Complex Systems **8** (1994) 295–309

Pattern Search Using Genetic Algorithms and a Neural Network Model

Shigetoshi Nara
Department of Electrical and Electronic Engineering,
Faculty of Engineering, Okayama University,
Tsuchimanaka 3-1-1, Okayama 700, Japan

Wolfgang Banzhaf*
Central Research Laboratory, Mitsubishi Electric Corporation,
Tsukaguchi Honmachi 8-1-1, Amagasaki,
Hyogo 661, Japan

Abstract. An information processing task that generates combinatorial explosion and program complexity when treated by a serial algorithm is investigated using both genetic algorithms and a neural network model. The task in question is to find a target memory from a set of stored entries in the form of "attractors" in a high-dimensional state space. The representation of entries in the memory is distributed ("an auto associative neural network" in this paper) and the problem is to find an attractor under a given access information where the uniqueness or even existence of a solution is not always guaranteed (an ill-posed problem). The genetic algorithm is used for generating a search orbit to search effectively for a state that satisfies the access condition and belongs to the target attractor basin in the state space. The neural network is used to retrieve the corresponding entry from the network. The results of our computer simulations indicate that the present method is superior to a search method that uses a Markovian random walk in state space. Our techniques may prove useful in the realization of flexible and adaptive information processing, since pattern search in a high-dimensional state space is common in various kinds of parallel information processing.

1. Introduction

Although great progress has been made with modern LSI computers based on the von Neumann type of serial algorithms, there has been growing interest

*Current address: Department of Computer Science, Dortmund University and Informatics Center Dortmund, Joseph-von-Fraunhofer-Straße 20, 44227 Dortmund, Germany

in other types of information processing such as parallel or flexible processing typically observed in biological systems. Serial algorithms run into problems with (1) combinatorial explosion and (2) program complexity in realizing flexible functions. Generally speaking, the main problem of flexible information processing is the fact that there are too many degrees of freedom for sequential control. One way to improve flexibility in information processing is the application of nonlinear dynamics, including chaos [1]. Nonlinear dynamics has been characterized as "emerging complex behavior generation" [2–4]. The great variety of possible dynamical structures in a chaotic system suggests to us that an application to complex information processing may avoid combinatorial explosion and/or control complexity [5–7].

In order to make the problem clearer and more practical, we shall concentrate on a particular information processing task. However, if we restrict the information processing context too severely, it will lead to a less than interesting result because of an ad hoc solution. The trade-off is that restriction usually allows easier modeling and formulation. Therefore, in setting the context, one attempts to maintain the universality of the application to a wide field of processing. We propose pattern search in a high-dimensional state space [8, 9, 12], in which a set of patterns are stored in the form of attractors [10]. An attractor is a state surrounded by a domain known as a basin, in which all sequences converge to the attractor over time. In more detail, the task is to retrieve one or more of the stored patterns if existent, under the condition that the given access information is not complete or sufficient to reach the target pattern directly. This is an ill-posed problem in which the uniqueness or even the existence of a solution is not guaranteed. It is a desirable function of the information processor to solve this complex problem efficiently, and there are two important points with respect to the search process in the state space [9].

1. How can the processor quickly and efficiently find the target basin from ambiguous access information?

2. If there is no stored pattern that satisfies the access information, can the search processor generate information close to the requested pattern?

Several methods are candidates for the envisioned function: (1) simulated annealing (for an example of a rather sophisticated one, see [11]); (2) chaotic dynamics [7, 8, 9, 12]; (3) genetic algorithms (i.e., evolutionary strategies); (4) neural networks; (5) cellular automata [13, 14]; and (6) mode-competitive nonlinear dynamics [15]. In this paper, we employ a method that uses both a neural network and a genetic algorithm. The former has a long history and became an especially active field during the past decade as a powerful method for parallel processing [16, 17]. Associated memory and classification of highly complicated signals [18] are two prominent applications of neural networks. Genetic algorithms have also attracted much attention in recent years, especially in the field of optimization in high dimensions [19–23].

2. The neural network model

To begin with, let us start with a description of the state space. Without loss of generality, we employ a space of states (image patterns) that consist of 20×20 pixels with one neuron corresponding to each pixel. Neural activity is restricted to two states, $+1$ (firing state) and -1 (non-firing state). Each pattern is specified by a 400-dimensional state vector $\mathbf{v} = (v_1, v_2, \ldots, v_{400})$, and the state space consists of all possible 400-bit patterns (the vertices of a hypercube in 400 dimensions). In the present model, each neuron is assumed to be coupled with every other neuron, and a pair's coupling strength is represented by a synaptic connection matrix, the dimension of which is 400×400. In the state space, we store 30 patterns as "attractors."

