Central Point Crossover for Neuro-genetic Hybrids

Soonchul Jung and Byung-Ro Moon

School of Computer Science and Engineering, Seoul National University
Shillim-dong, Kwanak-gu, Seoul, 151-742 Korea
{samuel, moon}@soar.snu.ac.kr

Abstract. In this paper, we consider each neural network as a point in a multi-dimensional problem space and suggest a crossover that locates the central point of a number of neural networks. By this, genetic algorithms can spend more time around attractive areas. We also apply representational normalization to neural networks to maintain genotype consistency in crossover. For the normalization, we utilize the Hungarian method of matching problems. The experimental results of our neuro-genetic algorithm overall showed better performance over the traditional multi-start heuristic and the genetic algorithm with a traditional crossover. These results are evidence that it is attractive to exploit central areas of local optima.

1 Introduction

Artificial neural networks (ANNs) have been used to address a variety of problems in pattern recognition, pattern classification, optimization, associative memory, control mechanism, prediction, function approximation, etc. [10] [11]. Even when such a problem addressed by ANNs is fairly small, the problem space is often intractably large that it is almost impossible to find an optimal solution by exhaustive or simple search methods. Thus, heuristic algorithms have been used as alternatives. They provide reasonable solutions — local optima — in acceptable computing time.

A lot of studies have been conducted on the ruggedness and the properties of problem search spaces. A good insight into problem spaces can provide a motivation for a good search algorithm. Kauffman [13] proposed the NK-landscape model that can control the ruggedness of a problem space. Sorkin [28] defined the fractalness of a solution space and proposed that simulated annealing is efficient when the space is fractal. Manderick *et al.* [20] measured the ruggedness of a problem space by autocorrelation function and correlation length obtained from a time series of solutions. Weinberger [30] conjectured that, if all points on a fitness landscape are correlated relatively highly, the landscape is bowl-shaped. Boese *et al.* [3] suggested that, through measuring cost-distance correlation for the traveling salesman and graph bisection problems, the cost surfaces are globally convex. Jones and Forrest [12] introduced fitness-distance correlation as a measure of search difficulty.

ANN is an information processing paradigm that was inspired by the way that biological nervous systems process information. An ANN learns from its environment through an iterative process of adjusting its weights. Although ANNs have been widely applied to a variety of problems, their quality is limited. They often get trapped in local optima due to the limit of their learning algorithms [10]; the back-propagation algorithm, the most popular learning method for ANNs, is basically a hill-climbing technique.

One way to overcome the shortcomings of hill-climbing techniques is to adopt a global search algorithm like evolutionary algorithms. Evolving neural networks has shown successful results [25] [18] [24]. According to Yao [32], evolution of ANNs can be classified into three representative approaches: i) finding a near-optimal set of connection weights globally for an ANN with a fixed architecture [31] [23] [27], ii) evolving architectures of an ANN (i.e., scale and connectivity) in order to adapt to different tasks [16] [9] [19], and iii) evolving learning rules to improve an ANN's learning ability [5] [7] [14].

Kim and Moon [15] extensively investigated the properties around central areas of local optima for the graph bipartitioning problem. Based on their observation, they suggested a new multi-parent crossover that exploits central areas of local optima. Their genetic algorithm (GA) was peculiar enough to select the total population as the parent set and perform the crossover on them. In this paper, we modify their multi-parent crossover to suit neural networks and incorporate the modified crossover into the conventional genetic algorithm. We name it *central point crossover*. We consider each neural network as a point in a multi-dimensional problem space, and compute the central point of multiple neural networks during crossover.

An ANN has a lot of other ANNs whose functionality is equivalent to it, which implies the many-to-one mapping between genotypes and phenotypes of ANNs. The existence of redundancy in encodings is known to be harmful in producing good offspring [6]. Thierens [29] transformed a neural network into its canonical form by flipping the signs of connection weights and reordering hidden neurons. Although the transformation to the canonical form reduced the redundancy, the "similarities" of neural networks were not well considered. We resolve this problem by defining a distance measure for neural networks and normalizing parental neural networks on the basis of the distance measure.

This paper is structured as follows. In the next section, we describe the architecture, isomorphism, and distance measure of neural networks. In Section 3 and 4, we explain the central point crossover and the experimental results, respectively. Finally, we make conclusions in Section 5.

