Some Experiments on the Use of Genetic Algorithms in a Boltzmann Machine

Matthew I Bellgard
matthew@cs.uwa.oz.au

Chi Ping Tsang
tsang@cs.uwa.oz.au

Logic & AI Laboratory
Department of Computer Science
The University of Western Australia
Nedlands, W.A. 6009

ABSTRACT

While it appears to be a good idea to use Genetic Algorithm(GA) to train a Neural network, past results do not confirm such optimism. The main problems encountered are the speed of convergence, convergence to the wrong answer, and failure to converge. In this paper we combine GA and Simulated annealing to form a Genetic Boltzmann Machine(GBM) and attempt to understand the properties of such an architecture by experiments. We introduce the concept of weight reordering and demonstrate that it overcomes most of the convergence problems. Results of other experiments are also shown relating to the selection of parameters for the GA: the effects of population, different crossover point operators, and hidden units are illustrated. We conclude that with careful design a GBM can perform nearly as well as a Boltzmann Machine in a scalar computer. However, GBM is easily amenable to parallel computation by process farming.

1 INTRODUCTION

In the last few years, Artificial Neural Nets (ANNs) have been used very successfully for many recognition problems. Typically they include pattern recognition, phoneme recognition, and classification problems. Various architectures have also been attempted including competitive learning, Hopfield nets, back-propagation, self-organising networks [Ve88] and the Boltzmann Machine (BM)[AHS85, AK89, HS86, Sm86].

A Genetic Algorithm (GA)[De88, Go89] is a search procedure which is motivated from standard models of heredity and evolution in the field of population genetics. It incorporates the concepts of adaptation present in natural systems. The hybrid combination of connectionism and GA is currently an interesting topic, and the results obtained so far attempt to evaluate GA's effectiveness as a search procedure within ANNs. However, the results have been relatively negative[Ki90]. Typically in recent studies, a GA is used to search for a set of weights in a feed-forward network which usually employs the back-propagation learning algorithm[Ki90, WH89]. In general, the solutions from GA are found to be coarse grained and are only an approximation. Hence it must be followed by a traditional learning algorithm. The contention is that whether GA is a cost effective approximating process. The use of GA introduces other problems. Firstly, GA itself may converge very slowly. Secondly, it may converge to a wrong answer and lastly it may not converge at all [Ki90, WH89].

While there has been some studies on the comparison of GA and various Neural learning algorithms[Ki90], there is no report yet on the use of GA in BM. Intuitively, the application of GA to BM is more natural than other Neural architectures, because both Simulated Annealing (SA) and GA are global optimization techniques. Furthermore SA itself is very slow and is highly dependent on the ultrametricity of the problem[KY85, Li90, MV85, SSW86,]. Hence an alternative randomly jumping optimization may complement SA nicely. In fact, one of the difficulties of BM is that it may fall into a wrong ravine in the early stage of the search [Li90]. It is hoped that the combination of GA and conventional BM will improve this. In this paper we examine the methods to combine GA and BM and evaluate the effect of the combined system.

This paper describes an attempt to use a GA to search for a set of weights for the BM. A set of weights can be thought of as being a point in the weight configuration space. The weights in a BM will be treated as genes (individuals of a population) and the genetic search will enable large jumps in the weight configuration space allowing the BM to escape entrapment from local minima.

The outline of this paper is as follows: section 2 provides some background to the BM and GA. The GBM algorithm and weight reordering is described in section 3. Section 4 describes experiments conducted on the encoder-

CH 3065-0/91/0000-2645 $1.00 © IEEE

decoder problem, and illustrates the properties of a GA in GBM. Section 5 summarizes the results and concludes the paper.

