Monograph:
Chipperfield, A.J. and Fleming, P.J. (1994) Parallel Genetic Algorithms: A Survey. Research Report. ACSE Research Report 518 . Department of Automatic Control and Systems Engineering

Parallel Genetic Algorithms: A Survey

A. J. Chipperfield and P. J. Fleming

Department of Automatic Control and Systems Engineering
University of Sheffield
PO Box 600
Mappin Street
Sheffield S1 4DU

e-mail: A.Chipperfield@shef.ac.uk
P.Fleming@shef.ac.uk

May 24th 1994

ACSE Research Report No. 518

Abstract: This report provides a wide-ranging survey of parallel genetic algorithms and reviews a number of approaches that have been adopted to parallelise them. A tutorial level introduction to genetic algorithms is given and the underlying mechanisms are described and discussed with many current developments reviewed. Three broad classes of parallel genetic algorithms are considered in detail and compared with conventional sequential implementations. It is argued that significant performance benefits can be realised by using distributed population structures and selection mechanisms even when the paradigm is implemented on a sequential machine.

Parallel Genetic Algorithms: A Survey

Andrew Chipperfield and Peter Fleming

Department of Automatic Control and Systems Engineering

University of Sheffield, UK

1 Introduction

Stochastic search and optimization methods, based on the principles of natural biological evolution, have received considerable and increasing interest over the past decade. Introduced in the 1970s by Holland [1], genetic algorithms (GAs) are part of the larger class of evolutionary algorithms that also includes evolution strategies and genetic programming. Compared to traditional optimization methods, such as calculus-based and enumerative strategies, the GA is robust, global and generally more straightforward to apply. In recent years, GAs have been applied to a broad range of problems including ecosystem modelling, combinatorial and parametric optimization, machine intelligence, analysis of complex systems and financial prediction.

This report provides a wide-ranging survey of the current trends and techniques in GAs and reviews a number of approaches that have been adopted to parallelise the GA. Starting with a tutorial level introduction, the basic operations in the GA are described in the form of a brief walk-through. Next, the major routines, such as fitness assignment and selection, are considered by referring to the model of a simple sequential GA and recent advances are described. Although the sequential model is used, most of the underlying mechanisms discussed are the same whether the GA implementation is sequential or parallel. A broad range of parallel implementations are described covering a representative selection of application areas. Finally, some conclusions are drawn about the benefits of parallel GAs over sequential ones.

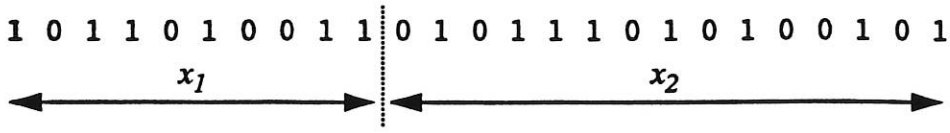
2 What are Genetic Algorithms?

The GA is a stochastic global search method that mimics the metaphor of natural biological evolution [1]. GAs operate on a population of potential solutions applying the principle of survival of the fittest to produce (hopefully) better and better approximations to a solution. At each generation, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the problem domain and breeding them together using operators borrowed from natural genetics. This process leads to the evolution of populations of individuals that are better suited to their environment than the individuals that they were created from, just as in natural adaptation.

2.1 Overview of GAs

Individuals, or current approximations, are encoded as strings, *chromosomes*, composed over some alphabet(s), so that the *genotypes* (chromosome values) are uniquely mapped onto the decision variable (*phenotypic*) domain. The most commonly used representation in GAs is the

binary alphabet {0, 1} although other representations can be used, e.g. ternary, integer, real-valued etc. For example, a problem with two variables, x_1 and x_2, may be mapped onto the chromosome structure in the following way:



where x_1 is encoded with 10 bits and x_2 with 15 bits, possibly reflecting the level of accuracy or range of the individual decision variables. Examining the chromosome string in isolation yields no information about the problem we are trying to solve. It is only with the decoding of the chromosome into its phenotypic values that any meaning can be applied to the representation. However, as described below, the search process will operate on this encoding of the decision variables, rather than the decision variables themselves, except, of course, where real-valued genes are used.

Having decoded the chromosome representation into the decision variable domain, it is possible to assess the performance, or *fitness*, of individual members of a population. This is done through an objective function that characterises an individual's performance in the problem domain. In the natural world, this would be an individual's ability to survive in its present environment. Thus, the objective function establishes the basis for selection of pairs of individuals that will be mated together during reproduction.

During the reproduction phase, each individual is assigned a fitness value derived from its raw performance measure given by the objective function. This value is used in the selection to bias towards more fit individuals. Highly fit individuals, relative to the whole population, have a high probability of being selected for mating whereas less fit individuals have a correspondingly low probability of being selected.

Once the individuals have been assigned a fitness value, they can be chosen from the population, with a probability according to their relative fitness, and recombined to produce the next generation. Genetic operators manipulate the characters (genes) of the chromosomes directly, using the assumption that certain individual's gene codes, on average, produce fitter individuals. The *recombination* operator is used to exchange genetic information between pairs, or larger groups, of individuals. The simplest recombination operator is that of single-point crossover.

Consider the two parent binary strings:

$$P_1 = 1\ 0\ 0\ 1\ 0\ 1\ 1\ 0, \text{and}$$

$$P_2 = 1\ 0\ 1\ 1\ 1\ 0\ 0\ 0.$$

If an integer position, i, is selected uniformly at random from the range [1, l-1], where l is the string length, and the genetic information exchanged between the individuals about this point,

then two new offspring strings are produced. The two offspring below are produced when the crossover point $i = 5$ is selected,

$$O_1 = 1\ 0\ 0\ 1\ 0 \mid 0\ 0\ 0, \text{and}$$

$$O_2 = 1\ 0\ 1\ 1\ 1 \mid 1\ 1\ 0.$$

This crossover operation is not necessarily performed on all strings in the population. Instead, it is applied with a probability P_x when the pairs are chosen for breeding. A further genetic operator, called *mutation*, is then applied to the new chromosomes, again with a set probability, P_m. Mutation causes the individual genetic representation to be changed according to some probabilistic rule. In the binary string representation, mutation will cause a random bit to change its state, $0 \Rightarrow 1$ or $1 \Rightarrow 0$. So, for example, mutating the fourth bit of O_1 leads to the new string,

$$O_{1m} = 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0.$$

Mutation is generally considered to be a background operator that ensures that the probability of searching a particular subspace of the problem space is never zero. This has the effect of tending to inhibit the possibility of converging to a local optimum, rather than the global optimum.

After recombination and mutation, the individual strings are then, if necessary, decoded, the objective function evaluated, a fitness value assigned to each individual and individuals selected for mating according to their fitness, and so the process continues through subsequent generations. In this way, the average performance of individuals in a population is expected to increase, as good individuals are preserved and bred with one another and the less fit individuals die out. The GA is terminated when some criteria are satisfied, e.g. a certain number of generations completed, a mean deviation in the performance of individuals in the population, or when a particular point in the search space is encountered.

2.2 GAs versus Traditional Methods

From the above discussion, it can be seen that the GA differs substantially from more traditional search and optimization methods. The four most significant differences are:

- GAs search a population of points in parallel, not a single point.
- GAs use probabilistic transition rules, not deterministic ones.
- GAs work on an encoding of the parameter set rather than the parameter set itself (except in where real-valued individuals are used).
- GAs do not require derivative information or other auxiliary knowledge; only the objective function and corresponding fitness levels influence the directions of search.

It is important to note that the GA provides a number of potential solutions to a given problem and

the choice of final solution is left to the user. In cases where a particular problem does not have one individual solution, for example, a family of Pareto-optimal solutions, as is the case in multiobjective optimization problems, then the GA is potentially useful for identifying these alternative solutions simultaneously.

3 Major Elements of the Genetic Algorithm

The simple genetic algorithm (SGA) is described by Goldberg [2] and is used here to illustrate the basic components of the GA. A pseudo-code outline of the SGA is shown in Fig. 1. The population at time t is represented by the time-dependent variable P, with the initial population of random estimates being $P(0)$. Using this outline of a GA, the remainder of this Section describes the major elements of the GA.

```
procedure GA
begin
        t = 0;
        initialize P(t);
        evaluate P(t);
        while not finished do
        begin
                t = t + 1;
                select P(t) from P(t-1);
                reproduce pairs in P(t);
                evaluate P(t);
        end
end.
```

Figure 1: A Simple Genetic Algorithm

3.1 Population Representation and Initialisation

GAs operate simultaneously on a number of potential solutions, called a population, consisting of some encoding of the parameter set. Typically, a population is composed of between 30 and 100 individuals, although, a variant called the micro GA uses very small populations, ~10 individuals, with a restrictive reproduction and replacement strategy in an attempt to satisfy real-time execution requirements [3].