2 Preliminaries

2.1 Architecture

We use a typical multilayer feed-forward network (Fig. 1). A network consists of a set of sensory units that constitutes the input layer, one or more hidden layers of computation neurons, and an output layer of computation neurons. In this paper, we focus on neural networks with one hidden layer. A neuron in any

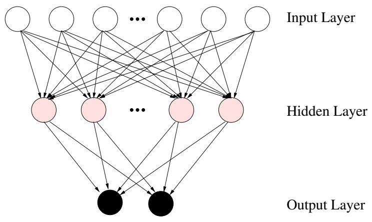

Fig. 1. Architectural graph of a multilayer feed-forward network

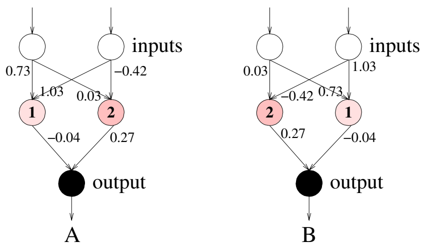

Fig. 2. Two isomorphic neural networks

layer of the network is connected to all the neurons in the previous layer. Each associated edge has a weight. Since we consider neural networks with only one hidden layer, the k^{th} output, $y_k^{\mathfrak{n}}$, of a neural network $\mathfrak{n}$, is given simply by

$$y_k^{\mathfrak{n}} = \varphi\Big\{\sum_{j=1}^{J} w_{kj}^{\mathfrak{n}} \cdot \varphi(\sum_{i=1}^{I} w_{ji}^{\mathfrak{n}} \cdot x_i)\Big\}$$

where I and J are the numbers of input and hidden neurons, respectively, x_i is the value of the i^{th} input neuron, and $\varphi(x)$[1] is a sigmoidal transfer function; $w_{kj}^{\mathfrak{n}}$ is the connection weight from j^{th} hidden neuron to k^{th} output neuron and $w_{ji}^{\mathfrak{n}}$ is the weight from i^{th} input neuron to j^{th} hidden neuron.

2.2 Isomorphism of Neural Networks

In Fig. 2, two neural networks A and B look different from each other. In other words, they have different representations. However, they are isomorphic because of their equivalent functionality; they output the exactly same value with respect to the same input vector. In fact, any permutation of the hidden neurons including incoming and outgoing weights, produces the same neural network with a different chromosomal representation. Therefore, there are $n!$ representations with respect to a neural network with n hidden neurons. This results in the many-to-one mapping between genotypes and phenotypes. Each genotype corresponds to a permutation from the base neural network. This problem, known as

[1] $\varphi(x) = \frac{1}{1+\exp(-x)}$

the permutation problem [1], makes crossover hard to consistently inherit traits of the parents.

2.3 Distance Measure

It is useful in GAs to define a measure of distance between neural networks. It can be used in selection, crossover, replacement operators, etc. Each neuron in the hidden layer can be represented by a weight vector of edges incident to input and output neurons. Thus, a neural network can be represented by a 2D matrix of weights. The distance between two neural networks can be defined as follows:

Definition 1. *Let I, J, and K be the numbers of input neurons, hidden neurons, and output neurons, respectively. Consider two neural networks* $\mathfrak{n} = \{\mathfrak{h}_1, \mathfrak{h}_2, \ldots, \mathfrak{h}_J\}$ *and* $\mathfrak{n}' = \{\mathfrak{h}'_1, \mathfrak{h}'_2, \ldots, \mathfrak{h}'_J\}$ *where* $\mathfrak{h}_j$[2], $\mathfrak{h}'_j \in \mathcal{R}^{I+K}$. *Given a metric* $\mathfrak{d} : \mathcal{R}^{I+K} \times \mathcal{R}^{I+K} \to \mathcal{R}$ *in Euclidean space*[3], *the distance between the two networks is defined as*

$$\Delta_J(\mathfrak{n}, \mathfrak{n}') = \sum_{j=1}^{J} \mathfrak{d}(\mathfrak{h}_j, \mathfrak{h}'_j).$$

The measure Δ_J is computed easily and fast, but it is not very useful due to the permutation problem. For example, consider a neural network A and its isomorphic neural network A'. The distance between them has diverse values according to the representations of A'; the distance little reflects the functional similarity of the two neural networks.