2 BACKGROUND

2.1 The Boltzmann Machine

The BM considered here is a 2-layer network consisting of a Hidden layer and an Input/Output layer (IO layer). There are only connections between units in different layers. There are no winner-take-all connections between units in the IO layer which minimizes the search space, rendering the network problem specific[AHS85, HS86]. In the learning phase, the network is annealed to thermal equilibrium by successively annealing the IO layer and the hidden layer with each environmental pattern. The effective polynomial-time cooling schedule described by [AK89] was used. Cooccurence statistics are then calculated to obtain estimates with which to update the connection weights in the network. Each annealing and update cycle is known as a sweep in the learning phase. After a large number of sweeps, the network stabilizes to a set of weights which characterize the input patterns presented at the learning phase. At the recognition phase, partial input patterns may be presented and the network is again annealed to complete the pattern, using the learned weights.

2.2 Genetic Algorithms

Parameter-tuning problems in GAs[De88, Go89] are represented in terms of populations of linear genes. To produce a successive generation of the population, two parents are randomly selected according to their relative fitness value. By using the crossover and mutation operators, two children are constructed from these parents. It is hoped that the population evolves to one where most of the individuals are of high fitness values. GAs will work extremely well in domains where the search space is unknown and unstructured.

3 GENETIC BOLTZMANN MACHINE

In order to incorporate the GA optimization method into the learning phase of a BM, we define the GBM in this section. Consider a 2-layer BM as in Fig. 1a with 4 units in the IO layer and 2 hidden units, the set of weights may be represented as a linear gene shown in Fig. 1b or by a 2-D gene as in Fig. 1c. The linear gene is built from connections from units in the hidden layer to those in the IO layer. Two bias units are introduced such that biases of the threshold can be accounted for. In the GBM, we treat the weights as genes of the system. During the learning phase, these genes have to be optimised such that they can be used to perform completion. We use the GA to perform initial optimization which is followed by the traditional BM learning mechanism. The outline of the GBM is shown in Fig 2.

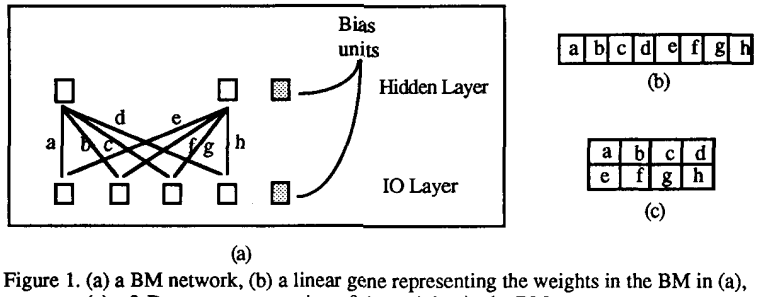

Figure 1. (a) a BM network, (b) a linear gene representing the weights in the BM in (a), (c) a 2-D gene representation of the weights in the BM.

Each gene in a population represents a set of weights for a predefined BM as in Fig. 1. Individuals in a random initial population are evaluated to determine how well the set of weights have learned the particular problem. In the GBM this is determined by the Hamming distance between the completed IO pattern and the environmental patterns. Completion is performed in the same manner as in the recognition phase with the exception that a shorter annealing schedule is used to save time. This Hamming distance is called the fitness value in this paper. After the fitness of a population is determined, a crossover operator is applied to two individuals which are probabilistically

chosen from the current population. The probability of an individual to be chosen depends on its fitness value [Go89]. From these two parents, two new individuals are generated. The generation procedure is made up of two steps. The children are first annealed and then the weights of these new individuals are reordered. The fitness of the new individuals can also be determined. A generation is completed when a new population of the same size is generated.

For the encoder-decoder problem, with an IO layer containing 8 units (similar to the 4-2-4 problem described elsewhere [AHS85]), only one unit needs to be clamped to "1" and the remaining 7 units are annealed to obtain the desired encoding. Thus there is a total of eight clampings. The result of each annealed solution is then compared to the desired solution to determine an error. The fitness value of an individual is the maximum fitness (56 in the case of the 4-2-4 problem) minus the sum of the errors for all patterns.

```
Initialize Population
Anneal both Children
Weight reordering on Children
Anneal both Children
Repeat
    For (pop/2) times
        Select 2 fit parents
        Perform Crossover
        Anneal both Children to obtain fitness
        Perform Weight reordering on Children
        Obtain fitness for both Children
Until (population convergence) or (100% gene fitness ) or
      (generations > maxgen)
```