The most commonly used representation of chromosomes in the GA is that of the single-level binary string. Here, each decision variable in the parameter set is encoded as a binary string and these are concatenated to form a chromosome. The use of Gray coding has been advocated as a method of overcoming the hidden representational bias in conventional binary representation as the Hamming distance between adjacent values is constant [4]. Empirical evidence of Caruana and Schaffer [5] suggests that large Hamming distances in the representational mapping between

adjacent values, as is the case in the standard binary representation, can result in the search process being deceived or unable to efficiently locate the global minimum. A further approach of Schmitendorgf *et al* [6], is the use of logarithmic scaling in the conversion of binary-coded chromosomes to their real phenotypic values. Although the precision of the parameter values is possibly less consistent over the desired range, in problems where the spread of feasible parameters is unknown a larger search space may be covered with the same number of bits than a linear mapping scheme, thus allowing the computational burden of exploring unknown search spaces to be reduced to a more manageable level.

Whilst binary-coded GAs are most commonly used, there is an increasing interest in alternative encoding strategies, such as integer and real-valued representations. For some problem domains, it is argued that the binary representation is in fact deceptive in that it obscures the nature of the search [7]. In the subset selection problem [8], for example, the use of an integer representation and look-up tables provides a convenient and natural way of expressing the mapping from representation to problem domain.

The use of real-valued genes in GAs is claimed by Wright [9] to offer a number of advantages in numerical function optimization over binary encodings. Efficiency of the GA is increased as there is no need to convert chromosomes to phenotypes before each function evaluation; less memory is required as efficient floating-point internal computer representations can be used directly; there is no loss in precision by discretisation to binary or other values; and there is greater freedom to use different genetic operators. The use of real-valued encodings is described in detail by Michalewicz [10] and in the literature on Evolution Strategies (see, for example, [11]).

Having decided on the representation, the first step in the SGA is to create an initial population. This is usually achieved by generating the required number of individuals using a random number generator that uniformly distributes numbers in the desired range. For example, with a binary population of N_{ind} individuals whose chromosomes are L_{ind} bits long, $N_{ind} \times L_{ind}$ random numbers uniformly distributed from the set {0, 1} would be produced.

A variation is the *extended random initialisation* procedure of Bramlette [7] whereby a number of random initialisations are tried for each individual and the one with the best performance is chosen for the initial population. Other users of GAs have seeded the initial population with some individuals that are known to be in the vicinity of the global minimum (see, for example, [12] and [13]). This approach is, of course, only applicable if the nature of the problem is well understood beforehand or if the GA is used in conjunction with a knowledge based system.

3.2 The Objective and Fitness Functions

The objective function is used to provide a measure of how individuals have performed in the problem domain. In the case of a minimization problem, the most fit individuals will have the lowest numerical value of the associated objective function. This raw measure of fitness is usually only used as an intermediate stage in determining the relative performance of individuals in a GA. Another function, the *fitness function*, is normally used to transform the objective function value into a measure of relative fitness [14], thus:

$$F(x) = g(f(x)) \tag{1}$$

where f is the objective function, g transforms the value of the objective function to a non-negative number and F is the resulting relative fitness. This mapping is always necessary when the objective function is to be minimized as the lower objective function values correspond to fitter individuals. In many cases, the fitness function value corresponds to the number of offspring that an individual can expect to produce in the next generation. A commonly used transformation is that of proportional fitness assignment (see, for example, [2]). The individual fitness, $F(x_i)$, of each individual is computed as the individual's raw performance, $f(x_i)$, relative to the whole population, i.e.,

$$F(x_i) = \frac{f(x_i)}{\sum_{i=1}^{N_{ind}} f(x_i)}, \tag{2}$$

where N_{ind} is the population size and x_i is the phenotypic value of individual i. Whilst this fitness assignment ensures that each individual has a probability of reproducing according to its relative fitness, it fails to account for negative objective function values.

A linear transformation which offsets the objective function [2] is often used prior to fitness assignment, such that,

$$F(x) = af(x) + b \tag{3}$$

where a is a positive scaling factor if the optimization is maximizing and negative if we are minimizing. The offset b is used to ensure that the resulting fitness values are non-negative.

The use of linear scaling and offsetting outlined above is, however, a possible cause of rapid convergence. The *selection* algorithm (see below) selects individuals for reproduction on the basis of their relative fitness. Using linear scaling, the expected number of offspring is approximately proportional to that individuals performance. As there is no constraint on an individual's performance in a given generation, highly fit individuals in early generations can dominate the reproduction causing rapid convergence to possibly sub-optimal solutions. Similarly, if there is little deviation in the population, then scaling provides only a small bias towards the most fit individuals.

Baker [15] suggests that limiting the reproductive range, so that no individuals generate an excessive number of offspring, prevents premature convergence. Here, individuals are assigned a fitness according to their rank in the population rather than their raw performance. One variable, SP, is used to determine the bias, or *selective pressure*, towards the most fit individual and the fitness of the others is determined by,

$$F(x_i) = 2 - SP + 2(SP - 1)\frac{x_i - 1}{N_{ind} - 1}, \tag{4}$$

where x_i is the position in the ordered population of individual i.

For example, for a population size of $N_{ind} = 40$ and selective pressure of $SP = 1.1$, individuals are given a fitness value in the range [0.9, 1.1]. The least fit individual has a fitness of 0.9 while the most fit is assigned a fitness of 1.1. The increment in the fitness value between adjacent ranks is thus 0.0051.

3.3 Selection

Selection is the process of determining the number of times, or *trials*, a particular individual is chosen for reproduction and, thus, the number of offspring that an individual will produce. The selection of individuals can be viewed as two separate processes:

1) determination of the number of trials an individual can expect to receive, and

2) conversion of the expected number of trials into a discrete number of offspring.

The first part is concerned with the transformation of raw fitness values into a real-valued expectation of an individual's probability to reproduce and is dealt with in the previous subsection as fitness assignment. The second part is the probabilistic selection of individuals for reproduction based on the fitness of individuals relative to one another and is sometimes known as *sampling*. The remainder of this subsection will review some of the more popular selection methods in current usage.

Baker [16] presented three measures of performance for selection algorithms, *bias*, *spread* and *efficiency*. Bias is defined as the absolute difference between an individual's actual and expected selection probability. Optimal zero bias is therefore achieved when an individual's selection probability equals its expected number of trials.

Spread is the range in the possible number of trials that an individual may achieve. If $f(i)$ is the actual number of trials that individual i receives, then the "minimum spread" is the smallest spread that theoretically permits zero bias, i.e.

$$f(i) \in \left\{ \lfloor et(i) \rfloor, \lceil et(i) \rceil \right\} \tag{5}$$

where $et(i)$ is the expected number of trials of individual i, $\lfloor et(i) \rfloor$ is the floor of et(i) and $\lceil et(i) \rceil$ is the ceiling. Thus, while bias is an indication of accuracy, the spread of a selection method measures its consistency.

The desire for efficient selection methods is motivated by the need to maintain a GAs overall time complexity. It has been shown in the literature that the other phases of a GA (excluding the actual objective function evaluations) are $O(L_{ind} N_{ind})$ or better time complexity. The selection algorithm should thus achieve zero bias whilst maintaining a minimum spread and not contributing to an increased time complexity of the GA.

3.3.1 Roulette Wheel Selection Methods

Many selection techniques employ a "roulette wheel" mechanism to probabilistically select individuals based on some measure of their performance. A real-valued interval, *Sum*, is determined as either the sum of the individuals' expected selection probabilities or the sum of the raw fitness values over all the individuals in the current population. Individuals are then mapped one-to-one into contiguous intervals in the range [0, *Sum*]. The size of each individual interval corresponds to the fitness value of the associated individual. For example, in Fig. 2 the circumference of the roulette wheel is the sum of all six individual's fitness values. Individual 5 is the most fit individual and occupies the largest interval, whereas individuals 6 and 4 are the least fit and have correspondingly smaller intervals within the roulette wheel. To select an individual, a random number is generated in the interval [0, *Sum*] and the individual whose segment spans the random number is selected. This process is repeated until the desired number of individuals have been selected.

