Neural Comput & Applic (1993)1:32–45

Neural
Computing
& Applications

Genetic Algorithms and Their Applications to the Design of Neural Networks

Antonia J. Jones

Department of Computing, Imperial College, London, UK

Keywords: Diffusion genetic algorithm; Evolutionary algorithms; Genetic algorithms; Network representation; Parallel genetic algorithms; Permutational redundancy; Premature convergence; Stochastic equilibrium; Travelling salesman problem

1. Review of the Ideas

Genetic algorithms, first introduced by Holland [1], have been applied to a variety of problems, and offer intriguing possibilities for general purpose adaptive search algorithms in artificial intelligence, especially, but not necessarily, for situations where it is difficult or impossible to model precisely the external circumstances faced by the program. Search based on evolutionary models had, of course, been tried before Holland. However, these models were based on mutation and natural selection, and were not notably successful. The principal difference of Holland's approach was the incorporation of a 'crossover' operator to mimic the effect of sexual reproduction.

From another perspective, GAs fall into the class of probabilistic heuristic algorithms which one might use to attack NP-complete or NP-hard problems (see, for example Horowitz [2, Chapters 11 and 12]), such as the *Travelling Salesperson Problem* (TSP), many of which have significant applications in engineering hardware or software design and commerical optimisation problems.

In this article we assume that the reader is familiar with the basic ideas of neural networks but perhaps less conversant with genetic algorithms. The aim is to describe the basic ideas of GAs, and then to survey their application to the design of neural networks. The basic idea of a GA is illustrated in Fig. 1.

We seek to optimise members of a population of 'structures'. These structures are encoded in some manner by a 'gene string'. The population is then 'evolved' in a very stylised version of the evolutionary process.

We are given a set A of 'structures' which we can think of, in the first instance, as being a set of strings of fixed length l. The object of the adaptive search is to find a structure which performs well in terms of a measure of performance:

$$v : A \rightarrow \mathbb{R}^+$$

where $\mathbb{R}^+$ denotes the positive real numbers.

The programmer must provide a representation for the structures to be optimised. In the terminology of GAs a particular structure is called a *phenotype*, and its representation as a string is called a

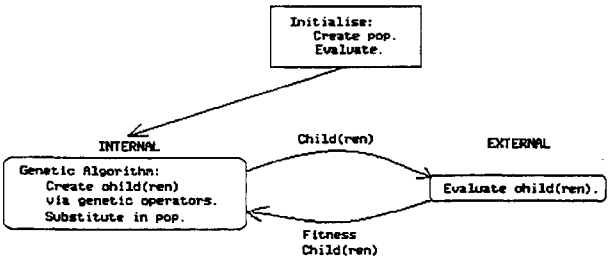

Fig. 1. Generic model for a GA.

Original manuscript received 19 February 1992

Correspondence and offprint requests to: Antonia J. Jones, Department of Computing, Imperial College of Science, Technology and Medicine, University of London, 180 Queen's Gate, London SW7 2BZ, UK.

chromosome or *genotype*. Usually, this representation consists of a fixed length string in which each component, or gene, may take only a small range of values, or *alleles*. In this context, 'small' often means two, so that binary strings are used for the genotypes.

There is nothing obligatory in taking a one-bit range for each allele, but there are theoretical reasons to prefer few-alleles-at-many-sites over many-alleles-at-few-sites (the arguments have been given by Holland [1, p 71], and Smith [3, p 56], and supporting evidence for the correctness of these arguments has been presented by Schaffer [4, p 107]).

The function v provides a measure of 'fitness' for a given phenotype and (since the programmer must also supply a mapping φ from the set of genotypes to the set of phenotypes) hence for a given genotype. Given a particular genotype or string, the goal function provides a means for calculating the probability that the string will be selected to contribute to the next generation. It should be noted that the composition function $v(\varphi)$ mapping genotypes to fitness is invariably discontinuous; nevertheless, GAs cope remarkably well with this difficulty.

'Fit' strings, i.e. strings having larger goal function values, will be more likely to be selected, but all members of the population will have some chance to contribute. Typically, the evaluation of the goal function for a particular phenotype, a process which strictly speaking is external to the GA itself, is the most time consuming aspect of the computation.

Alleles interact so that *adaptation becomes primarily the search for co-adapted sets of alleles*. In the environment against which the organism is tested, any individual exemplifies a large number of possible 'patterns of co-adapted alleles', or *schemata* as Holland calls them. In testing this individual we shall see that all schemata of which the individual is an instantiation are also tested. If the rules whereby genes are combined have a tendency to generate new instances of above average schemata, then the resulting adaptive system has a high degree of 'intrinsic parallelism'.[1] Considerations of this type offer an explanation of how evolution can proceed at all. If a simple enumerative plan were employed, and if 10^{12} structures could be tried every second it would take a time vastly exceeding the estimated age of the universe to test 10^{100} structures.

Figure 2 shows a sketch of the standard style GA. Given the mapping from genotype to phenotype, the goal function, and an initial random population,

1. Randomly generate a population of M structures
   $S(0) = \{s(1,0), \ldots, s(M,0)\}$
2. For each new string $s(i,t)$ in $S(t)$, compute and save its measure of utility $v(s(i,t))$.
3. For each $s(i,t)$ in $S(t)$ compute the selection probability defined by
   $p(i,t) = v(s(i,t))/(\Sigma_i\, v(s(i,t)))$
4. Generate a new population $S(t+1)$ by selecting structures from $S(t)$ via the selection probability distribution and applying the idealised genetic operators to the structures generated.
5. Go to 2.

Fig. 2. Algorithm 1.

the GA proceeds to create new members of the population (which progressively replace the old members) using genetic operators, typically mutation, crossover and inversion, modelled on their biological analogs.

For the moment we represent strings as:

$$a_1a_2a_3 \ldots a_l \quad [a_i = 1 \text{ or } 0]$$

Using this notation we can describe the operators by which strings are combined to produce new strings. It is the choice of these operators which produces a search strategy that exploits coadapted sets of structural components already discovered. Holland uses three such principal operators: *Crossover*, *Mutation*, and *Inversion*.

Crossover: in crossover one or two cut points are selected at random, and the operation illustrated in Fig. 3 is used to create two children. A variety of control regimes are possible, but we used the simplest, *viz* select one of the children at random to go into the next generation. Children tend to be 'like' their parents so that crossover can be considered as a focusing operator which exploits knowledge already gained, its effects are quite quickly apparent.

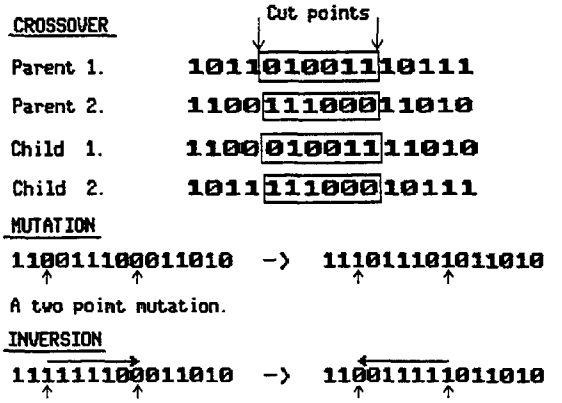

Fig. 3. Standard GA operators.