Figure 2. Description of the GBM algorithm

3.1 Weight Reordering

We have introduced the concept of weight reordering into the learning algorithm for GBM. This is a very important step which makes GBM viable and competitive. It is essentially a normalizing process. Its importance is demonstrated by our experiments. The problem arises when there are more hidden units than are absolutely necessary. Usually one can see that hidden units are like redundancies and are introduced into a Boltzmann machine such that the traditional BM has a higher likelihood of converging to a correct solution and taking less sweeps. However, for a GA, redundancy introduces multiple solutions which leads to diverging population. Hence in the presence of more hidden units and redundancies, the combination of genetic and annealing leads to non-convergence.

To overcome the degenerate solutions during the early generations of GA, we introduce the concept of weight reordering and its algorithm. First, we must realise that the arrangement of the IO units and weight connections can be subjected to a permutation as long as the resultant completion agrees with the environmental patterns(training examples). When the redundancy is high there will be degenerate solutions during early stages of the optimization process. If these different solutions are preserved, they will prevent the development of the final correct solution in a GA. Weight reordering is to overcome this problem by permuting the IO units and their corresponding weights such that there is least amount of error between the completed solution and the environment patterns. We base our ordering on the total number of errors between all the completed values and the environmental patterns of a particular IO unit. The IO units and weights are reordered such that the total error is minimised. Hence we call it weight reordering.

The form of weight reordering we performed will be better explained with a simple example. Consider the network as in Fig. 1a with a 2-D weight representation as in Fig. 1c. Each row of the 2-D weight representation corresponds to connections to a given hidden unit and each column corresponds to connections to an IO layer unit. Assume that the set of weights so far is able to complete the two patterns, 1001 and 0110 in the IO layer. The desired environmental patterns we require from the BM are: 1010 and 0101. As shown in Fig. 3, the columns of the weight set can be ordered to obtain the correct solution with the given set of weights. Finding the best possible

arrangement for a given set of weights is a time-consuming factorial search, but we perform weight ordering via a greedy search. Although the greedy method will not necessarily give us a global minimum, it is much faster.

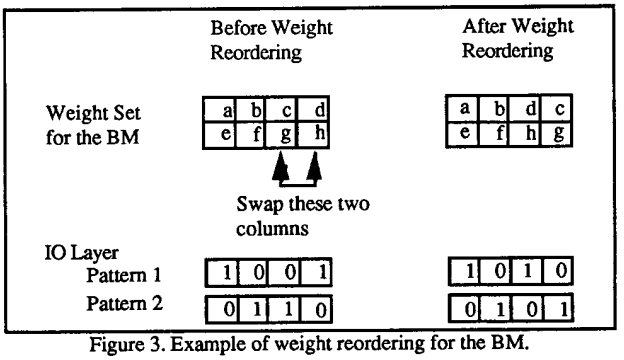

Figure 3. Example of weight reordering for the BM.

Weight reordering in the GBM algorithm is performed as follows: To reorder the weights, a rough and quick annealing is first applied to the given set of weights with the IO units unclamped according to the input-output requirements. This procedure is applied to all the environmental patterns. The result is that for each environmental pattern there is a completed IO vector. The total error between each column of the IO vector and the first column of the environmental vectors are then calculated. The weight columns are then permuted such that the lowest error IO column appears as the first column. This process is repeated for the remaining columns until the last column of the environmental columns. This weight reordering is essentially a minimisation of the error between the completed results and the environmental patterns. The computational cost of this process is relatively high. However, its benefit outweighs its costs as is shown by our results.

