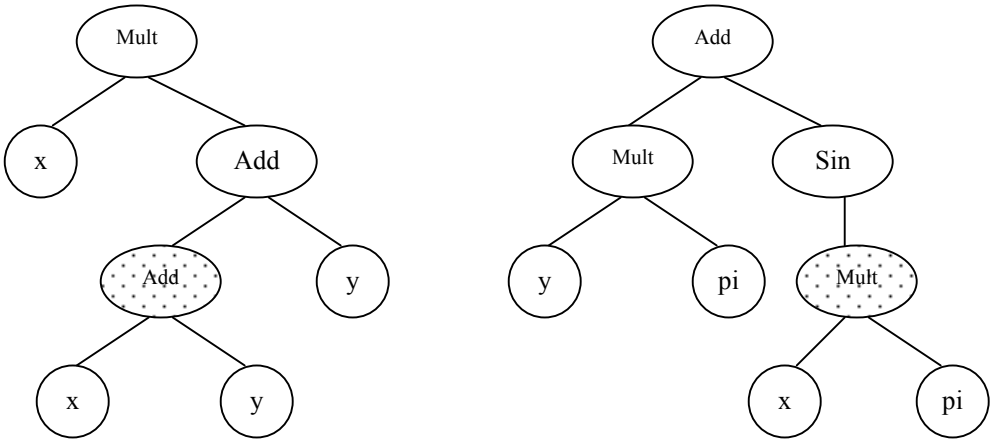
1980), each individual in a population is a complete LCS. The whole population is a number of competing LCSs. LCSs have been widely applied in a variety of domains including data mining for financial markets, real-world systems simulation and optimisation, control problems for industrial engineering. Please refer to Larry (2004) for a comprehensive coverage on recent advances in the applications of learning classifier systems.

2.7.4 Genetic programming

Genetic programming (GP) (Koza 1992; Banzhaf 1998; Koza *et al.* 1999; Koza *et al.* 2003) uses program trees as its representation for real-world problems. Computer programs can genuinely be described as functions, which produce outputs as their response to a set of given inputs. Genetic programming therefore can be described as function optimisation using genetic operations. Figure 2.15 shows an example of program tree representations used in GPs and the crossover operation on tree structures.



(a)



(b)

FIGURE 2.15 Genetic programming tree crossover. (a) shows two tree representations before crossover operation. Dotted nodes show the crossover point. (b) shows the new representations after the crossover.

Generally, program trees are constructed from a set of primitives, called *functions* and *terminals*. *Functions* specify operations within the program and are used as non-terminal nodes in the tree, e.g., a function set might be $\{+, -, *, \%, \text{sqrt}\}$. Note that *%* is protected division to stop division by zero errors. In Figure 2.15, *Mult*, *Add*, and *Sin* are functions. *Terminals* store values that functions act upon, e.g. *x*, *y*, and *pi* in Figure 2.15. In Figure 2.15(a), the tree structure on left hand side (LHS), represents a function $(\text{Mult x (Add (Mult x pi) y))}$ or $x*((x*pi)+y)$; the tree structure on right hand side (RHS) represents a function $(\text{Add (Mult y pi) (Sin (Add x y)))}$ or $(y*pi)+\sin(x+y)$. The two dotted nodes, *Mult* and *Add*, are crossover points. The two trees exchange their sub-trees below these two nodes. Figure 2.15(b) shows the two new offspring after the crossover operation. The function represented in Figure 2.15(a)-LHS is changed to $(\text{Mult x (Add (Add x y) y))}$ or $x*((x+y)+y)$. The function represented by Figure 2.15(b)-RHS is changed to $(\text{Add (Mult y pi) (Sin (Mult x pi)))}$ or $(y*pi)+\text{Sin}(x*pi)$.

Mutation generally happens in function nodes by randomly selecting a node from the tree and changing its operation. Selection methods, such as roulette wheel selection or tournament selection, are used to select GP trees for reproduction process. The fitnesses of trees are evaluated by actually executing programs with given training data sets as inputs. One of practical issues in GP is the size of trees that are produced (this is often known as *code bloat*). The size of program trees can grow very fast (both the width and the depth of the tree) during the evolutionary process. This will greatly degrade the efficiency of the algorithm, and may require considerable storage. At the same time, large size program trees also increase the difficulty in interpreting the solutions. Gustafson (2004) and Gustafson *et al.* (2004) provide in depth analysis on GP in terms of its dynamics, particularly on population size and population diversity, with the aim of encouraging efficient and effective search in genetic programming algorithms. More theoretical foundations of genetic programming can be found in Koza (1992), Angeline (1998), Soule and Foster (1998), Langdon *et al.* (1999), Koza *et al.* (1999) and Koza *et al.* (2003).

GP is a very useful tool in function approximation, and hence in the modelling and optimisation of real-world problems. Koza (1994) described a GP-based subsumption architecture for robot control. Poli (1996) described the use of genetic programming for feature detection and image segmentation in image analysis. In financial markets, genetic programming is used for option pricing (Keber 1999), modelling artificial stock markets (Chen and Yeh 2001), financial decision support (Tsang 2004). More

applications of genetic programming can be found in Spector *et al.* (1999), Koza *et al.* (1999) and Koza *et al.* (2003).

2.7.5 Ant colony optimisation

Ant colony optimisation (ACO) algorithms (Dorigo 1992; Dorigo *et al.* 1996; Dorigo and Di Caro 1999; Bonabeau *et al.* 1999; Dorigo and Stutzle 2004) are inspired by the collective intelligence of ant colonies. ACO can be described as a metaheuristic that draws upon strategies of ants when foraging for food, i.e. depositing pheromone trails such that shorter paths will eventually have more pheromone trails and more likely to be followed by subsequent ants. For implementing an ACO algorithm, a certain representation of the optimisation problem is given, usually in the form of a network/graph of nodes. Artificial ants/agents walk through the network by using a combination of a problem specific heuristic function and the amount of pheromone on a route for choosing a path, so that eventually a shorter path, i.e. a good (perhaps optimal) solution is discovered. Table 2.15 describes a typical implementation of ant colony optimisation algorithms, adapted from Dorigo and Stutzle (2004).

```
ACO algorithm ():
    Set parameters, initialise pheromone trails
    While (termination condition not met) do
        ConstructSolutions
        ApplyLocalSearch
        UpdatePheromoneTrails
    End
    Return the best solution
End
```

TABLE 2.15 Algorithmic skeleton for ant colony optimisation algorithms

Using the travelling salesman problem as an example, initially, the TSP problem is represented as a connected graph where each node represents a city from the problem.

Each arc (i, j) of the graph is associated with a variable τ_{ij} called *artificial pheromone trail*. Generally all arcs of the graph are initialised with the same, small, amount of pheromone. Before the search starts, each of the artificial ants is placed on a randomly chosen city. During each iteration, as shown in Table 2.15, every ant will construct a feasible solution, then the generated solutions are improved by applying a local search, and finally pheromone trails in the graph are updated before entering next loop. An ant constructs a solution using two criteria: the pheromone trail strength on the arc and locally available heuristic information that, in the case of travelling salesman problem (TSP), is usually a function of the arc length. Define N_i as the set of all the one-step neighbour nodes of node i in the graph. The probability of an ant k at node i choosing the city $j \in N_i$ to move to in the t-th iteration, $p_{ij}^k(t)$, is defined as equation 2.11 below:

$$p_{ij}^{k}(t) = \frac{a_{ij}(t)}{\sum_{l \in N_i^k} a_{il}(t)} \tag{2.11}$$

where

$$a_{ij}(t) = \frac{\left[\tau_{ij}(t)\right]^{\alpha}\left[\eta_{ij}\right]^{\beta}}{\sum_{l \in N_i}\left[\tau_{il}(t)\right]^{\alpha}\left[\eta_{il}\right]^{\beta}} \qquad \forall\ j \in N_i$$

α and β are the two parameters that control the relative weight of pheromone trail and local heuristic value in deciding which next node to move to. η_{ij} is the local heuristic value that is equal to the inverse of the distance between the two cities, in other words, the longer the distance the smaller the probability to be chosen.

After artificial ants have constructed possible solutions, local search methods are used to improve the quality of the solutions constructed. In general, local search looks at the neighbourhood of a solution T constructed and if a better solution T' is found, it replaces the current solution with the better solution and the local search is continued from T'. The most popular local search method used for TSP ants algorithms include 2-opt and 3-opt (Bentley 1992), which systematically test whether the current solution can be improved by replacing 2 or at most 3 arcs from the graph. Other local search methods can be found in Dorigo and Gambardella (1997), Stutzle and Hoos (1999), Hoos and Stutzle (2004).

Pheromone trails on all arcs in the graph are then updated as follows, before the algorithm goes into next loop. Define ant k's solution at iteration t as $T^k(t)$, the length of route $T^k(t)$ as $L^k(t)$, m is the total number of ants, $\Delta\tau_{ij}(t)$ for arc (i, j) is defined as follows:

$$\Delta \tau_{ij}(t) = \sum_{k=1}^{m} \Delta \tau_{ij}^{k}(t)$$

where

$$\Delta \tau_{ij}^{k}(t) = \begin{cases} \dfrac{1}{L^k(t)} & if\ (i, j) \in T^k \\ 0 & else \end{cases}$$

The pheromone on the arc (i, j) is then updated using equation 2.12.

$$\tau_{ij}(t) = (1 - \rho)\tau_{ij}(t-1) + \Delta\tau_{ij}^{k}(t) \qquad (2.12)$$

$\rho \in (0,1]$ is called the pheromone decay coefficient, which automatically decreases the pheromone intensity on the arcs over time so as to avoid the rapid convergence of the ant algorithm towards a sub-optimal region in the solution space. This process is also referred as *pheromone evaporation*.

Through iterations, the pheromone trails in the problem graph are updated following the procedures described above. As a result, the arcs contained in shorter tours and/or visited by many ants tend to receive a higher amount of pheromone and are therefore chosen with a higher probability in the following iterations. The typical ant colony optimisation algorithm described in Table 2.13 have since been developed into a number of variations to overcome limitations in the simple ACO algorithm and cope with additional constraints from different problem domains. Stutzle and Hoos (1998) introduced a MAX-MIN ant algorithm, which only uses one ant with the best solution to update pheromone trails, and limits the range of pheromone trails to the interval $[\tau_{\min}, \tau_{\max}]$. Di Caro and Dorigo (1998, 1998a) described the AntNet and its variation S-AntNet for routing problems in communications networks. Dorigo and Di Caro (1999) and Dorigo and Stutzle (2004) provide extensive surveys on various ants algorithms, which they termed as *ant colony optimisation meta-heuristic*, and their applications in various domains.

2.7.6 Particle swarm optimisation

Both ant colony optimisation and particle swarm optimisation originates from the study on *swarm intelligence* in nature. Swarm intelligence refers to the phenomena that some natural creatures behave as a swarm in which individuals of the swarm follow simple rules, leading to the swarm exhibiting complex, intelligent behaviour. Eberhart and Kennedy (1995) developed particle swarm optimisation (PSO) that is based on the analogy of birds flocking and fish schooling. The fundamental concept behind particle swarm optimisation algorithms is that individuals in a swarm exchange

previous experiences whilst the randomness of moving in the search space is maintained. To some extent, particle swarm optimisation algorithms are similar to other evolutionary algorithms, such as genetic algorithms, in that all these optimisation algorithms maintain a population of potential solutions. Genetic algorithms, which evolve potential solutions through selection and reproduction, differ to PSO where potential solutions, called *particles*, are flown through the problem hyperspace. The flying of particles in the problem space is controlled by *velocities*. At each iteration of the algorithm, each particle's velocity is stochastically accelerated towards its previous best position and towards a global best position. Table 2.16 describes a typical implementation of a PSO algorithm, adapted from Eberhart *et al.* (2001).

PSO algorithm ():
n is number of particles in the population. *d* is the dimension of the optimisation problem.

Initialise population of particles. Each particle has the form of (X_i, V_i, p^i_{best}), $1 \le i \le n$, where $Xi = (x_i^1, x_i^2, \cdots, x_i^d)$ is the vector of random initialised parameters, $Vi = (v_i^1, v_i^2, \cdots, v_i^d)$ is the vector of associated random initialised velocities on each dimension. Initially, for each particle, p^i_{best} is the particle itself.
EvaluateEachParticle.
While (termination condition not met) **do**
g_{best} := the particle with the highest fitness in the population.

For(i = 1, i++, $i \le$ n)
For(j = 1, j++, $j \le$ d)
Update particle i's velocity on the jth dimension v_i^j using p^i_{best} and g_{best}.
Update x_i^j using v_i^j
End for
EvaluateUpdatedParticle.
If (updated particle is better than the current p^i_{best})
p^i_{best} := the updated particle.
End if
End for
End while
Return g_{best}.
End

TABLE 2.16 Particle swarm optimisation.

The velocity of particle *i* on the *j*th dimension is updated using equation 2.13 below,

$$v_{i,j}^{k+1} = wv_{i,j}^{k} + c_1 rand_1(p_{best}^{i} - x_{i,j}^{k}) + c_2 rand_2(g_{best} - x_{i,j}^{k}) \qquad (2.13)$$

where $v_{i,j}^{k+1}$ is particle *i*'s velocity on the *j*th dimension at iteration *k*+1. $v_{i,j}^{k}$ is particle *i*'s velocity on the *j*th dimension at iteration *k*. *w* is a weighting function. C_1 and C_2 are weighting factors of values of 2.0 (Shi and Eberhart 1998). $s_{i,j}^{k}$ is particle *i*'s position on the *j*th dimension at iteration *k*. *pbest* is the historical individual best position of particle *i*. *gbest* is the historical global best position of the swarm. Weighting function, *w*, is calculated using equation 7.8 below:

$$w = w_{\max} - \frac{w_{\max} - w_{\min}}{iter_{\max}} \times iter \qquad (2.14)$$

where $w_{\max}$ is an initial weight of value 0.9, and $w_{\min}$ is the final weight of value 0.4 [Shi and Eberhart 1998]. $iter_{\max}$ is the maximum number of iterations. *iter* is the current iteration number. Finally, the new position of particle *i*, s_i^{k+1}, is calculated as equation 7.9 copied below:

$$s_i^{k+1} = s_i^{k} + v_i^{k+1}$$

One of the major features of PSO algorithms is their simplicity in implementation, i.e., keeping a population of possible solutions, and repeatedly tuning them towards the global best solution in the population and the particles' historical best position. There are no genetic operations in PSO, such as crossover and mutation, which could require considerable computational time, and hence the high computational efficiency of PSO algorithms. PSO was initially designed for continuous optimisation problems, e.g.,

evolving connection weights for artificial neural networks (Eberhart and Shi 1998) which achieved remarkable performance in terms of computational efficiency. Yoshida *et al.* (2001) described a modified version of the continuous PSO algorithm, which was able to handle both discrete and continuous variables, and applied the algorithm in solving reactive power and voltage control problems. Other recent applications of PSO can be found in surveillance mission control in army (Secrest 2001), and biochemistry study (Cockshott and Hartman 2001).

2.7.7 Cultural Algorithms

Cultural algorithms (Reynolds 1979, 1994, 2002) are evolutionary computation models that emulate cultural evolutionary processes. Figure 2.16 depicts the major components of a cultural algorithm (Reynolds 2002).

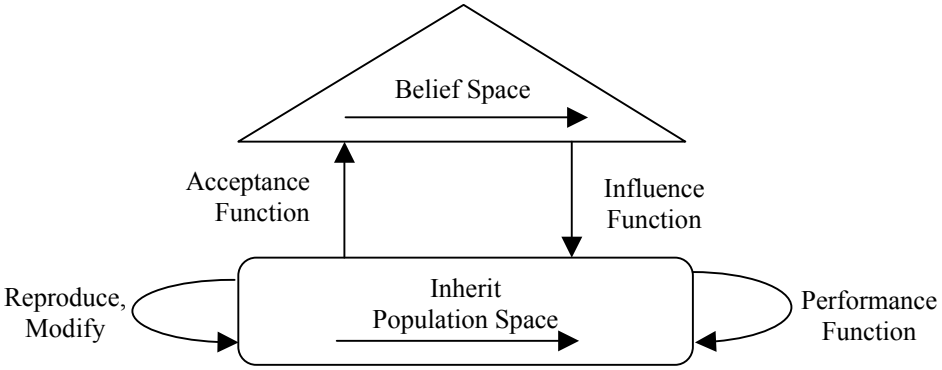


FIGURE 2.16 The framework of cultural algorithms**.**

As shown in Figure 2.16, a cultural algorithm consists of two levels of evolution: a microevolution in the *population space* and a macroevolution in the *belief space*. Through an acceptance function, the experiences of individuals in the population space are used to generate problem solving knowledge to be stored in the belief space. The belief space manipulates the knowledge which in turn guides the evolution of the

population space by means of an influence function. The basic pseudo code of cultural algorithms is shown in Table 2.17 below.

```
Cultural Algorithm ():
Begin
    t = 0;
    Initialise population POP(t);
    Initialise belief space BLF(t);
    Repeat
        Evaluate population POP(t);
        Adjust(BLF(t), Accept(POP(t));
        Adjust(BLF(t));
        Variation(POP(t) from POP(t-1));
    Until termination condition achieved
End
```

TABLE 2.17 Cultural Algorithms.

A variety of evolutionary paradigms can be used to model the population component such as genetic algorithms for optimisation problems (Reynolds and Sverdlik 1992), genetic programming for evolving agent strategies (Ostrowski *et al.* 2002; Reynolds and Ostrowski 2004), evolutionary programming for real valued function optimisation (Reynolds and Chung 1996; Reynolds and Zhu 2001), mutli-agent based models (Reynolds and Chung 1997; Reynolds *et al.* 2004; Lazar and Reynolds 2005) and particle swarms (Reynolds and Peng 2005). Knowledge models that have been employed for the implementation of the belief space include real-valued interval schemata (Reynolds and Chung 1998), regional schemata (Jin and Reynolds 2002), graphical models such as genetic program trees (Zannoni and Reynolds 1997), and semantic networks (Reynolds and Rychtyckyj 2002).

In general, five basic categories of knowledge can be stored in the belief space: Normative Knowledge, Situational Knowledge, Domain Knowledge, History Knowledge, and Topographical Knowledge (Reynolds and Saleem 2004). Normative

knowledge is the set of acceptable ranges for the parameters to be optimised. Normative knowledge in fact indicates the area of the solution space that is most likely to contain the optimum solution. Situation knowledge is abstracted from the successful agents evolved in the population space, who have a set of values in common. Domain knowledge uses knowledge about the problem domain to guide the search. For example, if a problem landscape is composed of cones, knowledge about cone shape and the related parameters will be useful. History knowledge is most useful for backtracking. For example, keeping a record of the global best performer from the past ten generations. Topographical knowledge is the knowledge that is useful in certain locations of the problem landscape. As an example, if we divide a problem landscape into grids, the best performer from each cell can be saved into the belief space as a topographical knowledge. All this knowledge will guide the evolution of the population space through the influence function.

Cultural algorithms have also been applied to function maximisation problems in dynamic environments (Reynolds and Saleem 2001, Reynolds and Peng 2004, Reynolds and Saleem 2004). These experiments all used the Morrison & De Jong's Dynamic Problem Generator "DF1" (Morrison 1999). For example, in Reynolds and Peng (2004), the problem landscape is generated as a two-dimensional plane with four cones randomly scattered on it. The four cones are of different heights (H_j) and slopes (R_j). The function to be maximised is give by equation 2.16,

$$f(x_1, x_2) = \max_{j=1,..,n} (H_j - R_j * \sqrt{\sum_{i=1}^{k} (x_i - C_{j,i})^2}) \qquad (2.16)$$

n is the total number of cones on the plane. k is the number of parameters to be optimised, in this case, $k = 2$, i.e. x_1 and x_2. $C_{j,i}$ is the position of a cone on the plane. The dynamic environment is generated by shifting the locations of cones counter-clockwise every 200 generations, i.e. the first cone moves to the fourth cone's position and the second cone moves to the first cone's position and so on. Because each cone has a different height and slope, when the cones move, the position of the optimal solution $(x_1^{'}, x_2^{'})$ also changes. Through experiment, cultural algorithms are able to find the new optimal point $(x_1^{'}, x_2^{'})$ when its environment changes. Reynolds and Saleem (2001) show that the cultural algorithm is more effective than a stand-alone evolutionary algorithm in finding the new optima under new environmental dynamics. Reynolds and Peng (2004) discuss how the learning of knowledge in the belief space ensures the adaptability of cultural algorithms.

Note that the dynamic environmental change in the experiments described above only refers to the change of values of environmental parameters, such as the change of values of $(H_j, R_{j,} C_{j,i})$. There is no new environmental parameters emerging or being introduced over time. In other words, the dimension of the problem space does not change. Other applications of cultural algorithms in dynamic environments, with multi-agent based population space, are carried out in the similar way. Reynolds and Ostrowski (2004) combine the cultural algorithm and the agent-based computational economics for the evolution of successful pricing strategies in an Original Equipment Manufacturer-Consumer market. The tested market model is a dynamic environment

in the sense that a *theta* parameter, which indicates an economic recession, is applied for various lengths of time.

The *new challenge* we discussed in section 2.6 (See p. 81) not only indicates that the status, e.g. values, of current existing environment parameters/variables could change, but also the possible emergence of new environment variables that brings new and different kinds of problems, e.g. learning to play checkers and learning to play poker at the same time, to an intelligent entity. In other words, the problem dimensions and domains could also change. In chapter 4, we will describe an integrated individual and social learning paradigm that is able to answer the new challenges from a dynamic environment.

2.8 Discussion

At the beginning of this chapter, we introduced machine learning as an essential part of artificial intelligence. Learning is the nature of intelligence. Individuals are intelligent because they can learn from experience and adapt to changing environments. On the other hand, individuals can learn because they are intelligent. Without learning capabilities, machine intelligence is very much limited since people have to hand-encode every single detail to allow a machine to complete a given task, while we ourselves do not completely understand intelligence and thinking. We described a number of popular machine learning paradigms in section 2.3, 2.4 and 2.7. This included the tree structures used in decision tree learning and database of past-

classified instances used by instance-based learning, Bayesian learning bring fuzziness into machine learning by means of incorporating imprecision in representing real-world problems as Bayesian networks, so that machines are able to learn to reason under uncertainties. Local search methods such as simulated annealing and tabu search enable machines to solve a problem by locally exploring neighbourhoods of potential solutions. Artificial neural networks, on the other hand, enables machines to learn through trial-and-error and represent knowledge in a distributed form, i.e. knowledge is stored in connections and neurons in a neural network. Compared to other learning paradigms, artificial neural networks are more noise tolerant and more closely emulate the existing form of human intelligence. All the learning paradigms mentioned above are examples where machines learn from experience. In reality, learning from experience is not always easy. Most of the time, a task can only be fulfilled through a sequence of actions. The reward for a single step is not always immediately observable, and we may have to wait until all the actions have been carried out. However, an intelligent agent has to make a decision what action to take at each time step. The concept of reinforcement learning has been designed to solve these types of problems.

Evolutionary learning, such as genetic algorithms or swarm intelligence (i.e. ants systems or particle swarm optimisation), simulates the evolutionary processes of nature. Rather than search from general-to-specific, e.g., instance-based learning, or search from simple-to-complex, e.g. decision tree learning, the biggest advantage of evolutionary learning is it searches in a global hypothesis space. Not all past instances

in instance-based learning can always be used to generate the solution for new problems. What if there are new factors that need to be considered for a new environment that never appeared in the previous history? Also, during the learning phase, when constructing a decision tree or a Bayesian network, it is possible that some promising architectures are missed by the learning algorithms. Evolutionary algorithms provide a global optimisation approach and can generate good solutions in acceptable computational time, as we have seen their applications in learning artificial neural networks in section 2.4.3.

Evolutionary learning also serves as a unifying principle of intelligence. The top-down approach for developing artificial intelligence always attempts to develop a model that can explain intelligence or a particular part of intelligence. Evolutionary computation presents another way, a bottom-up approach, in simulating intelligence. Fogel (2000) pointed out that whether it is an individual, a species, or a social group, every intelligent system in nature adopts a functionally equivalent process of reproduction, variation, competition, and selection. Each such system possesses a unit of mutability for generating new behaviours and a reservoir for storing knowledge, and evolves adapting its behaviour to achieve its goals in a range of environments. Evolutionary computation not only can recapitulate human behaviours, but also can generate new solutions and new behaviours that are not currently known to humans. This has great implications in the light of the fact that we are far away from completely understanding brain and intelligence. In *Bondie24*, the combination between evolutionary artificial neural networks and evolutionary programming enables a

computer checkers program to learn to play checkers at a human expert level, without any pre-injection of human knowledge. *Blondie24* is an important advance in computer game playing and artificial intelligence research. However, we concluded that *Blondie24* is still not an adaptive intelligence because it fails to respond to new challenges from the environment. We argued that the process for creating *Blondie24* is an adaptive learning process, but *Blondie24* itself, as an end product, is not adaptive.

We believe that the problem is with the way people regard artificial intelligence as a perfect end product rather than as an evolutionary process. Using human learning of speaking a language as an example, humans learn to speak certain languages from their parents and through their education at schools. However, a human does not stop learning to speak after he/she left school. When new terms, such as "internet" and "mp3" that did not exist a few decades ago, appear within a society, a person will pick them up from media or friends. When a person goes aboard to work, she/he will probably learn to speak another language (or at least a few phrases) from the local people. People even change their accents when they have friends speak with a different dialect or accent. This learning process of speaking entirely depends on its environment, which is essentially an incomplete or imperfect environment in the sense that it is not consistent at all time and always keeps on evolving. An incomplete environment is the fundamental reason behind the life-long continuous adaptive learning process in human intelligence. We argue that intelligence should not be considered as a perfect end product, but should be considered as an imperfect evolutionary system that constantly adapts itself and responds to the new challenges

from an incomplete environment. In this thesis, we propose ***imperfect evolutionary systems*** as a new learning paradigm for evolutionary computation and artificial intelligence research.

Among the various evolutionary algorithms and paradigms described in section 2.7, parallel genetic algorithms and memetic algorithms increase the efficiency of genetic algorithms by means of parallel implementation and incorporating local searches respectively. Learning classifier systems and genetic programming make use of more powerful and more complex symbolic representation for representing learning problems. Compared to artificial neural networks, learning classifier systems and genetic programming have the advantage that their solutions are more easily interpreted by humans. However, in learning classifier systems, the production rules need to be designed using some human expertise, which makes LCS human-dependent as expert systems. Ant colony optimisation and particle swarm optimisation mimics swarm behaviours in nature and generate intelligent solutions through the emergent behaviour of interactions among many individuals following simple rules. Cultural algorithms are more effective in adapting to dynamic changing environments due to its dual evolution at both the population level and the social knowledge level. We have seen many applications that incorporate one of the stand-alone evolutionary algorithms, such as GA, EP, GP and particle swarm, into a cultural algorithm in order to achieve better performance. However, cultural algorithms still have not answer the question of how to adapt to new challenges from a dynamic environment. In summary, none of the current existing evolutionary learning techniques or paradigms

demonstrates any properties in modelling an intelligent system as an imperfect evolutionary system, but rather pursue an optimal solution for a particular problem. In the next chapter, we will give a general introduction to stock markets under two motivations: (1) The idea of imperfect evolutionary system in fact originates from the study of stock market as imperfect information systems. (2) Stock markets provide an excellent test bed for experimenting with the idea of imperfect evolutionary systems.

Chapter 3

The Stock Market

This chapter gives an overview of the stock market with emphasis on two major areas: the stock market as an evolutionary system and the stock market as an imperfect information system. A general introduction to stock markets is given in sections 3.1 to 3.3 including discussions on the well known efficient market hypothesis and various fundamental analysis and technical analysis methods for financial investments in stock markets. Sections 3.4 and 3.5 discuss the employment of evolutionary computation techniques in the study of stock markets from two aspects: evolutionary computation for financial engineering and evolutionary computation for artificial stock market modelling. In EC for financial engineering, we focus on artificial neural networks for stock trading. In EC for market modelling, we focus on the employment of agent-based bottom-up approaches in studying stock markets as complex evolutionary systems. Section 3.6 describes the change from a perfect paradigm to an imperfect paradigm in economics study and its implications in financial markets. Section 3.7 provides a summary of the chapter.

3.1 Overview

The *stock market* is a general term for the organised trading of stocks through stock exchanges and over-the-counter transactions (Gough 2001). In stock markets, *stock* is

an instrument that signifies an ownership position of an investor in a corporation, and represents a claim on its proportional share in the corporation's assets and profits. *Share* refers to the certificate representing one unit of ownership in a corporation, mutual fund, or limited partnership. Ownership in the company is determined by the number of shares a person owns divided by the total number of shares outstanding. Most stock also provides voting rights, which give shareholders a proportional vote in certain corporate decisions. As one of the many types of financial markets, stock markets have become the most popular and important investment tool for both individual investors and institutional investors. In 2003, in the world's largest stock market, the New York Stock Exchange (NYSE), the average daily trading volumes was US$1,398 millions[1], there were 2,750 companies listed in NYSE. In the London Stock Exchange (LSE), in 2003, there were 1,557 companies listed (UK only, not including Irish), and the market capitalisation of UK and international companies amounted to GBP £3.3 trillion[2]. Figure 3.1 below shows the major components of a stock market.

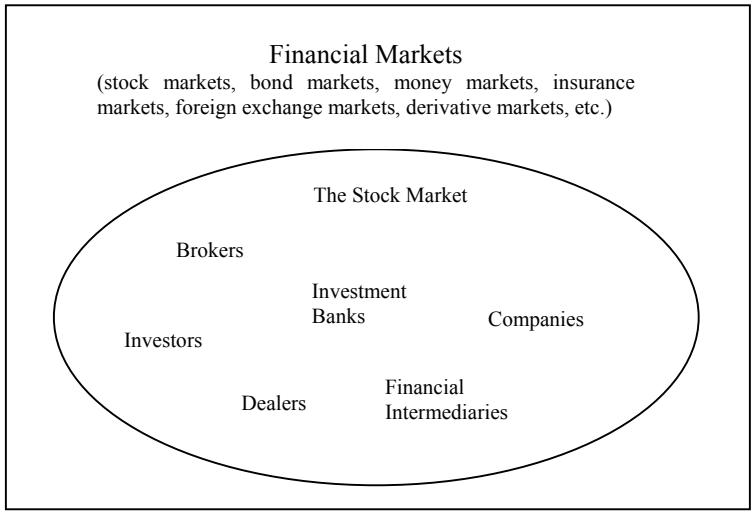


FIGURE 3.1 Major participants in a stock market.

[1] Data acquired from http://www.nyse.com.
[2] Data acquired from http://www.londonstockexchange.com.

Investors in stock markets are commonly divided into two categories: *individual investors* and *institutional investors*. In the early years, the major participants in stock markets were mainly large institutional investors, such as investment companies, mutual funds, insurance companies and pension funds etc., which usually possess a large amount of capital for investments. However, in the last few decades, more and more individual investors have invested in stock markets due to the radical drop in transaction costs and the advances in IT techniques, particularly the Internet, which make it much easier for an individual investor to access the markets nowadays. The privatisation initiative of the UK government in the 1980's also gave the opportunity for many individuals to own shares for the first time. This has had a profound effect on stock markets. Today's stock markets demonstrate more volatility than they did decades ago. A number of economic theories and models were developed attempting to understand the phenomena that emerged, such as market crashes and bubbles, and what regulations could be introduced to generate a more efficient and effective market structure (Bodie *et al.* 2002).

The *random walk theory* (Malkiel 1973) claims that market prices follow a random path up and down, without being influenced by past price movements, making it impossible to predict with any accuracy which direction the market will move at any point. In other words, the theory claims that the path a stock price follows is a random walk that cannot be determined from historical price information, especially in the short term. In Malkiel (1973), the author preaches that both technical analysis and fundamental analysis for stock selection and trading are largely a waste of time, and a

long-term buy-and-hold strategy is the best investment strategy. Malkiel backs up his claims with statistics showing that most mutual funds fail to beat benchmark averages such as the S&P 500 (Malkiel 1995).

The *Buy and Hold strategy* (Bodie *et al.* 2002) suggests that financial investors buy stocks and then hold them for a long period, usually over 10 years, regardless of the market's fluctuations. The assumptions behind the buy and hold strategy is that in the long term, stock prices will go up, but the average investor does not know what will happen tomorrow. In other words, any attempts to predict tomorrow's stock price are useless. The logic behind this is that in a capitalist society the economy will keep expanding, so profit will keep growing and both stock prices and stock dividends will increase as a result. There may be short-term fluctuations, due to business cycles, or rising inflation, but in the long term these will be smoothed out and the stock market as a whole will rise. Historical data from the stock market of the past 50 years supports this claim. Also, with a buy and hold strategy investors can avoid high trading commissions and taxes can be reduced and deferred.

The *Efficient Market Hypothesis (EMH)* (Fama 1965, 1970) is the most debated theory in the history of stock market study. What the EMH says is that, in an efficient market, at each time *t*, the current prices of financial assets reflect all available information relevant for judging the future returns of those assets. In finance, the term "efficient" is used specifically in the sense of informational efficient. An efficient stock market is a market where all available information is optimally used in the determination of

stock prices at each point in time. From the point of view of a stock market investor, the EMH implies (Malkiel 2003):

- An efficient stock market is based on the assumption that all news is promptly incorporated in stock prices. If all information is immediately reflected in stock prices, then tomorrow's prices change will reflect only tomorrow's news and will be independent of the price changes today. Since news is unpredictable, uncertain by its definition, the stock prices are unpredictable.
- From another perspective, the investors in the stock market buy and sell stocks under the assumption that the stocks they are buying are worth more than the price that they are paying, and the stocks they are selling are worth less than the selling price. But in an efficient stock market, at any point in time, the actual price of a stock will be a good estimate of its intrinsic value. Thus, the above speculations in attempts to outperform the market are a game of chance rather than a matter of good investment strategies.

However, it is not correct to simply conclude from the EMH that stock prices are unpredictable. The key point is that the EMH assumes the market is efficient, or assumes the stock prices reflect all available information. In fact, the EMH has three different forms based on different information sets. Fama (1970) describes a hierarchy of nested categories of information sets. As one moves down the hierarchy from the largest to the smallest set, efficiency is required with respect to ever decreasing amounts of information, and hence in an ever weaker sense.

- *Weak-Form Efficiency* asserts that the traders' information sets only contain the current and past price histories of the assets in a market. In other words, all past market prices and data are fully reflected in securities prices. This means technical analysis is of no use in a stock market with weak-form efficiency. From another point of view, in a weak-form efficient market, fundamental analysis may be of some use since publically available information about the company is not incorporated in its stock price. Traders can use this publically available information to predict the stock's future movement.
- *Semistrong-Form Efficiency* asserts that the traders' information set contains all publicly available information, i.e., the information set is not assumed to contain privately held information that has not been made public. In other words, all publicly available information is fully reflected in securities prices. This means fundamental analysis is of no use in a market with semistrong-form efficiency. From another point of view, in a market with semistrong-form efficiency, trading with insider information makes sense because the insider information is not reflected by securities prices.
- *Strong-Form Efficiency* asserts that the traders' information sets contain all available information which could possibly be relevant for the pricing of assets in a market. In other words, not only is all publicly available information embodied in securities prices, but all privately held information as well. That means insider information is of no use when attempting to predict future stock prices since the current securities prices already reflect all information available, including the insider information.

The debate on EMH has resulted in numerous empirical studies that attempted to determine whether stock markets are in fact efficient, and if so to what degree. A number of researchers have disproved the efficient market hypothesis by uncovering various stock market anomalies, such as the small-firm effect (Reinganum 1983), the January effect (Ritter 1988), market overreaction (Bondt and Thaler 1987), excessive volatility (Schwert 2001), etc. These market anomalies represent systems or patterns that can be used by investors to outperform the buy-and-hold strategy, hence, indicating that the EMH may not always be entirely true under all market conditions. Also, the stock market crashes in 1987 and 2000, particularly the abnormal high stock prices for internet-based companies in 1990's, have also convinced many economists that the stronger version of EMH, i.e. asset prices reflect all information including the insider information (in other words, the stock prices reflect their intrinsic value) is not true. Grossman and Stiglitz (1980) points out that the stock market cannot be perfectly efficient or there would be no incentive for professionals to uncover the information which gets so quickly reflected in market prices. In reality, financial markets are neither perfectly efficient nor completely inefficient. All financial markets are efficient to a certain extent, some more (such as bond markets, large capitalisation stocks), some less (such as small capitalisation and international stocks). In markets with substantial impairments of efficiency and information which is not fully incorporated in the stock prices, more knowledgeable investors can strive to outperform those with less information.

Grossman and Stiglitz (1980) demonstrate another research field in the study of stock markets, i.e. *asymmetric information theory*. Asymmetric information refers to the phenomenon that agents on one side of the market have much better information than those on the other side. In other words, different market participants engage in trades with each other on the basis of different information sets (Akerlof 1970; Spence 1974; Rothschild and Stiglitz 1976). We will look at asymmetric information theory in more detail in the study of stock markets using an imperfect information paradigm in section 3.6. In the following section, we will look at how more knowledgeable investors can outperform investors with less information by means of fundamental analysis and technical analysis.

3.2 Fundamental Analysis

Investors who employ fundamental analysis (Thomsett 1998; Gough 2001; Cahill 2003) to value individual stocks take into consideration only those variables that are directly related to the company itself, i.e., the company's finance and operation, especially sales, earnings, growth potential, assets, debt, management, products, and competition. In fact, professional fundamental analysts will usually examine the prospects for the industry as a whole, and then the records, plans, management of individual companies within the industry as a complete study. The overall economical and political environment is also considered. An estimate of a particular company's future earnings is made, incorporating estimates of future sales, overheads, accounting policies and a host of other factors that might affect profit. This estimated future

earning is usually used to calculate a company's Price/Earnings ratios (P/E ratio), which is used as a key analytical ratio for the valuation of a stock and provides an indication as to whether the stock is a worthwhile investment.