[1] The notion of 'intrinsic parallelism' will be discussed, but it should be mentioned that it has nothing to do with parallelism in the sense normally intended in computing.

Crossing over proceeds in three steps:

1. Two structures $a_1 . . . a_l$ and $b_1 . . . b_l$ are selected at random from the current population
2. A crossover point x, in the range 1 to $l-1$ is selected, again at random
3. Two new structures:

$$a_1 a_2 . . . a_x b_{x+1} b_{x+2} . . . b_l$$

$$b_1 b_2 . . . b_x a_{x+1} a_{x+2} . . . a_l$$

are formed

In modifying the pool of schemata (discussed below), crossing over continually introduces new schemata for trial while testing extant schemata in new contexts. It can be shown that each crossing over affects a great number of schemata.

There is large variation in the crossover operators which have been used by different experimenters. For example, it is possible to cross at more than one point. The extreme case of this is where each allele is randomly selected from one or other parent string with uniform probability – this is called *uniform crossover*. Although some writers have argued in favour of uniform crossover, there would seem to be theoretical arguments against its use, *viz.* if evolution is the search for co-adapted sets of alleles then this search is likely to be severely undermined if many cut points are used. In language we shall shortly develop, the probability of schemata disruption when using uniform crossover will be much higher than when using one or two point crossover.

The design of the crossover operator is strongly influenced by the nature of the representation. For example, if the problem is the TSP and the representation of a tour is a straightforward list of cities in the order in which they are to be visited, then a simple crossover operator will, in general, not produce a tour. In this case the options are:

- Change the representation.
- Modify the crossover operator.
- Effect 'genetic repair' on non-tours which may result.

There is obviously much scope for experiment for any particular problem. The danger is that the resulting algorithm may be so far removed from the original canonical Holland form that an analogous schemata theorem may not apply – in which case the whole justification for the method will have been lost.

Mutation: in mutation an allele is altered at each site with some fixed probability; thus the number of genes altered in a mutation of a long string will be according to a Poisson distribution. Mutation disperses the population throughout the search space, and so might be considered as an information gathering or exploration operator. Search by mutation is a slow process analogous to exhaustive search.

Each structure $a_1 a_2 . . . a_l$ in the population is operated upon as follows. Position x is modified, with probability p independent of the other positions, so that the string is replaced by:

$$a_1 a_2 . . . a_{x-1} z a_{x+1} . . . a_l$$

where z is drawn at random from the possible values. If p is the probability of mutation at a single position, then the probability of h mutations in a given string is determined by a Poisson distribution with parameter p. Mutation is a 'background' operator, assuring that the crossover operator has a full range of alleles so that the adaptive plan is not trapped on local optima.

Inversion:[2] before we can explain the effects of inversion we have to modify the string representation to be order-free. This means that the order of alleles in the string should not have any effect on the genotypical information contained within the string. It turns out that without such an order-free representation, inversion would be nothing more nor less than a rather brutal mutation.

We can create an order-free representation by redefining alleles as ordered pairs (a_i, P_i), in which P_i is an integer, $1 \leq P_i \leq l$, and P_i denotes the *position* of the allele a_i in the canonical (i.e. standard) representation. Thus, for example, the string $(a_1,2)(a_2,4)(a_3,1)(a_4,3)$ in this new representation, maps to the canonical string $a_3 a_1 a_4 a_2$ in the original representation. Note that, for any string $(P_1, P_2, . . ., P_l)$ is a permutation of $(1,2, . . ., l)$.

Considering the ordered pairs as units, inversion acts as follows. For some randomly selected positions $x < y$ in the string we perform the transformation:

$$(a_1,P_1)(a_2,P_2) . . . (a_l,P_l)$$

$$\rightarrow (a_1,P_1) . . . (a_x,P_x)(a_{y-1},P_{y-1})(a_{y-2},P_{y-2})$$

$$. . . (a_{x+1},P_{x+1})(a_y,P_y) . . . (a_l,P_l)$$

Thus the effect of an inversion is to reverse the order of the ordered pair alleles between $x+1$ and

[2] This explanation of inversion can readily be omitted on a first reading without losing the essential point of the genetic operators. However, the role of inversion has often been misunderstood (despite the fact that it was clearly explained by Holland), so that the reader might plan on returning to this section at a later stage.

$y-1$. A moment's thought reveals that this has no effect whatsoever on the genotypical information (i.e. the individual produced by this new string is identical to the individual produced by the original string). Plainly, inversion by itself can accomplish nothing. So what is the point of inversion? Before answering this question we need to describe how crossover operates with the new order-free representation. (Mutation acts just as before – changing the value of an a_i and having no effect on the associated P_i.)

As previously, the ordered pair alleles (a_i,P_i) are treated as indivisible units and the cut point(s) for crossover are always chosen *between* ordered pairs. To perform a crossover on the new representation we rearrange the second parent so that it is *homologous* to the first parent (literally, this means the two strings now have 'the same shape'). An example should serve to illustrate the process.

Example (homologous crossover)
Suppose the two strings are:

Parent 1: $(a_1,3)(a_2,1)\,(a_3,2)(a_4,4)$
Parent 2: $(b_1,2)(b_2,3)\,(b_3,4)(b_4,1)$

We first re-arrange the second string so that the second component in each ordered pair (i.e. the 'position indicators') line up with those in the first parent:

Parent 1: $(a_1,3)(a_2,1)\,(a_3,2)(a_4,4)$
Parent 2: $(b_2,3)(b_4,1)\,(b_1,2)(b_3,4)$

The second string is now homologous to the first. Now suppose the cut-point is taken at the second position. Then the possible children are:

Child 1: $(a_1,3)(a_2,1)\,(b_1,2)Pb_3,4)$
Child 2: $(b_2,3)(b_4,1)\,(a_3,2)(a_4,4)$

Note that both inherit the ordering of the first parent.

To understand what can be accomplished by a combination of inversion and crossover we recall the earlier description of evolution as 'the search for co-adapted sets of alleles'. Suppose now that we have found a string for which a particular pair of alleles are co-adapted and therefore contribute to the construction of a particularly fit phenotype from this genotype. This co-adapted pair may well be separated by a considerable distance along the string. Plainly, under these circumstances the probability that a crossover cut-point will separate the co-adapted pair is high.

On the other hand, the fact that this particular string has a high fitness means (as a consequence of the design of the GA) that it is likely to produce many offspring. Suppose that in the course of reproduction now and again an inversion occurs. In some of the child strings so produced the co-adapted pair will be present, and because of the inversion the distance between them may well be reduced. This child string has a high fitness, and an additional advantage that the co-adapted pair are less liable to be separated by a crossover. Moreover, the descendants of this child will inherit the new ordering (at least until another inversion takes place). So this rather abstract genotypical fact nevertheless confers a significant and real benefit. Note that the effectiveness of this process is reduced if the crossover operator has more than one cut point; the extreme case being uniform crossover, where we would predict that inversion would have no effect whatsoever.