3.2 Other Decisions for the Genetic Algorithm

There are many decisions to be made to get a genetic algorithm working. Typically, one has to specify the following parameters: the type of crossovers, the population size, the crossover rate, the criteria for convergence, selection criteria and mutation (if any). We have decided not to have any mutation, and fixed the crossover rate at 70%. The selection criteria chosen is the fitness value (hamming distance). Our criteria for convergence is when either a 100% fit gene is found or population convergence (over 2/3 of the population) occurs. Note this criteria is much stronger than most other GA applications to ANN. Most of the other work (eg. [Ki90, WH89] terminate GA when a certain approximate correct solution is found, not waiting for population convergence. In our experiments we investigate into the effects of population size as well as that of different crossovers. We have used 1-point, 2-point, Uniform crossovers[ECS89, Sy89] and the Row/Col crossover in the algorithm and the results are described below. 1-point, 2-point and the Uniform crossovers are well known in the literature and each can be described in terms of possessing a varying degree of positional and distributional bias[ECS89]. The Row/Col crossover is a new operator which utilizes the nature of the 2-D gene inherent in a neural network (Fig. 1c). In this operator, the rows or columns of the 2-D gene are randomly chosen as the crossover, and the child gene is formed by combining different parts of the parent genes which are partitioned by the crossover. Each row of this gene corresponds to connections to a given hidden unit and each column corresponds to connections to an IO layer unit.

4 EXPERIMENTAL RESULTS

Experiments were conducted on the the encoder-decoder problem. The BM we used contained only connections between the IO layer and the hidden layer[AHS85] with no inhibitory connections within the IO layer. The experiments are performed on the encoder-decoder problem with eight units in the IO layer. The weights in the initial population of genes were randomly assigned an integer value from the range (-64..+64) from a uniform distribution. This ensures that there is no bias to a given problem from the initial selection[Ki90]. For timing purposes, all trials were run on a Sparc 1+ sun workstation and all computational time in terms of Sparc 1+ seconds.

In all the experiments the crossover rate was fixed to 0.7 and there was no mutation operator. After population convergence or if 100% correct solution was found or when the maximum allowable generations of 200 had expired (this cut-off was introduced for timing purposes), Genetic search was terminated and the BM learning algorithm was performed to obtain a fine-grained solution. The bias weights were also initialized to zero.

Experiment 1: Convergence and Weight Sorting

This first experiment was designed to determine the convergence characteristics of GBM and the effectiveness of weight reordering. These experiments were performed with a fixed population of 100 and 5 hidden units. Fig 4a illustrates that a GA converges without the use of weight reordering. We note that population convergence occurs when the fitness reaches above 40. However, the approximated value has a low fitness value and requires on average a much longer BM training before the final correct convergence. For 5 hidden units, the BM learning algorithm will always find the correct solution even though it may take a long time, hence a maximum allowable number of sweeps of 5000 was introduced for timing purposes. This result is compared to those where weight reordering is used. Here in Fig 4b 5b, the GA converges to a set of weights with much higher fitness values. In some cases a perfect answer was found as shown in Fig. 4b. Furthermore, the follow-up SA takes only very little time to converge to the correct answer. For comparison purposes timings for 15 trials of just using the BM learning algorithm alone is reported in Fig. 5c. Here, the weights for each trial were set to small real numbers, randomly generated, less than 1.0. In table 1, we compare the average characteristics of a GBM with and without weight reordering. Computational mean time between correct solutions from 15 trials is reported.

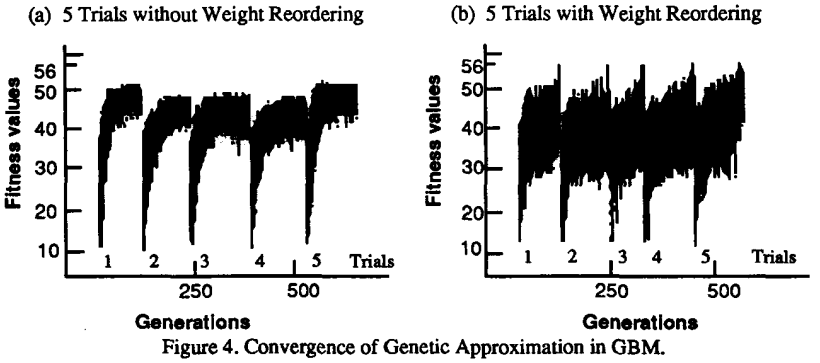

Figure 4. Convergence of Genetic Approximation in GBM.