The basic roulette wheel selection method is stochastic sampling with replacement (SSR). Here, the segment size and selection probability remain the same throughout the selection phase and individuals are selected according to the procedure outlined above. SSR gives zero bias but a potentially unlimited spread. Any individual with a segment size > 0 could entirely fill the next population.

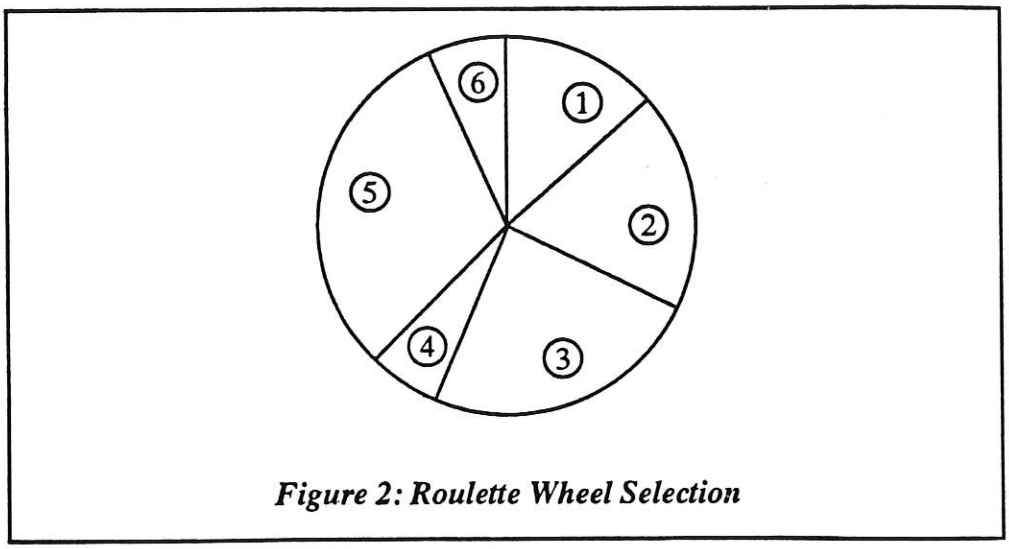


Figure 2: Roulette Wheel Selection

Stochastic sampling with partial replacement (SSPR) extends upon SSR by resizing an individual's segment if it is selected. Each time an individual is selected, the size of its segment is reduced by 1.0. If the segment size becomes negative, then it is set to 0.0. This provides an upper bound on the spread of $\lceil et(i) \rceil$. However, the lower bound is zero and the bias is higher than that of SSR.

Remainder sampling methods involve two distinct phases. In the integral phase, individuals are selected deterministically according to the integer part of their expected trials. The remaining individuals are then selected probabilistically from the fractional part of the individuals expected values. Remainder stochastic sampling with replacement (RSSR) uses roulette wheel selection to

sample the individual not assigned deterministically. During the roulette wheel selection phase, individual's fractional parts remain unchanged and, thus, compete for selection between "spins". RSSR provides zero bias and the spread is lower bounded. The upper bound is limited only by the number of fractionally assigned samples and the size of the integral part of an individual. For example, any individual with a fractional part > 0 could win all the samples during the fractional phase. Remainder stochastic sampling without replacement (RSSWR) sets the fractional part of an individual's expected values to zero if it is sampled during the fractional phase. This gives RSSWR minimum spread, although this selection method is biased in favour of smaller fractions.

3.3.2 Stochastic Universal Sampling

Stochastic universal sampling (SUS) is a single-phase sampling algorithm with minimum spread and zero bias. Instead of the single selection pointer employed in roulette wheel methods, SUS uses N equally spaced pointers, where N is the number of selections required. The population is shuffled randomly and a single random number in the range [0 *Sum/N*] is generated, *ptr*. The N individuals are then chosen by generating the N pointers spaced by 1, [*ptr*, *ptr*+1, ..., *ptr*+*N*-1], and selecting the individuals whose fitnesses span the positions of the pointers. An individual is thus guaranteed to be selected a minimum of $\lfloor et(i) \rfloor$ times and no more than $\lceil et(i) \rceil$, thus achieving minimum spread. In addition, as individuals are selected entirely on their position in the population, SUS has zero bias. For these reasons, SUS has become the most widely used selection algorithm in current GAs.

3.4 Crossover (Recombination)

The basic operator for producing new chromosomes in the GA is that of crossover. Like its counterpart in nature, crossover produces new individuals that have some parts of both parent's genetic material. The simplest form of crossover is that of single-point crossover, described in the Overview of GAs in Section 2.1. In this Section, a number of variations on crossover are described and discussed and the relative merits of each reviewed.

3.4.1 Multi-point Crossover

For multi-point crossover, m crossover positions, $k_i \in \{1, 2, \ldots, l-1\}$, where k_i are the crossover points and l is the length of the chromosome, are chosen at random with no duplicates and sorted into ascending order. Then, the bits between successive crossover points are exchanged between the two parents to produce two new offspring. The section between the first allele position and the first crossover point is not exchanged between individuals. This process is illustrated in Fig. 3 overleaf.

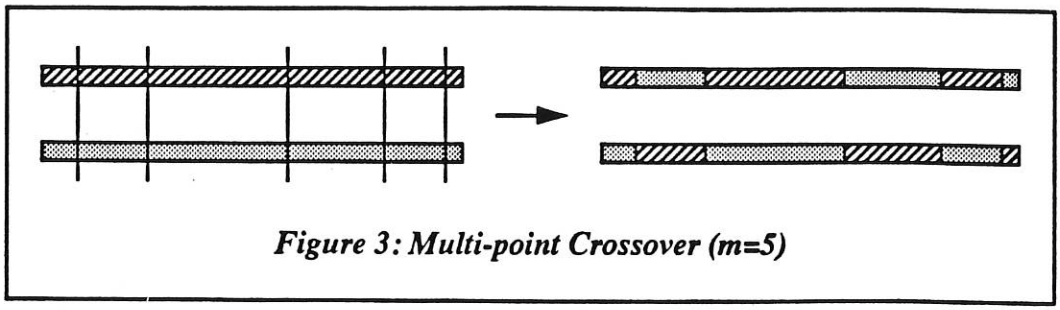


Figure 3: Multi-point Crossover (m=5)

The idea behind multi-point, and indeed many of the variations on the crossover operator, is that the parts of the chromosome representation that contribute most to the performance of a particular individual may not necessarily be contained in adjacent sub-strings [17]. Further, the disruptive nature of multi-point crossover appears to encourage the exploration of the search space, rather than favouring convergence to highly fit individuals early in the search, thus making the search more robust [18].

3.4.2 Uniform Crossover

Single and multi-point crossover define cross points as places between loci where a chromosome can be split. Uniform crossover [19] generalises this scheme to make every locus a potential crossover point. A crossover mask, the same length as the chromosome structures is created at random and the parity of the bits in the mask indicates which parent will supply the offspring with which bits. Consider the following two parents, crossover mask and resulting offspring:

P_1 = 1 0 1 1 0 0 0 1 1 1
P_2 = 0 0 0 1 1 1 1 0 0 0

Mask = 0 0 1 1 0 0 1 1 0 0

O_1 = 0 0 1 1 1 1 0 1 0 0
O_2 = 1 0 0 1 0 0 1 0 1 1

Here, the first offspring, O_1, is produced by taking the bit from P_1 if the corresponding mask bit is 1 or the bit from P_2 if the corresponding mask bit is 0. Offspring O_2 is created using the inverse of the mask or, equivalently, swapping P_1 and P_2.

Uniform crossover, like multi-point crossover, has been claimed to reduce the bias associated with the length of the binary representation used and the particular coding for a given parameter set. This helps to overcome the bias in single-point crossover towards short sub-strings without requiring precise understanding of the significance of individual bits in the chromosome representation. Spears and De Jong [20] have demonstrated how uniform crossover may be parameterised by applying a probability to the swapping of bits. This extra parameter can be used to control the amount of disruption during recombination without introducing a bias towards the length of the representation used. When uniform crossover is used with real-valued alleles, it is

usually referred to as *discrete recombination.*

3.4.3 Other Crossover Operators

A related crossover operator is that of *shuffle* [21]. A single cross-point is selected, but before the bits are exchanged, they are randomly shuffled in both parents. After recombination, the bits in the offspring are unshuffled. This too removes positional bias as the bits are randomly reassigned each time crossover is performed.