We employ

$$v_i(t+1) = \Theta\left(\sum_{j=1}^{400} T_{ij} v_j(t)\right) \tag{1}$$

as the time development rule of this neural network model, where each neuron is represented by a discrete variable $v_i = \pm 1$, and T_{ij} is the synaptic connection matrix. The mapping Θ is a step function. With respect to the 30 patterns used for actual simulations, the overlap between them is relatively large, as can be observed from Figure 1. Thus, retrieval performance is bad if an autocorrelation connection matrix is used. We therefore use orthogonalization of patterns [24, 25] by introducing adjoint state vectors $\mathbf{v}_\alpha^\dagger (\alpha = 1, 2, \ldots, 30)$ defined by

$$\mathbf{v}_\alpha^\dagger \bullet \mathbf{v}_\beta = \delta_{\alpha\beta}, \quad \mathbf{v}_\alpha^\dagger = \sum_\gamma a_{\alpha\gamma} \mathbf{v}_\gamma, \quad a = o^{-1}, \quad o_{\alpha\beta} = \mathbf{v}_\alpha \bullet \mathbf{v}_\beta \tag{2}$$

where a is the inverse matrix of a 30×30 correlation matrix, the elements of which are defined by the scalar products between two pattern vectors. The synaptic connection matrix is now defined as

$$\top = \sum_{\alpha=1}^{30} \mathbf{v}_\alpha \otimes \mathbf{v}_\alpha^\dagger, \tag{3}$$

also known as the pseudo-inverse of the autocorrelation matrix. The matrix T_{ij} is symmetric, so the energy function $E = -\Sigma_{ij} v_i T_{ij} v_j$ is a Lyapunov function for this system. Since the number of stored patterns is much smaller than the total number of neurons, the resulting basins of attraction in state space tend to be quite large, and it is expected that much information is distributed among these basins. We call this information "seeds" for the access. The "seed" contains many partial features of target patterns (attractors). Once some seed containing the partial feature of an attractor is given to the neural network, it is entrained into the attractors, as if one retrieved the specified feature by glancing at the shape of a face. However, in the present artificial neural network model, it should be noted that efficiency of retrieval strongly depends on the topological structure of the basins in the state space.

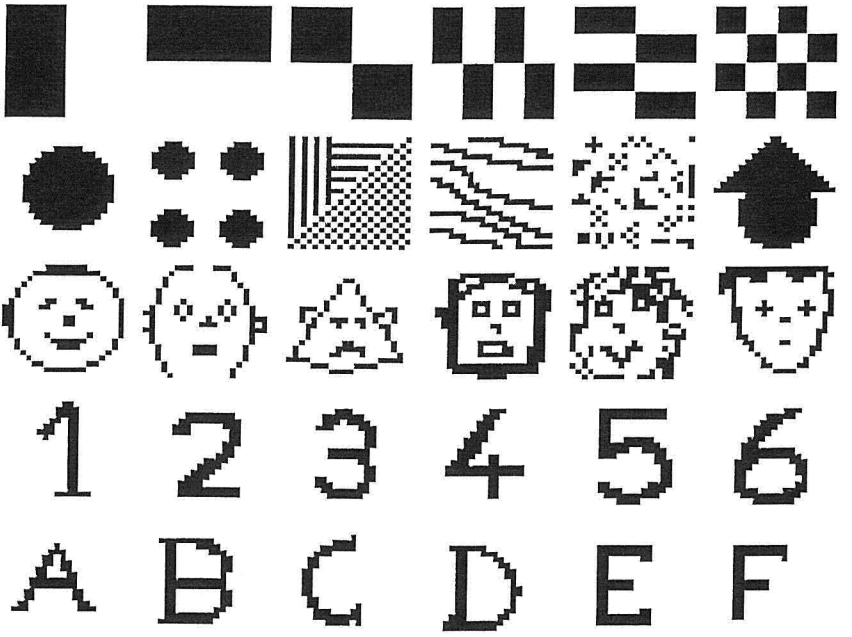

Figure 1: Thirty memory patterns consisting of 20×20 pixels (neurons). We assume the fourth pattern among the face patterns (third row in the figure) to be the target of the memory search.