The P/E ratio itself cannot tell if a stock is overvalued or undervalued. For example, a small company in a high technology industry may have a rather high P/E ratio because the company's earnings are expected to dramatically increase in a few years' time. This high P/E price does not necessarily mean the stock is overvalued. What the fundamental analyst need do is to compare the current market P/E ratio with their own estimate of what it should be based on their own detailed examinations of the company, then invest in the company if the company's stock is undervalued, or avoid or sell the company's stock if the stock is overvalued.

There are also other important financial ratios used in fundamental analysis. *Price/Book* ratio is the ratio of the market value of a company to the book value of a company in its balance sheet. Book value is what would be left over for shareholders if the company shut down and paid off all its debts. Price/Book ratio works best with companies that has a lot of tangible assets and measures what the market is paying for those net assets. The lower the price/book ratio, the better. *Dividend yield* is the dividend paid by the company divided by its stock price. It tells an investor what percentage of the purchase price the company will return to the investor in dividends. Generally, investors prefer stocks with a higher dividend yield. However, a high dividend yield does not necessarily mean better returns. A serious drop in the stock

price could also result in abnormally high dividend yields. All these financial ratios indicate the company's potential profitability in the future and how the market is currently pricing the company, i.e. undervalued or overvalued, hence, may indicate the future price trends in the company's stock. One difficulty with fundamental analysis is that the analyst must make estimates based on experience. There are always unpredictable random events happening in the real world, such as natural disasters and political events, which can skew even the soundest estimates of a company's future earnings.

In general, investors who favour fundamental analysis can be categorised into two types*: value investing* and *growth investing* (Gough 2001; Bodie *et al.* 2002). Value investing is an investment strategy is where investors try to find companies that represent good value, e.g. their price/earnings ratio is low compared to similar companies. The theory is that these companies are currently out of favour and so have low prices, but one day they will come back into favour and the price will go up. Also, as the company is already good value it is less likely to drop as far as other higher rated companies if the stock market took a down turn. Growth investors look for companies that are growing and will continue to grow quickly. These investors are less concerned about companies that have high P/E ratios as they expect that as the company rapidly grows its earnings will also rise quickly, which will eventually reduce the price/earnings ratio in the future. Usually growth investors live in the hope that the earnings growth of the company will justify the price they paid, but there is a

danger that prices could fall a long way if the company does not meet the market's expectations.

3.3 Technical Analysis

While fundamental analysis cares about the real nuts and bolts of a company, technical analysis (Bauer and Dahlquist 1998; Reuters 1999; Oz 1999; Achelis 2000) pays more attention on the price behaviour of the company's stock in the market. Technical analysis uses a stock's past activity in the market to predict its future trend and price movement. Technical analysts think less about what is actually happening in a company to cause such changes. Two important tools used by technical analysts are *charts* and *technical indicators*. Figure 3.2 below shows a very popular and reliable head and shoulders chart that is used for analysing price patterns in stock markets, adapted from Oz (1999).

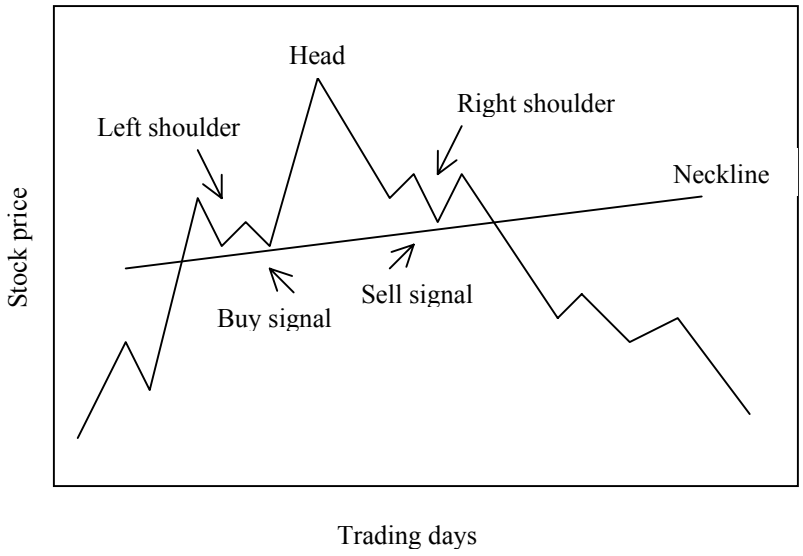


FIGURE 3.2 Head and Shoulders price pattern in technical analysis

The head and shoulder pattern usually has three peaks. The centre peak (Head) is higher than the outer two peaks (Left shoulder and Right shoulder), which are almost the same height. During a long up trend (left hand side of the Head), the investors are buying the stock, which push the price all the way up to the top of the left shoulder. The price then falls down to the neckline. If the price bounces back from the neckline, this usually indicates it is going higher. After the price reaches the top Head, some investors will start to sell for a profit, this will bring the price down to the neckline again. There are usually some oscillations around the left shoulder and right shoulder. However, when the price penetrates the neckline on the right shoulder, this strongly indicates the start of a downtrend, in other words, a strong sell signal. But this pattern also needs to be backed up by information about trading volumes (Reuters 1999). Usually the heaviest volume occurs in the formation of the left shoulder because more investors see this stock has a good prospect due to the long up trend. Then the trading volume decreases in the formation of the head, and decreases more in the right shoulder.

The underlying foundation of technical analysis is that there are many patterns of market behaviours that have long been recognised as significant, and for many of these patterns there is a high probability that they will produce the expected results (Reuters 1999; Achelis 2000). Other popular price patterns in stock markets include cup and handle, double bottom, declining and rising channels etc. History repeats itself; price patterns also repeatedly appear in the market. Besides chart patterns, technical analysts also use various technical indicators together with charts to predict

the future market movement. Here we describe some of the commonly used technical indicators by technical analysts, together with their usage. The definitions of the technical indicators described below are taken from Reuters (1999), unless otherwise specified.

- **Trading Volume**

Trading volume, or volume, is the total number of shares traded on a stock in a given period of time. As we have seen in the previous Head and Shoulders chart pattern example, trading volume is usually used to confirm the market trend, in other words, the volume is used to explain certain patterns and changes in supply and demand in the market. Thus, trading volume is generally used with other price-based indicators. Beside the daily trading volumes, other averaged daily volumes are also used such as *20-day average volume* ($\overline{V}_{20}$), which is calculated using equation 3.1 below:

(3.1)

V_i is the *i*th day's trading volume.

- **Moving Average (MA)**

Moving average is perhaps the most commonly used variable in technical analysis. It is the average price of a stock over a specified period. Depending on the different time frames used, there are *10-day MA (MA_{10}, 2-week MA)*, *20-day MA (MA_{20}, 4-week MA)*, *50-day MA (MA_{50}, 10-week MA*) and *200-day MA (MA_{200}, 40-week MA)*. MA_{10} and MA_{20} are short-term MAs while MA_{50} and MA_{200} are long-term MAs. Equation 3.2 below shows the calculation methods for these four popular MAs.

$$MA_n = \frac{\sum_{i=1}^{n} P_i}{n} \tag{3.2}$$

n takes the value of 10 (MA_{10}), 20 (MA_{20}), 50 (MA_{50}) and 200 (MA_{200}). P_i is the closing price of the stock on the ith day. MAs are used in order to spot pricing trends by flattening out large fluctuations in the daily stock prices. In general, the shorter the time frame used, the more volatile the prices will appear, so, for example, 20-day moving average lines tend to move up and down more than 200 day moving average lines. Figure 3.3 below demonstrates an IBM 2-Year moving average chart (closing price) with the MA_{10}, MA_{20}, MA_{50} and MA_{200} lines.

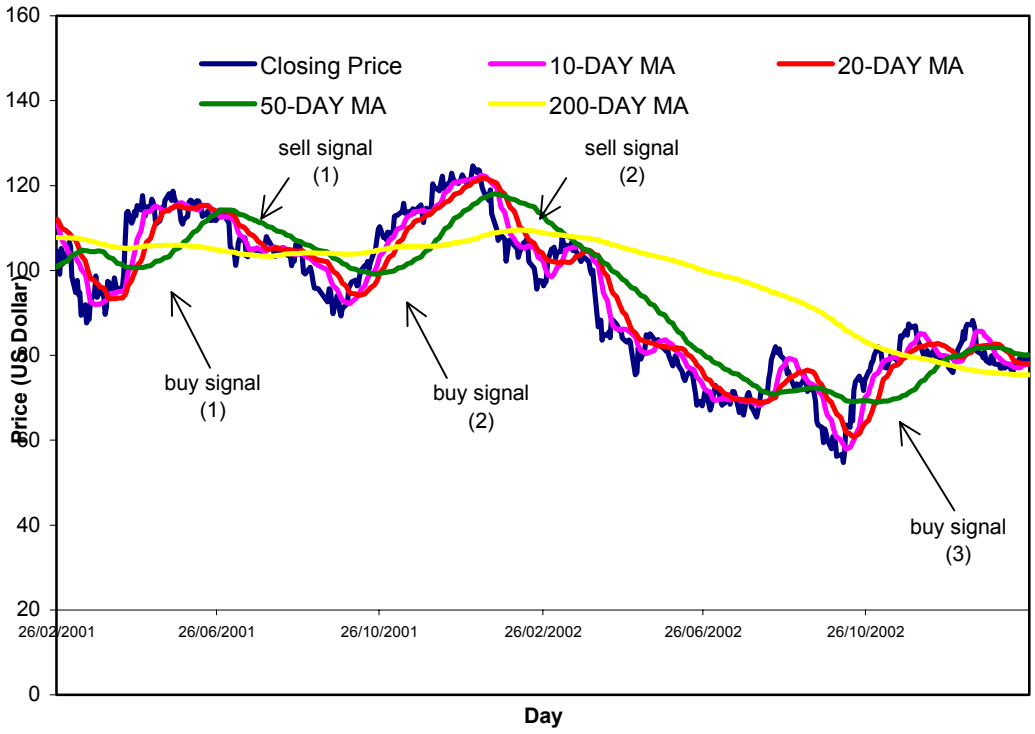


FIGURE 3.3 IBM 2-Year moving average Chart

As shown in Figure 3.3, MA_{50} line (green line) and MA_{200} line (yellow line) are much smoother than MA_{10} line (pink line) and MA_{20} line (red line), hence demonstrating the

market trend in longer time frame more clearly. In general, when a shorter term MA line (10-day MA and 20-day MA) crosses over a longer-term MA line (50-day MA) to the upside, and both slopes go up, it is a bullish signal (see buy signals in fig. 3). When a longer term MA line crosses over a shorter term MA line to the upside, and both slopes go down, it is a bearish signal (see sell signals in Figure 3.3). As we discussed previously, stock prices alone cannot be used to analyse the market trend. It is also important to use other indicators, such as volumes, to confirm the analysis.

- **Relative Strength Index (RSI)**

The RSI indicator is used to determine if a stock is overbought or oversold. The equation for calculating RSI is:

$$RSI = 100 - \frac{100}{1 + RS} \tag{3.3}$$

where RS is:

$$RS = \frac{Average\ of\ the\ closing\ prices\ of\ the\ up\ days\ during\ period\ N}{Average\ of\ the\ closing\ prices\ of\ the\ down\ days\ during\ period\ N}$$

In fact, RSI measures the magnitude of gains over a given time period against the magnitude of losses over that period. Depending on the different time frames for period *N*, we usually find *10-day RSI*, *14-day RSI*, and *21-day RSI* where *N* equals to 10 trading days, 14 trading days and 21 trading days respectively. Similarly, the shorter the time frame is, the more volatile the RSI indicator is. The value of a RSI indicator ranges from 1 to 100. In general, a RSI with a value of 30 or below indicates an oversold condition and a value of 70 or above indicates an overbought condition. In other words, stock traders usually will not buy stocks which have RSI greater than 70, and will not short sell stocks with a RSI below 30. Some analysts tend to use 80/20

rather than 70/30. However, it has to be stressed that RSI is used to determine if a stock is overbought or oversold. A RSI with a value over 70/80 does not mean the investor has to sell the stock. A RSI with a value less than 30/20 does not mean the stock is necessarily worth investing in, it probably because the company has gone into liquidation (Note that a RSI with a value of 30 or below generally indicates an oversold condition). Therefore, investors have to combine the RSI indicator with other indicators in order to make decisions (Oz 1999).

- **Relative Performance (RS)**[3]

Sometimes it is not enough to measure a stock's performance just by looking at the price changes alone, it is also necessary to measure the stock's relative performance compared with the whole market. *Relative performance*, or *relative strength*, is used by technical analysts to compare the price of a stock to the entire market. The relative strength of a stock is calculated by taking the percentage price change of a stock over a set period of time and ranking it on a scale of 1 to 100 against all other stocks on the market. This is shown in equation 3.4 below.

$$\text{RelativePerformance} = \frac{\frac{A2}{B2} - \frac{A1}{B1}}{\frac{A1}{B1}} \tag{3.4}$$

A1 is the last day closing price of the share. *B1* is the last day closing price of the index. *A2* is the first day closing price of the share. *B2* is the first day closing price of the index. Some technical analysts prefer stocks with high relative strength rankings, believing that stocks that have recently gone up are more likely to continue to go up. Other technical analysts believe that a very high relative strength can be an indication

[3] Definition taken from http://www.investorwords.com.

that the stock is overbought and is ready to fall. Thus, relative strength measures only how the stock has done in the past, not how it will do in the future.

- **Rate of Change (ROC)**

Rate of change indicator is used directly to demonstrate the changes in a stock price, trading volume or a market index within a given time frame. The equation used to calculate ROC is shown in equation 3.5 below:

$$ROC = \frac{P_i - P_{i-n}}{i} \tag{3.5}$$

P_i is the stock price on the ith day. n is the total number of days in the time frame that can be as short as one day. P_{i-n} is the stock price on the first day of the time frame. The shorter the time frame is, the more volatile the ROC is.

- **Chaikin Oscillator**

The major merit of the chaikin oscillator is that this indicator incorporates the trading volume in its calculation. As mentioned previously, volume analysis usually helps in identifying the underlying strength for the current price movement. A healthy advance in the price is usually accompanied by rising volumes and a strong volume accumulation, whereas, a lagging volume is a sign of less motivation to move the stock higher. Equations used for calculating the *chaikin oscillator (Osc)* are shown below.

$$Osc = \overline{AD_3} - \overline{AC_{10}} \tag{3.6}$$

AD refers to Accumulation Distribution, calculated as following.

$$AD = (Close - Open)/(High - Low) * Volume$$

Close refers to closing price. *Open* refers to opening price. *High* is intra-day high and *Low* is intra-day low. *Volume* is the trading volume. The most important signal generated by the chaikin oscillator occurs when a price reaches a new high or a new low, particularly at an overbought or oversold level, and the oscillator itself fails to exceed its previous extreme level and then reverses direction. For example, we see that a stock is currently in its up trend (e.g. the stock's price is above its 50-day MA), then an upturn of the oscillator in negative territory generates a buy signal, and a downturn of the oscillator in positive territory would be a sell signal.

- **Bias**

Bias is also an oscillatory indicator that shows the difference between a stock's price and its moving average. The equation is:

$$Bias = \frac{P - MA_N}{MA_N} \tag{3.7}$$

P_i is the ith day's stock price. MA_N is the N-day moving average of the stock price. N can be as short as 3 or 10 days, or other longer time frames. A positive bias indicates a strong bullish market. Bias needs to be used with other indicators to confirm the market trend.

- **Stochastic Oscillator (%K & %D)**

The Stochastic Oscillator has three forms: *K%*, *D%*, and *Slow D%*, thus, there are usually three lines on a Stochastic Oscillator chart. D% is a moving average of K%. Slow D% is a moving average of D%. K% is calculated using equation 3.8:

$$K\% = \left(\frac{\text{Today's Close - Lowest Low in K\% Period}}{\text{Highest High in K\% Period - Lowest Low in K\% Period}} \right) * 100 \tag{3.8}$$

K% period refers to the time frame used for the calculation. Usually the time frame is 3 days. Then, D% is 3-day moving average of K%. Slow D% is 3-day moving average of D%. The Stochastic Oscillator always ranges between 0% and 100%. There are several ways to interpret a Stochastic Oscillator chart. Three popular methods are: (1) Buy when the Oscillator (either K% or D%) falls below a specific level (e.g. 20) and then rises above that level. (2) Sell when the Oscillator rises above a specific level (e.g. 80) and then falls below that level. (3) Buy when the K% line rises above the D% (or slow D%) line and sell when the K% line falls below the D% line.

- **Moving Average Convergence Divergence (MACD)**

The MACD is calculated by subtracting a 26-day moving average of a stock's price from a 12-day moving average of its price. The MACD indicator oscillates above and below zero, similar to the way the Chaikin Oscillator does. When the MACD is above zero, it means the 12-day moving average is higher than the 26-day moving average. This is bullish as it shows that current expectations of the market (i.e., the 12-day MA) are more bullish than previous expectations (i.e., the 26-day MA). When the MACD falls below zero, it implies a bearish shift in the market supply/demand lines. Usually a 9-day moving average line of the MACD (note not the stock price) is also plotted on a MACD chart, which is called the "signal" line. When the MACD line rises above its signal line, it gives a buy signal. When the MACD line falls below its signal line, it gives a sell signal.

As a summary, investors using technical analysis for investing or trading decisions generally have a short-term view of the market. Technical investors believe the stock

price is predictable, price patterns do exist and that history repeats itself. To these investors, the stock price charts follow trends and patterns do not necessarily reflect the fundamentals of a company as the graph does not show how a company's absolute value is changing overtime, rather, charts or indicators describe how thousands of investors' opinion of the company are changing, in other words, the market sentiment.

3.4 Evolutionary Computation and Financial Engineering

As we have stressed, none of the patterns or indicators discussed above should be used alone to predict future movements of a market. An investment strategy is always a combination of the analysis of various factors. Computational intelligence provides an efficient and effective way in the automation of analysing stock markets. Evolutionary computation techniques, such as genetic algorithms, genetic programming, learning classifier systems and evolutionary artificial neural networks, have been employed in various areas of financial engineering including time series predictions, pattern recognition, and market modelling etc (Bauer 1994; Chen 2002, 2002a; Schulenburg and Ross 2002; Cheng and Wang 2003; Tsang and Martinez-Jaramillo 2004).

3.4.1 GP and financial decision support

Tsang *et al.* (1998, 2000) and Tsang and Li (2002) describe a genetic programming based financial decision support system, named *EDDIE* (*Evolutionary Dynamic Data Investment Evaluator*), which was designed to work with financial experts. Financial experts select a set of data, e.g. stock prices or market indicators. EDDIE is then used to assist the experts in analysing the data by using genetic programming as a search

engine to explore the hypothesis space of potential correlations and patterns inside the data. For example, EDDIE needs to find rules for predicting whether the following goal is achievable at a given day:

Goal G: the index will rise by 4% within 22 trading days

FGP (Financial GP) in EDDIE used {If-then-else, And, Or, Not, <, >, =} as its function set; and various technical indicators, thresholds, and conclusions as its terminal set. Thresholds are real values that act as coefficients specifying correlations among the indicators. Conclusions, in the case of predicting goal *G* above, are either *positive*, i.e. G is predicted to be achievable, or *negative*. Potential prediction rules are randomly generated in the form of genetic decision trees/program trees by the FGP. Genetic algorithms are then used to evolve the population of genetic decision trees to discover good prediction rules that gives the best prediction results. EDDIE was tested on a wide range of data sets. In Li and Tsang (1999), FGP was tested for predicting whether a return of 4% or more is achievable within 63 trading days (3 months) on the Dow Jones Industrial Average (DJIA). EDDIE achieved an accuracy of 57.06% in the prediction, which outperforms six other simple technical rules. Note that, EDDIE requires its user to define the prediction task, the grammar for describing hypotheses and the factors to be included in the data set. Tsang *et al.* (2004) described an EDDIE-Automation architecture, which enables its user to conduct large-scale learning and monitoring in the stock market more efficiently.

Kaboudan (2002) attempted to answer the question as to whether GP can in fact evolve equations that predict stock prices. The author proposed a predictability test

based on an *eta* statistic, and concluded that at the 5% level of significance, prices are at least 0.620 predictable while the returns are at most 0.280 predictable, which implies, that stock returns are less predictable than price series. The paper also developed a single-day-trading strategy, i.e. a stock trader buys a stock at a price close to the daily low and sells at a price close to the daily high, based on genetic programming. The task of GP here is to predict the daily low and high. For all four stocks tested, returns from trading based on GP forecasting are higher than trading based on a random walk.

Genetic programming has also been shown to be successful in developing effective models for option pricing, i.e. formulating the value of an option before its expiration date is called, in an evolutionary manner, which constantly beat the conventional analytical approaches, such as the well known Black-Scholes model derived under strict assumptions that most do not hold in real world (see Chen *et al.* 1998; Keber 1999; Chidambaran *et al.* 2002). Although most of the GP applications in financial engineering argued that the major advantage of GP over other EC techniques, such as artificial neural networks, is that GP is able to generate solutions that are human-interpretable, there is still the controversial issue in genetic programming itself that GP is unreliable because it tends to produce over-complex and insensible solutions, i.e. the resulting decision rules/trees are syntactically correct but may not be semantically valid or sensible in the real world. Chen (2001) provides a survey on technical issues regarding applying GP in agent-based computational economics with

regards to the selection of function set and terminal set, semantic restriction, genetic operators and architecture.

3.4.2 EANN and stock trading

The problem of predicting stock price time series can be treated as a linear autoregressive time series modelling problem which assumes the prediction function F^L as a linear combination of a fixed number of past series vectors,

$$P(t) = F^L(t) = \sum_{i=1}^{T} \alpha_i P(t-i) + \varepsilon(t)$$

P(t) is the price at time t. *T* is the time window for looking back in the price history. *P(t-i)* is the past prices. α_i is the coefficient for the linear regression. $\varepsilon(t)$ is the noise term at time *t*. Artificial neural networks such as the multiple-layer feedforward perceptrons we described in section 2.4.1 can be used to approximate non-linear functions F^{MLP} as shown below,

$$F^{MLP}(P) = (\sum_{j=1}^{k} w_{jl}\sigma(\sum_{i=1}^{n} w_{ij}P_i - \theta_j) - \theta_l), \qquad l = 1, \ldots, m$$

where *k* is the number of number of hidden units. *n* is the number of inputs. *m* is the outputs. w_{jl} and w_{ij} are connection weights. P_i are inputs, i.e. past stock prices. θ_l and θ_j are thresholds. σ is the non-linear activation functions used by the neural net. The non-linear autoregressive models represented by artificial neural networks are more powerful than linear models because they can model more complex underlying characteristics of time series (Dorffner 1996).

Conventionally, the non-linear function approximation in artificial neural networks for time series prediction is carried out through the training of neural network over datasets of past stock prices using learning algorithms such as backpropagation (see Table 2.11). Although one of the early studies, White (1988), used artificial neural networks for studying returns on the IBM stock and did not find evidence against the EMH theory, but the paper only experimented with a simple single-layer MLP. With more complex network architectures and learning methods, ANNs have been consistently shown to outperform statistical and regression models for stock market prediction. Kimoto *et al.* (1990) used a modular neural network consisting of 4 backpropagation networks trained on different data items, and outperformed the buy and hold strategy and the Tokyo index. The backpropagation MLP used in Yoon and Swales (1993) predicted price trends correctly 91% of the time as compared to 74% using an analytical model. Kamijo and Tanigawa (1993) developed a recurrent neural network approach that recognised the correct price pattern in 15 of 16 cases studied (93.8% accuracy). Chenoweth *et al.* (1996) demonstrates a neural network based stock trading system using technical indicators, which consists of a feature selection component, two specialised neural networks for stock returns prediction, and a rule base for prediction integration. The authors also used a filter for data pre-processing which categorises input data into an "up trend" dataset and a "down trend" dataset, and the two neural networks are trained on the two datasets respectively using a backpropagation algorithm. The predictions from the neural nets are integrated in the post-processing phase. The trading system was tested on the S&P 500 index for a five year trading period and yielded an annual rate of return of 15.99% and outperformed

the buy and hold strategy (11.05%). Numerous research papers in the literature have used artificial neural networks for financial time series prediction. Trippi and Turban (1993, 1995, 1996), Azoff and Azoff (1994), Zirilli (1996), and Shadbolt and Taylor (2002) provide comprehensive coverage on the advancement in neural networks for finance and investment.

Major technical issues involved in applying ANNs to stock markets include the determination of proper input data and the design of appropriate neural net architectures, such as the number of layers and hidden nodes, connectivity and connection weights optimisation, selection of activation functions etc. EANNs, as we discussed in section 2.4.3, provide an evolutionary approach for effectively developing artificial neural network models for stock market applications. Yao and Liu (1997a) first applied their EPNet, see Figure 2.10, to two chaotic time-series prediction problems, the Mackyey-Glass differential equation and the logistic map problems, and was able to discover novel and very compact ANN architectures that would be very difficult to design by human beings without priori knowledge. Liu and Yao (2001) then tested the EPNet for Hang Seng Stock Index Forecast, and their best-evolved neural network generalised very well and was able to capture almost all the changes in the tested index. Chen and Lu (1999) compared EANN models with a random walk model and several Backpropagation ANNs with pre-specified structures and all EANN models outperformed the random walk model in forecasting, but the performance of EANNs as compared to the Backpropagation ANNs are mixed. Kwon and Moon (2003) developed a neuro-genetic stock trading system that used 64 inputs

based on popular technical indicators. Their trading system uses a recurrent neural network for daily stock trading. A parallel genetic algorithm is used to evolve neural nets. The authors tested the system on 36 company stocks from the NYSE and NASDAQ over a period of 10 years. The trading system outperformed the buy and hold strategy in 153 cases, was comparable in 118 cases, worse in 88 cases.

3.5 Evolutionary Computation and Artificial Stock Markets

Modelling stock markets is not an easy task due to the fact that numerous different types of participants/agents, see Figure 3.1, co-exist in the market. The interaction and interrelationship among these heterogeneous agents are very complex. The traditional approach to model a stock market is to represent it as a representative analytical model based on the *rational expectations* theory of economics. The rational expectations theory (Muth 1961) has a number of strong assumptions over investors in stock markets: (1) It assumes all agents, e.g. an individual investor or a firm, have full knowledge/information of the market. (2) All agents are also assumed to possess the same perfect rationality in investing. (3) All agents are assumed to know that all other agents in the market are investing based on the same information and the same perfectly rational model. The strong assumptions that sustained the classic analytical models of stock markets seriously contradict the fact that the stock market is an evolutionary and adaptive system where heterogeneous agents with different beliefs coexist (Palmer 1994). As stressed by the asymmetric information theory, there is a lack of complete information in markets. Agents in a market may have to learn about the context or about other agents while the "game" is played out. Problem contexts

may themselves not be fully defined initially, e.g. the evolving market structure and market policies, only becoming explicit through the choices of agents. Different agents may well have different information about a situation, and may well use different approaches to interpret the information. Investors, in general, do not rely on others to duplicate their own reasoning, and real persons and companies are not always as rational as they are expected to be. All the realities from real world markets demand the stock market to be studied as an evolutionary system, rather than as a single representative model.

Evolutionary Artificial Stock Markets (ASM) provide the bottom-up evolutionary approach for modelling stock markets, where large populations of artificial agents are built initially with little knowledge and co-evolve with each other. The overall market dynamics, e.g. stock price time series, and market behaviours, e.g. herd following, is in fact the emergent property from the artificial agents' interactions.

3.5.1 Santa Fe artificial stock markets

One of the pioneering works in evolutionary artificial stock market was developed in Palmer *et al.* (1994) and used a learning classifier system, which was later developed into two software versions at the Santa Fe Institute. The following discussions on the Santa Fe artificial stock market are mainly adapted from Palmer *et al.* (1994), Arthur *et al.* (1997), Lebaron (1999), LeBaron (2001) and Tesfatsion (2005).

In the Santa Fe artificial stock market (SFI-ASM), time is broken up into discrete time periods $t = 1, 2, \ldots T$. The stock market consists of an Auctioneer together with I stock

market traders, *i = 1, 2, ... I*. Traders are identical except that each trader individually forms his expectations over time through an inductive learning process. More precisely, a learning classifier system is used as a forecasting model to form expectations of individual traders. A GA is used in the inductive learning process. Each trader has the same positive wealth W_0 in the initial period *0*. There exists a risk-free financial asset *F* (e.g. Treasury bills), available in infinite supply. Asset *F* pays a constant risk-free return rate, denoted by *r*. There exist *N* shares of a risky stock *A* available at the beginning of each period *t*. Artificial traders do not have precise knowledge of the fundamental share value for stock *A* in any time period *t*. This is because the dividend d_t paid by stock *A* in each period is generated by an exogenous stochastic process unknown to the traders. Traders have identical CARA (*constant absolute risk aversion*) utility-of-wealth functions for optimisation of the form:

$$U(c) = -\exp(-\lambda W)$$

Each trader *i* in each period *t* chooses his portfolio of financial assets in an attempt to maximise his expected utility of period *(t+1)* wealth subject to the budget constraint:

$$W_{t+1}^{i} = x_t^i * (P_{t+1} + d_{t+1}) + (1 + r)(W_t^i - P_t x_t^i)$$

Here x_t^i is the final holding of stock *A* for trader *i* at time *t*. $(W_t^i - P_t x_t^i)$ is the amount of risk-free financial asset *F*.

At the start of the market process in period 1, each trader *i* has a set of *M* if-then forecasting rules, i.e. the classifiers. Each trader selects a forecasting rule and generates a prediction of next period's stock price and dividend $(E_{t,i}(P_{t+1} + d_{t+1}))$ together with an updated estimate of the rule's forecast variance. The fitness of *j*th

forecasting rule depends on the rule's forecast variance ($\sigma^2_{t,i,j}$) and on its specificity ($s_{t,i,j}$) as below,

$$f_{t,i,j} = \sigma^2_{t,i,j} - c * s_{t,i,j}$$

Each trader individually evolves his own set of forecasting rules/classifiers over time on the basis of fitness, retaining the fittest rules and replacing the least fit rules with variants of rules with higher fitness, as we have discussed in Chapter 2.7.3.

In each time period, each trader *i* decides how much of his wealth to invest in the risky stock *A* and how much to invest instead in the risk-free asset *F*. Trader *i* does this by generating a demand $s_{t,i}$ for stock *A* under the CARA utility, i.e. how much trader *i* wants to buy or all on stock *A*, as a function of: (a) his current predictions for next period's price and dividend; and (b) the yet-to-be-determined period-*t* price of stock *A*, as shown in the equation below:

$$s_{t,i} = \frac{E_{t,i}(P_{t+1} + d_{t+1}) - P_t(1+r)}{\gamma \sigma^2_{t,i,p+d}}$$

This demand is communicated to the Auctioneer. The Auctioneer collects all demands from all traders, and then determines a market-clearing price for stock *A* depending on the overall demand (buyer) and supply (seller) in the market, and communicates this price back to each of the traders. Given this market-clearing price, each trader then goes ahead and purchases or sells his corresponding demand, and the artificial market goes into next time period.

As described above, the price time series of stock *A* is generated endogenously within the artificial market as an emergent behaviour through the trading activities of artificial traders. This clearly demonstrates a more appropriate way in modelling

market dynamics from bottom up, rather than from top-down through a representative analytical model. Arthur *et al.* (1997) and Lebaron *et al.* (1999) studied the time series generated in Santa Fe artificial stock markets, claiming that the artificial stock market was able to demonstrate bubbles, crashes, and continued high trading volumes that are observed in real world stock market price time series. LeBaron (2000) also studied the micro-behaviours of artificial traders regarding learning speed and its effect on market behaviour. The author discovered that in the case of slower learning (i.e. the element GA of the LCS system occurring after long periods), the resulting market behaviour appeared very close to the benchmark market under rational expectations, while when the learning occurs more frequently, the artificial traders begin to make use of the technical trading bits in the classifiers, and bits associated with dividend/price ratio, and market prices reveal volatility persistence and increased trading volume which have been observed in actual market.

Palmer *et al.* (1994) also points out the significance of studying evolutionary artificial stock markets in the context of modelling artificial lives. The authors argued that:

> *"… We find the self-formation of an autonomous economy that boot straps itself up from randomised 'stupid' behaviour to organized mutually-adapted behaviours. From a random soup of simple rules, an 'intelligent' system spontaneously organizes. Thus we have modelled the origins of economic life among interacting agents in the same sense that others model the origin of biological life among organic molecules."*

The paper concluded that evolutionary stock market models can be used as a fertile test bed for exploring, not only financial markets, but also adaptive agents and a class of artificial lives.

3.5.2 GP-based artificial stock markets

Chen and Liao (2000), Chen and Yeh (2001), and Chen and Yeh (2001a) described another type of evolutionary artificial stock market that is based on genetic programming. Their GP-based artificial stock market uses the same standard asset pricing model (Grossman and Stiglitz 1980) as in the Santa Fe artificial stock market. Please refer to Chen and Yeh (2001) and Chen *et al.* (2002) for a detailed description on their market structure.

Besides the study of market dynamics, the other important technical issue explored in GP-based artificial stock market is the differences between two distinguishing learning paradigms, i.e. the *multi-population GP (MGP)* and the *single-population GP (SGP)*, and their effect on modelling artificial lives. In a *single-population GP* (Andrew and Prager 1994; Chen and Yeh (1996, 1997)), each artificial trader in the artificial market is modelled using one single genetic decision tree. The population of artificial traders, i.e. a single population of genetic decision trees, is evolved through the direct communication between artificial traders by means of genetic crossover and mutation. In other words, each artificial trader's knowledge/trading strategy, i.e. his GP tree, is available to the public with no cost. This apparently contradicts the real-world market, where a trader's mind is generally not directly observable to others unless a trader

chooses to reveal himself to the public. However, *imitations, speculation* and *herd behaviours* are popular phenomena that do exist in most social environments, particularly in financial markets. Harrald (1998) argued that the adaptation at the *genotype* level (underlying genetics) may not be achievable via the simple imitation on the *phenotype* level (actions), i.e. simple imitations of other traders' actions may not be helpful for one trader to learn another's trading strategy. When an artificial agent's mind is not directly observable by another agent in a social environment, how to learn from others becomes an important issue that need to be resolved in SGPs.

Chen and Yeh (2001) made a first step to answer the problem of how to learn from others while an agent's mind is not observable. In their experiments, a new SGP architecture with a mechanism called "*school*" was proposed. The *school* mechanism can be viewed as the public media (e.g. financial publications and pubic seminars) or research consultants (e.g. financial advisors and academic researchers) in financial markets that conduct their business based on providing as much information/knowledge as possible to investors. The *school* is a collection of well-developed trading strategies and the *school* itself also evolves with the market. Artificial traders in the SGP-based ASM learn from the *school*, rather than by directly communicating with other agents in the market, hence the problem of unobservable minds is solved. Figure 3.4 demonstrates artificial traders' learning with the *school* mechanism, adapted from Chen and Yeh (2001).

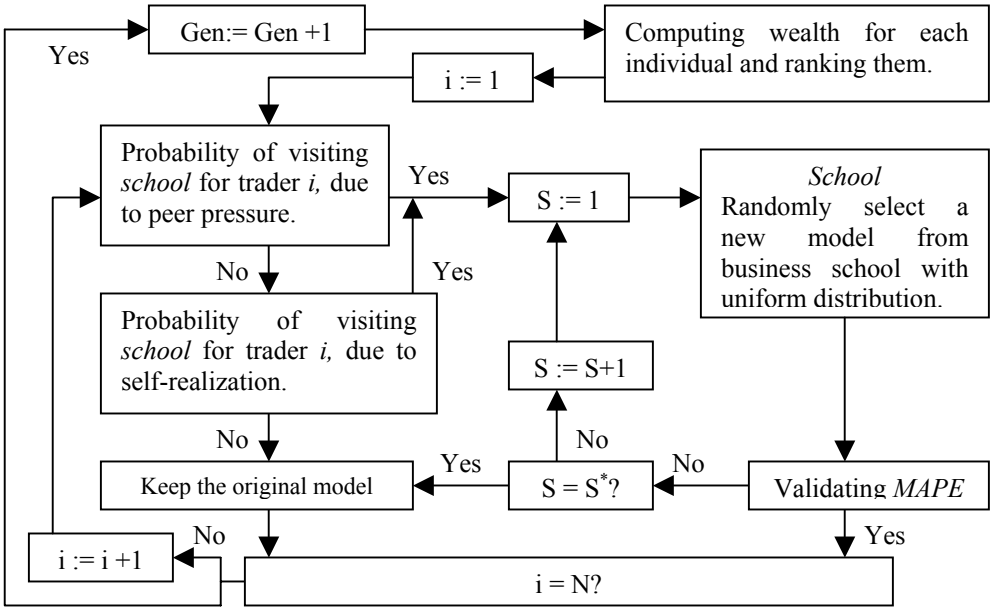


FIGURE 3.4 Single-population GP with the *school* mechanism

In Figure 3.4, N is the total number of artificial traders in the market. S is the maximum number of models a trader could select from the *school*. When a trader i decides to learn from the *school*, under pressure both from himself and other competitors in the market, the trader will repeatedly randomly select a forecasting model, i.e. a GP tree, from the *school* until the mean absolute percentage error (MAPE) for forecasting of the selected model over the last n-day period trading is better than the model that trader i is holding and replace his old forecasting model with the better one learned from the *school*; or stop when the value of S exceeds S^* and continue to the next trader or the next generation. The probability of a trader deciding to learn from the *school* is determined by two factors: *peer pressure* and *self-realisation*. *Peer pressure* examines how well an artificial trader has performed over the last n_1-day trading period, when compared with other traders. Assume the traders

are ranked by the net change of wealth over the last n_1-day trading days. Let $W_{i,t}^{n_1}$ be the net change of wealth of trader *i* at time period *t*, i.e.

$$\Delta W_{i,t}^{n_1} = W_{i,t} - W_{i,t-n_1}$$

and let $R_{i,t}$ be trader *i* rank. Then, the probability of trader *i* going to the *school* at the end of period *t* is defined as in equation 3.9:

$$p_{i,t} = \frac{R_{i,t}}{N} \tag{3.9}$$

so that traders that did not perform well will have more pressure to learn. *Self-realisation* simply evaluates how well the trader thinks he has performed in the last n_1-day trading days. Let the growth rate of wealth of trader *i* at time period *t* be

$$\delta_{i,t}^{n_1} = \frac{W_{i,t} - W_{i,t-n_1}}{\left| W_{i,t-n_1} \right|}$$

The probability of trader *i* will search in the *school* for better forecasting models is then defined as equation 3.10:

$$q_{i,t} = \frac{1}{1 + e^{\delta_{i,t}^{n_1}}} \tag{3.10}$$

The probability ($r_{i,t}$) that trader *i* decides to change his mind, i.e. learn from the *school* is then:

$$\begin{aligned} r_{i,t} &= p_{i,t} + (1 - p_{i,t}) q_{i,t} \\ &= \frac{R_{i,t}}{N} + \frac{N - R_{i,t}}{N} \times \frac{1}{1 + e^{\delta_{i,t}^{n_1}}} \end{aligned} \tag{3.11}$$

In Chen and Yeh (2001), the single-population based artificial stock market with the *school* learning mechanism was able to replicate major time series features of real

markets. The authors also demonstrated that their resultant artificial time series on price evidence that stock returns in the artificial stock market is independently and identically distributed, and hence supports the efficient market hypothesis. However, the authors also pointed out this time series that supports the EMH, is in fact generated by artificial traders who do not believe in the EMH and uses technical indicators as inputs to their GP forecasting tools.

