NAIST-IS-DT9561006

Doctor's Thesis

Associative Memory Model under Artificial Evolution

Akira Imada

February 8, 1999

Department of Information Systems
Graduate School of Information Science
Nara Institute of Science and Technology

Doctor's Thesis
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
DOCTOR of ENGINEERING

Akira Imada

Thesis committee: Akira Fukuda, Professor
Yutaka Takahashi, Professor
Minoru Ito, Professor

Associative Memory Model under Artificial Evolution*

Akira Imada

Abstract

We apply evolutionary computations to the fully-connected neural network model of associative memory. Though lots of applications of evolutionary algorithms to the layered typed neural networks have been reported, there remain few for the fully-connected neural networks.

In the model, some of the appropriate configurations of synaptic weights give a network a function of associative memory. So far, a modest amount of such configurations have been found heuristically, but they are not exhaustive at all. It is known that there exist an infinite number of these solutions, but we have had no information as to the number and distribution of these solutions. One of our goals is to address this still-unknown issue using evolutionary algorithms. For the purpose, various variants of evolutionary algorithms were exploited here, and a number of algorithms were found to be capable of locating these solutions in weight space. Thus far, we have a variety of weight solutions. We examined each of these solutions in terms of two parameters regarding synaptic weights: dilution ratio and degree of symmetry, and found that the behaviors given by these solutions are different from each other.

Then a question arises as to whether or not these solutions are representative samples of the whole solutions. To answer this question, two approaches were considered here. One is an analysis based on fitness landscape, or equivalently, a hyper-plane defined on all the possible weight configurations of a network by

*Doctor's Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DT9561006, February 8, 1999.

the degree to how each configuration works appropriately. The other is by visualizing the high dimensional weight space in which a number of points in high-dimensional weight space are mapped into two-dimensional locations with the distance information remaining as much as possible. Both of these analyses are novelties introduced in this thesis.

As results, we observe that (1) the global peaks in fitness landscape become narrower as the number of patterns to be stored increases; and (2) the solutions are uniformly distributed in weight space when the number of patterns is small, while they are gradually localized as the number of patterns becomes larger.

Keywords:

fully-connected-neural-network, associative-memory, evolutionary-algorithm, fitness-landscape, learning

Contents

List of Figures

List of Tables

1. INTRODUCTION

Some sort of sciences arises from the desire to create "Pigmarion", a humanoid created by a sculptor in an ancient Greece literature wishing her to be an ideal woman. Thus, modeling the brain has long been an incentive to study some discipline of our science.

Since a coincidental emergence of a life on Earth, *biological neural networks* have undergone genetic evolution and have been improved over a long period of time. It therefore seems plausible that evolutionary concepts could also be effective when applied to *artificial neural networks.* As one of these biologically motivated activities, Fogel (1995, p.97) cited Conrad (1974) who hypothesized that the brain has the similar type of learning mechanisms to evolution. Along a series of speculations by Conrad (1974; 1981; 1984; 1985; 1987; 1988; 1990), Kampfner and Conrad (1983) discussed simulations of neuronal learning systems that adapt through evolutionary changes in their paper titled:

- *Computational Modeling of Evolutionary Learning Processes in the Brain.*

Edelmann, Novel Laureate, also hypothesized competitions in brain (Edelmann, 1987). In his book "*The Theory of Neuronal Group Selection*", Edelmann explained the global architecture of the brain with neuronal selection, and suggested that the topological correspondence between retina and cortex is not totally genetically determined but sensory experience is also necessary for the development of the correct neural circuitry. Namely, he suggested that in early development of brain, evolution in neurons' level plays an important role to construct neural circuit in brain.

Anyhow, we might at least conjecture that "intelligence is inseparable from the trial-and-error process itself" (Atmar, 1990). Under this conjecture, we have evolved artificial neural networks employing Evolutionary Programming, Evolution Strategy and Genetic Algorithm toward the goal of modeling the brain in terms of evolution.

Since we evolve neural network models by evolutionary algorithms in this thesis,

we briefly describe both the evolutionary algorithms and the associative memory system in the following two sections.

1.1 Evolutionary Algorithm

1.1.1 What are Evolutionary Algorithms?

Given a problem, candidate solutions are represented by strings which are analogous to natural *chromosomes*. Elements of a string are either discrete symbols or real-valued variables which are analogous to *genes*. A collection of these candidate solutions represented by strings constructs a *population*. Each string in a population is sometimes called an *individual*. Evolutionary algorithms proceeds as follows: Two individual chromosomes mate and produce an offspring chromosome, by exchanging a part of their genes. This is analogous to a sexual reproduction with *crossover* operated on chromosomes. Genes of offspring chromosome are occasionally modified, and this is called a *mutation*. Note here that in some algorithms, sexual reproduction does not occur but only mutation produces an offspring from a parent chromosome. This is analogous to an asexual reproduction. Then, a number of individuals survive for the next generation by a *selection* from parents and offsprings according to the principle of *survival of the fittest*.

Although there proposed many variants of these algorithms, Genetic Algorithm, Evolutionary Programming, and Evolution Strategy, we abbreviate them hereafter to GA, EP, and ES, respectively, are most popularly employed these days. These three algorithms are partly characterized as *population-based* searches and *genetics-inspired* operators such as *crossover* and *mutation*. Note that some other stochastic search techniques such as *simulated annealing* do not employ this population-to-population scheme. The object or trait represented by a string of genes is sometimes referred to as *phenotype* and a string *per se* is called *genotype*. In GA, what are evolved are genotypes which are mostly represented by binary strings, while in EP and ES, each element of strings represents some trait[1] of the individual, and hence takes real-value in general and varies from one individual to another with respect to a Gaussian distribution. EP modifies individuals

[1] Biological examples are eye color, height, weight and so on.

by mutation operation alone, while GA and ES modifies them by mutation and crossover operation. Furthermore, in EP and ES these modifications of individuals are changed adaptively during an evolution, while in GA the modifications remain fixed.

1.1.2 Where They Came From?: A Brief History

As Fogel (1995, p.103) noted, *"simulated evolution has a long history. Similar ideas and implementations have been independently invented numerous times."* Goldberg (1989, pp.126-129, pp.219–220) listed 83 such works picking them up from the works done since 1962. In this subsection, some of these works are described by paraphrasing and summarizing these works in literatures.

The GAs are believed to be *"invented by John Holland in the 1960s and developed by Holland and his students and colleagues at the University of Michigan in the 1960s and the 1970s"* (Mitchell, 1996, p.2). In his book, Goldberg (1989, p.92) wrote: *" The first mention of the words Genetic Algorithm and the first published application of a Genetic Algorithm both came in Bagley's (1967) Ph.D dissertation at the University of Michigan."* In fact, almost all works concerning the Genetic Algorithm referred the Holland's seminal book published in 1975 as the origin of the Genetic Algorithm.[2]

Similarly, Fogel, Owens, and Walsh(1966) is said to be the origin of EP and Rechenberg (1965) and Schwefel (1965) are for ES.

However, there had been many techniques called *genetic* or *evolutionary* earlier than GA, EP and ES. Furthermore, the concept of artificial evolutions dates back to much earlier works. Fogel (1995, p.67) cited a work made as early as in 1932 by Cannon (1932) who *"pictured natural evolution as a process similar to learning"* which an individual undergoes in its lifetime. Fogel also cited the following three researchers as early speculations about artificial intelligence and evolution: Turing (1950) who recognized an obvious connection between machine learning and evolution; Friedman (1959) who speculated that a simulation of mutation

[2]Goldberg (1989) is often cited together with Holland since it has contributed for the GA to become very popular.

and selection would be able to design thinking machines; and Campbell (1960) who offered the conjecture that a blind variation-and-selective-survival process is involved in all processes leading to expansions of knowledge. Among others, it is interesting to point out that Turing (1950) suggested artificial intelligence researchers should build a pain-pleasure mechanism into their software.

Then in late 1950s, some biologists started to study computer simulation of natural evolution to understand the phenomena of natural evolution (e.g., see Barricelli (1957; 1962a; 1962b), Fraser (1957a; 1957b), Martin et al., (1960)). We won't go into details of these works, but only show their titles here.

- *Symbiogenetic Evolution Processes Realized by Artificial Methods* (Barricelli, 1957).
- *Numerical Testing of Evolution Theories* (Barricelli, 1962a; 1962b).
- *Simulation of Genetic Systems by Automatic Digital Computer: II. Effects of Linkage on Rates of Advance under Selection* (Fraser, 1957b).
- *High Speed Selection Studies* (Martin et al., 1960)

These works invoked various practical engineering applications of evolutionary theory, i.e., what Goldberg called *evolution-inspired algorithms*. For example, as Doctoral dissertations at the University of Michigan, Rosenberg (1967) simulated the evolution of a population of single-celled organism; and Weinberg (1970) argued the computer simulation of evolving DNA. Although their contributions were sometimes overlooked because of their emphasis on biological simulation, these are important to the subsequent development of GAs, as Goldberg noted.

Not only researches at the University of Michigan, but we can enumerate many other such works. We summarize these movements in the following subsections.

Machine Learning

Fogel (1995, pp.68–70) cited Box (1957) who advocated an application of a kind of evolutionary operation to manufacturing processes by viewing the plant as

an evolving species.[3] Friedberg (1958) who employed some chance operations (though the author did not claim to be simulating natural evolution) to gradually improve a machine language computer program, aiming a sort of automatic programming. In his work, a program evolved from a random sequence of instruction by interchanging instructions (crossover) and changing a randomly chosen instruction (mutation) under a task of calculating the sum of two inputs.[4]

Optimizations

Goldberg cited Hollstien (1971) as *"the first dissertation to apply GA to a pure problem of mathematical optimization."* And De Jong, following Hollstien, established the method to obtain global optima of some pure mathematical functions, which are even now used as a test suit.

However, we can see earlier works that argued optimization problem in more general form. As examples, Goldberg wrote that *"the studies of Bledsoe and Bremermann came closest to the modern notion of a Genetic Algorithm"* (Goldberg, 1989, p.104). Bledsoe (1961) employed concepts of "*individual-by-individual generation*, *mutation* and *save-the-better selection.*" Then, Bremermann expanded this concept in a series of studies ((Bremermann, 1958; 1962; 1967; 1968; 1973), (Bremermann and Rogan, 1964), (Bremermann, Rogan and Salaff, 1965; 1966)) by "generating *population-by-population scheme* of strings using selection and mutation", which means a number of descendants from a single ancestor are produced and the best of the descendants survives to the next ancestor (Bäck, p.58).[5] His proposed mutation rate $p_m = 1/l$, where l is a length of a string, is also adopted by many current implementations. He also proposed, but not show the result, the use of a *recombination* in which information was exchanged between individuals.

Bremermann recognized *evolution* as an *optimization problem*, and evolved a string of alphabet from $\{0, 1\}$ (Bremermann, 1958) as well as strings of real-

[3] This selection scheme is later called a $(1+\lambda)$ selection in ES community, which implies a parent produces λ children and the fittest of $(1 + \lambda)$ becomes the next parent.

[4] This selection scheme is later called a (1, 1) selection in ES community.

[5] This is a $(1, \lambda)$ selection in later ES terminology implying a parent produces and the fittest of the λ children (not including their parent) becomes the next parent.

valued elements[6] (Bremermann, 1962). Namely, Bremermann considered the problem of minimizing a real-valued function $f(x_1, x_2, \cdots, x_n)$, where $x_i \in \Re$. The elements x_i were claimed to be analogous to an organism's gene.[7] The Bremermann's latter method of evolving strings made up of real-valued elements is clearly one of the origin of today's Evolutionary Programming[8], although there is no such an explicit description in literatures. Nevertheless, Bremermann did not obtain any successful results of finding optima, which Atmar (1979) called Bremermann's disappointment (see Fogel, 1995, p.72). The failure is, as Bäck (1996, p.59) wrote, partly because the problem chosen were *"a much too simple problem domain, where evolutionary algorithms cannot compete with the variety of specialized optimization techniques."*

The nomenclature of EP is from the work of L. J. Fogel (1962; 1964), and the paper by Fogel, Owens and Walsh (1966) is now cited as the origin of EP. According to David Fogel, L. J. Fogel's son, the original EP is summarized as follows. A population of finite state machines is exposed to the environment which is described as a sequence of symbols taken from a finite alphabet. Each parent machine predicts next symbol as its output as each input symbol is offered to the machine. After the prediction of the last symbol in the sequence, the fitness of the machine is assigned as a function of the payoff for each symbol (e.g., average payoff per symbol). Offspring machines are created by randomly mutating each parent machine, i.e., modifying the transition table of the parent machine with respect to a probability distribution. The highest fitness machines are selected to become parents of the next generation, and the process is repeated. (The better of ancestor and offspring survives to be the ancestor of the next generation).[9] This is somewhat different from currently used EP, which is extended by David Fogel (Fogel, 1991), so that mutation can be operated on discrete parameters. Bremermann's idea might be said to be much closer to the original EP.

[6] Bremermann employed a discrete mutation even for continuous variables, since he based on the knowledge about the discrete nature of the genetic code.

[7] It is interesting to note that after his failure of evolving strings to the global optimum, Bremermann (1967) conjectured that many biological species are at a genetic stagnation point, rather than at an optimum (see Fogel, 1995, p.73).

[8] In current EP, however, elements are viewed as behavioral traits of an individual instead of genes along a chromosome (see Fogel, 1995, p.85).

[9] This is a $(1 + \lambda)$ selection in later ES terminology.

Yet another approach to real-valued parameter optimization using simulating evolution was originated at Technical University of Berlin under the name of Evolution Strategy (Evolutionsstrategie in Germany) by Rechenberg (1965), Schwefel (1965) and colleagues, independently from development of GA, EP or other techniques using evolution scheme. The hydrodynamical problems of optimizing shape such as a bended pipe of airfoil (Lichtfuss, 1965) and a flashing nozzle (Schwefel, 1968) or of minimizing drag of a joint plate (Rechenberg, 1965) were addressed by a scheme of *single parent single offspring competition for survival with poorer of the two being eliminated*. Further, Schwefel (1981) developed the use of multiple parents and offspring, following the earlier work of Rechenberg (1973) that used *multiple parents but a single offspring*. More recently multiple μ parents create multiple λ offspring and both compete for survival with the best μ being selected as parents of the next generation.

However, as Mitchel (1996) wrote, these works have been given little or none attention. Or worse, we can find many criticisms for these works. For example, Lenat (1983), cited in Fogel (1995), wrote that

> The early (1958–1970) researchers in automatic programming were confident that they could succeed by having programs randomly mutate into desired new ones. This hypothesis was simple, elegant, aesthetic, and incorrect.

Fogel (1994, p.84) also cited Lindsay's criticism (1968) of early EP, as *"perhaps the most pointed criticism"*:

> Lindsay commented that a random search *" is ... the most inefficient method of problem solving"* and cited the failure of Friedberg (1958) as evidence. He proceeded to claim, incorrectly, that the evolutionary search of Fogel et al. (1966) was no better than a completely random search and concluded by proclaiming the work of Fogel et al. (1966) to be "fustian" that *"may unfortunately alienate many psychologists from the important work being done in artificial intelligence ..."*

as well as other criticism offered by Michie (1970), Chandrasekaran et al. (1976), Jackson (1974), Rada (1981), and Lenat (1983).

1.2 Associative Memory and Evolutionary Algorithms

1.2.1 Associative Memory as a Dynamic System

Associative memory is a dynamical system which has a number of stable states with a domain of attraction around them (Komlós and Paturi, 1988). If the system starts at any state in the domain, it will converge to the stable state. Hopfield (1982) proposed a fully connected neural network model of associative memory in which information is stored by being distributed among neurons and is retrieved from dynamically relaxed neuron states. The dynamical behaviors of its neuron states strongly depend on synaptic strength between neurons. The synaptic strengths between neurons are called *weights*, and weight from neuron j to neuron i is denoted as w_{ij} in this thesis. Hopfield used the Hebbian rule (Hebb, 1949) to prescribe these w_{ij}'s, and succeeded in storing a set of patterns in the network, though the number of patterns are limited to a certain critical amount. Since then, many researchers, mostly physicists, have investigated dynamical behaviors of the model analytically. However, there remain many issues still open. We study them using some variants of evolutionary algorithms.

Evolutionary algorithms search for the optimal or near-optimal solution in a population of candidate solutions from one generation to the next. Each member of the population is assigned a value which indicates how appropriate it is as the optimal solution. The value is referred to as *fitness*. In this thesis, we estimate the fitness value as the degree to how the network stores a set of given patterns as fixed points (see below). Hence, by "optimal solution" we mean the network that stores all the given patterns as fixed points.[10]

1.2.2 Hopfield Model as a Peak in Landscape

The Hopfield model consists of N neurons and N^2 synapses. Each neuron can be in one of two states ± 1. The network is designed to memorize p bipolar patterns,

[10] Therefore, the term "optimal" does not imply "maximum" in number of patterns that can be stored in the network. Given a set of fixed number of patterns, there exist multiple optimal solutions that can store all the patterns as fixed points, unless the number exceeds the storage capacity.

sequences of +1 and –1, namely,

$$\xi^\nu = \{\xi_1^\nu, \xi_2^\nu, \cdots, \xi_{\rm N}^\nu\}, \quad (\nu = 1, 2, \cdots, p),$$

where each ξ_i^ν $(i = 1, 2, \cdots, N)$ takes value of either 1 or –1. These patterns are stored as equilibrium states in the network, in a distributed way, among neurons. We call these states *memorized patterns* to distinguish them from other states. Hopfield employed a discrete-time, asynchronous update scheme; that is, at most one neuron at a time updates its state according to the sign of the weighted sum of all the other neurons' states. That is,

$$s_i(t+1) = sgn\left(\sum_{j\neq i}^{N} w_{ij} s_j(t)\right),$$

where $s_i(t)$ is a state of i-th neuron at time t, and $sgn(z) = 1$ if $z \geq 0$ and -1 otherwise.

Thereupon, an input chosen from the memorized patterns, which is given a small noise within the size of the basin of attraction, should relax to the memorized pattern after several steps of update. If we do not give any noise to the input, all neuron states should remain unchanged from the start, and the patterns are said to be memorized as *fixed points*. Thus the network stores a number of patterns as fixed points when w_{ij}'s are determined appropriately. Hopfield employed the so-called Hebbian rule[11] to specify w_{ij}'s, i.e.,

$$w_{ij} = \frac{1}{N}\sum_{\nu=1}^{p} \xi_i^\nu \xi_j^\nu \, (i \neq j), \; w_{ii} = 0. \tag{1}$$

Then, giving one of the memorized patterns (possibly including a few errors) to the network as an initial state results in a stable state after updating.

As mentioned earlier, the network has an upper limit on the number of patterns to be both stored and recalled properly. Hopfield suggested that by using computer simulation, the maximum number of the patterns to be stored in a network with N neurons is $p = 0.15N$, if a small error in recalling is allowed. Later, this was calculated analytically by Amit, Gutfreund, and Sompolinsky (1985) by

[11] The rule was at first advocated by Hebb (1949) and later formalized by Cooper (1973).

using the spin-glass theory, showing that the storage capacity is $p = 0.138N$. McEliece et al. (1987) proved that when

$$p < N/4 \ln N$$

holds, then the Hopfield model will recall the memory without error.

Now, a question arises. Are there any other ways to specify w_{ij}'s? The specification of synaptic weights is conventionally referred to as *learning*. Borrowing from Kauffman (1993), learning is a walk in synaptic weight space seeking good attractors. Given a set of patterns to be memorized, assigning fitness values to all the possible weight configurations constructs a *fitness landscape* on weight space, and we can imagine the optima as peaks. The Hebbian weight configuration prescribed by Hopfield is one of the peaks. So far, a fair amount of peaks have been found heuristically (see, e.g., Hassoun (1993) and references quoted therein). The configuration prescribed by the *pseudo-inverse matrix method* proposed by Kohonen and Ruohonen (1973), which is an extension of Hebbian learning, is one of the other examples.

The number of peaks depends on the number of given patterns. Gardner (1988) discussed the number of solutions of weight configurations in terms of volume in weight space. She showed that the volume shrinks to zero as the number of patterns approaches to

$$p = 2N,$$

proposing an algorithm to obtain the weight values. The peak obtained by this algorithm is also another example. Paraphrasing from Gardner, there is no (global) peak on the landscape for $p > 2N$. On the other hand, there are multiple peaks on the landscape for $p < 2N$. However, the number and distribution of these peaks are still open issues. So, one of the ultimate goals of this study is to learn the whole geometry of the fitness landscape as a function of the number of to-be-stored patterns, and we believe evolutionary algorithms provide powerful tools for the purpose.

In this thesis, a population of points in weight space explores the landscape[12] starting at the two specific initial points: the Gaussian random weight configu-

[12]In order for a point to explore the landscape, a rule to move the point is needed. Here, the movement is specified according to which connections are pruned. We reported elsewhere more

ration and an *over-loaded* Hebbian one. By over-loaded we mean that the weight values are determined by the Hebb's rule — equation (1) above, with the number of given patterns exceeding the capacity (over-loaded), and hence the memory is damaged more or less. Therefore, the both of these configurations are not peaks. We will attempt to show how evolutionary algorithms lead the initial population toward one of the peaks.

1.2.3 Fitness Landscape on Weight Space and Pattern Space

In analyzing the Hopfield model, there have been basically two different approaches: one is to explore *pattern space* searching for attractors under a specific weight configuration, and the other is to explore *weight space* searching for an appropriate weight configuration that stores a given set of patterns. To be more specific, the former is an analysis of the Hamiltonian energy as a function of all the possible configurations of bipolar patterns given to the network, where the synaptic weights are pre-specified using a learning algorithm, usually the Hebb's rule, so that the network stores a set of p given patterns. In this context, the model for $p = 1$ corresponds to the Mattis model of spin-glass (Mattis, 1976), in which the Hamiltonian energy has two minima, while the model for infinitely large p corresponds to the Sherrington-Kirkpatrick model (1975), in which the synaptic weights become Gaussian random variables. Analyses of the former type have been made in between these two extreme cases (see (Amit, 1989)). The latter analysis was addressed by Gardner (1988). She discussed the optimal weight configurations for a *fixed* number of given patterns in terms of the volume of the solutions in weight space, suggesting that the volume shrinks to vanish when p approaches to $2N$. In short, the former approach searches for the optimal pattern configurations which minimize the Hamiltonian energy *in pattern space with the weights being fixed*, while the latter searches for the weight configurations *in weight space* that optimally store a set of given *fixed patterns*.

So far, we have studied the model with the latter approach. We have explored fitness landscapes of the model *defined on weight space*, and have found many solutions that store more patterns or store them with larger basin of attractions

direct exploration by adding a small random value η_{ij} to each w_{ij} (see Imada et al. (1997d; 1997e; 1997g)).

than, e.g., the Hebbian synaptic weights (e.g. (Imada et al., 1997a; 1997b)). Now, our interest is on the number and distribution of these solutions over the whole weight space, which is still an open problem. We think the niching GA is one of the appropriate tools to pursue these problems. However, since N^2-dimensional *continuous* weight space is much more difficult to wander around than N-dimensional *discrete* pattern space, we explored the *pattern space* instead to see preliminary how our fitness function works under the niching method (Imada et al., 1998f). In other words, we used the model as a test function of the niching technique in the sense that all solutions are *a priori* known, like in other studies using pure mathematical test functions.

1.3 Overview of this Thesis

In this section we have described what are the artificial evolutions and what is the associative memory. The remainder of this paper begins by introducing some theoretical aspects of Genetic Algorithms. Following sections develop a series of experiments. Since the application of evolutionary algorithms to the fully-connected neural network model of associative memory is quite a new experiment, various versions of evolutionary algorithms are tested to see how they evolve real-valued weights of the fully-connected neural networks. Each variant of the evolutionary algorithms are described in detail together with the results in Chapter 3. We also evolve weights by pruning some of the synaptic weight connections adaptively. We obtained somewhat of an amazing result that a network with randomly determined weights eventually evolves to store some of the patterns just by pruning some of the connections adaptively. The method and the results are given in Chapter 4. Natural creatures usually have an ability to learn how to adapt to their environment and to increase their fitness. Then how about in the artificial evolution? Chapter 5 addresses a relationship between learning during lifetime and evolution. Two hypothesis in evolutionary biology: the Lamarckian inheritance and the Baldwin effect are studied in the context of artificial evolution. Thus in Chapter 3, 4, and 5, we find a variety of weight configurations that give a network a function of associative memory by applying a lot of different versions of evolutionary algorithms. In Chapter 6, these weight configurations are studied in terms of two parameters of weights: the degree of

symmetry and dilution ratio. Associative memory stores information as attractors, and the attractors has their basin of attraction. This is why the associative memory has a tolerance for noises. Chapter 7 examines the basin of attraction of the weight configurations obtained by previous chapters. The main goal of this thesis is to study the number and distribution of the solutions in weight space. Toward this goal, the next two chapters explores high dimensional weigh space in two different ways. That is, Chapter 8 studies "Fitness Landscape", the hypersurface defined on the high-dimensional weight space, and Chapter 9 visualize the weight space by reducing dimensionality with maintaining the information on distances among solutions in weight space as much as possible. Throughout our experiments, we noticed that the exploration of the fully-connected neural network model has many good properties for a benchmark to test and evaluate evolutionary algorithms. This issue is described in Chapter 10. The concluding remarks and future works are given in Chapter 11.

2. GA THEORY

The *Schema Theorem* and its corollaries, the *Building Block Hypothesis* as well as the *Implicit Parallelism*, which are argued by Holland (1975) in his book, are said to be essential to understand how GAs work. However these concepts were devised only for the GA using binary chromosomes or at least discrete chromosomes. Since we do not use this type of GA in this thesis with an exception of Chapter 4, we describe the Schema Theorem and Building Block Hypothesis only briefly and we focus our discussion here mainly on convergence issue.

2.1 Basic Concepts of Holland's GA

Holland (1968) introduced the concept of *schema* to define *building blocks*. A schema is a string made up of ones, zeros, and asterisks. The asterisks are used for representing either of one and zero when we *don't care* which, and hence they are called "*wild cards*" or "*don't cares*". Thus a schema is a template that represents a set of bit strings. For example, a schema $(1**0)$ represents four strings $(1000), (1010), (1100)$, and (1110), since these four are all four-bit binary strings that begin with 1 and end with 0. Namely, schema defines a i-dimensional hyper-plane in the n-dimensional space of n-bit strings ($i \leq n$). In his book, Goldberg (1989, p.33) summarized Holland's Schema Theorem as:

Theorem 1 (Schema Theorem) *Short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations.*

This suggests that *selection* gradually confines the population on subsets of the search space that are defined by schemata with above-average fitness. Then Goldberg (1989, p.41) continued to summarize the Building Block Hypotheses as:

> Just as a child creates magnificent fortresses through the arrangement of simple blocks of wood, so does a Genetic Algorithm seek near optimal performance through the juxtaposition of short, low-order, high-performance schemata, or *building blocks.*

This suggests that *crossover* puts high-fitness building blocks together on the same string to create strings of increasingly higher fitness. Furthermore, he wrote (Goldberg, 1989, p.40):

> $\cdots$ despite the processing of only n structures each generation, $\cdots$ we get useful processing of something like n^3 schemata in parallel with no special bookkeeping or special memory other than the population itself.

Thus, in the early 1990's the *Schema Theorem* and the *Building Block Hypothesis* were almost the only basis of theoretical work on how Genetic Algorithms work.

2.2 Convergence

The *schema theorem*, however, does not provide any guarantee for convergence to the optimum solution (Beyer, 1996). Then how can we know when and on what problems do GAs work? In this section, we describe the conditions under which an evolutionary algorithm converges to an optimum with probability one.

2.2.1 Banach Theorem

In his book, Michalewicz (1996, pp.68–72) argued the convergence of a Genetic Algorithm based on Banach's fixed point theorem, which is one of basic theorems in the field of function analysis in Mathematics. The Banach theorem states that:

Theorem 2 (Banach's Fixed Point Theorem) *Any contractive mapping on a complete metric space has a unique fixed point.*

We now describe how the theorem is applied to evolutionary algorithms. Suppose the set S consists of all possible populations, i.e.,

$$S = \{P_1, P_2, P_3, \cdots\}.$$

We can regard S as a metric space by defining distance $\delta : S \times S \to \Re$ as[13] :

$$\delta(P_i, P_j) = \begin{cases} 0 & \text{if } P_i = P_j, \\ |1 + M - \overline{f}(P_i)| + |1 + M - \overline{f}(P_j)| & \text{if } P_i \neq P_j, \end{cases} \tag{2}$$

[13] If $\delta(x, y)$ satisfies for $\forall x, y \in S$, (1) $\delta(x, y) \geq 0$ and $\delta(x, y) = 0$ iff $x = y$; (2) $\delta(x, y) = \delta(y, x)$; (3) $\delta(x, y) + \delta(y, z) = \delta(x, z)$ then δ is called a *distance* and S is called a *metric space*.

where $\overline{f}(P)$ is the average fitness of the population P, and M is possible maximum value of $f(P)$. In evolutionary algorithms, a single iteration of generating the next generation from one generation can be regarded as a mapping

$$F : S \to S.$$

Thus we can say that if both (1) the metric space S is *complete* and (2) the mapping F is *contractive* on the metric space S hold, then F has a unique fixed point. In order for a metric space to be complete, any Cauchy sequence on the space should have a finite limit. In our context, the condition of *completeness* is satisfied if any Cauchy sequence of populations has a finite limit $P^\star$. Note here that a sequence $P_1, P_2, \cdots$ is referred to as Cauchy sequence iff there exists k for any ϵ such that $\delta(P_m, P_n) < \epsilon$ for all $m, n > k$. This is always satisfied in our case, since we consider a finite number of populations out of all possible populations. In fact, for any Cauchy sequence of populations $P_1, P_2, \cdots$, we always have the number k such that $P_n = P_k$ for all $n > k$. On the other hand, a mapping F is *contractive* iff there exists $\epsilon \in [0, 1)$ such that $\delta(F(x), F(y)) \leq \epsilon \cdot \delta(x, y)$ for all $x, y \in S$. Hence if our $\overline{f}$ satisfies

$$\overline{f}(P(t)) < \overline{f}(P(t+1)), \tag{3}$$

then the mapping F is *contractive*. Indeed,

$$\delta(F(P_1(t)), F(P_2(t))) < \delta(P_1(t), P_2(t)) \tag{4}$$

is almost clear from the equation (2). Note that we assume maximization problem here without any loss of generality. The equation (3) implies that the population is improved in terms of average fitness value. Thus, we can conclude that if the iteration $F : P(t) \to P(t+1)$ is *contractive* on the *complete* metric space S, the space of possible populations, the evolutionary algorithm converges to the population $P^\star$ which is a unique fixed point in S. That is,

$$P^\star = \lim_{i \to \infty} F^i(P(0)),$$

where

$$F^{i+1}(P(0)) = F(F^i(P(0))) \quad \text{and} \quad F^0(P(0)) = P(0).$$

If we recall $\overline{f}(P)$ is an average fitness of P, we recognize that $P^\star$ is a population in which elements in P are all identical. Obviously, the element of $P^\star$ is one of the possible global optima. Moreover, $P^\star$ is independent on starting population $P(0)$. The initial population only affects the convergence speed. To simply put, if average fitness of the population keeps increasing from one generation to the next, then the evolutionary algorithm is guaranteed to converge to $P^\star$, which is a unique fixed point in the space of all possible populations. However, these conditions are not so general. For example, if the fitness evaluation f is multi-modal, that is, f has more than one maximum, the mapping F is not contractive since for the two optimal populations $\delta(F(P_1(t)), F(P_2(t))) = \delta(P_1(t), P_2(t))$ holds instead of Equation(4).