Inversion has not been much used in the GA literature. Part of the reason for this may be the extra computational cost of an order-free representation; another may be that the mechanism of inversion seems not to be widely understood.

To summarise: with an order-free representation inversion increases the effectiveness of crossover by promoting close linkage between successful alleles. Linkage occurs when co-adapted alleles are close together in the genotype, thus reducing the probability that the group wil be separated by crossover. This requires an order free string representation and a mechanism for making strings homologous before crossover (see Holland [1, p 107–9]. The effects of inversion are only apparent over a relatively long time scale, i.e. a large number of generations.

Two other choices we must make for a GA concern the *birth* and *death* strategies. Is it advantageous to have several children per generation, or only one? Recent studies by Yuval Davidor suggest that one child per generation is optimal. What policy should be adopted concerning the selection of the strings to be replaced? Some experiments in controlling consanguinity between partners in crossover have also been carried out on the grounds that this may be another useful mechanism for preventing premature convergence. In general, the death or replacement policy has emerged as a useful mechanism of adding a controlled selection pressure. For example, one often used replacement rule is: *replace the least fit string in the current population*. As well as adding a selection pressure, this rule has the additional advantage of never deleting the best string found so far.

2. Schemata and Intrinsic Parallelism

Let A be the set of all strings. Representing strings as:

$$a_1a_2a_3. \ . \ .a_l \quad [a_i = 1 \text{ or } 0]$$

we can designate subsets of A which have attributes in common; these are called *schemata* (or hyperplane schemata), by using $*$ for 'don't care' in one or more positions. For example, $a_1 * a_3 * \ldots *$ represents the schemata of all strings with first element a_1 and third element a_3, all other elements being arbitrary.

In general, the a_i may take one of k values. Thus, including the $*$ symbol, the total number of characters possible in any one position is $k+1$. Since the string is of length l there are $(k+1)^l$ possible schemata. If $k = 2$ and $l = 20$, this is around 3.48×10^9 schemata.

Taking any particular string we may replace any r genes ($0 \leq r \leq l$) by the $*$ symbols to create a schemata for which the string is an instantiation, and all possible such schemata are created in this way. We can do this in:

$$\sum_{r=0}^{l} \binom{l}{r} = \binom{l}{0} + \binom{l}{1} + \ldots + \binom{l}{l} = (1+1)^l = 2^l \tag{1}$$

ways using the Binomial theorem.

Thus any particular string is an instantiation of 2^l schemata; if $k = 2$ and $l = 20$ this around 1.04×10^6. Hence the number of schemata sampled in a population of N strings will not be more than $N2^l$. Although this is a small fraction of the number of possible schemata, for increasing l it is still exponentially larger then the population size N. The ratio of the maximum number of schemata instantiated by strings in the population over the total number of schemata is given by:

$$N\left(\frac{2}{k+1}\right)^l \tag{2}$$

and so tends to zero as $l \rightarrow \infty$. Note that this ratio is greatest, i.e. most favourable, when k is small.

The key to understanding why genetic algorithms provide such an efficient search mechanism is the observation that evaluation of just one string yields information about a large number of schemata. This is called *intrinsic parallelism*, and we next examine how this comes about.

We can define the observed fitness of a schemata ξ at time t as the average:

$$v(\xi,t) = \frac{1}{N(\xi,t)} \sum_{s \in I} v(s) \tag{3}$$

where if Pop_t is the population at time t then $I = \xi \cap \text{Pop}_t$ and $N(\xi,t)$ is the total number of strings in I. The fraction of Pop_t which are in ξ is just:

$$P(\xi, t) = \frac{N(\xi, t)}{N} \tag{4}$$

where $N = |\text{Pop}_t|$. Of course, what we should like to know is the actual average fitness of a schemata across the entire possible population. In practice, the number of strings in a given schemata is so huge that it is normally quite impractical to calculate this information; hence the intersection with Pop_t yields the *observed* fitness.

The following theorem, stated for the case where pure crossover is used, helps to explain why successful structures emerge surprisingly rapidly:

Schemata theorem (Holland). *The number of strings in the population belonging to a given schemata can be expected to increase or decrease over time at a rate directly proportional to the observed performance of the schemata (– implicit parallelism).*

More formally;

$$P(\xi,t+1) \geq \left(1 - p_c \frac{l(\xi)}{l-1}(1 - P(\xi,t))\right) \frac{v(\xi,t)}{\bar{v}(t)} P(\xi,t) \tag{5}$$

where p_c denotes the proportion of individuals undergoing crossover during a generation, and $\bar{v}(t)$ is the average fitness of the entire population. The defining length $l(\xi)$ of a schemata is the length of the string from the first to the last locus that is *not* a $*$, and it varies from 0 to $l-1$. Thus the probability of a single cut point falling into the part of a string which will disrupt ξ is $l(\xi)/(l-1)$.

The anatomy of this theorem is as follows. The product of the terms outside the large brackets is the probability that a parent chosen according to the selection probabilities will be a member of ξ. The term in the large brackets is the probability that a child of this parent will not have its membership of ξ disrupted by the crossover operator, and will therefore remain in ξ in the next generation. This is simply one minus the probability of disruption. The probability of disruption is the product of the probabilities of three independent events; that crossover will occur p_c; that it will fall within the defining length of the schemata $l(\xi)/(l-1)$; and that the mate of the parent who is a member of ξ is not also a member of ξ, $1 - P(\xi,t)$. This last term comes from the selection of mates with a uniform probability, and the fact that a crossover

event between two members of ξ cannot fail to produce offspring who are members of ξ. Finally, it is noted that the representation of ξ in the next generation is not limited to the offspring of the members in the current generation. Crossover between parents who are not members of ξ may produce offspring who are – hence the greater-than-or-equal relation.

It is important to note that the proof of the theorem rests on a very precise prescription of the algorithm. But given the result then *it applies to all schemata simultaneously* – implicit parallelism. The point to grasp is that although in the first instance the GA paradigm is a search, one at a time, through the space of strings, in fact, the process yields an intrinsically parallel search through the much larger space of schemata.

To summarise: the main advantages of the GA adaptive strategy are:

- It concentrates samples increasingly towards schemata that contain structures of above average utility.
- Since it works over a knowledge base (i.e. the population of structures) that is distributed over the search space, it is all but immune to getting trapped on local optima (provided the population is sufficiently large).

Schemata were invented as a conceptual tool to explain how it is that genetic algorithms with crossover work as well as they indeed do. The fact is that the theory provided by Holland is mathematically incomplete, except for the simplest cases, and further developments in the theory, which would also be important in genetics, are required before we can truly claim a good understanding of the mechanisms of crossover.