3.5.3 Individual learning and social learning

Minsky (1986) described his well-known *society of mind* theory that claims intelligence is an emergent property from societies of processes that constantly challenge one another, rather than from a centralised principle. *Multi-population GP*, similarly, represents each artificial agent using a population of genetic decision trees. Therefore, the GA is in fact run inside each agent so that an agent evolves his own mind of forecasting models. In most applications of MGP for modelling social systems, direct communication among agents do not exist due to the fact that minds are not directly observable. The multi-population and single-population learning paradigm can be easily generalised to other evolutionary algorithms (EAs), e.g., single-population parallel GAs and multi-population parallel GAs and the Michigan and Pitts approach in LCSs. However, note that for most implementations of multi-population parallel GAs, i.e. island-GAs, direct communications among '*islands*' do generally exist.

Comparing single-population EAs (SEA) and multi-population EAs (MEA), within the single-population architecture, agents essentially learn from other agents' experience, whereas within the multi-population architecture, agents basically learn from their own experience when their minds are not directly observable. Therefore, in Vriend (2000), single-population learning paradigm is defined as *social learning;* multi-population learning paradigm is defined as *individual learning*. Figure 3.5 diagrammatically illustrates the concepts of individual learning and social learning.

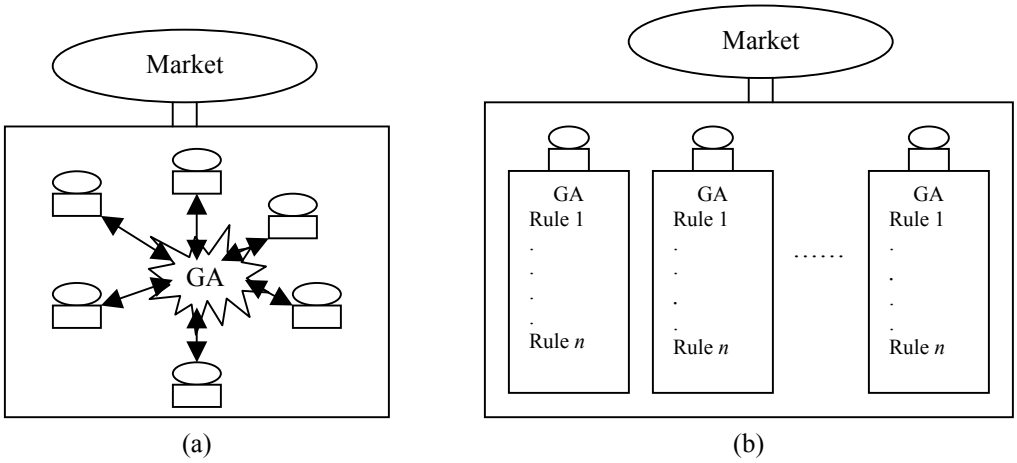


FIGURE 3.5 (a) Social learning with SEAs (b) Individual Learning with MEAs

Vriend (2000) experimented with the individual learning paradigm and social learning paradigm in the context of a standard Cournot oligopoly game, and demonstrated that individual learning and social learning converged to two different output levels and utility levels of firms from the game. The author concluded that *"...the computational modelling choice made between individual and social learning algorithms should be made more carefully, since there may be significant implications for the outcomes generated."*

Although Chen and Yeh (2001) overcame the problem of unobservable minds by means of a *school* mechanism, it is obvious that individual learning is missing in their single-population based learning architecture. Chen and Yeh (2001a) attempted to integrate the individual learning and social learning processes in the multi-agent artificial stock market by replacing the single-population GP with a multi-population GP. There is no change to the *school* mechanism. After trader *i* calculates his probability ($r_{i,t}$) and decides if he wants to change his mind using equation 3.11, the trader now has two options: either go to the *school* to search for new ideas, or learn by himself through individual learning. Let $p_{i,t}^{search}$ be the probability that a trader would like to look for new ideas from the *school*, the probability that a trader would like to work out some new ideas by himself is then $(1-p_{i,t}^{search})$. $p_{i,t}^{search}$ is determined as following:

$$p_{i,t}^{search} = \begin{cases} p_{i,t-1}^{search} - (r_{i,t} - r_{i,t-1}) p_{i,t-1}^{search}, & \text{if } r_{i,t} - r_{i,t-1} \geq 0, \text{ Case 1.} \\ p_{i,t-1}^{search} - (r_{i,t} - r_{i,t-1})(1 - p_{i,t-1}^{search}), & \text{if } r_{i,t} - r_{i,t-1} < 0, \text{ Case 1.} \\ p_{i,t-1}^{search} + (r_{i,t} - r_{i,t-1})(1 - p_{i,t-1}^{search}), & \text{if } r_{i,t} - r_{i,t-1} \geq 0, \text{ Case 2.} \\ p_{i,t-1}^{search} + (r_{i,t} - r_{i,t-1}) p_{i,t-1}^{search}, & \text{if } r_{i,t} - r_{i,t-1} < 0, \text{ Case 2.} \\ p_{i,t-1}^{search} & \text{Case 3.} \end{cases} \quad (3.12)$$

Case 1 means that the trader looked for a new idea from the *school* at period $t-1$. Case 2 means the trader developed a new idea by himself at period $t-1$. Case 3 means the trader did not change his mind at period $t-1$. What equation 3.12 implies is that if a trader has a high motivation to change his mind, then he will think about whether the result is due to the wrong decision, for example, consulted the *school*, made in the previous period or not. Therefore, he is prone to reduce his probability to consult the *school* at time *t*. Traders learn by themselves through individual learning by means of

conventional GA operations. With the integrated individual learning and social learning architecture, Chen and Yeh (2001a) demonstrated that there is little difference in the macro-structures of the artificial stock market, e.g. price time series and stock returns that supports EMH, was observed during the experiments compared to the single-population GP based artificial stock market. However, their experiments also showed different learning architectures resulted in different microstructures of the resulting artificial stock markets regarding traders' beliefs and market behaviour. In the following chapters, we will also demonstrate how the integrated individual learning and social learning paradigm is employed for the development of imperfect evolutionary systems.

3.5.4 ANN and LCS based artificial stock markets

A recent application of artificial neural networks in modelling artificial stock markets can be found in Yang (2002), which distinguishes itself from the conventional Santa Fe ASM in its trading institutional design. Rather than using a central auctioneer as in most ASMs, Yang (2002) considered a double auction market mechanism, which is one more step closer to real world stock markets, and hence makes it possible to compare the simulation results with the behaviour of real data, e.g., tick-by-tick data. In the artificial double auction market, artificial traders are modelled using one-hidden-layer neural nets as forecasting tools. Yang's artificial market was able to replicate basic market dynamics as learnt from the conventional artificial stock markets. Yang also studied the presence of momentum traders, i.e., traders who based their trading strategies on technical indicators in the artificial market. The author then

finds that the presence of momentum traders can not only drive the market price away from the HREE (homogeneous rational expectation equilibrium) price, but also generate a lot of interesting phenomena, such as excess volatility in the market, and high trading volume. More early applications of ANNs in modelling financial markets can be found in Beltratti *et al.* (1996).

The LCS-based artificial stock market developed in Schulenburg and Ross (2000, 2001) differs from other artificial stock markets in that the basic underlying market scenario, such as daily stock prices, dividends, trading volumes, etc. is given exogenously, rather than generated endogenously as in the *Santa Fe* ASM. Therefore, the aim of Schulenberg and Ross's artificial stock market is not to replicate the time series features of real markets, but to evolve better trading strategies within the artificial stock market. In Schulenburg and Ross (2000), three different types of traders are modelled using LCS for trading decision making using different information sets. The first type of trader mainly uses price statistics and moving averages (see section 3.3). The second type of trader basically uses trading volume statistics. The third type of trader uses both price and volume statistics in addition to other information such as the accumulated wealth of both the buy-and-hold strategy and the risk-free bank savings, and a recollection of past action. Each type of trader is modelled as a LCS system in the *Pitts* fashion, i.e. each type of trader has a set of classifier rules and evolves himself by means of individual learning. The three types of artificial traders are tested on a single stock for a 9-year trading period – the adaptive phase, i.e. traders evolve their set of classifier rulers through reinforcement learning

that act upon immediate reward with the aim to accumulate as much wealth as possible in the long run, and a 1 year testing period, i.e. no learning occurs and it simply tests if the evolved rules can be generalised into different market scenarios. Results from experiments demonstrated that all three types of artificial traders were able to evolve novel trading strategies that outperformed both the buy-and-hold strategy and the risk-free bank savings. Artificial traders were able to detect useful trading signals, e.g. upwind or downwind trend, from the market. During the testing period, artificial traders were able to beat the buy-and-hold strategy and bank savings. Schulenburg and Ross (2001) extended the experiments to multiple stocks and compared artificial traders' performance to a trend-following strategy. Artificial traders achieved very promising performance in most cases. The authors claimed their evolutionary LCS approach was an excellent approach to make trading decisions in the stock market.

3.6 From A Perfect Paradigm to An Imperfect Paradigm

Conventional competitive equilibrium model of economics (Arrow and Debreu 1954) view financial markets as competitive systems and assert that competitions of market players will eventually result in equilibriums, e.g., supply and consume, assets pricing or allocation of resources etc. One of the strong assumptions that sustain the standard competitive paradigm of economics is the assumption of the existence of perfect information in markets. Under this assumption, markets are fully informationally efficient. Information is disseminated efficiently and perfectly throughout the

economy. All participants of the market have the same information, i.e., "I know what you know and you know what I know". The efficient market hypothesis, see section 3.1, is one such market model based on perfect information and says that stock prices convey all the relevant information from the informed to the uninformed, i.e., stock prices reflect a stock's intrinsic value, hence, any speculation on stock prices is pointless. Rational expectations, see section 3.5, is another example of such perfect information paradigm that models market players with perfect rationality.

The spirit of the standard competitive paradigm centres on the word "equilibrium". However, many phenomena we observed from real world markets objects to this "equilibrium" mechanism. For example, the persistent and large-scale unemployment in labour markets or the excessive volatility in assets prices from financial markets and property markets. The competitive paradigm of economics cannot explain such phenomena where markets failed to achieve the desired equilibrium. *Information economics* (Stiglitz 2003) attempts to explain these market failures by questioning the assumption of the existence of perfect information in competitive paradigms. Early work in information economics mainly dealt with how markets overcame problems of information asymmetries (Akerlof 1970; Spence 1974; Rothschild and Stiglitz 1976). Later work was more focused on how actors in markets create such information problems. Grossman and Stiglitz (1976, 1980) criticises the efficient market hypothesis by showing that, when information is costly to collect, stock prices necessarily aggregate information imperfectly. The author argued that under perfect competition, stock prices reflect all of the information as stated in the efficient market

hypothesis. But if the market is perfectly efficient, no one will have any incentive to gather any information. Investors can get all the information by just looking at prices. If no one gathers information, however, then stock prices do not have any information to reflect. This paradox lays the basis for the argument that imperfect information in markets is likely to be the rule rather than the exception. Therefore, markets are neither perfectly efficient nor completely inefficient. Stock prices are essentially the aggregators and disseminators of information. The authors concluded that the costliness of information eventually results in the impairment of informational efficiency in the market and asymmetrically informed stock traders. This implies better informed investors will have opportunities to take advantage of inferior investors; hence, speculation is not necessarily meaningless when information is costly and the speculative behaviour of traders causes volatility in stock prices. The study of information economics also extends into different fields in economy, for example, corporate governance under an imperfect information paradigm (Shleifer and Vishny 1989) or welfare economics in imperfect economies (Arrow *et al.* 2003).

Arnott *et al.* (2003) describes information economics as a change from the perfect information paradigm in competitive models to an imperfect information paradigm for economics. It points out that many features of real-world transactions can only be understood in an imperfect information framework. The imperfect information paradigm provides an alternative way that helps people understand economic phenomena much better. Under the same principle, when we struggle with the adaptation problem of the conventional research paradigms in artificial intelligence,

where artificial intelligence is regarded as a complete perfect end product, we cannot help thinking if artificial intelligence research needs a new perspective: an imperfect perspective? The answer is yes. If economics for an imperfect world has been proved important in solving economic problems, artificial intelligence for an imperfect world is also demanded in order to understand intelligence better. Minsky (1986) described his well-known *society of minds* theory as following:

> *"…What magical trick makes us intelligent? The trick is that there is no trick. The power of intelligence stems from our vast diversity, not from any single, perfect principle. Our species has evolved many effective although imperfect methods, and each of us individually develops more on our own. Eventually, very few of our actions and decisions come to depend on any single mechanism. Instead, they emerge from conflicts and negotiations among societies of processes that constantly challenge one another…"*

Minsky's description on intelligence reveals intelligence as an emergent property in social environments, rather than a single isolated entity. We cannot talk about the development of intelligence without considering the incomplete environment that the intelligence resides in. If the world we are living in is imperfect, how can we expect AI to be a perfect end product, which is unable to cope with an imperfect and constantly changing environment? A change from the conventional perfect paradigm to an imperfect paradigm is needed in artificial intelligence research, which leads to our proposal of imperfect evolutionary systems. Minsky's description also points out that the development of intelligence is not only processes of individual learning but

also from interactions among societies of processes that constantly challenge each other, i.e. social learning. We believe the integrated individual learning and social learning architecture examined in this chapter provides an effective mechanism for implementing imperfect evolutionary systems.

3.7 Summary

Three major conclusions from this chapter are:

1. Due to its high complexity and heterogeneity, the stock market serves as an excellent test bed for modelling evolutionary artificial lives.
2. Changing from the conventional perfect information paradigm to an imperfect information paradigm points out an alternative way for artificial intelligence research.
3. Integrated individual and social learning paradigm provides an effective mechanism for developing imperfect evolutionary systems.

In this chapter, we also looked at various investment strategies related to the stock market including the buy and hold strategy and various fundamental and technical trading strategies. We discussed the applications of evolutionary computation techniques in terms of financial engineering. We described various techniques in modelling stock markets as evolutionary adaptive systems including learning classifier systems, genetic programming and artificial neural networks. Although most researchers who used learning classifier systems or genetic programming for ASMs argued that these two techniques are more easily interpreted and hence human

readable compared with artificial neural networks, there are still issues that need careful consideration regarding LCS and GP. In learning classifier systems, the rules of classifiers need to be hand-coded by system developers. For example, in Schulenberg and Ross (2001), their artificial trader (type 1) used a rule with one condition as: $p_t > 1.025 * p_{MA30}$, which says check if the current price is higher than the 30-day moving average multiplied by 1.025. Here, people could ask why was a value of 1.025 chosen as the coefficient and not 1.035? The design of rules in classifier systems is entirely dependent on the developers' and other experts' knowledge on a particular problem.

Regarding genetic programming, the exponential increase in the size of genetic decision trees is a major technical issue in GP applications (see section 2.7.4). Chen (2001) discusses most of the major technical issues in applying GP in agent-based computational economics with regards to the selection of the function set and the terminal set, semantic restriction, genetic operators and architecture, and argued that unless the above issues are well addressed, genetic programming is not well grounded in consideration of human behaviour, and it would be premature to claim that GP has modelled a population of agents learning over time. In the following chapters, for the implementations of imperfect evolutionary systems, we choose artificial neural networks for the modelling of artificial agents for several reasons: First, artificial neural networks most closely emulate the human brain; Secondly, artificial neural networks do not require human expertise in solving a problem (see *Blondie24* in

section 2.6); Third, artificial neural networks have the capability to model highly non-linear functions and noise tolerant for forecasting problems (see section 3.4.2).

Chapter 4

Imperfect Evolutionary Systems

This chapter gives a formal description of our imperfect evolutionary systems. We use two examples, i.e. stock market and game playing, to illustrate our definition of imperfect evolutionary systems. An integrated individual learning and social learning paradigm is described as one of the possible mechanisms for developing an imperfect evolutionary system.

4.1 Definition of An Imperfect Evolutionary System

In chapter 2 of this thesis, we gave an analytical review on artificial intelligence research and evolutionary computation. We questioned the conventional research paradigm in artificial intelligence where AI is treated as a perfect end product and asked whether perfect intelligence is what AI is aiming for. In chapter 3, we examined how the change from a perfect information paradigm to an imperfect paradigm in information economics has changed people's understanding on economic problems, and concluded that artificial intelligence research also requires a new perspective: from AI as perfect end products to AI that continuously learns to adapt in incomplete environments. We argue that the fundamental problem, as with many existing evolutionary approaches in their attempt to develop artificial intelligence, is the ignorance of an intelligent entity's membership to its environment. Furthermore, it is a

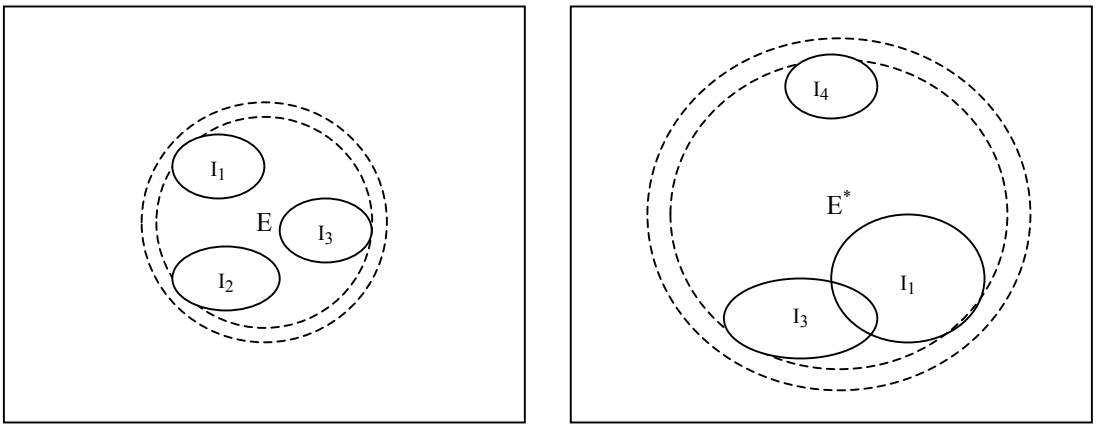


FIGURE 4.1 Imperfect Evolutionary Systems

membership to an incomplete environment. By saying an evolutionary system is imperfect, we mean not only that the participants of an evolutionary system are imperfect because they are unable to understand their environment completely and every individual has (possibly) a different partial view of the environment due to the complexity of the environment, but also that the environment itself is incomplete as it is constantly evolving as new information and knowledge emerge over time. Figure 4.1 above depicts an imperfect evolutionary system schematically.

We define an incomplete environment E as a set of environmental variables (v_j), as shown in equation 4.1,

$$E = \{v_1, v_2, \cdots, v_m\}. \tag{4.1}$$

In the incomplete environment E, there exist a set of individuals I.

$$I = \{I_1, I_2, \cdots, I_n\}. \tag{4.2}$$

Where I_k is an imperfect individual. Each I_k itself is a complex system that continuously learns to adapt to the changing environment E with resources available.

Denote the available resource to an individual I_k as Inf_k. Inf_k is then a subset of E, i.e. $Inf_k \subseteq E$.

I_k has two tasks:

1. Optimise its own utility with available resources through an individual learning process.
2. Adapt to the changing environment through the search of better information and knowledge by means of social learning.

In other words, I_k can be represented as in equation 4.3.

$$I_k = (Inf_k, IL_k, SL_k) \tag{4.3}$$

Where

IL_k = Individual learning mechanism of individual I_k.
SL_k = Social learning mechanism of individual I_k.

We use two examples to illustrate Figure 4.1 and our concept on imperfect evolutionary systems. For the first example, assume the environment *E* in Figure 4.1 is a stock market. Initially, in Figure 4.1(a), available information/knowledge in the market includes stock prices (*price*), 10-day moving average (MA_{10}), 50-day moving average (MA_{50}), trading volume (*volume*):

$$E = \{price, MA_{50}, MA_{200}, volume\}$$

where

$$I_1 = \{price, MA_{50}\}; I_2 = \{price, MA_{200}\};$$

$$I_3 = \{price, MA_{50}, volume\}.$$

Three different types of traders exist in the market and they have different beliefs in what makes a good trading strategy. Along with the evolution of the market, especially when more and more individual investors invest in the stock market and most of them take a momentum trading or day trading approach, the market has changed. The environment E^* in Figure 4.1(b) expands and incorporates new information and knowledge:

$$E^* = \{price, MA_5, MA_{10}, MA_{50}, MA_{200}, volume, S\ \&\ P500, RSI, dividend, interest\ rate\}$$

Existing in such an imperfect evolutionary environment, traders also evolve to adapt to the changing market. Individual I_1 learns to use market indexes and macroeconomic indicators:

$$I_1 = \{price, MA_{10}, MA_{50}, MA_{200}, S\ \&\ P500, interest\ rate\}.$$

Trader I_3 learns to incorporate more short-term price trends in his strategy:

$$I_3 = \{price, MA_5, MA_{10}, MA_{50}, MA_{200}\}.$$

Individual I_2 retreats from the market due to serious losses, but a new trader, I_4 joins the market with new belief in investing:

$$I_4 = \{MA_5, RSI, S\ \&\ P500, volume\}.$$

Under the conventional research paradigm of artificial intelligence, where intelligence is treated as a perfect end product, we could not find an explanation or a solution to emulate the adaptive intelligence of traders in an imperfect evolutionary market.

For the second example, let us consider imperfect evolution on a micro-level. Assume the environment *E* corresponds to the "game playing" environment a human living in:

$$E = \{\{featureset\ of\ tic\text{-}tac\text{-}toe\};\ \{featureset\ of\ checkers\};\{featureset\ of\ poker\}\}$$

Individuals I_i here can be viewed as specialised sections in a simulated brain that are in charge of different game playing domains. A stimulus from the environment termed as "tic-tac-toe" will evoke section I_1 and be ready for a tic-tac-toe game.

$$I_1 = \{input_1^{tic}, input_2^{tic}, \cdots, input_m^{tic}\}.$$

A stimulus flagged as "checkers" will activate section I_2 and enables the brain to play a game of checkers.

$$I_2 = \{input_1^{checkers}, input_2^{checkers}, \cdots, input_n^{checkers}\}.$$

A stimulus signalled as "poker" will trigger section I_3 to play a poker game.

$$I_3 = \{input_1^{poker}, input_2^{poker}, \cdots, input_p^{poker}\}$$

Note that these specialised sections are not static. The simulated brain always learns from the games it has played. Assume we use artificial neural networks for modelling game strategies. Section I_1 in Figure 4.1(b) will differ from I_1 in Figure 4.1(a) in both the network architecture and the game features used as inputs. The new section I_4 in Figure 4.1(b) shows the simulated brain learns to play a new game and has lost interest in checkers. Once again, when we view intelligence from a perfect perspective, we could not explain why humans are able to learn from the games they played and also learn to play new games. Only when we view human intelligence as imperfect and evolutionary, will we find explanations to understand the adaptability of human intelligence better.

4.2 An Integrated Individual and Social Learning Paradigm (ISP)

In chapter 3, we showed that the integrated individual and social learning mechanism provides an effective way for modelling co-evolution of artificial lives in complex and dynamic environments such as the stock market. Individual and social learning is not new in the AI literature, which can be found either in the more abstract theories such as society of minds, or from more concrete evolutionary learning paradigms, such as the migrations among islands in distributed GAs, the pheromones in ant colony optimisation, the memes in memetic algorithms, the velocities in particle swarm optimisation and belief space in cultural algorithms. However, none of the existing machine learning methods had paid attention to the incompleteness of environments in their learning paradigms, which is the major reason why these learning paradigms cannot fully explain the adaptability and creativity of human intelligence. Incorporating imperfectness into individual and social learning, we propose an integrated individual and social learning paradigm (ISP) as a general framework for developing imperfect evolutionary systems. The ISP paradigm is schematically depicted in Figure 4.2 below, and consists of four building blocks: *The Incomplete Environment (E)*, defined in equation 4.1, not only provides its participants with resource for surviving and acts as the media for evolution, but also it itself evolves while new information and knowledge emerge over time, as shown in Figure 4.1. In such an imperfect environment, *imperfect individuals* I_k, as defined in equation 4.3, respond to the new challenges from their environment, and continuously adapt to the

changed environment not only by means of improving themselves, i.e., *individual learning* (*IL*), but also by learning from others, i.e. *social learning* (*SL*).

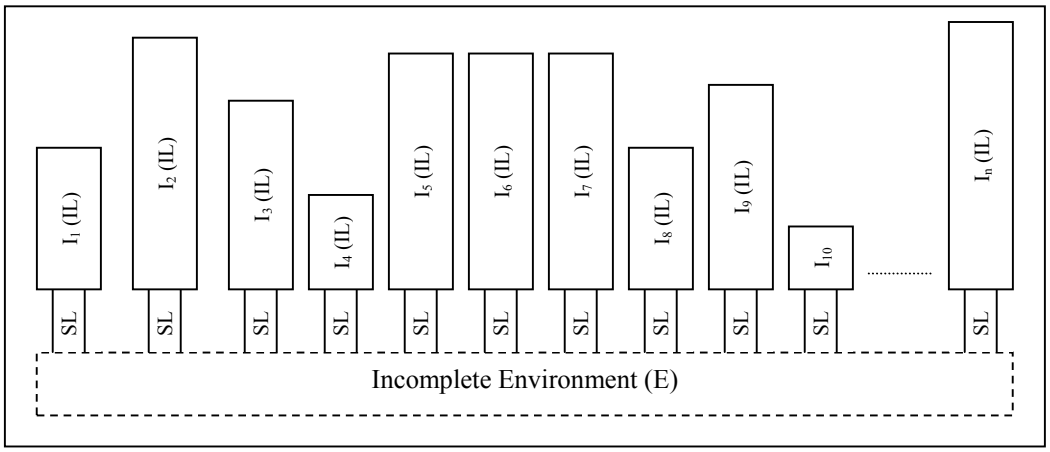


FIGURE 4.2 An Integrated Individual and Social Learning

In Figure 4.2, individuals (I_k) are represented as rectangles with different sizes indicating the problem space of each individual is of different dimensions and is non-static, depending on the information sets (Inf_k) used by individuals. The social learning mechanism works as the channel through which individuals will learn from each other so that information and knowledge, including new information from the changed environment, is disseminated among the evolutionary intelligent entities. There are similarities between the integrated individual and social learning paradigm and cultural algorithms (See section 2.7.7). For example, they both evolve at the individual level and they both maintain a knowledge repository. However, there are some fundamental differences between these two learning paradigms:

- **Incomplete environment** – ISP learning paradigm stresses on the importance of the awareness of an incomplete environment by an intelligent agent. An incomplete environment is always changing and evolving. Things

that are previously unobservable become observable. Things that are previously unknown become known. When Isaac Newton discovered the law of gravity, what should other people do with this new discovery? A dynamic environment does not only mean the possible changes in quantity, but also the potential changes in quality. The concept of an incomplete environment is not reflected in the design of cultural algorithms.

- **Imperfect individuals** – More importantly, the design of an imperfect evolutionary system is dependent on the development of imperfect individuals. It is only when an intelligent entity considers himself as imperfect, rather than a perfect end product, he has the opportunity and motivation to learn and adapt to the new challenges from the environment. Imperfect individuals also indicate that each individual can only understand his world imperfectly and each one has a (possible) different view of the world. In terms of problem solving, this implies that each individual in an imperfect evolutionary system could have a different problem dimension because they use different sets of information, and even a different problem domain because they are looking at the different side of the world. In cultural algorithm, there is no concept of imperfect individuals.
- **Imperfect world –** Compared with other machine learning paradigms in AI, imperfect evolutionary systems deal with learning problems from a different point of view. For example, imaging we are building a simulated brain (we do not mind if it works as the way the human brain does or not, since none of us know exactly how the brain works). If we employ the cultural learning

algorithm, the first thing we need to do is to create a population space using multi-population GA/GP. Each sub-population/individual in the MGA/MGP does a specialised task. The cultural algorithm then evolves the population space and updates the belief space. The question needed to ask is how many subpopulations we need create in order to cover all the functions performed by the brain? There is no answer for this. And how does the cultural learning system learn new things from its environment, by creating a new subpopulation with a different problem domain, or modify an existing subpopulation? Unfortunately, neither of these two mechanisms exists in cultural algorithms. On the other hand, if we employ the imperfect evolution methodology, we can start with just one individual which basically does nothing (a group of dummy neural networks). When there is a new environmental variable emerges in the environment, the imperfect evolutionary "mind" will perceive it and decide either to incorporate it into current networks in the ISP, or create a new individual for this piece of new information, or simply delete the new information. In this thesis, as a first attempt in developing imperfect evolutionary systems, we implement a simple imperfect evolutionary market, where each artificial stock trader represents (possibly) different problem domains depending on the information set the trader is using. We demonstrate how the ISP learning paradigm enables artificial traders to adapt to new environmental variables (v_j) (equation 4.1) emerging from the imperfect market in chapter 6.

4.3 Summary

In this chapter we described our concept of an imperfect evolutionary systems. The best example of an imperfect evolutionary system is human society. Every single person in society maximises his own utility with the resources available to him and prepares for new challenges from the environment. Likewise, an intelligent humanoid will face similar problems. An imperfect evolutionary system is a system where intelligent individuals optimise their own utilities with resources available whilst adapting themselves to the new challenges from an incomplete evolutionary environment. Note that the "individuals" in our definition can refer to any kind of intelligent objects. They can be traders in an evolutionary market, or researchers in a specific discipline, or intelligent robots living in a human society. They can also be the functional sections where the brain is treated as an imperfect evolutionary system.

In the next two chapters we use the stock market as a test bed for the practical implementation of an imperfect evolutionary system. The experimental studies are divided into two stages: In the first stage, we develop a simulated stock market where imperfect traders learn to trade stocks within a perfect complete environment, as shown in chapter 5; In the second stage, we implement the incomplete environment, and test if the integrated individual and social learning paradigm enables artificial traders to react to the new challenges from their environment and the generation of new knowledge.

Chapter 5

A Multi-Agent Based Simulated Stock Market

In chapter 2, we examined the problem of how machine intelligence responds to new challenges from a changing environment. We argued that the problem lies in the fact that the conventional AI is regarded as a perfect end product. Inspired by changing from a perfect paradigm to an imperfect paradigm in information economics, in chapter 3, we proposed an imperfect evolutionary system as an alternative approach in which intelligent entities respond to the new challenges from an evolving environment and constantly adapt to the changed environment through continuous learning. In chapter 4, we described our definition on imperfect evolutionary systems using two examples: stock market and game playing. In this thesis, we will use the stock market as a test bed for the implementation of imperfect evolutionary systems.

This chapter presents the first stage of our experimental work on imperfect evolutionary systems. A multi-agent based simulated stock market is developed in which imperfect stock traders, under partial understanding of the market, learn to trade stocks profitably by means of an integrated individual and social learning mechanism. The simulated stock market is described from the following four perspectives: the market structure; modelling artificial traders using evolutionary artificial neural networks; individual learning by means of evolutionary programming; and social

learning through a central memory. The experimental study includes three parts: Testing the simulated stock market on a single stock; Testing the simulated stock market on different types of stocks; Role of individual learning in the integrated individual and social learning paradigm. Discussions on the experimental results are provided.

This chapter has been disseminated via the following publications: Kendall and Yan (2003, 2003a).

5.1 The Model

In order to differentiate conventional artificial stock markets where the market dynamics are generated endogenously and the main task of the ASMs is to replicate major time series features observed from real markets, we termed our market model a simulated stock market where basic market scenarios are given extraneously and the major task of the simulated stock market is to evolve successful artificial stock traders. Our simulated stock market also differs the learning classifier system based market model developed in Schulenberg and Ross (2000) and Schulenberg and Ross (2001) (described in section 3.5.4) in following aspects:

- Our artificial stock traders are modelled using artificial neural networks so that no human expertise in stock trading is incorporated in the evolutionary learning process.
- Our artificial stock traders are modelled with imperfect intelligence in the sense that traders can only understand the market partially and have different beliefs based on different information sets for decision-making.

- Individual learning of artificial traders is integrated with social learning within the trading society, so that the problem of unobservable minds is avoided.

Figure 5.1 diagrammatically depicts our multi-agent based simulated stock market. Basic market features and a timeline of activities in the market is explained below:

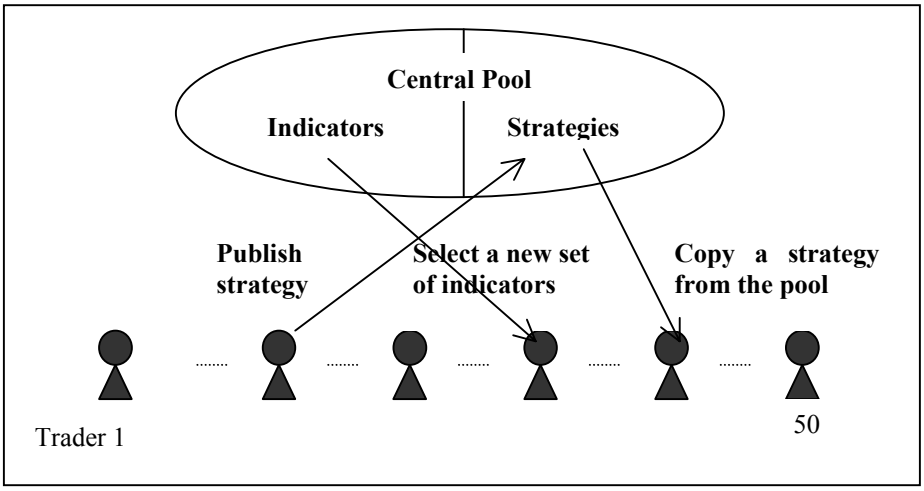


FIGURE 5.1 A Multi-agent Based Simulated Stock Market

1. Before trading starts, there are 50 active traders in the simulated stock market. There are 20 indicators and zero trading strategies in the central pool. The 20 available indicators are assigned an equal score of 1. Each trader selects a random set of indicators as inputs to their trading models.
2. With the set of indicators selected, each trader generates ten different artificial neural network models for forecasting. These ten models may have different network architectures, but they use the same set of indicators selected by the trader. The aim is for the trader to evolve models from these ten by the means of individual learning.

3. The time span of the experiment covers 3750 trading days, which is divided into 30 intervals. Each interval contains 125 days (6-month trading).
4. Each 125-day trading is sub-divided into intervals of 5 days. Each trader trades for 5 days, and then undertakes individual learning by means of EANNs with evolutionary programming.
5. At the end of each 125-day trading, social learning occurs and each trader is given the opportunity to decide whether to look for more successful strategies from the pool or whether to publish his successful strategies to the public.
6. After social learning has finished, the system enters the next 125 trading days and steps 4, 5 and 6 are repeated.
7. For every transaction, buy means using all the cash in the trader's account and sell means selling all his holdings. Both margin accounts, where traders could buy stocks on credit, and short selling, where traders could sell stocks she/he does not hold, and buy it back at a later time, are not allowed. Traders are asked to pay a trading fee of £10 for each transaction. Traders are also paid interest for any cash in their account, with an annual interest rate of 5%. Interest is calculated every half year. On the first trading day, all traders and investors are given a portfolio of £10,000 cash in bank and 1000 shares.
8. There are also one investor who sticks to the buy-and-hold strategy and one investor who saves all money in a bank, i.e. risk-free. Their performance will serve as benchmarks for the 50 active traders.

Besides the primitive historical stock prices, other financial data is also used to compose 20 popular technical indicators to be used by traders for implementing

trading models. This data includes: trading volume; intra-day high, intra-day low; FTSE-100 index; DJ Oil&Gas Index (UK), S&P 500 Index and DJ INDU AVERAGE. All data was acquired from Yahoo Finance (http://uk.finance.yahoo.com/) and DataStream financial data service (http://www.datastream.com). Table 5.1 lists the 20 technical indicators used for our experiment on a single stock, the British Petroleum (BP) share.

Technical Indicator	Description	Technical Indicator	Description
IND1	10 days Moving Average (price)	IND11	14 days Relative Strength Index
IND2	20 days Moving Average (price)	IND12	21 days Relative Strength Index
IND3	50 days Moving Average (price)	IND13	Stochastic Oscillators (K%)
IND4	200 days Moving Average (price)	IND14	Fast Stochastics (D%)
IND5	Closing Price (normalised)	IND15	Slow Stochastics (slow D%)
IND6	Rate of Change (price)	IND16	FTSE-100 Index rate of change
IND7	Oscillator (price)	IND17	Relative performance of FTSE 100
IND8	10 days bias (price)	IND18	S&P 500 Index rate of change
IND9	20 days volume Rate of Change	IND19	DJIA Index rate of change
IND10	10 days Relative Strength Index	IND20	DJ Oil&Gas Index (UK) rate of change

TABLE 5.1 20 technical indicators that are used as inputs into neural networks. All values are normalised into the range of [0,1].