2.2.2 Markov Chain Model

The evolutionary algorithms can be also formulated as a finite-dimension Markov chain. Markov chain is characterized by a finite number of states and transition probabilities from one state to another. The probability of transitioning from state i to state j in one step is denoted as p_{ij} here. If a state can not be transitioned to any other state either in one step nor multiple steps, the state is said to be *absorbing*. For evolutionary algorithms, the states of the chain can be defined by every possible configuration of an entire population of bit strings (Fogel, 1995, p.126). Fogel shows that the chain generated by a Genetic Algorithm with only crossover and selection (no mutation) will transition with probability one to an absorbing state. Note however that the absorbing state is a state in which the states in the chain are all identical since crossover in this case cannot modify these states (recall that this is a Genetic Algorithm without mutation). Hence an absorbing state does not necessarily implies the global optimum, but may be one of local optima.

Fogel proceeds the argument to EP in which the state is a real valued vector. By forming an equivalent class of all states that contain a global best vector and describing the class as a single state, the state containing global optima can be regarded as the only absorbing state. Thus Fogel (1992; 1994) shows that the probability of reaching a global optimum is one.

Rudolph (1994) also used the Markov chain analysis to argue convergence issue of simple Genetic Algorithm.[14] He showed a sequence of solutions generated does not converge to any point including any global optima, but the global convergence is always guaranteed by maintaining the best solution in the population into successive generations (i.e., elitist strategy).

Many other researchers have also approached the convergence properties of Genetic Algorithms using Markov chain analysis (see e.g., (Goldberg et al., 1987), (Davis et al., 1991), (Eiben et al., 1991), (Nix and Vose, 1991), (Vose and Liepins, 1991), and (Vose, 1992))

The analysis of GAs taking it for a Markov process captures an exact microscopic behavior of GAs. However deriving macroscopic dynamical equation, such as the expected fitness change over time, from the transition matrices of the Markov chain has not been successful (Beyer, 1994).

2.2.3 Beyer's Hypothesis

Beyer (1995)[15] gave yet another approach in between *macroscopic* and *microscopic* level of evolutionary algorithm, denoting it *mesoscopic*. To understand how an evolutionary algorithm works in general, he proposed three basic principles: *Evolutionary Progress Principle*, *Genetic Repair Hypothesis*, and *Mutation Induced Species by Recombination Principle*, instead of Holland's *Schema Theorem*, *Building Block Hypothesis*, and *Implicit Parallelism*. We introduce here only the convergence aspects from his proposition.

We now assume a function optimization of N parameters. The average distance of each individual to the global optimum should decrease from one generation to the next. If we denote $R(t)$ as the average distance of each individual to the

[14] Namely, fixed length binary genes, one point crossover, bit-flipping mutation, and proportional selection.

[15] This paper is somewhat unusual as he said *"This paper has been rejected for presentation at the 6th ICGA. Having lost the opportunity to defend the new ideas and the points attacked by the reviewers at the conference, the author has added his replies to the reviewers comments."*

global optimum at generation t, the expectation of $R(t+1) - R(t)$ are called the progress rate $\varphi(t)$, i.e.,

$$\varphi(t) = E\{R(t+1) - R(t)\}.$$

Beyer (1994) derived the asymptotic ($N \to \infty$) formulae of this progress rate for (μ, λ)-ES[16],

$$\varphi^*(t) = c \cdot \sigma^*(t) - (\sigma^*(t))^2/2, \tag{5}$$

where c is called progress coefficient, and φ^* and σ^* are normalized as

$$\varphi^*(t) = \varphi^*(t) \cdot N/R(t), \quad \text{and} \quad \sigma^*(t) = \sigma^*(t) \cdot N/R(t).$$

The equation (5) suggests that

- progress is positive (i.e. converge to the optimum) when $0 < \sigma^* < 2c$;
- the maximal progress $c^2/2$ is achieved when $\sigma^* = c$.

If we re-normalize σ then the condition of the maximal progress becomes

$$\sigma = c \cdot R/N, \tag{6}$$

indicating that

$$\lim_{t\to\infty} \sigma(t) = 0 \Rightarrow \lim_{t\to\infty} R(t) = 0.$$

That is, convergence to the optimum. The condition $\lim_{t\to\infty} \sigma(t) = 0$ implies that mutation should be infinitely small at the last stage of the search.

2.2.4 Global Random Search

In his book, Bäck (1996, p.48) started the discussion of the issue of convergence by noting *uniform random search*:

Algorithm 1 (Uniform Random Search)

1. *Set current-max-fitness to* $-\infty$.
2. *Sample* x *uniformly at random.*

[16] That is, λ offspring are generated from μ parents ($\mu < \lambda$) via mutation and crossover, and the μ best offspring are selected for the next generation.

3. *Evaluate fitness of x.*
4. *If the fitness is larger than current-max-fitness, replace the current-max-fitness with fitness of x, otherwise do nothing.*
5. *If a set number of iterations has been performed, return the current-max-fitness. Otherwise go to step 2.*

Then Bäck showed the sequence $x_1, x_2, \cdots$ generated by Algorithm 1 converges to the global optimum $x^\star$ with probability one, by citing the proof from Zhigljavsky (1992, pp.78–79).

Theorem 3 (Zhigljavsky's Theorem) *If a Genetic Algorithm satisfies:*

(1) for a sequence of populations, the best fitness at each generation is monotonically increased;

(2) for $\forall x, x' \in M$, x' is reachable from x by mutation and recombination.

Then the sequence of populations converges to a global optimum with probability one.

However, as Bäck wrote, Algorithm 1 is not evolutionary in that the probability distribution remains constant throughout iterations. In evolutionary algorithms, in general, the probability distribution function with which members of the population are selected as parents should be determined by fitness of the members, and therefore the distribution of the selected parents changes from generation to generation.

Bäck proceeds to the Zhigljavsky's global random search algorithm (Zhigljavsky, 1992) as the basis of proofs of global convergence of various evolutionary algorithms. The algorithm allows us to construct a new probability distribution at each iteration. The algorithm can be paraphrased as:

Algorithm 2 (Global Random Search)

1. *Choose a probability function. Call this function current-probability-function.*
2. *Sample x according to the current-probability-function.*
3. *Evaluate the fitness of x.*

4. *Renew the current-probability-function by constructing a new probability function using a pre-fixed rule.*
5. *If a set number of iterations has been performed, return the current-max-fitness.*

The renewed probability distribution at each iteration may depend on the result of the previous iteration, and hence force this algorithm encompasses certain versions of evolutionary algorithms (Bäck, 1996, p.87). This algorithm produces a sequence of vectors

$$x_1, x_2, \cdots$$

and we can calculate the probability with which each x_i falls into $U_\epsilon(x^\star)$, the ϵ-environment.[17] of the global optimum $x^\star$, i.e.,

$$\wp(x_i \in U_\epsilon(x^\star))$$

which depends on the current-probability-distribution in Algorithm 2 Now, let $q_t(\epsilon)$ be the infinimum of this sequence of probability, i.e.,

$$q_t(\epsilon) = \inf_{1 \le i \le t} \{\wp(x_i \in U_\epsilon(x^\star)\}.$$

Then the convergence theorem is summarized as:

Theorem 4 (Zhigljavsky's Convergence Theorem) *Let f be continuous in the vicinity of $x^\star$ and assume that for $\forall \epsilon > 0$,*

$$\sum_{t-1}^{\infty} q_t(\epsilon) = \infty.$$

Then the sequence of vectors $x_1, x_2, \cdots$ generated by Algorithm 2 falls infinitely often into the vicinity[18] of $s^\star$ with probability one.

By employing this Zhigljavsky's Convergence Theorem, Rudolph (1992) proved the convergence of the so-called (1+1)-Evolution Strategy. In (1+1)-Evolution Strategy, one parent generates one child by mutation and better of the two survives. Mutation is done using Gaussian random variable with standard deviation being modified by Rechenberg's 1/5 rule (Rechenberg, 1973). To be more specific,

[17] $U_\epsilon(x^\star) = \{x \mid \|x - x^\star\| < \epsilon\}$

[18] Instead of exact value of $x^\star$. This is due to the representation of real numbers by digital computers. Bäck uses the notation $L_{f^\star + \epsilon} = \{x \mid f(x) \le f(x^\star) + \epsilon\}$, noting that the global maximization problem can be considered as solved if a member of the set $L_{f^\star + \epsilon}$ has been found.

Algorithm 3 ((1+1) Evolution Strategy)

1. *Initialize x.*
2. *Evaluate fitness of x.*
3. *Mutate x by adding a Gaussian random variable with mean zero and standard deviation σ.*
4. *Evaluate fitness of the mutant x'.*
5. *Select either x or x' according to their fitness.*
6. *If the ratio of successful mutations so far is greater than 1/5, then increase σ by multiplying pre-fixed constant c. If the ratio is smaller than 1/5, then decrease σ by dividing by c.*
7. *If a set number of iterations has been performed, return the current-max-fitness. Otherwise go to step 2.*

2.2.5 Logarithmic Convergence Theorem

Vose and Wright (1994) formalized simple GA as an instance of more general stochastic search, *Random Heuristic Search*, and later Vose (1996) gave a general convergence theorem for this class of search. In this section, we briefly review the theorem.

Random Heuristic Search is a search in which a transition rule τ successively generates P_{i+1} by applying τ to P_i starting with P_0, a collection of elements chosen from the search space Ω, until a stopping criterion is satisfied. P_i is called a population. To obtain an appropriate representation for the population to characterize τ mathematically, Vose defined a *simplex*

$$\Lambda = \{(x_0, \cdots, x_{2^l-1}) \mid x_j \in \Re, \quad x_j \geq 0, \quad \sum x_j = 1\},$$

where x_j is an occurrence ratio of the element of Ω whose binary representation is decimal j. An element p of Λ, a vector which is comprised of 2^l decimal numbers, represents a population. In the population, if the binary expression of an individual is translated into decimal integer j, then the proportion of the

individual contained in the population is x_j. For example, if

$$P = \{(1,0,1),(0,0,1),(1,0,1),(1,1,1)\} \in \Omega$$

then

$$p = \{0, 1/4, 0, 0, 0, 1/2, 0, 1/4\} \in \Lambda.$$

Thus a function $G : \Lambda \to \Lambda$ produces a new population from current population. So we might say that $G(p)$ is a *bias* according to which search space is explored. We call the function $G(p)$ *a heuristic function*. Since

$$G(\lim_{l\to\infty} G^l(p)) = \lim_{l\to\infty} G^{l+1}(p) = x$$

holds, x satisfies $G(x) = x$ and is called fixed points of G and denoted as $\omega(x)$.[19]

Note here that *time to convergence* is defined as the time taken by

$$G(x), G^2(x), G^3, \cdots$$

to reach within δ of $\omega(x)$. It is easy to see that the time to convergence depends on the initial population and it is almost certain that there exist some sequences of populations that diverge and hence the time to convergence is ∞. To state this formally, we first define a term "*focused*" as follows:

Definition 1 (Focused Heuristic Function) *A random heuristic search G is referred to as focused if G is continuously differentiable and if the sequence*

$$p, G(p), G(G(p)), \cdots$$

converges for any $p \in \Lambda$.

Then our intuition of the existence of sequences that diverge leads to the following theorem and proof.

Theorem 5 *The time to convergence cannot be uniformly bounded.*

[19] We use this term later in somewhat different context. Do not be confused.

Proof. Consider a *focused* random heuristic search G. Let u and v be distinct fixed points, and $s(t) = tu + (1-t)v$. Further, let t^* be the *supremum* of $t \in [0,1]$ such that $\omega(s(t)) = v$. Now assume that the time to convergence to v are *uniformly bounded* by k. Then $G^k(s(t^*))$ is mapped within a distance δ from v for small enough δ, hence converges to v. This is because G^{*k} is *uniformly continuous* due to the fact that G^k is continuous and Λ is *compact*. But this contradicts the assumption that t^* is the supremum, because the same continuity arguments imply that an open *neighborhood* of t^*converges to v. Therefore the time to convergence cannot be uniformly bounded. □

However it is quite possible that time to convergence is uniformly bounded for *almost all* of the initial populations. To formalize the term "*almost all*", we define the concept of *the logarithmic convergence of the infinite population algorithm.*

Definition 2 (Logarithmic Convergence) *If there exists a set A such that the number of generations required for the initial population $p \in A$ to satisfy $\|G^k(p) - \omega(p)\| < \delta$ is $O(-log\delta)$ where $0 < \delta < 1$, then the algorithm is said to converge in logarithmic time.*

We also assume here a probability density ρ is given over Λ and defines the probability that for any $A \subset \Lambda$ the initial population is contained in A, as $\int_A \rho d\lambda$ where λ is a Lebesgue measure. We now can define the expression of "*almost all*" above. If the probability that the initial population $p \in A$ results in the logarithmic convergence is at least $(1-\epsilon)$ for $0 < 1 \ll 1$, then we may say that time to convergence is uniformly bounded for "*almost all*" of the initial populations.

Before proceeding further, we make several more definitions.

Definition 3 (Hyperbolic Fixed Point) *A fixed point x is hyperbolic if any eigenvalue of the differential of G at x is neither 1 nor –1.*

Definition 4 (Well Behaved Heuristic Function) *G is said to be well behaved if $C \subset A$ has measure zero implies that $G^{-1}(C)$ also has measure zero.*

We now can describe Vose's general convergence theorem for *Random Heuristic Search* of which simple GA is an instance. The theorem is formalized as:

Theorem 6 (Logarithmic Convergence Theorem) *If G is focused, well behaved, and its fixed points are hyperbolic, then the infinite population algorithm converges in logarithmic time.*

3. EVOLUTION OF WEIGHTS

As we described earlier, an associative memory is a dynamical system. The dynamical behaviors of a fully-connected neural network are strongly dependent on weight values. In this chapter, we argue how these weight values can be determined in terms of evolutionary processes so that they give a network a function of associative memory.

When an evolutionary algorithm is applied to a problem, each of candidate solutions should be represented by a single vector. Borrowing terminology from evolutionary biology, researchers in evolutionary algorithm's community have labeled the vector a *chromosome*[20] and its components

genes. A set of these chromosomes is called a *population*, and the members of population, i.e. chromosomes, are sometimes called *individuals*.

In our problem of evolving synaptic weights, the goal of the evolution is to produce a network which stores a set of given patterns as associative memory. Then how are the candidate solutions represented? The most straightforward way is to represent the weights *per se* as genes in a chromosome. In this case, a population of these chromosomes undergoes evolution and hopefully yields a solution of the best performance. Alternatively, a population of candidate weight configurations can be obtained by modifying one *a priori* determined weight configuration. For example, a new configuration of weights is generated by adding a small perturbation on each weight value of a fixed network. In this case, genes are made up of these small perturbations and it is these perturbations instead of weights that are evolved.

In any case, a population is constructed with a number of these chromosomes, and the population is initialized at the beginning of a run.[21] Then evolutionary processes such as *crossover* and *mutation* are operated on these chromosomes, which produces their offspring. Each of these offspring is estimated the capabil-

[20] A string representing an individual is also referred to as its genome.

[21] Usually each gene takes one of the alleles at random and therefore the search starts with N different randomly chosen individuals. However, the search sometimes starts with individuals in the first population being all identical. We describe the issue of this initialization later in this chapter in more details.

ity to store a set of given patterns which is called a *fitness* of the individual, and some of these offspring are *selected* according to their fitness values to survive to the next generation. In this thesis, selection, crossover and mutation are implemented as follows, unless otherwise stated.

Selection. Two parent chromosomes are chosen *randomly* from the best $T\%$ of the population. They mated randomly (except for itself) and are recombined with crossover to produce one offspring which is mutated occasionally (see below). This is repeated until the number of offspring is equal to $(100-T)\%$ of the population. The worst $(100-T)\%$ of a population are replaced with these offspring. And these offspring and the remaining the best $T\%$ parents survive to the next generation. This selection is referred to as *truncate selection* (Mühlenbein et al., 1995).

Crossover crosses two parents $(u_1, \cdots, u_n)$ and $(v_1, \cdots, v_n)$ to produces an offspring $(w_1, \cdots, w_n)$ such that w_i is either u_i or v_i with equal probability. This crossover is referred to as *discrete recombination* (Mühlenbein et al., 1995).[22]

Mutation modifies genes of these produced offspring chromosomes. Typically, in the case of continuous genes, the modifications are made by replacing a gene chosen at random with probability p_m with a real number taken uniformly at random from $[-1, 1]$.

The cycle of reconstructing the new population with better fitness individuals and restarting the search is repeated until one of the global optima is found or a set maximum number of generation has been reached.

Fitness Evaluation. Before proceeding further, we now look at how the fitness evaluations are made. The task of each individual is to make a network store a set of p random bipolar patterns ξ^ν $(\nu = 1, 2, \cdots, p)$ that is *a priori* determined before a run. When one of the patterns ξ^ν is given to the network as an initial state, possibly including a few errors, the state of neurons varies from time to

[22] This nomenclature is when genes take real values. If genes take discrete values, this is referred to as *uniform crossover* (Syswerda, 1989)

time afterwards. In order for the network to function as an associative memory, these instantaneous states $s_i^\nu(t)$ must be similar to the initial state. The similarity as a function of time is defined by

$$m^\nu(t) = \frac{1}{N}\sum_{i=1}^{N} \xi_i^\nu s_i^\nu(t).$$

This is conventionally called an *overlap*. If an instantaneous state of the network is equal to the input, then the overlap takes the value 1, while if they are uncorrelated, the value will be 0. As the network enters an attractor, the values of the overlap become constant. Thus the quality of retrieval of memorized patterns is represented by taking a temporal average of $m^\nu(t)$ over a certain time interval t_0.[23] This is denoted as $\langle m^\mu \rangle$. We evaluate the fitness value of a network by further averaging these $\langle m^\mu \rangle$'s over all memorized patterns. Namely, our objective function f is

$$f = \langle\langle m \rangle\rangle = \frac{1}{p \cdot t_0}\sum_{\nu=1}^{p}\sum_{t=1}^{t_0} m^\nu(t). \tag{7}$$

In this thesis, t_0 is set to $2N$, twice the number of neurons. Note that the fitness 1 implies all the p patterns are stored as fixed points, while all other cases have a fitness less than 1. Then our goal is to optimize $\mathbf{w} = (w_{11}, w_{12}, \cdots, w_{NN})$ such that $\langle\langle m \rangle\rangle$ takes the maximum value 1.

Initialization. In evolutionary computations, the issue of initialization is very important, though little attention has been paid. Usually, the first population is initialized at random. Namely, individuals are uniformly distributed at random in the search space at the beginning.

What then are the possible initializations with our fully connected neural network model? Essentially, we start algorithms with three different initial population: random weight matrices, over-loaded Hebbian matrices, and zero matrices.

[23] Instead of the above time consuming evaluation (7), the overlap of the ultimate state

$$\frac{1}{p}\sum_{\nu=1}^{p} m^\nu(\infty)$$

might seem to be enough. However, our experiment shows that this fitness evaluation often yields limit cycle solutions.

In the first case, individuals are distributed randomly in weight space, while in the second and third cases all individuals are identical.

An input given to the network with random weight will result in a chaotic trajectory due to the asymmetry of the weight matrix. An input to the over-loaded Hebbian matrix, on the other hand, will converge to a stable attractor in between the fixed-point attractor and spin-glass attractor (stable but far from the initial state). These behaviors are visualized, for instance, by plotting over time the Hamming distances between the initial state given to the network and every instantaneous network state. We referred to this state transition as trajectory. Examples of the trajectories are shown in Figure 1.

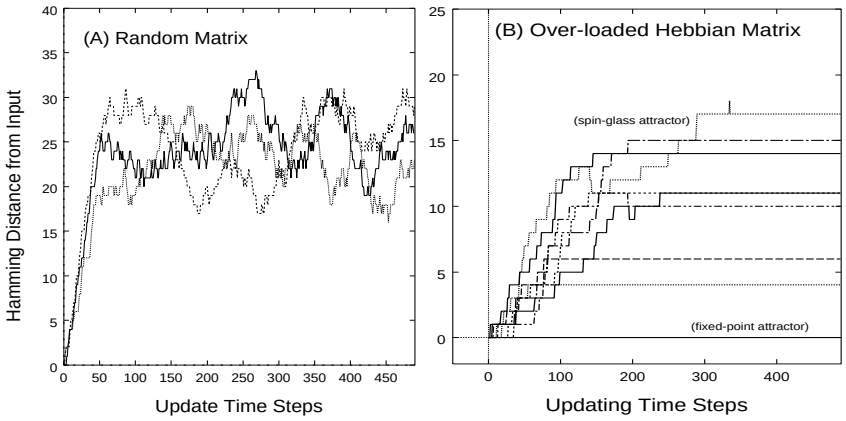

Figure 1. Typical trajectories before evolution.

Note here that our fitness evaluation relates to the sum of areas under these trajectories, and the goal is to minimize these areas.[24]

In the following several sections, we describe some variants of evolutionary algorithm implementation that employ real-valued genes to obtain the optimal weights of fully-connected neural network model of associative memory.

[24]When the areas become zero, all the given patterns are stored as fixed point attractors, or equivalently, the algorithm found an appropriate configuration of weights that creates fixed point attractors exactly at the location of given patterns.

3.1 Evolutionary Algorithms with Real-valued Encoding

GAs usually evolve binary strings. As Michalewicz (1996, p.97) wrote, this is due to the fact that the binary representation *"offers the maximum number of schemata per bit of information of any coding"* and *"facilitates theoretical analysis and allows elegant genetic operators."* However, as Michalewicz also pointed out, the binary encoding *"has some drawbacks when applied to multi-dimensional high-precision numerical problems."* And determining weights of neural networks is one such problem. As alternatives, some GAs that evolve continuous genes have been proposed (e.g., (Goldberg, 1990), (Wright, 1991)). Though specific binary encodings such as Gray coding were also devised for these problems (Caruana et al., 1988), Michalewicz suggests that *"the results of real encoding GAs are better than those from binary representation."* In this thesis, we mainly focus on evolutions of real-valued genes.

In the classical analysis of the Hopfield model, Sompolinsky (1986) gave small perturbations on the Hebbian synaptic weights to see the robustness of the networks for the synaptic noise. In other words, each Hebbian synaptic weight J_{ij} was added small perturbation η_{ij}, resulting in

$$J_{ij} + \eta_{ij}.$$

We also modify synaptic weights based on this scheme using evolutionary algorithms, with the difference being that Sompolinsky chose perturbations *randomly*, while we choose them *adaptively*. We have two different versions of giving perturbations. One is by chromosomes and the other is by mutations. Note that *perturbation by chromosome* exploits a promising region, i.e., search points are restricted within a certain region (volume-oriented), while *perturbation by mutation* explores new regions, i.e., search points may wander all over the search space (path-oriented).

3.1.1 Direct Mapping of Weights onto Genes: Perturbation of Weights by Mutation

Here, we evolve real-valued weights that are directly mapped onto genes in a chromosome (Imada et al., 1997a). In this implementation, each weight value of a configuration is encoded directly to each gene of a chromosome. To be more

specific, the ij-th entry of a weight matrix corresponds to the $\{N(i-1)+j\}$-th gene of a chromosome using the standard *row-scan method*. Namely,

$$x_{N(i-1)+j} = w_{ij},$$

where $x_{\{.\}}$ represents a gene value.

The algorithm may start by initializing gene values in the following three ways. (1) All genes are chosen randomly from $[-1, 1]$. (2) All genes are set to zero. (3) Weights calculated by the Hebb's rule are assigned to the corresponding genes.

Then they undergo recombination, mutation and selection from one generation to the next.

3.1.2 Modifying of an Ancestral Matrix: Perturbation of Weights by Chromosomes

In this implementation, we determine a matrix at the beginning of a run as an ancestor. This matrix remains unchanged during evolution. In each generation, chromosomes composed of small perturbations η_{ij} produce copies of the ancestor. These copies are slightly different from the ancestor depending on the range of η_{ij}. We may start by determining the ancestral matrix as either of a random matrix, zero matrix, and over-loaded Hebbian matrix (Imada et al., 1997b; 1997f), Each chromosome is defined as a N^2-dimensional real-valued vector, representing small perturbations to be added to elements of the ancestral weight matrix. We denote the chromosome as

$$(\eta_{11}, \eta_{12}, \cdots, \eta_{1N}, \eta_{21}, \cdots\cdots, \eta_{NN}).$$

These chromosomes are initialized randomly at the beginning of a run. In each generation, a population of chromosomes make ancestor's copies by adding each gene to its corresponding component of the ancestor matrix. Namely,

$$w_{ij}^k = W_{ij} + \eta_{ij}^k,$$

where w_{ij}^k is ij-component of the copied matrix, η_{ij}^k is $(iN+j)$-th gene of the chromosome, and the superscript k denotes the individual number in the population.

η_{ij} is taken each time from a Gaussian distribution of mean zero and standard deviation σ.

The fluctuation σ plays a similar role to p_m, the bit-flipping probability in standard canonical GA (Beyer, 1995).

The chromosomes are modified through crossover and mutation operation. The fitness values of the corresponding phenotype are evaluated. According to the fitness values, individuals of the next generation are selected. The rests of the procedure are similar to those described in the previous section.

3.1.3 Evolutionary Programming (EP)

We may also use EP to evolve real-encoded weights (Imada et al., 1997d; 1998a). As stated in Section 1.1.2, EP is a technique for optimization, proposed by L. Fogel, Owens and Walsh (1966) (see also D. Fogel (1995)).

In EP, we employ a Gaussian mutation in which Gaussian variables with mean zero and small variance are added to genes. This is an elaborate version of simple *hill climbings*. Typically in a simple *hill climbing*, the Gaussian mutation is given λ times to the current point, and among these λ points, the point obtained the highest fitness value is selected as the next point. In EP, on the other hand, μ points construct a population. Each of these μ points is mutated once, which produces μ mutants. The mutation in EP is adaptive. That is to say, each point has additional variables σ_i for each coordinate, which are used as a standard deviation of a Gaussian random variable to be added to the i-th coordinate of the point. After the mutation, these σ_i are also modified as:

$$\sigma_i^{\text{new}} = \sigma_i^{\text{old}} + 0.01 \cdot \sigma_i^{\text{old}} \cdot N_i(0,1),$$

where $N_i(0,1)$ is normally distributed random variable of mean 0 and standard deviation 1 sampled for each individual i.

All the σ_i are initialized to σ_0 ($= 0.002$ here) at the beginning of a run. Now, we have the original μ points and their μ mutants. The fitness value of each of the 2μ points are compared to those of q points which are chosen randomly at every time of the comparison from the whole 2μ points. Then the 2μ points are ranked according to the number of wins, and the best μ points survive (q-tournament selection). Note that EPs do not use crossover operations.

3.1.4 Evolution Strategy (ES)

ES has some similarities with EP in the sense that ES also employs *adaptive mutation scheme.* But it also has an important difference from EP, that is, ES uses crossover operation like GA. We apply ES to our problem of evolving real-valued weights (Imada et al., 1997e).

At the beginning of a run, a population of chromosomes are initialized in the same way as the previous three implementations described in this section. In each generation, these chromosomes are modified through crossover and mutation operation. According to the fitness values, individuals of the next generation are selected using a $(\mu + \lambda)$-strategy.

To compare the results with those of EP, we use a mutation similar to EP. That is, we mutate object variable x_i and strategy parameter σ_i as follows:

$$\begin{cases} x_i' = x_i + \sqrt{f_i} \cdot N_i(0,1) \\ \sigma_i' = \sigma_i + \sqrt{0.01 \cdot f_i} \cdot N_i(0,1), \end{cases}$$

instead of usual mutation in ES:

$$\begin{cases} x_i' = x_i + \sigma_i' \\ \sigma_i' = \sigma_i + \exp N_i(0,1). \end{cases}$$

3.1.5 Breeder Genetic Algorithm

The Breeder Genetic Algorithm (BGA) were proposed by Mühlenbin et al. (1996). The BGA is based on "*artificial selection similar to that used by human breeder*" expecting "*artificial selection to be more efficient for optimization than natural selection*," as Mühlenbin et al. stated. Mühlenbein et al. applied the BGA to multi-modal test functions and concluded that "*the BGA mutation scheme is able to optimize many multi-modal functions.*" The free connected neural network model of associative memory has multi-modal solutions. That is, there exist multiple weight configurations to store a fixed number of given patterns if the number is less than the capacity. Hence, we applied the BGA to our problem of evolving real-valued weights (Imada et al., 1997g).

At the beginning of a run, a population of chromosomes are initialized. Chromosomes are real-valued vectors each of which represents a set of synaptic weights

of an individual network, i.e.,

$$(w_{11}, w_{12}, \cdots, w_{1N}, w_{21}, \cdots\cdots, w_{NN}).$$

In each generation, these chromosomes are modified through discrete crossover and BGA-mutation operation. We now look at this rather specific BGA-mutation. In mutating chromosomes, a gene x_i is chosen with probability p_m to be mutated as usual. The BGA mutation is to give a small perturbation $\Delta x_i \times \delta$ on a variable x_i, where Δx_i is a mutation range for the variable x_i, and δ is calculated as

$$\delta = \alpha_0 + \frac{1}{2}\alpha_1 + \frac{1}{2^2}\alpha_2 + \cdots + \frac{1}{2^{15}}\alpha_{15}.$$

Each α_k takes the value 1 with probability 1/16, and otherwise takes the value 0. Then mutated new allele is

$$x_i \pm \Delta_i \times \delta,$$

with the sign $+/-$ being chosen with equal probability.

3.1.6 Diploid Chromosomes

So far, we have described representations in which N^2 synaptic weights of a network are represented as a string of real-values of a single vector. On the other hand, there is an alternative way to represent a configuration of weights using two one-dimensional arrays as a pair of chromosomes, which are called *diploid* chromosomes in the field of GAs. We might call the former one a *haploid* chromosome to distinguish it from the latter one. The GAs using diploid chromosomes are said to be more biologically realistic than those using haploid chromosomes. At the same time, the associative memory taking advantage of dynamical behaviors of fully connected neurons might be regarded as a model of human memory. This is one of the reasons of our interest in exploiting diploid chromosomes to evolve the associative memory network (Imada et al., 1997i; 1998c; 1998e), though this is too simple to be compared to our memory mechanisms. Here, w_{ij} and w_{ji}, which occupy two symmetric positions in the weight matrix, are mapped onto the same position in each pair of diploid chromosome. If two genes at the same position of a chromosome pair take a same value as a gene, the position is said to be

homozygous, and *heterozygous* otherwise. Hillis (1991) used diploid chromosomes in his GA to optimize a sorting network. He expected an evolutionary pressure to decrease the number of heterozygous positions in the chromosome pairs, which would minimize the number of comparisons of sorting entries. He showed that the number of heterozygous sites decreases as generation proceeds. In our problem, homozygous site means that the two symmetric elements in the weight matrix, w_{ij} and w_{ji}, are identical. Our previous results show that when random synaptic weights, which are entirely asymmetric at the beginning, become somehow symmetric after evolving to be able to store some patterns (Imada et al., 1997a). Hence, we expect a more efficient evolution from an entirely asymmetric weight configuration to somehow a symmetric weight configuration by exploiting diploid chromosomes than exploiting usual haploid chromosomes.