3. Design Issues – What Do You Want the Algorithm to Do?

Now we have to ask just what it is we want of a GA. There are several, sometimes mutually exclusive, possibilities. For example:

- Rapid convergence to a global optimum.
- Produce a diverse population of near optimal solutions in different 'niches'.
- Be adaptive in 'real-time' to changes in the goal function.

We shall deal with each of these in turn, but first let us briefly consider the nature of the search space. If the space is flat with just one spike then no algorithm short of exhaustive search will suffice. If the space is smooth and unimodal then a conventional hill-climbing technique should be used. Somewhere between these two extremes are problems in which the goal function is a highly non-linear multimodal function of the gene values – these are the problems of hard combinatoric search for which some style of GA may be appropriate.

3.1. Rapid Convergence to a Global Optimum

Of course, this is rather simplisitic: Holland's theory holds for large populations. However, in many AI applications it is computationally infeasible to use large populations, and this in turn leads to a problem commonly referred to as *premature convergnce* (to a suboptimal solution) or *loss of diversity* in the literature of GAs. When this occurs the population tends to become dominated by one relatively good solution, and locked into a suboptimal region of the search space. For *small* populations the schemata theorem is actually an explanation for premature convergence (i.e. the failure of the algorithm) rather than a result which explains success.

Premature convergence is related to a phenomenon observed in nature. Allelic frequencies may fluctuate purely by chance about their mean from one generation to another; this is termed *random genetic drift*. Its effect on the gene pool in a large population is negligible, but in a small effectively interbreeding population, chance alteration in Mendelian ratios can have a significant effect on gene frequencies, and can lead to the fixation of one allele and loss of another. For example, isolated communities within a given population have been found to have frequencies for blood group alleles different from the population as a whole. Figure 4 illustrates this phenomenon with a simple function optimisation GA.

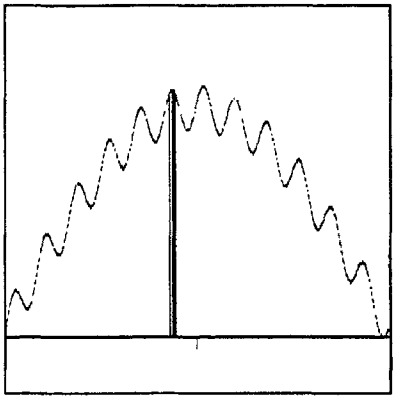

Fig. 4. Premature convergence – no sharing.

The inexperienced often tend to attempt to counteract premature convergence by increasing the rate of mutation. However, this is not a good idea. A high rate of mutation tends to devalue the role of crossover in building co-adapted sets of alleles, and in essence pushes the algorithm in the direction of exhaustive search. Whilst some mutation is necessary, a high rate of mutation is invariably counter-productive.

In trying to counteract premature convergence we are essentially trying to balance the *exploitation* of good solutions found so far against the *exploration* which is required to find hitherto unkown promising regions of the search space. It is worth observing that, in computational terms, any algorithm which often inserts copies of strings into the current population is wasteful. This is true for the Traditional Genetic Algorithm (TGA) outlined as Algorithm 1 in Fig. 2.

3.2. Produce a Diverse Population of Near Optimal Solutions in Different 'Niches'

The problem of premature convergence has been addressed by a number of authors using a diversity of techniques. Many of the papers in Davis [5] contain discussions of precisely this point. The methods used to combat premature convergence in TGAs are not necessarily appropriate to the parallel formulations of GAs (PGA), which we shall discuss shortly.

Cavicchio [6], in his doctoral dissertation, suggested a *preselection* mechanism as a means of promoting genotype diversity. Preselection filters children generated, possible picking the fittest, and replaces parent members of the population with their offspring.

De Jong's [7] *crowding* scheme is an elaboration of the preselection mechanism. In the crowding scheme, an offspring replaces the most similar string from a randomly drawn subpopulation having size CF (the crowding factor) of the current population. Thus a member of the population experiences a selection pressure in proportion to its similarity to other members of the population. Emperical determination of CF with a five function test bed determined CF = 3 as optimal.

Booker [8] implemented a *sharing* method in a classifier system environment which used the bucket brigade algorithm. The idea here was that if related rules share payments then subpopulations of rules will form naturally. However, it seems difficult to apply this mechanism to standard GAs. Schaffer [4] has extended the idea of subpopulations in his *VEGA* model, in which each fitness element has its own subpopulation.

A different approach to help maintain genotype diversity was introduced by Mauldin [9] via his *uniqueness* operator. The uniqueness operator helped to maintain diversity by incorporating a 'censorship' operator in which the insertion of an offspring into the population is possible only if the offspring is genotypically different from all members of the population at a number of specified genotypical loci.

Somewhat later, Goldberg introduced the idea of *shared value* [10]. The idea of *sharing* is very simple: if a number of strings in the current population cluster closely together, where 'closeness' can refer to the topology either of the space of phenotypes or that of genotypes, then their goal function values might be expected to be similar. If the goal function is large in this region, then a simple GA would give each string of the cluster a large selection probability, thus reinforcing the tendency for succeeding generations to further cluster in the same region – an effect which often leads not to an exhaustive examination of the optimal region, but to premature convergence.

Goldberg proposed that if two or more strings were closer together than a specified 'niche width' then their goal function values should be shared between the group in some way. Let the ith member of the population have phenotype x_i and the population size be n. Then the niche count N_i of the ith member is a real number in the range $[1,n]$ computed on the basis of a characteristic function for the niche.[3] Note that $N_i = 1$ if the string is isolated, as measured by the specified niche width, and $N_i = n$ if the niche width is taken so large that all members of the population are counted as being as being in the niche. The shared value of the ith string is taken to be:

$$V_S(x_i) = \frac{v(x_i)}{N_i} \tag{6}$$

where $v(x)$ is the usual fitness or goal function. It is the shared value which is used to compute the selection probabilities. Specifically, the probability of selecting the ith string is:

$$P_i = \frac{V_S(x_i)}{\sum_j V_S(x_j)} \tag{7}$$

[3] This is one aspect of sharing which is rather *ad hoc* – specification of the niche width requires *a priori* knowledge of the search space, and to progressively refine the search the niche width must dynamically shrink at the right rate.

Thus as more strings cluster in a specific region of a large goal function, their relative advantage in enhanced selection probability over other members of the population is correspondingly reduced. If a new good string is discovered in a hitherto unexplored region (thus having a low niche count), then its selection probability will be exceptionally high, thereby ensuring the rapid formation of a new, well populated niche. This ensures the rapid switching of attention, or adaptability, clearly a desirable property.