5.2 Artificial Stock Traders

Artificial stock traders are modelled using multi-layer feed-forward perceptrons. The artificial neural networks are either 2-layer with no hidden layers or 3-layer with one hidden layer. Two different types of activation functions (sigmoid and tanh) are used. There is one single output node from the network. In order to facilitate the EANN learning process, the description file of each neural network is designed in a way such

that it can be used as a real-valued vector in the evolutionary programming algorithm, as shown in Figure 5.2:

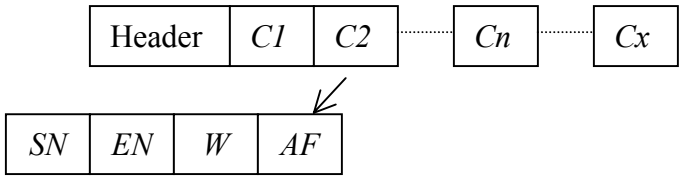


FIGURE 5.2 A real-value representation for evolutionary artificial neural networks. Each real-valued vector consists of a header and a number of connections. The header contains general information about the network architecture: starting input node, ending input node, starting hidden node, ending hidden node. Each connection, Cn, contains four components: starting node (SN), ending node (EN), weight (W), and activation function (AF). During the evolutionary process, both the connection weights and activation functions are mutated.

Besides the mutation of connection weights and activation functions, the architectures of the neural networks are also evolved by means of randomly adding a new hidden node or deleting a hidden node from the network, which will be described in detail in the individual learning process, in the next section. SN and EN are used to keep track of the order of connections in the neural network. As stated above, our artificial traders are modelled as imperfect intelligent individuals in the sense that they only can understand the market incompletely due to the complexity of the market. Each artificial trader observes the market from different aspects and uses different information sets for constructing trading models. For example, Table 5.2 below shows the number of indicators that are randomly chosen by trader no. 1 to trader no. 50 in their information sets on the first day of trading during the experiments with the BP share. As we can see, some artificial traders chose to use as few as one market indicator to predict the market trends, while others chose as many as 19 indicators to be included in their information sets.

T	NOI	T	NOI	T	NOI	T	NOI	T	NOI
1	18	11	14	21	16	31	18	41	2
2	3	12	18	22	5	32	6	42	2
3	8	13	14	23	14	33	1	43	9
4	2	14	10	24	2	34	4	44	17
5	14	15	1	25	7	35	2	45	19
6	3	16	12	26	12	36	6	46	12
7	8	17	2	27	14	37	9	47	17
8	6	18	18	28	12	38	16	48	16
9	15	19	17	29	7	39	10	49	7
10	16	20	2	30	19	40	15	50	14

TABLE 5.2 Information sets used by imperfect traders on the first trading day of BP share. T stands for trader. NOI stands for number of indicators.

5.3 Individual Learning

Individual learning occurs during every 125-day trading period. At the start of each period, each trader randomly decides on a set of indicators that will be used to build their prediction models. Each trader then randomly builds ten models based on their selected indicators. These ten models all use the same information sets as inputs, but with different network architectures. Each trader evolves his ten models in an attempt to discover better prediction models, using evolutionary programming described below.

At the beginning of a 125-day trading session, a trading model is chosen, using roulette wheel selection, for the next 5 days trading. The selection is based on the ten models' scores. At the end of each 5-day trading, trader *i* rate of profit (*ROP*) in percentage is calculated using equation 5.1:

$$ROP_i = \frac{W_t - W_{t-5}}{W_{t-5}} \times 100 \tag{5.1}$$

W_t is the value of trader's current assets (cash + shares). W_{t-5} is the value of trader's assets one week before. The score of selected model of trader i, m_i, is then updated using equation 5.2:

$$m_i = m_i + ROP_i \tag{5.2}$$

Based on the new updated scores, four models are selected as parents, using roulette wheel selection. Another four models, those with the lowest scores, are selected and will be replaced by four new offspring (produced by the four parents through mutation). The four parent models selected and the two remaining models will stay intact and continue to the next generation together with the four new offspring. Model scores for the four new offspring are given by adding a small variance to its parent model's score, as shown in equation 5.3 where σ_m is a random Gaussian number with a mean of zero and standard deviation of 0.1.

$$m_{offspring} = m_{parent} + \sigma_m \tag{5.3}$$

Since real-value representations are used for EANN, no crossover occurs during the reproduction process. The connection weights and activation functions of parent networks are randomly mutated to generate new offspring, while the number of hidden nodes are randomly increased or decreased, as summarised in Table 5.3.

```
Select a ANN to be eliminated;
Select a ANN to be mutated using roulette
   wheel selection;
Decide number of connections to be
   mutated, m;
i = 0;
While(i < m){
   Randomly select a connection;
   Weight = weight + Δw;
   i = i + 1;
   }
With 1/3 probability add a hidden node and
   randomly generate new connections;
With 1/3 probability delete a hidden node
   and delete all connections to it;
Replace the network to be eliminated with
   the mutated ANN;
```

TABLE 5.3 An individual learning algorithm.

The number of connections to be mutated, *m*, is a random integer between 0 and the total number of connections in the selected model. *Δw* is a random Gaussian number with a mean of zero and standard deviation of one. Besides the mutation of weights, we also evolve the structure of the network by allowing the probability of adding or deleting of hidden nodes from the neural networks. The mutation factor *Δw* is generated anew for each mutation. We did not evolve the mutation factor simultaneously as described in Table 2.6, and also our individual learning algorithm for EANN is much simple than the EPNet described in Figure 2.10. This is mainly due to the consideration on reducing computational time, and results from experiments showed that the individual learning algorithm is effective in discovering good trading models. Individual learning occurs for all traders at the end of every 5-day trading session, until they reach the end of every 125-day trading period, where social learning takes place.

5.4 Social Learning

After 25 weeks (125 days) of trading and individual learning, all traders enter a social learning stage. Our social learning mechanism is similar to the *school* mechanism described in Figure 3.4, it is, however, simpler. During social learning, all traders have the opportunities to see how other traders are performing. Traders may decide to learn from other traders, or publish their own successful trading strategies to the trading society. At this stage, each trader will carry out a self-assessment. The trader's decision in social learning depends on the result of this self-assessment. Based on the methods used in Chen and Yeh (2001a), our traders' assessments are calculated using equations 5.4, 5.5 and 5.6. First, the traders' rate of profit for the past six months is calculated (see equation 5.1), and then, the 50 traders are ranked from 0 to 49 according to their six-month *ROP*:

$$S^{i}_{peer} = 1 - \frac{R_i}{49} \tag{5.4}$$

R_i is the rank of trader *i* in the range of [0,49] (0 indicates highest rank with greatest *ROP*). Equation 5.4 gives each trader a score in terms of peer pressure from other traders. In other words, this score shows trader *i*'s performance compared to other traders.

$$S^{i}_{self} = \frac{ROP - ROP'}{100} \tag{5.5}$$

ROP here refers to the rate of profit for the current six months trading. ROP' refers to the rate of profit for the previous six months. Equation 5.5 gives the trader's score in terms of his own performance in the past six months compared to the previous six

months. Finally, these two types of performance are composed into equation 5.6, which gives the overall assessment for trader *i* as following:

$$assessment_i = S^i_{peer} + \frac{1}{1+e^{(1-S^i_{self})}} \tag{5.6}$$

The final assessments for 50 traders are then normalised into the range of [0,1]. Depending on their assessments, social learning take place as shown in Table 5.4:

1. If a trader's assessment is 1, and the trader is not using a strategy drawn from the pool, then publish the strategy into the central pool. Go into the next six months trading using the same strategy.
2. If a trader's assessment is 1, and the trader is using a strategy copied from the pool, do not publish it again, but update this strategy's score in the pool using their six-month ROP. Go into next six months trading using the same strategy.
3. If a trader's assessment is less than 0.9, the trader has 0.5 probability of copying a strategy from pool, which means the trader will discard whatever model he is using, and select a better trading strategy from the pool using roulette selection, and go into the next six months trading with this copied strategy. Or, with 0.5 probability, the trader will decide to discard whatever strategy he is using, and select another set of indicators as inputs, build 10 new models and go into next six months trading with these 10 new models.
4. If assessment is between 1 and 0.9, the trader is satisfied with his performance in past six months and continues using that strategy.

TABLE 5.4 A social learning algorithm under perfect environments.

Traders will also update scores of indicators they have used in the central pool based on their performance in the current six months using equation 5.7 below.

$$I^n_i = I^n_i + ROP \tag{5.7}$$

where *n* is the n^{th} indicator from the information set used by trader *i* in the current six-month trading. *ROP* here refers to the rate of profit of the trader *i* in the current six months trading. This process mimics the evolution of public opinions on the utilities of the 20 technical indicators in the market.

5.5 Simulation On A Single Stock and Discussions

Shares of BP PLC from the London Stock Market are selected to be traded in the simulated stock market. Figure 5.3 shows BP's historical price chart.

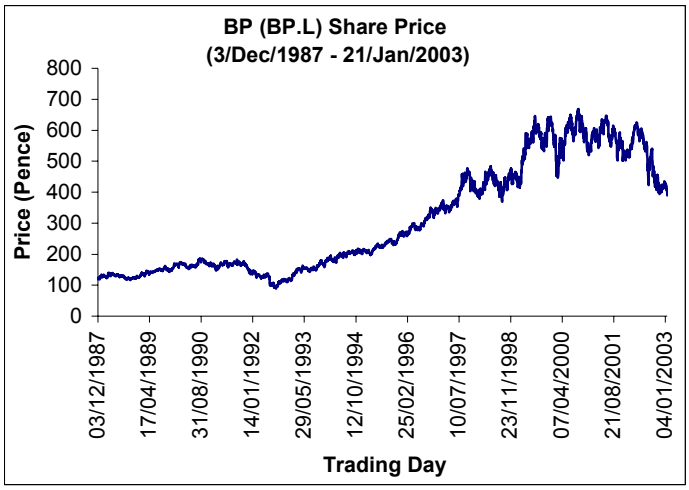


FIGURE 5.3 BP PLC (BP.L) price chart

The main consideration for choosing BP as the stock to be tested in our simulated stock market is that its price history shows an upturn in the overall trend, see Figure 5.3. A buy-and-hold strategy will, obviously, bring the investor a positive return in the case of BP. In order to beat the buy and hold strategy, an active trader must be able to predict the market trend and seize trading opportunities during the ups and downs of the market. In other words, if our artificial traders could achieve a higher return than the classical buy-and-hold strategy, that will prove the integrated individual and social learning algorithm is an effective learning paradigm for artificial traders in the simulated stock market, which will give us a solid foundation for continuing to the development of the imperfect evolutionary market in the next step of the experiments.

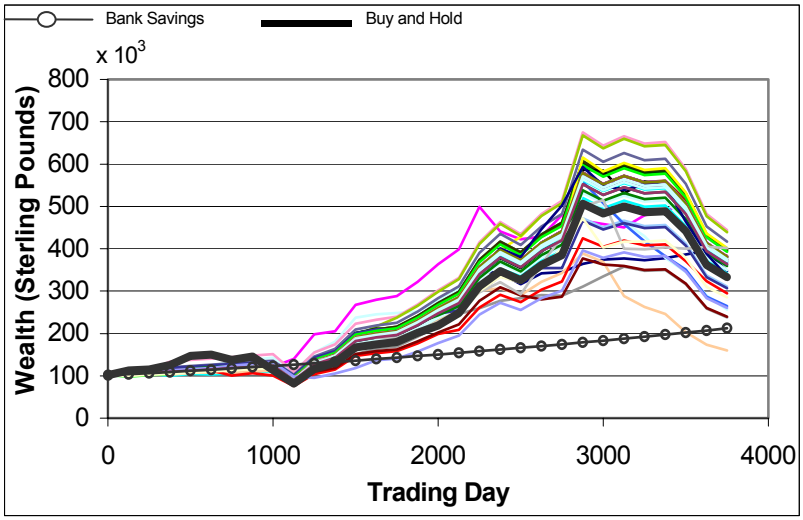


FIGURE 5.4 Artificial stock traders' performance on BP share

Ten simulations are run on the BP stock. The results from the best run are demonstrated in Figure 5.4 and Table 5.5, which show that successful trading strategies have been evolved by artificial traders in the simulated stock market. Figure 5.4 demonstrates the growth of wealth of the 50 artificial traders during the 3750 days trading period on BP. The thick black line indicates the growth of wealth of the investor with buy-and-hold strategy. The thin line at the bottom of the diagram with o markers indicates the growth of wealth of the investor who saved his money in bank for 15 years. From Figure 5.4 we see the majority of the artificial traders (represented by the lines above the thick black line) were able to learn to predict the trend in the stock, i.e. start to buy in stocks when the share is going up and start to sell it when it is going down. The more accurately the trader was able to control the timing of buying and selling, the faster they accumulated wealth. Table 3 gives the statistic results on the 50 artificial traders. It shows 40 out of the 50 artificial traders beat the buy and

hold strategy. On average, 50 active traders outperform the benchmark buy and hold strategy by 25.84%.

Description	Result
Return from bank savings	109.76%
Return from buy-and-hold strategy	228.69%
Average return from the 50 traders	254.53%
Maximum cumulative return among the 50 traders	338.53%
Minimum cumulative return among the 50 traders	57.7%
No. of traders who outperform savings	49
No. of traders who outperform buy and hold	40

TABLE 5.5 50 artificial traders' performance compared with buy and hold strategy and bank savings. All returns are calculated as the difference between traders' wealth on day 3750 and trader's original wealth divided by the original wealth.

A cumulative return, i.e., not annualised, is calculated using equation 5.8.

$$Cumulative\ Return = \frac{TotalAsset(start) - TotalAsset(end)}{TotalAsset(end)} \times 100\% \tag{5.8}$$

In order to see the learning dynamics of 50 artificial agents more clearly, we selected 3 traders from the 50 traders and depict them in Figure 5.5. Trader 14 is the best performer who achieved a return of 338.53% on his original wealth (note this is a cumulated return over a period of 15-year investing). Overall, trader 14 was able to predict the trend in the price of the stock fairly well. The transaction records of trader 14 shows this trader, in fact, learned a strategy from the central pool in the early days, and kept it until the last trading day. Some other traders also used the same strategy copied from the pool, but trader 14 refined this strategy constantly through his own individual learning, which results in his outstanding performance compared to others. Trader 16 is the worst performer whose wealth line finally runs below the bank saving line. This trader, showed by his transaction records, did not consult the central pool for

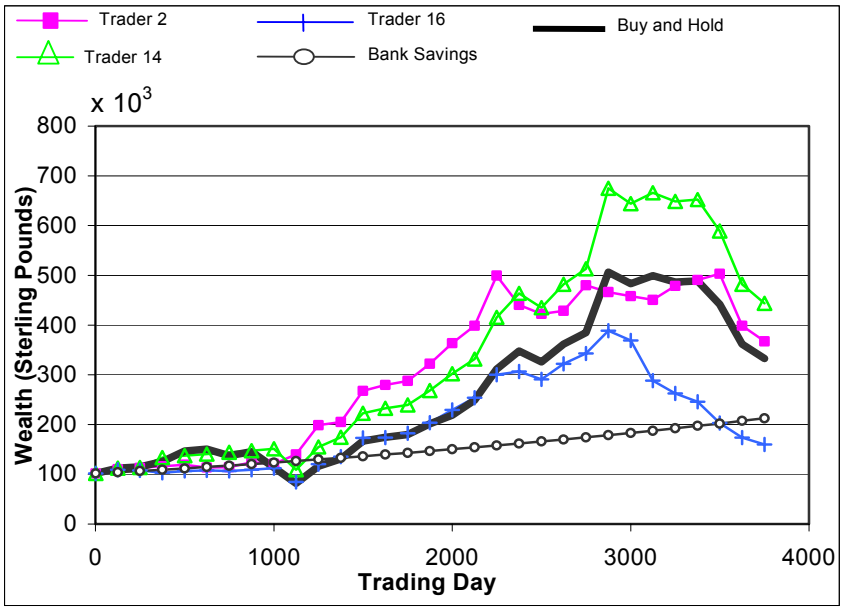


FIGURE 5.5 Artificial traders' learning dynamics (BP.L).

other traders' strategies, but relied on his own individual learning. He basically followed the buy-and-hold strategy in the early stage of the trading period. The trader's own individual learning did not help him too much. Around day 2200, he made a mistake by selling the stock when the price was still going up. When the stock price dropped dramatically around day 3000, this trader's strategy completely failed. On the contrary, trader 2, shown by his transaction records, constantly searched for good strategies from the pool, and tried out different strategies during the different stage of the trading period. Before day 2300, trader 2 used a strategy learned from central pool, which worked quite well during the upturn period. However, during the downturn after day 2300, this strategy didn't work well. Trader 2 went on and tried different strategies from pool, finally, still managing to outperform the buy-and-hold strategy.

Day	Successful Trade (Daily)	Successful Trade (Weekly)	Successful Trade (Monthly)	Social Learning
2000	72.8%	72.0%	70.4%	Satisfied, Keep on going
2125	76.8%	72.8%	73.6%	Satisfied, Keep on going
2250	84.0%	80.0%	79.2%	Satisfied, Keep on going
2375	80.0%	69.6%	55.2%	Satisfied, Keep on going
2500	26.4%	12.8%	16.0%	Copied a strategy from the central pool
2625	70.4%	72.0%	78.4%	Satisfied, Keep on going
2750	74.4%	68.0%	82.4%	Satisfied, Keep on going
2875	88.0%	74.4%	67.2%	Satisfied, Keep on going
3000	60.8%	41.6%	40.0%	Satisfied, Keep on going
3125	61.6%	50.4%	40.8%	Satisfied, Keep on going
3250	52.8%	41.6%	35.2%	Satisfied, Keep on going
3375	48.0%	39.2%	32.8%	Satisfied, Keep on going
3500	46.4%	40.0%	33.6%	Satisfied, Keep on going

Table 5.6 Trader 14's transaction statistics (Day 1875 to Day 3500, BP.L)

Tables 5.6 to 5.8 demonstrate the transaction statistics of traders 14, 16, and 2 in relation to their social learning behaviours. We evaluate the successfulness or correctness of each trading decision in terms of three different investment horizons, i.e. daily, weekly and monthly. There are three different trading decisions that a trader could take on each trading day: buy, sell or hold. Assume today's stock price is £1.25, and the price stays same on the next trading day. After one week, the stock price rises up to £1.40. After one month, the stock price in fact falls back to £1.00. In this case, a decision of buying in the stock will make profits in short term, but causes losses in the long-term sense. Equation 5.9 below is used to evaluate the successfulness or correctness of a trading decision on the present day in terms of its profitability in a day, a week or a month time.

Day	Successful Trade (Daily)	Successful Trade (Weekly)	Successful Trade (Monthly)	Social Learning
2000	71.2%	63.2%	76.8%	Satisfied, keep on going
2125	80.8%	74.4%	88.8%	Satisfied, keep on going
2250	76.0%	69.6%	61.6%	Satisfied, keep on going
2375	60.8%	56.0%	46.4%	Copied a strategy from the central pool
2500	61.6%	59.2%	48.0%	Satisfied, keep on going
2625	56.0%	53.6%	60.8%	Satisfied, keep on going
2750	71.2%	72.0%	56.8%	Satisfied, keep on going
2875	60.0%	61.6%	58.4%	Satisfied, keep on going
3000	52.8%	52.0%	56.0%	Copied a strategy from the central pool
3125	47.2%	54.4%	60.0%	Select a new set of indicators
3250	68.0%	55.2%	62.4%	Satisfied, keep on going
3375	74.4%	69.6%	72.0%	Satisfied, keep on going
3500	82.4%	71.2%	61.6%	Publish the strategy to the central pool

Table 5.7 Trader 2's transaction statistics (Day 1875 to Day 3500, BP.L).

$$D_{d,h} = \begin{cases} f(p_{d,h} - p_{d,0}), & d = buy \\ f(p_{d,h} - p_{d,0}), & d = hold_s \\ f(p_{d,0} - p_{d,h}), & d = hold_c \\ f(p_{d,0} - p_{d,h}), & d = sell \end{cases} \tag{5.9}$$

Where

$$f(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases} \tag{5.10}$$

And

$$h = \begin{cases} 1 & if\ investmetn\ horizon\ is\ one\ day \\ 5 & if\ investment\ horizon\ is\ a\ week \\ 20 & if\ investment\ horizon\ is\ a\ month \end{cases} \tag{5.11}$$

In equation 5.9, $D_{d,h}$ indicates the successfulness of a trading transaction d. The options for d include buy transactions, sell transactions, holding the stocks on hand

Day	Successful Trade (Daily)	Successful Trade (Weekly)	Successful Trade (Monthly)	Social Learning
2000	61.6%	58.4%	64.0%	Satisfied, keep on going
2125	64.0%	60.0%	66.4%	Satisfied, keep on going
2250	72.0%	57.6%	53.6%	Satisfied, keep on going
2375	48.8%	46.4%	41.6%	Satisfied, keep on going
2500	44.0%	42.4%	48.0%	Select a new set of indicators
2625	52.8%	48.0%	48.8%	Satisfied, keep on going
2750	57.6%	50.4%	55.2%	Satisfied, keep on going
2875	67.2%	52.0%	48.0%	Satisfied, keep on going
3000	52.8%	48.0%	41.6%	Satisfied, keep on going
3125	29.6%	22.4%	17.6%	Select a new set of indicators
3250	25.6%	20.8%	16.8%	Satisfied, keep on going
3375	24.0%	20.0%	14.4%	Select a new set of indicators
3500	16.0%	13.6%	12.0%	Select a new set of indicators

Table 5.8 Trader 16's transaction statistics (Day 1875 to Day 3500, BP.L).

($hold_s$) or holding the cash in a bank account ($hold_c$). A positive $D_{d,h}$ indicates a successful trade with regards to a specific investment horizon (h), namely, daily or 1 day, weekly or 5 days, monthly or 20 days. $p_{d,0}$ and $p_{d,h}$ in equation 5.9 refer to the stock price on the present day of trading and the stock price after h days respectively. Function f decides the value of $D_{d,h}$ as shown in equation 5.10. For example, when the current stock price is £1.25, and the price rises up to £1.40 in a week time, $D_{buy,5}$ has a value of +1. But if the stock price drops to level of £1.00 after a month time, $D_{buy,20}$ will have the value of –1. The statistical results on trading decisions from Tables 5.6 to 5.8 are then calculated using equation 5.12 below,

$$D\% = \frac{\sum_{n=1}^{125} D^{n}_{d,h} \left| D^{n}_{d,h} > 0 \right.}{125} \tag{5.12}$$

Simulation	Average Return(%)	Maximum Return(%)	Traders (> Bank Savings)	Traders (>Buy & Hold)
1	260.33	300.59	42	39
2	232.12	287.75	39	35
3	254.53	338.53	49	40
4	250.12	310.34	45	36
5	212.09	256.88	38	30
6	239.1	303.77	41	35
7	249.66	320.06	46	36
8	205.13	224.24	32	28
9	229.89	310.76	40	31
10	250.06	322.11	48	40
AVEG	238.30	297.50	42.00	35.00
SD	18.42	33.83	5.16	4.19

TABLE 5.9 Average results from ten simulations run on the BP stock. AVEG refer to the average results under each of the four criteria. SD refers to standard deviation.

Equation 5.12 evaluates the percentage of successful trades during a 125-day trading period, in regard to a specific investment horizon *h*. We choose the 125 days as the time interval because social learning occurs after every 125 days' trading. In such a way, we are able to examine the effects of social learning during the evolution of artificial traders.

As shown in Table 5.6, trader 14 chose to learn a strategy from the central pool through social learning during the small downturn of the market around day 2500. This strategy worked particularly well during the long upturn period afterwards. During this upturn period of the market, trader 14 rapidly accumulated a large amount of wealth through frequent short-term trading. The trading strategy learned by trader 14 from the central pool through social learning is particularly good at seizing short-term trading opportunities during the upturn of the market, which largely counts for the outstanding performance of trader 14. Although this learned strategy through social learning did not adjust to the changed market condition well (by means of trader

14's own individual learning) during the downturn of market after day 3250, its ability to seize short-term trading opportunities still helped trader 14 to outperform other traders in the market and the returns from risk-free bank savings. Trader 14 therefore chose to keep the strategy learned through social learning.

From Table 5.7, we observed that trader 2 tried out different trading strategies from the central pool when the market conditions changed during the social learning phases. At the end this trader decided to select a new set of technical indicators as his choice during a social learning period, and built a new trading strategy by himself that worked particular well when the market made a downturn. Trader 2 also published his new strategy to the market as a strategy that works well during the downturn of the market. On the contrary, due to the randomness in the social learning algorithm (see Table 5.4), trader 16 from Table 5.8 chose to try out different market indicators and mainly relied on its own individual learning. Tables 5.6 to 5.8 demonstrate the important effects of social learning on the artificial traders' learning behaviours and learning abilities. We also demonstrate the statistical results from the total ten simulations run on the BP stock in Table 5.9.

As a summary, compared to the traders with specifically pre-defined types studied in Schulenberg and Ross (2000) and Schulenberg and Ross (2001), the 50 artificial traders here were generated completely randomly without defining what types of rationality or belief they should have. Each of them only understands the market imperfectly and uses different sets of information from the market for making trading decisions. The results from the simulation on BP shows that the artificial agents

learned successfully to trade stocks, and outperformed the baseline buy-and-hold strategy. Moreover, the 50 randomly generated artificial traders demonstrated rich dynamic learning behaviours that are very similar to traders in real world markets.

5.6 Simulations On Different Types of Stocks and Discussions

After experimenting with the simulated stock market on a single stock (BP. L) where the artificial agents demonstrated dynamic behaviours and strong learning abilities, we further tested the evolutionary market on another five different types of stocks with different profiles, i.e. different fundamentals of the tested companies and different price patterns presented in their stock history. Table 5.10 below lists the profiles of the five selected different types of stocks from the London Stock Exchange (LSE).

Company Name	Symbol	Sector
British Airways PLC	BAY.L	Transport
BT Group	BT.L	Telecommunications
Kingfisher	KGF.L	Retailers
Barclays	BARC.L	Banks
GLAXOSMITHKLINE	GSK.L	Pharmaceuticals

TABLE 5.10 Five different types of stocks selected from the London Stock Exchange.

Each of the five stocks above is traded in the simulated stock market for a period of 3750 trading days (usually starts from November 1987 and ends in January 2003). As discussed previously, each trader only has a partial understanding of the market by selecting a different information sets for implementing trading strategies. Ten simulations are run on each stock. The results from the best runs are depicted in Figures 5.6 to 5.10 with the wealth growth lines of the 50 artificial traders for simulations on each stock.

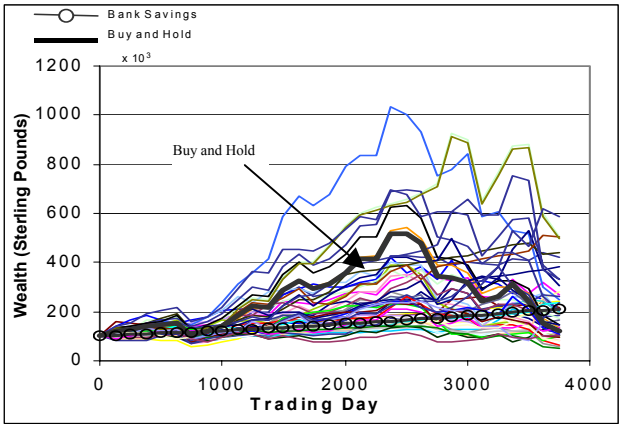


FIGURE 5.6 Simulation of Trading on BAY.L

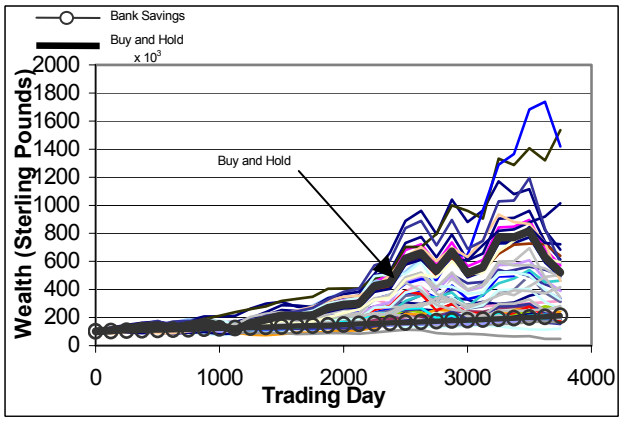


FIGURE 5.9 Simulation of Trading on BARC.L

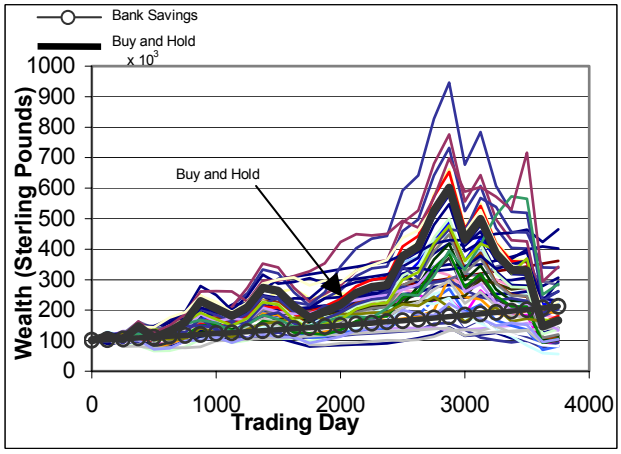


FIGURE 5.8 Simulation of Trading on KGF.L

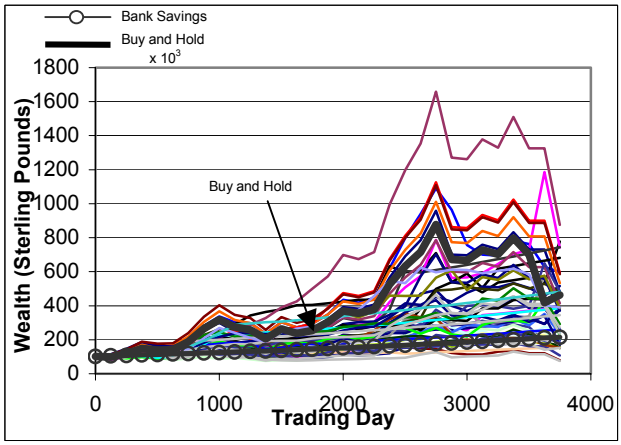


FIGURE 5.10 Simulation of Trading on GSK.L

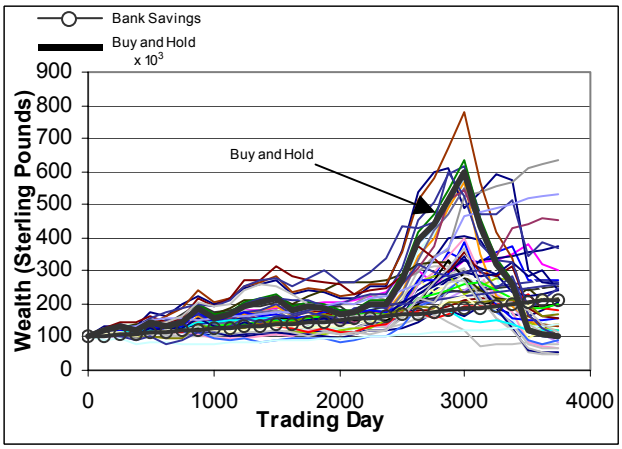


FIGURE 5.7 Simulation of Trading on BT.L

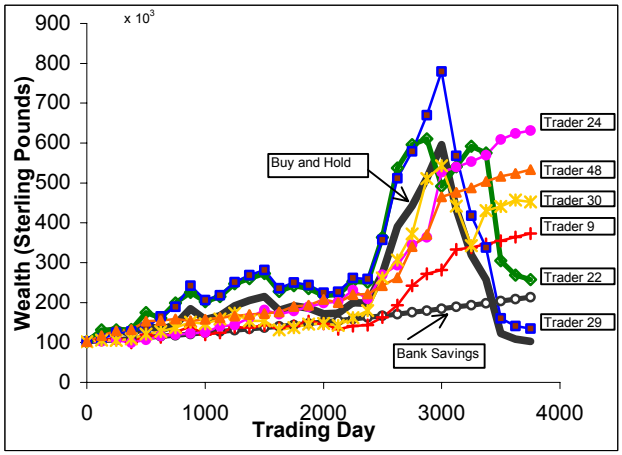


FIGURE 5.11 Different types of traders and trading strategies developed during the simulation on BT share

Description	BAY.L	BT.L	KGF.L	GSK.L	BARC.L
Outperform Buy & Hold (out of 50 traders)	36	36	28	15 (30%)	11 (22%)
Outperform Bank Savings (out of 50 traders)	18 (36%)	16 (32%)	18 (36%)	33	35
Cumulative Total Return (Buy & Hold)	18.26%	0.97%	65.03%	351.26%	417.97%
Cumulative Total Return (Bank Savings)	109.76%	109.76%	109.76%	109.76%	109.76%
Maximum Cumulative Total Return (Best Trader)	482.86%	519.63%	359.85%	753.53%	1424%

TABLE 5.11 Artificial agents' performance compared with benchmarks, B&H and bank saving on multiple stocks.

The results will be discussed in two different aspects: artificial traders' performance and behaviours, and the importance of social learning when agents only understand their environment incompletely. Figures 5.6 to 5.10, except for the buy and hold lines and the bank savings lines, show the performance of the 50 artificial traders in each simulation respectively. From the buy and hold lines in each figure which essentially indicate the historical price movement of the particular share, we can see that each of the five stocks, ignoring their different background fundamentals, demonstrates different price patterns. To explain the evolution of artificial traders more clearly, we use the results from the simulation on BT's share in Figure 5.7 as an example, and select six typical evolved traders to illustrate the adaptive learning of the agents and the different types of traders and trading strategies developed, as depicted in Figure 5.11. In Figure 5.11, trader 29, 22 and 30 can be described as 'aggressive traders' who followed the trend of the stock price closely and accumulated their wealth in frequent trading. These types of strategies worked well during the upturn of the market, however, during the downturn of the market, the adaptability of the agents becomes essential to the success of the trader. While trader 29 failed to adapt to the changed market, trader 22 and 30 managed to adapt their strategy to the changing market and ultimately beat the bank savings.

On the other hand, trader 24, 48 and 9 can be described as 'passive traders' who are usually more cautious about the market. These traders had less frequency in trading compared to the aggressive traders, thus usually having lower growth lines than the aggressive traders during the upturn of the market. However, once the market changed

to downturn, these conservative traders usually adapted themselves to the new environment much faster, and transferred their assets from the stock market back to the less risky banks. We can see during the simulation, the traders are evolving, their trading strategies are also evolving and adapting to the new environment. We conclude that the adaptability of the agents is essential in a volatile environment such as the stock market, rather than just good timing of trading.

Referring to figures 5.6 to 5.11, we observe similar evolutionary processes of the traders in each of the simulations: different types of traders and different type of good trading strategies were developed, regardless of the different background of the tested stocks. In the cases of BAY.L, BT.L, and KGF.L share, where the stock prices basically follow a pattern of an upturn followed by a sharp downturn, bank savings will perform well. On the contrary, in the cases of BARC.L and GSK.L share, buy and hold strategy performs well. Thus, we compared the performance of the artificial traders with bank saving for BAY.L, BT.L and KGF.L share, and with buy and hold strategy for GSK.L and BARC.L share and present the results in Table 5.11. Table 5.11 shows, during the five simulations, 22% to 36% of the artificial traders were able to learn to develop good trading strategies that beat the market or risk-free investments. This percentage is lower than the result from the experiment on the single stock BP (see Table 5.5). This can be explained by the fact that the BP share presents a relatively smoother up trend (see Figure 5.3) where no dramatic drops in price happened, which means agents have more time to adapt since the market changes gradually. In Figure 5.6 to 5.11, each of the stocks demonstrates more volatile

Simulation (BA)	Average Return	Maximum Return	Traders (> Savings)	Simulation (BT)	Average Return	Maximum Return	Traders (>Savings)
1	361.12%	441.42%	15	1	411.31%	499.96%	15
2	350.03%	410.23%	14	2	432.32%	519.63%	16
3	381.19%	450.06%	16	3	402.13%	487.37%	13
4	363.91%	461.11%	16	4	379.89%	465.63%	11
5	333.47%	397.70%	12	5	440.01%	506.14%	16
6	390.11%	482.86%	18	6	408.67%	479.98%	12
7	356.77%	455.32%	16	7	386.53%	455.93%	10
8	376.43%	473.21%	17	8	395.58%	471.29%	11
9	303.46%	384.56%	10	9	389.13%	469.77%	12
10	366.79%	459.87%	15	10	429.13%	510.13%	15
AVEG	358.33%	441.63%	14.90	AVEG	407.47%	486.58%	13.10
SD	0.251	0.331	2.38	SD	0.207	0.215	2.23

TABLE 5.12 Average results from ten simulations run on the BA stock and the BT stock. AVEG refer to the average results under each of the four criteria. SD refers to standard deviation.