By introducing the state space of the neural network, our ill-posed problem of pattern search becomes more practical. Suppose there is an accessor who wants to find a face from a data base composed of many different faces using ambiguous access information such as the feature of eye shape. In our simulation, we search the state space for face patterns that have a specific eye-shape feature. Face patterns consist of 400 bits of information, but the eye-shape feature consists of only 40 bits. We gave eye-shape data (40 bits) of the fourth face pattern in the third row of Figure 1 as access information. The search algorithm is described in the next section.

3. The genetic algorithm

In this section, we consider the application of a genetic algorithm [26] to generate an effective search orbit in the state space.

Let us define a gene string as consisting of 401 elements, where the first element x_0 is a header containing the evaluation value of the gene and the other 400 elements x_i are a state vector. We introduce a certain number of these genes and taken together they form a gene pool. Here, as in most GAs, two different kinds of operations are performed: "mutations" and "recombi-

nations." The former are operations that randomly change a certain number of components of a gene. The latter are operations in which two or more genes interact to generate a new gene string composed of a certain number of elements from each parent string. In both operations, the elements are chosen using random number generators.

In the following, we give a brief description of the possible operations.

3.1 Mutation

Mutations in the present paper consist of the following three operations. They are:

1. **Single flip:** choose a gene; generate a random number $1 \leq n_1 \leq 400$; flip component n_1.
2. **Complement:** choose a gene; generate two random numbers $1 \leq n_1, n_2 \leq 400$; flip all components between n_1 and n_2.
3. **Inversion:** choose a gene; generate two random numbers $1 \leq n_1, n_2 \leq 400$; invert the order of components between n_1 and n_2.

3.2 Recombination (crossover)

Choose two genes, say, genes, α and β, and generate two random numbers such that $1 \leq n_1, n_2 \leq 400$. Take $n_3 = \text{Min}[n_1 + n_2, 400]$ and exchange the components of genes α and β between n_1 and n_3.

These operations generate strings that are then selected as follows.

3.3 Selection criteria

Each trial string, the result of an application of one of the above operations, is put into the neural network as an initial pattern. The neural network is updated according to equation (1) until the output converges on a pattern. Then the converged pattern is evaluated in comparison with the given access information and subsequently replaces the predecessors from which it was generated if it possesses higher or equal quality. It is discarded if its quality is lower than that of its predecessors. Since every string carries its own evaluation value, this is a totally local selection method. It has been successfully applied in the case of the travelling salesperson problem [26]. Figure 2 shows the overall process.

4. Results of the simulations

A simulation was carried out that employed nine genes in the gene pool and used both mutation and recombination processes as discussed in the previous section. A comparison was done between random search and genetic search. In both cases, we started with randomly chosen patterns. Figure 3 shows intermediate patterns using the GA. Note that all the patterns correspond