The *reduced surrogate* operator [17] constrains crossover to always produce new individuals wherever possible. Usually, this is implemented by restricting the location of crossover points such that crossover points only occur where gene values differ.

3.4.4 Intermediate Recombination

Given a real-valued encoding of the chromosome structure, intermediate recombination is a method of producing new phenotypes around and between the values of the parents phenotypes [22]. Offspring are produced according to the rule,

$$O_1 = P_1 \times \alpha (P_2 - P_1), \tag{6}$$

where α is a scaling factor chosen uniformly at random over some interval, typically [-0.25, 1.25] and P_1 and P_2 are the parent chromosomes (see, for example, [22]). Each variable in the offspring is the result of combining the variables in the parents according to the above expression with a new α chosen for each pair of parent genes. In geometric terms, intermediate recombination is capable of producing new variables within a slightly larger hypercube than that defined by the parents but constrained by the range of α. as shown in Fig 4.

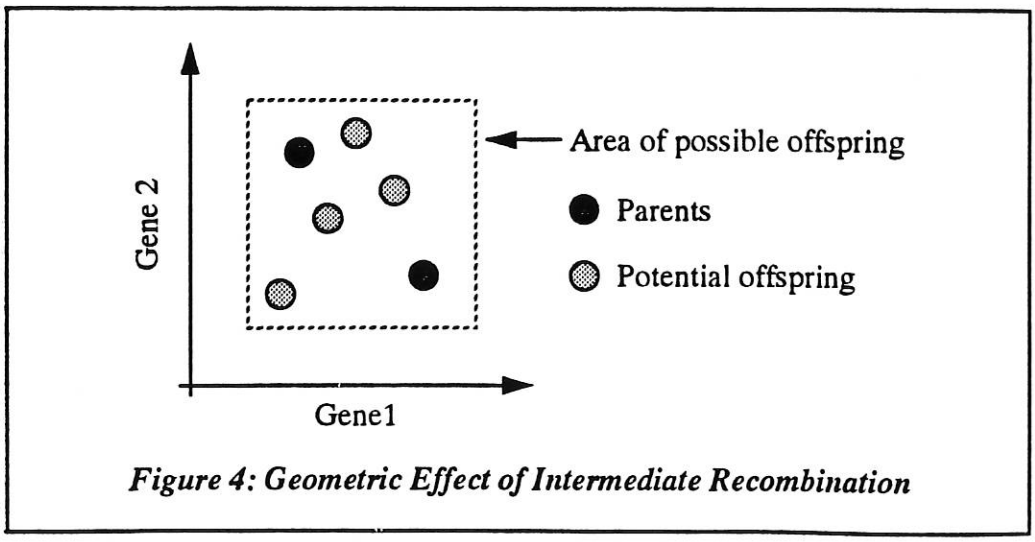


Figure 4: Geometric Effect of Intermediate Recombination

3.4.5 Line Recombination

Line recombination [22] is similar to intermediate recombination, except that only one value of α

is used in the recombination. Fig. 5 shows how line recombination can generate any point on the line defined by the parents within the limits of the perturbation, α, for a recombination in two variables.

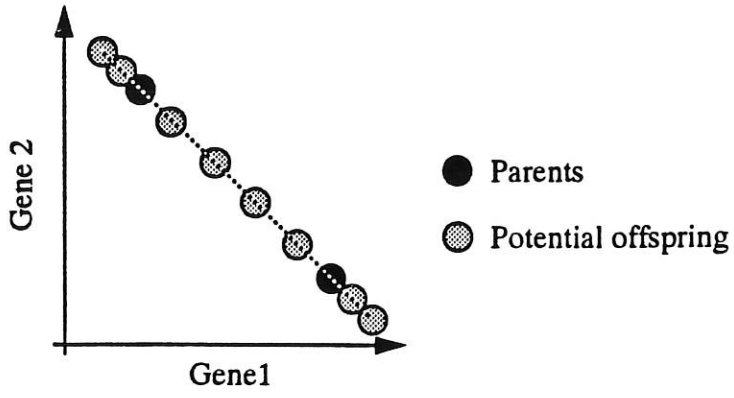


Figure 5: Geometric Effect of Line Recombination

3.4.6 Discussion

The binary operators discussed in this Section have all, to some extent, used disruption in the representation to help improve exploration during recombination. Whilst these operators may be used with real-valued populations, the resulting changes in the genetic material after recombination would not extend to the actual values of the decision variables, although offspring may, of course, contain genes from either parent. The intermediate and line recombination operators overcome this limitation by acting on the decision variables themselves. Like uniform crossover, the real-valued operators may also be parameterised to provide a control over the level of disruption introduced into offspring. For discrete-valued representations, variations on the recombination operators may be used that ensure that only valid values are produced as a result of crossover [23].

3.5 Mutation

In natural evolution, mutation is a random process where one allele of a gene is replaced by another to produce a new genetic structure. In GAs, mutation is randomly applied with low probability, typically in the range 0.001 and 0.01, and modifies elements in the chromosomes. Usually considered as a background operator, the role of mutation is often seen as providing a guarantee that the probability of searching any given string will never be zero and acting as a safety net to recover good genetic material that may be lost through the action of selection and crossover [2].

The effect of mutation on a binary string is illustrated in Fig. 6 for a 10-bit chromosome representing a real value decoded over the interval [0, 10] using both standard and Gray coding and a mutation point of 3 in the binary string. Here, binary mutation flips the value of the bit at the loci selected to be the mutation point. The effect of mutation on the decision variable, of course,

depends on the encoding scheme used. Given that mutation is generally applied uniformly to an entire population of strings, it is possible that a given binary string may be mutated at more than one point.

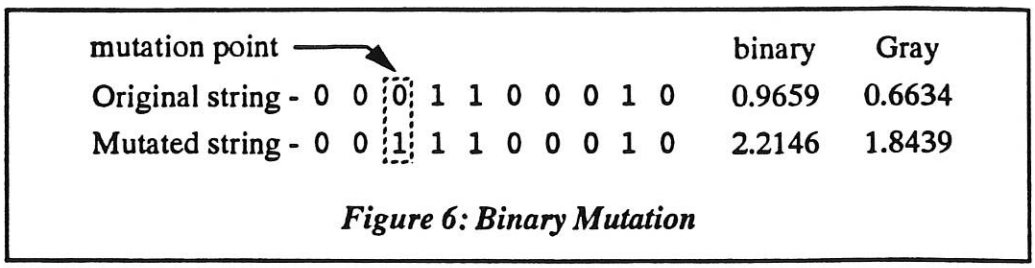


Figure 6: Binary Mutation

With non-binary representations, mutation is achieved by either perturbing the gene values or random selection of new values within the allowed range. Wright [9] and Janikow and Michalewicz [24] demonstrate how real-coded GAs may take advantage of higher mutation rates than binary-coded GAs, increasing the level of possible exploration of the search space without adversely affecting the convergence characteristics. Indeed, Tate and Smith [25] argue that for codings more complex than binary, high mutation rates can be both desirable and necessary and show how, for a complex combinatorial optimization problem, high mutation rates and non-binary coding yielded significantly better solutions than the normal approach.

Many variations on the mutation operator have been proposed. For example, biasing the mutation towards individuals with lower fitness values to increase the exploration in the search without losing information from the fitter individuals [26] or parameterising the mutation such that the mutation rate decreases with the population convergence [27]. Mühlenbein [22] has introduced a mutation operator for the real-coded GA that uses a non-linear term for the distribution of the range of mutation applied to gene values. It is claimed that by biasing mutation towards smaller changes in gene values, mutation can be used in conjunction with recombination as a foreground search process. Other mutation operations include that of *trade mutation* [8], whereby the contribution of individual genes in a chromosome is used to direct mutation towards weaker terms, and *reorder mutation* [8], that swaps the positions of bits or genes to increase diversity in the decision variable space.