Simulation (KGF)	Average Return	Maximum Return	Traders (> Savings)	Simulation (GSK)	Average Return	Maximum Return	Traders (>Hold)
1	299.71%	359.85%	18	1	589.91%	711.56%	12
2	281.34%	336.73%	16	2	581.12%	708.55%	11
3	276.75%	340.05%	18	3	597.77%	724.25%	12
4	235.51%	311.27%	14	4	573.34%	668.98%	9
5	259.99%	320.15%	15	5	613.69%	734.59%	13
6	270.01%	337.61%	17	6	592.27%	713.82%	11
7	200.05%	306.89%	12	7	636.72%	739.88%	14
8	214.83%	315.44%	14	8	659.13%	753.53%	15
9	263.72%	329.91%	16	9	580.17%	705.25%	10
10	272.13%	328.73%	17	10	582.35%	700.09%	10
AVEG	257.40%	328.66%	15.70	AVEG	600.65%	716.05%	11.70
SD	0.312	0.159	1.95	SD	0.278	0.237	1.89

TABLE 5.13 Average results from ten simulations run on the KGF stock and the GSK stock. AVEG refer to the average results under each of the four criteria. SD refers to standard deviation.

movements in their price history, generally accompanied with sharp rise and drops in prices. Nevertheless, the individual and social learning paradigm still enables a reasonable number of its participants to survive in the market. It is not a market failure to see failed artificial traders in our simulated stock market. On the contrary, it is exactly what happens in real world. Table 5.11 also shows the performance of the best trader from each simulation in the comparison of the benchmark buy and hold strategy and the risk-free bank saving. In addition, in tables 5.12 to 5.14, we demonstrate the

Simulation (BARC)	Average Return(%)	Maximum Return(%)	Traders (>Buy & Hold)
1	852.41%	955.87%	9
2	901.69%	1011.01%	10
3	811.17%	921.55%	7
4	956.18%	1424.01%	11
5	869.73%	1212.33%	10
6	861.99%	1058.71%	9
7	823.54%	982.36%	8
8	899.13%	1030.32%	10
9	918.55%	1161.58%	10
10	835.28%	997.36%	9
AVEG	872.97%	1075.51%	9.30
SD	0.46	1.52	1.16

TABLE 5.14 Average results from ten simulations run on the Barclays stock. AVEG refer to the average results under each of the four criteria. SD refers to standard deviation.

statistical results of the ten simulations run on each of the five stocks. From tables 5.12 to 5.14, we can see that the standard deviations of the values under each criterion reside in reasonable ranges. However, we want to stress that, due to the randomness in our individual and social learning algorithm, we expect to see significant differences on each run of the experiments on each stock, with regards to both the performance of artificial traders and the learning behaviours of artificial agents.

In order to examine the learning of agents in the simulated stock market more closely, we select a single trader from the experiment on the Barclays stock (BARC.L, as shown in Figure 5.9). The selected trader is trader 28 whose overall performance is plotted in Figure 5.12 below.

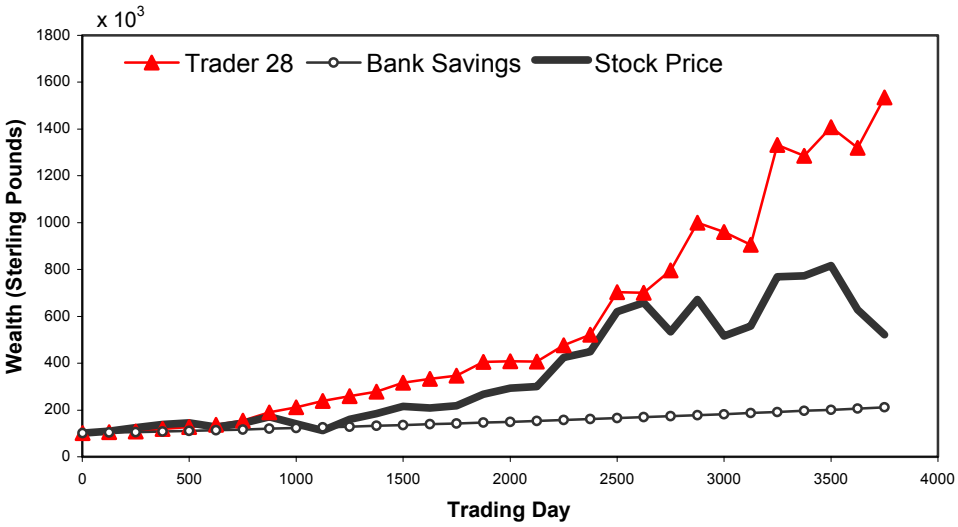


FIGURE 5.12 Trader 28 overall performance (BARC.L)

As shown in Figure 5.12, trader 28 can generally predict the trend in price change and thus trade profitably. The detailed transaction history of trader 28 on BARC.L is demonstrated in Figures 5.13 and 5.14 as price/volume charts.

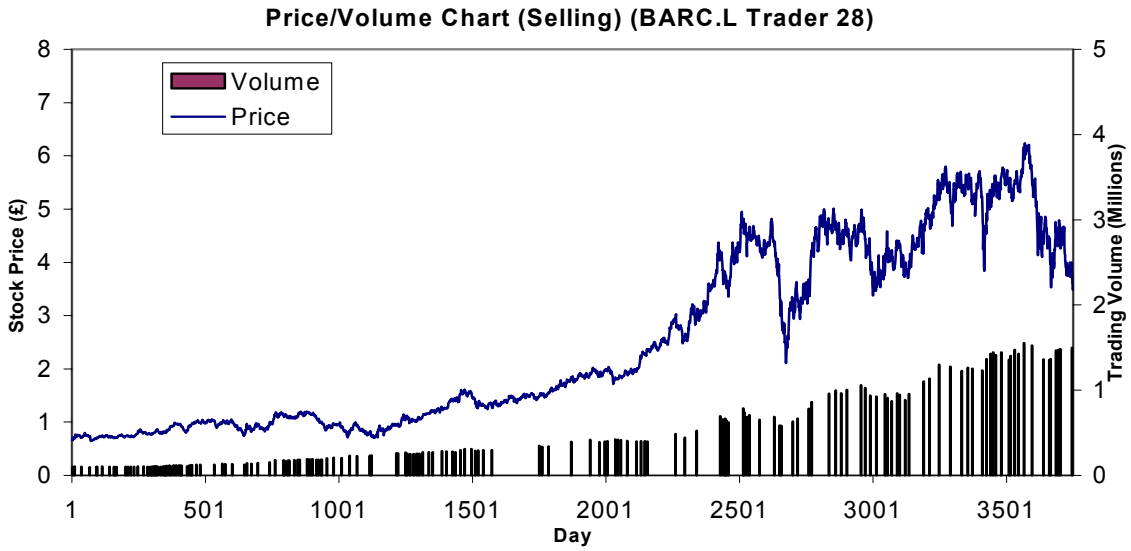


Figure 5.13 Price/Volume chart for selling transactions of trader 28 on BARC.L.

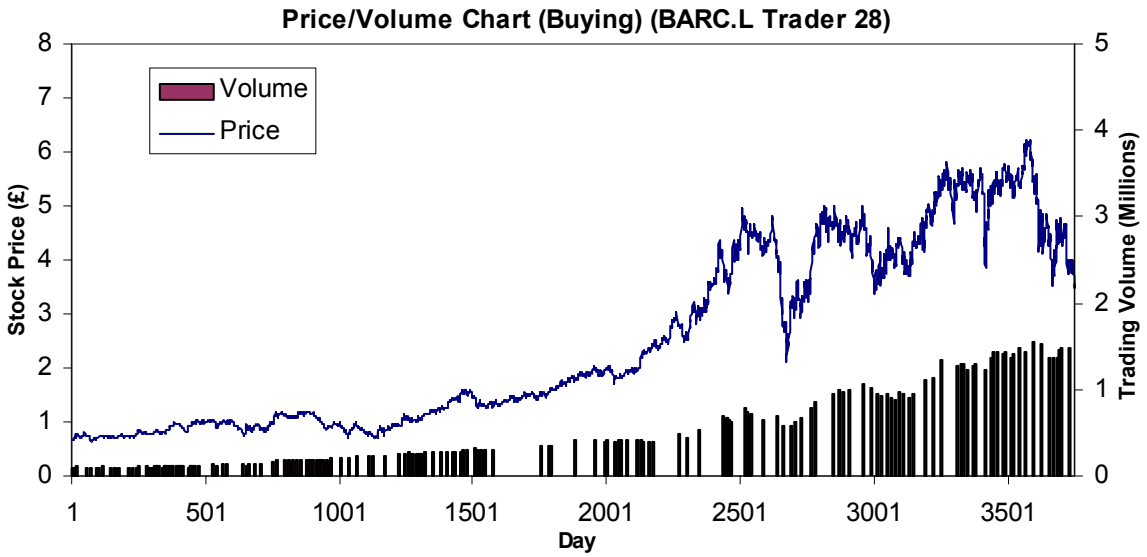


Figure 5.14 Price/Volume chart for buying transactions of trader 28 on BARC.L.

In Figure 5.13 and 5.14, there are two periods where the frequencies of trading are higher than other trading days. The first period occurs around day 1500. During this period, especially at the start of the experiment, neural networks are randomly

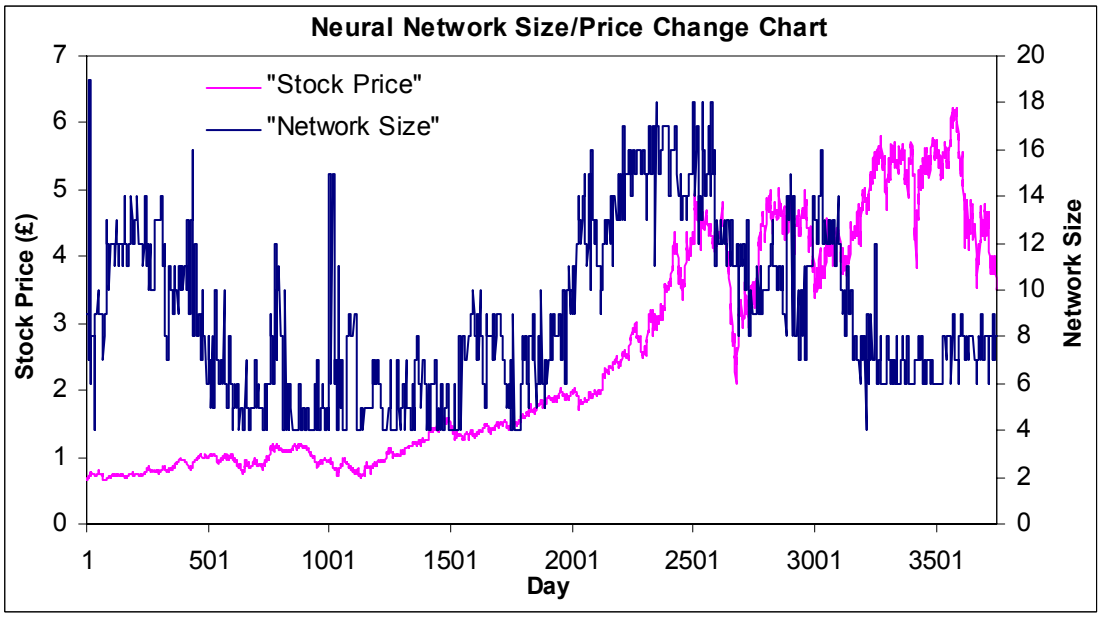


FIGURE 5.15 The change of size in artificial neural network models (BARC.L Trader 28).

generated as trading models, which may or may not be able to make sensible predictions on the market trend. Therefore, the artificial trader frequently experiments

with different trading models where we can find a number of "irrational" transactions. The artificial trader learns through a process of trial and error by means of intensive individual learning and social learning. The second period occurs approximately after day 2500 where the price pattern presents an intensive volatility, compared to the smooth rising period between day 1500 and day 2500. The strategies learned between day 1500 and 2500 seem not work well under the new market conditions after day 2500. The trader starts to look for new strategies by both searching the central pool and refining models through individual learning. Figure 5.15 demonstrates the learning of the trader from another aspect by examining the change in the size of neural networks used throughout the trading. In Figure 5.15, the network size is measured by the number of hidden nodes used in a network model. We can see the size of neural networks increases gradually, but changes restlessly after day 2500. At the end of the trading, trader 28 prefers to choose networks with small sizes. One possible explanations for this is that the output from a neural network with less hidden nodes more directly reflects the values of the inputs, i.e., the technical indicators, which is more useful when making short-term trading decisions.

Form another point of view; if we view the simulated stock market as a hypothesis space of trading strategies, the evolutionary process of the simulation is essentially the process of artificial agents searching for optimal trading strategies under certain market scenarios. Within this search space, due to the imperfectness of artificial traders, there are different regions where each region represents trading strategies using a certain information set from the environment. Whether the traders are able to

escape from one region and move into another region to search for better information and solutions becomes critical in terms of the adaptability of individual traders and the quality of the whole trading society. The social learning mechanism plays an essential role inside the integrated individual and social learning paradigm in the sense that it acts as a bridge through which individuals escape from a local region learning and search in the global space for the better information and knowledge. Without social learning, individuals will not be able to learn from each other. The evolution and the learning will become much harder and slower. Social learning is also vital in the sense that it is the only way that new information can be absorbed by a society and new knowledge can be created and disseminated within the society, which will be experimented and discussed under imperfect environments in the coming chapter.

5.7 Summary

In this chapter, we have developed a multi-agent based simulated stock market model in which imperfect artificial stock traders co-evolve and learn to trade stocks profitably. The evolutionary market model is built upon an integrated individual learning and social learning paradigm, where the individual learning mechanism is implemented by means of EANNs with evolutionary programming and the social learning mechanism is implemented through a central reservoir of information and knowledge. The simulated stock market is tested on a single stock and various other different types of stocks. The results from experiments demonstrate that the integrated individual and social learning algorithm is an effective and efficient learning paradigm

for modelling continuous learning in incomplete environments. Under the ISP learning paradigm, a reasonable percentage of artificial stock traders in the simulated stock market were able to learn to discover successful trading strategies, and demonstrated rich learning behaviours that closely mimic real world traders.

The artificial stock traders in our simulated stock market are modelled as imperfect individuals in the sense that every trader only can understand the market partially due to the complexity of the stock markets and uses different information sets for their decision-making. Imperfect artificial traders improve their performance by means of individual learning and learn from each other through social learning. However, the simulated stock market described in this chapter is still not a true imperfect evolutionary market in the sense that the hypothesis space of artificial traders is a static problem space with no new challenges. In next chapter, we will implement the imperfect evolutionary market by introducing an incomplete problem space for artificial stock traders which constantly changes, and demonstrate how the integrated individual and social learning algorithm enables imperfect individuals to react to new challenges from their environments.

Chapter 6

Imperfect Evolutionary Market

This chapter implements an imperfect evolutionary market under the integrated individual and social learning paradigm. The imperfect evolutionary market is studied from three aspects: absorption of new challenges and creation of new knowledge; roles of individual learning in terms of its intensity; and roles of social learning in terms of its popularity. This chapter has been disseminated via the following publications: Kendall and Yan (2004, 2006).

6.1 Modelling An Incomplete Environment

The problem of modelling an imperfect environment, i.e. an incomplete environment that constantly evolves, is essentially the problem of modelling an incomplete problem space. In our simulated stock market model, described in the last chapter, the problem space of artificial traders consists of 20 market indicators. The task for artificial traders is to discover successful trading models from these 20 indicators. During a 15-year trading period, the market maintains a static problem space of 20 market indicators. This is generally not true in real world markets, where novel financial techniques and models are continually being developed. What will happen when a new market index or financial tool is introduced to the market? When a new market index is introduced, it will be experimented with and analysed by investors and analysts from the financial

markets. These people who have used the index will then tell other investors in the market what they think about the index, possibly with suggestions on how to refine the use of the index in making financial decisions. Gradually the new market index will be accepted by most of the investors if it works well with the market. If we view the market as a perfect information system, there will be no space or opportunities for new techniques to be introduced. A perfect information system will refuse to react to new challenges from its environments. Note that in information economics, imperfect information is more related to asymmetric information. In our imperfect evolutionary system, imperfect information is more related to new challenges.

Therefore, we have changed the perfect problem space of 20 market indicators in the simulated stock market model to an incomplete problem space where new challenges are introduced to artificial traders at varying times. Instead of initialising the simulated market with 20 stock indicators, we initialise the market with 10 market indicators, i.e. IND1 to IND10 in Table 6.1 below.

Technical Indicator	Description	Technical Indicator	Description
IND1	10 days Moving Average (price)	IND11	14 days Relative Strength Index
IND2	20 days Moving Average (price)	IND12	21 days Relative Strength Index
IND3	50 days Moving Average (price)	IND13	Stochastic Oscillators (K%)
IND4	200 days Moving Average (price)	IND14	Fast Stochastics (D%)
IND5	Closing Price (normalized)	IND15	Slow Stochastics (slow D%)
IND6	Rate of Change (price)	IND16	Primary Market Index Rate of Change
IND7	Oscillator (price)	IND17	Relative performance to Primary Market Index
IND8	10 days bias (price)	IND18	Secondary Market Index Rate of Change
IND9	20 days volume Rate of Change	IND19	Relative performance to Secondary Market Index
IND10	10 days Relative Strength Index	IND20	Third Market Index Rate of Change

TABLE 6.1 Dynamic environmental variables for an incomplete problem space. All values are normalised in the range of [0, 1] before being used as input to the neural networks. For all stocks, the Secondary Market Index refers to DJ INDU AVERAGE. For stocks from HKEx, the Primary Market Index refers to Hang Seng Index; the Third Market Index refers to NIKKEI 225 index. For stocks from TSE, the Primary Market Index refers to NIKKEI 225 index; the Third Market Index refers to Hang Seng Index.

The other 10 indicators, i.e. IND11 to IND20, are gradually introduced into the simulated stock market as new indexes every two years. Artificial traders in the imperfect evolutionary market must react to the 10 new indicators when they appear in the market and learn how to use them. For experiments on imperfect evolutionary market, we use five stocks from the Hong Kong Stock Exchange and TOKYO Stock Exchange. Indicators IND16 to IND20 are based on three major market indexes, i.e. the Dow Jones Industrial Average, the Hang Seng Index, and the NIKKEI 225, respectively.

6.2 Integrated Individual Learning and Social Learning

The market model is changed slightly to reflect more realistic account on transactions and costs:

- Every trader is given an initial portfolio of 10,000 cash and 5000 shares of a stock. A more realistic scheme used for transactions is employed compared to the either sell or buy it all scheme used in the previous chapter (Kendall and Yan 2003, 2003a). For every transaction, the amount of buying or selling is indicated by the output from the neural network, δ, in percentages as shown in Table 6.2.

δ	**Mapped trading decision**
$\delta > 0.5$	Buy (COH*(δ-0.5)/0.5)
$-0.5 \leq \delta \leq 0.5$	Hold
$\delta < -0.5$	Sell (SOH*(δ+0.5)/0.5)

TABLE 6.2 Mapping the outputs (δ) from the artificial neural networks to trading decisions. δ is in the range of [-1,1]. COH stands for cash_on_hold that equals to the amount of cash the trader is holding currently. Formula, COH*(δ-0.5)/0.5, gives the amount of cash the trader will use to buy the share, e.g., when δ is 0.70, the trader will use 40% of his cash to buy shares. SOH stands for share_on_hold which equals to the amount of shares the trader is holding currently. Formula, SOH*(δ+0.5)/0.5, gives the amount of shares the trader will sell, e.g., when δ is -0.70, the trader will sell 40% of the shares he holds.

- For every transaction, traders are required to pay a commission fee of 0.1% of the total transaction amount.

While the individual learning mechanism is the same as described in Table 5.3, the social learning mechanism of the integrated individual and social learning paradigm in the simulated stock market is modified so that artificial traders will be given opportunities to explore new stock indicators appeared in the market. The modified social learning algorithm is presented in Table 6.3 below.

1. If a trader's assessment is 1, and the trader is not using a strategy drawn from the pool, then publish the strategy into the central pool. Go into the next six months trading using the same strategy.

2. If a trader's assessment is 1, and the trader is using a strategy copied from the pool, do not publish it again, but update this strategy's score in the pool using their six-month ROP. Go into the next six months trading using the same strategy.

3. If a trader's assessment is less than 0.9, the trader has 0.5 probability of copying a strategy from pool, which means the trader will discard whatever model he is using, and select a better trading strategy from the pool using roulette selection, and go into the next six months trading with this copied strategy. Or, with 0.5 probability, the trader will decide to discard whatever strategy he is using, and select another set of indicators as inputs, build 10 new models and go into next six months trading with these 10 new models.

4. If the assessment is between 0.9 and 1, the trader is currently satisfied with his performance over the past six months. With 0.7 probabilities the trader will continue using the same set of indicators. With 0.3 probabilities the trader decides to try a new set of indicators.

TABLE 6.3 A social learning algorithm in the imperfect evolutionary market

Two steps in the modified social learning algorithm ensure artificial traders react to new challenges from the market, and learn to use the newly introduced market indicators. In step 3, when a trader is not satisfied by his performance so far, as well as having the chance to learn a strategy from other traders through the central pool, the trader can also choose to select a new set of indicators for trading instead. This allows

newly introduced indicators to be used. In step 4, when a trader is satisfied with his performance, he can choose to continue with his current information set, or the trader may choose to explore newly introduced indicators, which might be used to construct even better trading models and hence produce higher returns.

6.3 Adaptation and Creativity

Table 6.4 lists the detailed profiles of the five selected stocks used in this set of experiments for our imperfect evolutionary market. As we have discussed, an imperfect evolutionary system does not have a consistent problem space. Environmental variables, i.e. the 20 indicators in Table 6.1, change dynamically when new information and knowledge are introduced or discovered. When new technical indicators are introduced into the evolutionary market in the form of new environmental variables, the problem space of artificial traders in fact extends into more dimensions. The absorption and dissemination of new information and knowledge inside the evolutionary system becomes crucial to the adaptability and creativity of intelligent individuals.

Company	Symbol	Market	Sector	Trading Period	Return (BnH)	Return (Bank)
CHENG KONG (CK)	0001.hk	HKEx	Real Estate	25/April/88 to 03/July/03 (3750 trading days)	410.46%	109.76%
CATHAY PAC AIR (CATHAY)	0293.hk	HKEx	Transports	25/April/88 to 03/July/03 (3750 trading days)	46.21%	109.76%
WHARF HOLDINGS (WHARF)	004.hk	HKEx	Conglomerate	25/April/88 to 03/July/03 (3750 trading days)	134.11%	109.76%
TOYOTA INDUS CORP (TOYOTA)	6201.jp	TSE	Automobiles	15/May/92 to 04/July/03 (2750 trading days)	122.6%	72.16%
SONY CORP (SONY)	6758.jp	TSE	Electronics and Electrical	15/May/92 to 04/July/03 (2750 trading days)	-16.93%	72.16%

TABLE 6.4 Five selected stocks traded in the imperfect evolutionary market. HKEx refers to Hong Kong Stock Exchange. TSE refers to Tokyo Stock Exchange. Return refers to accumulated total return over the whole trading period; Return (BnH) = Cumulative total return from employing the Buy and Hold strategy (BnH) on that stock, Return (Bank) = Cumulative total return from investing all assets in bank savings over the same trading period (with an interest rate of 5%, paid annually).

The 20 indicators from Table 6.1 are divided into two groups: IND1 to IND10 (the *static variables*) and IND11 to IND20 (the *new challenges*). The group of 10 indicators from IND1 to IND10 are given to the imperfect evolutionary market as initial settings, i.e., from the start of trading. The remaining 10 indicators are introduced into the market every two generations, i.e., indicator IND11 is introduced to the market at generation 2, indicator IND12 is introduced to the market at generation 4, and so on. Thus, the market is gradually injected with new techniques such as stochastic oscillators and new knowledge such as Hang Seng Index, Nikkei 225 Index and the Dow Jones Industry Average. The environmental space of artificial traders expands as the number of environmental variables increases. Generations here refer to the generations of social learning, i.e. each generation consists of 125 days trading, or two generations approximately one-year trading.

Within the integrated individual and social learning paradigm, social learning acts as a mechanism for the absorption and dissemination of new information into the trading society. During the social learning stage, besides traders being able to publish their successful trading strategies, traders with poor performances have the opportunity to discard the old set of indicators they used, and select a new set of indicators from the environment, which possibly includes newly introduced indicators. In addition, for traders who are satisfied with their performance thus far, besides allowing them to keep their old trading models, we also give these traders the opportunity to try out other new indicators from the market, as shown in Table 6.3. When successful trading strategies are developed through the employment of newly introduced indicators, these

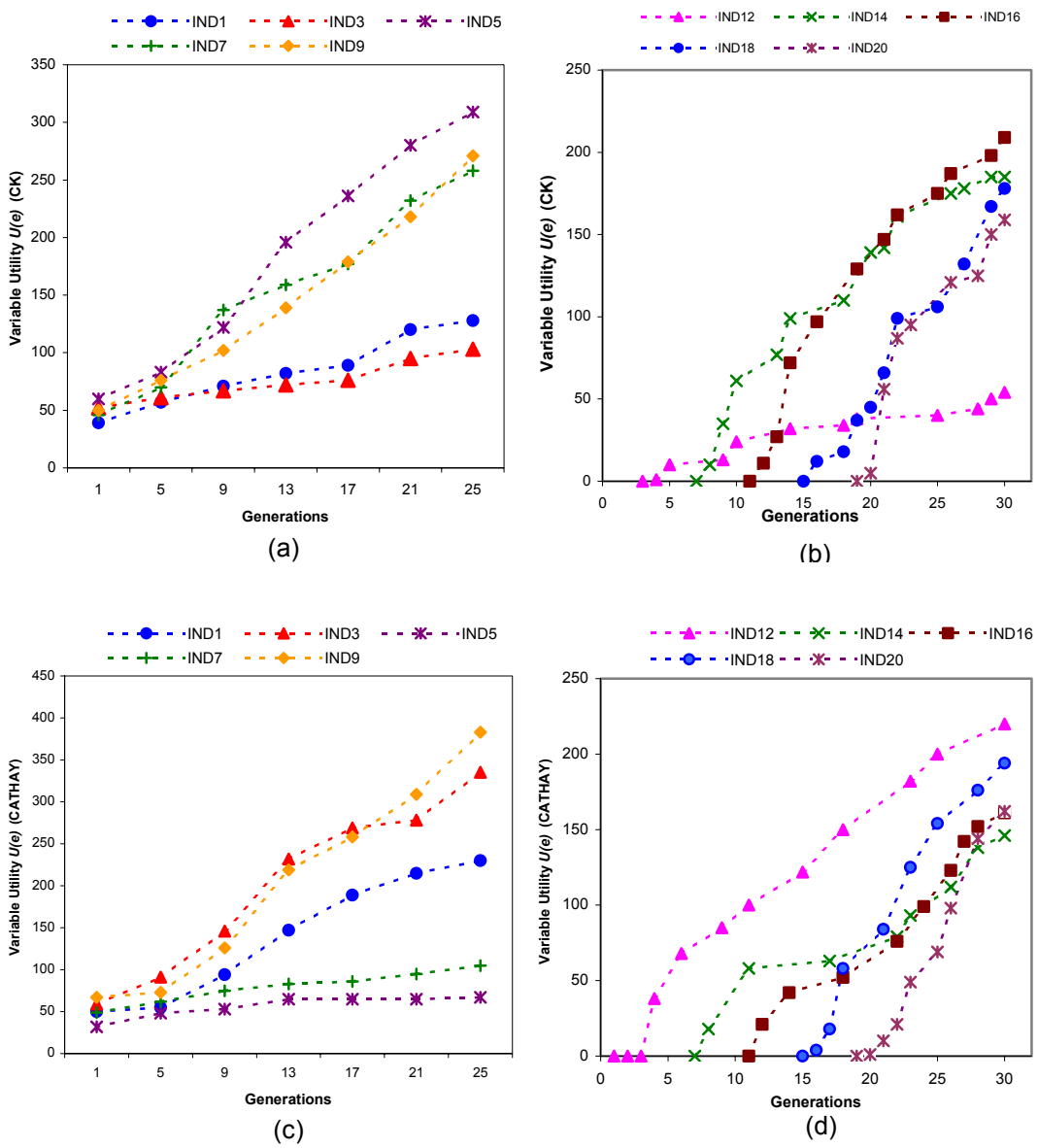


FIGURE 6.1 Usage of environmental variables and absorption of new information (CK(a/b) and CATHAY(c/d)).

good trading strategies will also be published to the society and learned by other traders. Thus, the new indicators are disseminated among the trading society. In order to put pressure on the use of new information, we assign the newly introduced indicators with the highest score among the current indicators. This mimics the phenomenon that new things appear in a society generally attract most attention from the public.

We define $U(e)$ as the utility of an environmental variable. $U(e)$ can be understood as the usefulness or worthiness of an environmental variable. In the imperfect evolutionary market, an indicator's $U(e)$ simply equals the number of times that it has been selected for use by traders. By examining the changes in an indicator's utility $U(e)$, we will be able to see how new information is absorbed and disseminated within the simulated stock market. Ten simulations are run on each of the five selected stocks with the dynamically changing environmental variables. Figure 6.1, 6.2 and 6.3 depict 20 indicators' $U(e)$ from the best run on each stock. The $U(e)$ of indicators IND1 to IND10 are recorded starting from generation 1. For indicators IND11 and IND20, the $U(e)$ are recorded starting from the generation before they are introduced into the market. For example, IND20 is introduced in generation 20, and its $U(e)$ is depicted starting from generation 19, when it has the value of 0. Therefore, all utilities of indicators IND11 to IND20 start with the value of 0, which, in fact, indicates that they represent new information in the environment. Figure 6.1(a) shows the utilities of the *static variables* from the simulation run on the CK stock with 5 randomly selected indicators' $U(e)$ from IND1 to IND10. Figure 6.1(b) shows the traders' response to the *new challenges* in the environment from the simulation on the CK stock with 5 randomly selected indicators' $U(e)$ from IND11 to IND20. In Figure 6.1(a), the indicators are static environmental variables given to the market at the beginning of the evolution. Each line in Figure 6.1(a) starts at generation 1 with a non-zero value, which indicates the frequency the indicator is selected in the very first round of social learning during the evolution. Because a trader is allowed to select an indicator more

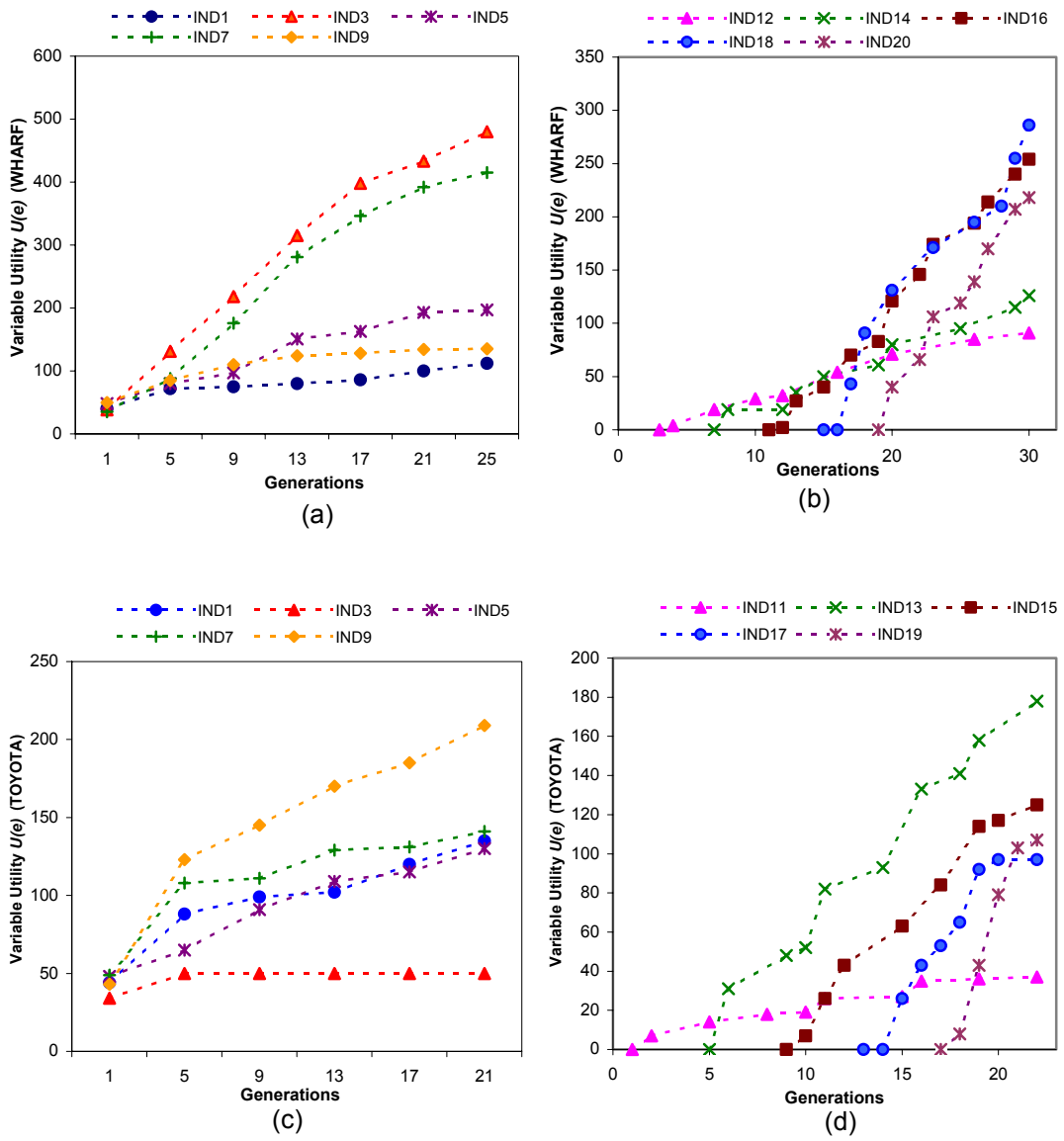


FIGURE 6.2 Environmental variable utilities and absorption of new information (WHARF(a/b) and TOYOTA(c/d)).

than once, we see some indicators' $U(e)$ started with value that is greater than 50. As the evolutionary process continues, the indicators' utilities increase. In Figure 6.1(b), things happen in a different way. Each indicator in Figure 6.1(b) is introduced to the market at different time. Before it is introduced, the indicator does not exist in the market. Each $U(e)$ line in Figure 6.1(b) starts at a different point on the x-axis, which

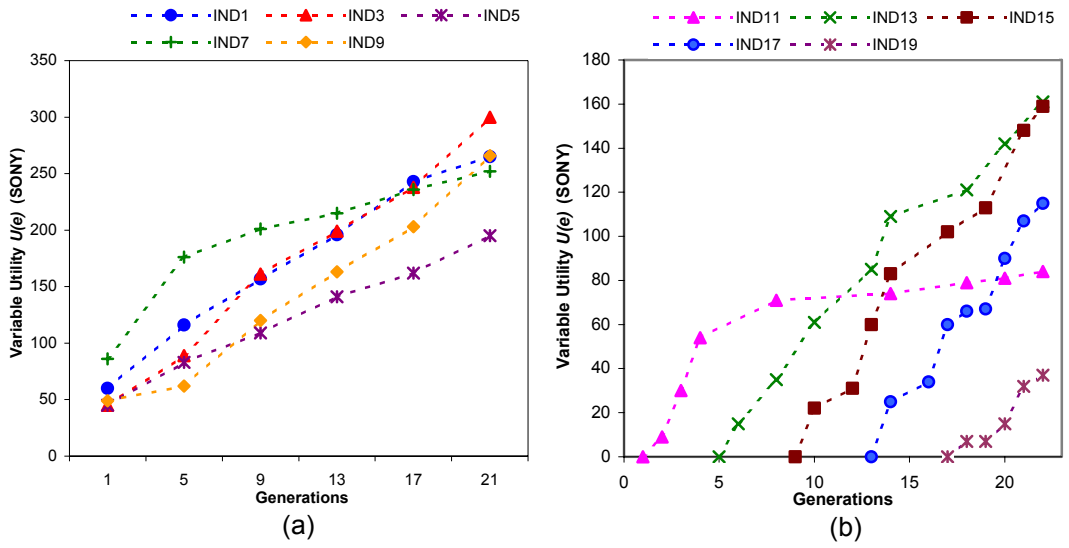


FIGURE 6.3 Environmental variable utilities and absorption of new information (SONY)

corresponds to the generation before it is introduced to the market, and has a utility of 0. When it comes to the generation the indicator is injected, the social learning mechanism will put pressures on the traders to try this new indicator for building new trading models. This results in the increase in $U(e)$ values after the starting point of each line. As shown in the diagram, new indicators are gradually selected and used by traders. If an indicator is useful for trading, it will be accepted by more traders through the social learning process, and subsequently has a high utility, such as IND16 in Figure 6.1(b). If the new information is not as useful, it gradually loses favour with traders, such as IND12 in Figure 6.1(b). Figure 6.1(b) clearly demonstrates the absorption and dissemination of new environmental variables by the trading society during the simulation on CK stock. Similar phenomena can also be observed on the CATHAY stock (Figure 6.1(c)&(d)), the WHARF stock (Figure 6.2(a)&(b)), the TOYOTA stock (Figure 6.2(c)&(d)), and the SONY stock (Figure 6.3(a)&(b)). Some indicators' utilities stayed at zero for few generations after they have been introduced

to the market, such as IND18 in Figure 6.2(b) and IND17 in Figure 6.2(d). That is because they have not been selected for use by traders when they were first introduced, either because of the randomness in the selection procedure, or because at the time the traders are generally doing well, and there is little pressure on social learning. Interestingly, by looking at Figure 6.1(b), Figure 6.1(d), and Figure 6.2(b), we find that IND16, IND18, and IND20 are generally well accepted by the artificial stock traders. According to Table 6.1, these three indicators are rates of change of the three major market indexes. It is accepted by investors in the stock market, market indexes have a significant impact on the stock prices. Often stock prices simply rise and fall along with the rise and fall in major market indexes regardless of the company's actual performance. We want to stress that the *new challenges* (IND11 to IND20) are gradually introduced to the market as black boxes, i.e., the traders do not know what these indicators represent. The traders are only aware that some new information has appeared. Through evolution, artificial agents find the best way to make use of the new information. As a summary, the experiments clearly demonstrate the absorption and dissemination of new information and knowledge within the imperfect evolutionary market using the ISP learning paradigm, and the adaptation of artificial stock traders to the new challenges from their imperfect environment.