Goldberg does not explicitly state his replacement strategy, but by simple experiments one finds that using the rule 'child replaces string with least shared value', it is a straightforward matter to reproduce his results precisely. In experiments one finds that after 100 generations with pure crossover, good-niche clustering has taken place and the selection probabilities based on shared value are virtual equal across the population. At this point an extremely stable stochastic equilibrium is in force (see Fig. 5). The stability can be judged from the observation that if sharing is now removed ($N_i = 1$ for all i), the overall distribution of the population in phenotype space remains essentially unchanged for several hundred generations. Essentially the same results are obtained using genotype sharing, except that genotype sharing is more reliable (it distinguishes between 1000 and 0111, for example).

It is not so much that Goldberg sharing is a particularly desirable technique for the production of niches in a real algorithm – what is more interesting about this result is that this simple and intuitive technique so nicely illustrates the existance of a stable stochastic equilibrium. The principal moral we draw from these observations is that *a system that is in stable equilibrium is a system that can be controlled.*

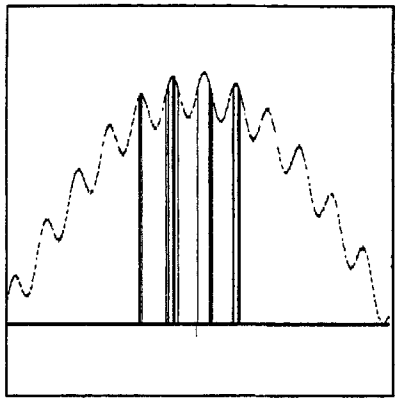

Fig. 5. Stochastic equilibrium with stable niches using Goldberg sharing.

Mutation can be added to ensure detailed search of the established niches, and additional selection pressures such as replacing the least fit string can ensure that the least valuable niches are gradually evacuated. The actual effect of sharing can be simulated by less costly techniques such as Cavicchio preselection.

3.3. Be Adaptive in 'Real-time' to Changes in the Goal Function

For any realistically complex problem it seems unreasonable to ask 'real-time' performance of GAs. In nature it certainly is the case that the goal function, insofar as this is anything other than a useful abstraction of a very large number of factors, is a time varying function. A species that cannot adapt to change, or that does not encompass sufficient diversity, is certainly doomed. However, regardless of the algorithm's design and hardware, the fact remains that a well designed GA addressing a difficult search problem must be expected to run for a large number of generations (typically in excess of a thousand) to produce useful results. If useful results are obtained in 10–300 generations this suggests that the problem is not really very hard in some sense, and that some other, possibly simpler, algorithm might do just as well, if not better.

4. Massively Parallel Evolutionary Algorithms

In the work of the group led by Heinz Muhlenbein at GMD, Bonn, a new class of algorithms is emerging which derive some of their important features from architectural considerations; he calls these *evolutionary algorithms*.

The architecture used is a 64-transputer bank (although it is planned to extend the system) with configurable links. The underlying idea is to use genetic algorithms and associate one member of the population with each transputer. It has been found that good results are obtained if each member of the population is allowed to do some 'local hill climbing' before proceeding to crossover. However even with this model, loss of population diversity, with the associated problems of premature convergence, is still experienced.

The model was further modified by Martina Gorges-Schleuter [11], so that instead of choosing a partner for crossover based on selection probabilities computed at a global level, crossover was only permitted between members of small (overlapping)

groups – demes. The topology of these groups is dictated by the transputer link graph selected by the programmer. In many of her experiments a ring-topology was used – exemplified in nature by the Ring-Gull around the Arctic coast.

The Ring-Gull population originated after the last ice age as seagulls returned from their retreat in a region of the Caspian sea. The seagulls' habitat became an annular region around the North Pole. A limited gene flow caused by this long chain of small, poorly linked populations produced sufficient isolation to ensure that different species developed one after the other: *Larus graelsii* (lives in Great Britain), *L. fuscus fuscus* (Scandinavia), *L. argentus vegae* (Siberia), *L. argentus smithoniatus* (North America), and *L. argentus argentus* (again in Britain). The latter cohabits, without interbreeding, with the first, although collectively the Ring-Gulls share a common gene pool.

Some experiments used a ring topology where the immediate neighbours and the next but one are directly connected. This produces a maximally connected regular graph with connectivity 4. Since each transputer can have four physical direct links to other transputers, the ring topology may be immediately mapped onto a transputer system. Other experiments used the topology of Fig. 6 in which overlapping demes are used and an individual's fitness is computed relative to the phenotype values of its respective deme members.

In biological terms this whole approach is attractive, since it is a closer model to nature. In practice, population members do not select their partners from the whole population, but rather from a smaller group locally available. This encourages local groupings of genotype similarities and, provided there is a small level of interbreeding between different local groups, the stage is set for a good distribution of the entire population across a variety of attractive niches. Thus the overall design of the PGA avoids premature convergence rather than the special algorithmic measures that have been applied in TGAs. In general, it is often the mixing of two dissimilar genotypes which produces a radical improvement in the species.

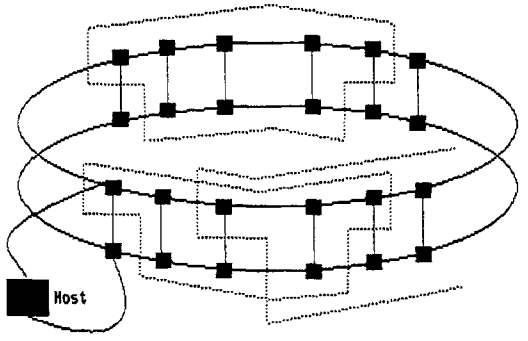

Fig. 6. Transputer link graph showing demes of size 8.

In computational terms the new regime also has many advantages: The communication overhead between processes is reduced, since there is now no need to collect information globally to compute the selection probabilities, and less computation is required. Moreover, when inter-process communication occurs it does so between transputers closely connected, so that message passing is reduced.

This system is able to solve the *Travelling Salesperson Problem* for around 520 cities in about 10 minutes, substantially faster than a Cray. The limitation here in the number of cities was imposed by the amount of available memory per node. Indeed, memory per node was so limited that all distance pairs between cities could not be stored, and so only city co-ordinates were stored locally – which meant that in evaluating a tour length each component edge length had to be calculated from the co-ordinate pairs. This fact slowed the algorithm considerably. Thus in this case, not only did the memory per node limit the problem size, it was also a major constraint on the solution time. Of course, T-800 transputers may each have attached a very large amount of memory – so that this limitation on the research was essentially imposed by financial constraints.[4] It is an intersting question whether or not the Gorges–Scheulter algorithm can be modified to run on a vector-based SIMD architecture like the Cray.

In fact, diffusion GAs *have* been implemented on massively parallel SIMD machines. A diffusion GA with a torus population structure is presented in Manderick and Spiessens [12]. Neighbours are those directly adjacent on the grid. Here selection and crossover in each deme occurs as in a TGA; there is no filtering of children. Using a few test functions, the diffusion GA was demonstrated to search more widely than a TGA, but performance was very much the same. An implementation on the ICL Data Array Processor (DAP) studying the effect of some parameter settings (selection scheme and crossover type) has followed, Spiessens and Manderick [13].