3.6 Reinsertion

Once a new population has been produced by selection and recombination of individuals from the old population, the fitness of the individuals in the new population may be determined. If fewer individuals are produced by recombination than the size of the original population, then the fractional difference between the new and old population sizes is termed a generation gap [28]. In the case where the number of new individuals produced at each generation is one or two, the GA is said to be steady-state [29] or incremental [30]. If one or more of the most fit individuals is deterministically allowed to propagate through successive generations then the GA is said to use an *elitist strategy.*

To maintain the size of the original population, the new individuals have to be reinserted into the old population. Similarly, if not all the new individuals are to be used at each generation or if

more offspring are generated than the size of the old population then a reinsertion scheme must be used to determine which individuals are to exist in the new population. An important feature of not creating more offspring than the current population size at each generation is that the generational computational time is reduced, most dramatically in the case of the steady-state GA, and that the memory requirements are smaller as fewer new individuals need to be stored while offspring are produced.

When selecting which members of the old population should be replaced the most apparent strategy is to replace the least fit members deterministically. However, in studies, Fogarty [31] has shown that no significant difference in convergence characteristics was found when the individuals selected for replacement where chosen with inverse proportional selection or deterministically as the least fit. He further asserts that replacing the least fit members effectively implements an elitist strategy as the most fit will probabilistically survive through successive generations. Indeed, the most successful replacement scheme was one that selected the oldest members of a population for replacement. This is reported as being more in keeping with generational reproduction as every member of the population will, at some time, be replaced. Thus, for an individual to survive successive generations, it must be sufficiently fit to ensure propagation into future generations.

3.7 Termination of the GA

Because the GA is a stochastic search method, it is difficult to formally specify convergence criteria. As the fitness of a population may remain static for a number of generations before a superior individual is found, the application of conventional termination criteria becomes problematic. A common practice is to terminate the GA after a prespecified number of generations and then test the quality of the best members of the population against the problem definition. If no acceptable solutions are found, the GA may be restarted or a fresh search initiated.

4 Parallel GAs

Given the preceding description of the GA, it is clear that the GA may be parallelised in a number of ways. Indeed, there are many variations on parallel GAs, many of which are very different from the original GA presented by Holland [1]. Most of the major differences are encountered in the population structure and the method of selecting individuals for reproduction. The motivation for exploring parallel GAs is manifold. One may wish to improve speed and efficiency by employing a parallel computer, apply the GA to larger problems or try to follow the biological metaphor more closely by introducing structure and geographic location into the population. As this Section will show, the benefits of using parallel GAs, even when run on a sequential machine, can be more than just a speed-up in the execution time.

In the remainder of this Section, we describe a number of parallel GAs and use three broad categories to classify them:- global, migration and diffusion. These categories reflect the different ways in which parallelism is exploited in the GA and the nature of the population structure and recombination mechanisms used. The global GA treats the entire population as a single breeding unit and aims to exploit the algorithmic parallelism inherent in the GA. Migration GAs divide the

population into a number of sub-populations, each of which is treated as a separate breeding unit under the control of a conventional GA. To encourage the proliferation of good genetic material throughout the whole population, individuals migrate between the sub-populations from time to time. Generally, the migration GA is considered coarse-grained. The diffusion GA treats each individual as a separate breeding unit, the individuals it may mate with being selected from within a small local neighbourhood. The use of local selection and reproduction rules leads to a continuous diffusion of individuals over the population. The diffusion GA is usually considered fine-grained.

4.1 Global GAs

Examination of the pseudo-code outline of the sequential Simple GA given in Fig. 1 reveals that a significant proportion of the computation in a GA is composed of taking pairs of individuals, combining them to form new offspring, applying mutation and evaluating a cost function. Taking a population size of, say, 50 and assuming that reproduction of two individuals creates two new offspring, then the inner loop of Fig. 1 contains 25 discrete operations that may be performed concurrently at each generation. The Worker/Farmer architecture in Fig. 7 demonstrates how this geometric parallelism may be exploited by a parallel computer.

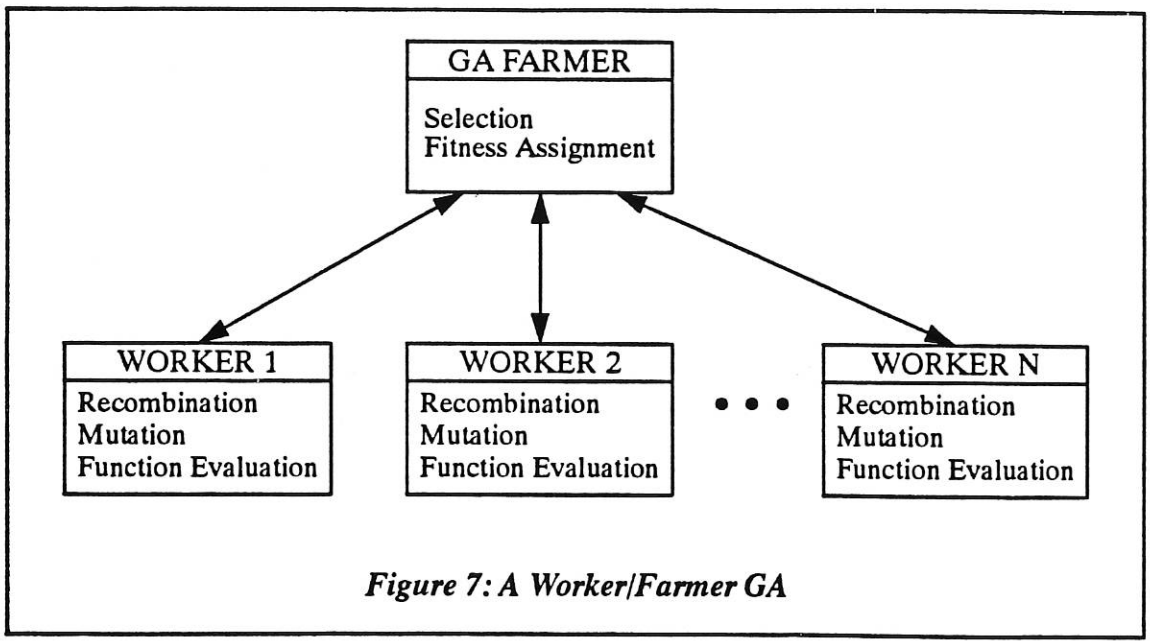


Figure 7: A Worker/Farmer GA

The GA Farmer node initialises and holds the entire population, performs selection and assigns fitness to individuals. The Worker nodes recombine individuals, apply mutation and evaluate the objective function for the resulting offspring. Goldberg [2] describes a similar scheme, the *synchronous master-slave*, whereby a hybrid GA uses a local search routine at each worker to further refine the estimates generated at each node. Others, notably Fogarty and Huang [30] and Dodd *et al* [32] use the processor farm for the evaluation of objective functions only.

Although the farmed GA does not embrace all of the parallelism inherent in the GA, near linear speed-up has been reported in cases where the objective function is significantly more

computationally expensive than the GA itself. In particular, when the objective function being minimized is of low computational cost, then there is potentially a bottleneck at the farmer while fitness assignment and selection are performed. The computational efficiency, of course, depends on the balance between the cost of the parallel parts of the GA and the sequential elements. Thus, the farmed GA may be inefficient if the objective function evaluation times vary greatly.

Goldberg [2] also describes a *semi-synchronous master-slave* GA that overcomes this potential bottleneck by relaxing the requirement for strict synchronous operation. Here, individuals are selected and inserted into the population as and when the worker nodes complete their tasks. However, both the synchronous and semi-synchronous models are potentially unreliable because of the dependence on the single farmer process.

A further issue is that of the message size used to pass individuals from the Farmer to the Workers. Whilst smaller messages allow greater control over the load balance on the nodes in the parallel system, they may be less efficient in using available bandwidth. Thus, more than one pair of individuals may be sent from the Farmer to a Worker in a communication interval. If the objective function is sufficiently large, then the worker/farmer GA may prove a useful method of reducing the execution time of the GA when the nodes used are networked workstations.

A more robust extension to the worker/farmer implementations is the *asynchronous, concurrent* GA [2]. Using a number of identical processors, genetic operators and objective function evaluations are performed independently of one another on a population stored in a shared memory. This requires that no individual be accessed by more than one processor simultaneously. Although more complicated to implement than the conventional farmed GAs described above, this scheme is highly tolerant to processor and memory failure. Even if only one processor and some of the shared memory are functioning, it is still possible for useful processing to be performed.