3.2 Results and Discussion

In the previous section, we have described various implementations for real-valued weights of a network to evolve to store a set of given patterns as associative memory. In this section, we show some results of each of these implementations.

3.2.1 Chaotic Trajectories to Fixed Point Attractors

Associative memory is a dynamical system in which an initial state changes its state along a trajectory. In this subsection, we observe how an evolution affects the trajectories. As mentioned earlier in this chapter, the dynamics of neurons' state can be visualized by plotting over time the Hamming distances between the initial state given to the network and instantaneous network states afterwards.

In Figure 2, we show a series of snapshots of the trajectories during an evolution (Imada et al., 1997h). Here, a GA evolves random weights of a network with 49 neurons under the goal of storing seven pre-determined random bipolar patterns. Since we start with a random configuration of weights in this example, all the trajectories in an early stage of the evolution are chaotic like in Figure 2a. Note that our fitness evaluation relates with total sum of areas under these trajectories, and the task is to minimize these areas. As evolution proceeds, we

can see some of the trajectories converge to stable states as in Figure 2b, though the states do not usually coincide with their corresponding inputs (spin-glass attractors). Then the number of trajectories that converge to stable states increases (Figure 2c), and finally all the trajectories converge (Figure 2d). In the mean time, we occasionally observe limit cycles like in Figure 2e. Gradually, these stable states approach their corresponding inputs (Figure 2f and g), and eventually all the given patterns are stored as fixed point attractors, or equivalently, the GA found an appropriate configuration of weights that creates fixed point attractors exactly at the locations of patterns to be memorized (Figure 2h).

3.2.2 Experimental Setup

When we search for an appropriate configuration of weights of a network that can work as an associative memory, it is known that there exist multiple solutions of these configurations of weights unless the number of patterns to be stored exceeds the capacity, that is, twice as much as the number of neurons. However, the task to search for one of these appropriate configurations of weights becomes hard as the number of patterns increases. Here we mainly focus on the effect on evolution of varying p, the number of given patterns. We repeat each simulation 30 times with different random number seed. If we find the perfect solution(s) then we increment p.

All the experiments here were carried out on networks with 49 neurons. So, note that the upper limit of storage is 98 patterns and the capacity of the Hebbian learning is 8 at most.

3.2.3 Perturbation by Mutation

As the first example, we evolve random continuous weights that are encoded directly into genes and mutated by being replaced with random Gaussian variables (Imada et al., 1997a). We use the algorithm described in Subsection 3.1.1, i.e., the GA to evolve real random weights using *perturbation by mutation*.

In Figure 3, we show a typical result of the best fitness versus generation. We can see that the GA have found an appropriate configuration of weights at generation 2,249 starting with totally random weights. We were able to observe the similar convergent phenomena unless the number of patterns to be stored

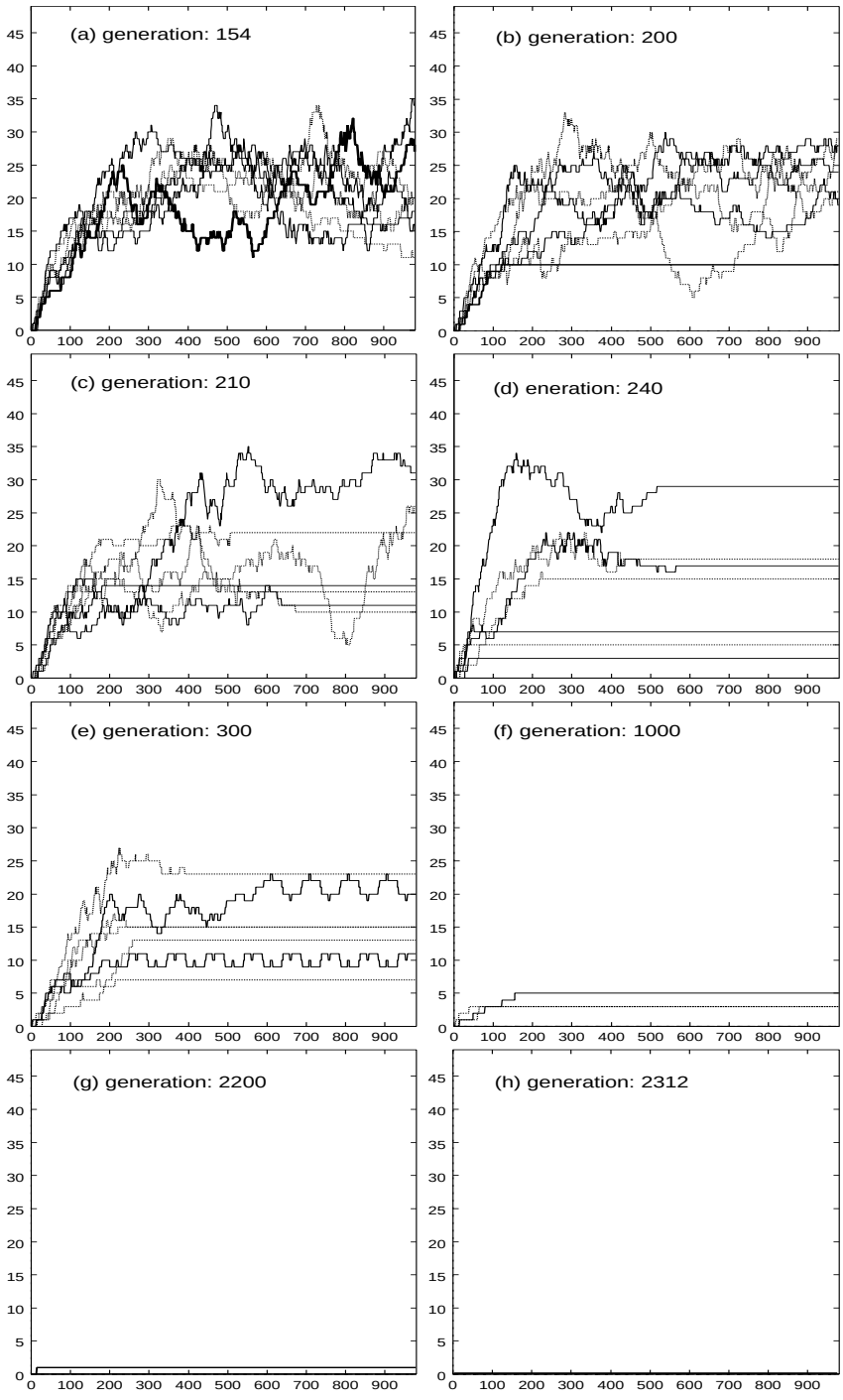

Figure 2. Trajectories resulted from 7 inputs. (X-axis represents updating time and Y-axis represents Hamming distance from the initial state.)

exceeds nine. Here we should note that the storage of nine patterns is slightly larger than the storage capacity of the Hebbian learning (at most eight).

In Figure 3, we also show a result of the same algorithm except that we do not use crossover operation. We see no convergence in this case. Hence we conjecture that the crossover operation in this implementation plays a significant role. As will be described later, this is contrary to the evolution by EP in which crossover is not used.

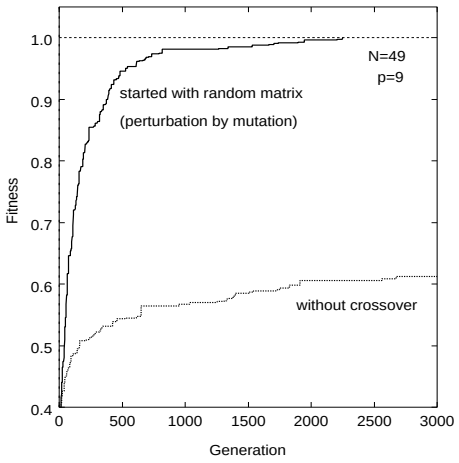

Figure 3. An evolutions of real weights using perturbation by mutation.

3.2.4 Perturbation by Chromosomes

The second example is an evolution using perturbation by chromosomes described in Subsection 3.1.2. In this implementation, we found that the random initialization of population like in the previous subsection did not work in finding the optima. So, we evolve an over-loaded Hebbian weight configuration.

The network of this type was analytically studied by Amit et al. (1985b). When the number of patterns p approaches to the capacity, some of the memorized states are shifted slightly. As p increases further, some inputs of these patterns relax to the other attractors, and finally, the network will be dominated by the vast amount of spin-glass attractors.

Here, we succeeded in evolving the Hebbian weights that had learned a maximum of 18 patterns eventually to store these 18 patterns as fixed points. Hence, we might say that the algorithm enlarges the the Hebbian capacity[25] more than double by giving perturbations on the weight values. The best fitness versus generation of the evolution is shown in Figure 4. We see that the fitness value started at around 0.95 and reached 1 at generation 3,760.

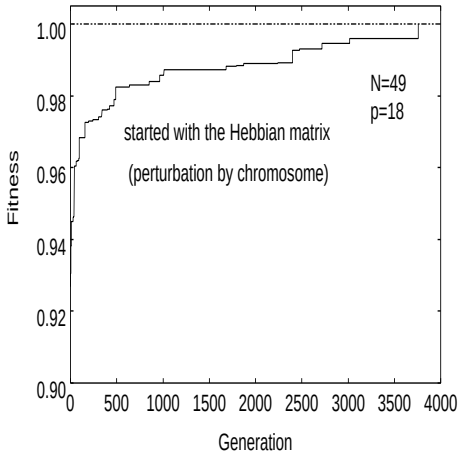

Figure 4. An evolutions of real weights using perturbation by chromosomes.

3.2.5 Evolution Strategy

The initial population of all identical zero weights and all identical Hebbian weights are employed. As in the previous section, random initialization did not work in this implementation either. The effect of varying the number of patterns to be stored on the evolution is studied, and we found a weight configuration evolved to store a maximum of 11 patterns started with a population of *all zero weights*[26] , and a maximum of 17 patterns started with over-loaded Hebbian weights. Representative samples of the best fitness versus generation for both of the evolutions are shown in Figure 5. The fitness value reaches 1 at generation

[25] At most eight patterns when the number of neurons is 49.

[26] This is sometimes referred to as learning from *tabula rasa*.

1,062 when started with all zero weights, and at generation 4,374 when started with the Hebbian weights.

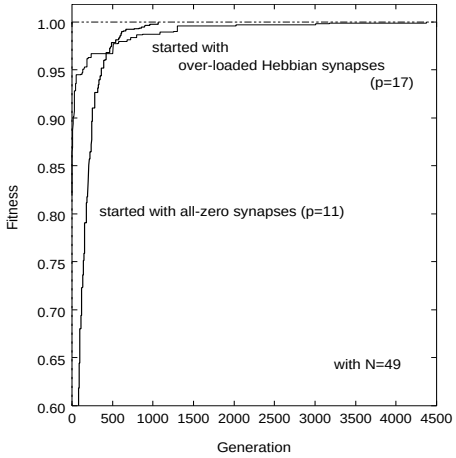

Figure 5. Fitness of the best of generation: ES.

3.2.6 Evolutionary Programming

In this EP implementation, the population is also initialized in two different way: all zero weights and over-loaded Hebbian weights.

In Figure 6, typical results for the above two different initializations are shown. We can see the convergence to the perfect solution at generation 7,848 starting with all zero weights, and at generation 10,858 starting with over-loaded Hebbian weights. The weights emerged stores a maximum of 6 patterns and 13 patterns, respectively.

In usual implementation of GAs or ESs, crossover operation plays an important role to search for the optimum. We observed that without mutation, neither GA nor ES reaches the optimum solutions in almost all runs. However, as we have shown in the experiment of EP here, we can reach the optimum solution without crossover, though not so effective.

Later in Chapter 8, we will argue a more effective usage of EP to learn how the local/global optima are distributed in the search space.

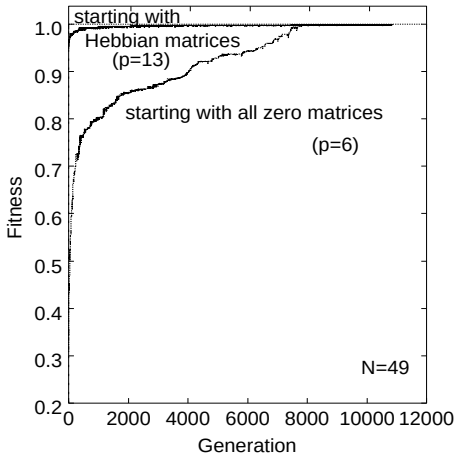

Figure 6. Fitness of the best of generation: EP.

3.2.7 Breeder Genetic Algorithm

In this implementation, we started searches with three different initialization: random, over-loaded Hebbian, and all zero weights. We show the details bellow.

Departure from Random Weights

In the first experiment, individuals are initialized at random. The effect of varying p, the number of patterns to be stored, on the evolution is studied. As a result, we were able to emerge the weight configuration that creates fixed point attractors exactly at the location of given patterns up to $p = 9$. Note that this limit of the storage is slightly higher than the Hebbian capacity ($p \approx 8$). In Figure 7, we show the best fitness versus generation curve when nine patterns are given.

Departure from Over-loaded Hebbian Weights

Then we proceed to the next experiment. We initialize all the individuals with over-loaded Hebbian weights.

As shown in Figure 8, we successfully evolved these weights to re-store all the

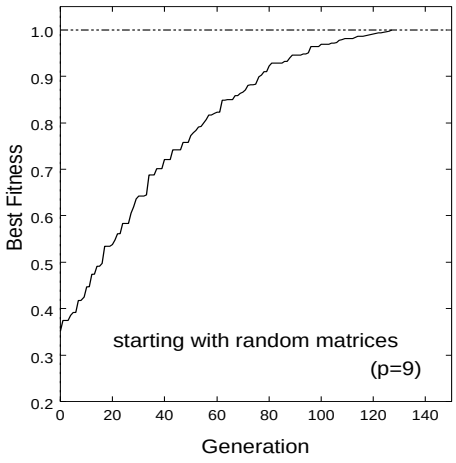

Figure 7. Best fitness vs generation during an evolution by BGA.

given patterns as fixed point attractors. We obtained a maximum storage of 25 patterns.

Departure from Zero Weights

Then the above enhancement of the storage from 9 to 25 patterns is due to the Hebbian learning? To answer this question, we tested another no-Hebbian initialization — all zero weights;

and we succeeded in evolving the weights to store all the 25 patterns as fixed points. Therefore, we might conclude the above enhancement in storage capacity is not due to the Hebbian learning but due to the search from one specific point rather than from randomly distributed points.[27]

The next question is which of the two different starting points, an over-loaded Hebbian weight configuration and all zero weight configuration, performs better? In the above experiments, runs with each of the two starting points were repeated 30 times with different random number seed. As a result, we found that the BGA

[27] The success, however, depends on the point to be started. We have not been able to observe the solution started with all identical configurations of arbitrary determined random weights for 25 patterns.

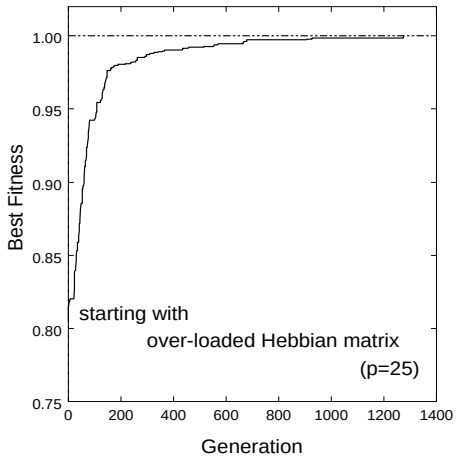

Figure 8. Best fitness vs generation: BGA started with over-loaded Hebbian weights.

starting with the over-loaded Hebbian weights located one of the solutions in 7 out of 30 runs, while the BGA starting with zero weights located it in 25 out of 30 runs. This suggests that there are more solutions around all zero weight configuration than around the over-loaded Hebbian weight configuration.

We further increment p with this all zero weight initialization. Thus far, we have observed that the all zero weights can be evolved eventually to store up to 90 patterns as fixed points – surprisingly large number of storage. We show the result in Figure 9.

We must note here that we can trivially realize a large storage capacity with identity matrix or with a matrix whose diagonal elements take large values compared with off-diagonal elements, where we can expect no or little error correcting capability in recalling storage pattern. However, diagonal elements of the weight matrices emerged here were of the same order of magnitude as the off-diagonal elements.

Identical versus Random Initialization

Usually in Genetic Algorithms, individuals are initialized randomly as our first ex-

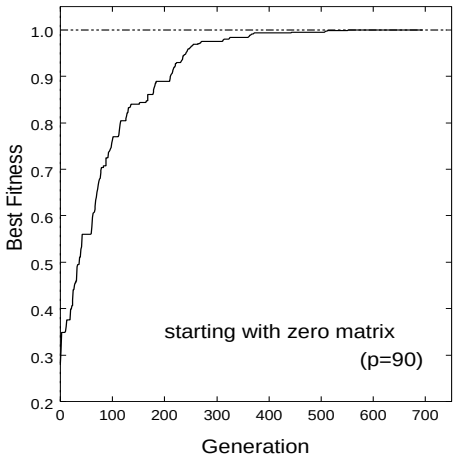

Figure 9. Best fitness vs generation: BGA started with all zero weights.

periment, so that starting points are distributed randomly over the search space. And, as Mühlenbein et al. (1996) stated, convergence implies that individuals become all identical.

In the latter two of our experiments, however, all individuals start the search at a specific one point in the space, and we found that this initialization works much more effective than the initialization of randomly distributed points, as mentioned already. When all individuals in the population are identical, the discrete crossover does not work. However, the BGA mutation increases diversity of the population quickly, which enables the evolution. In Figure 10, the standard deviation of the fitness values is plotted against generation number for the evolution shown in Figure 8. The fitness diversity grows rapidly in early stages. Then, as can be seen in Figure 11, enough diversity of population, say at generation 50, is obtained, and finally the most of the population converge to a solution.

Does Search Become Difficult as p Increases?

In our problem of searching for an appropriate weight configurations, it usually becomes difficult to locate a solution as the number of patterns p increases. Let's take a look at an example of how p affects the number of successful trial. In

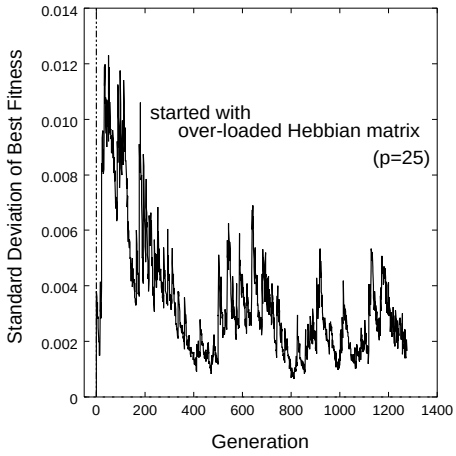

Figure 10. Time development of standard deviation of fitness values: BGA started with over-loaded Hebbian weights.

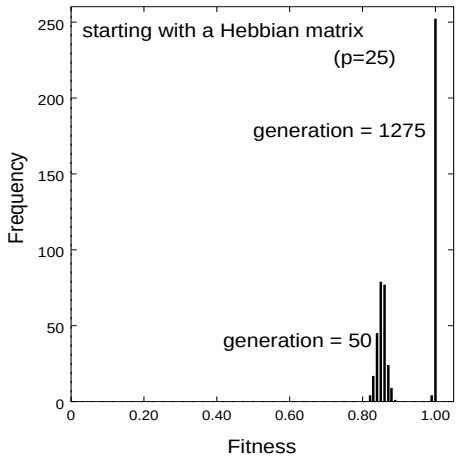

Figure 11. Fitness distribution: BGA started with all zero weights.

Table 1, results when a GA with perturbation by chromosomes is applied to the over-loaded Hebbian weights (Imada et al., 1997f) are shown.

Table 1. Number of patterns to be stored vs successful run out of 30 trials: a GA with perturbation by chromosomes starting with an over-loaded Hebbian weight configuration.

Number of patterns	14	15	16	17	18
Successful trial	22	17	5	2	0

In the BGA starting with zero weights, however, shows somewhat of a peculiar behavior. It seems to locate a solution more easily for larger number of given patterns. We have observed above that the BGA finds a solution in 27 out of 30 trials when $p = 25$, while it locates a solution in all of 30 trials when $p = 49$. For 90 patterns, although we have tried only 5 trials, we succeeded to find the optimum in all of the 5 trials.

Can the BGA Locate All the Optima?

It is difficult to locate all the optima in multi-modal fitness landscape, while it is comparatively easy to locate one of the optima. We are running the BGA starting with zero weights for a given set of 49 patterns which remains fixed over the runs. We observed the spatial distribution of the solution by measuring the Euclidean distance of each solution from the origin. So far, we have tried 68 runs and observed that 67 runs locate the optimum solution. Only two out of the 67 runs have located one exactly the same solution, and others have located different solutions. The Euclidean distances of the optimum from the origin were ranging from 4.556 to 16.297 depending on the random number seed at the start. This result suggests that there exit many different solutions within this domain of hyper-sphere. Hence, we might conclude that the BGA locates these multi-modal solutions

3.2.8 Evolution with Diploid Chromosomes

We have experimented three different versions of the GA that employs diploid chromosomes. The first one is the evolution in which an *over-loaded Hebbian weight configuration* is diluted using information in the diploid chromosome (Imada et al., 1997i, see also the next chapter). The second one is the evolution of *real-valued random synaptic weights* which are encoded directly into the diploid chromosomes (Imada et al., 1998e). The third one is the evolution of *random synaptic weights* which are clipped into two values, 1 and -1[28] with other schemes being same as the second experiment (Imada et al., 1998c). Here, we focus on the third experiment and we see that the diploidy works more efficiently in the third case than the first two cases.

We started the GA with a population of networks with random clipped weights (± 1), and the effect of varying p, the number of given patterns, on evolution is studied. Thus far, we have found a weight configuration evolved to store a maximum of five patterns. We show the best fitness versus generation of the evolution for $p = 5$ in Figure 12.

This evolution starts with totally asymmetric weights ($\approx 0\%$ symmetry), and observed that they evolved to be considerably symmetric ($\approx 60\%$). The degree of symmetry achieved was much higher than the one when we employed haploid chromosomes ($\approx 30\%$). Note that if we employ the Hebbian algorithm alone (without evolution) to prescribe weights of a network with the same size (49 neurons), the network will store around eight patterns, or six patterns if the weights are clipped to ± 1. In both cases, the weight matrix is totally *symmetric*.

Although the phenomenon of achieving the high degree of symmetry of the weights was also observed in the experiment of evolving *real-valued random* weights, the maximum storage obtained was only two patterns (Imada et al., 1998e). The GA applied to the over-loaded Hebbian weights, on the other hand, showed better performance (stored a maximum of 13 patterns) by reducing the degree of symmetry (Imada et al., 1997i). In that case, however, we did not observe any significant difference between diploid and haploid chromosomes.

[28] Sometimes weight values are discretized into several integers for the purpose of hardware implementations. Here, we evolve a random configuration of weights whose values are clipped into 1 or –1: the extreme cases of the discretization.

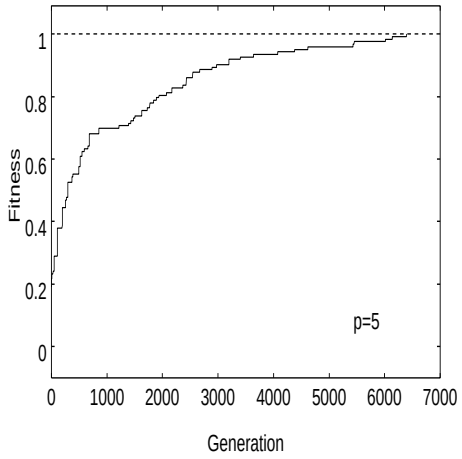

Figure 12. The best fitness vs generation: a GA using diploid chromosomes started with random clipped weights.

The Hopfield associative memory has an upper bound in capacity to store patterns, i.e., a network stores a maximum of only $2N$ patterns, as Gardner (1988) discussed. However, it is not so difficult to extend the model to the one with a higher (λ-th) order synapses, where storage capacity scales N^{λ} (Baldi and Venkatesh, 1987). We think we can employ *multi-ploidy* chromosomes also to express the network with this higher order synapses.

3.3 Summary

In this chapter, we have described how a population of real-valued weight configurations of a network evolves to give the network a function of associative memory starting with random, all-zero, or over-loaded Hebbian weights. We have tested a wide variety of evolutionary algorithms, and have found that six variants: GA with perturbation by mutation, GA with perturbation by chromosomes, EP, ES, BGA, and GA using diploid chromosomes. All of these algorithms succeeded more or less in emerging a weight configuration which stores a number of a given set of random bipolar patterns as fixed point attractors. For each of these variants, we repeated the experiments with the number of patterns to be stored increasing from one to the number at which the variant cannot search for the solutions any

more. As results, we found that the maximum storage obtained varies from one method to another. It is interesting to note that the BGA, among others, found solutions that store patterns whose number is almost the theoretical upper bound of the storage capacity in the sense of Gardner (1988), i.e., twice the number of neurons.

4. OTHER REPRESENTATION: PRUNING SYNAPSES

4.1 Pruning Some of the Hebbian or Random Connections

Since Hopfield proposed the neural network model of associative memory, many researchers, mostly physicists, have investigated how a slight modification of the Hebbian synaptic weights affects dynamical behaviors of the model. Here, we put our interest on the observations by Derrida, Gardner, and Zippelius (1987) among others.

They pruned a certain fraction of the connections without seriously affecting the storage of patterns. In other words, Derrida et al. showed plasticity of the synaptic connections. We made similar experiments with the difference being the following two points. First, we also prescribe the weights by the Hebb's rule, but the number of patterns to be stored exceeds the storage capacity. Hence, the capability of the network to store patterns collapses more or less at the beginning. Second, we selected the connections to be pruned *adaptively*, while Derrida et al. selected them *randomly*. We pruned connections using a GA, expecting that the GA eventually finds an optimal combination of connections to be pruned (Imada et al., 1995a; 1995b; 1997b).

4.2 Experiments

In the GA that prunes synaptic weights of the Hopfield network, a random weight configuration R_{ij} or over-loaded Hebbian weight configuration J_{ij} is predetermined as an ancestor, which remains fixed during evolution, as in the subsection 3.1.2. These weights are modified by a set of μ chromosomes, each of which is comprised of N^2 genes, c_{ij}, whose values are chosen from the allele $\{1, 0, -1\}$. This chromosome, we might call it *ternary chromosomes*, has some similarities with, but also important differences from those used in the so-called *canonical GA* in which *binary chromosomes* is widely used.[29] What is essentially different

[29] Higher cardinality alleles are sometimes said to be preferable, like nature uses *quaternary* DNA.

here is occurrence probabilities of each allele value. The probabilities with which 1, 0, and –1 are chosen as genes in the initial chromosome are set to $p_{c=1}$, $p_{c=0}$, and $p_{c=-1}$, respectively, such that

$$p_{c=1} \gg p_{c=0} = p_{c=-1}.$$

The ancestral weight configuration is modified as:

$$w_{ij} = c_{ij} \times R_{ij}, \quad c_{ij} \in \{1, 0, -1\},$$

or

$$w_{ij} = c_{ij} \times J_{ij}, \quad c_{ij} \in \{1, 0, -1\}.$$

This process yields a population of μ weight matrices. Since we choose mostly 1 as the value of c_{ij}, these matrices are only slightly different from the ancestor.

The scheme of selection and recombination we used here are essentially similar to those described in Section 3.1. Namely, we use truncation selection in which two parent chromosomes are chosen *randomly* from the best $T\%$ of the population, and uniform crossover which is a discrete version of discrete crossover used in that section. Mutation is made by rotating the allele as follows.

$$\{1\} \rightarrow \{-1\}, \quad \{-1\} \rightarrow \{0\}, \quad \{0\} \rightarrow \{1\},$$

which is different from either of standard binary GA or real-encoded GA mentioned in the previous chapter.

Here, the GA parameters are chosen as follows on the basis of trial and error. The population number is 256. This is simply because of our computer resources. The mutation probability p_m is set to 0.01. In selecting two parents, T is chosen to be 40%. Searching procedure is iterated until 12,000 generations unless perfect solution is not found. The initial probability of generating each allele $p_{c=1}$, $p_{c=0}$, and $p_{c=-1}$ are set to 68/70, 1/70, and 1/70, respectively.

As a result, and to our surprise, we found that the GA, by adaptively pruning some of the synapses, not only recovers the collapsed memory of the over-loaded Hebbian weights, but also evolves the random Gaussian synaptic weights eventually to store a number of patterns. We show some of the results in the following section.

4.3 Results and Discussion

4.3.1 Start with Over-loaded Hebbian Synapses

In this section, we start the GA with a Hebbian weight configuration which learned a set of patterns exceeding the capacity of the Hebbian learning rule (over-loaded).

The networks of this type were analytically investigated by Amit et al. (1985a; 1985b). As the number of patterns increases, the correlation of these patterns cannot be neglected. When the number exceeds the capacity, some fixed point attractors are gradually shifted to their neighborhood, and some other attractors around memory states are merged into spin-glass attractors (attractors located far from the memorized state and whose temporally averaged overlap is much lower than 1 (see (Amit, 1989)). To see these phenomena, let us look at an example where 16 patterns are stored by the Hebb's rule in a network with 49 neurons. The storage capacity in this case is around 8 patterns. Each input of the memorized patterns to the network results in the convergence to a stable attractor (rather than chaotic trajectory to be mentioned in the next section) due to the symmetry of the weight matrix. However, it is unlikely that the stable attractor is a memorized state. We observed in this example that the Hamming distances of these 16 final stable states from their starting states (input) were, 0, 0, 0, 0, 1, 2, 3, 3, 5, 6, 9, 11, 16, 17, 21, and 23. Specifically, four are fixed point attractors and the rest are distributed from near fixed point attractors to spin-glass attractors. We use this weight matrix as the ancestral matrix for our Genetic Algorithm. The best fitness value in each generation is shown in Figure 13. Surprisingly, we obtained a matrix which stores all these 16 patterns perfectly as fixed points (at generation 10,795). The representative samples for $p = 15$ and $p = 17$ are also shown in the figure. While we obtained the perfect solution for both $p = 15$ and $p = 16$, we were not able to observe the success for $p = 17$, among 30 runs with a different random number seed. The example shown in Figure 13 for $p = 17$ stored 14 out of 17 learned patterns as fixed points when the GA terminated.