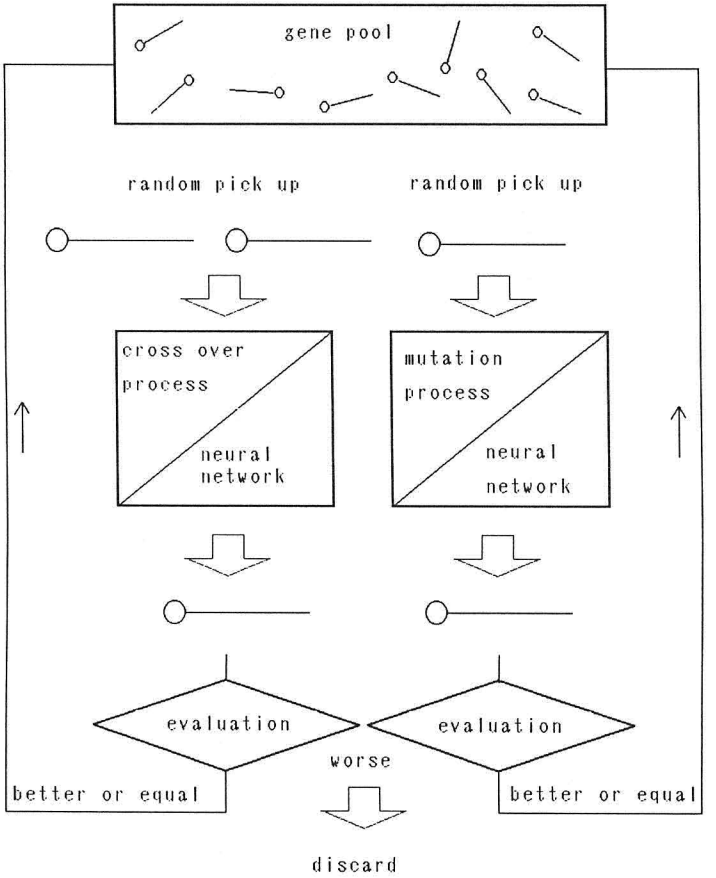

Figure 2: Overall algorithm of the pattern search simulation. Note the following. [1] In the mutation process, when we apply one of the mutation operations defined in section 3, the sequence of operation in the repeating process is taken as cyclic, i.e. [→ mutation.1 → mutation.2 → mutation.3 →]. [2] The generated pattern (gene) is given to the neural network as an initial condition and the recurrent updating of firing patterns is done until it converges. [3] If any converged pattern possesses the target feature, the search process stops and is regarded as successful; otherwise, the evaluation value is compared with the predecessor (gene). During the recombination process, the evaluation procedure is the same as the mutation process.

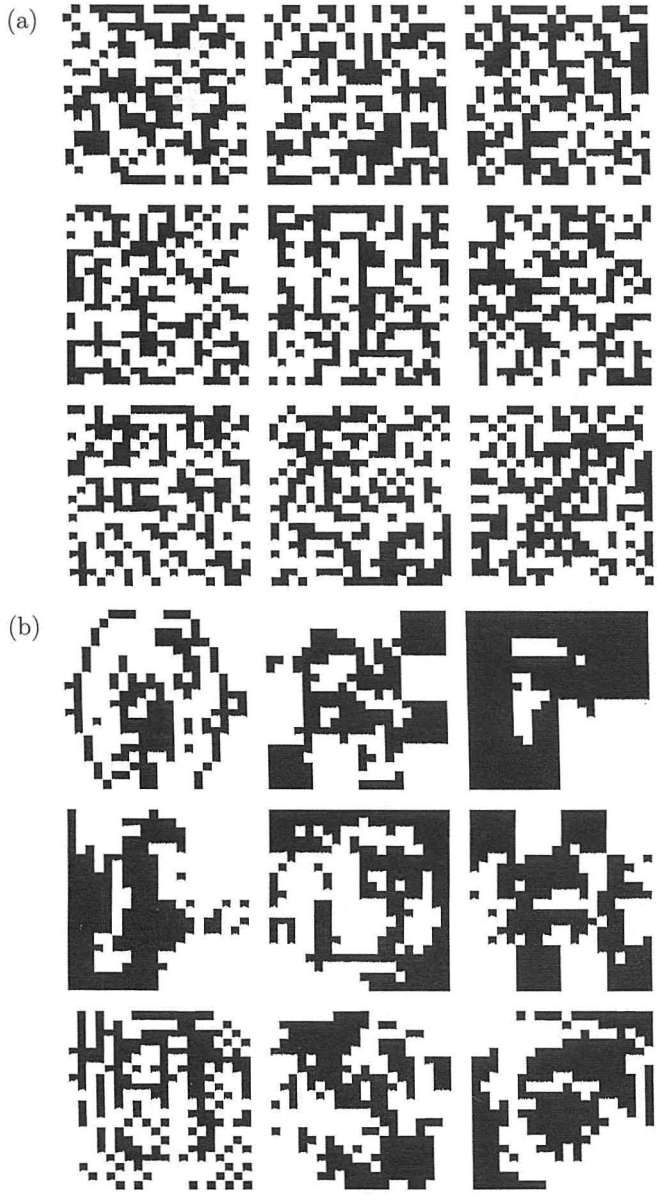

Figure 3: (a) Initial random patterns given to nine genes; (b) Patterns after the first generation.