We define a better distance measure D_J as follows:

Definition 2. *Given the metric* $\mathfrak{d}$, *and the two neural networks* $\mathfrak{n}, \mathfrak{n}'$ *as in Definition 1, we define the distance* D_J *between the two networks as*

$$D_J(\mathfrak{n}, \mathfrak{n}') = min_{\sigma \in \Sigma_J} \left(\sum_{j=1}^{J} \mathfrak{d}(\mathfrak{h}_j, \mathfrak{h}'_{\sigma(j)}) \right)$$

where σ denotes a permutation.

Since D_J is computed according to such a mapping that each hidden neuron of $\mathfrak{n}'$ is mapped to a similar one to $\mathfrak{n}$, it better reflects the functional similarity of the two neural networks than does Δ_J.

D_J is a *metric* in the neural network space and thus satisfies the triangle inequality ($D_J(x,z) \le D_J(x,y) + D_J(y,z)$). The proof is omitted by space limitation. A brute-force algorithm to calculate D_J consumes $O(J!)$ time if it enumerates all the $J!$ permutations to find the optimal one. Fortunately, there is an efficient way to compute D_J; we can formulate the problem of computing D_J as the optimal assignment problem [8]. In other words, this is exactly the

[2] $\mathfrak{h}_j = [w_{j1}, \ldots, w_{ji}, \ldots, w_{jI}, w_{1j}, \ldots, w_{kj}, \ldots, w_{Kj}]^T$

[3] $\mathfrak{d}(\mathfrak{h}_a, \mathfrak{h}_b) = \|\mathfrak{h}_a - \mathfrak{h}_b\| = \sqrt{\sum_{i=1}^{I}(w_{ai} - w_{bi})^2 + \sum_{k=1}^{K}(w_{ka} - w_{kb})^2}$

problem of finding an assignment (permutation) with the minimum summation of Definition 2. It can be computed efficiently in $O(J^3)$ by the Hungarian method [17]. Fig. 3 shows the assignment weight matrix $M = (m_{ij})$ between two neural networks $\mathfrak{n}$ and $\mathfrak{n}'$. Each element m_{ij} means $\mathfrak{d}(\mathfrak{h}_i, \mathfrak{h}'_j)$. We reorder the hidden neurons of $\mathfrak{n}'$ by the Hungarian method and pursue maximal genotypical consistency between neural networks.

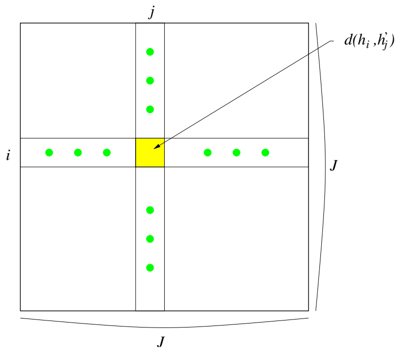

Fig. 3. The assignment weight matrix between two neural networks

3 Central Point Crossover

3.1 Normalization of Neural Networks

As mentioned before, a neural network has a number of isomorphic ones. It causes the existence of redundant encodings in a GA where neural networks are represented by chromosomes. Redundant encodings are known to be harmful in keeping the respectfulness and combination power that crossovers have to possess [26] [6]. The permutation problem is highly related to the redundant encoding in GAs. From the viewpoint of GAs, normalization is an approach that transforms the genotype of one parent to be consistent with that of the other one [6]. It alleviates inconsistency caused by redundant encodings in GAs. We perform normalization by the Hungarian method.

Since our central point crossover manipulates a set of parents, the normalization routine has to process more than two neural networks unlike usual GAs. The algorithm is as follows:

1. Find out the fittest one $\mathfrak{n}_b$ among a set of N parent neural networks.
2. For each neural network $\mathfrak{n}$ in the set, a) use the Hungarian method to find the optimal assignment between $\mathfrak{n}$ and $\mathfrak{n}_b$, and b) transform $\mathfrak{n}$ into the isomorphic one by relocating its hidden neurons and connection weights according to the assignment of (a).