4.2 Migration GAs

The GA as described thus far operates globally on a single population. That is, individuals are processed probabilistically on their performance in the population as a whole and any individual has the potential to mate with any other individual in the entire population. This treatment of the population as a single breeding unit is known as *panmixia*.

In natural evolution, species tend to reproduce within subgroups of the entire population, isolated to some extent from one another, but with the possibility of mating occurring across the boundaries of the subgroups. A population distributed amongst a number of semi-isolated breeding groups is known as *polytypic*. Humans, for example, are polytypic in that they consist of groups of the species isolated from one another geographically, culturally and economically. Whilst breeding may occur between individuals from different subgroups of the species, it is much more likely that individuals from within the same group will reproduce together.

The *migration* or *island* model of the GA introduces geographic population distribution by dividing a large population into many smaller semi-isolated sub-populations or *demes*. Each sub-population is a separate breeding unit using local selection and reproduction rules to locally evolve the species. From time to time, migration of individuals occurs between sub-populations

such that individuals from one population are introduced into another sub-population. The pattern of migration limits how much genetic diversity can occur in the global population. This pattern of migration is determined by the number of individuals migrated, the interval between migration and the migration paths between sub-populations. This movement of individuals between demes is often termed the *stepping stone model.*

The traditional sequential GA can readily be extended to encompass the migration model. A pseudo-code outline of the modified algorithm for the migration GA is shown in Fig. 8. The population is divided into a number of sub-populations each of which is evolved by an independent GA. Additional routines are included to exchange individuals between sub-populations according to the communications topology employed and a global termination criteria introduced. Fig. 9 shows a possible implementation of the migration GA and some of the migration paths between the population islands.

```
-- Each node (GAi)

WHILE not finnished
    SEQ
        ... Selection
        ... Reproduction
        ... Evaluation
        PAR
            ... Send emmigrants
            ... Receive immigrants
```

Figure 8: Pseudo-code Outline of the Migration GA

Grosso [33] first introduced a geographically isolated population structure in 1985 using an island model of the GA with five independent sub-populations. From his study, he found that semi-isolated populations improved the performance of the GA in terms of the quality of solution and the number of function evaluations required. He asserted that limited migration of individuals between sub-populations was more effective than either complete sub-population interdependence or independence.

In 1987, Petty *et al* [34] presented a migration GA influenced by earlier work on population isolation conducted by Wilson [35] in his ANIMAT classifier-based learning system and Schaffer's VEGA [36]. ANIMAT modelled an artificial creature seeking food and avoiding trees in a two-dimensional wood. Each individual specifies one action that this creature was to make, e.g. in which direction to move. If an individual was selected for reproduction, then its mate was chosen from the sub-population of individuals specifying the same action. Schaffer's VEGA, Vector Evaluated GA, was designed to solve problems with multiple objectives. The population was divided into a sub-populations, each sub-population associated with a particular objective. Individuals were then selected according to their performance within a sub-population, and mating was allowed to occur across the sub-population boundaries. The idea behind this

methodology is that highly-fit individuals in one objective domain mate with other fit individuals from other objective domains to produce new individuals well suited to all of the objectives.

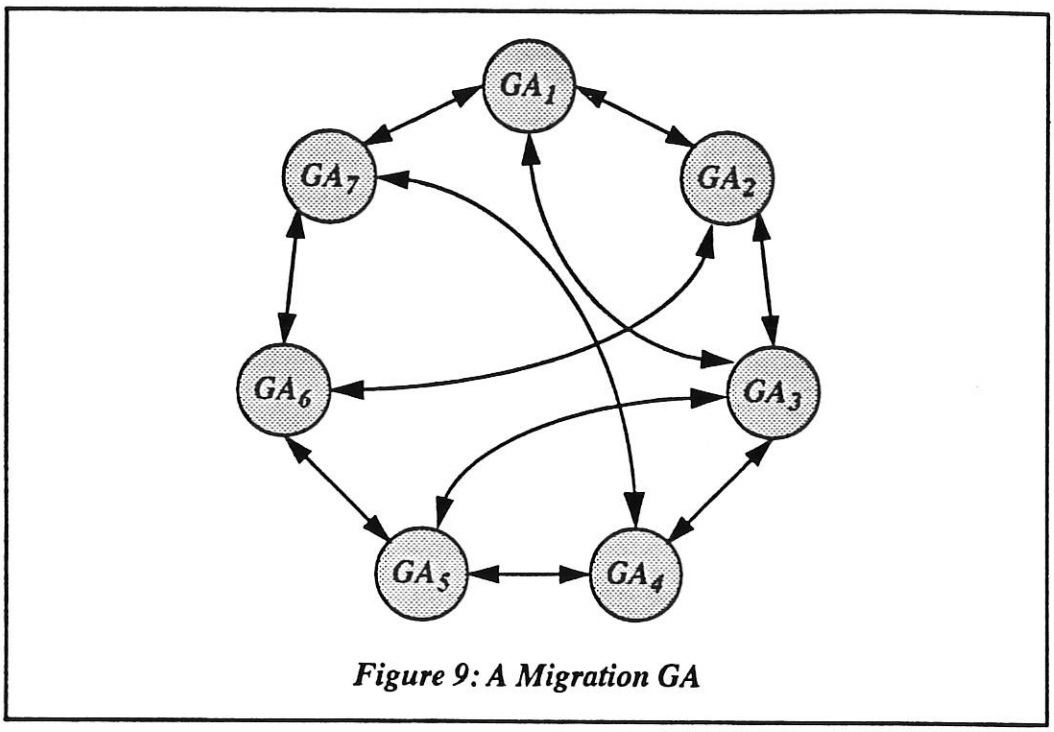


Figure 9: A Migration GA

Petty's migration GA was implemented on an Intel iPSC message-passing multi-processor system with a binary n-cube interconnection network. One GA, called a nodal GA, was placed on each processor. At each generation, the best individual on each node is communicated to its neighbours. Each sub-population therefore receives one individuals from each neighbouring node which it inserts into its current population. A number of insertion strategies were tested:- replacing the worst individuals with immigrants, uniform replacement of individuals by immigrants and replacement of individuals with the smallest Hamming distance from the immigrants. However, the effect of the insertion methods is not reported. Petty *et al* found a significant performance increase in terms of the convergence characteristics when the parallel GA was applied to numerical optimization problems selected from De Jong's test-bed functions [14], as well as the speed-up associated with running the GA on a parallel machine. Typically, the migration GA was able to converge to solutions not found by a global GA although the results showed a degree of problem dependence.

Tanse [37] also reported on a migration GA in 1987. He studied the migration model and compared its performance with a partitioned GA, i.e. a GA whose population is divided into sub-populations that evolve entirely independently with no migration between sub-populations. Again, the GA was implemented on a hypercube machine, although Tanse's parallel computer employed custom VAX-like CPUs. This implementation used two new parameters to specify the migration interval, at which generation migration should take place, and the migration rate, the number of individuals transferred between sub-populations. In early experiments [37], Tanse

selected individuals for migration probabilistically from the subset of the sub-population whose fitness was at least equal to the average fitness of the sub-population. Likewise, individuals were selected for replacement by immigrants probabilistically from the subset of individuals whose fitness was no greater than the average for that sub-population. Later [38], it appears that a new strategy was adopted. At a migration generation, each node produced more offspring than the current sub-population size. The migrants were then uniformly selected from the offspring and removed from the sub-population, thus maintaining the correct sub-population size. The receiving sub-population uniformly replaced individuals with immigrants. The philosophy behind this approach is that the most fit individuals are more likely to reproduce and are therefore most likely to migrate. The actual migration took place bidirectionally along one dimension of the hypercube, selected on the basis of the generation number. Thus, the neighbour selected to receive individuals from a node will also send its migrants to that node.

The results presented by Tanse showed a near-linear speed-up when compared against a sequential GA with a population size equal to the sum of the individual sub-populations. Comparing the migration GA with the partitioned GA, the migration GA consistently found superior individuals and had a higher average fitness over the entire population. However, due to the limited number of test functions, no conclusion can be drawn about the general effect of the migration rate and interval. The effect of the mutation and crossover operators used with the migration GA was also investigated. The results indicate that it is feasible to use different crossover and mutation rates on different nodes, allowing the balance between exploration and exploitation to be varied locally, but with migration ensuring that good individuals should survive in at least some sub-populations.