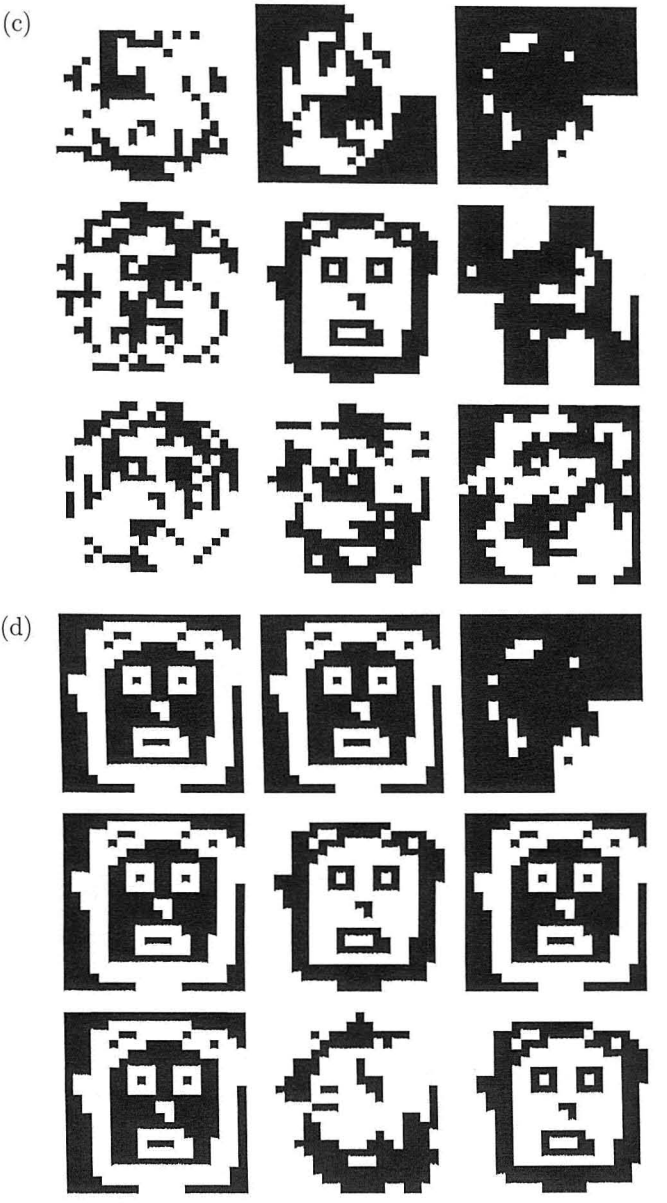

Figure 3: (c) Patterns after the 42nd generation (note that the pattern in the left corner is a superimposed pattern between the shape of the first face and the eye, nose, and mouth of the fourth face in Figure 1; (d) patterns after the 140th generation; most patterns have converged to the target pattern.

to local minima of the energy landscape since the trial patterns created by GA operations are mapped onto local minima of energy by the neural network.

In Figure 3, there appears a pattern worth noting because it is not a stored pattern. It indicates that one can find spurious attractors using genetic algorithms that possess high value or quality.

Now let us turn to an evaluation of the search performance. Define $f(N)$ as the ratio between the number of successful trials and the total number of trials. One trial consists of a search process involving nine genes with random patterns as the initial states. A trial is successful if the access information is satisfied within the required number N of iterations. If there is no appearance of the target pattern within N iterations, we regard the trial as unsuccessful. The same quantity is calculated for random search and shown for comparison.

In the case of random search, it is easy to calculate $f(N)$. Note that it is possible to define the one step success probability p in random search. It is simply the ratio of the basin volume to the total volume of the state space since each step can be regarded as completely random. Thus the process is considered to be Markovian and there is no correlation or memory effect between two succeeding steps. Therefore, $f(N)$ has the form

$$f(N) = \sum_{r=0}^{N-1} (1-p)^r p = p\frac{1-(1-p)^N}{1-(1-p)} = 1-(1-p)^N \tag{4}$$

In order to understand the dependence of $f(N)$ on N in more detail, one regards N as a continuous variable. Then one differentiation yields

$$\begin{aligned}\frac{d}{dN}f(N) &= (1-p)^N \log\left(\frac{1}{1-p}\right) \\ &= \exp\left(-N\log\left(\frac{1}{1-p}\right)\right)\log\left(\frac{1}{1-p}\right).\end{aligned} \tag{5}$$

This indicates that $df(N)/dN$ depends on N with exponential damping in random search. We show the results of our simulation in Figure 4 and Figure 5, both in the case of random search and also genetic search, where differentiation was done numerically over discrete, finite intervals. The results for random search indicate exponential damping, which is quite plausible as noted above. On the other hand, as shown in Figure 5, genetic search indicates a characteristic distribution of differential success rates as a function of N. This tendency is not accidental or due to statistical fluctuation. The same dependency as shown in Figure 6, obtained for the average success rate of 1000 samples, was obtained by averaging over 10000 samples. It is an interesting question why the distribution function indicates a very different behavior in genetic search as compared to a random search.