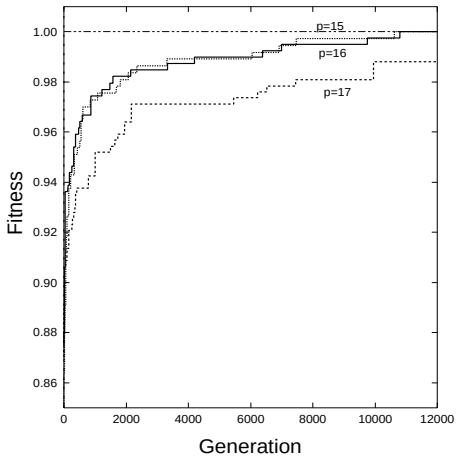

Figure 13. The best fitness of each generation: starting with over-loaded Hebbian weights ($p = 15, 16$, and 17).

4.3.2 Start with Random Synapses

If we further increase p, the number of patterns, the Hebbian synaptic weights J_{ij} approach the Gaussian random variables with the exception of symmetry $J_{ij} = J_{ji}$ (Amit et al., 1985b). This is equivalent to the Sherrington-Kirkpatrick model of Ising spin-glasses (SK-model) (Kirkpatrick and Sherrington, 1978), and includes a full of spin-glass attractors.

Such a network with *symmetric* random synaptic weights was investigated by iterated modification of weight values; for example, Toulouse et al. (1986), Nadal et al. (1986) and Sompolinsky (1986) modified random Gaussian synaptic weights, denoted here as R_{ij}, by adding them a small perturbations η_{ij} which were determined using the Hebbian rule[30], that is, each weights modified as:

$$w_{ij} = R_{ij} + \eta_{ij}.$$

These methods strengthened the low peaks existed near patterns to be stored, and eliminated many other attractors.

[30] Note that their experiments are more or less close to our simulations of *Lamarckian evolution* of random synaptic weights which will be described later.

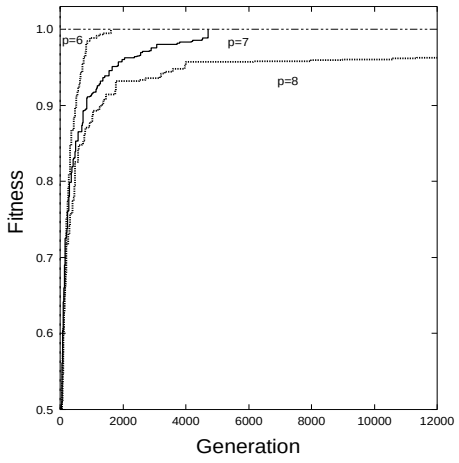

Figure 14. The best fitness of each generation: starting with random weights ($p = 6, 7$, and 8).

Here, on the other hand, we do not use *any* learning algorithm but only prune some of the connections, namely,

$$w_{ij} = r_{ij} \times c_{ij}.$$

Moreover, the synaptic weights to start with are totally asymmetric. Any input pattern to such a network results in a chaotic trajectory such as those shown in Figure 2.

We run the GA with the ancestor being a random weight matrix, expecting eventually to store a set of pre-determined patterns as fixed points. Again to our surprise, the successful network emerged which stored a maximum of 7 patterns perfectly as fixed points. The best fitness versus generation for $p = 6$, 7, and 8 are shown in Figure 14. We can see that the GA succeeded in finding a solution for $p = 7$ (at generation 4,719). We can also see that a solution was found more easily for $p = 6$, while no solution was found for $p = 8$.

4.4 Summary

In this chapter, we have presented quite a different scheme from those described in the previous chapter. A pre-fixed synaptic weights are adaptively pruned

by a population of chromosomes that are to be evolved by a GA. Results were amazing. Even a random weight configuration eventually stores a number of given patterns as fixed point attractors only by being pruned some of the synapses. The maximum storage obtained in this method was a half of the storage capacity of the Hebbian weight configuration. On the other hand, by starting with an overloaded Hebbian matrix that had unsuccessfully learned a set of patterns twice as much as the capacity, we obtained weights that store the patterns perfectly as fixed points. Namely, we double the Hebbian storage capacity.

5. LEARNING

In our experiments described so far, individuals have not used any learning algorithm during their lifetime. The only case we used a learning algorithm is when evolution starts with an over-loaded Hebbian weight configuration, where the primordial individuals *a priori* learn patterns with the Hebbian learning algorithms. What then if individuals learn patterns during their lifetime? In this chapter, we address this issue. To be more specific, two analogies from biological concept as for relationship between learning and evolution: *Baldwin effect* and *Lamarckian inheritance.*

In Baldwin evolution, learning is used to change the fitness surface, but the solution that is found is not encoded back into the genes, while in Lamarckian learning, genes are updated to match the solution found by the learning procedures.

Note that the learning, in the context of evolutionary algorithms, can be said to be good at finding local optima in the region in which the algorithm converges.

5.1 Baldwin Effect

The effect of life-time-learning on evolutions was first studied as a biological process by Baldwin (1886). Now this is known as the Baldwin effect. As an analog of this effect, many researches have addressed the relationship between *learning on a population level through evolution* and *on an individual level during its lifetime.* Especially, researches of combining GAs and neural networks have had great interests on this issue. Gruau et al. (1993), for instance, showed an enhancement of performances of their neural networks by introducing the Baldwin effect on the evolution (see also (Hinton et al., 1987) and (Parisi et al., 1995)).

We also incorporated the Baldwin effect to our evolution of neural network model of associative memory (Imada et al., 1997c), using the GA implementation described in Section 4.1. The implementation enables us to test this Baldwin effect on the evolution. Starting with a random weight configuration, each individual in the population undergoes evolution in the same way as before, but they learn a set of pre-determined random patterns in their lifetime before its fitness evaluation.

At the beginning of a run, a weight configuration W_0 is produced randomly so that each weight w_{ij} is chosen from $\{-1, 1\}$. Hence, the weight configuration does not store any patterns. This weight configuration remains unchanged during evolution, but instead, weight configurations produced by a population of chromosomes are evolved. Chromosomes are generated randomly at the beginning of a run. Every chromosome has a fixed length of N^2 genes which are chosen from $\{-1, 0, 1\}$ with the probability $p_{c=-1}$, $p_{c=0}$, and $p_{c=1}$, respectively, where $p_{c=-1}$ and $p_{c=0}$ are much smaller than $p_{c=1}$. Each component of the original weight matrix w_{ij} is multiplied by one of these alleles, i.e.,

$$w_{ij}^{(n)} = w_{ij} \cdot c_{ij}^{(n)},$$

where $c_{ij}^{(n)}$ is the ij-th allele of the n-th chromosome and $w_{ij}^{(n)}$ is the ij-th component[31] of the n-th copy of the original matrix. Gene 0 implies to prune the corresponding connection, and -1 to reverse the excitatory/inhibitory connection. Thus chromosomes produce a population of the copies slightly different from W_0. Each individual phenotype learns a set of pre-determined random patterns ξ^μ by the Hebbian learning rule as follows.

$$w_{ij} = w_{ij} + \lambda \sum_{\mu=1}^{p} \xi_i^\mu \xi_{j\neq i}^\mu.$$

These matrices are used only for fitness evaluation, and the learning results do not affect chromosomes. This is the reason we can regard it as the Baldwin effect on the evolution. Two parent chromosomes are chosen uniformly at random from the best $T\%$ of the population, and are recombined to produce one offspring by uniform crossover. The offspring are occasionally mutated with probability p_m, where the mutation rotates the value of randomly chosen gene in a chromosome cyclically, as

$$(1) \to (-1), (-1) \to (0), (0) \to (1). \tag{8}$$

After the offspring are replaced with the worst $(100 - T)\%$ of the population the above procedures are iterated.

[31] To be more precise, by the ij-th component we mean the $(j + (i-1) \cdot N)$-th component.

5.2 Lamarckian Inheritance

In the experiment of the above Baldwin effect, the results of individuals' learning do not change their chromosomes, but only affect the selection after fitness evaluation. However it is reported that incorporation of learning results into chromosomes may also enhance the performance of GAs (see (Gruau et al., 1993) for example). This is known to be the Lamarckian inheritance. We also investigated the effect of the Lamarckian inheritance on the evolution of associative memory, though a different implementation had been required where the chromosomes should be comprised of components of the weight matrix (Imada et al., 1996c).

Starting with a random weight configuration, each individual learns a set of pre-determined random patterns in its lifetime from generation to generation in the similar way to the Baldwin evolution, but the learning result affects its chromosome here. At first, a population of weight matrices, W^ν, are produced so that each component of the matrices w_{ij}^ν are chosen randomly either from -1 or 1. Hence every matrix in the first population does not store any patterns at this moment. In each generation thereafter, each weight matrix in the population learns a set of pre-determined random patterns ξ^μ by the Hebbian learning rule as follows.

$$w_{ij} = w_{ij} + \lambda \sum_{\mu=1}^{p} \xi_i^\mu \xi_{j\neq i}^\mu.$$

These components of the Hebbian learned weight matrices are incorporated directly to their chromosomes. That is, each gene of the chromosome is encoded again as:

$$c^\nu(i + N \cdot j) = w_{ij}^\nu, \quad i, j = 0, 1, 2, \cdots, (N-1).$$

This allows us to call this evolution as Lamarckian. Then two parent chromosomes are chosen uniformly at random from the best $T\%$ of the population, and are recombined by uniform crossover to produce an offspring chromosome. Furthermore, mutation is operated in the same way as Equation (8) in the previous subsection. We must note that zero components of weight matrices are introduced only by this mutation operation.

5.3 Results and Discussion

5.3.1 Baldwin Effect

We use the Hebbian learning rule as a life-time-learning in the simulation of the Baldwin effect. We show some of the results in Figure 15. In this experiment, the value of λ when individuals learn the given patterns in their lifetime was set to 0.4. As a result, we were able to store a maximum of 19 patterns. In this evolution, the perfect solution was emerged at generation 5,408. In the figure, we also show the evolution for the number of patterns to be stored were 17 and 20.

Although the increase in capacity is not so drastic compared to the no-Baldwin evolution starting with the Hebbian matrix, the convergence speed is tremendously improved. For example, the perfect solution was obtained at generation 996 for 17 patterns, while the evolution without the Baldwin effect for 16 patterns required 10,795 generations to converge (see Chapter 4).

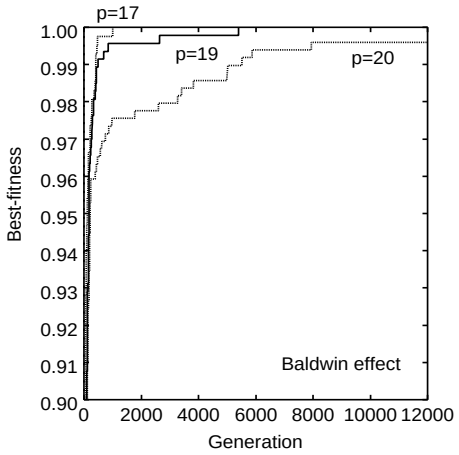

Figure 15. Baldwin evolution.

In Figure 16, we show the best-fitness versus generation of the three experiments for comparison purpose. One is the Baldwin evolution (the same one as Figure 15) the other two are no-Baldwin evolutions with two different initializations. The two no-Baldwin GAs used in this experiment were similar to the Baldwin version except for the Hebbian learning process before fitness evaluation.

The goal of the GAs is to store a set of 19 patterns as fixed points. We can see that the evolution under the Baldwin effect outperforms the other two. As mentioned above, a matrix which stores all these 19 patterns as fixed points emerged under the Baldwin effect, while the other two evolutions were never converged. The no-Baldwin GA, when started with over-loaded Hebbian matrix, only evolved to store 10 out of 19 learned patterns, and starting with a random matrix resulted in only one pattern of storage as fixed point.

We conjecture that the evolution pressures in these three GAs are different from each other. In the Baldwin version, the pressure works to enhance learnability rather than to directly increase a capacity. That is, individuals which have higher ability to learn tend to survive.

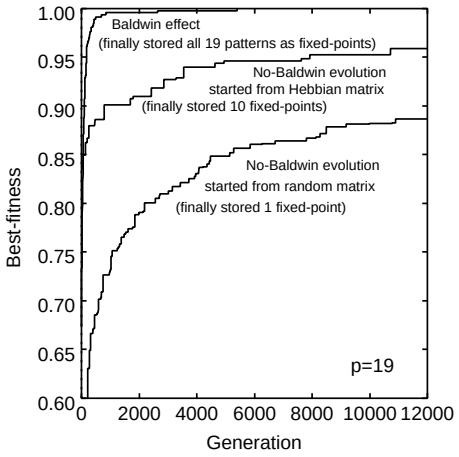

Figure 16. Baldwin vs no-Baldwin.

5.3.2 Lamarckian Inheritance

Here, we used the Hebbian learning rule as a life-time-learning with the learning ratio λ being 0.02, but unlike the Baldwin evolution, we encoded back the results of the learning into genes in reproducing an offspring.

As shown in Figure 17, the Lamarckian inheritance emerges a network which stores a maximum of 17 patterns while a similar experiment without the life-time-learning emerges only a network that stores a maximum of 9 patterns. The

perfect solution under Lamarckian inheritance was obtained at generation 2,147, while the no-learning version for 9 patterns converged at generation 6,931. Hence, we can conclude that the Lamarckian inheritance improves the convergence speed as well as the storage capacity.

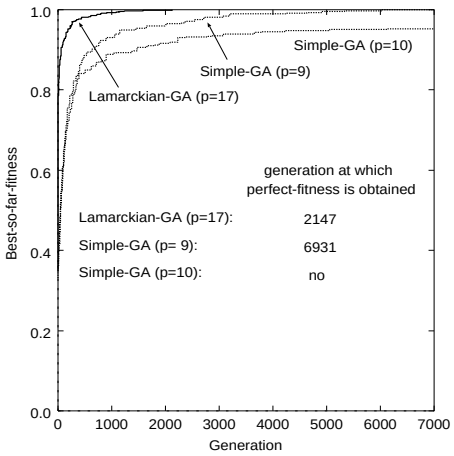

Figure 17. Fitness vs generation under Lamarckian inheritance.

5.4 Summary

Besides seven different evolutionary algorithms described in the previous two chapters, we have examined two more implementations in this chapter. Two hypotheses from evolutionary biology are incorporated into our artificial evolutions: the Baldwin effect and the Lamarckian inheritance. In other words, we have studied effects of life-time-learning on evolution. In the Baldwin evolution, the life-time-learning does not affect on genes but only on selection, while in the Lamarckian evolution it affects on inheritance of genes. Starting with random weights, each individuals in the population learns, generation by generation, a set of given patterns by the Hebbian learning algorithms, and we have found that the both hypotheses improved the performance of our evolutionary algorithms.

6. SYMMETRY AND DILUTION OF SYNAPTIC WEIGHT

Thus far, we have described various implementations of evolutionary algorithms that found a number of the optimal weight configurations for an associative memory network. Clearly, these solutions of weight configurations are different from one implementation to another. This chapter gives a consideration on what are the differences based on two parameters: *degree of symmetry* and *dilution ratio* of weight matrices.

6.1 Pruned-weight Case

In the late 1980's, researchers of the Hopfield model of associative memory extensively argued how pruning some of the synaptic weights affects performances in terms of the dilution ratio and degree of symmetry, and reported many interesting phenomena. So, we begin this chapter by arguing the dilution ratio and degree of symmetry of the weight matrices when they are evolved by the GA described in Section 4.1 that prunes some of the weight connections.

6.1.1 Effect of Asymmetry of Weight Matrices

Hertz et al. (1987) suggested in their analysis of the Hopfield model of associative memory that asymmetry of synaptic weights destabilizes spin-glass attractors. Parisi (1986) also suggested that asymmetry converts the spin-glass attractors into chaotic trajectories. Although these analyses were made on the condition that the number of memorized patterns was within the capacity, we conjecture that the success of evolving over-loaded Hebbian synaptic weights mentioned in Subsection 4.3.1 is also due to this reduction of spin-glass attractors.

Then, what about the success of evolving random synaptic weights mentioned in Section 4.3.2? They are totally asymmetric at the beginning. We suspect that some symmetric components introduced by the GA create attractors near the locations of given patterns.

To bear out these conjectures, we investigated the time evolution of degree of symmetry of weight matrices. Following after Krauth et al. (1988), we define the

degree of symmetry as

$$\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}w_{ji} \cdot \left(\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij}^2\right)^{-1}.$$

In Figure 18, we plot this degree of symmetry as a function of generation.

The Hebbian matrix, however over-loaded it may be, is totally symmetric. As evolution proceeds, however, the degree of symmetry decreases and gradually approaches a value around 0.7, and retrieval states emerge. Our guess is that the asymmetry destabilized the many spin-glass attractors which had existed at the beginning. On the other hand, when started with random synapses, totally asymmetric weight matrix gradually obtains some degree of symmetry, and when retrieval states emerge, the value is around 0.2. The result seems to support our belief that a few symmetric components of the weight matrix play an important role to create memory states. However, very little is known about how this symmetry influences a network full of chaotic attractors.

In any case, the role of asymmetry of the synaptic weights is still somewhat of a mystery.

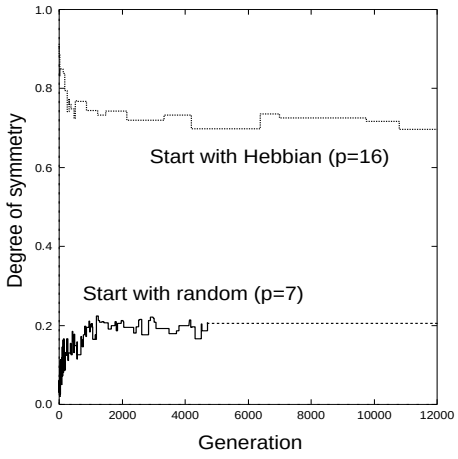

Figure 18. Evolution of degree of symmetry when weights are pruned by GA.

6.1.2 The Effect of Dilution of the Weight Matrix

To witness the effect of dilution of synaptic weights of our experiment, we investigated the dilution rate as a generation proceeds. The results for both experiments are shown in Figure 19. Here, we must note that when an even number of patterns are learned by the Hebbian rule, some synaptic weights might be equal to zero, whereas in the case of odd number patterns, the Hebbian rule does not allow any zero in the weight matrix. For that reason, we show results of both $p = 15$ and $p = 16$, in the case of departure from an over-loaded Hebbian matrix. We can see in the figure, the difference between 15 patterns and 16 patterns is only the initial difference. However, the behaviors resulted from two specific starting matrices are quite different. When an algorithm starts with Hebbian synapses, the zero density does not increase drastically. On the other hand, when the run starts with random synapses, it increases rapidly at the beginning, and approaches a value around 0.17. It is interesting here to note that the value was independent on the initial percentage of zero in chromosomes (i.e., $p_{c=0}$).

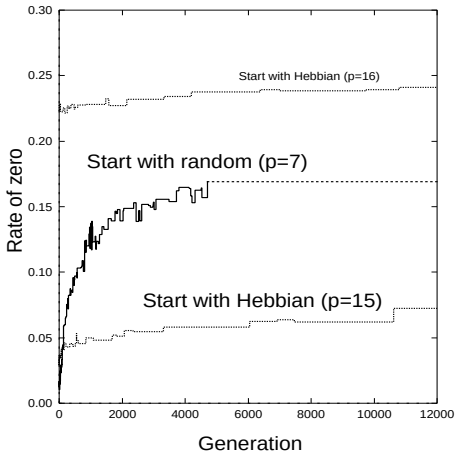

Figure 19. Evolution of zero density when weights are pruned by GA.

Though Derrida et al. (1987) showed that the dynamical equation of state transition for a randomly connected network is exactly solvable if connections are extremely sparse, it is much more difficult to analyze our experimental results because of the crossover operations in the GA. In addition, not only do we prune

some of the connections, but we also change a role of synapses from excitatory to inhibitory, or vise versa, using –1 in our chromosomes. We surmise that this reversal of excitatory/inhibitory synapses plays an important role in the evolution. In fact, we observed that without –1, the evolution stagnated to local optima very rapidly (Imada et al., 1997g). So far, though, we have not fully understood the reason for this behavior.

6.2 The Other Cases

Next, we compare the above-mentioned results with those obtained by the other implementations.

6.2.1 Evolution under Perturbation by Chromosome

Here, we look at the degree of symmetry when an over-loaded Hebbian weight configuration undergoes an evolution by adding random genes of a chromosome (perturbation by chromosomes). We show an example in Figure 20. As we can see, the behavior is almost similar to the result of the prune-synapses-GA (Figure 18).

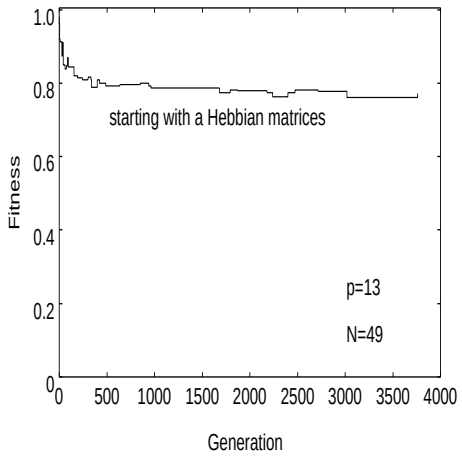

Figure 20. Degree of symmetry when over-loaded Hebbian weights are evolved under perturbation by chromosomes.

6.2.2 Evolution under Perturbation by Mutation

We now turn to the degree of symmetry during an evolution of directly mapped weights. Here the genes mapped from real random weights are perturbed by mutation. In contrast with the evolution of over-loaded Hebbian weights, the random weights are totally asymmetric at the beginning. An example of the results is shown in Figure 21. We can see that the degree increases from zero and asymptotically approaches the value around 0.25 as the spin-glass attractors are destabilized and retrieval states are emerged. The behavior is almost similar again to the evolution of real random weights by the GA that prunes synaptic weights described in Section 6.1.1.

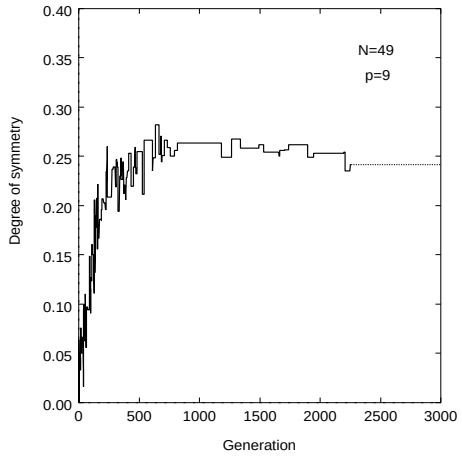

Figure 21. Degree of symmetry when random weights are evolved under perturbation by mutation.

6.2.3 Evolutionary Programming

Here, we employ an EP, and observe how the degree of symmetry evolves. Note that EP perturbs genes also by mutation as in the previous subsection, but the perturbations are adaptive in EP. We started with both over-loaded Hebbian weights and all-zero weights. Typical examples for both of the starting populations are shown in Figure 22. Alternating two different levels during certain

generations, the degree of symmetry approaches some value.

When started with an over-loaded Hebbian weight configuration, the degree decreases from the value one and approaches the value around 0.6 as a retrieval state emerges. This value is somewhat smaller than the one obtained in the above two implementations with the same initial population (over-loaded Hebbian matrices).

When started with all-zero weight configuration, the degree also fluctuates generation by generation around 0.1 searching for an appropriate level, and finally found the optimal one close to zero.[32]

This suggests that some small degree of symmetry is needed to obtain the function of associative memory. However, the role of asymmetry of the synaptic weights is still an open question.

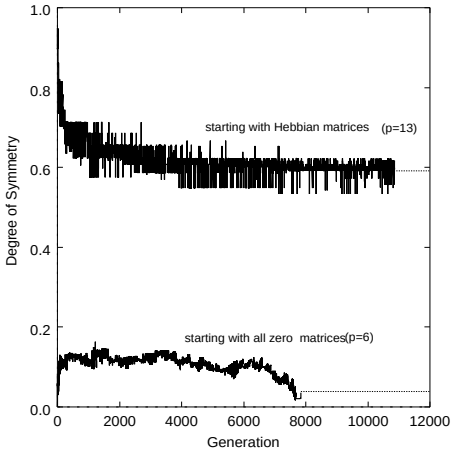

Figure 22. Degree of symmetry when real weights are evolved by EP.

6.2.4 Evolution Strategy

Then what happens when we employ an ES? ES employs crossover operation besides the perturbation by adaptive mutation. Here, we study the degree of

[32] All-zero component matrix is symmetric. However, immediately after a perturbation is given to each component, the matrix becomes totally asymmetric. Although we cannot see it in the Figure, the degree of symmetry is 1 at the generation zero and 0 at the generation one.

symmetry starting with both over-loaded Hebbian weights and all-zero weights. We show the results in Figure 23. We can see that when started with random weights, the degree of symmetry increases rapidly, and approaches to the value around 0.4.

When started with over-loaded Hebbian weight matrices, the degree of symmetry decreases and asymptotically approaches to the value around 0.8 and retrieval states emerge. Again we can ascertain our conjecture that the asymmetry destabilized the many spin-glass attractors and enable network states to approach the retrieval states.

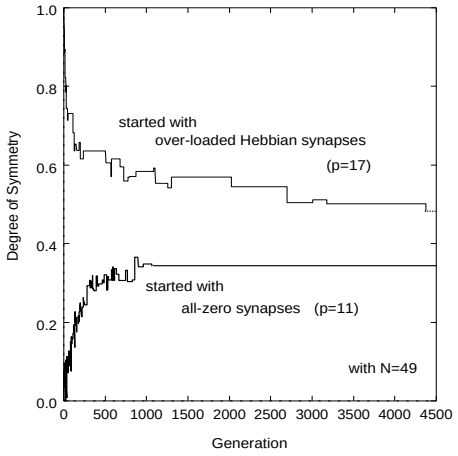

Figure 23. Degree of symmetry when an over-loaded Hebbian and real random weights are evolved by ES.

6.2.5 Breeder Genetic Algorithm

Next, we observe the degree of symmetry during an evolution by BGA. As stated in subsection 3.2.7, the BGA is extremely efficient in searching for solutions that give a network a large storage capacity. Starting with over-loaded Hebbian weights, the BGA found a solution which stores 25 patterns (with 49 neurons), whereas the so-far-maximum storage has been 19 patterns which is obtained by the Baldwin evolution (see Section 5.3). The difference is more tremendous when we start with all-zero weights. In this case, the BGA found solutions which stores

more than 90 patterns, while with other method we only obtained a maximum of 11 patterns which is by ES. Hence we expected the different behaviors in the evolution of the degree of symmetry.

In Figure 24, we show a result of both starting with the over-loaded Hebbian and all-zero weights. At the beginning of a run, both matrices are totally symmetric. The BGA started with the Hebbian matrix seems to hill-climb by introducing small amount of asymmetry to the Hebbian matrix as usual. The Hebbian matrix decreases its degree of symmetry as evolution proceeds, and the degree approaches the value around 0.6. This is not so significantly different from the other implementations mentioned so far. However, in the evolution started with zero matrix, the degree of symmetry behaves amazingly in different way. It decreases abruptly to zero and remains the value thereafter.

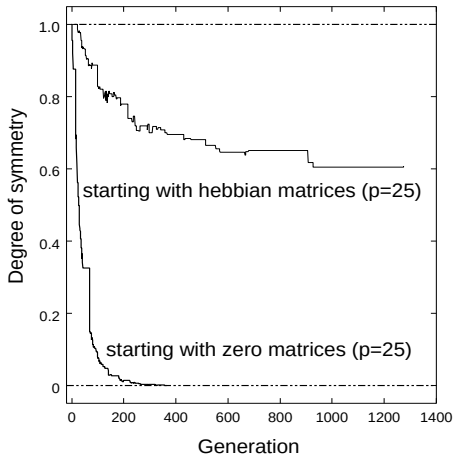

Figure 24. Degree of symmetry when weights are evolved by BGA.

6.2.6 Diploid Chromosomes

In this section, we see how different is the affect of using diploid chromosomes to the evolution from the one using haploid chromosomes. As Hertz (1987) suggested, asymmetry of the synaptic weights is relevant to the number of spurious attractors, and hence, the function of associative memory. And as we described

in Section 3.1.6, weight matrices undergo evolutionary pressure to be more symmetric under diploidy evolution than usual haploidy evolution. Hence we can expect that when started with random weights, which are totally asymmetric, the degree of symmetry will reach larger value than the value in the case of the same initial population but under haploidy evolution. The results are shown in Figure 25. If we compare the result with Figure 18 and Figure 21, we can observe a significant difference. The degree of symmetry when the fitness reaches 1 is around 0.6 with diploid chromosome, whilst $0.2 - 0.25$ with haploid chromosome. We conclude that the difference is due to the pressure to symmetry (homozygous genes in chromosome) under diploidy evolution.

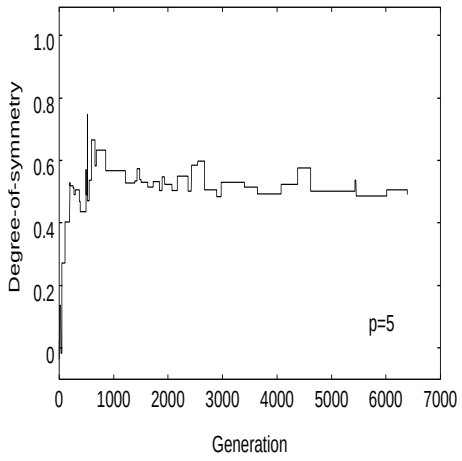

Figure 25. Time development of degree of symmetry under diploidy evolution.

6.2.7 Baldwin effect

In the last two subsections, we study how does the Baldwin effect and the Lamarckian inheritance affects on symmetry and dilution ratio of weight matrices during an evolution.

First, we evolve a random weight matrix using the Hebbian rule as a life-time-learning of each individual, that is, the Baldwin evolution. In the Baldwin evolution, as we can see in Figure 26 (left), the degree of symmetry remains unchanged

from the beginning (around 0.8). This is very different than behaviors of symmetry in other evolutions. It seems that the learning that affects individuals only in their lifetime plays some role for the phenomenon.

Figure 26 (right) shows the temporal development of zero density in chromosome of the best individual under Baldwin effect (solid line), together with the one from no-Baldwin version starting with an over-loaded Hebbian and all-zero weights (dashed lines). Here also we can see significant differences from other evolutions.

So, from both of these two figures, we surmise that this difference is due to the evolutionary pressure that works to enhance learnability rather than other capabilities such as storage capacity. In other words, individuals that have higher ability to learn tend to survive under this evolution.