3.2 Central Point Crossover for Neural Networks

Boese *et al.* [3] analyzed relationships among local minima for the traveling salesman problem and the graph bisection problem. They conjectured that cost surfaces of both problems are globally convex, which implies that good solutions are highly probable to be located near other good solutions. Kim and Moon [15] conjectured from their experimental results that, given a subspace of local optima, the "central point" of the subspace is near the optimal solution. They performed experiments on approximate central points[4] for the graph bipartitioning problem, and showed that it was attractive to exploit central areas of multiple solutions.

Fig. 4 shows the template of the central point crossover for neural networks. The offspring of the central point crossover is exactly the central point of the parents. Fig. 5 shows an example crossover with three parents. The central point crossover on a real-number vector like a neural network is technically a generalization of the arithmetic crossover [22]. Because the use of the arithmetic mean tends to lead to premature convergence of a GA, we need to use a rather strong mutation to slow down the convergence speed.

```
central_point_crossover( $P$, $n$ )
// $P$: a set of parents, $n$: the number of parents
// $I, J, K$: the numbers of neurons in the input, hidden, and output layers,
//          respectively.
{
    $P \leftarrow$ normalize( $P$ );
    for each $j$ in $\{1, 2, ..., J\}$
        for each $i$ in $\{1, 2, ..., I\}$
        {
            $w_{ji}^{off} \leftarrow \frac{\sum_{p \in P} w_{ji}^{p}}{n}$;
        }
    for each $k$ in $\{1, 2, ..., K\}$
        for each $j$ in $\{1, 2, ..., J\}$
        {
            $w_{kj}^{off} \leftarrow \frac{\sum_{p \in P} w_{kj}^{p}}{n}$;
        }
    return the offspring off;
}
```

Fig. 4. The template of central point crossover

[4] In the graph bipartitioning problem, it is not easy to calculate the exact central point.

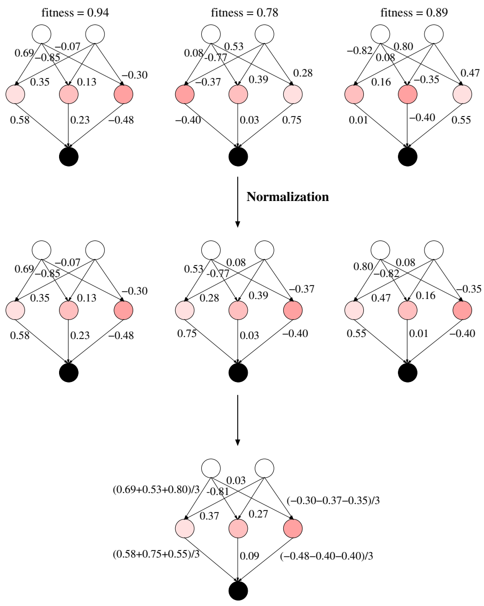

Fig. 5. An example of the central point crossover using three parents

4 Experimental Results

4.1 GA Framework

Fig. 6 shows the steady-state hybrid GA using neural networks as chromosomes. This GA allows more than two parents. Note that the parents are normalized during crossover. We denote the GA by CXGA.

- Initialization of population — Each weight of neural networks in the population ($n_{\text{pop}} = 200$) is initialized to a random number ranging from -0.5 to 0.5. After that, neural networks are local-optimized by the back-propagation algorithm.
- Selection — The selection operator here is based on the binary tournament selection. Two parent candidates are selected at random in the population, and the fitter is chosen as a parent. This process is repeated until n_{parent} distinct parents are selected.
- Mutation — The mutation operator is applied to the offspring with a specific probability ($p_{\text{mut}} = 0.1$). Each weight of the offspring is set to a random number between -0.5 and 0.5 with probability $p_{\text{mut2}} = 0.5$. Consequently, about 50% of weights are changed with new values by mutation in one of every ten offspring. Because the central point crossover may decrease population diversity quickly, we use a rather strong random mutation not to lose the diversity. Another reason of this high mutation rate is that local optimization follows the mutation.

```
**GA**( $n_{\text{pop}}$, $n_{\text{parent}}$, $p_{\text{mut}}$ )
// $n_{\text{pop}}$: population size, $n_{\text{parent}}$: the number of parents
// $p_{\text{mut}}$: probability to apply the mutation operator
{
    // Generate the local-optimized population $P = \{\mathfrak{n}_1, \mathfrak{n}_2, \ldots\}$.
    for each $i$ in $\{1, 2, ..., n_{\text{pop}}\}$
    {
        initialize( $\mathfrak{n}_i$ );
        $\mathfrak{n}_i \leftarrow$ back_propagate( $\mathfrak{n}_i$ );
    }
    $B \leftarrow$ the best among $\{\mathfrak{n}_1, \mathfrak{n}_2, \ldots, \mathfrak{n}_{\text{pop}}\}$;

    // Start the main loop.
    do
    {
        $Parent \leftarrow$ select( $P$, $n_{\text{parent}}$ );
        $off \leftarrow$ central_point_crossover( $Parent$, $n_{\text{parent}}$ );
        if ( random(0.0, 1.0) $< p_{\text{mut}}$ ) $off \leftarrow$ mutate( $off$ );
        $off \leftarrow$ back_propagate( $off$ );
        $P \leftarrow$ replace( $P$, $off$ );
        $B \leftarrow$ the best between $B$ and $off$;
    } while ( check_stop_condition() );
    return $B$;
}
```