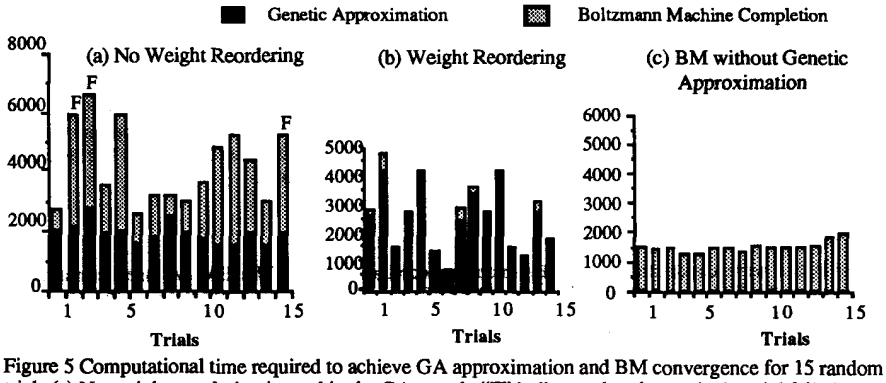

Figure 5 Computational time required to achieve GA approximation and BM convergence for 15 random trials.(a) No weight reordering is used in the GA search. "F" indicates that the particular trial failed to converge to a correct solution, (b) GA search with weight reordering and (c) BM to a correct solution without GA.

Fig 4 shows the convergence properties when weight reordering is applied or not applied. It should be noted that in the case where weight reordering is used (Fig. 4b), the convergence does not level out and there is a wide spread of search values. In fact the GA stage is terminated only because a 100% fit gene is found. On the other hand, the search without weight reordering levels out with a population convergence and results in a bad solution. This fact is further shown by Fig 5 where we can see that the BM time required to achieve final fine grain convergence is very small when weight reordering is used. Another noticeable result is that the overall computational time required for the case with weight reordering is substantially smaller than otherwise. These results demonstrate clearly that weight reordering permits convergence to the correct answer as well as improve the speed of convergence.

Table 1. Statistics of computational time for GA approximation and BM completion of a GBM with 5 hidden units from 15 independent trials.

Population	GA approximation		BM completion	
	mean	std	mean	std
No weight reordering Pop = 100, hid = 5	2440	1556	2783	2527
With weight reordering Pop = 100, hid = 5	2464	1100	148	182

Experiment 2: Effect of Population size

The size of the population is extremely important to the efficiency of GA, because it is directly proportional to the computational time required. This experiment was performed with 5 hidden units, the Row/Col crossover, and various sizes of the population. Weight reordering was used in all the experiments. Table 2. shows the results from varying the population against successful termination.

Table 2. Statistics of computational time for GA approximation and BM completion of a GBM with 5 hidden units from 15 independent trials.

Population	GA approximation		BM completion	
	mean	std	mean	std
20	320	232	3664	3673
40	1037	639	1737	2677
60	1989	1058	518	519
80	1706	795	272	296
100	2464	1100	148	182
120	3297	1674	130	106

In table 2, the results show that the mean of the BM time required to get the final solution decreases with the increase in population during the GA stage. This indicates that GA results in a better approximation when a larger population is used. Another interesting result is that the average total computational time required is at a minimum when the population is around 80. This demonstrates that for each problem there is an optimal population size and for our example it is near 80. Fig 6 shows the computational time for each trial. It is interesting to note that for populations 20 and 40 the mean time is poor. This is due to a few bad results from GA approximation leading to a worse than normal BM convergence. As the population increases, the chance for a wrong approximation diminishes, but the actual time of computation also increases. The opposing effects resulted in a minimisation of the mean time between correct convergences at a population around 80. This shows that there exists an optimal population selection for GBM.