Similar results to Tanse are reported by Starkweather *et al* [39] and Cohoon *et al* [40]. Starkweathers *et al's* GA has a number of notable differences to the implementations described so far. Rather than using a generational GA in which most or all of the population is replaced at each generation, their parallel GA was based on Whitley's GENITOR program [29] that uses one-at-a-time reproduction replacing a single individual at each reproduction step. sub-populations are placed on a ring or a circle and migration of the fittest individuals takes place at regular predefined intervals. At migration step M, individuals on node X are migrated to the sub-population given by $mod(M+X,N)$ where N is the number of sub-populations numbered 0 to N-1. For example, at migration 1, sub-population 0 individuals migrate to sub-population 1 and at migration 2 individuals from sub-population 0 migrate to sub-population 2. The migration GA was applied to a wide range of problems including neural network optimization, a mapping problem and a 105-city travelling salesman problem. In all test cases, the migration GA produced better results than a comparable sequential one. When the migration GA was implemented on a sequential machine it was found that it would find better solutions and execute faster than a standard GA with the same population size on a number of test cases. In addition, the use of an adaptive mutation rate, initially high and reducing with generations, was found to improve the convergence characteristics of the GA.

In 1991, Mühlenbein *et al* [41] described a real-valued parallel GA for use as a "black-box solver" in high-dimensional optimization problems. A conventional single-point crossover operator was employed that operated directly on the ANSI-IEEE floating-point representation of the decision variables. The mutation operator worked only on the fractional part of a variable's representation,

thus mutation is exponentially biased towards producing a new individual in the region of the original rather than one a larger numerical distance away. In experiments, the parallel GA was able to find global solutions to problems of up to a dimension of 400.

The distributed breeder GA of Mühlenbein and Schlierkamp-Voosen [22], rather than modelling the natural and self-organized evolution of the earlier parallel GA described above, is based on a model of rational selection in human breeding groups. Whereas the parallel GA models natural selection, the breeder GA models artificial selection. Using influences from Evolution Strategies [11] and GAs, the breeder GA selects the best $T\%$ of a population, where T is a predefined parameter, and randomly mates them until sufficient offspring are produced. As well as ensuring that no individuals mate with themselves, the fittest individual also survives into the following generation. This selection and reproduction process is known as *truncation selection* as only a subset of each generation are used as potential parents.

The breeder GA operates on populations of real-valued individuals and has new genetic operators designed specifically for this representation, such as intermediate and line recombination, which have been described earlier in Sections 3.4 and 3.5. The parallel GA uses a local hill-climbing algorithm on certain individuals to improve a current local estimate. In the breeder GA, the mutation operator was found to be almost as effective as local hill-climbing but was much less complex and computationally demanding to implement. In all cases, the breeder GA was found to be more effective than the earlier parallel GA and managed to solve numerical optimization problems of dimension 1000.

A recent variation of the migration GA intended to overcome the problem of premature convergence is presented by Potts *et al* [42]. This algorithm, called GAMAS (GA based on Migration and Artificial Selection), uses four interdependent binary populations labelled SPECIES I to SPECIES IV. At initialisation, SPECIES II, SPECIES III and SPECIES IV are generated with a predefined bias. SPECIES II is created with a bias in favour of 0's in the chromosome representation, SPECIES IV biased towards 1's and SPECIES III with equal probability of 0's and 1's. Since the ratio of 0's to 1's contained in the optimal solution is not known *a priori*, this bias intends to encourage the exploration of different areas of the search space. SPECIES I is later filled as the result of an artificial selection process. The sub-population size used in all SPECIES was set to 36 individuals.

To encourage both exploitation and exploration of the species, GAMAS uses different mutation rates in the three evolving sub-populations. SPECIES II is used for exploration and has a high mutation rate, SPECIES IV for exploitation with a low mutation rate and SPECIES III for both exploration and exploitation with a slightly higher mutation rate than SPECIES IV. At each generation, the most fit individuals from SPECIES II, II and IV are artificially selected and placed in SPECIES I, replacing less fit individuals where necessary. Thus, SPECIES I is used as a dominant incipient species and is held in isolation with no reproduction or crossover. To further enforce exploration, migration of randomly selected individuals takes place between SPECIES II, III and IV. This migration takes place at high frequency in early generations and is gradually reduced in subsequent generations. At predetermined generation intervals, the entire contents of SPECIES I are used to replace SPECIES IV, the exploitation species. The rationale being to further develop the good approximations already obtained. A further refinement to the migration GA is the concept of *recycling* whereby the three evolving species are reinitialised without affecting the individuals

contained in SPECIES I. Recycling is intended to allow the GA to explore new regions of the search space and overcome the initial population bias reported by Goldberg [2].

GAMAS has been tested on a NP-hard combinatorial optimization problem - the file allocation problem, the XOR neural network which contains many optimal solutions and the multimodal, multivariate sine envelope sine wave function of Schaffer [43]. Again, in all these cases GAMAS found superior solutions to the sequential GA. In the XOR neural network problem, GAMAS also demonstrated an ability to find a number of different optimal solutions.

Clearly, the migration model of the GA is well suited to parallel implementation on MIMD machines. Given the range of possible population topologies and migration paths between them, efficient communications networks should be possible on most parallel architectures from small multiprocessor platforms to clusters of networked workstations. The semi-isolation of sub-populations and limited communication between them also encourages a high degree of fault tolerance. In a well-designed migration GA, in the event of the loss of individual sub-populations or communications paths between them, the GA can still perform useful computation.

The migration GA has generally been reported as a more efficient search and optimization method than conventional sequential GAs. From the preceding text, it should be clear that this is the effect of local selection and migration rather than parallel implementation. However, the migration GA is slightly more complex to use as further parameters are introduced to control migration between sub-populations.

4.3 Diffusion GAs

An alternative model of a distributed population structure is provided by the diffusion GA. Whereas migration introduces discontinuities into the population structure with barriers between the borders of the islands containing the sub-populations, diffusion treats the population as a single continuous structure. Each individual is assigned a geographic location on the population surface and is allowed to breed with individuals contained in a small local neighbourhood. This neighbourhood is usually chosen from immediately adjacent individuals on the population surface and is motivated by the practical communication restrictions of parallel computers. The diffusion GA is also known as the *neighbourhood, cellular* or *fine grained* GA.

Fig.10 shows a pseudo-code outline of the diffusion GA. Consider the population distribution shown in Fig. 11(a) where each individual, $I_{j,k}$, is assigned a separate node on a toroidal-mesh parallel processing network. The Figure shows that there are no specific islands in the population structure, rather a contiguous geographic distribution of individuals, however, there is potential for a similar effect. Given that mating is restricted to adjacent processors, then individuals on distant processors may take as many generations to meet and mate as individuals in different sub-populations in the island model. Wright [44] refers to this form of isolation within a species as *isolation by distance*. From Fig. 10, each individual is first initialised, either randomly or using a heuristic, and its performance evaluated. Each node then sends its individual to its neighbours and receives individuals from those neighbours. For example, in Fig. 11(a), individual $I_{3,1}$ sends a copy of itself to $I_{2,1}$, $I_{3,2}$, $I_{3,5}$ and $I_{4,1}$ and receives copies of the individuals on those nodes. The purpose of this communication is to provide a pool of potential mates from the incoming individuals. Thus, selection of a mate for individual $I_{3,1}$ is made on the basis of a neighbourhood