A Connection Machine implementation of a diffusion GA on one and two dimensional grids is discussed in Collins and Jefferson [14]. Selection of parents is done by two random walks of fixed lengh from the location, the best encountered on each

[4] At present, the best results in terms of size/speed for the TSP have been reported by Johnson at Bell Labs, and his algorithms are for a conventional von Neumann architecture.

walk being selected. A multilevel graph partitioning problem was used to compare this local selection with panmictic selection methods. Local selection was found to be more robust, and to find solutions faster. Population sizes used were very large (2^{13}–2^{19}), but despite this panmictic GAs could only discover one of two optimal solutions for the problem, whereas the diffusion model always discovered both. This seems particularly significant, since it is often supposed that problems such as premature convergence in TGAs can be overcome by simply making the population large. Whilst this is true in theory, these results suggest that in practice the population may need to become impractically large to solve the problem in this way.

A theoretical analysis of another diffusion GA using a 2-dimensional grid (similar to Manderick [12]) concludes that local dominance of highly fit schemata causes very fast initial niche formation and exploration of multiple peaks in the search space. These niches expand and compete, leading to one of the niches dominating and taking over the rest.

5. Generation of Neural Networks by Evolutionary Techniques

Since the design of neural networks optimised for a specific application is still very much a research issue, and since, from the evolutionary viewpoint, nature has performed this task magnificently, it is natural to ask to what extent we might use our present knowledge of GAs to design neural networks. Bibliographies of work in this field (Rudnick [15], Weiss [16]) are available up to 1990.

In such a paradigm we might expect the computational cost to be high, but is it possible that from a number of such studies we may begin to extract our own design rules by examining the solutions produced by the GA.

A very early example of such work is that of Jones and Valenzuela *et al.* [17], in which the mapping of a WISARD network applied to speech recognition was optimised for the specific task using GAs. For more conventional neural net models we can initially identify several ways in which we might try to use GAs to help design neural networks optimised for a specific task:

- Given the number of nodes and the connectivity, we might use the GA to determine the weights, i.e. compare the GA learning weights with other learning algorithms, e.g. backpropagation.
- Given the learning rule, we might use the GA to determine the connectivity structure, e.g. for a back-propagation feedforward network we might use the GA to determine an optimal or near optimal network topology including the number of hidden layers, the number of units within each layer, and the interconnecting links.
- Given the number of nodes and the connectivity, we might try to use GAs to determine a suitable learning rule (here the 'task' is learning, and so a number of specific tasks must be used).

Using GAs to learn the weights has been tried on a number of occasions with a moderate degree of success [18,19], but in general we might expect that algorithms such as the simulated annealing of Boltzmann machines, or the back-propagation algorithm for feedforward nets (admittedly this only gives a local solution in weight space) which were customed designed for the problem, might do better. This does not seem a particularly interesting way to apply GAs to neural net design, but it probably should not be completely ignored. For example, the paper by Montana and Davis [20] is particularly interesting.

Using GAs to determine the *number* of hidden layers for a simple pattern classification problem is probably not computationally efficient, because this number is invariably going to be one, two or (at most three). However, given that we have (arbitrarily) set the number of hidden layers, it might be worthwhile to use GAs to determine the number of nodes in each layer.

There have been several papers using GAs to design the overall topology of a network [21–26], and these have yielded interesting results on the optimisation of network design. They have not, however, led to qualitatively new kinds of connectionist processes. Moreover, functionally equivalent networks with different topologies can easily fill up the population with networks which all perform the same task to the same level of competence.

Thus, representations which factor out such equivalences would be desirable. Indeed, this seems to be emerging as one of the principal issues to be addressed by research in the field [20,27]. Apart from an enormous enlargement of the search space arising from permutational redundancy, there are other potentially serious problems which can arise from naive representations. This point is made forcefully by Radcliffe [28]:

> ". . .the danger is exemplified by the case where two equivalent networks (identical up to a relabelling of hidden units) can be recombined to produce a child which is not equivalent to them. This is a phenomenon which is not seen in 'conventional' genetic search (for example, single parameter optimisation) and it has been strongly argued elsewhere the problem is highly detrimental to the effective-

ness of the search process both in the specific context of neural networks (Radcliffe [27], Belew [18], and more generally Radcliffe [29,30]).

Radcliffe develops the notion of *formae* (in their original conception formae were introduced as equivalence classes induced by arbitrary equivalence relations over the search space) to deal with this problem. The counting argument used to suggest that binary representations are to be preferred over those of higher cardinality apply only to the traditional schemata formulation. It is argued that a 'schemata theorem' applies to general formae in exactly the same way as with conventional schemata, provided suitable recombination strategies are applied.

Leaving aside these (rather critical) theoretical considerations, we briefly review some of the recent implementations of genetic search applied to neural networks. Of particular interest are the representations selected and the extent to which the crossover operator designed for the representation is liable to preserve structurally coadapted building blocks. In some cases, the authors commented on the problem of permutational redundancy.

Dodd [21] applied a GA to optimise a structured network for a pattern recognition problem classifying dolphin sounds. The network was called a *spread* network, and consisted of a two dimensional grid of input nodes, an input 'retina', and a number of hidden nodes connected to rectangular patches on the retina. Networks were encoded by a set of parameters that specified the sizes and overlap of patches on the retina. He reports that a standard GA was able to find a network that was superior to any that he had created by hand.

In Miller *et al.* [22] a binary encoding of a connection matrix for a fixed number of nodes is used. This direct encoding was applied to the 2-bit parity problem, and it was noted that, in contrast to the usual solutions, good networks tended to be irregular and have direct connections from inputs to the output node. Similar findings were reported in Whitley [24], where a fully connected feedforward network was trained and then variants with a subset of the connections were searched for using a GA. These results suggest that direct connections can lead to improved training times.

Similarly, in Robbins *et al.* [31] a system is described in which each gene directly determines the presence or absence of a single connection. This gives relatively long strings which are unsuitable for large networks. Two benchmark tasks were used in this work; the XOR, and Wieland's two spirals. Results similar to those in Miller *et al.* [22] were obtained for 2-bit parity. The authors observe that direct connections do not speed up weight adjustment by some form of reinforcement, but rather that the reduction in learning time may be due to an increased flexibility in dealing with weights, possibly (as originally suggested by Miller *et al.* [22]) as a result of symmetry breaking.

A higher level scheme is used in Harp [32], where a more abstract 'blueprint' defines a layered feedforward network. The encoding uses variable length strings that are divided into a number of 'areas'. Each area defines a set of nodes (described by a fixed length binary coded parameter string), and a number of projections from the area to other areas. The number of projections is not fixed. Each area has a spatial organisation defined by the area parameters. Extra 'punctuation' symbols are embedded in the encoded strings so as to allow identification of the start of an area. In this way, the variable length strings can be expressed to form legal networks. Crossover also uses the punctuation to align areas so that fixed regions of the strings correspond. Two problems were considered; a 4×8 digit recognition problem, and the 2-bit parity problem.