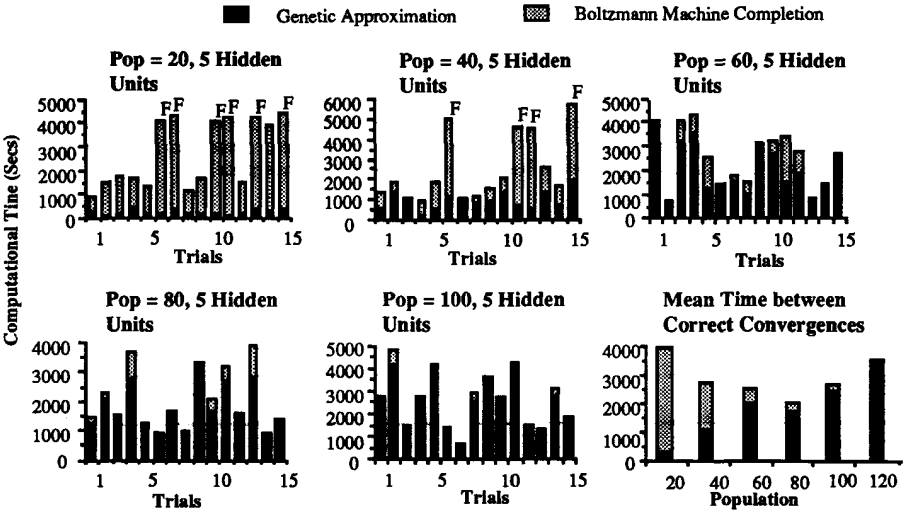

Fig 6. Computational time required for GA approximation and BM convergence for different population sizes. "F" indicates that the particular trial failed to converge to a correct solution. The last graph shows the mean time between correct convergences for various populations.

Experiment 3: Effects of hidden units

Fig. 7 plots the number of successful terminations against different number of hidden units for the GBM at a fixed population of 100 and for the traditional BM. It can be seen that the GBM with more hidden units more consistently finds a solution than with a GBM with fewer hidden units. The average number of sweeps required to obtain a correct solution decreases with an increase in the number of hidden units due to correct convergences. Table 3 displays the mean time between correct solutions for both the GBM and the BM.

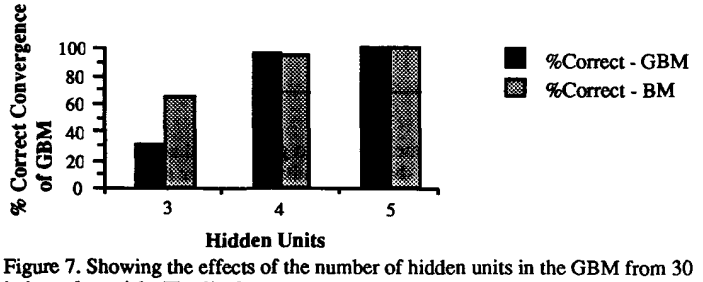

Figure 7. Showing the effects of the number of hidden units in the GBM from 30 independent trials. The % of correct convergences increases as the number of hidden units increase.

Table 3. Statistics of computational time between correct solutions for the GBM and the BM from 15 trials.

No. of Hidden Units	GBM Computational Time between Correct Convergence		BM Computational Time between Correct Convergence	
	mean	std	mean	std
3	10494	8230	3122	2504
4	3609	1174	1640	767
5	2612	1193	1436	216

Our results show, Fig. 7, that GBM performs as well as BM in terms of convergence to the correct answer for four and five hidden units. For three hidden units GBM does not perform as well as BM. This is because the bias weights were set to 0. One important result is that by using enough hidden units we can arrive at 100% correctness. This is one of the reasons why we use 5 hidden units in our previous experiments. Once we are certain about the outcome, the computational time in the BM stage will indicate how good the genetic approximation is. The mean time between correct answers shows that without further selection processes GBM will take 1.8 times as long to arrive at a correct answer. However, if we look at the result for population 100 in Fig 6., we see that the time for BM is reduced and that most of the time to find a correct solution is taken up by the GA search. Of course we have not considered the possibilities for parallel implementations yet.