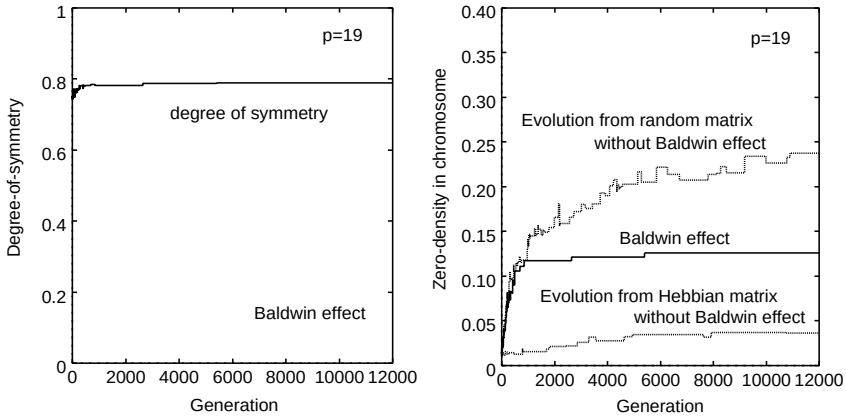

Figure 26. Degree of symmetry and rate of zero under the Baldwin evolution.

6.2.8 Lamarckian Inheritance

In the previous chapter, we saw how symmetry and rate of zero in weight matrices during an evolution under the Baldwin effect. In the Baldwin evolution, results of life-time-learning is never coded back into genes of individuals, and we had observed that the symmetry of weights did not change significantly during the evolution, which is very different than the results of no-Baldwin version.

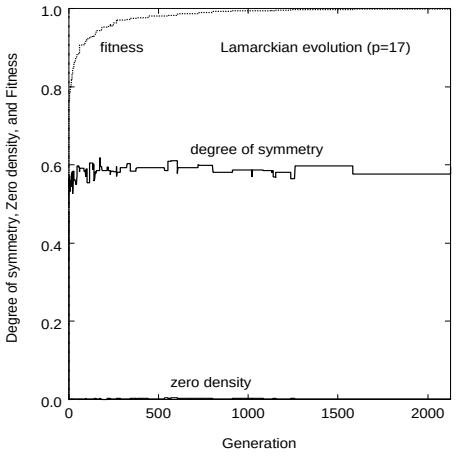

Figure 27. Degree of symmetry and zero density in an evolution with Lamarckian inheritance.

Then, how are the degree of symmetry and rate of zero under Lamarckian evolution? Since the Lamarckian evolution coded back the results of life-time-learning into genes of individuals in reproducing offspring, we expect some different results from those of the Baldwinian evolution. We now study the influences of the Lamarckian inheritance on symmetry and dilution ratio during an evolution. In order for the Lamarckian inheritance to be incorporated, we evolve here random weight matrices whose entries are clipped to either 1 or –1, and use the Hebbian rule as a life-time-learning of each individual (see Section 5.2 more in detail).

The results (of the same run as in Figure 17) are shown in Figure 27, together with the fitness curve for convenience. As can be seen in the figure, the zero density in the weight matrices remains almost zero, and the degree of symmetry keeps the value around 0.6. Both ratios do not change throughout the evolution, despite the fitness curve shows the typical improvement. These behaviors of both ratios are totally different from no-Lamarckian evolutions. However the difference between the Baldwin and Lamarckian evolution has not been so obvious.

6.3 Diversity of the Solutions

In this section, we have described behaviors of two parameters: degree of symmetry and rate of zero of weight matrix under various types of evolutions. What we want to emphasize here is a diversity of the results. To see it, in Figure 28 we summarize the diversity by picking up some of the typical examples again from Figure 18 to 27. Though the examples are not exhaustive, we can induce that solutions obtained are different from implementation to implementation.

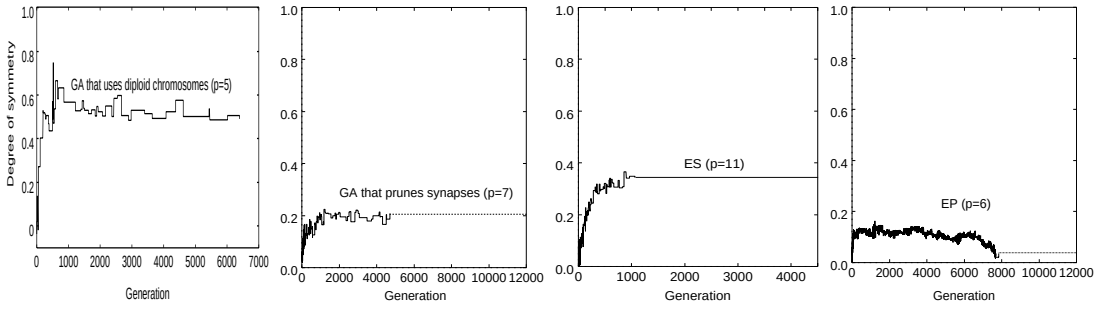

Figure 28. Diversity of the result of the degree of symmetry in various implementations.

6.4 Summary

The solutions obtained by a total of nine different evolutionary algorithms mentioned in the above three chapters are studied in terms of two parameters of the weight matrix: the degree of symmetry and the rate of zero. As results, we have found that we now have a wide variety of solutions in weight space. Our concern is the number and distribution of these solutions, and they will be argued in Chapters 8 and 9 bellow.

7. BASIN OF ATTRACTION

Thus far, we have found many weight configurations that give a network a function of associative memory. In these searches, the criteria were how many patterns can a network store as fixed points, i.e., storage capacity. However, as the number of patterns to be stored increases, the ability of the network to tolerate noisy or partial input decreases. In other words, in order for the network to store many patterns as fixed point attractors, the basin of attraction of the attractors should become small, i.e., a trade-off between storage capacity and size of basin of attraction. Personnaz et al. (1984) indicated that the basin of attraction of a stored memory falls sharply as the number of patterns approaches to a half of the number of neurons.

In this chapter we explore the trade-off between storage capacity and basin size with using a Genetic Algorithm. The Genetic Algorithm was based on pruning some of the synaptic weights adaptively (see Chapter 4). We evolve here the network which stores p patterns by the Hebbian learning rule where the size of the basin of attraction of the Hebbian attractors becomes smaller as p increases (Imada et al., 1996a; 1996d). We conjecture that the basin size of a peak can be optimized so that the radius of an attraction domain will be extended to a half of the distance from its nearest neighbor peak (in N-dimensional string space). The synaptic weights w_{ij} define an energy function on an N-dimensional bipolar string space, i.e.,

$$E(S) = -\frac{1}{2} \sum_{i,j(i \neq j)} w_{ij} S_i S_j,$$

where S moves over all possible combinations of N-dimensional vector $\{-1, 1\}^N$ and S_i is the i-th component of the state S. If the number of stored patterns is small enough, this function takes a local minimum where S corresponds to one of stored patterns. The stored patterns are said to be *attractors* and the size of this local minimum are referred to as *basin of attraction*. As the number of stored patterns p increases, spurious minima become more frequent and finally they become randomized. When input is one of the stored patterns that was *a priori* given small noise within the size of basin of attraction, this input pattern relaxes to the original stored pattern after several steps of the update. Our goal

here is to extend this basin of attraction.

7.1 To Enlarge the Basin Size: GA Implementation

Toward the goal of extending the basin of attraction, we redesign the GA to try to control the size of basin of attraction. The GA has some similarities with, but also important differences from the GA described in Chapter 4.

(1) A weight matrix W_0 is generated in certain way. Here, we use a weight matrix obtained from a version of our genetic algorithms (no noisy-input) in which random weights eventually evolve to store all the given patterns perfectly as fixed points. As we simulate with 49 neurons of Hopfield network, the size of W_0 is 49×49. This original weight matrix remains unchanged during the evolution. **(2)** Then 256 chromosomes are generated randomly. The chromosome has a fixed length of $2401 (= 49 \times 49)$ genes, and the value of the genes are chosen randomly from $\{-1, 0, 1\}$, where the probability of choosing either -1 or 0 is set to 0.02. As will be mentioned in procedure (3) below, each component of the original weight matrix w_{ij} is multiplied by one of these alleles. Allelic value of 0 corresponds to pruning the connection to which it is multiplied, and -1 to reverse excitatory/inhibitory connection. We denote the i-th gene of the n-th chromosome as $c^{(n)}(i)$. **(3)** Each chromosome modifies the original weight matrix W_0 as follows:

$$w_{ij}^{(n)} = w_{ij} \cdot c^{(n)}(49i + j) \quad (i, j = 0, 1, 2, \cdots, 48; \quad n = 1, 2, 3, \cdots 256),$$

where $w_{ij}^{(n)}$ denotes ij component of the n-th weight matrix in the population. In each generation, this produces 256 weight matrices slightly different from W_0. We note that the larger the population number, the higher the performances. Due to computational resource's limits, however, we set it to 256. **(4)** To evaluate fitness value f, randomly chosen ν bits of each stored pattern are flipped. Then they are given to the network and updated. Each inner product of one of the stored patterns and instantaneous neurons' states after giving a noisy version of the pattern is averaged over all the updating time steps up to certain time t_0 and over all the stored patterns. This process is repeated for n different combinations

of the ν-bit noisy inputs and averaged over them. That is, the fitness function is

$$f = \frac{1}{n \cdot p \cdot (t_0 - 1) \cdot 49} \sum_{k=1}^{n} \sum_{\mu=1}^{p} \sum_{t=2}^{t_0} \sum_{j=1}^{49} \xi_j^\mu \cdot s_{j\nu k}^\mu(t),$$

where $s_{j\nu k}^\mu(t)$ is the state of the j-th neuron at time t when the k-th combination of ν-bit noisy μ-th pattern is given to the network. Updating time t_0 is set to 98, i.e., twice the number of neurons, and we observed that was enough long. We must note that fitness 1 implies that all the noisy patterns are retrieved correctly, while fitness less than 1 includes many possible cases. If we evaluated the fitness at one point of updating time instead, some solutions might be limit cycle. We adopted the above time-consuming fitness evaluation in order to avoid the oscillatory solutions. **(5)** Two parent chromosomes are chosen uniformly at random from upper 40% of the population which is ranked by fitness. Then those are recombined to make one child chromosome, and an individual in the lower 60% of the population is replaced with it. This process is repeated until all the individuals in the lower subpopulation are replaced. **(6)** Recombinations are made with uniform-crossover. We tested several types of crossover including one- and two-point-crossover, and observed that the uniform-crossover outperformed the others. Furthermore, the offspring are mutated in time with probability 0.01 (mutation rate), where mutation rotates the value of randomly chosen allele in chromosome $c^{(n)}(i)$ cyclically, i.e.,

$$(1) \rightarrow (-1), \quad (-1) \rightarrow (0), \quad (0) \rightarrow (1).$$

(7) Unless highest fitness value reaches the value of 1 nor generation exceeds 12,000, individuals in upper subpopulation (40%) survive to constitute the next generation with their offspring (60%), and the processes from (2) to (7) are repeated.

7.2 Results and Discussion

In this section, we compare the results of the fitness evaluation using noisy inputs with the one obtained from our no-noisy version of GA. All these simulations were

carried out on networks consisting of 49 neurons as usual.

In Figure 29, we show the best-fitness versus generation resulted from fitness evaluation with noisy/no-noisy input. The dotted line in the figure is the representative sample from no-noisy version of the GA. A network with random weights is evolved and can store eventually a maximum of 8 patterns as fixed points at generation 6,449. Then this matrix is used as W_0 in noisy counterpart of our Genetic Algorithm. The results are shown with solid line in the figure. The perfect solution emerged at the 6,369-th generation. This matrix also stores the above 8 patterns as fixed points. In this latter experiment, we evaluate fitness by averaging over 20 repetitions giving 5 random noises each time. Note that in this case, the best-individual will not be necessarily the best in the next generation even under the elitist strategy. We also run this noisy version starting with random weight matrix instead, however we have not obtained 100% correct individuals to date (not shown in the figure).

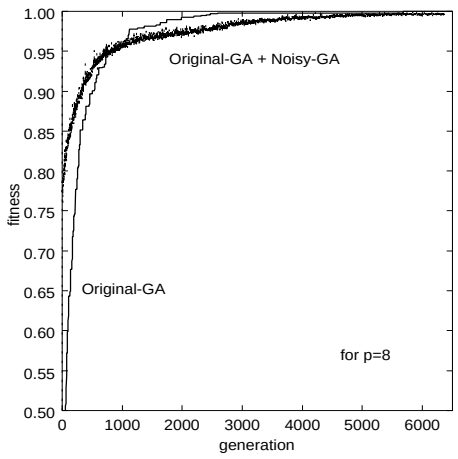

Figure 29. Fitness vs generation resulted from GA with noisy/no-noisy input.

This fitness evaluation with noisy input is designed to enlarge the size of basin of attraction. In Figure 30, we show the degree of tolerance to noisy input of the two networks appeared in Figure 29, together with the original Hebb-rule associative

network. In the figure, we plot the similarity of updated neuron-states for the noisy input to its complete pattern as a function of number of noisy bits given to the input. Network starts out with an initial configuration of neuron-states, and this configuration changes in discrete time steps according to asynchronous update. When one of noisy input of stored patterns is given to the network, we are to obtain a configuration of the neuron-states after updating enough time steps (= 98 here). And this is compared with the initial configuration of complete version of its input. The comparison is made by using cosine of the angle between two vectors which represent the two configurations above. These are averaged both over all inputs of stored patterns and over several runs (= 800 here). To be more specific, each of the stored pattern ξ^μ is added ν bits of noise at random, and given to the network to test the tolerance to the noise. This is repeated 800 times. Then the similarity $\zeta(\nu)$ is defined by

$$\zeta(\nu) = \frac{1}{800 \cdot p \cdot 49} \sum_{k=1}^{800} \sum_{\mu=1}^{p} \sum_{j=1}^{49} \xi_j^\mu \cdot s_{j\nu k}^\mu(98),$$

where $s_{j\nu k}^\mu(t)$ is the state of the j-th neuron at time t when the k-th set out of these 800 combinations of ν-bit-noisy μ-th pattern is given to the network. The slower the decay of the curve, the broader the size of basin of attraction. As we can see in the figure, the decay of the curve for the network obtained from no-noisy version is extremely steep. It implies that although the GA can evolve a random matrix to store all the 8 patterns as fixed points, the size of basin of attraction is much smaller than that of the original Hebb-rule associative memory. However, GA with noisy input in evaluating fitness improves the size significantly, though it is still smaller than the original Hebb-rule associative memory.

7.3 Summary

Previous chapters have mainly concerned with storage capacity of the weight configurations obtained by evolutionary algorithms. The storage capacity is traded-off with the basin of attraction, the capability of the network to tolerate noises. In this chapter, the basin of attraction of weight configurations obtained by a GA has been addressed. We found that the basin size of networks obtained by evolutionary algorithm is very small in general. At the first glance, it seems to be

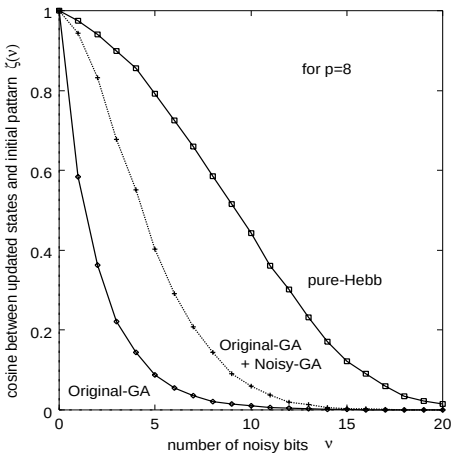

Figure 30. Similarity of updated states of a noisy input to its corresponding memory.

easy to enlarge the basin size of these networks also by an evolutionary algorithm, but all we observed was that the enlargement is just in between the basin size at the beginning of a run and the basin size of the Hebbian weights.

8. FITNESS LANDSCAPE

The concept of the fitness landscape was first introduced by Wright (1932) to study biological evolutionary processes. Since then, this concept has been used not only in evolutionary biology but also in chemistry, physics, computer science and so on.

In chemistry, for example, a molecule can be represented as a string of N letters with each letter being chosen from an alphabet of size k (Macken, Hagan and Perelson, 1991). Twenty amino acids ($k = 20$) for proteins or four nucleotides ($k = 4$) for nucleic acids can be considered as examples of the alphabet. The k^N possible combinations of the letters construct a configuration space of the string. Then, for example, the free energy of RNA folding into secondary structures (Fontana and Schuster, 1987), or the ability of peptides to bind to a particular substrate to catalyze a specific reaction (Maynard Smith, 1970) is assigned as a fitness value to each configuration. In physics, the Hamiltonian energy of Ising spins defines a fitness landscape on the configuration space of N spins. Each spin takes the value either 1 or -1 ($k = 2$). Bray and Moore (1980) argued about the number and distribution of meta-stable states (local optima) of the Hamiltonian energies.

To explore these fitness landscapes, we need a rule by which a point in the space moves to one of its neighbors. Then, consecutive movements of a point to the neighbors form a *walk* on the landscape. Macken et al. (1991) used *random point mutation* that changes a single letter in the string to specify neighbors of the string. Then, by sampling points along an *evolutionary walk* in which point moves to the *firstly found fitter* neighbor, they studied the statistical properties of the landscape of the chemical affinity of antibody for antigen in immune response. Weinberger (1990) used two different walks: "gradient walk" in which the walker steps to the *best* of its neighbors and *"random adaptive walk"* in which the next step is chosen *at random* from the set of better neighbors, to investigate the Kauffman's NK landscape which is a model formulated in more general form (Kauffman and Levin, 1987).

The fitness landscapes described above are all constructed on the space of *discrete* configurations of string. What we are concerned, however, is a fitness landscape defined on the *continuous* weight space, following Gardner (1988) who

discussed solutions in terms of the volume of the solutions in the space. However, the number and distribution of these solutions are still open problem. We explore the fitness landscape to obtain the information as to the number and distribution of the solutions in weight space. To explore the fitness landscape defined on this continuous weight space, we use a Gaussian random mutation, which is described later in detail.

8.1 Hopfield Model and Fitness Landscape

Although the analysis of the Hopfield model is somewhat of a classical problem, many issues such as the number and distribution of the global/local optima, height of the local optima are still unknown. To study these issues is one of our goal of this thesis.

The Hopfield model described in this thesis, when synaptic weights are selected randomly, can be considered as the Sherrington-Kirkpatrick (Kirkpatrick and Sherrington, 1978) model of Ising spin-glasses (SK-model). Bray et al. (1980; 1981) studied the number of meta-stable states of the SK-model, using correlations between the Hamiltonian energies of the meta-stable states. Derrida and Gardner (1986) solved exactly the number of meta-stable states in one dimensional spin-glass chain, answering the question of what stable state will it fall into if a spin configuration is evolved in time. Amitrano et al. (1987) analyzed the Hamiltonian energy landscape in the study of chemical evolution of *information-carrying macromolecules* represented by a set of Ising-spin.

Thus, fitness landscapes of the spin-glass model, or equivalently, the Hopfield model of associative memory have been fairly well studied. However, it was *discrete* fitness landscapes that were explored in those studies. Namely, the landscapes were defined on binary string space. On the other hand, Gardner (1988) discussed weight configurations of the Hopfield associative memory in terms of the volume of solutions in synaptic weight space, where the number and distribution of solutions in weight space is still an open issue. This is one of the reasons of our interest in the fitness landscape defined on weight space rather than defined on pattern space.

To construct a fitness landscape and explore it, we need two rules, as Macken et al. (1991) wrote. One is to determine the fitness value of each point in the space,

and the other is to specify evolutionary paths on the landscape. As for the fitness evaluation of points, we can use our fitness function described earlier. To specify paths on the landscape, we use the Gaussian random mutation. Usually, to make a point explore the landscape defined on *binary* string space, a bit-flip-mutation is employed. However, this mutation does not make sense on our *continuous* space. In this thesis, a point moves by the Gaussian random mutation. That is, a point

$$(x_1, x_2, \cdots, x_N)$$

is mutated by adding a random Gaussian variable to each coordinate x_i, i.e.,

$$x_i' = x_i + \sigma \cdot N_i(0, 1),$$

where $N_i(0, 1)$ is normally distributed random variable of mean 0 and standard deviation 1 sampled for each individual i. Note that the step length in Euclidean distance between parent and child is

$$\left(\sum_{i=1}^{N} (\sigma \cdot N_i(1, 0))^2 \right)^{1/2}.$$

8.2 Statistical Analyses

Our studies of finding an appropriate configuration of synaptic weights using evolutionary algorithms show that the more the number of patterns to be stored, the more difficult the task becomes. This is probably due to the increasing local optima as well as the decreasing global optima.

The questions now are how many local/global optima are, how they are distributed, how large their basins of attraction are, and so on. To obtain the information to answer these questions, the ruggedness of the fitness landscape is investigated statistically here. The ultimate goal is to learn the whole geometry of the fitness landscape.

The simplest way to learn the distribution of fitness values would be by enumerating the fitness value of each point of weight space. This, however, is not realistic due to the huge number of points in our weight space. In the current problem, even if we restricted each weight value to the floating-point number from $\{-1.00, -0.99, -0.98, \cdots, 1.00\}$, the weight space is constructed with 201^N

points. As such, statistic properties of a collection of randomly sampled points picked up either from the whole landscape, or along an evolutionary path should be employed.

In the following subsections, two methods which were originally devised for discrete landscapes are applied to our continuous landscape of the Hopfield network, with the aim being similar.

8.2.1 Correlation Coefficient

Both Manderick et al. (1993) and Lipsitch (1993) proposed a method to calculate the correlation coefficient of a single evolutionary operator such as a mutation or a crossover. To be more specific, the correlation coefficient of a set of fitness pairs of two points generated by the operator. We use this method to calculate the correlation coefficient ρ of Gaussian random mutation (Imada et al., 1998d). From weight space, q points are randomly chosen as parents. Then, by applying the Gaussian random mutation of mean zero and standard deviation σ to each point, q mutants are obtained as children. Fitness values of both parents and children are denoted as $f_p^i(\sigma)$ and $f_c^i(\sigma)$ $(i = 1, 2, \cdots, q)$, respectively. These $2q$ fitness values are repeatedly calculated for the value of σ from 0.001 to 0.5. And the correlation coefficient as a function of σ is obtained as follows:

$$\rho(\sigma) = \frac{\sum_{i=1}^{q} (f_p^i(\sigma) - \overline{f_p})(f_c^i(\sigma) - \overline{f_c})}{\left(\sum_{i=1}^{q} (f_p^i(\sigma) - \overline{f_p})^2\right)^{1/2} \left(\sum_{i=1}^{q} (f_c^i(\sigma) - \overline{f_c})^2\right)^{1/2}},$$

where $\overline{f_p}$, $\overline{f_c}$ denotes average over q samples of parents and children, respectively.

In Figure 31, we show four examples of correlation coefficient of the landscapes for $p = 1, 6, 49$, and 98, as a function of σ. The number of sampled points, q, was 600. As Lipsitch wrote: correlation provides a measure of how rugged a given landscape is. Highly correlated landscapes are smooth because nearby points have similar fitness values while less correlated landscapes are more rugged having larger fluctuations in fitness over short distance.

The results, as shown in Figure 31, suggest that fitness values between two separate points are more correlated when p, the number of patterns to be stored,

is larger. This is against our intuition that the landscape becomes more rugged as p increases. Though we suspect the results are probably because of increasing area of flat region where the fitness is relatively low, we have not known the reason of this phenomenon.

Lipsitch (1993) defined the *correlation length* of a landscape as the value of σ at which $\rho(\sigma)$ first becomes non-positive. The figure suggests that the correlation length is longer for larger p. This is also against our intuition.

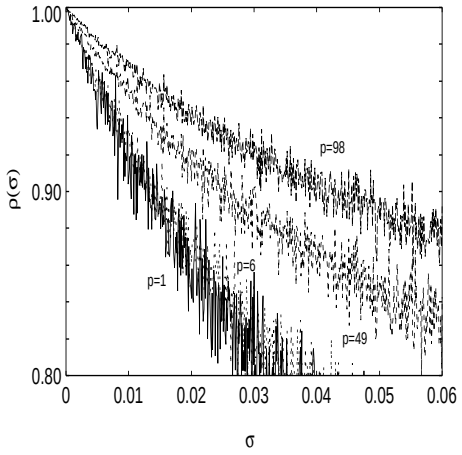

Figure 31. Correlation coefficient as a function of σ for 4 landscapes.

8.2.2 Autocorrelation Function

In this subsection, we calculate the autocorrelation function along a walk on fitness landscape defined by the Hopfield model. The autocorrelation function calculated here are based on the method by Weinberger (1990). Weinberger's fitness landscape is explored by a random walk via one point mutation.[33] That is, starting with a randomly chosen point, one randomly chosen bit in the string is repeatedly flipped until fitter neighbor is found.[34] Then the walker moves to the neighbor, and the process of the mutation is iterated until no fitter point is

[33] Neighboring points differ by exactly one bit from the current point.

[34] Alternative way is to choose the fittest point after testing all the neighbors.

found. Thus a time series of fitness value y_t is generated, and the autocorrelation as a function of time interval i is defined as:

$$\rho_i = \frac{E(y_t \cdot y_{t+i}) - E(y_t) \cdot E(t_{t+i})}{V(y_t)},$$

where $E(\cdot)$, $V(\cdot)$ is expectation and variance taken over all times t and all initial point y_0.

We apply this to our *continuous* landscapes defined by the Hopfield model (Imada et al., 1998d). First, we produce a random walk on the search space. It starts with a point chosen at random and the next point in the walk is determined repeatedly by adding a Gaussian random variable of mean zero and standard deviation σ to each coordinate of current point. The walk lasts q steps. At each point, fitness value is calculated. Thus we obtain a sequence of fitness values:

$$f_0, f_1, f_2, \cdots, f_q$$

where f_0 is a fitness value of the starting point. The autocorrelation function $\rho(h)$ of the fitness sequence represents the correlation coefficient between a pair of fitness values of two points separated by h steps along the random walk, and is calculated as follows:

$$\rho(h) = \frac{R(h)}{R(0)},$$

where

$$R(h) = \frac{1}{M} \sum_{i=0}^{q-h} (f_i - \overline{f})(f_{i+h} - \overline{f}) \quad \text{and} \quad \overline{f} = \frac{1}{q+1} \sum_{i=0}^{q} f_i.$$

This roughly corresponds to the distance one can jump maintaining some information about the fitness.

The results for $p = 1$, 6, and 49 are shown in Figure 32. For each value of p, we observed ten walks with different random number seed. Following Lipsitch (1993), we defined *correlation length* of a landscape as the step length at which $\rho(h)$ first becomes non-positive, and two representative results each of which has the shortest and the longest correlation length (out of ten runs) were shown. We can see the results vary from one run to another. Moreover, we cannot find any dependence of the results on p. These observations indicate that the fitness landscape is not isotropic.

Manderick et al. (1993) also defined the correlation length τ as the distance where the autocorrelation function is

$$\rho(\tau) = 1/2.$$

As Figure 32 says, the correlation lengths of the landscapes seem to be too long in order for us to regard the correlation length as a number of steps where information of the initial points still remains.

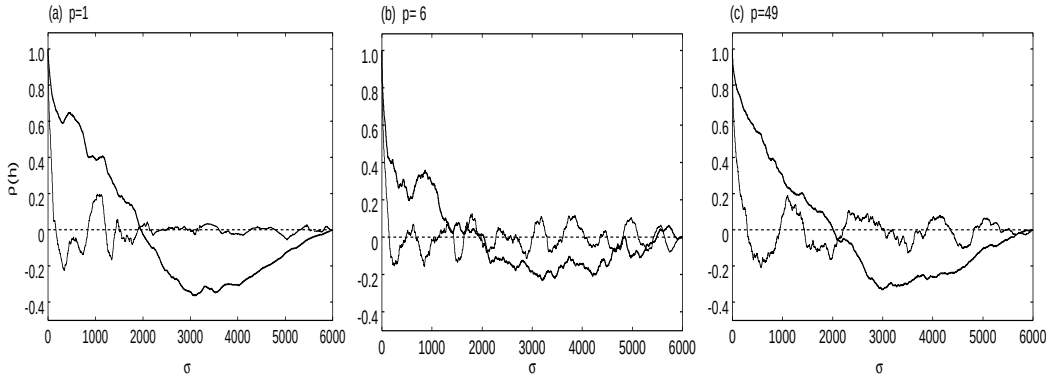

Figure 32. Fitness correlation coefficient along a random walk.

We have described, in the context of our *continuous* fitness landscape of the Hopfield model, two statistical methods that were successfully employed to measure the ruggedness of *discrete* fitness landscapes.

One method showed the opposite result to our conjecture, and the other was found not to work in our problem. Our conjecture is that the fitness landscape becomes more rugged, namely, the number and diversity of local optima increase and their basins' sizes decrease, as the number of patterns to be stored increases.

One of the possible reasons of these unsuccessful results is that the sampled points are not representatives of the whole fitness landscape. This holds true in whichever case where these samples are taken uniformly at random from the space, or along a random walk. For example, it is highly unlikely that arbitrary sampled points happen to include the global optimum, or even a local optimum. Therefore, as will be mentioned in the next section, we started to study the fitness

landscapes by observing directly the traces of down-hill-walks from a peak, which gives us a little bit more precise image of the shape of peaks (Imada et al., 1998a; 1998b). However, the information in this case is quite a local one. Hence the statistical analysis will still be needed. Now, we study other statistical methods like *information measure* proposed by Vassilev (1997).

8.3 What does the Landscape of a Hopfield Associative Memory Look Like?

One of the goals of this study is to obtain the number and distribution of the solutions of weight configurations which give a network a function of associative memory. In other words, one of our aims is to learn a geometry of a fitness landscape defined on weight space.

8.3.1 A Bird's Eye View of the Landscape

Since experiments here were carried out on networks with 49 neurons, the fitness landscape is defined on the $49^2 (= 2401)$ dimensional Euclidean space. Fitness of each point in the space is associated with the capability of the corresponding network to store a set of given patterns as associative memory. That is to say, the fitness surface is determined by the fitness value of each point of weight space. Hence the distribution of these fitness values gives us an information as to a bird's eye view of the landscape. To obtain the information of the landscape for, say $p = 6$, we picked up 240,000 points[35] randomly from the space. Note that p, the number of patterns to be stored, determines the ruggedness of the fitness landscape. The results are shown in Figure 33. The observed fitness values were distributed only from 0.10 to 0.46. Peaks are too narrow to be viewed.

8.3.2 The Shape of the Global Optimum

Besides the goal of obtaining the number and distribution of solutions in weight space, the analysis of fitness landscape will give us another information of a search space. When we search for an infinitely large number of solutions by

[35] The number of samples here is taken from the typical number of fitness evaluations in a run of our evolutionary algorithms.