Genetic search is superior to random search by almost an order of magnitude as indicated in Figure 7. The same simulation was done for different numbers of genes in the gene pool. Results are shown in Figure 8 and a considerable improvement can be observed if we increase the population size M. Note that in order to obtain Figure 8, we divided the total number of

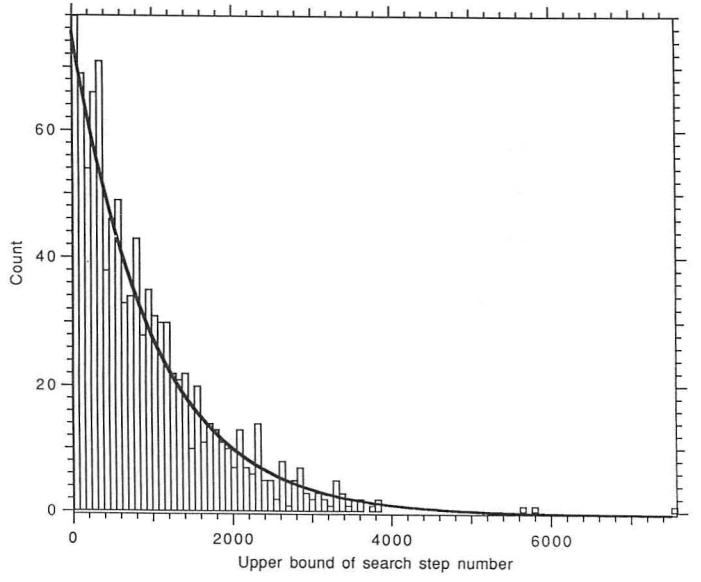

Figure 4: Differential success rate versus upper limit of search step in random search (1000 samples).

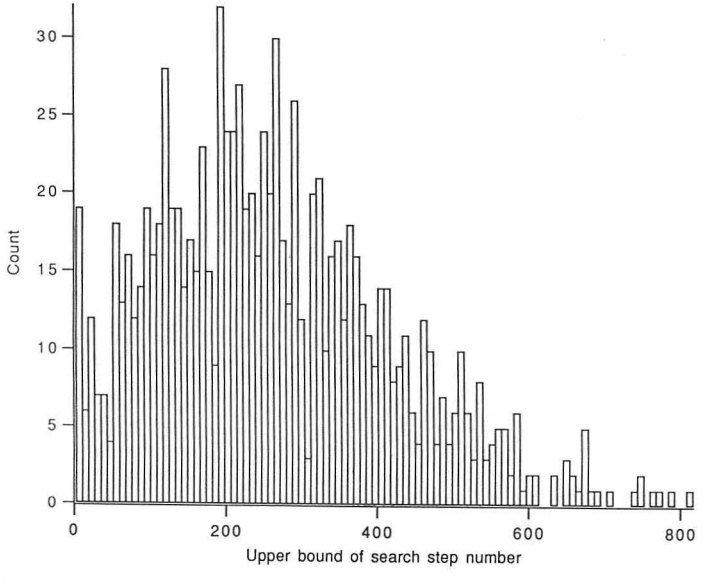

Figure 5: Differential success rate versus upper limit of search step in genetic search (1000 samples).

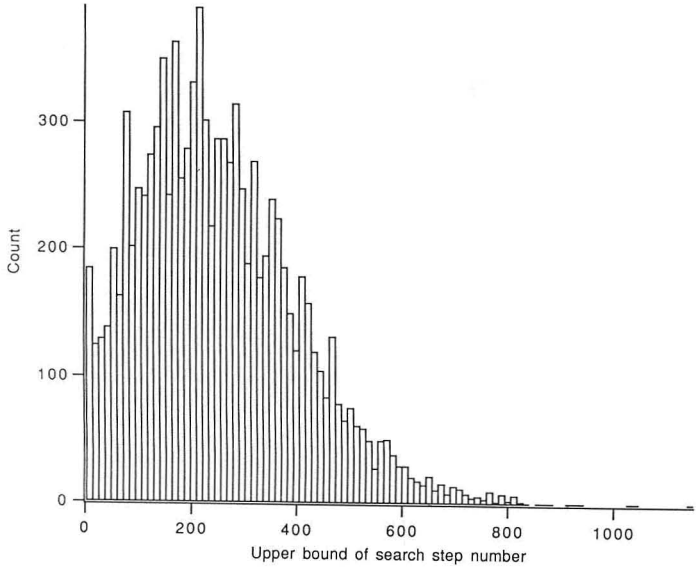

Figure 6: Differential success rate versus upper limit of search step in genetic search (1000 samples).