5 CONCLUSIONS

We have designed and implemented a GBM for the study of the properties of GA on Boltzmann Machines. The most important result is that we have introduced the concept of weight reordering to a population of genes and demonstrated that it improves the convergence characteristics of the GA. In fact, when weight reordering is not used GA generally leads to a bad approximation which eventually locks the BM into a wrong ravine. Our experiments demonstrate that by increasing the population in conjunction with weight reordering a finer grain solution can be achieved by GA such that only a very small proportion of the time is needed for BM completion. GA can provide a good set of weights for the BM to solve the problem. Lastly, further experiments conducted show that the Row/Col crossover performed better than 1-point, 2-point and the uniform crossover operators for this problem. With the above results, we have shown that it is quite possible to construct a GBM which is computationally competitive with a BM in the conventional sequential machine, and it is amenable to parallel farming.

REFERENCES

[AHS85] Ackley, D. H., Hinton, G. E. and Sejnowski, T. J., "A Learning Algorithm for Boltzmann Machines", Cognitive Science 9, pp147-169, (1985).

[AK89] Aart, E. and Korst, J., *Simulated Annealing and Boltzmann Machines*, John Wiley and Sons, Brisbane, (1989).

[Ch90] Chalmers, D. J., "The Evolution of Learning: An Experiment in Genetic Connectionism", Proc. 1990 Connectionist Summer School, San Mateo, CA, Morgan Kaufmann, (1990).

[De88] DeJong, K., "Learning with Genetic Algorithms: An Overview", Machine Learning, Langley, P. (Ed.), Vol 3, No. 2/3, pp.121-138, Kluwer Academic Publishers (1988).

[ECS89] Eshelman, L. J., Caruana, R. A. and Schaffer, J. D., "Biases in the Crossover Landscape", Proc. of the Third Int. Conf. on Genetic Algorithms, ICGA'89, pp.10-19, Morgan Kaufmann,(1989).

[Go89] Goldberg, D. E., *Genetic Algorithms in Search Optimization & Machine Learning*, Addison-Wesley, (1989).

[HS86] Hinton, G. E. and Sejnowski, T. J., "Learning and Relearning in Boltzmann Machines", in Parallel Distributed Processing Vol 1 Edited by Rumelhart D.E. and McClelland J.L., MIT Press, (1986).

[Li90] Lister, R., "On Leaping Tall Maxima in a Single Bound: An Improved Learning Algorithm for the Boltzmann Machine", Basser Tech. Rep. TR386, Oct., (1990).

[Ki90] Kitano, H., "Empirical Studies on the Speed of Convergence of Neural Network Training using Genetic Algorithms", AAAI, pp.789-795, AAAI Press/The MIT Press, (1990).

[KY85] Kirkpatrick, S. & Toulouse, G., "Configuration Space Analysis of Travelling Salesman Problems", Physique, 46, pp.1277-1292, (1985).

[MV85] Mezard, M. & Virasoro, M. A., "The Microstructure of Ultrametricity", Physique, 46, pp.1293-1307, (1985).

[Sm86] P. Smolensky, "Information Processing in Dynamical Systems: Foundations of Harmony Theory", in Parallel Distributed Processing Edited by Rumelhart D.E. and McClelland J.L., MIT Press, (1986).

[Sy89] Syswerda, G., "Uniform Crossover in Genetic Algorithms", Proc. of the Third Int. Conf. on Genetic Algorithms, ICGA'89, pp.2-9, Morgan Kaufmann, (1989).

[SSW86] Solla, S. A., Sorkin, G. B. and White, S. R.,"Configuration Space Analysis for Optimization Problems", Disordered Systems and Biological Organization (NATO ASI Series, Vol. F20), Bienenstock, E. et al. (Eds), (1986).

[Ve88] V. Vemuri, "Artificial Neural Networks: Theoretical Concepts", IEEE Computer Society Press, (1988).

[WH89] Whitley, D., Hanson, T., "Optimizing Neural Networks Using Faster, More Accurate Genetic Search", Proc. of the Third Int. Conf. on Genetic Algorithms, ICGA'89, pp.391-197, Morgan Kaufmann, (1989).