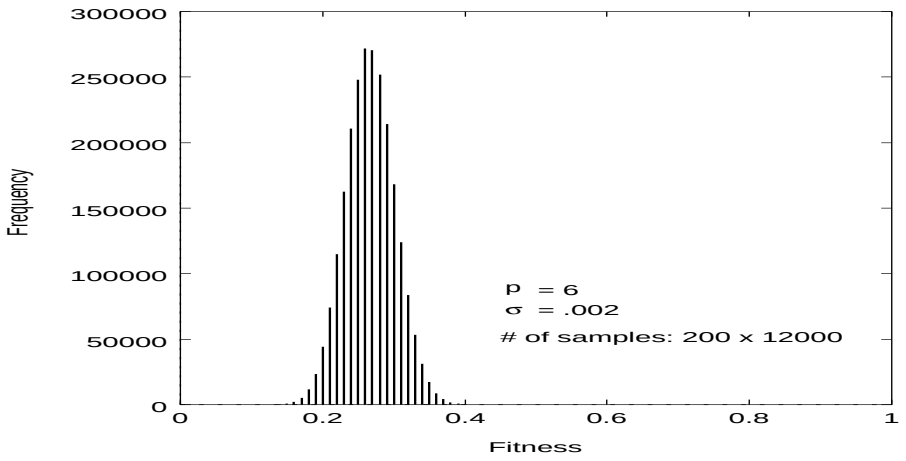

Figure 33. Distribution of fitness values on the landscape for $p = 6$.

evolutionary algorithms, it is helpful to learn the topology of the fitness landscape to know whether the solutions we obtained are representative samples of the whole solutions. Some solutions are easy to be approached and others are not in general. As a step to learn the whole geometry of a fitness landscape, we exploit a downhill walk by Evolutionary Programming to reveal the shape of global peaks on the fitness landscape defined on weight space.

In his Ph.D dissertation, Jones (1995) argued that *"a reverse hill-climbing algorithm allows the determination of details of the basin of attraction of points on a landscape."* Or, *"the method makes it possible to compute statistics regarding basins of attraction in a landscape graph."* The landscapes that Jones explored were defined on a discrete domain, that is, on vertices of a hyper cube. Here, we extended the idea to a landscape that is defined on a continuous domain.

As a preliminary experiment to walk down a hill, we use a *simple random walk* in which a point moves consecutively to one of its neighboring points at random by the Gaussian mutation. That is, we give small perturbation on each coordinate of the current point, x_i $(i = 1, 2, \cdots, 2401)$, by adding a Gaussian random variable with mean 0 and standard deviation σ_0, i.e.,

$$x_i^{\text{new}} = x_i^{\text{old}} + \sigma_0 \cdot N_i(0, 1).$$

This mutation enables a point to make a *random walk* on the fitness landscape,

where $\sigma_0 \cdot N_i(0,1)$ implies a *step length*.

We observed some *down hill random walks* from this Hebbian peak with several values of σ_0. Results are shown in Figure 34a. From this experiment, we tentatively set σ_0 to 0.002. We also made another experiment with this random walk, starting at a point chosen randomly from the space. A result is shown in Figure 34b. We can see that a walker wanders around a comparatively flat region with low fitness values, not even being able to approach to local peaks.

Then in order for walkers to take almost steepest path, we use Evolutionary Programming. Although the population in EP is usually randomized at the beginning of a run, we start it with all identical points, i.e., the top of the hill. This allows us to call it a *walk* and especially here a *down hill walk*.

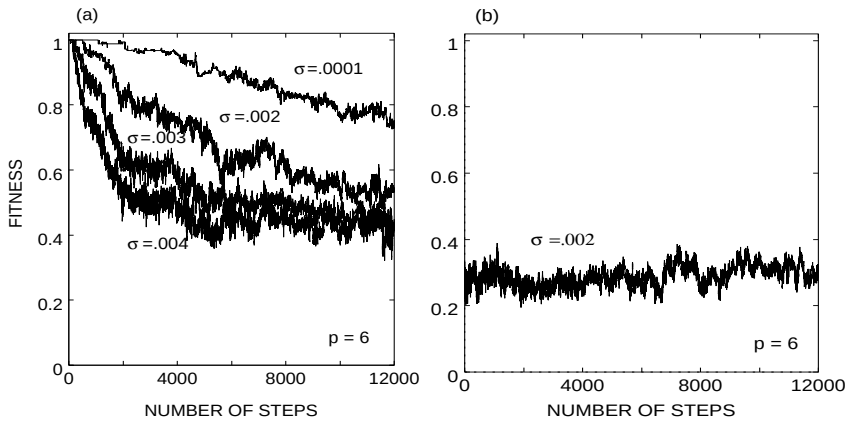

Figure 34. Examples of random walk: (a) downhill walks from a Hebbian peak; (b) random walk from a randomly chosen point.

In simple *hill climbing*, the Gaussian mutation is given λ times to the current point. Then, the point which obtains the highest fitness value is selected as the next point. In EP-walk, on the other hand, μ points construct a population. Each of these μ points is mutated once, which produces μ mutants. Each point has additional variables σ_i for each coordinate, which are used as a standard deviation of a Gaussian random variable to be added to the i-th coordinate of the point. After the mutation, these σ_i are also modified as:

$$\sigma_i^{\text{new}} = \sigma_i^{\text{old}} + 0.01 \cdot \sigma_i^{\text{old}} \cdot N_i(0,1).$$

All the σ_i are initialized to σ_0 ($= 0.002$) at the beginning of a walk. Now, we have the original μ points and their μ mutants. The fitness value of each of the 2μ points are compared to those of q points which are chosen randomly at every time of the comparison from the whole 2μ points. Then the 2μ points are ranked according to the number of wins, and the best μ points survive (q-tournament selection). The cycle of reconstructing the new population with better fitness points and restarting the search is repeated until a set number of iteration has been reached. Here, both λ and μ are set to 200, q is chosen to be 10, and the number of iteration is set to 12,000.

Thus, a walker will be able to walk down a hill. However, walkers sometimes cannot escape the basin of attraction of the hill. When we use the Gaussian mutation to move a point, the probability to escape from local/global optimum is not zero, at least theoretically. In practice, however, the distribution of step sizes by the Gaussian mutation was like the one shown in Figure 35a. We observed that the distribution of these step lengths ranges from 0.0102 to 0.0250.[36] If we obtain the minimum step size with which walkers can start down hill, we can estimate the basin size of global/local peaks when our walkers walk in the landscape.

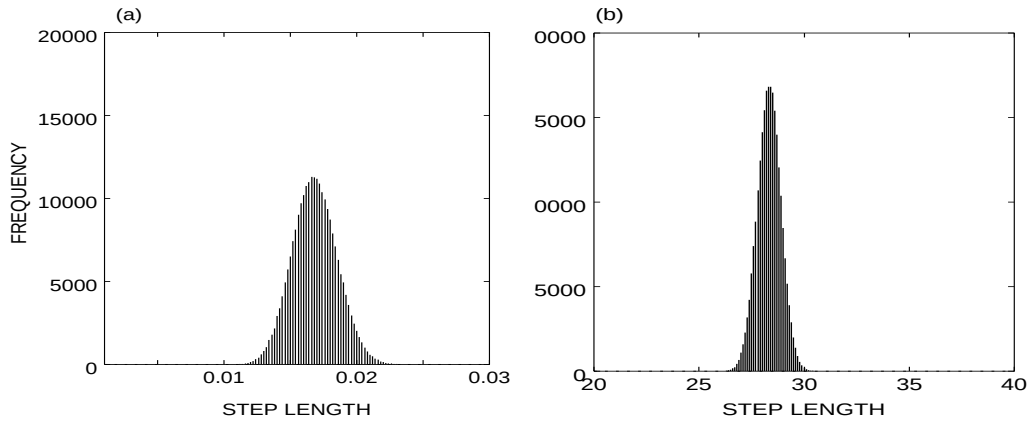

Figure 35. Distribution of step lengths. (a) Gaussian mutation; (b) Discrete crossover.

Note that if we use *discrete crossover* instead of mutation, step lengths from

[36] We can control the range by tuning σ_0.

one of the parents also shows Gaussian-like distribution, but step lengths take larger values, and in addition, we cannot control the distribution in this case. In our experiment, the lengths range from 25.8 to 30.8 as shown in Figure 35b. This is rather a long jump, or another version of random sampling, much more effective though. Since our aim is to learn the geometry of the fitness landscape, rather than to locate the global optimum efficiently, we use walks by mutation alone expecting to obtain the quasi-continuous information of the landscape along a walk of small step length.

Although we have thus far used the term *hill* or *peak* for the optimal configuration of w_{ij}, we must mention here the following feature of our fitness landscape. Since each w_{ij} can take an arbitrary real value, there are infinite number of equivalent configurations which differ only by scaling factor. In other words, for any scaling factor κ, κw_{ij} works exactly in the same way as w_{ij} in updating neuron states. Therefore, our landscape of the Hopfield network, if defined on the weight space, is made up of infinitely extended straight *ridges* emanating from the origin. The height of each ridge remains constant, but different from each other.

Exploration of Peak Shape by EP

In this section, we study EP-downhill walks from the top of the global peaks of two categories: the Hebbian peaks and peaks found by EP. All the networks we studied here were made up of 49 neurons.

Hebbian Peaks. Since every peak has its basin of attraction, walkers should walk with sufficiently large step sizes to start a downhill. We surmise that the minimum step size which enables a walker to start a downhill reflects the basin size of the peak. To obtain the minimum step size, the downhill walks are repeated with σ_0 being incremented starting at σ_0 with which walkers cannot start a downhill. For each σ_0, a downhill is repeated 10 times with different random number seed.

First, the six Hebbian peaks each of which corresponds to p, the number of stored patterns, from 1 to 6 were explored. The average of the minimum σ_0's obtained from 10 runs are plotted in Figure 36. We can see that the more the

number of patterns to be stored, the smaller the value of σ_0 is required. This suggests that the size of the top of a hill decreases as the number of patterns is increased.

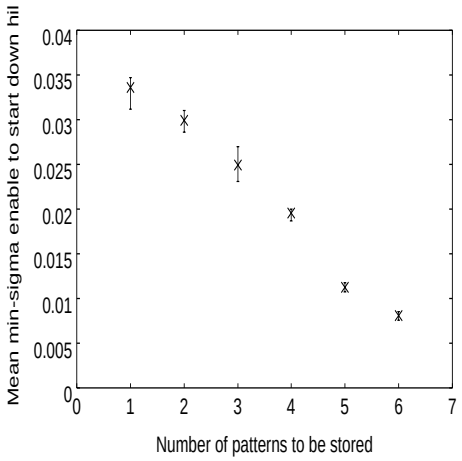

Figure 36. Dependence of the Hebbian peak width on the number of stored patterns.

Next, we observed the traces of the downhill walks from the Hebbian peaks. The traces are shown in Figure 37. We found that the traces from the peaks for $p = 1, 2$, and 3 were significantly different from each other, while the peaks for p more than 3 were somewhat of a similar one for $p = 3$ (not shown in the figure). In this experiment, σ_0 is set to 0.035 for all three peaks. With this value of σ_0, a walker can walk down all the three hills with the common step size, which allows us to compare the traces. We can observe that the less the number of stored patterns, the steeper the side wall is.

In observing downhill walks from the top of a hill, we often observe that a walker remains unmoved at the top for certain period. The walker struggles trying to escape the basin of attraction of the peak. At first we suspected that the period also reflects the basin size of the peak, but the length of the period is totally stochastic and we have not found a relation between the length of the period and the number of patterns.

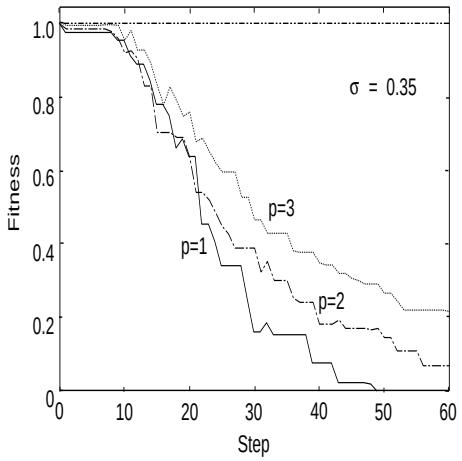

Figure 37. EP-downhill walk from the top of the Hebbian peaks.

Peaks Found by EP. The evolutions that emerged these two solutions are shown in Figure 39. The EP employed here were as follows. A fixed set of 6 random bipolar patterns is given to the network to be stored as associative memory. A population is constructed by μ points each of which has 49^2 coordinates x_i ($i = 1, 2, \cdots, 49^2$) representing weight values of a network. Each point has additional 49^2 variables σ_i ($i = 1, 2, \cdots, 49^2$) which are used for mutating the point itself. In this thesis, all the x_i are initialized to zero and all the σ_i are initialized to 0.002. Each of these μ points is mutated and selected in the same way as the EP walk described in Subsection 8.3.2. Both λ and μ are set to 200 and q is chosen to be 10.

Thus, we picked up two different global peaks to analyze their shape here: one that was found at a comparatively early (9,199-th generation) and one that required very long time to be found (53,735-th generation).

Then a EP walker walks down these two hills. Since the number of stored patterns is $p = 6$, σ_0 is required to be set as small as 0.008 this time. The results are shown fin Figure 39, together with the one for the Hebbian peak for $p = 6$. We can see that the side wall of both peaks found by EP are much steeper than the one of the Hebbian peak. Furthermore the side wall of the peak found earlier by EP is steeper than the one taken a longer time to be found. Hence we may

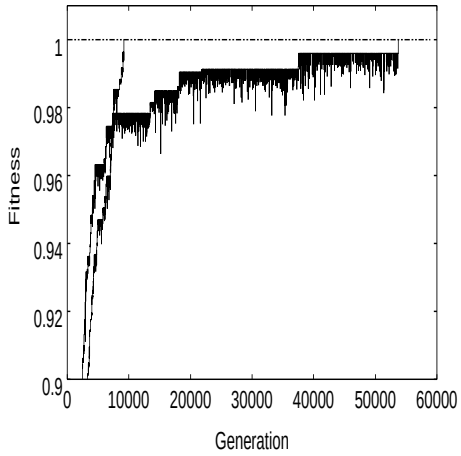

Figure 38. Two peaks found by EP.

conjecture that the EP we employed to find a weight configuration for associative memory tends to reach a sharper peak more easily than broader peaks.

8.4 Summary

To know the number and distribution of solutions in weight space, fitness landscape is considered in this chapter. Since the fitness landscape is defined on weight space and the solutions in weight space are represented as peaks on the landscape, learning the geometry of the landscape will give us the information as to the number and distribution of the solutions in weight space. First, the landscape is analyzed statistically using correlation function between pairs of points on the landscape. We have observed that the landscape becomes more rugged as the number of patterns to be stored increases, which is the reason why search becomes difficult when the number of patterns is large. However, the method does not give us more precise image of landscapes probably because the landscape is not isotropic. Then the geometry of the landscape was studied by learning the shape of peaks using a evolutionary walk. As a result, we observed that top of the peaks becomes narrower and the side wall of the peak becomes less steep as the number of patterns increases.

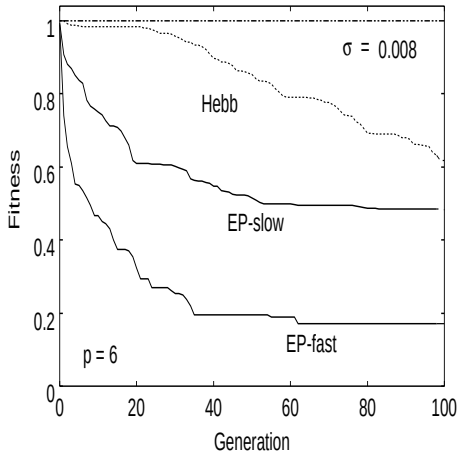

Figure 39. EP-downhill walk from peaks found by EP.

9. TO VISUALIZE SOLUTIONS IN WEIGHT SPACE: SAMMON MAPPING

In studies using evolutionary algorithms, visualization of high-dimensional space provides various aspects of insight into the search space explored. We can imagine, for instance, convergence/divergence behaviors of a population, topology of a fitness landscape, what does a walk from a random point to the global optimum look like, and so on. The problem of mapping a number of points in multi-dimensional space to points in 2-dimensional space with the distances among the original points remaining as much as possible is one of those techniques. Shine et al. (1997) and Collins (1997) argued such a technique together with other possible alternatives. Collins call this technique *"Sammon Mapping"* after Sammon (1969) who proposed this technique originally (Shine et al. call this *"Distance Map"*). Since the technique is an optimization problem, we can employ a genetic algorithm (GA) to solve this problem. Here we employ this technique in somewhat of a different way, that is, we apply the Sammon Mapping to visualize our weight space.

Since neither Collins nor Shine gave us any description such as how large dimensionality can be explored, or how many points can be mapped properly, we start by visualizing two known shapes in the space of high dimensionality. Then we apply the technique to our weight space of the neural network model of associative memory.

9.1 Sammon Mapping and its GA Implementation

As Collins (1997) wrote, the dimension reduction has been an important technique for visualization of the space of high dimensionality. The Sammon Mapping (1969) is one of these techniques. This enables us to map a set of $\mathcal{N}$ points in n-dimensional space to 2-dimensional location data so that the distance information is preserved as much as possible, or as Shine (1997) wrote *"so that the n-dimensional distances are approximated by 2-dimensional distances with a minimal error."* This problem is an optimization problem.

Shine et al. (1997) and Collins (1997) proposed a method to solve this problem by a Genetic Algorithm, as follows. First, the distance matrix whose entries are

Euclidean distances between all possible pairs of $\mathcal{N}$ points in the n-dimensional space is calculated. Then tentative $\mathcal{N}$ points in 2-dimensional space are determined representing the original $\mathcal{N}$-points in the n-dimensional space. The distance matrix of these $\mathcal{N}$ 2-dimensional points is also calculated, which then will be subtracted from the original n-dimensional distance matrix, yielding an error matrix. A GA is used to minimize this error matrix.

For the sake of simplicity, we assume here the dimension reduction from 2401-dimensional space to 2-dimensional space.[37] Given $\mathcal{N}$ points in 2401-dimensional space

$$X^1, \cdots, X^{\mathcal{N}},$$

where each point X^k is expressed by 2401 coordinates as

$$X^k = (z_1^k, \cdots, z_{2401}^k).$$

Then the square distance between m-th point and n-th point is calculated as

$$R^{mn} = \sum_{i=1}^{2401} (z_i^m - z_i^n)^2.$$

The values for all the possible combination of m and n construct a distance matrix. Since the matrix is symmetric with zero diagonal elements, we use the lower triangle elements alone($m < n$).

Then we generate $\mathcal{P}$ sets of the 2-dimensional $\mathcal{N}$ points at random, i.e.,

$$\begin{aligned}
&\Psi^1(1), \cdots, \Psi^{\mathcal{N}}(1) \\
&\Psi^1(2), \cdots, \Psi^{\mathcal{N}}(2) \\
&\cdots \\
&\Psi^1(\mathcal{P}), \cdots, \Psi^{\mathcal{N}}(\mathcal{P})
\end{aligned}$$

where the k-th point of the i-th set is represented as:

$$\Psi^k(i) = (\zeta_1^k, \zeta_2^k).$$

Thus the i-th set of these $\mathcal{N}$ points has its distance matrix whose elements are

$$r^{mn}(i) = \sum_{i=1}^{2} (\zeta_i^m - \zeta_i^n)^2.$$

[37] Note that our concern here is the $49 \times 49 = 2401$ dimensional weight space.

The objective function of the i-th sets $f(i)$ can be defined as

$$f(i) = \sum_{m<n} r^{mn}(i) - R^{mn}.$$

Starting with a random configuration of $\mathcal{N}$ points in 2-dimensional space, the GA corrects these points one generation to the next applying crossover and mutation[38] to 2-dimensional coordinates. The correction is repeated until the error converges to an acceptable minimum.

9.2 Results and Discussion

Towards the goal of visualizing solutions in weight space, we apply the dimension-reduction technique to two toy examples, as test functions, in which distribution of the point is known. One is a set of points on a hyper-line, and the other is a set of points in the two separate regions. Our experiments of the fully-connected neural network model of associative memory are carried out on networks with 49 neurons, which implies the weight space is $49^2 = 2401$ dimensional space. So the dimensionality of the space to be visualized is set to 2401 in this thesis.

Hyper-line

The first test is a visualization of a hyper-line.

In mapping some points in high dimensionality to points in 2-dimensional space, there exists some constraints in general. It is clear, for example, that the four vertices of a tetrahedron in 3-dimensional space can never been exactly mapped to four points in 2-dimensional space. On the other hand, there is no such constraint in the case of points on a hyper-line. In that sense, hyper-line is a good benchmark for the algorithm.

First, we pick up 120 points that are distributed with equal interval on a diagonal line of the 2401-dimensional hyper-cube. To be more specific, the points are:

$$(x_1^k, x_2^k, \cdots, x_{2401}^k), \quad k = 0, 1, \cdots, 119$$

[38] We employ uniform crossover (1989) and BGA mutation (Mühlenbein et al., 1996) here.

where

$$x_1^k = x_2^k = \cdots = x_{2401}^k = -1 + k \cdot (2/119).$$

Then they are mapped to 120 points on 2-dimensional space so that the distance relation among the 120 points on the 2401-dimensional space is kept as much as possible. A result is shown in Figure 40 (left). We can see a straight line in the 2-dimensional space. The task to search for an appropriate configuration of 2-dimensional points is quite easy in this case. As evolution proceeds, the objective function that took the value 7,560,777 at the start asymptotically approaches the small value around 0.1. The evolution is shown in Figure 40 (right).

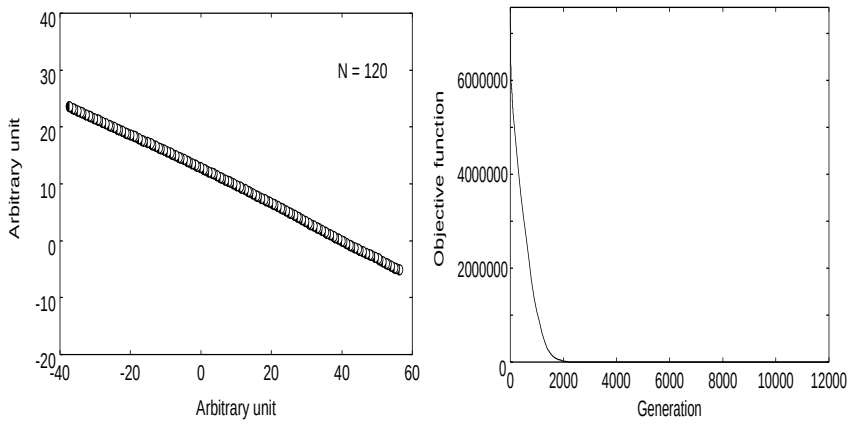

Figure 40. Points mapped to the 2-dimensional space from 120 points on a diagonal line of the 2401-dimensional space (left), and the time evolution of objective function (right).

Two Hyper-cubes

Next, we proceed to an example in which we can imagine the shape of the region in high-dimensional space. We sampled 60 points randomly from the 2401 dimensional region whose coordinates are all between 0.5 and 1.0 as well as the other 60 points from the region whose coordinates are between –0.5 and –1.0. Namely, points are sampled either from two separate hyper-cubes of the same size. In Figure 41 (left), a result of dimension reduction of these 120 points is

shown, together with a point that corresponds to the origin of 2401-dimensional space. The evolution of the objective function is shown in Figure 41 (right). The value starts with 14,567,428, and eventually approaches 106,125. Though the final value of the objective function is not so small, the ratio of the final value is 0.7% of the initial value. We can clearly see the two separate regions in the 2-dimensional space.

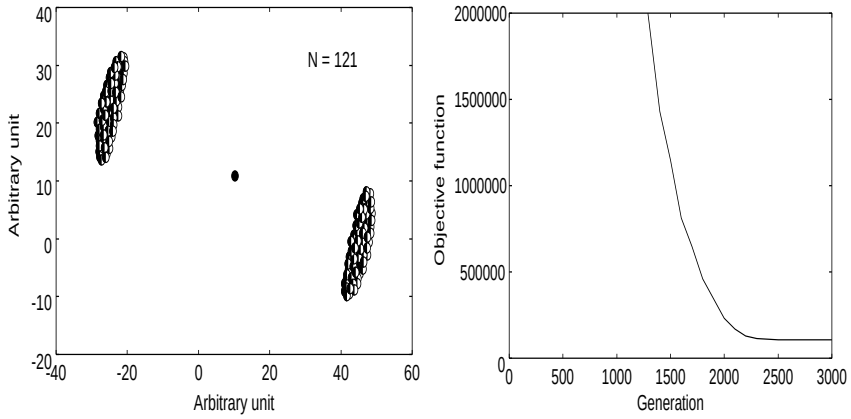

Figure 41. Two regions of the 2401-dimensional space mapped to the 2-dimensional space (left), and the time evolution of objective function (right). Filled-in circle • indicates the origin.

Hyper-sphere

As stated earlier, multiple configurations of weights give a network a function of associative memory. The number of these configurations is known to be dependent on p, the number of patterns to be stored. Storing just one pattern gives a maximum number of solutions of weights, while as p approaches twice the number of neurons, all the solutions vanishes (1988). However, the number and distribution as a function of p is still unknown. Here, we study the solutions found by the Breeder Genetic Algorithm (BGA) among others, since only this algorithm has been able to search for solutions for a wide range of p (see (Imada et al., 1997)). Our experiments were carried out on networks with 49 neurons and the BGA found solutions for up to $p = 90$ (see Subsection 3.2.7). The solutions that the BGA found are also expected to be different from run to run, as Mühlenbein

et al. (1996) wrote: *"the BGA mutation scheme is able to optimize many multi-modal functions."* As a preliminary stage of the goal of learning the number and distribution as a function of p, we sampled 30 such solutions for $p = 1$. It is important to note here that since each weights, w_{ij}, can take an arbitrary real value, there are infinite number of equivalent configurations which differ only by scaling factor. In other words, for any scaling factor κ, κw_{ij} works exactly in the same way as w_{ij} in updating neuron states (see equation (1)). So, we normalized the solutions obtained such that they locate on the hyper-sphere of radius 1. We suspect that these normalized solutions for $p = 1$ are distributed uniformly on the surface of the hyper-sphere. We show the results of the 2-dimensional points mapped from the 2401-dimensional solution space for the number of solutions $\mathcal{N} = 9, 10$, and 30 in Figure 43 (a), (b), and (c), respectively. As can be seen in Figure 43 (a), nine 2-dimensional points corresponding to the solutions are almost uniformly distributed on the circle whose center corresponds to the origin of 2401-dimensional space, while the distribution of these 2-dimensional points are disturbed more or less for $\mathcal{N}$ more than 9.

In Figure 44, we show the time evolution of each objective function for $n = 9$ and $n = 30$. The value for $\mathcal{N} = 9$ starts with 1,758 while the value for $\mathcal{N} = 30$ starts with 37,115 and ended up 8 for $\mathcal{N} = 9$ and 129 $\mathcal{N} = 30$. The difference of these values is due to the degree of constraint of the dimension reduction problem.

We also tested the similar experiment with the dimensionality 256 instead of 2401, but we found that the results were almost the same (not depicted here), in that the limit in which points of high dimensionality are properly mapped to 2-dimensional space was around $\mathcal{N} = 9$.

Next, we apply the technique to solutions also obtained by the BGA runs for $p = 90$. This number of patterns to be stored is almost the upper bound of the storage capacity for a network with 49 neurons, and the solutions are expected to be localized into small region of weight space (Gardner, 1988). A result is shown in Figure 43 (d). Though resolution is not so good, we can anyhow imagine the localized solutions.

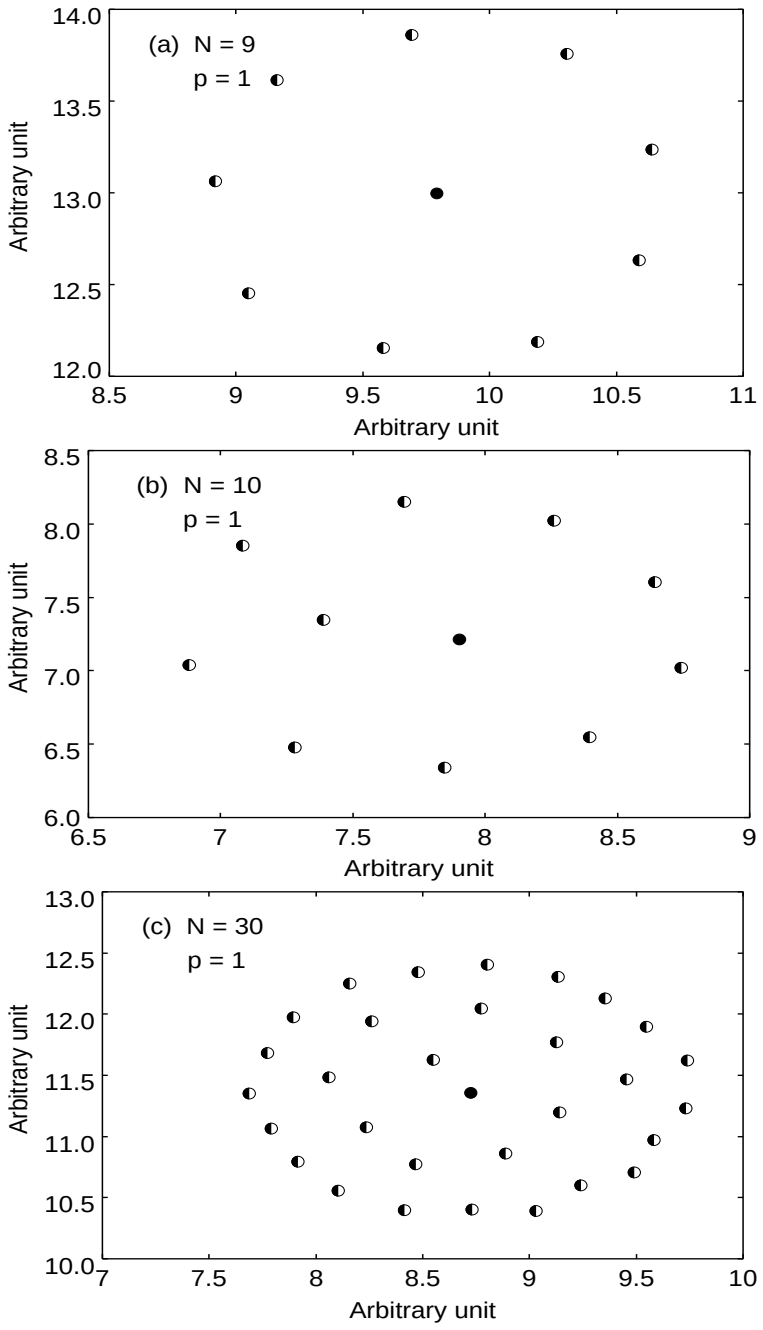

Figure 42. 2-dimensional points mapped from solutions in the 2401-dimensional weight space. Solutions that store 1 pattern where the number of mapped points is (a) 9, (b) 10, and (c) 30. Filled-in circle • indicates the origin.