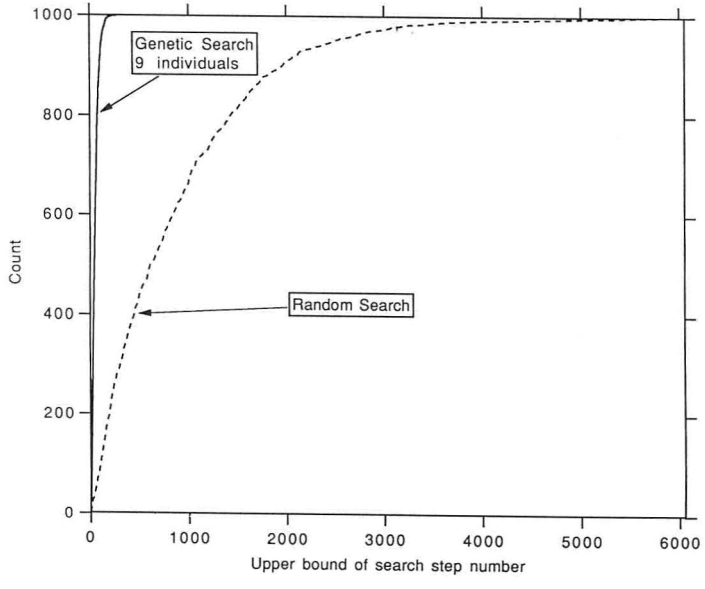

Figure 7: Comparison of success rate between random search and genetic search.

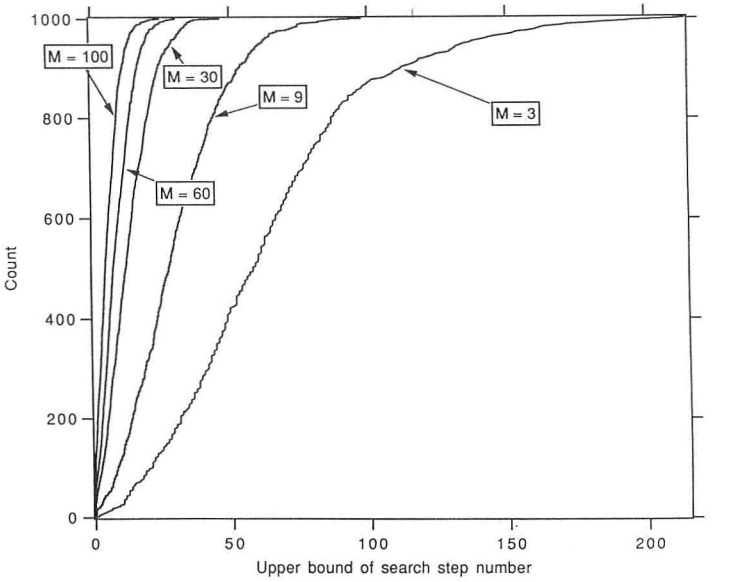

Figure 8: Comparison of success rates for different numbers of genes in the gene pool.

trials by the size of the population. A further improvement can be obtained by carefully adjusting the recombination frequency (see Figure 9).

It is interesting to speculate why the search using GA is more efficient than search using a simple random walk. We do not yet have a definitive answer, but the following explanations seem plausible:

1. A genetic algorithm can, with high probability, detect spurious attractors having high value or quality, even if the basin is relatively small compared to the basins of the stored patterns.

2. There is a memory effect in the development of genes produced by genetic operations, whereas in a random walk, there is no memory effect between the updating of patterns in state space.

Concluding Remarks

1. Genetic search algorithms are superior to simple random search (Markovian random walk) by almost an order of magnitude.

2. The utilization of both mutation and recombination operations improves search performance. Optimization of mutual frequency of operations is beneficial.