The digit recognition problem consisted of correctly identifying fixed representations of the digits 0–9. Strings evolved to solve this problem produced networks that had no hidden nodes, and which could solve the problem perfectly. The experimenters found this surprising, as they were not aware that the problem was linearly separable.

A compact encoding scheme is used in Mjolsness [33], where the connection matrix of a network is specified by recursive application of duplication operators to an initial pattern so as to construct the final matrix. The network search space was defined by lists of these operators. GAs were not employed, but conventional search techniques were used to find solutions for an analogue-to-digital conversion network. It was suggested that the connection

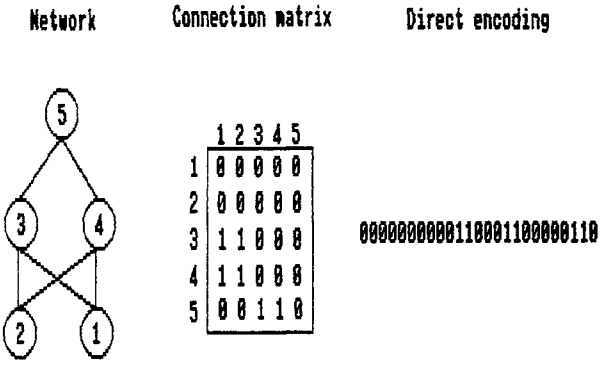

Fig. 7. Direct encoding of a 2-bit parity (XOR) network.

patterns described by the recursive encodings could be *scaled* up to cope with larger instances.

More recent work that uses recursive descriptions of networks [34] describes a context free production grammar that can be used to generate connection matrices. This is a modified L-system [35], where each production rule gives rise to a 2×2 matrix of further production rules or terminal symbols. Rules can either reference themselves or one another, thus introducing recursion. Such systems give rise to structured (at least visually) connection matrices. A GA was used to search for good sets of production rules in 4-×-4 encoder problem. This was compared with a GA using explicitly encoded connection matrices. Kitano claims that the production rule system performed well as the number of hidden nodes was increased, while the solutions for the GA using the explicit encoding scheme degraded. Such production schemes for neural networks are rather attractive, suggesting possibilities for compact representations which may sidestep some of the problems associated with permutational redundancy. Figure 8 shows an example production system that will produce a 2-bit parity solving connection matrix after four steps.

A more complicated developmental system is defined in Gruau [36]. The system starts with a single neuron and then develops (grows) into a full network. A tree structured program controls the development of the network (it can be viewed as the genetic program that each cell inherits). The idea has many appealing properties, and would seem to be the most 'biologically realistic' method so far for generating networks. The system is context sensitive, and can express recursive network topologies very concisely with the addition of several features such as conditional branching, also allowing collections of labelled programs that can be invoked as an operation.

The third possibility, that of evolving the learning rule, is intrinsically more interesting because it

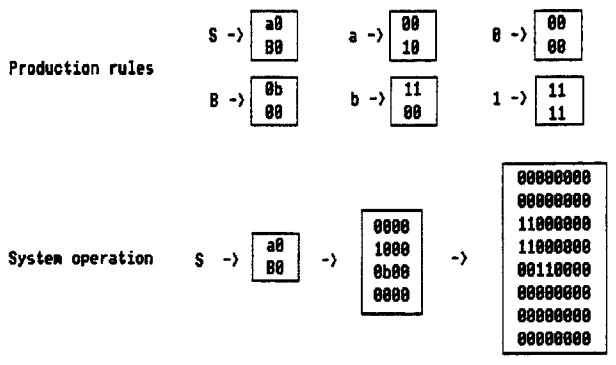

Fig. 8. Production system producing a 2-bit parity network.

begins to explore how an evolutionary process can produce systems that learn. Evolution is considered as a kind of second order adaptation: it is a process that produces individual systems that are not immediately adapted to their environment, but which have the ability to adapt themselves to many environments by the first order adaptive process of learning. Viewed from this perspective, the learning mechanisms themselves are the object of evolution. A vital component of the overall situation which produces learning systems is the environment. If the environment were relatively static, there might be little need for learning systems to evolve.

This is the approach taken in Chalmers [37]. In these experiments, supervised learning was applied to fully-connected, single layer, feed-forward networks with sigmoidal output units, with a built in bias unit to allow for the learning of thresholds. Here a string encodes the weight space *dynamics* of a connectionist system – that is, it encodes a potential learning procedure.

6. What is Wrong with Present GAs?

The TSP is typical of hard combinatoric search. For n cities there are $\frac{1}{2}(n-1)!$ undirected tours. This number rises extremely rapidly as n becomes large. For all known algorithms the time required to determine a minimal length tour thus rises exponentially as the problem size increases.

Suppose we weaken the requirements. Instead of seeking a global optimum, suppose we ask for a near optimal solution with *high probability*. Here the probability measure is defined over the space of all possible problems – for example, over all possible sets of n points chosen from the unit square. Then the situation changes radically, we have the following

Theorem Karp [38][5]., *For every* $\varepsilon > 0$ *there is an algorithm* $A(\varepsilon)$ *such that* $A(\varepsilon)$ *runs in time* $C(\varepsilon)$ $n+O(n \log n)$ *and with probability* 1, $A(\varepsilon)$ *produces a tour of length not more than* $1 + \varepsilon$ *times the length of a minimal tour.*

How is this magic achieved? The answer is by using a basic method in computer science, *viz. divide and conquer.*

The cellular disection algorithm proceeds by breaking the problem into parts (clusters of cities), solving these smaller problems and then gluing the

[5] See also Steele [39].

solutions together. Of course, we can apply this idea recursively as many times as necessary.

As another example, consider the evolution of the nervous system. The speed and function of neurons has not increased much in evolution, once a few tricks like myelination (electrical insulation of the axon) were developed. Rather, the evolutionary process, having developed one kind of electrically active cell useful for information transmission, then went on to develop small circuits of such cells with simple functional significance, and then more complex circuits using the earlier circuits as building blocks. Once this had been accomplished it is arguable that a radical redesign of the basic neural component would be 'counter-productive', since it would probably require the whole process to begin again. Competition between such a new organism and existing more highly developed organisms would tend to reduce the opportunities for neural circuits to develop using the radically new neuron.

We can view the natural evolution of neural circuitry in biology as a kind of 'bottom up' divide and conquer, in which increased performance is obtained by using the results of early search as building blocks for the more sophisticated later solutions.

My feeling regarding existing GAs stems from this perception: *divide and conquer is an essential ingredient of successful (i.e. scalable) combinatoric search algorithms*. As far as I am aware, GAs at present, including the evolutionary algorithms of Muhlenbein and Gorges–Schleuter, lack this essential ingredient, and so, even granted intrinsic parallelism, cannot hope to scale in a reasonable fashion.

On the other hand, there is no reason why the evolutionary approach cannot be equipped with these ideas[6], and it seems that this is one direction in which research in GAs might profitably proceed.