In Table 6.5 we compare the performance of artificial traders who evolve under the static 20 environmental variables, against traders who evolve with dynamically changing environmental variables. We use four criteria for the comparison of artificial stock traders' performance with two benchmark investment strategies: buy and hold

Stock	Static Environments				Dynamically Changing Environmental Variables			
	Outperform BnH	Outperform BS	Best Return (%)	Average Return (%)	Outperform BnH	Outperform BS	Best Return (%)	Average Return (%)
C.K.	44	50	2953.47	947.21	28	49	1862.19	672.33
CATHAY	50	49	1155.03	413.88	34	19	440.9	98.3
WHARF	50	50	1639.89	732.79	34	40	1312.99	280.17
TOYOTA	42	49	417.14	228.23	16	33	359.94	102.93
SONY	49	38	718.07	175.75	49	27	591.99	109.79

TABLE 6.5 Comparison of results on static and incomplete environment (artificial traders' trading performance). Results are taken from the best run out of 10 runs on each stock under different conditions, i.e., static environmental variables and dynamic environmental variables. Return refers to accumulated total return over the whole trading period; Outperform BnH = number of traders outperform the buy and hold strategy (BnH) out of the 50 traders, Outperform BS = number of traders outperform bank savings (BS) out of the 50 traders, Best Return = maximum return from the 50 traders, Average Return = average return from the 50 traders.

strategy and risk-free bank savings. The four criteria used are: number of traders who outperformed the buy and hold strategy; number of traders outperformed the risk-free bank savings; the best trader's cumulative total return; the average cumulative total return of 50 traders from the best run. Clearly, from Table 6.5, traders using dynamic environmental variables performed poorly compared to traders using static environmental variables. This is understandable as the traders with the static environmental variables have more opportunities to make use of indicators IND11 to IND20 from the beginning of the evolution, whereas, the traders with dynamic environmental variables are only able to make use of indicators IND11 to IND20 when they appear in the market. Obviously traders in an imperfect environment are disadvantaged in terms of knowledge and time. The comparison shows that even for many scientific problems, people know what the exact problem space is, however, for real life problems, the environmental space is not of a static nature. Dealing with an imperfect environment with new challenges is a problem that should not be ignored.

6.4 Studies on Individual Learning

How often should individual learning and social learning take place in the integrated individual and social learning paradigm? Will more frequent individual learning of artificial traders benefit the trader, or actually cause chaos in the learning process? In this section and the next, we investigate the individual and social learning algorithm, focusing on the sensitivity of the parameters of the learning paradigm. We first experimented with the ISP learning paradigm with various intensity on individual and social learning.

Timescale	Description
5-125	Individual learning occurs every 5 trading days Social learning occurs every 125 trading days
25-125	Individual learning occurs every 25 trading days Social learning occurs every 125 trading days
5-250	Individual learning occurs every 5 trading days Social learning occurs every 250 trading days
25-250	Individual learning occurs every 25 trading days Social learning occurs every 250 trading days

TABLE 6.6 Individual learning and social learning under various intensities.

In Table 6.6, we define a fast individual learning (every 5 trading days) and a slow individual learning (every 25 trading days). We also define a fast social learning (every 125 trading days) and a slow social learning (every 250 trading days). A 5-day trading period is approximately one week of trading (sometimes it is shorter e.g. when there is a public holiday). A 125-day trading period is approximately a 6-month trading period. Simulations, under the four timescales in Table 6.6, are run on each of the five stocks from Table 6.4. Ten runs are carried out on each timescale. The results from the best runs of each stock are depicted in Figures 6.4 and 6.5.

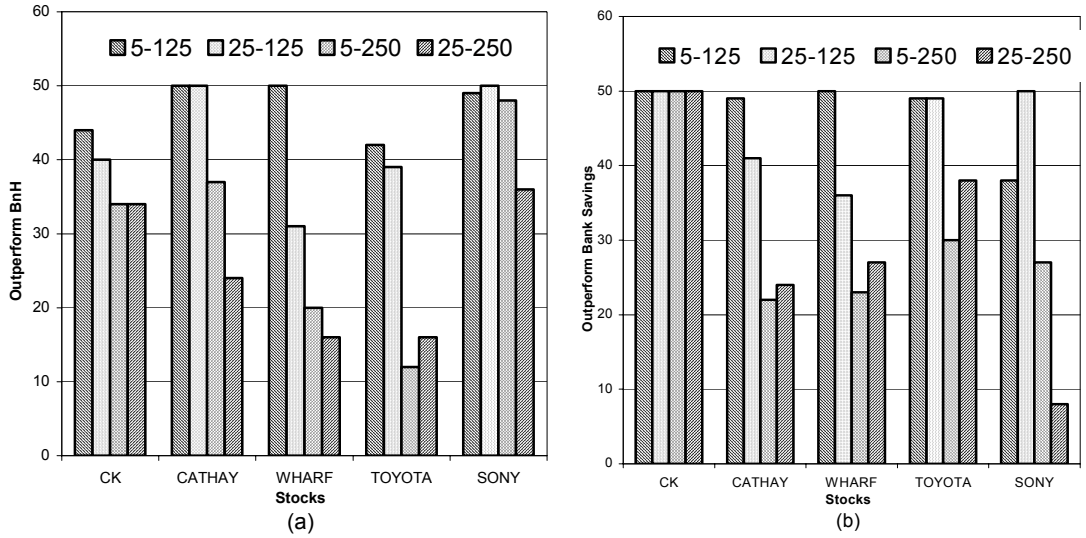


FIGURE 6.4 Comparisons of individual learning and social learning under different intensities (A).

We use the same four criteria for the comparison of artificial traders' performance under different frequencies in individual and social learning: the number of traders who outperformed the benchmark BnH strategy shown in Figure 6.4(a); number of traders who outperformed the risk-free bank savings shown in Figure 6.4(b); the top trader's cumulative total return as shown in Figure 6.5(a); the average cumulative total return of 50 traders from the best run as shown in Figure 6.5(b). As shown in Table 6.4, the classical Buy and Hold strategy does not suit every one of the five selected stocks. For example, CATHAY and SONY are two stocks that suffered a serious loss in the past few years due to the change in economic climate or a failed corporate development strategy. The apparent choice for investors in this case is to leave their money in the bank. Nevertheless, the artificial active stock traders in the simulated stock market, who use artificial neural networks as trading models to detect buy and sell signals from the market, have developed successful trading strategies that beat investments in bank savings in the case of CATHAY and SONY as shown in Figure

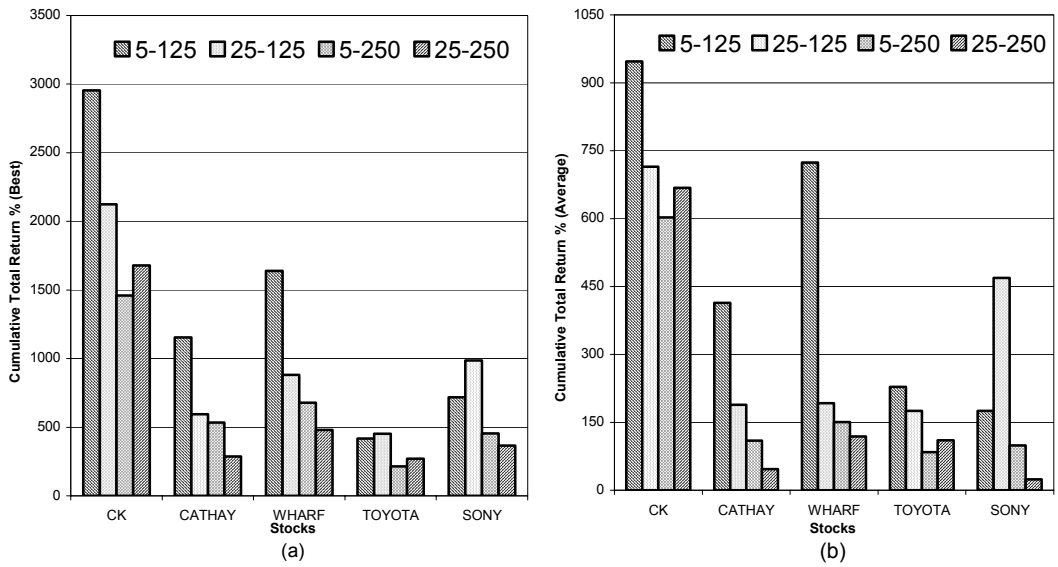


FIGURE 6.5 Comparisons of individual learning and social learning under different intensities (B).

6.4(b), and beat the Buy and Hold strategy in the case of CK, WHARF and TOYOTA, as shown in Figure 6.4(a).

With regard to the effect of changing frequencies in individual and social learning, Figure 6.4 and Figure 6.5 show very clearly that the learning ability of artificial traders declines when the frequency of social learning is reduced. Simulations with frequent social learning (5-125 and 25-125) generally improve the agents' performance significantly in all four criteria. This phenomenon can be explained in two ways. The reduction in the frequency of social learning results in less opportunities that allow agents to learn from other successful agents, and hence it takes agents much longer to find a good trading strategy in which time they may suffer from serious losses from which they may never recover. The reduction in social learning also means less successful trading strategies are developed in the society as agents have less opportunities to publish their good strategies to the society and have less opportunities

to discard inferior information and select new sets of indicators from the market for building new trading models. However, the effects of individual learning occurring at different frequencies are not so clear. For example, consider CK in Figure 6.4(a), fast individual learning improves the performance of the best trader when social learning is fast, whereas fast individual learning did not improve the performance when the social learning speed was reduced. However, WHARF in Figure 6.4(a), fast individual learning produced better results when both social learning is fast and social learning is slow. We also see mixed results on other stocks with other criteria regarding the effects of individual learning speed. Apparently, it is not reasonable to simply assert that fast (or slow) individual learning is better. The mixed results on individual learning are due to several reasons. Fast individual learning accelerates the discovery of good trading models, and also enables traders to adapt to changes in the market more quickly. However, as the performance of artificial traders is evaluated using trader's profits from the past week, or past six months, there are problems that the fitness of a trading model may not be completely reflected within a short generation. As an example, during the correction of a stock market, the stock prices will drop from a previous high, but are usually followed by another surge in the stock price. A good trading strategy in a period like this is to buy stocks when the prices have dropped to a certain resistance level, and wait for a bull market. The correction of a market may take a few weeks, or a few months. Individual learning with a short timescale of 5 days will certainly not fully reflect the trading strategy's profitability. This causes the possible loss of good trading strategies, and the evolution may step in a wrong direction.

In fact, what the results show us is that a dynamic individual and social learning rate is probably more appropriate than the fixed learning frequencies in the ISP paradigm. Instead of designing the agents to learn at a certain time, the learning entities should have the right to decide when they need individual learning or when they need to look to the society for help. Thus, every individual within the dynamic imperfect environment will adjust their learning frequencies depending on their own evolutionary process. In addition, forward learning methods, i.e. evaluate a trading model's fitness by calculating its future profitability, should be used in coordination with backwards learning in order to offset the problems caused by fitness evaluation timescales.

6.5 Studies on Social Learning

We also conducted a further study on social learning in Kendall and Su (2004), where different social learning sentiments are mimicked in the trading society. The social learning algorithm is modified in the following way. After each trader's final assessments are evaluated by using equation 5.6, the assessments are then normalised into the range of [0,1] and denoted as ω_i where *i* is the *i*th trader. An arithmetic mean value (Φ) of all 50 traders' normalised final assessments (ω_i) is calculated using equation 6.1 below.

$$\Phi = \frac{1}{50} \sum_{i=1}^{50} \omega_i \tag{6.1}$$

A trader's social behavior will now fall into four different cases depending on the four parameters: ω_i, Φ, θ_1 and θ_2 (θ_1 and θ_2 are two thresholds, see below in the detailed

experimental settings). The four cases, i.e. four different types of social behaviours, are still the same as those in the original model. That is:

CASE1: The trader is successful and he is not using a strategy learned from the pool. The trader will publish his novel trading strategy into the central pool and enter the next six months trading using the same strategy.

CASE2: The trader is successful, but he is using a strategy copied from the pool. Do not publish the strategy again, but update this strategy's score in the pool using their six-month ROP. The trader will then enter the next six months trading using the same strategy.

CASE3: The trader is not satisfied with his performance in the last six-month trading. The trader will have 0.5 probability of copying a strategy from pool, which means the trader will discard whatever model he is using, and select a better trading strategy from the pool using roulette selection, and go into the next six months trading with this copied strategy. Or, with 0.5 probability, the trader will decide to discard whatever strategy he is using, and select another set of indicators as inputs, build 10 new models and go into the next six months trading with these 10 new models.

CASE4: The trader is satisfied with his performance in the past six months, either continues using that strategy or choose a new set of indicators.

Four different experimental settings are explained below:

SETTING1 – Social learning is turned off while only individual learning occurs. Each trader in the simulated stock market evolves independently from each other. Each trader can only search their own parameter space that is defined by the indicators they have selected from the environment.

SETTING2 – Both individual and social learning are turned on. The parameter θ_1 for social learning is set to 1. The parameter θ_2 is set to 0.9. The traders' social behaviour during the social learning is described in Table 6.7 (see equation 6.1 for ω_i):

Parameters	**Trader *i*'s Social Behaviour**
$\omega_i = \theta_1$	CASE1 (use a strategy from the pool) CASE2 (do not use a strategy from the pool)
$\omega_i < \theta_2$	CASE3
$\omega_i > \sigma_i \geq \omega_i$	CASE4

TABLE 6.7 Social learning with strong motivation.

Setting2 mimics an environment where only the best players are accepted and individuals have strong motivation to learn from each other.

SETTING3 – Both individual and social learning are turned on. The parameter θ_1 for social learning is set to 1. The parameter θ_2 is set to the mean value Φ (see equation 6.1). The traders' behaviour during the social learning is described in Table 6.8:

Parameters	Trader *i*'s Social Behaviour
$\omega_i = \theta_1$	CASE1 (use a strategy from the pool) CASE2 (do not use strategy from the pool)
$\omega_i \leq \theta_2$	CASE3
$\theta_1 > \omega_i > \theta_2$	CASE4

TABLE 6.8 Social learning with weak motivation.

Setting3 creates an environment where only the best players are accepted but individuals have less motivation to learn from each other compared with setting1.

SETTING4 – Both individual learning and social learning are turned on. The parameter θ_1 for social learning is set to 0.9. The parameter θ_2 is set to the mean value Φ. Traders' behaviour during the social learning is described in Table 6.9:

Parameters	Trader *i*'s Action
$\omega_i > \theta_1$	CASE1 (use a strategy from the pool) CASE2 (do not use a strategy from the pool)
$\omega_i \leq \theta_2$	CASE3
$\theta_1 \geq \omega_i > \theta_2$	CASE4

TABLE 6.9 Social learning with moderate motivation.

Setting4 simulates an environment where more individuals have the opportunity to distribute their knowledge to the society while the learning atmosphere in the society is moderate.

The four different social learning scenariors are tested on the five stocks from Table 6.4. The experimental results are depicted in Figure 6.6 which compares the algorithm where no social learning is allowed, i.e. Setting 1, with the algorithms with integrated

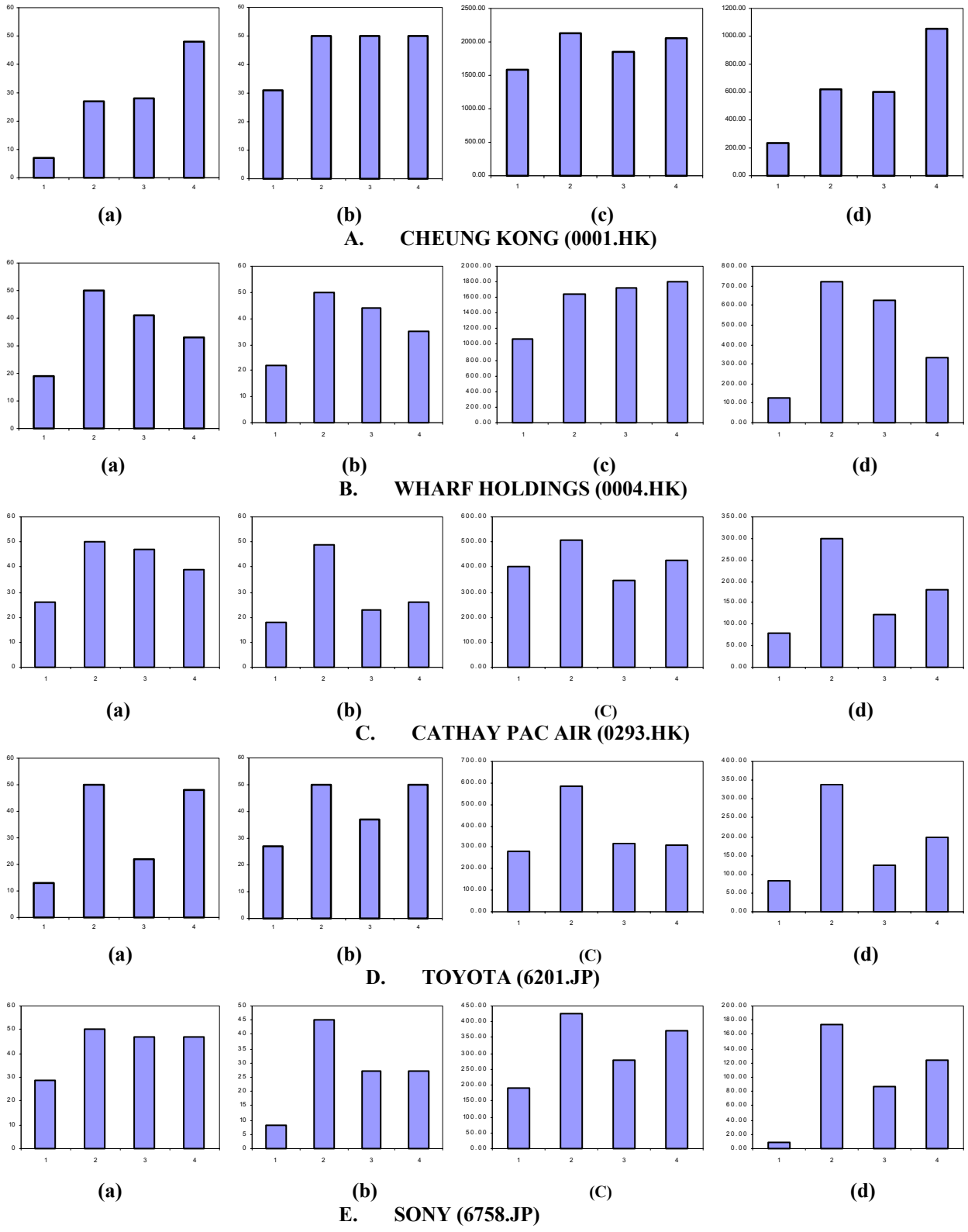


FIGURE 6.6 Comparison between the algorithm without social learning (SETTING1) and the algorithms with different pressures on social learning (SETTING2, 3, 4). On all the X axes, 1 refers to SETTING1, 2 refers to SETTING2, 3 refers to SETTING3, and 4 refers to SETTING4 (see section). (a) – Number of traders outperformed the bank savings. (b) – Number of traders outperformed the buy and hold strategy. (c) – The cumulative total return of the best trader from the 50 traders. (d) – The average cumulative total return of all 50 traders.

individual learning and social learning of different settings. We compared the traders performance from each simulation with the two benchmarks: bank savings and buy-and-hold strategy. In Figure 6.6, A(a) to E(a), show the number of traders who outperformed bank savings under four different setting for the particular stock. A(b) to E(b) show the number of traders who outperformed the classic buy-and-hold strategy. A(c) to E(c) show the cumulative total ret/urns of the best traders under four different settings for the particular stock. A(d) to E(d) show the average cumulative total returns of 50 traders. See equation 5.8 for the calcualtion of cumulative returns.

We use numbers "1" to "4" to represent four different settings on social learning in each diagram of Figure 6.6. The first observation we can drawn from Figure 6.6 is that, in each diagram, each column marked as "1" is shorter than the other three columns. In other words, when social learning is switched off, the performance of the artificial agents in the simulated market was severely degraded. Secondly, by looking at the diagrams A(c) to E(c) vertically, we observe that the columns marked as "2" generally achieve much higher values than the other three settings in the same diagram, except for Wharf Holdings in B(c) where the performance of the three settings (2, 3, 4) with social learning are similiar. Category "c" in Figure 6.6 refers to the cumulative total return of the best trader from the 50 traders, and setting "2" refers to the learning algorithm which provides a strong motivation on individuals to take part in social learning. In other words, diagrams A(c) to E(c) show that the enforcement of social learning in the imperfect market encourage the development of good trading strategies, i.e., successful traders.

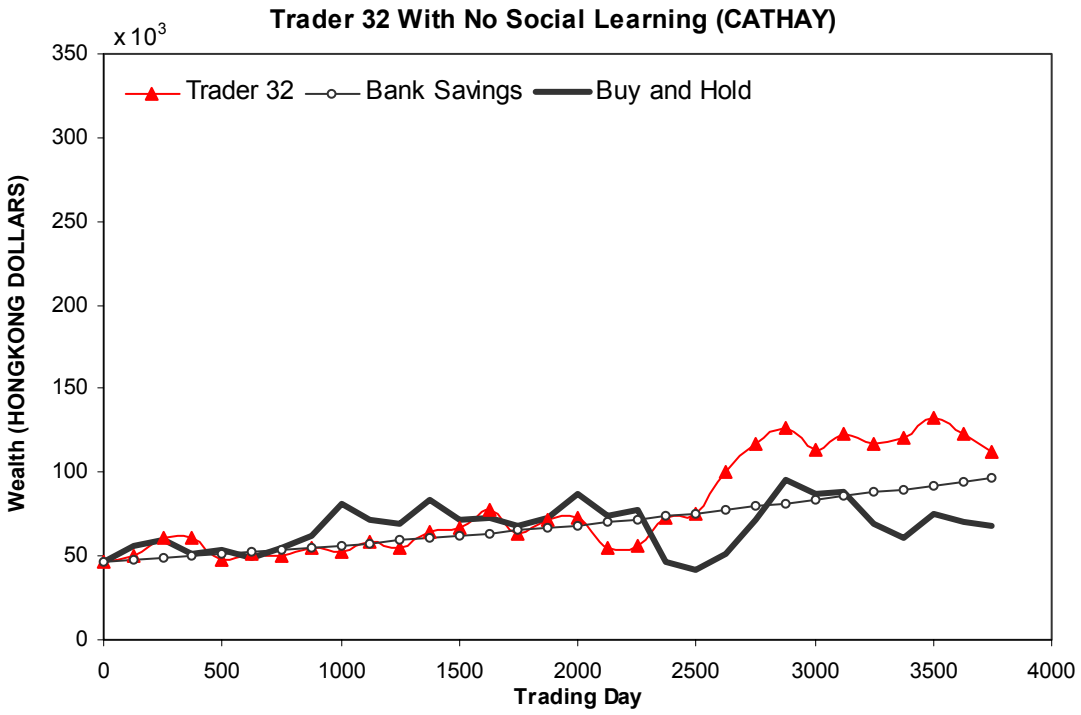


Figure 6.7 An artificial stock trader with no social learning under setting 1.

In order to examine the effects of social learning on the evolution of artificial stock traders in the simulated stock market, we also selected four distinct traders from the simulations run on the CATHAY stock. Each of these four traders demonstrates an example of the typical learning behaviours of artificial traders under the four different settings of social learning in the integrated individual and social learning algorithm, i.e. setting 1 where the social learing is turned off during the simulation and the traders are completely relied on their own individual learning; setting 2 where a strong pressure is put on the artificial traders to encourage them to take part in the social learning; setting 3 where the artificial traders only have moderate motivation to choose to learn from others through social learning; setting 4 where the moderate motivation for taking part in social learning is maintained but more opportunities are given to the

Day	Successful Trade (Daily)	Successful Trade (Weekly)	Successful Trade (Monthly)	Social Learning
1500	24.8%	32.8%	23.2%	Social Learning is turned off
1625	31.2%	32.0%	20.8%	Social Learning is turned off
1750	22.4%	19.2%	17.6%	Social Learning is turned off
1875	24.0%	20.0%	20.0%	Social Learning is turned off
2000	26.4%	21.6%	18.4%	Social Learning is turned off
2125	21.6%	19.2%	22.4%	Social Learning is turned off
2250	23.2%	20.0%	20.8%	Social Learning is turned off
2375	28.8%	27.2%	21.6%	Social Learning is turned off
2500	30.4%	29.6%	31.2%	Social Learning is turned off
2625	38.4%	35.2%	39.2%	Social Learning is turned off
2750	43.2%	44.0%	44.8%	Social Learning is turned off
2875	46.4%	43.2%	37.6%	Social Learning is turned off
3000	37.6%	34.4%	31.2%	Social Learning is turned off

Table 6.10 Trader 32's trading statistics when social learning is turned off.

artificial traders to publish their strategies to the trading society.

Figure 6.7 and Table 6.10 demonstrate trader 32's performance and trading statistics when no social learning occurs. Figure 6.8 and Table 6.11 demonstrate trader 12's performance and trading statistics with social learning under setting 2. Figure 6.9 and Table 6.12 demonstrate trader 44's performance and trading statiscs with social learning under setting 3. Figure 6.10 and Table 13 demonstrate trader 2's performance and trading statistics with social learning under setting 4. We use the same criteria, described in equations 5.9 to 5.12, to evaluate the successfulness of a particular transaction. In other words, the successfulness of a trade is evaluated in terms of different investment horizons, namely, daily, weekly, and monthly. A trade that seems

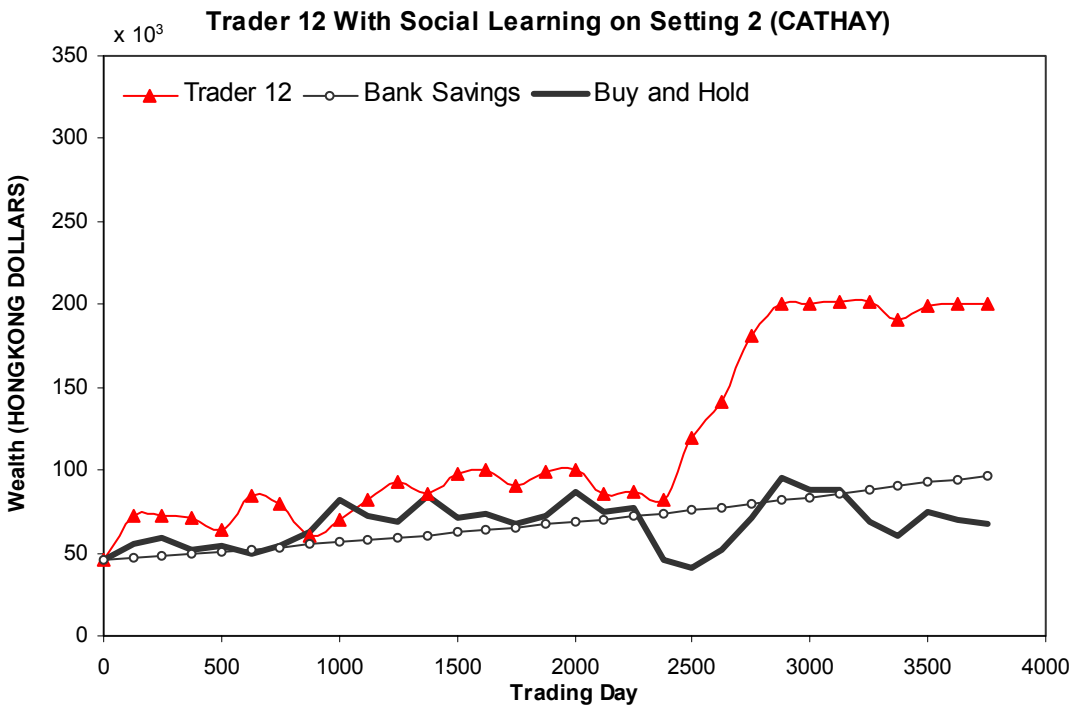


Figure 6.8 An artificial stock trader with strong motivation for social learning under setting 2.

meaningless in a short time period may actually generate a considerable profit after a month time when the stock price goes up. A trade that seems making some profits at the present time may in fact cause even bigger losses in the long-term sense. We collect the trading statistics on each of the four traders in the period of trading day 1375 to trading day 3000.

Comparing to traders that learn from others through social learning, as shown in Figures 6.8 to 6.10, we can see from Figure 6.7 that it requires a much longer time for the trader 32, who did not use any forms of social learning, to evolve better trading strategies. From the start of the trading, trader 32 keeps on evolving his trading strategies completely relying on his own individual learning experiences. As shown in

Day	Successful Trade (Daily)	Successful Trade (Weekly)	Successful Trade (Monthly)	Social Learning
1500	51.2%	48.0%	42.4%	Satisfied, keep on going
1625	53.6%	50.4%	44.0%	Satisfied, keep on going
1750	41.6%	39.2%	36.0%	Copied a strategy from the central pool
1875	48.8%	44.8%	40.0%	Satisfied, keep on going
2000	52.0%	46.4%	41.6%	Satisfied, keep on going
2125	40.8%	37.6%	36.0%	Select a new set of indicators
2250	42.4%	38.4%	36.0%	Satisfied, keep on going
2375	36.0%	33.6%	31.2%	Copied a strategy from the central pool
2500	67.2%	76.0%	80.0%	Satisfied, keep on going
2625	65.6%	75.2%	82.4%	Satisfied, keep on going
2750	81.6%	80.8%	84.8%	Satisfied, keep on going
2875	87.2%	79.2%	77.6%	Satisfied, keep on going
3000	80.0%	71.2%	69.6%	Satisfied, keep on going

Table 6.11 Trader 12's trading statistics with strong motivation for social learning under setting 2.

Figure 6.7, before the trading day 2500, the performance of trader 32 was worse than the performance of the risk-free bank savings at most of the time. After day 2500, along with the changing market conditions, trader 32 started to make profits in the rising market. The trading statistics of the trader 32 from Table 6.10 show that the evolved trading strategies learned by trader 32 through individual learing were able to seize some trading opportunities during the upturn of the market, and were able to transfer stocks into bank savings when the stock price started to fall afterwards. However, comparing to the trading statistics of the traders that learn through both individual learning and social learning as shown in Tables 6.11 to 6.13, artificial traders with no social learning generally demonstrate a much lower rate in successful

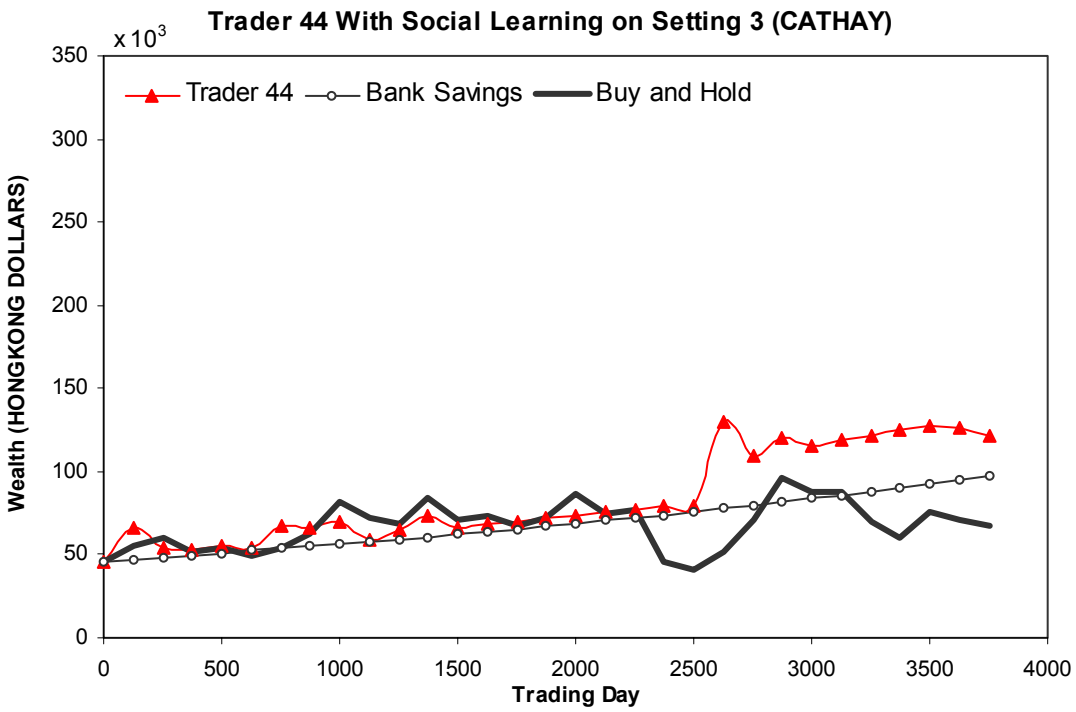


Figure 6.9 An artificial stock trader with moderate motivation for social learning under setting 3.

trades in the same period of time. It requires more time to evolve good trading strategies when the evolution of artificial traders is dependent on individual learning solo. This suggests that social learning plays an important role in the evolution of artificial agents in dynamic and incomplete environments. This observation can be further justified by the learning behaviours of trader 12 shown in Figure 6.8 and Table 6.11.

In Figure 6.8, trader 12 evolves through both individual learning and social learning. There is also a strong pressure on trader 12 that encourages the trader to learn from other successful traders in the market through social learning. As shown in Table 6.11, there are some sudden jumps in the values of successful trades, such as from day 1750

Day	Successful Trade (Daily)	Successful Trade (Weekly)	Successful Trade (Monthly)	Social Learning
1500	34.4%	32.0%	31.2%	Copied a strategy from the central pool
1625	36.8%	32.8%	31.2%	Satisfied, keep on going
1750	36.0%	34.4%	30.4%	Satisfied, keep on going
1875	38.4%	35.2%	32.0%	Satisfied, keep on going
2000	39.2%	35.2%	28.8%	Satisfied, keep on going
2125	43.2%	38.4%	34.4%	Satisfied, keep on going
2250	47.2%	41.6%	39.2%	Satisfied, keep on going
2375	46.4%	41.6%	36.0%	Satisfied, keep on going
2500	35.2%	31.2%	30.4%	Satisfied, keep on going
2625	50.4%	49.6%	54.4%	Satisfied, keep on going
2750	33.6%	30.4%	26.4%	Copied a strategy from the central pool
2875	44.0%	40.0%	41.6%	Satisfied, keep on going
3000	39.2%	36.0%	32.8%	Satisfied, keep on going

Table 6.12 Trader 44's trading statistics with moderate motivation for social learning under setting 3.

to day 1875 and from day 2375 to day 2500. These sudden changes in the trader's performance are not caused by individual learning, but rather are results from effective social learning behaviours. Under setting 2, traders whose self assessments are below 0.9 will be required to join a social learning process where traders have the opportunities to discard their current strategies, which has performed poorly in a certain period of time, and learn other traders' successful strategies or even select a new set of market indicators to build new trading models. For example, in Figure 6.8, on day 1750, trader 12 decides to learn a strategy from the central pool after a loss. This learned strategy works well during the upturn period of the market afterwards, but fails when the market changes its direction. Trader 12 then selects a new set of market indicators through social learning on day 2125, and tries to experiment with

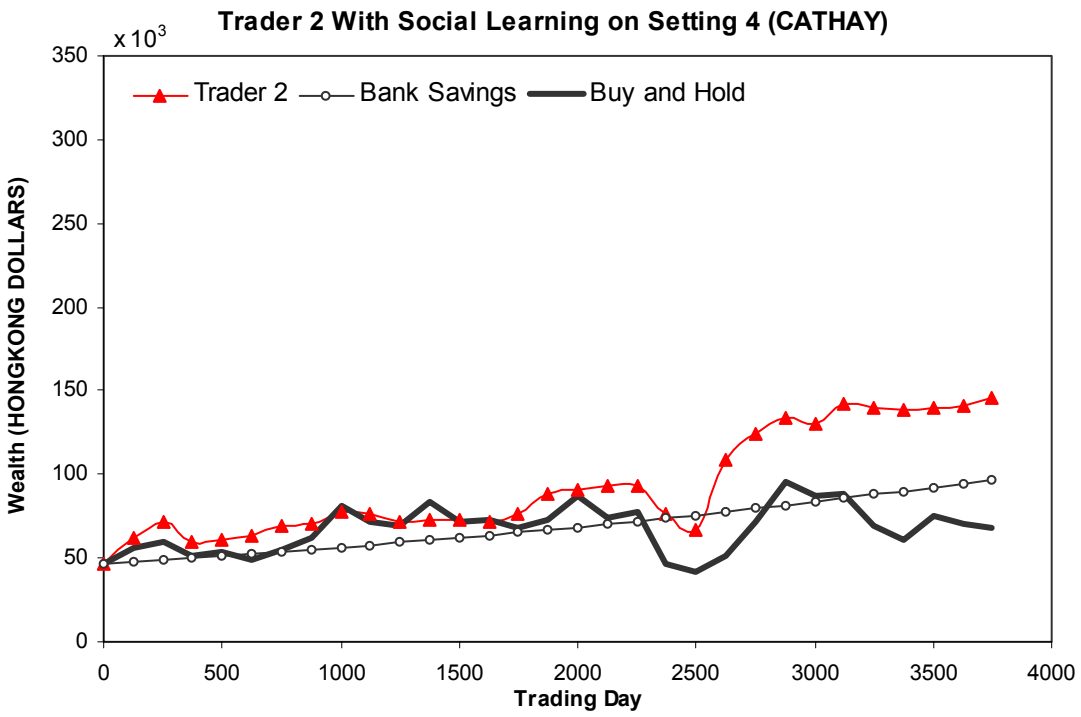


Figure 6.10 An artificial stock trader with moderate motivation for social learning under setting 4.

different sets of information. Finally, a strategy learned on day 2375 performs well during both the upturn and downturn periods of the market. As we can see, comparing to the trader 32 in Figure 6.7 with no social learning, trader 12 with social learning in Figure 6.8 demonstrates more flexibility in adapting to a changing and evolving environment. And because trader 12 is able to adapt to the changed market condition more easily and quickly, trader 12 can accumulate his wealth more quickly avoiding serious losses and subsequently higher returns on his investments. This is proven by the high rates of successful trading in the trading statistics shown in Table 6.11. On the other hand, when we set less motivation on artificial stock traders for them to learn from each other through social learning, we observe the loss of adaptability of artificial traders as shown in Tables 6.12 and 6.13, and the degrade of performance as

Day	Successful Trade (Daily)	Successful Trade (Weekly)	Successful Trade (Monthly)	Social Learning
1500	42.4%	39.2%	38.4%	Satisfied, keep on going
1625	45.6%	40.0%	40.0%	Satisfied, keep on going
1750	46.4%	42.4%	39.2%	Satisfied, keep on going
1875	44.0%	42.4%	40.0%	Satisfied, keep on going
2000	49.6%	43.2%	41.6%	Satisfied, keep on going
2125	49.6%	45.6%	43.2%	Satisfied, keep on going
2250	52.0%	40.8%	37.6%	Satisfied, keep on going
2375	41.6%	32.8%	30.4%	Satisfied, keep on going
2500	34.4%	24.0%	20.8%	Copied a strategy from the central pool
2625	70.4%	66.4%	68.8%	Satisfied, keep on going
2750	69.6%	64.0%	70.4%	Satisfied, keep on going
2875	77.6%	65.6%	63.2%	Satisfied, keep on going
3000	64.0%	59.2%	56.0%	Satisfied, keep on going

Table 6.13 Trader 2's trading statistics with moderate motivation for social learning under setting 4.

shown in Figures 6.9 and 6.10. Therefore, we conclude that social learning plays an important role in ensuring the adaptability of artificial agents in an incomplete environment that keeps on changing and evolving.