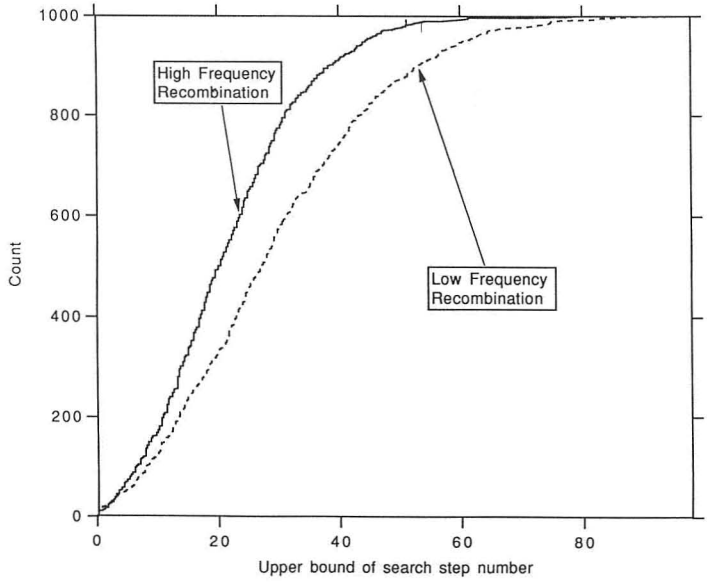

Figure 9: Comparison of success rate between different frequencies of recombination in genetic search.

3. Search performance is improved considerably when the number of genes in the gene pool is increased. This indicates that the present method is especially suited for parallel processing.

4. In performing the simulation, spurious attractors were found, many of which were useful in the sense of synthesizing patterns (information generation) from stored patterns.

Acknowledgments

The authors deeply thank Dr. Peter Davis for his valuable comments.

References

[1] J. A. Kelso, A. J. Mandell, and M. F. Shlesinger, editors, *Dynamic Patterns in Complex Systems* (Singapore: World Scientific, 1988).

[2] H. G. Schuster, *Deterministic Chaos* (VCH Weinheim, 1985).

[3] R. L. Devaney, *An Introduction to Chaotic Dynamical Systems*, (Menlo Park, CA: Benjamin Cummings, 1986).

[4] K. Kaneko, *Physica D*, **41** (1990) 137.

[5] J. S. Nicolis, *Rep. Prog. Phys.*, **49** (1986) 80.

[6] *Neural and Synergetic Computers*, vol. 42, Springer Series in Synergetics, edited by H. Haken (Berlin: Springer Verlag, 1989).

[7] I. Tsuda, E. Koerner, and H. Shimizu, *Prog. Theor. Phys.*, **78** (1987) 51.

[8] P. Davis and S. Nara, *Tech. Dig. of Int. Conf. on Fuzzy Logic and Neural Networks*, Iizuka (1990).

[9] P. Davis and S. Nara, page 97 in *Proceedings of the First Symposium on Nonlinear Theory and its Applications* (1990).

[10] S. Kirkpatrick, C. D. Gellat, and M. P. Vecci, *Science*, **220** (1983) 671.

[11] Y. Mori, P. Davis, and S. Nara, *Journal of Physics A*, **22** (1989) L525.

[12] S. Nara and P. Davis, *Neural Networks*, **6** (1993) 963.

[13] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Language and Computation* (Reading, MA: Addison-Wesley, 1979).

[14] S. Wolfram, *Theory and Application of Cellular Automata*, (Singapore: World Scientific, 1986).

[15] H. Haken, *Information and Self-Organization* (Berlin: Springer-Verlag, 1988).

[16] J. A. Anderson and E. Rosenfeld, editors, *Neurocomputing* (Cambridge, MA: MIT Press, 1988).

[17] J. A. Anderson, A. P. Pelliioniz, and E. R. Rosenfeld, editors, *Neurocomputing 2* (Cambridge, MA: MIT Press, 1991).

[18] D. Rumelhart et al., in *Parallel Distributed Processing*, edited by J. L. McClelland, D. E. Rumelhart, and the PDP Research Group (Cambridge, MA: MIT Press, 1986).

[19] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley, 1989).

[20] L. Davis, editor, *Genetic Algorithms and Simulated Annealing* (London: Pitman, 1989).

[21] J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms* (San Mateo, CA: Morgan Kaufman, 1989).

[22] R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Diego (San Mateo, CA: Morgan Kaufmann, 1991).

[23] H. P. Schwefel and R. Manner, editors, *Proceedings of the First International Conference on Parallel Problem Solving from Nature*, Dortmund (Berlin: Springer Verlag, 1990).

[24] L. Personnaz, I. Guyon, and G. Dreyfus, *Phys. Rev.*, **A34** (1986) 4217.

[25] Fuchs and H. Haken, page 33 in *Dynamic Patterns in Complex Systems*, J. A. Kelso, A. J. Mandell, and M. F. Shlesinger, editors (Singapore: World Scientific, 1988).

[26] W. Banzhaf, *Biological Cybernetics*, **64** (1990) 7.