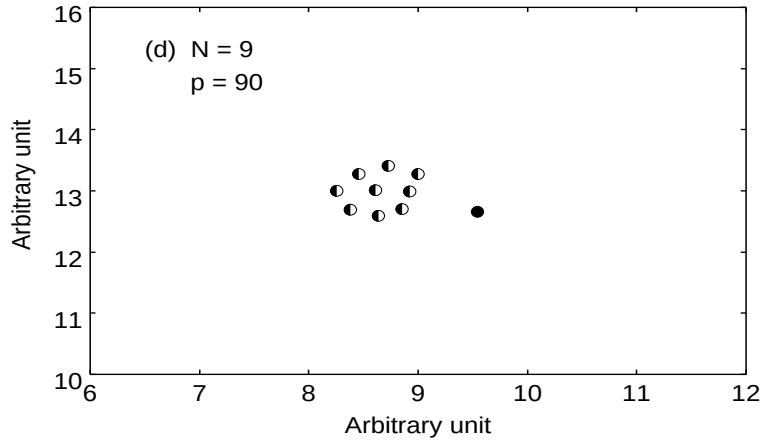

Figure 43. 2-dimensional points mapped from solutions in the 2401-dimensional weight space (continued). (d) Nine 2-dimensional solutions that store 90 patterns. Filled-in circle • indicates the origin.

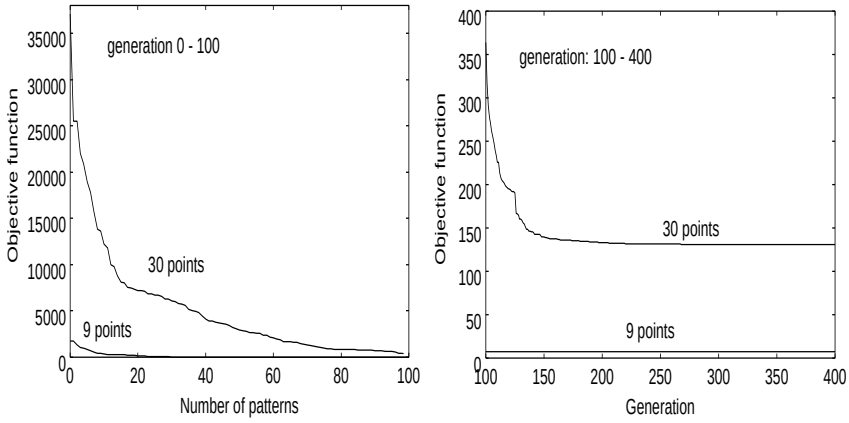

Figure 44. Time evolution of objective function.

9.3 Summary

This chapter has studied the distribution of the solutions in weight space, by mapping points in the high-dimensional weight space into two-dimensional space with the distance relations remaining as much as possible. Although the resolution is not so good, we observed that the solutions are uniformly distributed in weight space when the number of patterns is small, while the locations of the solutions are localized when the number of patterns becomes larger.

10. THE HOPFIELD MODEL AS A TEST FUNCTION

When we apply some variants of evolutionary computations to the fully-connected neural network model of associative memory, we regard it as a parameter optimization problem, we notice that the model has some favorable properties as a test function of evolutionary computations. So far, many functions have been proposed for comparative study. However, as Whitley and his colleagues suggested, many of the existing common test functions have some problems in comparing and evaluating evolutionary computations. In this chapter, we focus on the possibilities of using the fully-connected neural network model as a test function of evolutionary computations.

10.1 Overview

In the Genetic Algorithm community, many functions have been proposed to compare and evaluate different variants of Genetic Algorithms. Among others, De Jong's (1975) test suite has long been used as standard test functions. However, Davis (1991) pointed out that some of the De Jong's test functions can be solved more quickly using the next-ascent random bit hill-climbing technique.

Mahfoud (1995) used the following functions in his comparative study of niching techniques that are devised to solve multi-modal functions.

$$f_1(x) = \sin^6(5\pi x)$$

$$f_2(x) = \exp\left(-2\ln(2)(\frac{x-0.1}{0.8})^2\right)\sin^6(5\pi x)$$

$$f_3(x) = \sin^6\left(5\pi(x^{3/4}-0.05)\right)$$

$$f_4(x) = \exp\left((-2\ln(2))(\frac{x-0.08}{0.8})^2\right)\sin^6\left(5\pi(x^{3/4}-0.05)\right)$$

These four functions have five peaks. This set of test functions is typical in that f_1 has uniformly-distributed peaks of equal height, f_2 has uniformly-distributed peaks of unequal height, f_3 has non-uniformly-distributed peaks of equal height, and f_4 has non-uniformly-distributed peaks of unequal height.

In his paper, Mahfoud grouped eleven problems into three categories based

on difficulty, and the four functions above are categorized into the easiest group since they were also solved by his random hill-climbing. Furthermore, these functions are basically for Genetic Algorithms with binary encoded genes, where the problem of small dimensionality is worthwhile.

What we are interested in here are the real-valued parameter optimization problems with high dimensionality. Using a vector $\mathbf{x}$ which includes n parameters, we formulate the problems as

$$f^* = f(\mathbf{x}^*) = \max_{\mathbf{x} \in D} f(\mathbf{x}), \quad D \subset \Re^n. \tag{9}$$

We assume, without loss of generality, a maximization problem hereafter. As Bäck (1996) argued, since scaling defines the computational complexity of the algorithm, an arbitrary scaling of the dimension n is necessary. In order for a function to be scalable with respect to the dimension, a separable function is often used, as Whitley et al. (1995) wrote. Separable function F can be decomposed into n separate sub-functions $S(x)$, i.e.,

$$F(x_1, \cdots, x_n) = \sum_{i=1}^{n} S(x_i). \tag{10}$$

Mühlenbein et al. (1996) used the following two separable functions to evaluate their Breeder Genetic Algorithm.

$$f_5(x_1, \cdots, x_n) = \sum_{i=1}^{n} (10 \cos 2\pi x_i - {x_i}^2)$$

and

$$f_6(x_1, \cdots, x_n) = \sum_{i=1}^{n} x_i \sin \sqrt{|x_i|},$$

named Rastrigin and Schwefel function, respectively. Voigt et al. (1995) also argued that a test function should be defined for an arbitrary number of variables so that the number of steps to the optimum is a function of the dimension, noting that varying the problem size gives valuable information about the efficiency of the search method. Whitley et al. (1995), however, pointed out that since these two functions are separable, they are solved with fewer function evaluations using their Line Search Algorithm, which was designed to exploit the separability of functions by searching over all values of each parameter.

Mühlenbein et al. (1996) also used the following function named the Griewank function, which is not separable.

$$f_7(x_1, \cdots, x_n) = \prod_{i=1}^{n} \cos \frac{x_i}{\sqrt{i}} - \sum_{i=1}^{N} \frac{x_i^2}{4200}.$$

However, this is again criticized by Whitley et al. (1995) as follows. *"As the dimensionality of the search space is increased, the contribution of the product term is reduced because the number of local optima becomes smaller."* Whitley et al. (1995) then proposed two methods to construct scalable but non-separable functions using existing separable functions.

On the other hand, Bäck (1996) wrote in his recent book that these functions are possibly not representative of the average complexity of real-world problems, and proposed a fractal objective function using the Weierstrass-Mandelbrot function as a multi-modal function of high complexity.

For all these functions described above, we have information about the optimum, such as location, height, etc. For example, f_6 has an infinite number of maxima at the position $x_k \approx \pm k\pi\sqrt{i}$, of which N maxima are global with height 1.

"But are such problems typical applications?" as Mühlenbein (1996) said. In this chapter, we propose a more challenging test function using the fully-connected neural network model of associative memory. In this case, we have little *a priori* information about the optima. Hence, we believe that this study also sheds new light on the analysis of the model as well.

10.2 Features of the Model as a Test Function

10.2.1 Whitley et al.'s Requirements

The Model is Scalable but not Separable

In their paper, Whitley et al. (1995) noted that test functions should be scalable but not separable. To test the capability of an evolutionary technique, the difficulty of the problem should be tuned by changing the scale of the problem. However, most currently used test functions are separable to be scalable which gives us incorrect information as to the difficulty of the algorithm to be tested.

When we use the neural network model of associative memory as a test function, we can easily scale it by varying N, the number of neurons (scalable). Our objective function may look like a separable function at first glance; however due to its recursive calculations, as described in the previous section, it is not separable.

The Model is Resistant to Hill-climbing

Whitley et al. also argued that a test function should be resistant to hill-climbing. Although a fair amount of hill-climbing techniques have been proposed for the neural network model of associative memory, such as *perceptron learning* (Rosenblatt, 1962), (Gardner, 1988), what is denoted as hill-climbing here is a simple *population based* hill-climbing technique that enables to compare to our evolutionary computations. This allows us to study, for example, what characterizes the fitness landscapes for which *crossover* will be an effective operator (Forrest et al., 1993). To study mechanisms of evolutionary computations, we need fitness landscapes on which an evolutionary computation is likely to perform better than such a hill-climber. For the purpose, we tested a *hill-climbing walk* using the Gaussian mutation.

All the possible configurations of weight w_{ij} of a network with N neurons constructs the N^2-dimensional weight space. A walk on the space is formed by consecutively applying the Gaussian mutation to the current point. To be more specific, a small Gaussian random variable with mean 0 and standard deviation σ is given to each coordinate of the current point, i.e.,

$$w_{ij}^{\text{next}} = w_{ij}^{\text{current}} + \sigma \cdot N_{ij}(1,0), \tag{11}$$

where $N_{ij}(1,0)$ means that the Gaussian random variable is sampled anew at each time. This is called the Gaussian mutation. Then, the Gaussian mutation is given λ times to the current point, and the point with the *best* fitness among λ mutants is selected as the next point.

Thus the walker climbs a hill in the fitness landscape. We used this walk to explore the fitness landscape defined by the fitness of the network which had learned p patterns by the Hebbian rule (Imada et al., 1998a; 1998b). All the walks here started at the origin of the weight space. The landscapes are expected

to have many global peaks with fitness 1 besides many local peaks. The results are shown in Table 2. The Gaussian random hill-climbing located the global optimum only in the case that one pattern is stored, which is the easiest case.

Table 2. Statistics of the Gaussian hill-climbing.

Number of patterns	Number of successes	Ultimate-fitness			
		max	min	avg	std
1	10	1.000	0.871	0.967	0.00609
2	0	0.984	0.866	0.927	0.00645
3	0	0.926	0.789	0.879	0.00667

10.2.2 A Variety of Initializations

In Evolutionary computations, the issue of initialization is very important, though little attention has been paid. Usually, the first population is initialized at random, namely, individuals are randomly distributed in the search space at the beginning.

Surry et al. (1996) reviewed works concerning the issue of initialization. They emphasized *non-random initialization* to incorporate domain knowledge to GAs. Surry et al. cited Fogarty's (1989) work, where a GA is used to set the air inlet valves to the burners of multiple boiler furnaces to minimize combustion stack-loss in the common flue. Four different mutation strategies were tested on two initial settings: one starting with all air valves open and the other being randomly determined in between fully open and closed. The result is that the mutation whose rate is *initially high but exponentially decreasing*, worked effectively when starting with a *completely homogeneous population* while not particularly effective when starting with a *randomized population*, as Surry et al. summarized.

We can address the issue of initialization with the Hopfield model. In the classical analysis of the Hopfield model, Sompolinsky (1986) gave small perturbations on synaptic weights to see the robustness of the network for the synaptic

noise. Namely, each Hebbian synaptic weight J_{ij} was added small perturbation η_{ij}, resulting in

$$J_{ij} + \eta_{ij}.$$

We also modify synaptic weights based on this scheme using Genetic Algorithms, with the difference being that Sompolinsky chose synapses *randomly* while we choose them *adaptively*. We have two different versions of giving perturbations. One is by chromosomes and the other is by mutations. In the first implementation, we determine a matrix at the beginning of a run as an ancestor. This matrix remains unchanged during evolution. In each generation, chromosomes composed of small perturbations η_{ij} produce copies of the ancestor. These copies are slightly different from the ancestor depending on the range of η_{ij}. In the second implementation, on the other hand, weight values of a network are directly mapped into a chromosome and they are perturbed by random mutations. Then, *perturbation by chromosome* exploits a promising region, i.e., search points are restricted within a certain region (volume-oriented), while *perturbation by mutation* explores new regions, i.e., search points may wander all over the search space (path-oriented).

In the GA that uses *perturbation by chromosome*, we may start with either of an over-loaded Hebbian matrix[39], zero matrix or random matrix. Or in the GA that uses *perturbation by mutation*, chromosomes in the first generation might be initialized such that the corresponding weight matrices are either of all identical over-loaded Hebbian matrices, all identical zero matrices, or different random matrices.

If we give a pattern to a network of the above three types, the behavior will be different with each other. An input given to the random matrix will result in a chaotic trajectory due to the asymmetry of the weight matrix. An input to the over-loaded Hebbian matrix, on the other hand, will converge to a stable attractor in between the fixed-point attractor and spin-glass attractor (stable but far from the initial state). These behaviors are visualized by plotting over time the Hamming distance between the initial state given to the network and every instantaneous network state. Examples of the trajectories are shown in Figure 2. The goal of the genetic algorithm is to find a configuration of w_{ij} which makes

[39] Over-loaded means that the number of learned patterns exceeds the capacity.

all the trajectories a horizontal line from the origin (fixed point attractor).

10.2.3 Multi-modality

To control the complexity of the problem, we only have to vary p, the number of patterns to be stored. If we use our regular fitness function of how many patterns can be stored as fixed points, the fitness landscape has multiple global optima of height 1, and the number of the optima depends on p. Hence, we can evaluate a GA by the capability to locate one of these optima, with p being incremented.

Let us introduce here two GA implementations as examples, where we give perturbations on the weights either by chromosome or by mutation, as described in the previous subsection.

The Genetic Algorithm, whichever the implementation, proceeds as follows. Before each run, a set of p random bipolar patterns ξ^ν ($\nu = 1, 2, \cdots, p$) is produced. The goal of the algorithm is to search for the weight matrices that store all of these patterns as fixed points. At the beginning of a run, a population of chromosomes is initialized. A chromosome is a real-valued vector which comprises N^2 elements representing either a weight values $(w_{11}, w_{12}, \cdots, w_{NN})$ of an individual network or small perturbations $(\eta_{11}, \eta_{12}, \cdots, \eta_{NN})$ to be given on synaptic weights of an ancestor. Thus, a chromosome (genotype) corresponds to a weight matrix (phenotype). The fitness is evaluated as a capability of the phenotype to store given patterns as associative memory. According to the fitness, these chromosomes are ranked.

Two chromosomes are randomly chosen as parent from the best $T\%$ of the population. They are recombined with uniform crossover, i.e., two parents $(u_1, \cdots, u_n)$ and $(v_1, \cdots, v_n)$ produce an offspring $(w_1, \cdots, w_n)$ such that w_i is either u_i or v_i with equal probability. Every gene in the offspring chromosome, then, has a chance to be mutated with the probability p_m, either by being replaced with other random variable in the case of chromosome comprised of η_{ij}, or by being added a small random variable in the case of chromosome comprised of w_{ij}. The worst $(100 - T)\%$ of the population are replaced with offsprings produced in this way, and they survive to the next generation with their parents ($T\%$).

The cycle of reconstructing the new population and restarting the search is

repeated until a perfect solution is found or a set maximum number of generations has been reached. When the cycle terminates, it returns a weight matrix, possibly a optimal one.

All the experiments described in this section were carried out on networks with 49 neurons. Parameters used for the Genetic Algorithm are as follows. The population number is 256, In selecting two parents, T is chosen to be 40%. The mutation probability p_m is set to 0.05. The searching procedure is iterated until 12,000 generations unless a perfect solution is found.

Here, we evolve the *over-loaded Hebbian synapses*. For a fixed set of p patterns, we repeated a GA run 30 times with different random number seed. When at least one run succeeds in locating one of the global optima, we incremented p. Thus the maximum p is obtained. We made the experiment with two implementations mentioned above. We show the number of successes out of 30 runs in Table 3. We also show in the table the other statistics such as maximum, average, and standard deviation, of the best fitness values found in each run out of 30 trials. Whichever schemes we may use, the task becomes difficult as p becomes large. Hence, the maximum p thus obtained represents the capability of the method to locate one of the optima.

In the above two experiments, if a run converges to a solution, then individuals in the final population are almost similar (though each run might converge to a different solution). However, the Breeder Genetic Algorithm proposed by Mühlenbein et al. (1996) gives us another story. Mühlenbein et al. wrote, the BGA mutation scheme is able to optimize many multi-modal functions. As described above, our *perturbation by mutation* scheme chooses a gene x_i in chromosome with probability p_m and mutates it by adding a small *random* variable as a perturbation. Instead of generating this *small variable* randomly, the BGA calculate the perturbation to be added to x_i as follows: Assume that appropriate mutation range for each parameter x_i is pre-determined as Δ_i. Every time x_i is chosen, the factor:

$$\delta = \frac{\alpha_0}{2^0} + \frac{\alpha_1}{2^1} + \frac{\alpha_2}{2^2} + \cdots + \frac{\alpha_{15}}{2^{15}} \tag{12}$$

is generated such that α_k $(k = 0, \cdots, 15)$ takes the value 1 with probability 1/16, and otherwise 0. Note that δ ranges from 0 (i.e., $\alpha_0 = \cdots = \alpha_{15} = 0$) to

Table 3. Statistics of two GA runs.

Number of patterns	Number of successes	The best fitness in each run			
		max	min	avg	std
(perturbation by chromosome)					
14	22	1.000	0.995	0.999	0.00024
15	17	1.000	0.994	0.998	0.00041
16	5	1.000	0.989	0.997	0.00054
17	5	1.000	0.988	0.995	0.00067
18	1	1.000	0.979	0.990	0.00081
19	0	0.999	0.965	0.985	0.00124
(perturbation by mutation)					
25	7	1.000	0.991	0.998	0.00046
30	1	1.000	0.991	0.996	0.00040
31	1	1.000	0.992	0.996	0.00037
32	2	1.000	0.990	0.996	0.00040
33	0	0.999	0.990	0.995	0.00045

$2(1 - 1/2^{15})$ (i.e., $\alpha_0 = \cdots = \alpha_{15} = 1$). Then x_i is mutated to be

$$x_i \pm \Delta_i \times \delta, \tag{13}$$

with the sign $+/-$ being chosen with equal probability.

Furthermore, they produce $(\mu - 1)$ offsprings, and all survive to the next generation with the best individual in the current generation (recall that μ is population size). This is in contrast to our selection that $(100 - T)\%$ offsprings and $T\%$ parents construct next generation.

Voigt et al. (1995) suggested that multi-modal function where the highest maxima are clustered in an area are easily optimized. We found that starting with zero matrix, the BGA can locate solutions in the vicinity of zero matrix very easily even for a large p. We have observed that the BGA locates solutions for a given set of 90 patterns. The solutions in synaptic weight space of the fully-connected network seem to cluster within a certain domain around the origin in the space. Moreover we also observed that the BGA locates different solutions within a run (see (Imada et al., 1997g)). In the analysis of the neural network model of associative memory, those issues like how many solutions exist or how they are distributed in the weight space are still open.

Niching is a technique devised more intentionally to expand a conventional GA to locate all optima of a multi-modal function simultaneously. This is an analogy from nature of dividing a population into multiple subpopulations or *species*. In parameter optimization, the location of each optima is a *niche* which is filled with individuals of some species. Basically two methods have been proposed based on this analogy: *Fitness Sharing* (Goldberg and Richardson, 1987) and *Crowding* (De Jong, 1975). If these methods for the multi-modal function optimization locate all the optimal configurations of synaptic weights, it will give us new insight into the analysis of the model.

However, unfortunately, the task is extremely hard due to the high-dimensionality and continuous genes. Hence, we show an experiments in which a niching GA explores string space instead of weight space (Imada et al., 1998f). The fitness landscape we explore is defined on string space such that p peaks are created by the Hebbian rule. A fully-connected neural network with 49 neurons are employed, and a set of 49-bit random bipolar patterns are stored. So the domain to be searched for is the 49-dimensional hyper-cube instead of the 2401-dimensional

Euclidean space. The fitness of a point is evaluated as usual according to how the instantaneous neurons' states after the point is given to the network as an input are similar to either of the *a priori* stored patterns. The task is to search for p points of fitness one out of all possible 2^{49} points. The algorithm we adopted here is the *deterministic crowding* (Mahfoud, 1992) because of its niching capability (Mahfoud, 1995) as well as the simplicity for implementation.

Algorithm 4 (Deterministic Crowding GA) *In each generation the current population is reproduced as follows.*

(1) Choose two parents, p_1 and p_2, at random, with no parent being chosen more than once.

(2) Produce two children, c_1 and c_2, with uniform crossover (Syswerda, 1989).

(3) Mutate the children by flipping bit chosen at random with probability p_m, yielding c_1' and c_2'.

(4) Replace parent with child as follows:

- *IF* $d(p_1, c_1') + d(p_2, c_2') > d(p_1, c_2') + d(p_2, c_1')$
  - *IF* $f(c_1') > f(p_1)$ *THEN replace* p_1 *with* c_1'
  - *IF* $f(c_2') > f(p_2)$ *THEN replace* p_2 *with* c_2'
- *ELSE*
  - *IF* $f(c_2') > f(p_1)$ *THEN replace* p_1 *with* c_2'
  - *IF* $f(c_1') > f(p_2)$ *THEN replace* p_2 *with* c_1'

where $d(\zeta_1, \zeta_2)$ is the Hamming distance between two points (ζ_1, ζ_2) in pattern configuration space.

An example of the evolutions are shown in Figure 45.

We can see that the task is quit easy. It requires only around 30 generations for the best individual to reach one of the embedded niches (attractors). Average fitness of population also approaches the fitness one, which suggests all the individuals are attracted to one of the niches.

The number of individuals that converges at one of the niches are compared with the simple GA in Figure 46. We can see that all the individuals are attracted

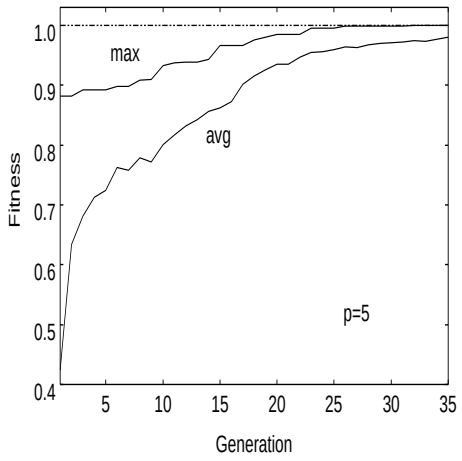

Figure 45. Best and average fitness vs generation obtained by the Deterministic Crowding GA.

to one of the niches in Deterministic Crowding GA, while the number do not increase after it reaches some level in Simple GA.

What then is the number of each individual in each niche? We show this in Figure 47.

Finally, examples of the numbers of individuals in each niche when the GA is terminated are shown in Table 4. Experiments are made by increasing p from four to eight. We succeeded in locating all the embedded niches for $p = 4$ and 5, however for p more than 5 the algorithm miss some of the niches.

10.2.4 Multi-objectivity

Besides multi-modality, the simultaneous optimization of multiple conflicting objectives, or equivalently, multi-objective optimization also has been addressed in the Genetic Algorithm community. In the conventional approaches, different objectives were combined into one objective to produce a compromise solution. On the other hand, the Genetic Algorithm, as Srinivas (1994) wrote, can capture a number of solutions simultaneously in a single run since it works with a population of points, and a modest amount of methods to solve the multi-objective problem has been proposed (Srinivas et al., 1994; Fonseca et al., 1995).

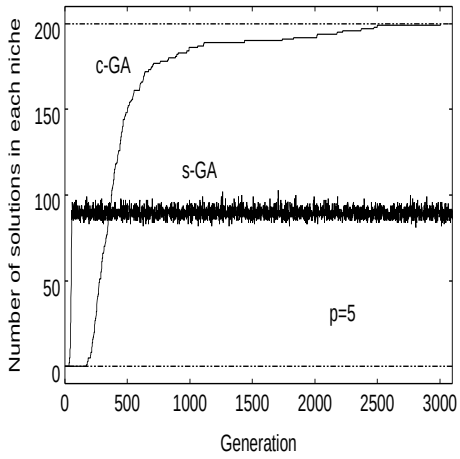

Figure 46. Number of individuals that reach solutions: comparison between the Deterministic Crowding GA and simple GA.

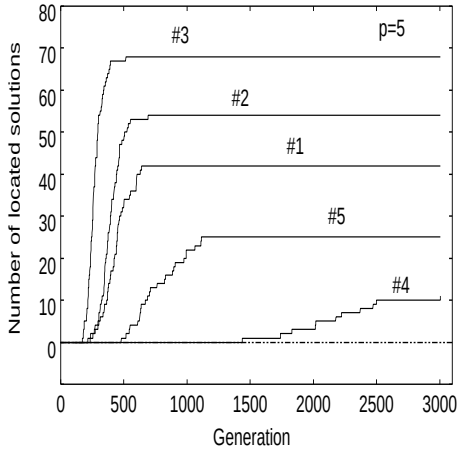

Figure 47. Number of individuals converged on each niche: the Deterministic Crowding GA.

Table 4. The number of solutions converged to each attractor.

	Number of located attractors						Total
p	#1	#2	#3	#4	#5	#6	
4	40	96	27	59	-	-	200
4	68	26	24	82	-	-	200
4	109	7	20	64	-	-	200
4	23	52	76	49	-	-	200
4	23	43	78	56	-	-	200
4	18	79	11	92	-	-	200
4	54	8	7	131	-	-	200
4	31	28	38	103	-	-	200
5	85	71	5	34	5	-	200
5	42	54	68	11	25	-	200
5	15	102	26	52	5	-	200
6	33	17	58	50	0	42	200

The associative memory model has multi-objectivity. There exists a trade-off between storage capacity and the size of the basin of attraction. We reported (Imada et al., 1996d) that we can expand the basin size by a Genetic Algorithm using a fitness function g, which is derived from the fitness function f simply by replacing ξ^μ with the noisy input (see Chapter 4). The fitness function g is competitive with a fitness function f which is for the enhancement of the storage capacity. Hence, we can regard the model as an example of a multi-objective function.

However, we have found it very difficult to expand the basin size even if we neglect the simultaneous demand of enlarging the storage capacity. Here, we show only a result of using single objective optimization based on g.

We visualized the basin size of the network as follows. One of the memorized patterns is randomly selected. After d bits of the pattern are flipped at random, the pattern is given to the network. Overlap between the memorized pattern and an instantaneous network state after the input is given, is calculated at every time step of updating. The overlaps are then temporally averaged after a certain period of time ($= 2N$ steps here). These processes of sampling an input pattern, giving it to the network after flipping d bits and calculating temporal average of the overlaps are repeated 800 times for fixed number of noisy bits d, and the temporal averages of overlaps calculated in each repetition are further averaged over the 800 repetitions. Thus, the averaged overlaps with d being incremented from 0 to 20, are plotted in Figure 30. The measurements were made for the following three weight matrices. (A) the matrix evolved from a random matrix under the fitness function f; (B) the matrix further evolved started with the matrix obtained in A using fitness function g[40]; and (C) the matrix which learned the given patterns by the Hebbian rule alone. They all store the same set of eight patterns as fixed points. However, the size of the basin of attraction obtained by the Genetic Algorithm with fitness evaluation g, is still smaller than that of Hebb rule associative memory. This is probably because of the remaining spurious attractors around the created memories.

[40]This implies restart of the GA with fitness function g after convergence of the GA with fitness function f.

10.3 Summary

We have described the possibility of using the fully-connected neural network model of associative memory as a test function of evolutionary computations. The model has some favorable properties as a test function for a parameter optimization problem:

- the objective function is *scalable but not separable*, and the scalability is easily changed by varying the number of neurons;
- the complexity is controlled with the number of a given set of patterns to be stored by the network;
- the model is *resistant to hill climbing*;
- the model is sensitive to the difference in initialization; and
- the model has *multi-modality* as well as *multi-objectivity*.

Although these properties have not been tested exhaustively, the model as a parameter optimization problem is closer to real-world problems than currently used test functions, in that we have little *a priori* information about the optimum in addition to its complexity.

We believe that this study also contributes to the new analysis of the associative memory model. In fact, many issues still remain open, despite the extensive discussions since the proposal of the model by Hopfield. If we locate, for example, all the solutions of synaptic weights which give a network a function of associative memory, it will give an insight into the number and distribution of the solutions in synaptic weight space, about which we have now only limited information.

11. CONCLUSION AND FUTURE WORK

We have described behaviors of a fully connected neural network model of associative memory under artificial evolution. A number of variants of evolutionary algorithm were applied to the model. The Darwin's principle of survival of the fittest, Baldwin effect, Lamarckian inheritance and so on. As a result, we have found that almost all variants successfully evolved over-loaded Hebbian weights, all zero weights, and random weights eventually to store a set of given patterns as fixed point attractors. Or equivalently, the algorithms found appropriate weight configurations that create fixed point attractors exactly at the location of patterns to be stored.

Simulations in this thesis were carried out on networks with 49 neurons. The storage capacity of this size of network is around 8 when the weight values are determined by the Hebbian learning algorithm alone. On the other hand, the networks obtained through evolutionary algorithm here have the maximum storage ranging from 1 to 90. Hence we conclude that the solutions of weight configuration of a network have a great diversity from implementation to implementation or even from run to run. That is to say, we have found many different global peaks in the fitness landscape defined on weight space. The difference of these peaks were argued in terms of degree of symmetry and rate of zero.

Although the analysis of the fully connected neural network model of associative memory is somewhat classical, many issues are still open. The number and distribution of the solutions in weight space are one such issue, and one of the goals of this thesis is to address this issue. For the purpose, we study fitness landscape defined on weight space and walkers walk down a hill from the top of the hill. And we observed that the more we try to store patterns, the narrower the top of peaks. Namely, the size of basin of attraction decreases as the number of patterns to be stored increases.

Further, we try to visualize these solutions by reducing the dimensionality of our $49^2 = 2401$ dimensional weight space to 2-dimensional points using Sammon mapping. The problem is an optimization problem, and we also use a GA to solve it. Gardner (1988) discussed the number of solutions of the weight configurations in terms of volume in weight space, and showed that the volume shrinks to zero as the number of patterns approaches twice the number of neurons. On the

contrary, if the number of patterns to be stored is one there are a plenty of solutions. We tackled this phenomenon by visualizing solutions in weight space. Although we have not been able to visualize many points in weight space due to a strong constraint of the problem, we observed that 9 different solutions are almost uniformly distributed when the number of storing pattern is one while they are localized when the number of patterns is 90.

Thus we believe these studies using evolutionary computations represents a fresh investigative approach, and sheds new light on the analysis of the Hopfield model of associative memory. At the same time, we believe that we can contribute to the evolutionary computation's community by discussing the possibility of using the fully-connected neural network model of associative memory as a more challenging test function than those currently used.

Acknowledgments

I owe it to many people that I will be able to complete this doctoral thesis. It is one of my pleasures to have this opportunity to thank them.