As a summary, in chapter 5, we analysed the importance of social learning through the examination of the micro trading behaviours of artificial traders. In this section, we experimented with learning algorithms that employ social learning to different extents. Through these experiments, we draw the conclusion that the social learning mechanism plays an important role in enabling the agents to escape from non-promising search spaces which are limited by the set of information they perceived from incomplete environments, and explore new search spaces constructed on new

information, or other promising search spaces which have been well explored by other agents in the environment.

6.6 Summary

In this chapter, we have illustrated the concept of imperfect evolutionary systems by using an imperfect evolutionary market. We implemented the incomplete problem space of artificial traders by means of gradually introducing new information into the evolutionary market. We experimented with the imperfect evolutionary market in various scenarios and achieved very promising results. The integrated individual and social learning paradigm has been shown to be effective in modelling co-evolution of imperfect individuals in dynamic and incomplete environments. The ISP paradigm is also successful in modelling the evolutionary process of disseminating in a society and the generation of new knowledge from the newly emerging information. In other words, the individual and social learning mechanism ensures the continuous learning and hence the adaptability and creativity of intelligent individuals in an incomplete environment that constantly evolves. We also studied the roles of individual learning and social learning in various experimental settings.

In Chapter 4, we used two examples to explain what imperfect evolutionary systems are. The second example we used described a simulated brain with specific functional sections in the context of game playing. This is a possible future direction for the work

presented here. We propose that it is possible to investigate using game playing as a test bed for the implementation of imperfect evolutionary functional sections.

Chapter 7

Conclusions and Future Work

This chapter summarises the research work and contributions from this thesis into two major sections: (1) imperfect evolutionary systems as a solution to the problem of how artificial intelligence response to new challenges from an incomplete environment by means of continuously learning and adaptation; (2) artificial intelligence for computational modelling of financial markets. The chapter also presents recommendations for future work.

7.1 Conclusions

The main aim of this thesis is to introduce imperfect evolutionary systems as a new approach for solving the problem of how artificial intelligence response to new challenges from an incomplete environment that is constantly changing and evolving. By introducing the concept of imperfectness into AI, we hope to bring people a new perspective on artificial intelligence. We conduct the research by carrying out a broad and comprehensive literature review on different areas that are closely related to AI including various machine learning paradigms, computer game playing, evolutionary computation and artificial neural networks. As a critical part of artificial intelligence,

machine learning develops effective learning algorithms that enable machines to learn from experience, and subsequently determines the adaptiveness of machine intelligence. Different machine learning paradigms make use of different representations of the real world and accordingly specific methods in automated reasoning and learning, which puts more or less constraints on the problems they are able to represent or learn. Artificial neural networks are a unique representation of intelligence that aims to emulate the organisation of human intelligence in human brains, i.e. a decentralised representation of knowledge in the form of weights and connections from a network of simple processing units. Evolutionary algorithms differ from other learning algorithms in that they are population based learning heuristics and are able to discover solutions that are even currently unknown to humans. The combination between artificial neural network and evolutionary learning, namely, the evolutionary artificial neural networks, enabled a computer program, *Blondie24*, to learn to play checkers without pre-injected human knowledge. However, although the creation of *Blondie24* can be regarded as an intelligent learning process, but as a perfect end product, *Blondie24* itself is not intelligent in terms of adaptability and creativity. We argued that the problem lies in the conventional machine learning methodologies where the aim of artificial intelligence research is to develop a perfect end product, rather than an adaptive learning entity. The problem also lies in the ignorance of an intelligent entity's membership to its environment. We say human intelligence is imperfect because the environments humans live in are so complex that everyone has a different and partial view of the world. If human intelligence is imperfect, so is artificial intelligence.

An imperfect evolutionary system is defined as a system where intelligent entities optimise their own utilities with resources available whilst adapting themselves to the new challenges from an evolutionary incomplete environment. Here we introduce the imperfect evolutionary system as a new research methodology for the design of artificial intelligence in the following sense: an intelligent individual, e.g. a humanoid or an artificial agent in a computer model, must be imperfect or incomplete due to the fact that it can only understand its environment partially because of its complexity; the environment that an intelligent individual lives in must also be incomplete because the environment itself is also evolutionary due to the fact that there are always new challenges emerge, e.g. new techniques invented by its members. The aim of the artificial intelligence research presented here is not to create one ultimate perfect super intelligence, but to create heterogeneous individuals with imperfect intelligence who respond to new challenges from their environments by means of continuous learning and also contribute to the evolution of their societies, i.e., being creative.

An integrated individual and social learning paradigm is described as a general framework for designing imperfect evolutionary systems. We used the stock market as a test bed and implemented an imperfect evolutionary market. In the imperfect evolutionary market, imperfect stock traders who perceive different information sets from the market co-evolve and learn to trade stocks profitably. By gradually injecting new information into the imperfect evolutionary market, we demonstrated how new information is successfully absorbed by imperfect artificial traders and how new knowledge is successfully created and disseminated within the imperfect market. In

chapter 5 and 6, we also analyse the importance of the social learning process in the integrated individual and soical learning paradigm through the careful examination of the micro trading behaviours of artificial traders. We conclude that social learning plays an important role in ensuring the adaptability of artificial agents in an incomplete environment that keeps on changing and evolving.

In conclusion, although there are many debates on what AI should be. We adhere to the opinion that AI is about the understanding of human intelligence and hence its replication in machine forms. Imperfect evolutionary systems changes AI from a perfect paradigm to an imperfect paradigm, and provides a new perspective on the understanding of adaptability and creativity in human intelligence. We believe, as described by Marvin Minsky's *society of minds*, intelligence is not about isolation, but interaction and co-evolution of many imperfect individuals. To develop an intelligent robot is not about developing a machine that performs routine jobs but one interacts and continuously learns from its environments. The study on imperfect evolutionary systems has shown a new perspective for AI research.

In this thesis, we also investigate the application of artificial neural networks in modelling artificial stock markets. In agent-based computational economics, artificial stock markets play an important role in studying major time series from real work markets and testing of major economic theories. Both learning classifier systems and genetic programming have been widely applied in modelling stock markets as multi-agent systems. In this thesis, we employed evolutionary artificial neural networks and

developed a multi-agent based simulated stock market. In our simulated stock market, artificial traders developed successful trading strategies that beat the major market benchmarks and risk-free investments. Our artificial stock traders also demonstrated rich and dynamic trading behaviours that closely mimic real-world traders.

7.2 Future Work

Although we present a complete thesis here there is always scope to take the ideas further. Below we present some ideas for possible future research directions:

- Imperfect Evolutionary Systems and Game Playing

  Is it possible to develop a computer program that can learn to play different games by 'himself' depending on whomever he meets? Let us call the program "*Hakuna Matata (H&M)*". One day, Su comes to *H&M* and they play tic-tac-toe. The next day, Graham asks *H&M* to join a poker game. *H&M* could be a good learner, could be a bad learner. *H&M* is not designed to play a specific game, but rather he learns from the environment, or more precisely the people, surrounds him. We have seen quite a number of computer game programs developed in the field of AI, and similarly, they claimed themselves successful because their programs have played games to a level of human experts, or even defeat the human champions. But how many world chess champions, or checkers experts are there? Does the creation of a computer program that can play a game better than human mean the

creation of intelligence? We doubt it. Imperfect evolutionary systems might not be able to answer all those questions and doubts, however, it shows us a new angel to think about intelligence, a new methodology in designing intelligence. We propose extending the imperfect evolutionary system paradigm into the simulation of specialised functional sections for game playing, potentially a better understanding on the way human brain works.

- ISP learning paradigm and Cultural Algorithm

A number of problems need to be solved when implementing *Hakuna Matata*. How does an imperfect evolutionary system create a new specialised functional section for a new task? How do we deal with the interaction among specialised functional sections? How do we differentionate information? What will happen when there is more than one objective function in one system? We believe that the ISP learning paradigm still can be improved from many aspects. One of them is the knowledge management and adaptation in the belief space from cultural algorithm can be used to improve the social learning process in the ISP algorithm. In fact, in recent years, there have been studies on extended cultural algorithms with MGA/MGP in recent years (Reynolds *et al.* 2003). The ISP learning paradigm can be extended to incorporate cultural algorithms.

- Evolutionary Artificial Neural Networks

As an important machine learning techniques, there are many potential application areas of evolutionary artificial neural networks, such as game playing, financial forecasting and modelling, other domains in artificial intelligence such as

computer vision, patter recognition and robotics. We intend to investigate the applications of evolutionary artificial neural network in various different domains.

Bibliography

Aamodt, A. and Plazas, E. "Case-based Reasoning: Foundational Issues, Methodological Variations, and System Approaches." *AI Communications*, 1994, 7(1), 39-52.

Aarts, E. H. L.; Korst, J. H. M. and van Laarhoven, P. J. M. "Simulated Annealing," in E. H. L. Aarts and J. K. Lenstra, eds., *Local search in combinatorial optimization,* John Wiley & Sons, 1997, 91-120.

Abramson, D.; Mills, G. and Perkins, S. "Parallelisation of A Genetic Algorithm for the Computation of Efficient Train Scheduels," in *Proceedings of Parallel Computing and Transputers*, 1993, 139-149.

Achelis, S. B. *Technical analysis from A-Z*. McGraw-Hill Education, 2000.

Aha, D.; Kibler, D. and Albert, M. "Instance-based Learning Algorithms." *Machine Learning*, 1991, 6, 37-66.

Akerlof, G. A. "The Market for Lemons: Quality Uncertainty and the Market Mechanism." *Quarterly Journal of Economics*, 1970, 84(3), 488-500.

Allen, J. *Natural language understanding*. CA: Benjamin Cummings, 1995.

Anderson, E. J. and Ferris, M. C. "A Genetic Algorithm for the Assembly Line Balancing Problem," in *Integer Programming and Combinatorial Optimization: Proceedings of a 1990 Conference Held at the University of Waterloo*, University of Waterloo Press, 1990, 7-18.

Anderson, J. A. and Rosenfeld, E., eds., *Neurocomputing: Foundations of research*. MA: MIT Press, 1988.

Andrew, M. and Prager, R. "Genetic Programming for the Acquisition of Double Auction Market Strategies," in K. E. Kinnear, Jr., ed., *Advances in genetic programming*. MIT Press, 1994, 355-368.

Angeline, P. J. "A Historical Perspective on the Evolution of Executable Structures." *Informaticae*, 1998, 36, 179-195.

Arnott, Richard; Greenwald, Bruce; Kanbur, Ravi and Nalebuff, Barry, eds., *Economics for an imperfect world: Essays in honor of Joseph E. Stiglitz*. MA: The MIT Press, 2003.

Arrow, Kenneth and Debreu, Gerard "Existence of a Competitive Equilibrium for a Competitive Economy." *Econometrica*, 1954, 22(3), 265-290.

Arrow, Kenneth; Dasgupta, Partha and Maler, Karl-Goran "Welfare Economics in Imperfect Economies," in R. Arnott; B. Greenwald; R. Kanbur and B. Nalebuff, eds., *Economics for an imperfect world*, MA: MIT Press, 2003, 299-330.

Arthur, W. B., Holland, J. H., Lebaron, B., Palmer, R. and Tayler, P. "Asset Pricing under Endogenous Expectations in an Artificial Stock Market**,"** in W. B. Arthur, S. Durlauf and D. Lane, eds., *The economy as an evolving complex system II.* Addison-Wesley, 1997, 15-44.

Atkeson, C. G.; Schaal, S. A. and Moore, A. W. "Locally Weighted Learning." *AI Review*, 1997, 11, 11-73.

Axelrod, R. "The Evolution of Strategies in the Iterated Prisoner's Dilemma," in L. Davis, ed, *Genetic algorithms and simulated annealing*, 1987, 32-41.

Azoff, E. Michael and Azoff, Eitan M. *Neural network time series forecasting of financial markets*. NY: John Wiley & Sons, 1994.

Bäck, T. and Schwefel, H. P. "An Overview of Evolutionary Algorithms for Parameter Optimisation." *Evolutionary Computation*, 1993, 1(1), 1-23.

Banzhaf, W.; Nordin, P.; Keller, R. E. and Francone, F. D. *Genetic programming – An introduction: On the automatic evolution of computer programs and its applications*. Morgan Kaufmann, 1998.

Bauer, R. J. Jr *Genetic algorithms and investment strategies*. NY: John Wiley & Sons, 1994.

Bauer, R. J. Jr. and Dahlquist, J. *Technical market indicators*. John Wiley & Sons, 1998.

Baxter, J. "The Evolution of Learning Algorithms for Artificial Neural Networks," in D. Green and T. Bossomaier, eds., *Complex Systems*, Amsterdam: IOS, 1992, 313-326.

Bellman, R. E. *An introduction to artificial intelligence: Can computers think?* San Francisco: Boyd&Fraser Publishing Company, 1978.

Beltratti, A.; Margarita, S. and Terna, P. *Neural networks for economic and financial modelling*. Thomson, 1996.

Bentley, J. L. "Fast Algorithms for Geometric Travelling Salesman Problems." *ORSA Journal on Computing*, 1992, 4(4), 387-411.

Bezdek, J. C. "Fuzzy Models - What Are They, and Why?" *IEEE Transactions on Fuzzy Systems*, 1993, 1(1), 1-6.

Billings, D.; Davidson, A., Schaeffer, J. and Szafron, D. "The Challenge of Poker." *Artificial Intelligence Journal*, 2002, 134(1-2), 201-240.

Billings, D.; Burch, N.; Davidson, A.; Holte, R.; Schaeffer, J.; Schauenberg, T. and Szafron, D. "Approximating Game-Theoretic Optimal Strategies for Full-Scale Poker," in *Proceedings of the Eighteenth international conference on artificial intelligence*, 2003, 661-668.

Billings, S. A. and Zheng, G. L. "Radial Basis Function Network Configuration Using Genetic Algorithms." *Neural Networks*, 1995, 8(6), 877-890.

Bodie, Z.; Kane, A. and Marcus, A. J. *Investments*. McGraw-Hill, 2002.

Bonabeau, E.; Dorigo, M. and Theraulaz, G. *Swarm intelligence: From natural to artificial systems*. Oxford University Press, 1999.

Bondt, W. D. and Thaler, R. "Further Evidence on Investor Overreaction and Stock Market Seasonality." *Journal of Finance*, 1987, 62, 557-580.

Booker, L. B. "Intelligent Behavior as an Adaptation to the Task Environment," Ph. D. dissertation, Department of Computer and Communications Science, University of Michigan, Ann Arbor, 1982.

Bornholdt, S. and Graudenz, D. "General Asymmetric Neural Networks and Structure Design by Genetic Algorithms." *Neural Networks*, 1992, 5(2), 327-334.

Brachman, Ronald and Levesque, Hector. *Knowledge representation and reasoning*. Morgan Kaufmann, 2004.

Braun, H. C. "On Solving Travelling Salesman Problems by Genetic Algorithms," in H. P. Schwefel and R. Manner, eds, *Parallel problem solving from nature*. Berlin: Springer-Verlag, 1990, 129-133.

Bremermann, H. J. "The Evolution of Intelligence - The Nervous System as a Model of its Environment." Technical Report No. 1, Contract No. 477(17), Dept. of Mathematics, Univ. of Washington, Seattle, 1958.
Brill, Eric and Mooney, Raymond J. "An Overview of Empirical Natural Language Processing." *AI Magazine*, 1997, 18(4), 13-24.

Bull, L., ed. *Applications of learning classifier systems*. Springer-Verlag, Series: Studies in Fuzziness and Soft Computing, Vol. 150, 2004.

Burke, E. K.; MacCarthy, B.; Petrovic, S. and Qu, R. "Structured Cases in Case-Based Reasoning - Re-using and Adapting Cases for Time-tabling Problems." *Knowledge-Based Systems*, 2000, 13(2-3), 159-165.

Cahill, M. *An investor's guide to analysing companies and valuing shares*. FT Prentice Hall, 2003.

Callan, R. *The essence of neural networks*. Prentice Hall, 1999.

Campbell, Murray and Marsland, T. Anthony. "A Comparison of Minimax Tree Search Algorithms." *Artificial Intelligence*. 1983, 20(4), 347-367.

Campbell, Murray; Hoane, A. Joseph, Jr.; Hsu, Feng-hsiung. "Deep Blue." *Artificial Intelligence*, 2002, 134(1-2), 57-83.

Cantu Paz, E. "A Survey of Parallel Genetic Algorithms," *Calculateurs Parallels, Resequx et Systems Repartis*, 1998, 10(2), 141-171.

Cantu Paz, E. *Efficient and accurate parallel genetic algorithms*. Kluwer: Genetic algorithms and evolutionary computation volume 1, 2000.

Caudell, T. P. and Dolan, C. P. "Parametric Connectivity: Training of Constrained Networks Using Genetic Algorithms," in J. D. Schaffer, ed, *Proceedings of the 3rd Internal Conference on Genetic Algorithms and Their Applications*. CA: Morgan Kaufmann, 1989, 370-374.

Chalmers, D. J. "The Evolution of Learning: An Experiment in Genetic Connectionism," in *Proceedings of Connectionist Models Summer School*, 1990, 81-90.

Chellapilla, K. and Fogel, D. B. "Evolving an Expert Checkers Playing Program without Using Human Expertise." *IEEE Transactions on Evolutionary Computation*, 2001, 5(4), 422-428.

Chen, S. H. "Fundamental Issues in the Use of Genetic Programming in Agent-based Computational Economics," in *Proceedings of the first international workshop on agent-based approaches in economic and social complex systems*, 2001, 175-185.

Chen, S. H., ed. *Evolutionary computation in economics and finance*. NY: Physica-Verlag, Studies in Fuzziness and Soft Computing, Vol. 100, 2002.

Chen, S. H., ed. *Genetic algorithms and genetic programming in computational finance*. Kluwer Academic Publishers, Norwell, MA, 2002a.

Chen, S. H. and Wang, P., eds., *Computational intelligence in economics and finance*. Springer-Verlag, 2003.

Chen, S. H. and Liao, C. C. "Price Discovery in Agent-based Computational Modelling of Artificial Stock Market," in *Proceedings of the second asia-pacific conference on genetic algorithms and application*, 2000, 380-387.

Chen, S. H. and Lu, C. F. "Would Evolutionary Computation Help in Designs of Artificial Neural Nets in Forecasting Financial Time Series?" in *Proceedings of the congress on evolutionary computation*, 1999, 267-274.

Chen, S. H. and Yeh, C. H. "Genetic Programming Learning and the Cobweb Model," in P. Angeline, ed., *Advances in genetic programming*. MA: MIT Press, 1996, 443-466.

Chen, S. H. and Yeh, C. H. "Modelling Speculators with Genetic Programming," in P. Angeline; R. G. Reynolds; J. R. McDonnell and R. Eberhart, eds., *Evolutionary programming VI*, LNCS, Vol. 1213. Berlin: Springer, 1997, 137-147.

Chen, S. H. and Yeh, C. H. "Evolving Traders and Business School with Genetic Programming: A New Architecture of the Agent-based Artificial Stock Market." *Journal of Economics and Control*, 2001, 25(3-4), 363-393.

Chen, S. H. and Yeh, C. H. "Towards an Integration of Social Learning and Individual Learning in Agent-based Computational Stock Markets: The Approach Based on Population Genetic Programming." *Journal of Management and Economics*, 2001a, 5(5), 1-13.

Chen, S. H. and Yeh, C. H. "On the Emergent Properties of Artificial Stock Markets: The Efficient Market Hypothesis and the Rational Expectations Hypothesis." *Journal of Economic Behavior & Organization*, 2002, 49(2), 217-239.

Chen, S. H.; Yeh, C. H. and Lee, W. C. "Option Pricing with Genetic Programming," in *Proceedings of the third annual genetic programming conference*, Morgan Kaufmann Publishers, 1998, 32-37.

Chen, S. H.; Liao, C. C. and Yeh, C. H. "On AIE-ASM: A Software to Simulate Artificial Stock Market with Genetic Programming," in S. H. Chen, ed., *Evolutionary computation in economics and finance*. NY: Physica-Verlay, Studies in Fuzziness and Soft Computing, Vol. 100, 2002, 107-122.

Chenoweth, T.; Obradovic, Z. and Lee, S. S. "Embedding Technical Analysis into Neural Network Based Trading Systems." *Applied Artificial Intelligence*, 1996, 10(6), 523-542.

Chidambaran, N.; Triqueros, J. and Lee, C. W. "Option Pricing Via Genetic Programming," in S. H. Chen, ed., *Evolutionary computation in economics and finance*. NY: Physica-Verlag, Studies in Fuzziness and Soft Computing, Vol. 100, 2002, 384-397.

Clark, D. "Deep Thoughts on Deep Blue." *IEEE Expert*, 1997, 12(4), 31.

Cockshott, A. R. and Hartman, B. E. "Improving the Fermentation Medium for Echinocandin B Production, Part II: Particle Swarm Optimisation." *Process Biochemistry*, 2001, 36, 661-669.

Coley, D. A. *An introduction to genetic algorithms for scientists and engineers*. World Scientific Publishing, 1999.

Cooper, G. and Herskovits, E. "A Bayesian Method for the Induction of Probabilistic Networks from Data." *Machine Learning*, 1992, 9, 309-347.

Cover, T. and Hart, P. "Nearest Neighbor Pattern Classification." *IEEE Transactions on Information Theory*, 1967, 13, 21-27.

Davis, M.; Liu, L. and Elias, J. G. "VLSI Circuit Synthesis Using a Parallel Genetic Algorithm," in *Proceeding of the first IEEE Conference on Evolutionary Computation*, 1994, 104-109.

Dean, T.; Basye, K. and Shewchuk, J. "Reinforcement Learning for Planning and Control," in S. Minton, ed., *Machine learning methods for planning*. San Francisco: Morgan Kaufmann, 1993, 67-92.

De Jong, K. A. "Learning with Genetic Algorithms: An Overview." *Machine Learning*, 1988, 3, 121-138.

De Jong, K. A.; Spears, W. M. and Gordon, D. F. "Using Genetic Algorithms for Concept Learning." *Machine Learning*, 1993, 13, 161-188.

Dengiz, B.; Altiparmak, F. and Smith, A. E. "Local Search Genetic Algorithm for Optimal Design of Reliable Network." *IEEE Transactions on Evolutionary Computation*, 1997, 1(3), 179-188.

Di Caro, G. and Dorigo, M. "AntNet: Distributed Stigmergetic Control for Communications Networks." *Journal of Artificial Intelligence Research*, 1998, 9, 317-365.

Di Caro, G. and Dorigo, M. "Two Ant Colony Algorithms for Best-effort Routing in Datagram Networks," in *Proceedings of the Tenth IASTED International Conference on Parallel and Distributed Computing and Systems*, 1998a, 541-546.

Davis, L. *Handbook of genetic algorithms*. Van Nostrand Reinhold, 1991.

Dorffner, G. "Neural Networks for Time Series Processing." *Neural Network World*, 1996, 6(4), 447-468.

Dorigo, M. "Optimization, Learning and Natural Algorithms." Ph. D. Thesis (in Italian), Dip. Elettronica, Politecnico di Milano, Italy, 1992.

Dorigo, M.; Maniezzo, V. and Colorni, A. "The Ant System: Optimisation by a Colony of Cooperating Agents." *IEEE Transactions on Systems, Man, and Cybernetics*, 1996, 26(1), 29-41.

Dorigo, M. and Gambardella, L. M. "Ant Colony System: A Cooperative Learning Approach to the Travelling Salesman Problem." *IEEE Transactions on Evolutionary Computation*, 1997, 1(1), 53-66.

Dorigo, M. and Di Caro, G. "The Ant Colony Optimization Meta-Heuristic," in D. Corne, M. Dorigo and F. Glover, eds., *New ideas in optimisation*. McGraw-Hill, 1999, 11-32.

Dorigo, M. and Stutzle, T. *Ant colony optimisation*. MIT Press, 2004.

Dowsland, K. A. "Simulated Annealing," in C. R. Reeves, ed., *Modern heuristics techniques for combinatorial problems*. McGraw-Hill, 1995, 20-69.

Dudani, S. A. "The Distance Weighted K-Nearest-Neighbor Rule." *IEEE Transactions on Systems, Man and Cybernetics*, 1975, SMC-6, 4, 325-327.

Eberhart, R. C. and Kennedy, J. "Particle Swarm Optimisation," in *Proceedings of IEEE International Conference on Neural Networks*, 1995, 1942-1948.

Eberhart, R. C. and Shi, Y. "Evolving Artificial Neural Networks" in *Proceedings of International Conference on Neural Networks and Brain*, 1998, PL5-PL13.

Eberhart, R. C.; Kennedy, J. and Shi, Y. *Swarm intelligence*. San Francisco: Morgan Kaufmann Publishers, 2001.

Elman, J. L., "Finding Structure in Time." *Cognitive Science*, 1990, 14, 179-211.

Fama, E. F. "The Behaviour of Stock Market Prices." *Journal of Business*, 1965, 34-105.

Fama, E. F. "Efficient Capital Markets: A Review of Theory and Empirical Work." *Journal of Finance*, 1970, 383-417.

Fang, J. and Xi, Y. G. "Neural Network Design Based on Evolutionary Programming." *Artificial Intelligence in Engineering*, 1997, 11(2), 155-161.

Fausett, L. *Fundamentals of neural networks: Architectures, algorithms, and applications*. NJ: Prentice Hall, 1994.

Eglese, R. W. "Simulated Annealing: A Tool For Operational Research." *European Journal of Operational Research*, 1990, 46, 271-281.

Fogel, D. B. "Evolving Artificial Intelligence." Ph. D. Thesis, UCSD, 1992.

Fogel, D. B. "Using Evolutionary Programming to Create Neural Networks That Are Capable of Playing Tic-Tac-Toe," in *Proceeding of IEEE International Conference on Neural Networks.* NJ: IEEE Press, 1993, 875-880.

Fogel, D. B. "An Introduction to Simulated Evolutionary Optimisation." *IEEE Transactions on Neural Networks*, 1994, 5(1), 3-14.

Fogel, D. B.; Wasson, E. C. And Boughton, E. M. "Evolving neural networks for detecting breast caner." *Cancer Letters*, 1995, 96(1), 49-53.

Fogel, D. B. *Evolutionary computation: Toward a new philosophy of machine intelligence* (Second edition). NJ: IEEE Press, 2000.

Fogel, D. B. *Blondie24, Playing at the edge of AI*. Morgan Kaufmann Publishers, 2002.

Fogel, L. J.; Owens, A. J. and Walsh, M. J. *Artificial intelligence through simulated evolution*. NY: John Wiley, 1966.

Forsyth, David A. and Ponce, Jean. *Computer vision: A modern approach*. Prentice Hall, 2002.

Fraser, A. S. "Simulation of Genetic Systems By Automatic Digital Computers. II. Effects of Linkage on Rates Under Selection." *Australian Journal of Biol. Sci.*, 1957, 10, 492-499.

Fraser, A. S. 1960. "Simulation of Genetic Systems By Automatic Digital Computers. IV. Epistatis." *Australian Journal of Biol. Sci.*, 1960, 13, 329-346.

Fu, LiMin. *Neural networks in computer intelligence*. McGraw-Hill.

Garis, H. "GenNets: Genetically Programmed Neural Nets Using the Genetic Algorithm to Train Neural Nets Whose Inputs and/or Output Vary in Time," in *Proceedings of IEEE International Conference on Neural Networks*, 1991, 1391-1396.
Giarratano, J. and Riley, G. *Expert systems: Principles and programming* (Third edition). PWS Publishing Company, 1998.

Glover, F. "Future Paths for Integer Programming and Links to Artificial Intelligence." *Computers and Operations Research*, 1986, 5, 533-549.

Glover, F. and Laguna, M. "Tabu Search," in C. R. Reeves, ed., *Modern heuristics techniques for combinatorial problems*. McGraw-Hill, 1995, 70-150.

Goldberg, D. *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley, 1989.

Gough, L. *How the stock market really works*. Pearson Education: Financial Times Prentice Hall, 2001.

Greenwood G. W. "Training Partially Recurrent Neural Networks Using Evolutionary Strategies." *IEEE Transactions on Speech Audio Processing*, 1997, 5, 192-194.

Grefenstette, J. J. "Credit Assignment in Rule Discovery Systems Based on Genetic Algorithms." *Machine Learning*, 1988, 3, 225-245.

Grefenstette, J. J.; Gopal, R.; Rosimaita, B. and Gucht, D. V. "Genetic Algorithms for the Traveling Salesman Problem," in *Proceedings of the 2nd International Conference on Genetic Algorithms and their Applications*, Carnegie Mellon publishers, 1985, 160-168.

Grönroos, M.; Whitley, D. and Pyeatt, L. "A Comparison of Some Methods for Evolving Neural Networks," in *Proceedings of Genetic and Evolutionary Computation Conference*, 1999, 1442-1449.

Grossman, J. S. and Stiglitz, J. E. "Information and Competitive Price Systems." *American Economic Review*, 1976, 66(2), 246-253.

Grossman, J. S. and Stiglitz, J. E. "On the Impossibility of Informationally Efficient Markets." *American Economic Review*, 1980, 70, 393-408.

Gruau, F. "Automatic Definition of Modular Neural Networks." *Adaptive Behaviour*, 1994, 3, 151-183.

Gustafson, S. M. "An Analysis of Diversity in Genetic Programming." Ph. D. Dissertation, the University of Nottingham, Nottingham, U.K., 2004.

Gustafson, S.; Ekart, A.; Burke, E. K. and Kendall G. "Problem Difficulty and Code Growth in Genetic Programming," in *Genetic programming and evolvable machines*, Kluwer, 2004, 5(3), 271-290.

Harley, E. "Book Review: Blondie 24, Playing at the Edge of AI." *The IEEE Computational Intelligence Bulletin*, 2002, 1(1), 25-27.

Harp, S. A.; Samad, T. and Guha, A. "Designing Application-specific Neural Networks Using the Genetic Algorithm," in D. S. Tourezky, ed., *Advances in Neural Information Proceesing Systems 2*, CA: Morgan Kaufmann, 1990, 447-454.

Harrald, P. "Economics and Evolution." The panel paper given at the Seventh International Conference on Evolutionary Programming, March 25-27, San Diego, U.S.A, 1998.

Hart, W. E. *Adaptive global optimization with local search*. Ph.D. Thesis, University of California, San Diego, 1994.

Hancock, P. J. B. "Genetic Algorithms and Permutation Problems: A Comparison of Recombination Operators for Neural Net Structure Specification," in D. Whitley and J. D. Schaffer, eds., *Proceeding of International Workshop Combinations of Genetic Algorithms and Neural Networks*, CA: IEEE Computer Society, 1992, 108-122.

Hauser, R. and Manner, R. "Implementation of Standard Genetic Algorithm on MIMD Machines," in Y. Davidor, H. P. Schwefel, and R. Manner, eds., *Parallel Problem Solving from Nature, PPSN III*. Berlin: Springer-Verlag, 1994, 504-513.

Heckerman, D. "A Tutorial on Learning with Bayesian Networks," in M. I. Jordan, ed., *Learning in graphical models.* Kluwer, 1998.

Heimes, F.; Zalesski, G. Z.; Land, W. and Oshima, M. "Traditional and Evolved Dynamic Neural Networks for Aircraft Simulation," in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, 1997, 1995-2000.

Henrion, M. "Propagation of Uncertainty in Bayesian Networks by Probabilistic Logic Sampling," in J. F. Lemmer and L. N. Kanal, eds., *Uncertainty in artificial intelligence 2*. Elsevier, 1988, 149-163.

Holland, J. H. "outline for a Logical Theory of Adaptive System." *Journal of the Association for Computing*, 1962, 3, 297-314.

Holland, J. H. *Adaptation in natural and artificial systems.* University of Michigan Press, 1975.

Holland, J. H. "Using Classifier Systems to Study Adaptive Nonlinear Networks," in D. L. Stein, ed, *Lectures in the science of complexity*, CA: Addison-Wesley, 1988, 463-499.

Holland, J. H. *HIDDEN ORDER – How adaptation builds complexity*. MA: Perseus Books, 1995.

Hoos, H. and Stutzle, T. *Stochastic local search: Foundations and applications*. Morgan Kaufmann, 2004.

Hopfield, J. J. "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," in *Proceeding of National Academy of Sciences*, 1982, 74, 2554-2558.

Howard, M. "The Evolution of Trading Rules in an Artificial Stock Market," in *Computing in Economics and Finance 1999*, series by Society for Computational Economics, No. 712, 1999.

Igel, C. and Stagge, P. "Graph Isomorphisms Affect Structure Optimization of Neural Networks," In *International Joint Conference on Neural Networks*, 2002, 142-147.

Jackson, Peter. *Introduction to expert systems* (Third edition). MA: Addison-Wesley, 1999.

Jin, X. and Reynolds, R. G. "Data Mining Using Cultural Algorithms and Regional Schemata," in *Proceedings of 14th IEEE International conference on Tools with Artificial Intelligence*, 2002, 33-40.

Jones, Joseph L. and Flynn, Anita M. *Mobile robots: Inspiration to implementation*. Massachusetts: A K Peters, 1993.

Jurafsky, Daniel and Martin, James H. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Prentice-Hall, 2000.

Kaelbling, L. P.; Littman, M. L. and Moore, A. W. "Reinforcement Learning: A Survey." *Journal of AI Research*, 1996, 4, 237-285.

Kaboudan, M. "GP Forecasts of Stock Prices for Profitable Trading," in S. H. Chen, ed., *Evolutionary computation in economics and finance*. NY: Physica-Verlag, Studies in Fuzziness and Soft Computing, Vol. 100, 2002, 359-379.

Kaindl, H. "Tree Searching Algorithms," in T. A. Marsland and J. Schaeffer, eds., *Computers, Chess, and Cognition*, NY: Springer-Verlag, 1990, 133-158.

Keber, C. "Option Pricing with the Genetic Programming Approach." *Journal of Computational Intelligence in Finance*, 1999, 7(6), 26-36.

Kendall, G. and Hingston, P. "Learning versus Evolution in Iterated Prisoner's Dilemma," in *Proceedings of the Congress on Evolutionary Computation*, 2004.

Kendall, G. and Su, Y. "Co-evolution of Successful Trading Strategies in A Multi-agent Based Simulated Stock Market," in *Proceedings of the 2003 international conference on machine learning and applications*, 2003, 200-206.

Kendall, G. and Su, Y. "A Multi-agent Based Simulated Stock Market: Testing on Different Types of Stocks," in *Proceedings of the congress on evolutionary computation*, 2003a, 2298-2305.

Kendall, G. and Su, Y. "Learning with Imperfections - A Multi-Agent Neural-Genetic Trading Systems with Differing Levels of Social Learning." in *Proceedings of the IEEE Conference on Cybernetic and Intelligent Systems*, 2004, 47-52.

Kendall, G. and Su, Y. "Imperfect Evolutionary Systems." *IEEE Transactions on Evolutionary Computation*. (Submitted). 2006.

Kendall, G. and Whitwell, G. "An Evolutionary Approach for the Tuning of a Chess Evaluation Function using Population Dynamics," in P*roceedings of Congress on Evolutionary Computation*, 2001, 995-1002.

Kendall, G.; Yaakob, R. and Hingston, P. "An Investigation of an Evolutionary Approach to the Opening of Go," in *Proceedings of the 2004 Congress on Evolutionary Computation*, 2004.

Kamijo, K. and Tanigawa, T. "Stock Price Pattern Recognition: A Recurrent Neural Network Approach," in R. R. Trippi and E. Turban, eds., *Neural networks in finance and investing*, Chicago: Probus, 1993, 357-370.

Kimoto, T; Asakawa, K.; Yoda, M. and Takeoka, M. "Stock Market Prediction System with Modular Neural Networks," in *Proceedings of the international joint conference on neural networks*, 1990, 1-6.

King, D. *Kasparov v Deeper Blue: The Ultimate Man v Machine Challenge.* Badsford, 1997.

Kirkpatrick, S.; Gelatt, C. D. and Vecchi, M. P. "Optimization by Simulated Annealing." *Science*, 1983, 671-680.

Kitano, H. "Designing Neural Networks Using Genetic Algorithm with Graph Generation System." *Complex Systems*, 1990, 4, 461-476.
Knowles, J. D. and Corne, D. W. "M-paes: A Memetic Algorithm for Multiobjective Optimisation," in *Proceedings of the Congress on Evolutionary Computation*, 2000, 325-332.

Kohonen, T. *Self-organizing maps*, second edition. Berlin: Springer-Verlag, 1997.

Kolodner, Janet L. *Case-based reasoning*. CA: Morgan Kaufmann, 1993.

Konar, Amit. *Artificial intelligence and soft computing – behavioral and cognitive modelling of the human brain*. CRC Press, 2000.

Koza, J. R. *Genetic programming: On the programming of computers by means of natural selection*. MA: MIT Press, 1992.

Koza, J. R. "Evolution of a Subsumption Architecture That Performs a Wall Following Task for an Autonomous Mobile Robot Via Genetic Programming," in T. Petsche, ed., *Computational learning theory and natural learning systems*. MA: MIT Press, 1994, 321-346.

Koza, J. R.; Andre, D.; Bennett, F. H.; Keane, M. *Genetic programming III: Darwinian invention and problem solving*. Morgan Kaufman, 1999.