Acknowledgements. I am indebted to many friends and students for drawing my attention to relevant papers that I might well have otherwise overlooked. In particular, I should mention Nick Beard of Price Waterhouse who over the years has sent me dozens of articles and papers of mutual interest, and Donald Macfarlane of Buckingham University who kindly allowed me to see an early version of his thesis.

This work was supported in part by SERC GR/G18391.

[6] Indeed, a little thought suggests a number of different ways in which this might be done.

References

1. Holland JH. Adaptation in natural and artificial systems. Ann Arbor (MI): University of Michigan Press, 1975
2. Horowitz E, Sahni S. Fundamentals of computer algorithms. London: Pitman, 1978
3. Smith SF. A learning system based on genetic adaptive algorithms [dissertation]. Pittsburg (PN): Univ of Pittsburg, 1980
4. Schaffer JD. Some experiments in machine learning using vector evaluated genetic algorithms [dissertation]. Vanderbilt University (unpublished) 1984
5. Davis L, editor. Genetic algorithms and simulated annealing. London: Pitman, 1987
6. Cavicchio DJ. Adaptive search using simulated evolution [dissertation]. Ann Arbor (MI): Univ of Michigan (unpublished), 1970
7. De Jong K. An analysis of the behavior of a class of genetic adaptive systems [dissertation]. Ann Arbor (MI): Univ of Michigan, 1975
8. Booker LB. Intelligent behavior as an adaption to the task environment [dissertation]. Ann Arbor (MI): Univ of Michigan, 1982
9. Mauldin ML. Maintaining diversity in genetic search. In: Proceedings of the National Conference on Artificial Intelligence; 1984: 247–250
10. Goldberg DE, Richardson J. Genetic algorithms with sharing for multimodal function optimization. In: Proceedings of the Second International Conference on Genetic Algorithms; 1987: 41–49; Cambridge (MA)
11. Gorges-Schleuter M. Genetic algorithms and population structures, A massively parallel algorithm [dissertation]. Dortmund (Germany): Univ of Dortmund, Germany, 1990
12. Manderick B, Spiessens P. Fine grained parallel genetic algorithms. In: Proceedings of the Third International Conference on Genetic Algorithms. New York (NY): Morgan Kaufmann, 1989
13. Spiessens O, Manderick B. A massively parallel genetic algorithm – implementation and first analysis. In: Proceedings of the Foruth International Conference on Genetic Algorithms. New York (NY): Morgan Kaufmann, 1991
14. Collins RJ, Jefferson DR. Selection in massively parallel genetic algorithms. In: Proceedings of the Fourth International Conference on Genetic Algorithms; 1991; New York (NY): Morgan Kaufman.
15. Rudnick M. A bibliography of the intersection of genetic search and neural networks. Beaverton (OR): Oregon Graduate Centre, Technical Report CS/E 90-001, 1990
16. Weiss G. Combining neural and evolutionary learning: Aspects and approaches Forschungsberichte Künstliche Intelligenz, Technical Report FKI-132-90, 1990
17. Jones AJ, Valenzuela CL, Badii A, Binstead MJ, Stonham TJ. Applications of *N*-tuple sampling and genetic algorithms to speech recognition. In: Aleksander I, editor. Neural Computing architectures. New York: Kogan Page, 1988
18. Belew R, McInerney J. Schraudolph NN. Evolving networks: Using the genetic algorithm with connectionist learning. San Diego (CA): University of California, CSE Technical Report CS90-174, 1990

19. Wilson S. Perceptron redux. Physica D (forthcoming)
20. Montana DJ, Davis L. Training feedforward neural networks using genetic algorithms. In: Proceedings of the Eleventh International Joint Conference on Artificial Intelligence; 1989: 762–767
21. Dodd, N. Optimization of neural network structures using genetic techniques. Malvern (UK): Royal Signals and Radar Establishment, Internal Report RIP-REP/1000/63/89, 1989
22. Miller G, Todd P, Hegde S. Designing neural networks using genetic algorithms. In: Proceedings of the Third Conference on Genetic Algorithms and their Applications; 1989: San Mateo (CA)
23. Whitley D, Hanson T. Optimising neural networks using faster, more accurate genetic search. In: Proceedings of the Third International Conference on Genetic Algorithms, New York (NY): Morgan Kaufmann, 1989; 391–396
24. Whitley D, Starkweather T, Bogart C. Genetic algorithms and neural networks: optimizing connections and connectivity. Parall Comput 1990; 14
25. Whitley D, Dominic S, Rajarshi D. Genetic reinforcement learning with multilayer neural networks. In: Proceedings of the Fourth International Conference on Genetic Algorithms. New York (NY): Morgan Kaufmann, 1991
26. Dodd N, Macfarlane D, Marland C. Optimisation of artificial network structure using genetic technique implemented on multiple transputers. In: Styles D, Kunii T, Bakkers A, editors. Transputing 91 (Vol 2). Geneva: ISO Press, 1991
27. Radcliffe NJ. Genetic neural networks on MIMD computers [dissertation]. Edinburgh (UK): Univ of Edinburgh, 1990
28. Radcliffe NJ. Genetic set recombination and its application to neural network topology optimisation. Edinburgh (UK): Edinburgh Parallel Computing Centre, Technical Report EPCC-TR-91-21, 1991
29. Radcliffe NJ. Equivalence class analysis of genetic algorithms. Complex Syst 1991; 5(2): 183–205
30. Radcliffe NJ. Forma analysis and random respectful recombination. In: Proceedings of the Fourth International Conference on Genetic Algorithms. New York (NY): Morgan Kaufmann, 1991
31. Robbins GE, Hughes JC, Plumbley MD, Fallside F, Prager R. Generation and adaptation of neural networks by evolutionary techniques (GANNET). Neural Comput & Applic 1993; 1: 22–30
32. Harp SA, Samad T, Guha A. Towards the genetic synthesis of neural networks. In: Proceedings of the Third International Conference on Genetic Algorithms. New York (NY): Morgan Kaufmann, 1989
33. Mjolsness E, Sharp DH, Alpert BK. Scaling, machine learning, and genetic neural nets. Adv Appl Math 1989; 10
34. Kitano H. Designing neural networks using genetic algorithms with graph generation system. Complex Syst 1990; 4: 461–476
35. Lindenmayer A. Developmental systems without cellular interaction, their languages and grammars. J Theor Biol 1971; 30: 455–484
36. Gruau F. Synthèse de réseaux de neurones par algorithme génétique [dissertation]. Lyon (France): Ecole Normale Supérieure de Lyon, 1991
37. Chalmers DJ. The evolution of learning: An experiment in genetic connectionism. In: Proceedings of the Connectionist Models Summer School; 1990; San Marco (CA)
38. Karp RM. Probabilistic analysis of partitioning algorithm for the travelling-salesman problem in the plane. Math Oper Res 1977; 2(3): 209–224
39. Steele JM. Probabilistic algorithm for the directed traveling salesman problem. Math Oper Res 1988; 11(2): 343–350