Fig. 6. The steady-state hybrid genetic algorithm for neural networks

- Local optimization — We use the back-propagation learning algorithm [10] as the local optimizer. It is the most popular algorithm for the supervised training of multilayer feed-forward networks due to its simple implementation and fast computational speed. We use the cross-validation method [10] to prevent over-fitting and identify when to stop training. After each epoch of training, the network is tested on the validation set and then its learning degree $l(\mathfrak{n})$[5] is computed. When its current learning degree does not show improvement during n_{epoch}(=12) consecutive epochs, the learning stops.
- Replacement — We use an extended version of the genitor-style replacement operator [4]. Among parents whose fitnesses are worse than that of the offspring, it replaces the most similar one to the offspring (using distance measure D_J). If none of the parents are worse, it replaces the worst in the population with the offspring.
- Stop condition — The GA stops after a fixed number (=1500) of generations.

The fitness of a neural network is defined as $f(\mathfrak{n}) = \mu_t - \epsilon_t$ where μ_t and ϵ_t are the accuracy[6] and mean-square error on the training set, respectively.

[5] $l(\mathfrak{n}) = \mu_v - \epsilon_v$ where μ_v and ϵ_v are the accuracy and mean-square error on the validation set, respectively.

[6] $\mu_t = \dfrac{\text{\# of correctly classified examples}}{\text{\# of all the examples}}$

Table 1. The Accuracies of MS, TGA, and CXGA on the Test Sets

	CHDD		WBCD		ACCAD	
	ave[†]	best[‡]	ave	best	ave	best
MS	82.77	85.91	96.20	96.78	85.48	87.46
TGA	80.84	82.55	96.29	96.78	83.88	85.63
CXGA	83.26	85.91	96.20	96.20	86.42	87.16

† Average accuracy in percent over 100 runs on the test set.
‡ Best accuracy in percent over 100 runs on the test set.

4.2 Problem Instances

We selected three well-known problem instances from the UCI machine learning repository [2] to measure the performance of our GA. We removed the ones with missing attributes from the original examples.

Cleveland heart disease database (CHDD) contains 297 examples that consist of 160 healthy and 137 unhealthy cases. Each example has 13 attributes. We used as chromosomes neural networks with 13 input, 10 hidden, and 2 output neurons for CHDD. The examples were divided into 111 training, 37 validation, and 149 test examples.

Wisconsin breast cancer database (WBCD) [21] contains 683 examples with 444 benign and 239 malignant cases. Each example is described by a case number, nine attributes, and a binary class label. We used the 9-16-2 network structure for WBCD. The examples were divided into 256 training, 85 validation, and 342 test examples.

Finally, Australian credit card assessment database (ACCAD) has 653 examples with 296 granted and 357 ungranted credits. The output has two classes. Each example is comprised of six continuous attributes and nine discrete ones. We used the 15-14-2 network structure for ACCAD. The 653 examples were partitioned into 245 training, 81 validation, and 327 test examples.

4.3 Comparison with Other Heuristics

We compare CXGA with the traditional multi-start heuristic (MS) and a traditional GA without normalization using linear strings as chromosomes (TGA). CXGA here used 20 parents for crossover.

MS first generates a specific number of neural networks ($n_{\mathrm{ms}} = 1800$) obtained with the back-propagation, and then returns the best one among them. In order to make a fair comparison, the value of n_{ms} was determined for MS to spend the same time as CXGA. TGA used the two-point crossover on neural networks represented by one-dimensional arrays. The other settings were the same as CXGA.