First of all, I am very grateful to three Professors Akira Fukuda, Yutaka Takahashi, and Minoru Ito who reviewed this thesis and gave me inspiring discussions and comments as thesis committee members at Nara Institute of Science and Technology (NAIST). Professor Fukuda has supervised me with his trust and generosity since I set out the theme of this thesis. When I enrolled at NAIST, I was fortunate to be supervised by Professor Keijiro Araki who is currently a professor at Kyushu University. At that time he had treated me as if my idea is very important. Both of the professors always has given me appropriate suggestions and constant encouragement. Thanks also goes to Professor Shunsuke Uemura whose laboratory happened to be the same floor as ours for the first two years of my life in NAIST. Ever since he has encouraged me at every once in a while.

I am very grateful to Katsunori Shimohara at Advanced Telecommunication Research Institute (ATR) who offered me an opportunity to work as a student researcher in a stimulating and encouraging environment at ATR for seven months. I would never want to leave. At ATR, I had been extremely lucky to meet a variety of researchers. I have profited from many discussions with Thomas Ray, who had proposed that the world-famous Tierra in Artificial Life community. I thank Shin Ishii, who is now assistant professor at NAIST, for giving me basic principles of the Hopfield model of associative memory. I was totally inspired from the stimulating discussions with Peter Davis regarding the dynamics of the Hopfield network.

It was tremendous for me to have many opportunities to attend international conferences. I have interacted many friends who inspired and influenced me a lot and were very friendly as well.

Without them all, it had not been for this thesis.

References

Amit, D. J., H. Gutfreund, and H. Sompolinsky (1985a) *Statistical Mechanics of Neural Networks near Saturation.* Annals. of Physics, 173, pp.30–67.

Amit, D. J., H. Gutfreund, and H. Sompolinsky (1985b). *Storing Infinite Number of Patterns in a Spin-glass Model of Neural Networks.* Physical Review Letters, Vol. 55, pp.1530-1533.

Amit, D. J. (1989). *Modeling Brain Function: The World of Attractor Neural Networks.* Cambridge University Press.

Amitrano, C., L. Peliti, and M. Saber (1987) *Population Dynamics in a Spin-glass Model of Chemical Evolution.* Journal of Molecular Evolution, 29, pp.513–525.

Atmar, W. (1990) *Natural Processes which Accelerate the Evolutionary Search.* R. R. Chen (Ed.) Proceedings of the 24th Asilomar Conference on Signals, Systems, and Computers. Maple Press Pacific Grove, CA, pp.1030–1035.

Bäck, T. (1996) *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, New York.

Bagley, J. D. (1967) *The Behavior of Adaptive Systems which Employ Genetic and Correlation Algorithms.* Doctoral dissertation. University of Michigan, Dissertation Abstracts International, 28(12), 5106B (University Microfilms No.68-7566).

Baldi, P., and S. S. Venkatesh (1987) *Number of Stable Points for Spin-Glasses and Neural Networks of Higher Orders.* Physical Review Letters, 58(9), pp.913–915.

Baldwin, B. J. (1886) *A New Factor in Evolution.* The American Naturalist, Vol.30, pp.441-451 & pp.536-553.

Banach, S. (1922) *Sur les Opérations dans les Ensembles Abstraits et leur applications aux équations intégrales.* Fundamental Mathematica, Vol.3, pp.133–181.

Barricelli, N. A. (1954) *Esempi Numerici di Processi di Evoluzine.* Methods, pp.45–68.

Barricelli, N. A. (1957) *Symbiogenetic Evolution Processes Realized by Artificial Methods.* Methods, Vol.9, No.35–36, pp.143–182.

Barricelli, N. A. (1962a) *Numerical Testing of Evolution Theories: I. Theoretical Introduction and Basic Tests.* Acta Biotheoretica, Vol.16, pp.69–98.

Barricelli, N. A. (1962b) *Numerical Testing of Evolution Theories: II. Preliminary Tests of Performance Symbiogenesis and Terrestrial Life.* Acta Biotheoretica, 16, pp.99–126.

Beyer, H.-G. (1994) Evolutionary Computation, Vol.2, No.4, MIT Press, pp.381–407.

Beyer, H.-G. (1995) *How GAs do NOT Work: Understanding GAs without Schemata and Building Blocks.* Technical Report SYS-2/95, University of Dortmund. (available via anonymous ftp from ftp://lumpi.informatik.uni-dortmund.de/pub/GA/papers/sys-95-2.ps.gz

Bledsoe, W. W. (1961) *The Use of Biological Concepts in the Analytical Study of Systems.* Paper presented at the ORSA-TIMS National Meeting, San Francisco.

Box, G. E. P. (1957) *Evolutionary Operation: A Method for Increasing Industrial Productivity.* Applied Statistics, Vol.6, pp.35–46.

Bray, A., and M. Moore (1980) *Metastable States in Spin Glasses.* Journal of Physics C: Solid State Physics 13, pp.L469–L476.

Bray, A., and M. Moor (1981) *Metastable States in Spin Glasses with Short-ranged Interactions.* Journal of Physics, C: Solid State Physics, 14, pp.1313–1327.

Bremermann, H. J. (1958) *The Evolution of Intelligence: The Nervous System as a Model of its Environment.* Technical Report No.1, Contract No.477(17), Department of Mathematics, University of Washington.

Bremermann, H. J. (1962) *Optimization through Evolution and Recombination.* M. C. Yovits, G T. Jacobi, and G. D. Goldstine (Eds.) Self-Organizing Systems, Spartan Books Washington DC, pp.93–106.

Bremermann, H. J. (1967) *Quantitative Aspects of Goal-Seeking Self-Organizing Systems.* Progress in Theoretical Biology, Vol.1, Academic Press, New York,

pp.59–77.

Bremermann, H. J. (1968) *Numerical Optimization Procedures Derived from Biological Evolution Processes.* H. L. Oestreicher, and D R. Moore (Eds.), Cybernetic Problems in Bionics, Gordon & Breach, New York, pp.543–562.

Bremermann, H. J. (1973) *On the Dynamics and Trajectories of Evolution Processes.* A Locker (Ed.) Biogenesis, Evolution, Homeostasis, Springer-Verlag, pp.29–37.

Bremermann, H. J., and M. Rogson (1964) *An Evolution-Type Search Method for Convex Sets.* ONR Technical Report, Contracts 222(85) and 3657(58), Berkeley.

Bremermann, H. J., M. Rogson, and S. Salaff (1965) *Search by Evolution.* M. Maxfield, A. Callahan, and L. J. Fogel (Eds.), Biophysics and Cybernetic Systems, Spartan Books, Washington DC, pp.157–167.

Bremermann, H. J., M. Rogson, and S. Salaff (1966) *Global Properties of Evolution Processes.* H. H. Pattee, E A Edlsack, L. Fein, and A. B. Callahan (Eds.), Natural Automata and Useful Simulations, Spartan Books, Washington DC, pp.3–41.

Caruana, R. A., and J. D. Schaffer (1988) *Representation and Hidden Bias: Gray vs. Binary Coding for Genetic Algorithms.* S. Forrest (Ed.) Proceedings of 5th International Conference on Machine Learning, Morgan Kaufmann, San Mateo, CA, pp.12–14.

Campbell, D. T. (1960) *Blind Variation and Selective Survival as a General Strategy in Knowledge-Processes.* M. C. Yovits and S. Cameron (Eds.) Self-Organizing Systems, Pergamon Press, New York, pp.251–231.

Cannon, W. D. (1932) *The Wisdom of the Body.* Norton and Company, New York.

Chandrasekaran, B., and L. H. Recher (1974) *Artificial Intelligence – A Case for Agnosticism.* IEEE Trans. on System, Man and Cybernetics, Vol.SMC-4, No.1, pp.88–94.

Collins, T. D. (1997) *Using Software Visualization Technology to Help Evolutionary Algorithm Users Validate their Solutions.* T. Bäck (Ed.) Proceed-

ings of the 7th International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, pp.307–314.

Conrad, M. (1974) *Evolutionary Learning Circuits.* Journal of Theoretical Biology, Vol.46, pp.167–188.

Conrad, M. (1981) *Algorithmic Specification as a Technique for Computing with Informal Biological Models.* BioSystems, Vol.13, pp.303–320.

Conrad, M. (1984) *Microscopic-Macroscopic Interface in Biological Information Processing.* BioSystems, Vol.16, pp.345–363.

Conrad, M. (1985) *On Design Principles for a Molecular Computer.* Comm. of the ACM, Vol.28, No.5, pp.464–480.

Conrad, M. (1987) *Rapprochement of Artificial Intelligence and Dynamics.* European Journal of Operations Research, Vol.30, pp.280–290.

Conrad, M. (1990) *The Geometry of Evolution* BioSystems, Vol.24, pp.61–81.

Cooper, L. N. (1973) *A Possible Organization of Animal Memory and Learning.* Proceedings of the 24th Nobel Symposium on Collective Properties of Physical Systems, pp.252–264.

Davis, L. (1991) *Bit-Climbing, Representation Bias, and Test Suite Design.* R. Belew, and L. Booker (Eds.) Proceedings of the 4th International Conference of Genetic Algorithms. Morgan Kaufmann, San Mateo, CA, pp.18–23.

Davis, T. E., and J. C. Principe (1991) *A Simulated Annealing Like Convergence Theory for the Simple Genetic Algorithm.* R. Belew, and L. Booker (Eds.) Proceedings of the 4th International Conference of Genetic Algorithms. Morgan Kaufmann, San Mateo, CA, pp.174–181.

De Jong, K. A. (1975) *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* Ph.D theses, University of Michigan.

Derrida, B., and E. Gardner (1986) *Metastable states of a Spin Glass Chain at 0 Temperature.* Journal of Physique, 47, pp.959–965.

Derrida, B., E,Gardner, and A. Zippelius (1987) *An Exactly solvable Asymmetric Neural Network Model.* Europhysics Letter, Vol.4, No.2, pp.167–173.

Edelmann, G. (1987) *Neural Darwinism: The Theory of Neuronal Group Selection.* Basic Books, New York.

Eiben, A. E., E. H. L. Aarts, and K. M. van Hee (1991) *Global Convergence of Genetic Algorithms: On Infinite Markov Chain Analysis.* H. -P. Schwefel, and R. Männer (Eds.) Proceedings of 1st International Conference on Parallel Problem Solving from Nature, Lecture Notes in Computer Science, Vol.496, Springer-Verlag. pp.4–12.

Fogarty, T. C. (1989) *Varying the Probability of Mutation in Genetic Algorithm.* J. D. Schfer (Ed.) Proceedings of 3rd International Conference on Genetic Algorithms, Morgan Kaufmann San Mateo, CA, pp.104–109.

Fogel, L. J. (1962) *Autonomous Automata.* Industrial Research, Vol.4, pp.14–19.

Fogel, L. J. (1964) *On the Organization of Intellect.* Ph.D thesis, University of California.

Fogel, L. J., A. J. Owens, and M. J. Walsh (1966) *Artificial Intelligence through Simulated Evolution.* Wiley, New York.

Fogel, D. B. (1992) *Evolving Artificial Intelligence.* Ph.D thesis, University of California San Diego.

Fogel, D. B. (1994) *Asymptotic Convergence Properties of Genetic Algorithms and Evolutionary Programming: Analysis and Experiments.* Cybernetics and Systems, Vol.25, No.3, pp.389–407.

Fogel, D. B. (1995) *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press.

Fonseca, C. M., and P. J. Fleming (1995) *An Overview of Evolutionary Algorithms in Multiobjective Optimization.* Evolutionary Computation, Vol.3, No.1, MIT Press, pp.1–16.

Fontana, W., and P. Schuster (1987) *A Computer Model of Evolutionary Optimization.* Biophysical Chemistry Vol.26, pp.123–147.

Forrest, B. M., and D. J. Wallace (1995) *Storage Capacity and Learning in Ising-Spin Neural Networks.* Models of Neural Networks I (2nd updated edition), Physics of Neural Networks, Springer-Verlag.

Forrest, S., and M. Mitchell (1993) *Relative Building-Block Fitness and the Building-Block Hypothesis.* L. D. Whitley (Ed.) Foundations of Genetic Algorithms-2, Morgan Kaufmann San Mateo, CA, pp.109–126.

Fraser, A. S. (1957a) *Simulation of Genetic Systems by Automatic Digital Computer: I. Introduction.* Australian Journal of Biological Science, Vol.10, pp.484-491.

Fraser, A. S. (1957b) *Simulation of Genetic Systems by Automatic Digital Computer: II. Effects of Linkage on Rates of Advance under Selection.* Australian Journal of Biological Science, 10, pp.492-499.

Friedberg, R. M. (1958) *A Learning Machine: Part I* IBM Journal of Research and Development, Vol.2, pp.2–13.

Friedman, G. J. (1959) *Digital Simulation of an Evolutionary Process.* Biological Cybernetics, Vol.53, pp.1–9.

Gardner, E. (1988) *The Phase Space of Interactions in Neural Network Models.* Journal of Physics 21A, pp.257–270.

Geszti, T. (1990) *Physical Models of Neural Networks.* Word Scientific, Singapore.

Goldberg, D. E., and P. Segrest (1987) *Finite Markov Chain Analysis of Genetic Algorithms.* J. J. Grefenstette (Ed.) Proceedings of 2nd International Conference of Genetic Algorithms. Lawrence Erlbaum Associates, Hillsdale, NJ. pp.1–8.

Goldberg, D. E., and J. Richardson (1987) *Genetic Algorithms with Sharing for Multimodal function Optimization.* J. J. Grefenstette (Ed.) Proceedings of 2nd International Conference of Genetic Algorithms. Lawrence Erlbaum Associates, Hillsdale, NJ, pp.41-49.

Goldberg, D. E. (1989). *Genetic Algorithms: In Search, Optimization and Machine Learning.* Addison Wesley, Reading, Massachusetts.

Goldberg, D. E. (1990). *Real-Coded Genetic Algorithms, Virtual Alphabets, and Blocking.* Technical Report of University of Illinois at Urbana-Champaign No.90001.

Gruau, F., and D. Whitley (1993) *Adding Learning to the Cellular Development of Neural Networks: Evolution and the Baldwin Effect.* Evolutionary Computation, Vol.1, No.3, pp.213-233.

Hassoun, M. H. (Ed.) (1993) *Associative Neural Memories: Theory and Implementation*, Oxford University Press.

Hebb, D. O. (1949) *The Organization of Behavior, Wiley.*

Hertz, J. A., G. Grinstein, and S. A. Solla (1987). *Irreversible Spin Glasses and Neural Networks.* L. N. van Hemmen and I. Morgenstern (Eds.) Heidelberg Colloquium on Glassy Dynamics. Lecture Notes in Physics, No.275, Springer-Verlag, pp.538–546.

Hillis, W. D. (1991) *Co-Evolving Parasites Improve Simulated Evolution as an Optimization Procedure.* C. G. Langton, C. Taylar, J. D. Farmer, and S. Rasmussen (Eds.) Artificial Life II. SFI Studies in the Science of Complexity, Vol. X, Addison-Wesley, pp.295–312.

Hinton, G. E., and S. J. Nowlan (1987) *How Learning Can Guide Evolution* Complex System, Vol.1, pp.495-502.

Holland, J. (1968) *Hierarchical Descriptions of Universal Spaces and Adaptive Systems.* Technical Report ORA Projects 01252 and 08226. University of Michigan, Department of Computer and Communication Sciences.

Holland, J. (1975) *Adaptation in Natural and Artificial Systems.* The University of Michigan Press.

Hollstien, R. B. (1971) *Artificial Genetic Adaptation in Computer Control Systems.* Ph.D dissertation, University of Michigan. Dissertation Abstracts International, 32(3), 1510B. (University Microfilms, No.71-23, 773).

Hopfield, J. J. (1982) *Neural Networks and Physical Systems with Emergent Collective Computational Abilities.* Proceedings of the National Academy of Sciences, USA, 79, pp.2554–2558.

Imada, A., and K. Araki (1995a) *Genetic Algorithm Enlarges the Capacity of Associative Memory.* T. Bäck (Ed.) L. J. Eshelman (Ed.) Proceedings of the 6th International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, pp.413–420.

Imada, A., and K. Araki (1995b) *Mutually Connected Neural Network Can Learn Some Patterns by Means of GA.* Proceedings of World Congress on Neural Networks, Vol.1, pp.803–806.

Imada, A., and K. Araki (1996a) *Basin of Attraction of Associative Memory as it is Evolved by a Genetic Algorithm.* Proceedings of the 2nd Online Workshop on Evolutionary Computation, pp.41–44.

Imada, A., and K. Araki (1996b) *Evolution of Associative Memory with Environmental Change.* Poster presented at An Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems.

Imada, A., and K. Araki (1996c) *Lamarckian Evolution of Associative Memory.* Proceedings of IEEE International Conference on Evolutionary Computation, pp.676–680.

Imada, A., and K. Araki (1996d) *Basin of Attraction of Associative Memory as it Evolves from Random Weights.* Proceedings of the 1st Asia-Pacific Conference on Simulated Evolution and Learning, pp.271–278.

Imada, A., and K. Araki (1997a) *Hopfield Model of Associative Memory as a Test Function of Evolutionary Computations.* The 1st International Workshop on Frontiers in Evolutionary Algorithms, Proceedings of Joint Conference of Computer Science, Vol.1, pp.180–183.

Imada, A., and K. Araki (1997b) *Evolved Asymmetry and Dilution of Random Synaptic Weights in Hopfield Network Turn a Spin-glass Phase into Associative Memory.* The 2nd International Conference on Computational Intelligence and Neuroscience, Proceedings of Joint Conference of Computer Science, Vol.2, pp.223–226.

Imada, A., and K. Araki (1997c) *The Baldwin Effect on the Evolution of Associative Memory.* Proceedings of the 3rd International Conference on Artificial Neural Networks and Genetic Algorithms, Springer-Verlag, pp.354–358.

Imada, A., and K. Araki (1997d) *Searching Real-Valued Synaptic Weights of Hopfield's Associative Memory using Evolutionary Programming.* Proceedings of the 6th Annual Conference on Evolutionary Programming, Springer-Verlag, Lecture Notes in Computer Science, No.1213, pp.13–22.

Imada, A., and K. Araki (1997e) *Application of an Evolution Strategy to the Hopfield Model of Associative Memory.* Proceedings of the IEEE International Conference on Evolutionary Computation, pp.679–683.

Imada, A., and K. Araki (1997f) *Random Perturbations to Hebbian Synapses of Associative Memory using a Genetic Algorithm.* J. Mira, R. Moreno-Díaz, and J. Cabestany (Eds.) Proceedings of International Work-Conference on Artificial and Natural Neural Networks, Springer-Verlag, Lecture Notes in Computer Science No.1240, pp.398–407.

Imada, A., and K. Araki (1997g) *Evolution of Hopfield Model of Associative Memory by the Breeder Genetic Algorithm.* T. Bäck (Ed.) Proceedings of the 7th International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, pp.784–791.

Imada, A., and K. Araki (1997h) *Evolution of Random Synaptic Weights of the Hopfield Associative Memory: How Chaotic Trajectories Turn into Fixed Point Attractors?* Proceedings of the 4th International Conference on Neural Information Processing and Intelligent Information Systems, Vol. 1, Springer-Verlag, pp.452–455.

Imada, A., and K. Araki (1997i) *Evolution of Associative Memory using Diploid Chromosomes.* The 10th Australian Joint Conference on Artificial Intelligence, Proceedings of the Workshop on Evolutionary Computation, pp.24–36.

Imada, A., and K. Araki (1998a) *What does the Landscape of a Hopfield Associative Memory Look Like?* Proceedings of the 7th Annual Conference on Evolutionary Programming, Springer-Verlag, Lecture Notes in Computer Science, pp.647–656.

Imada, A., and K. Araki (1998b) *Explorations of Fitness Landscapes of a Hopfield Associative Memory with Random and Evolutionary Walks.* Proceedings of the 2nd International Conference on Knowledge-based Intelligent Electronic Systems, Vol.2, pp.364–369.

Imada, A., and K. Araki (1998c) *Does Diploidy Affect Evolutions of Hopfield Associative Memory?* Proceedings of the 7th Turkish Symposium on Artificial Intelligence and Neural Networks, pp.62–70.

Imada, A., and K. Araki (1998d) *Some Statistical Analyses of Fitness Landscape Created by the Hopfield Model of Associative Memory.* Proceedings of the International Technical Conference on Circuits/Systems, Computer and Communication, Vol.2, pp.1031–1034.

Imada, A., and K. Araki (1998e) *How Real-valued Random Synapses Evolve toward Symmetry with Diploid Chromosomes?* Proceedings of the International ICSC/IFAC Symposium on Neural Computation, pp.164–169.

Imada, A., and K. Araki (1998f) *Can a Niching Method Locate Multiple Attractors Embedded in the Hopfield Network?* Proceedings of Asia-Pacific Conference on Simulated Evolution and Learning, Springer-Verlag, Lecture Notes in Artificial Intelligence, (to appear).

Jackson, P. C. (1974) *Introduction to Artificial Intelligence.* Dover Publications (reprint), New York.

Jones, T. C. (1995) *Evolutionary Algorithms, Fitness Landscapes and Search.* Ph.D thesis, University of New Mexico.

Kamphner, R. P., and M. Conrad (1983) *Computational Modeling of Evolutionary Learning Processes in the Brain.* Bulliten of Mathematical Biology, Vol.45, No.6, pp.931–968.

Kauffman, S. A., and S. Levin (1987) *Towards a General Theory of Adaptive Walks on Rugged Landscapes.* Journal of Theoretical Biology Vol.128, pp.11–45.

Kauffman, S. A. (1993) *The Origin of Order: Self-organization and Selection in Evolution.* Oxford University Press.

Kirkpatrick, S., and D. Sherrington (1978) *Infinite-ranged Models of Spinglasses.* Physical Review, Vol.B17, No.11, pp.4384–4403.

Kohonen, T., and M. Ruohonen (1973) *Representation of Associated Data by Matrix Operators.* IEEE Trans. Computers, Vol.C-22, No.7, pp.701–702.

Komlós, J., and R. Paturi (1988) *Convergence Results in an Associative Memory Model.* Neural Networks, Vol.1, pp.239–250.

Krauth, W., J.-P. Nadal, and M.Mezard (1988) *The Roles of Stability and Symmetry in the Dynamics of Neural Networks.* Journal of Physics A: Math.

Gen. **21**, pp.2995–3011.

Lenat, D. B. (1983) *The Role of Heuristic in Learning by Discovery: Three Case Studies.* R. S. Michalske, J. G. Carbonell, and T. M. Mitchell (Eds.) Machine Learning, Tiga Publishing, Palo Alto, CA, pp.243–306.

Lichtfuss, H. J. (1965) *Evolution eines Rohrkrümmers.* Diplomarbeit, Technische Universität Berlin.

Lipsitch, M. (1993) *Adaptation on Rugged Landscapes Generated of Neighboring Genes.* R. Belew, and L. Booker (Eds.) Proceedings of the 4th International Conference of Genetic Algorithms. Morgan Kaufmann, San Mateo, CA, pp.143–150.

Nadal, J. P., G. Toulouse, J. P. Changeux, and S. Dahaene (1986) *Networks of Formal Neurons and Memory Palimpsest.* Europhysics Letter, Vol.1, No.10, pp.535–542.

Nix, A., and M. D. Vose (1991) *Modeling Genetic Algorithms with Markov Chains.* Annals of Mathematics and Artificial Intelligence, Vol.5, pp.79–88.

Macken, C., P. Hagan, and A. Perelson (1991) *Evolutionary Walks on Rugged Landscapes.* SIAM Journal of Applied Mathematics Vol.51, pp.799–827.

Mahfoud, S. W. (1992) *A Comparison of Parallel and Sequential Niching Methods.* Proceedings of the 2nd Parallel Problem Solving from Nature, pp.27–36.

Mahfoud, S. W. (1995) *A Comparison of Parallel and Sequential Niching Methods.* L. J. Eshelman (Ed.) Proceedings of the 6th International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, pp.136–143.

Manderick, B., M. de Weger, and P. Spiessens (1993) *The GA and the Structure of the Fitness Landscape.* R. Belew, and L. Booker (Eds.) Proceedings of the 4th International Conference of Genetic Algorithms. Morgan Kaufmann, San Mateo, CA, pp.128–135.

Martin, F. G., and C. C. Cockerham (1960) *High Speed Selection Studies.* O. Kempthorne (Ed.) Biometrical Genetics, Pergamon Press, London, pp.35–45.

Maynard Smith, J. (1970) *Natural Selection and the Concept of a Protein Space.* Nature, Vol.225, pp.563–564.

McEliece, R. J., E. C. Posner, E. R. Rodemick, and S. S. Venkatesh (1987) *The Capacity of the Hopfield Associative Memory.* IEEE Trans. Information Theory, Vol.IT-33, pp.461–482.

Michalewicz, Z. (1996) *Genetic Algorithms + Data Structures = Evolution Programs.* 3rd edition, Springer-Verlag.

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

Michie, D. (1970) *Future for Integrated Cognitive Systems.* Nature, Vol.228, pp.717-722.

Mühlenbein, H., and D. Schlierkamp-Voosen (1996) *Predictive Models for the Breeder Genetic Algorithm I. Continuous Parameter Optimization.* Evolutionary Computation, Vol.1, No.4, MIT Press, pp.25–49.

Parisi, G. (1986) *Asymmetric Neural Networks and the Process of Learning.* Journal of Physics, Vol.A19, pp.L675–L680.

Parisi, D., and S. Nolfi (1995) *How Learning can Influence Evolution within a Non-Lamarckian Framework.* Plastic Individuals in Evolving Populations. Santa Fe Institute Series, Addison-Wesley.

Personnaz, L., I. Guyon, and G. Dreyfus (1986) *Collective Computational Properties of Neural Networks: New Learning Mechanisms.* Physical Review, Vol.A34, No.5, pp.4217–4227.

Rada, R. (1981) *Evolution and Gradualness.* Biosystems, Vol.14, pp.211–218.

Rechenberg, I. (1965) *Cybernetic Solution Path of an Experimental Problem.* Ministry of Aviation, Royal Aircraft Establishment, Library Translation No.1122 (UK).

Rechenberg, I. (1973) *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution.* Frommann-Holzboog, Stuttgart.

Rosenberg, R. S. (1967) *Simulation off Genetic Populations with biochemical Properties.* Doctoral dissertation. University of Michigan, Dissertation Abstracts International, 28(7), 2732B (University Microfilms No.67-17836)

Rosenblatt, F. (1962) *Principles of Neurodynamics.* Spartan, New York.

Rudolph, G. (1992) *Parallel Approaches to Stochastic Global Optimization.* W. Joosen, and E, Milgrom (Eds.) Parallel Computing: From Theory to Sound Practice, Proceedings of the European Workshop on Parallel Computing, IOS Press, Amsterdam, pp.256–267.

Rudolph, G. (1994) *Convergence Analysis of Canonical Genetic Algorithms.* IEEE Trans. on Neural Networks, Vol.5, No.1, pp.96–101.

Sammon, J. W. (1969) *A Nonlinear Mapping for Data Structure Analysis.* IEEE Trans.
on Computers, Vol.C-18, No.5, pp.401–408.

Schewefel, H.-P. (1965) *Kybernetische Evolution als Strategie der Experimentellen Forschung in der Strömungstechnik.* Diplomarbeit, Technische Universität Berlin.

Shine, W. B., and C. F. Eick (1997) *Visualizing the Evolution of Genetic Algorithm Search Processes.* Proceedings of the IEEE International Conference on Evolutionary Computation, pp.367–372.

Sompolinsky, H. (1986) *Neural Network with Non-linear Synapses and Static Noise.* Physical. Review, A34, pp.2571-2574.

Srinivas, N., and K. Deb (1994) *Multiobjective Optimization using Nondominated Sorting in Genetic Algorithms.* Evolutionary Computation, Vol.2, No.3, MIT Press, pp.221–248.

Surry, P. D., and N. J. Radcliffe (1996) *Inoculation to Initialize Evolutionary Search.* Evolutionary Computing: Proceedings of Artificial Intelligence and Simulation of Behavior (AISB) Workshop, Lecture Notes in Computer Science, Vol.1143, Springer-Verlag, pp.269–285.

Syswerda, G. (1989) *Uniform Crossover in Genetic Algorithms.* J. D. Schfer (Ed.) Proceedings of 3rd International Conference on Genetic Algorithms, Morgan Kaufmann San Mateo, CA, pp.2–9.

Toulouse, G., S. Dehaene, and J. P. Changeux (1986) *Spinglass Model of Learning by Selection.* Proceedings National Academy of Science (USA), Vol.83, pp.1695–1698.

Turing, A. M. (1950) *Computing Machinery and Intelligence.* Mind, Vol.LIX, No.59, pp.433–460.

Vassilev, V. K. (1997) *An Information Measure of Landscapes.* T. Bäck (Ed.) Proceedings of the 7th International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, pp.49–56.

Voigt, H.-M., H. Mühlenbein, and D. Cvetcorić (1995) *Fuzzy Recombination for the Breeder Genetic Algorithm.* S. Forrest (Ed.) Proceedings of 5th International Conference on Genetic Algorithms, pp.104-111.

Vose, M. D., and G. E. Liepins (1991) *Punctuated Equilibria in Genetic Search.* Complex systems, Vol.5, pp.31-44.

Vose, M. D. (1992) *Modeling Simple Genetic Algorithms.* D. Whitley (Ed.) Foundations of Genetic Algorithms 2, Morgan Kaufmann San Mateo, CA, pp.63–73.

Vose, M. D., and A. H. Wright (1993) *Simple Genetic Algorithms with Linear Fitness.* Evolutionary Computation, Vol.2, No.4, MIT Press, pp.347–368.

Vose, M. D. (1996) *Logarithmic Convergence of Random Heuristic Search.* Evolutionary Computation, Vol.4, No.4, MIT Press, pp.395-404.

Weinberg, R, (1970) *Computer Simulation of a Living Cell.* Doctoral dissertation. University of Michigan, Dissertation Abstracts International, Vol.31, No.9, 5312B (University Microfilms, No.71-4766).

Weinberger, E. D. (1990) *Correlated and Uncorrelated Fitness Landscapes and How to Tell the Difference.* Biological Cybernetics, Vol.63, pp.325–336.

Whitley, W., K. Mathias, S. Rana, and J. Dzubera (1995) *Building Better Test Functions.* L. J. Eshelman (Ed.) Proceedings of the 6th International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, pp.239–246.

Wright, A. H. (1991) *Genetic Algorithms for Real Parameter Optimization.* G. Rawling (Ed.) Foundations of Genetic Algorithms. First Workshop on the Foundations of Genetic Algorithms and Classifier Systems, Morgan Kaffmann San Mateo, CA, pp.205–218.

Wright, S. (1932) *The Roles of Mutation, Inbreeding, Crossbreeding and Selection in Evolution.* L. J. Eshelman (Ed.) Proceedings of the 6th International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, CA, pp.356–366.

Zhigljavsky (1992) *Theory of Global Random Search: Mathematics and Its Applications.* Kluwer, Dordrecht.