Koza, J. R.; Keane, M.; Streeter, M.; Mydlowec, W.; Yu, J. and Ianza, G. *Genetic programming IV: Routine human-competitive machine intelligence*. Kluwer Academic Publisers, 2003.

Koza, J. R. and Rice, J. P. "Genetic generation of both the weights and architecture for a neural network," in *Proceedings of IEEE Joint Conference on Neural Networks*, 1991, 397-404.

Krasnogor, N. "Memetic Algorithms." A tutorial given in the *Seventh International Conference on Parallel Problem Solving from Nature (PPSN VII)*. [Online]. Available: www.cs.nott.ac.uk/~nxk. September, 2002.

Krasnogor, N. and Pelta, D. A. "Fuzzy Memes in Multimeme Algorithms: A Fuzzy-Evolutionary Hybrid," in J. L. Verdegay, ed., *Fuzzy sets based heuristics for optimization*, Verdegay Physica Verlag, 2002.

Krasnogor, N. and Smith, J. "A Memetic Algorithm With Self-Adaptive Local Search: TSP As a Case Study," in *Proceedings of the International Genetic and Evolutionary Computation Conference,* 2000, 987-994.

Kristinsson, K. and Dumont, G. A. "System Identification and Control Using Genetic Algorithms." *IEEE Transactions on Systems, Man and Cybernetics*, 1992, 22(5), 1033-1046.

Kroger, B.; Schwenderling, P. and Vornberger, O. "Parallel Genetic Packing on Transputers," in J. Stender, ed, *Parallel genetic algorithms: Theory and applications*. IOS Press, 1993, 151-185.

Kwon, Y. K. and Moon, B. R. "Daily Stock Prediction Using Neuro-Genetic Hybrids," in *Proceedings of genetic and evolutionary computation conference*, Berlin: Springer-Verlag, LNCS 2724, 2003, 2203-2214.

Larrhoven, P. J. M. and Aarts, E. H. L. *Simulated annealing: Theory and applications*. Dordrecht: Kluwer, 1988.

Langdon, W.; Soule, T.; Poli, R. and Foster, J. "The Evolution of Size and Shape," in L. Spector *et al.*, eds., *Advances in genetic programming 3*, MA: MIT Press, 1999, 163-190.

Lavigne, S. "Modelling an Artificial Stock Market: When Cognitive Institutions Influence Market Dynamics." Economics and Social Research Group, Universite Montesquieu-Bordeaux IV, France. Working Papers No. 2004-04, 2004.

Lazar, A. and Reynolds, R.G. "Evolution-Based Learning of Ontological Knowledge for a Large-Scale Multi-Agent Simulation," in M. Grana, R.J. Duro, A.D. Aryou, and P.P Wang, eds., *Information processing and evolutionary algorithms-from industrial applications to academic speculation*, Berlin, Springer-Verlag, 2005.

LeBaron, B. Arthur, W. B. and Palmer, R. G. "Time Series Properties of an Artificial Stock Market Model." *Journal of Economic Dynamcis and Control*, 1999, 23, 1487-1516.

LeBaron, B. "Agent Based Computational Finance: Suggested Readings and Early Research." *Journal of Economic Dynamics and Control*, 2000, 24(5-7), 679-702.
LeBaron, B. "A Builder's Guide to Agent-Based Financial Markets." *Quantitative Finance*, 2001, 1(2), 254-261.

Levine D. "A Parallel Genetic Algorithm for the Set Partitioning Problem." Tech. Rep. No. ANL-94/23, Argonne National Laboratory, Mathematics and Computer Science Division, Argonne, IL, 1994.

Li, J. and Tsang, E. P. K. "Investment Decision Making Using FGP: A Case Study," in *Proceedings of The Congress on Evolutionary Computation*, 1999, 1253-1259.

Likothanassis, S. D.; Georgopoulos, E. and Fotakis, D. "Optimizing the Structure of Neural Networks Using Evolution Techniques," in *Proceedings of 5th International Conference on Application of High-Performance Computers in Engineering*, 1997, 157-168.

Liu, Y. and Yao, X. "Evolutionary Design of Artificial Neural Networks with Different Nodes," in *Proceedings of IEEE International Conference on Evolutionary Computation*, 1996, 670-675.

Liu, L. and Yao, X. "Evolving Neural Networks for Hang Seng Stock Index Forecast," in *Proceedings of the congress on evolutionary computation*, IEEE Press, 2001, 256-260.

Luger, George F. *Artificial intelligence: Structures and strategies for complex problem solving* (Fifth edition). Addison-Wesley, 2005.

Malkiel, Burton G. *A random walk down wall street*. W.W. Norton & Company, 1973.

Malkiel, Burton G. "Returns from Investing in Equity Mutual Funds 1971 to 1991." *Journal of Finance*, 1995, 50(2), 549-572.

Malkiel, Burton G. "The Efficient Market Hypothesis and Its Critics." *Journal of Economics Perspectives*, 2003, 17(1), 59-82.

Manderick, B. and Spiessens, P. "Fine-grained Parallel Genetic Algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms*, 1989, 428-433.

Maniezzo, V. "Genetic Evolution of the Topology and Weight Distribution of Neural Networks." *IEEE Transactions on Neural Networks*, 1994, 5, 39-53.

Marr, David. *Vision*. MIT Press, 1982.

McCulloch, W. S. and Pitts, W. "A Logical Calculus of the Ideas Immanent in Nervous Activity." *Bulletin of Mathematical Biophysics*, 1943, 5, 115-133.

McDonnell, J. R. and Waagen, D. "Evolving Recurrent Perceptrons for Time Series Modelling." *IEEE Transactions on Neural Networks*, 1994, 5, 24-38.

McDonnell, J. R.; Page, W. and Waggen, D. "Neural Network Construction Using Evolutionary Search," in *Proceedings of the Third Annual Conference on Evolutionary Programming*, 1994a, 9-16.

McLachlan, G. J. and Krishnan, T. *The EM algorithm and extensions.* New York: Wiley, 1997.

Michalewicz, Z. *Genetic algorithms + Data Structures = Evolution Programs*. New York: Springer-Verlag, 1992.

Mitchell, Melanie. *An introduction to genetic algorithms*. MIT Press, 1996.
Mitchell, Tom M. *Machine learning*. McGraw-Hill, 1997.

Mingers, J. "An Empirical Comparison of Selection Measures for Decision-tree Induction." *Machine Learning*, 1989, 3(4), 319-342.

Minsky, M. L. (1961) "Steps Toward Artificial Intelligence," in *Proceedings of I.R.E.*, 1961, 49, 8-30. Reprinted in *Computers and Thought*, McGraw-Hill, 1963.

Minsky, M. L. and Papert, S. *Perceptrons*. MA: MIT Press, 1969.

Minsky, M. L. *The society of mind*. NY: Simon & Schuster, 1986.

Moriarty, D. E. and Miikkulainen, R. "Discovering Complex Othello Strategies Through Evolutionary Neural Networks." *Connection Science*, 1995, 7, 195-209.

Moriarty, D. E. and Mikkulainen, R. "Forming Neural Networks Through Efficient and Adaptive Evolution." *IEEE Transactions on Evolutionary Computation*, 1997, 4(5), 373-399.

Morrison, R. and Kenneth De Jong "A Test Problem Generation for Non-Stationary Environments," in *Proceedings of IEEE Congress on Evolutionary Computation*, 1999, 2047-2053.

Moscato, P. A. "On Evolution, Search, Optimisation, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms." Tech. Rep. No. 826, Caltech Concurrent Computation Program, Caltech, Pasadena, California. 1989.

Muhlenbein, H.; Schomisch, M. And Born, J. "The Parallel Genetic Algorithm as Function Optimizer," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991, 271-278.

Murphy, R. *Introduction to AI robotics*. MA Cambridge: MIT Press, 2000.

Muth, John F. "Rational Expectations and the Theory of Price Movements." 1961. Reprinted in *The new classical macroeconomics*. U.K.: Elgar, 1992, 3-23.

Newborn, M. *Kasparov vs. Deep Blue: Computer chess comes of age*. Springer-Verlag, 1996.

Nikolopoulos, C. and Fellrath, P. "Hybrid Expert System for Investment Advising," *Expert Systems*, 1994, 11(4), 245-248.

Ostrowski, D. A.; Tassier, T.; Everson, M. and Reynolds, R. G. "Using Cultural Algorithms to Evolve Strategies in Agent Based Models," in *Proceedings of te 2002 Congress on Evolutionary Computation*, 2002, 741-746.

Oz, T. *Stock trading wizard – Advance short-term trading strategies for swing and day trading*. Tony Oz Publications, 1999.

Palmer, R. G., Arthur, W. B., Holland, J. H., LeBaron, B. and Taylor, P. "Artificial Economic Life: A Simple Model of A Stockmarket." *Physica D*, 1994, 75, 264-274.

Park, J. and Sandberg, J. W. "Universal Approximation Using Radial Basis Functions Network." *Neural Computation*, 1991, 3, 246-257.

Parker, D. "Learning Logic." MIT Technical Report TR-47, MIT Center for Research in Computational Economics and Management Science, 1985.

Pearl, J. "Fusion, Propagation and Structuring in Belief Networks." *Artificial Intelligence*, 1986, 29, 241-288.

Petrovic, S. and Fayad, C. "A Fuzzy Shifting Bottleneck Hybridized with Genetic Algorithm for Real-world Job Shop Scheduling," in C. H. Antunes and L. C. Dias, eds., *Proceedings of Mini-EURO Conference, Managing Uncertainty in Decision Support Models*, 2004, 1-6.

Pettey, C. B.; Leuze, M. R. and Grefenstette, J. J. "A Parallel Genetic Algorithm," in *Proceedings of the Second International Conference on Genetic Algorithms*, 1987, 333-341.

Poli, R. "Genetic Programming for Feature Detection and Image Segmentation," in T. C. Fogarty, ed., *Evolutionary computation*, Springer-Verlag, LNCS, No. 1143, 1996, 110-125.

Pollack, J. B. and Blair, A. D. "Co-evolution in the Successful Learning of Backgammon Strategy." *Machine Learning*, 1998, 32(3), 225-240.

Poole, D.; Mackworth, A. K. and Goebel, R. *Computational intelligence: A logical approach*. Oxford University Press, 1998.

Porto, V. W.; Fogel, D. B. And Fogel, L. J. "Alternative neural network trainning methods." *IEEE Expert*, 1995, 10, 16-22.

Quinlan, J. R. "Induction of Decision Trees." *Machine Learning*, 1986, 1(1), 81-106.

Quinlan, J. R. *C4.5: Programs for machine learning*. CA: Morgan Kaufmann, 1993.

Rechenberg, I. "Cybernetic solution path of an experimental problem," Royal Aricraft Establishment, Library Translation No. 1122, August, 1965.

Reinganum, M. R. "The Anomalous Stock Market Behavior of Small Firms in January: Empirical Tests of Tax Loss Selling Effects." *Journal of Financial Economics*, 1983, 12, 89-194.

Resende, M. G. C. and Ribeiro, C. C. "Greedy Randomized Adaptive Search Procedures," in F. Glover and G. Kochenberger, eds., *Handbook of metaheuristics*, Kluwer Academic Publishers, 2002, 219-249.

Reuters. *An introduction to technical analysis*. The Reuters Financial Training Series, John Wiley & Sons, 1999.

Reeves, C. R., ed. *Modern heuristic techniques for combinatorial problems.* McGraw-Hill, 1995.

Reynolds, R. G. "An Adaptive Computer Model of the Evolution of Agriculture for Hunter-gatherers in the Valley of Oaxaca," Ph. D. Dissertation, University of Michigan, Ann Arbor, 1979.

Reynolds, R. G. "An Introduction to Cultural Algorithms," in *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, 1994, 131-139.

Reynolds, R. G. "On the Evolution of Schemata for Function Optimization," in A. Arbor, ed., *Holland fest: New directions in evolutionary computation inspired by the work of John Holland*. Michigan, May 16-18, 1999.

Reynolds, R. G. "Cultural Algorithms: A Tutorial." A tutorial given in *the 2002 World Congress on Evolutionary Computation*. [Online]. Available: http://ai.cs.wayne.edu/ai/pub.htm. 2002.

Reynolds, R. G. and Chung, C. J. "A Test Bed for Solving Optimization Problems Using Cultural Algorithms," in J. R. McDonnel and P. Angeline, eds., *Evolutionary programming V*. MA: MIT Press, 1996, 225-236.

Reynolds, R. G. and Chung, C. J. "A cultural Algorithm Framework to Evolve Multiagent Cooperation with Evolutionary Programming," in *Proceeding of Evolutionary Programming VI*, 1997, 323-333.

Reynolds, R. G. and Chung, C. J. "CAEP: An Evolution-Based Tool for Real-Valued Function Optimization Using Cultural Algorithms." *International Journal on Artificial Intelligence Tools*, 1998, 7(3), 239-293.

Reynolds, R. G. and Maletic, J. "The Use of Version Space Controlled Genetic Algorithms to Solve the Boole Problem." *International Journal on Artificial Intelligence Tools*, 1993, 2(2), 219-234.

Reynolds, R. G. and Ostrowski, D. "Using Cultural Algorithms to Evolve Strategies for Recessionary Markets", in *Proceedings of IEEE International Congress on Evolutionary Computation*, 2004, 1780-1785.

Reynolds, R. G. and Peng, B. " Cultural Algorithms: Knowledge Learning in Dynamic Environments," in *Proceedings of IEEE International Congress on Evolutionary Computation*, 2004, 1751-1758.

Reynolds, R. G. and Peng, B. "Knowledge Learning and Social Swarms in Cultural Algorithms." *Journal of Mathematical Sociology*, 2005, 29, 1-18.

Reynolds, R. and Rychtyckyj, N. "Knowledge base Maintenance Using Cultural Algorithms: Application to the DLMS Manufacturing Process System," in *Proceedings of World Congress on Computational Intelligence*, 2002, 855-860.

Reynolds, R.G., and Saleem, S. "Function Optimization with Cultural Algorithms in Dynamic Environments," in *IEEE Proceedings of the Particle Swarm Optimization Workshop*, 2001, 63-79.

Reynolds, R. G. and Saleem, S. "The Impact of Environmental Dynamics on Cultural Emergence," in L. Booker; S. Forrest; M. Mitchell and R. Riolo, eds., *Perspectives on adaptation in natural and artificial systems - Essays in honor of John Holland (Santa Fe Institute Studies on the Sciences of Complexity)*, Oxford University Press, 2004.

Reynolds, R.G.; Stefan, J.; Fotouhi F.; Lu S.; Dong, M. and Aristar, T. "Evolution-Based Approaches to the Preservation of Endangered Natural Languages," in *Proceedings 2003 IEEE Proceedings of Congress on Evolutionary Computation*, 2003, 1980-1987.

Reynolds, G. R. and Sverdlik, W. "Dynamic Version Spaces in Machine Learning," in *Proceedings of IEEE Conference on Tools for Artificial Intelligence*, 1992, 10-13.

Reynolds, R. G. and Zhu, S. "Fuzzy Cultural Algorithms with Evolutionary Programming for Real-Valued Function Optimisation." *IEEE Transactions on Systems, Man, and Cybernetics*, 2001, 31(1), 1-18.

Reynolds, R. G.; Kobti, Z. and Kohler, T. "The Effect of Culture on the Resilience of Social Systems in the Village Multi-Agent Simulation", in *Proceedings of IEEE International Congress on Evolutionary Computation*, 2004, 1743-1750.

Rich, E. and Knight, K. *Artificial intelligence* (Second edition). NY: McGraw-Hill. 1991.

Richards, N.; Miikkulainen, R. and Moriarty, D. "Evolving Neural Networks to Play Go." *Artificial Intelligence*, 1998, 8, 85-96.

Ritter, J. R. "The Buying and Selling Behavior of Individual investors at the Turn of the Year." *Journal of Finance*, 1988, 43, 701-717.

Rosenblatt, F. "The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain." *Psychological Review*, 1959, 65, 386-408.
Rosenblatt, F. *Principles of neurodynamics*. NY: Spartan Books, 1962.

Rothschild, M. and Stiglitz, J. E. "Equilibrium in Competitive Insurance Markets: An Essay on the Economics of Imperfect Information." *The Quarterly Journal of Economics*, 1976, 90(4), 630-649.

Rumehart, D. E.; Hinton, G. E. and Williams, R. J. "Learning Internal Representation by Error Propagation," in *Parallel distributed processing: Expectations in the microstructures of cognition*, MA: MIT Press, Vol. 1, 1986.

Russell, S.; Binder, J.; Koller, D. and Kanazawa, K. "Local Learning in Probabilistic Networks with Hidden Variables," in *Proceedings of the 14th international joint conference on artificial intelligence*. San Francisco: Morgan Kaufmann, 1995, 1146-1152.

Russell, Stuart and Norvig, Peter. *Artificial intelligence: a modern approach* (Second edition). Prentice Hall, 2003.

Samuel, A. L. "Some Studies in Machine Learning Using the Game of Checkers." *IBM Journal of Research and Development*, 1959, 3(3), 210-229.

Samuel, A. L. (1959) "Some Studies in Machine Learning Using the Game of Checkers." *IBM Journal on Research and Development*, 3(3), 210-229. Reprinted in: E. A. Feigenbaum and J. Feldman, eds., *Computer and Thought*, NY: McGraw-Hill, 1963. Reprinted in: *IBM Journal on Research and Development*, 2000, 44(1/2), 207-226.

Samuel, A. L. (1967) "Some Studies in Machine Learning Using the Game of Checkers II – Recent Progress." *IBM Journal on Research and Development*, 11(6), 601-617. Reprinted in: D. L. Levy, ed., *Computer games*, NY: Springer-Verlag, 1988, 366-400.

Sarkar, M. and Yegnanarayana, B. "Evolutionary programming-based probabilistic neural networks construction technique," in *Proceedings of IEEE International Conference of Neural Networks*, 1997, 456-461.

Sarma, J. and Jong, K. D. "An Analysis of the Effects of Neighborhood Size and Shape on Local Selection Algorithms." *Parallel problem Solving from Nature IV*, Berlin: Springer-Verlag, 1996, 236-244.

Schaeffer J.; Lake R. and Lu P. "CHINOOK The World Man-Machine Checkers Champion." *AI Magazine*, 1996, 17(1), 21-30.

Schaeffer, J. *One jump ahead: Challenges human supermacy in checkers*. Berlin: Springer-Verlag, 1997.

Schaerf, A. "Local Search Techniques for Constrained Portfolio Selection Problems." *Computational Economics*, 2002, 20(3), 177-190.

Schwehm, M. "Implementation of Genetic Algorithms on Various Interconnection Networks," in M. Valero, E. Onate, M. Jane, J. L. Larriba and B. Suarez, eds., *Parallel Computing and Transputer Applications*. IOS Press, 1992, 195-203.
Schwert, G. W. "Anomalies and Market Efficiency," in G. Constantinides *et al.*, eds., *Handbook of the economics of finance*, North Holland, 2001.

Schlang, M.; Poppe, T. and Gramckow, O. "Neural Networks for Steel Manufacturing." *IEEE Expert Intelligent Systems*, 1996, 11(4), 8-10.

Schulenberg, S. and Ross, P. "An Adaptive Agent Based Economic Model," in P. Lanzi, W. Stolzmann and S. Wilson, eds., *Learning classifier systems: From foundations to application,* Berlin: Springer-Verlag, Vol. 1813 of LNAI. 2000, 265-284.

Schulenberg, S. and Ross, P. "Strength and Money: An LCS Approach to Increasing Returns," in P. L. Lanzi, W. Stolzmann and S. W. Wilson, eds., *Advances in learning classifier systems.* Berlin: Springer-Verlag, Vol. 1996 of LNAI. 2001, 114-137.

Schulenber, S. and Ross, P. "Exploration in LCS Models of Stock Trading," in P. L. Lanzi, W. Stolzmann and S. W. Wilson, eds., *Advances in learning classifier systems,* volume 2321 of Lecture Notes in Artificial Intelligence. Berlin: Springer-Verlag, 2002, 150-179.

Schwefel, H. P. "Kybernetische evolution als strategie der experimentellen forschung der strmungstechink." Diploma Thesis, Technical Univ. of Berlin, 1965.

Schwefel, H. P. *Numerical optimization of computer models.* Chichester, U.K.: Wiley, 1981.

Secrest, B. R. "Travelling Salesman Problem for Surveillance Mission Using Particle Swarm Optimization." AFIT/GCE/ENG/01M-03, Air Force Institute of Technology, 2001.

Shleifer, A. and Vishny, R. W. "Management Entrenchment: The Case of Manager-Specific Assets." *Journal of Financial Economics*, 1989, 25(1), 123-139.

Shadbolt, J. and Taylor, J. G. *Neural networks and the financial markets: Predicting, combing and portfolio optimisation*. London: Springer-Verlag, 2002.

Sharp, W. "Capital Asset Prices: A Theory of market Equilibrium Under Conditions of Risk." *Journal of Finance*, 1964, 19(3), 425-442.

Shi, Y. and Eberhart, R. C. "Parameter Selection in Particle Swarm Optimization," in *Proceeding of the 7th Annual conference on Evolutionary Programming*, 1998, 591-600.

Shleifer, A. and Vishny, R. W. "Management Entrenchment: The Case of Manager-Specific Assets." *Journal of Financial Economics*, 1989, 25(1), 123-139.

Smith, S. F. "A Learning System Based on Genetic Adaptive Algorithms," Ph. D. dissertation, Department of Computer Science, University of Pittsburgh, PA, 1980.

Soule, T. and Foster, J. "Effects of Code Growth and Parsimony Pressure on Populations in Genetic Programming." *Evolutionary Programming*, 1998, 6(4), 293-309.

Specht, D.F. "Probabilistic Neural Networks," *Neural Networks*, 1990, 3, 110-118.

Spector, L.; Langdon, W. B.; O'Reilly, U. M. and Angeline, *Advances in genetic programming 3*, MA: MIT Press, 1999.

Spence, M. *Market signaling: Informational transfer in hiring and related processes*. Cambridge: Harvard University Press, 1974.

Stanley, K. and Miikkulainen, R. "Evolving Neural Networks Through Augmenting Topologies." *IEEE Transactions on Evolutionary Computation*, 2002, 10(2), 99-127.

Stender, J., ed. *Parallel genetic algorithms: Theory and applications*. IOS Press: Frontiers in artificial intelligence and applications volume 14, 1993.

Stiglitz, Joseph E. "Information and the change in the paradigm in economics." *Les Prix Nobel*. The Noble Foundation, 2001, 472-540.

Stiglitz, Joseph E. "Information and the change in the paradigm in economics," in R. Arnott; B. Greenwald; R. Kanbur and B. Nalebuff, *Economics for an imperfect world*. MA: MIT Press, 2003, 570-639.

Stutzle, T. and Dorigo, M. "ACO Algorithms for the Travelling Salesman Problem," in K. Miettinen; M. M. Makela; P. Neittaanmaki and J. Periaux, eds., *Evolutionary algorithms in engineering and computer science*, John Wiley & Sons, 1999, 163-183.

Stutzle, T. and Hoos, H. H. "Improvements on the Ant Systems: Introducing the MAX-MIN Ant System," in R. F. Albrecht; G. D. Smith and N. C. Steele, eds., *Artificial neural networks and genetic algorithms*, NY: Springer Verlag, 1998, 245-249.

Stutzle, T. and Hoos, H. H. "MAX-MIN Ant Systems and Local Search for Combinatorial Optimization Problems," in S. Voss, S. Martello, I. H. Osman and C. Roucairol, eds., *Meta-Heuristics: Advances and trends in local search paradigms for optimization*, Boston: Kluwer, 1999, 313-329.

Sutton, Richard S. "Two Problems with Backpropagation and Other Steepest-descent Learning Procedures for Networks," in *Proceedings of 8th Annual Conference of Cognitive Science Society*. NJ: Hillsdale, 1986, 823-831.

Sutton, Richard S. and Barto, Andrew G. *Reinforcement learning*. MA: MIT Press, 1998.

Tamaki, H. and Nishikawa, Y. "A Paralleled Genetic Algorithm Based on a Neighborhood Model and Its Application to the Job Shop Scheduling," in R. Manner and B. Derick, eds., *Parallel problem solving from nature II*. Elsevier Science, 1992, 573-582.

Tanese, R. "Distributed Genetic Algorithms," in *Proceedings of the Third International Conference on Genetic Algorithms*, 1989, 434-439.

Tesauro, G. and Sejnowski, T. J. "A Parallel Network that Learns to Play Backgammon." *Artificial Intelligence*, 1989, 39, 357-390.

Tesfatsion, L. "Agent-based Computational Economics: A Constructive Approach to Economic Theory," in K. L. Judd and L. Tesfatsion, eds., *Handbook of computational economics, Volume 2: Agent-Based Computational Economics*. North-Holland, 2005.

Thomsett, M. *Mastering fundamental analysis*. Dearborn Trade, 1998.

Trippi, R. R. and Turban, E. *Neural networks in finance and investing*. Chicago: Probus Publishing Company, 1993.

Trippi, R. R. and Turban, E. *Neural networks in finance and investing: Using artificial intelligence to improve real-world performance*. NY: McGraw-Hill, 1995.

Trippi, R. R. and Turban, E. *Neural networks in finance and investing: Using artificial intelligence to improve real-world performance* (second edition). Irwin Professional Publishing, 1996.

Tsang, E. P. K.; Li, J. and Butler, J. M. "EDDIE Beats the Bookies." *International Journal of Software*, 1998, 28(10), 1033-1043.

Tsang, E. P. K.; Li, J.; Markose, S.; Er, H.; Salhi, A. and Iori, G. "EDDIE In Financial Decision Making." *Journal of Management and Economics*, 2000, 4(4).

Tsang, E. P. K. and Li, J. "EDDIE for Financial Forecasting," in S. H. Chen, ed., *Genetic algorithms and programming in computational finance*. Kluwer Series in Computational Finance, 2002, 161-174.

Tsang, E. P. K. and Martinez-Jaramillo, S. "Computational Finance." *IEEE Computational Intelligence Society Newsletter*, August 2004, 3-8.

Tsang, E. P. K.; Yung, P. and Li, J. "EDDIE-Automation: A Decision Support Tool for Financial Forecasting." *Journal of Decision Support Systems*, 2004a, 37(4), 559-565.

Turing, A. M. "Computing Machinery and Intelligence." *Mind*, 1950, 59, 433-460.

Vriend, Nicolaas J. "An Illustration of the Essential Difference Between Individual and Social Learning, and Its Consequences for Computational Analyses." *Journal of Economic Dynamics and Control*, 2000, 24(1), 1-19.

White, H. "Economic Prediction Using Neural Networks: The Case of IBM Daily Stock Returns," in *Proceedings of the IEEE international conference on neural networks*, 1988, 451-458.

White, G. and Ligomenides, P. "GANet: A Genetic Algorithm for Optimizing Topology and Weights in Neural Network Design," in *Proceedings of International Workshop on Artificial Neural Networks*, 1993, 322-327.

Whitley, D.; Starkweather, T. and Bogart C. "Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity." *Parallel Computation*, 1990, 14(3), 347-361.

Whitley, C.; Starkweather, T. and Shaner, D. "The Travelling Salesman and Sequence Scheduling Quality Solutions Using Genetic Edge Recombination," in L. Davis, ed., *Handbook of Genetic Algorithms*. New York: Van Nostrand Reinhold, 1991, 350-372.

Widrow, B. and Hoff, M. E. "Adaptive Switching Circuits." *IRE WESCON Convention Record*, 1960, 4, 96-104.

Williams, R. and Zipser, D. "Gradient-based Learning Algorithms for Recurrent Networks and Their Computational Complexity," in Y. Chauvin and D. Rumelhart, eds., *Backpropagation: Theory, architectures, and applications*. NJ: Lawrence Erlbaum Associates, 1995, 433-486.

Winston, P. H. *Artificial intelligence* (Third edition). MA: Addison-Wesley, 1992.

Yang, J. "Heterogeneous Beliefs, Intelligent Agents, and Allocative Efficiency in an Artificial Stock Market," in *Computing in Economics and Finance 1999*, series by Society for Computional Economics, No. 612, 1999.

Yang, J. "The Efficiency of an Artificial Double Auction Stock Market with Neural Learning Agents," in S. H. Chen, ed., *Evolutionary computation in economics and finance*, Studies in fuzziness and soft computing, Vol. 100. NY: Physica-Verlag, 2002, 85-103.

Yao, S.; Wei, C. J. and He, Z. Y. "Evolving wavelet neural networks for function approximation." *Electronic Letters*, 1996, 32(4), 360-361.

Yao X. "An Empirical Study of Genetic Operators in Genetic Algorithms." *Microprocessing and Microprogramming*, 1993, 38(1-5), 707--714.

Yao, X. "The Importance of Maintaining Behavioral Link Between Parents and Offspring," in *Proceedings of IEEE Conference on Evolutionary Computation*, 1997, 629-633.

Yao, X. "Evolving Artificial Neural Networks," in *Proceedings of the IEEE*, September 1999, 87(9), 1423-1447.

Yao, X. *Evolutionary computation – Theory and applications*. World Scientific Publishing, 1999a.

Yao, X. and Liu, Y. "Evolving Artificial Neural Networks for Medical Applications," in *Proceedings of Australia-Korea Joint Workshop on Evolutionary Computation*, 1995, 1-16.

Yao, X. and Liu, Y. "A New Evolutionary System for Evolving Artificial Neural Networks." *IEEE Transactions on Neural Networks*, 1997, 8(3), 694-713.

Yao, X. and Liu, Y. "EPNet for Chaotic Time Series Prediction," in X., Yao; J. H. Kim and T. Furuhashi, eds., *Selected Papers of 1st Asia-Pacific Conference on Simulated Evolution and Learning*, vol. 1285 of *Lecture Notes in Artificial Intelligence*, Berlin: Springer-Verlag, 1997a, 146-156.

Yoon, Y. and Swales, G. "Predicting Stock Price Performance: A Neural Network Approach," in R. R. Trippi and E. Turban, eds., *Neural networks in finance and investing*, Chicago: Probus Publishing Company, 1993, 392-342.

Yoshida, H.; Kawata, K. and Fukuyama, Y. "A Particle Swarm Optimisation for Reactive Power and Voltage Control Considering Voltage Security Assessment." *IEEE Transactions on Power Systems*, 2001, 15(4), 1232-1239.

Zadeh, Lotfi. "Fuzzy Sets." *Information and Control*, 1965, (8), 338-353.

Zadeh, Lotfi. "Outline of a New Approach to the Analysis of Complex Systems." *IEEE Transactions on Systems, Man and Cybernetics*, 1973, 3, 28-44.

Zadeh, Lotfi. "The Calculus of Fuzzy Restrictions," in L. A. Zadeh *et al.*, eds., *Fuzzy sets and applications to cognitive and decision making processes*. NY: Academic Press, 1975, 1-39.

Zannoni, E. and Reynolds, R. G. "Learning to Control the Program Evolution Process in Genetic Programming Systems Using Cultural Algorithms", *Journal of Evolutionary Computation*, 1997, 5(2), 181-211.

Zhang, N. L. and Poole, D. "A Simple Approach to Bayesian Network Computations," in *Proceedings of the 10th Canadian conference on artificial intelligence*, Morgan Kaufmann, 1994, 171-178.

Zirilli, J. S. *Financial prediction using neural networks*. MA: International Thomson Publishing, 1996.

Glossary

Agent-based computational economics (ACE) Agent-based computational economics (ACE) is the computational study of economies modelled as evolving systems of autonomous interacting agents.

Artificial intelligence (AI) Artificial intelligence research is the scientific understanding of the mechanisms underlying though and intelligent behaviours and their embodiment in machines.

Artificial stock market (ASM) Artificial stock markets are multi-agent based computational models that simulate a stock market in which heterogeneous artificial agents choose between investing in risk-free assets and risky stocks. In contrasts to conventional representative mathematical models of stock markets based on unrealistic assumptions of trader's knowledge and rationality, artificial stock markets are able to demonstrate features that are observed from real-world markets.

Environmental variables In an imperfect evolutionary system, an environmental variable is a piece of information that individuals perceives from the imperfect environment, which is then transferred into skill and knowledge by imperfect individuals.

Evolutionary algorithms A class of population-based stochastic search algorithms including genetic algorithms, evolutionary strategies, evolutionary programming, memetic algorithms, etc.

Evolutionary computation The study of computational systems that use ideas and inspirations from natural evolution and adaptation.

Incomplete environment A dynamic environment that constantly presents new challenges to its participants.

Imperfect evolutionary system (IES) A system where intelligent individuals optimise their own utilities, with resources available, whilst adapting themselves to the new challenges from an evolutionary imperfect environment.

Imperfect individual In an imperfect evolutionary system, individuals do not completely understand their environment due to the complexity of the environment and every individual only has a (possibly) different view of the imperfect environment.

Information economics Information economics is a branch of economics that studies how information affects economic decisions. Information economics focuses on three areas: the study of information asymmetries, the economics of information goods, and the economics of information technology. The study of information economics represents a change from the standard competitive equilibrium paradigm to an imperfect information paradigm.

Machine learning Te study of algorithms that automatically improve a machine's performance at some task through experience.

Reinforcement learning A computational approach to learning whereby an agent tries to maximise the total amount of reward it receives when the machine is interacting with a complex, uncertain environment.

Short selling (Short sale) The sale of shares not owned by the investor but borrowed through a stock broker and later repurchased to replace the loan. Profit is earned if the initial sale is at a higher price than the repurchase price.

Soft computing soft computing differs from conventional (hard) computing in that it is tolerant of imprecision, uncertainty, partial truth, and approximation. The guiding principle of soft computing is: Exploit the tolerance for imprecision, uncertainty, partial truth, and approximation to achieve tractability, robustness and low solution cost.

Index of Notation

Below is a summary of the notation used in this thesis.

Gain(S,A): Measures how well a given attribute separates the training examples according to their target classifications in decision tree learning.

$\sum_{i=1}^{n} x_i$: The sum $x_1 + x_2 + \ldots + x_n$

$|S|$: The total number of instances in a collection or sample *S*.

Entropy(S): Measures the impurity in a collection *S*, equals to the minimum number of bits of information needed to encode the classifications of an arbitrary member of *S*.

$\arg\max_{x \in X} f(x)$: The value of *x* that maximizes *f*(*x*). For example, $\arg\max_{x \in (1,2,-3)} x^2 = -3.$

$\hat{f}(x)$: A function that approximates the function *f*(*x*).

$d(x_i, x_j)$: The Euclidean distance between two instances in instance based learning.

$\neg A$: A is not true.

N(S): The neighbourhood of current solution *S*.

M^T: The tabu list in tabu search.

$N(0,1)$: A normally distributed random number with mean 0 and standard deviation 1.

μ: The number of parents in evolutionary strategies or evolutionary computation

λ: The number of offspring generated in evolutionary strategies or evolutionary programming.

$\delta(s_t, a_t)$: The next state to move to, given the current state s_t and its associated action a_t, used in reinforcement learning.

$r(s_t, a_t)$: The reward from the action a_t given the current state as s_t, used in reinforcement learning.

V^*: The maximum discounted cumulative reward that an agent can obtain form a state $\delta(s_t, a_t)$. Used in reinforcement learning.

π^*: The optimal policy in reinforcement learning.

$Q(s,a)$: The Q function in reinforcement learning.

$w_{j,i}$: In artificial neural networks, the connection weight from node *i* to node *j*.

c: Learning rate in perceptron learning.

η: Learning rate in backpropagation training for multiplayer perceptrons.

$(c_{i,j})_{N\times N}$: $N \times N$ matrix, where c_{ij} indicates the presence or absence of a connection in the matrix, used in evolving neural network architechtures.

ply: A ply is a move in a game.

σ: Standard deviation or variance. Used in both Gaussian mutation and financial portfolio management.

$p_{ij}^{k}(t)$: The probability of an ant *k* at node *i* choosing city *j* to move at the *t*-th iteration, used in ants algorithms.

τ_{ij}: Artificial pheromone used in ants algorithms.

ρ: Pheromone decay coefficient in ants algorithms.

$v_{i,j}^{k}$: Velocity of the *i*th particle on the *j*th dimension at iteration *t*+1, used in particle swarm optimisation.

p_{best}: Personal best position in an agent's search history, used in particle swarm optimisation.

g_{best}: Global best position that is accepted by the entire swarm, used in particle swarm optimisation.

w: A weighting function used by particle swarm optimisation algorithms.

POP: Population space of a cultural algorithm.

BLF: Belief space of a cultural algorithm.

P/E ratio: Price/earnings ration in fundamental analysis.

MA_{10}: 10-day price moving average in technical analysis.

RSI: Relative strength index used in technical analysis.

ROC: Rate of change (price) in technical analysis.

d: Dividends used in the Santa Fe artificial stock market.

$U(c)$: Utility of wealth in the CARA model, used in the in the Santa Fe artificial stock market.

$MAPE$: Mean absolute percentage error for measuring GP prediction error, used in GP-based artificial stock markets.

E: An imperfect environment.

v_i : An environmental variable in an imperfect environment.

I_i : An imperfect individual in an imperfect evolutionary system.

INDn : The nth technical indicator used in the imperfect evolutionary market.

SN : Starting node of a connection, used in real-value vector representation of an ANN.

EN : Ending node of a connection, used in real-value vector representation of an ANN.

W : Weight of the connection between *SN* and *EN*, used in the real-value vector representation of an ANN.

AF : Activation function of the end node *EN*.

T : An artificial stock trader.

NOI : The total number of indicators used by an artificial stock trader.

ROP : Rate of profit, used in the imperfect evolutionary market.

S^i_{peer} : Peer pressure during the social learning in the ISP learning paradigm

S^i_{self} : Self-motivation during the social learning in the ISP learning paradigm.

$U(e)$: Utility of environmental variables used in the imperfect evolutionary market.

ω_i : The normalised score of an artificial trader in the imperfect evolutionary market.

Φ : The arithmetic mean of all artificial stock traders' normalised scores.

θ_1 : Parameter 1 used for social learning study in the imperfect evolutionary market

θ_2 : Parameter 2 used for social learning study in the imperfect evolutionary market