We tried 100 runs for each of MS, TGA, and CXGA. Table 1 shows the accuracies on the test sets. CXGA overall outperformed MS and TGA. It is extraordinary that MS worked better than TGA for two problems. We suspect that the problem of redundant encoding caused some negative effect on TGA.

Table 2. The Accuracies of CXGAs on the Test Sets

	CHDD		WBCD		ACCAD	
	ave[†]	best[‡]	ave	best	ave	best
CXGA_2	82.15	83.22	96.35	96.78	85.68	86.24
CXGA_{10}	81.56	83.22	96.20	96.20	85.38	86.24
CXGA_{20}	83.26	85.91	96.20	96.20	**86.42**	87.16
CXGA_{30}	78.40	82.55	96.19	96.20	86.35	86.85
CXGA_{40}	81.44	83.22	**96.77**	96.78	86.36	86.85
CXGA_{50}	82.29	84.56	96.13	96.49	86.28	87.16
CXGA_{60}	82.17	83.89	96.19	96.49	85.62	86.85
CXGA_{70}	81.29	82.55	95.88	96.20	85.68	86.85
CXGA_{80}	**84.03**	85.91	95.88	96.20	85.19	86.54
CXGA_{90}	83.56	84.56	95.91	96.20	85.22	86.85
CXGA_{100}	82.54	85.23	95.93	96.20	85.30	86.85

† Average accuracy in percent over 100 runs on the test set.
‡ Best accuracy in percent over 100 runs on the test set.

4.4 Comparison of CXGAs with Different Parent Sizes

In this section, we examine the performance of CXGAs on a spectrum of different degrees of central area exploitation. We denote by $\text{CXGA}_{n_{\text{parent}}}$ the CXGA with n_{parent} parents. We set the number of parents n_{parent} to be 2 through 100 in the experiments. One hundred trials were conducted for each CXGA.

Table 2 shows the accuracies of CXGAs on the test sets in percent. CXGA_{80}, CXGA_{40}, and CXGA_{20} were the best for CHDD, WBCD, and ACCAD, respectively. The best numbers of parents were different depending on problems. Overall, the results show that too low or too high exploitation of the central areas is not desirable.

5 Conclusion

Kim and Moon [15] examined central areas of local optima for the graph bipartitioning problem, and showed that it was attractive to exploit central areas. We extended their study to the field of neural networks. We represented a neural network as a weight matrix, and regarded it as a point in a multi-dimensional problem space. The proposed crossover computes the exact central point of parents. Overall, our GA outperformed the multi-start heuristic and a traditional neuro-genetic hybrid. The experimental results showed that it was attractive to exploit central areas of local optima, and that too low or too high exploitation was not desirable.

We also defined a new distance measure for neural networks. The semantic distance between two networks could be computed fast by the Hungarian method. By normalizing neural networks on the basis of the distance measure, we avoided the permutation problem that occurs in evolving neural networks.

Acknowledgments. This study was supported by a grant of the International Mobile Telecommunications 2000 R&D Project, Ministry of Information & Communication, Republic of Korea. This study was also supported by Brain Korea 21 Project. The Institute of Computer Technology at Seoul National University provided research facilities for this study. Y.-H. Kim contributed to the definition of the distance measure for neural networks.

References

1. R. K. Belew, J. McInerney, and N. N. Schraudolph. Evolving networks: Using the genetic algorithm with connectionist learning. In C. G. Langton, C. Taylor, J. D. Farmer, and S. Rasmussen, editors, *Artificial Life II*, pages 511–547. Addison-Wesley, Redwood City, CA, 1992.
2. C. L. Blake and C. J. Merz. UCI repository of machine learning databases, 1998.
3. K. D. Boese, A. B. Kahng, and S. Muddu. A new adaptive multi-start technique for combinatorial global optimizations. *Operations Research Letters*, 15:101–113, 1994.
4. T. N. Bui and B. R. Moon. A new genetic approach for the traveling salesman problem. In *IEEE Conference on Evolutionary Computation*, pages 7–12, 1994.
5. D. J. Chalmers. The evolution of learning: An experiment in genetic connectionism. In *Proceedings of the 1990 Connectionist Models Summer School*, pages 81–90, 1990.
6. S. S. Choi and B. R. Moon. Normalization in genetic algorithms. In *Genetic and Evolutionary Computation Conference*, pages 862–873, 2003.
7. D. Crosher. The artificial evolution of a generalized class of adaptive processes. In *Preprints of AI'93 Workshop on Evolutionary Computation*, pages 18–36, 1993.
8. D. Gale. *The Theory of Linear Economic Models*. McGraw-Hill Book Company, Inc., 1960.
9. P. J. B. Hancock. Genetic algorithms and permutation problems: a comparison of recombination operators for neural net structure specification. In *Proc. Int. Workshop Combinations of Genetic Algorithms and Neural Networks*, pages 108–122, 1992.
10. S. Haykin. *Neural Networks, A Comprehensive Foundation*. Prentice Hall, 1999.
11. A. James and M. David. *Neural Networks, Algorithms, Applications, and Programming Techniques*. Addison Wesley, 1994.
12. T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. 1995. To appear in *Sixth International Conference on Genetic Algorithms*.
13. S. Kauffman. Adaptation on rugged fitness landscapes. *Lectures in the Science of Complexity*, pages 527–618, 1989.
14. H. B. Kim, S. H. Jung, T. G. Kim, and K. H. Park. Fast learning method for back-propagation neural network by evolutionary adaptation of learning rates. *Neurocomputating*.
15. Y. H. Kim and B. R. Moon. Investigation of the fitness landscapes and multi-parent crossover for graph bipartitioning. In *Genetic and Evolutionary Computation Conference*, pages 1123–1134, 2003.
16. J. R. Koza and J. P. Rice. Genetic generation of both the weights and architecture for a neural network. In *IEEE Int. Joint Conf. Neural Networks*, pages 71–76, 1991.

17. H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Res. Logist. Quart.*, 2:83–97, 1955.
18. C. T. Lin and C. P. Jou. Controlling chaos by GA-based reinforcement learning neural network. *IEEE Trans. on Neural Networks*, 10(4):846–869, 1999.
19. Y. Liu and X. Yao. Evolutionary design of artificial neural networks with different nodes. In *IEEE Conference on Evolutionary Computation*, pages 670–675, 1996.
20. B. Manderick, M. de Weger, and P. Spiessens. The genetic algorithm and the structure of the fitness landscape. In *International Conference on Genetic Algorithms*, pages 143–150, 1991.
21. O. L. Mangasarian and W. H. Wolberg. Cancer diagnosis via linear programming. *SIAM News*, 23(5):1–18, 1990.
22. Z. Michalewicz. *Genetic Algorithms + Data Structure = Evolution Programs.* Springer-Verlag, Berlin, 1992.
23. D. Montana and L. Davis. Training feedforward neural network using genetic algorithms. In *11th International Joint Conference on Artificial Intelligence*, pages 762–767, 1989.
24. V. Petridis, E. Paterakis, and A. Kehagias. A hybrid neural-genetic multimodel parameter estimation algorithm. *IEEE Trans. on Neural Networks*, 9(5):862–876, 1998.
25. J. Pujol and R. Poli. Evolving neural networks using a dual representation with a combined crossover operator. In *IEEE Conference on Evolutionary Computation*, pages 416–421, 1998.
26. N. J. Radcliffe. Forma analysis and random respectful recombination. In *International Conference on Genetic Algorithms*, pages 222–229, 1991.
27. R. S. Sexton, R. E. Dorsey, and J. D. Johnson. Toward global optimization of neural networks: A comparison of the genetic algorithm and backpropagation. *Decision Support Systems*, 22(2):171–185, 1998.
28. G. B. Sorkin. Efficient simulated annealing on fractal landscapes. *Algorithmica*, 6:367–418, 1991.
29. D. Thierens. Non-redundant genetic coding of neural networks. In *IEEE Conference on Evolutionary Computation*, pages 571–575, 1996.
30. E. D. Weinberger. Fourier and Taylor series on fitness landscapes. *Biological Cybernetics*, 65:321–330, 1991.
31. D. Whitley, T. Starkweather, and C. Bogart. Genetic algorithms and neural networks: Optimizing connections and connectivity. *Parallel Computing*, 14(3):347–361, 1990.
32. X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.