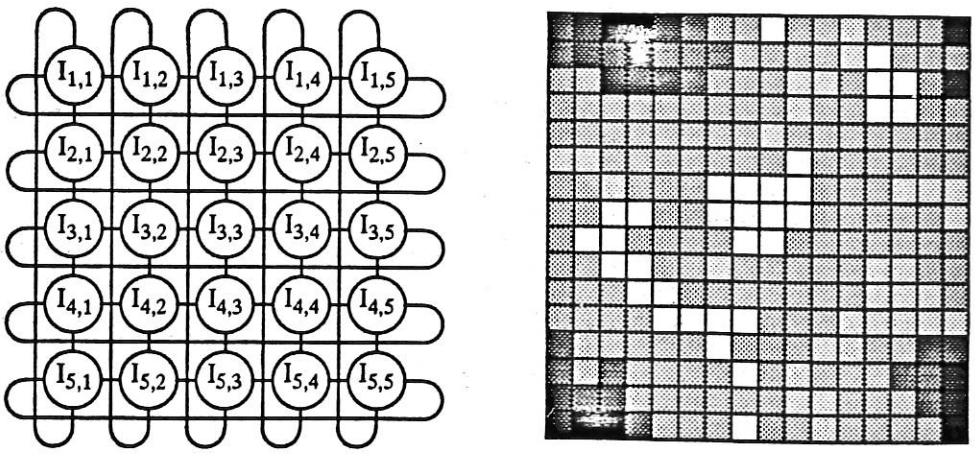
fitness over the individuals $I_{2,1}$, $I_{3,2}$, $I_{3,5}$ and $I_{4,1}$. Reproduction involves the usual crossover and mutation operators and is used to produce a single individual to replace the parent residing on the node. However, rules may be applied to retain the original parent if neither of the offspring is sufficiently fit to replace it.

```
-- Each node (Ii,j)

Initialise
WHILE not finished
  SEQ
    ... Evaluate
    PAR
      ... Send self to neighbours
      ... Receive neighbours
    ... Select mate
    ... Reproduce
```

Figure 10: Pseudo-code Outline of the Diffusion GA

At initialisation, the distribution of genetic material over the population surface is random, assuming that the population has not been seeded heuristically. After a few generations, local clusters of individuals with similar genetic material and fitness may appear in the population giving rise to *virtual islands*. This phenomena is shown in Fig. 11(b) where the shading is used to represent individuals with similar genetic material. The drift in the population caused by local selection tends to reduce the number of clusters whilst increasing their size over generations as the most fit individuals diffuse over the population.



(a) Neighborhood Communications *(b) Virtual Islands*

Figure 11: A Diffusion GA

The first attempt at a massively-parallel fine-grained GA known to the authors was by Robertson [45] in 1987. Robertson used a SIMD Connection Machine and assigned one individual per processor. However, global selection and recombination was performed on the host machine and the individual processors were only used for function evaluation implementing a massive processor farm (Section 4.1). Even considering the large communications overhead of this scheme, the objective function evaluation was significantly large for huge speed-up to be reported. By 1989 a more subtle and appropriate scheme for the Connection Machine was presented by Manderick and Spiessens [46] and later implemented on an AMT DAP [47]. Individuals were again placed on separate processors in a planar grid, but a local selection strategy based on a neighbourhood fitness distribution was used. This first diffusion algorithm was not only motivated by the desire to use the Connection Machine's architecture more effectively, but also to align the GA more closely with natural biological evolution. Spiessens and Manderick argue that in nature there is no global selection or fitness-distribution. Rather, natural selection is a local phenomenon where individuals find a mate in their local environment. Their implementation is similar to that described here with the exception that one parent is chosen from the local neighbourhood probabilistically on the basis of the neighbourhood fitness function and recombined with a randomly selected mate within the same locality. This implementation was tested on the De Jong test-bed functions and compared with a conventional GA. The results indicate that the lower selective pressure, due to the local selection mechanism, encourages greater exploration of the search space and helps inhibit the early domination of the population by good individuals. The results also showed that the parallel GA was more effective when the objective function was multimodal.

At around the same time, Mühlenbein [48] and Gorges-Schleuter [49] introduced an asynchronous parallel GA, ASPARAGOS. Using ideas from population genetics, the GA was implemented on a connected ladder network, or ring, topology using Transputers with one individual per processor as shown in Fig. 12. An individual's neighbourhood is defined by its mobility, so, for example, given a 'moving radius' of two yields a neighbourhood size of eight.

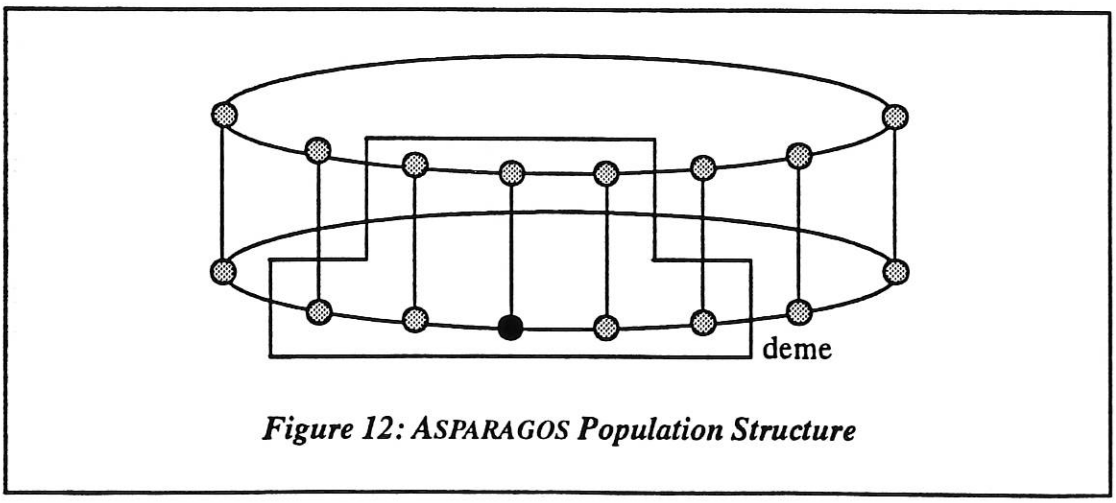


Figure 12: ASPARAGOS Population Structure

ASPARAGOS also employs local hill-climbing when a new individual is created. The rationale behind this idea is that hill climbing can more quickly improve the fitness of an individual than

genetic operators. When applied to the quadratic assignment problem, ASPARAGOS found a new optimum for the largest published problem [48]. In other areas, such as numerical optimization, ASPARAGOS has been found to perform well, and super-linear speed-up has been reported. Although, as Mühlenbein points out, measuring performance of parallel GAs is difficult due to the probabilistic nature of the algorithm and the effective change in the search strategy with varying numbers of processors. Furthermore, in some experiments, a single population GA was not able to locate the minimum at all.

Davidor [50] in his version of the diffusion GA, called ECO GA, used a 2-D grid with wrap-around to produce a population surface in which each individual had eight neighbours. He used a one-at-a-time reproduction strategy and allowed the offspring produced by a particular neighbourhood to replace probabilistically an individual in the vicinity of its parents. Davidor also described the phenomena of niche and speciation where the virtual islands on the population surface represent near local optima.

An interesting variation on local selection is given by Collins and Jefferson [51]. Instead of selecting a mate from within a small local neighbourhood, an individual takes a random walk and selects a mate from individuals encountered on the way. This was found to be highly efficient in the graph partitioning problem and demonstrated a capability of finding multiple optima in a single population. In addition, four metrics where used to measure the differences in evolutionary dynamics between polytypic and panmictic populations. They were the diversity of alleles and genotypes, an inbreeding coefficient measuring the similarity between parents and speed and robustness.

A hierarchically structured distributed GA is described by Voight *et al* [52]that is suitable for fine- and coarse-grain multiprocessor implementations. A population model using different diffusion rates corresponding to the hierarchical structure of the evolution surface is used. Interacting breeding units, or local environments, are organised hierarchically such that interaction within individual local environments is greater than the interactions between individuals in different local environments. Thus, the population structure allows the local interactions of the diffusion GA whilst also modelling the semi-isolation of the migration GA. To accommodate different granularities, local environments or single individuals may reside on separate processors.

More recently, a number of researchers have focused on the nature of the population structure and its effect on the diffusion and convergence characteristics of the GA. For example, Baluja [53] reports on a comparative study of neighbourhood topologies. Three topologies are considered:- a linear neighbourhood, a two dimensional toroidal array and a linear neighbourhood with a rightward discontinuity. When tested on a wide range of test-bed problems the toroidal array neighbourhood consistently outperformed the other population structures. However, for some problems the linear neighbourhoods were found to produce the best convergence pattern. Baluja argues that these results are due to a combination of the effect of genetic mobility and total population size. In particular, the parameters of the different implementations, such as crossover and mutation rates, where held constant and not tuned to a particular neighbourhood structure. However, in an earlier study, Gorges-Schleuter [54] observed that as the borders in one-dimensional population structures are smaller than those in two dimensions, local niches once established tend to survive for longer periods.

The diffusion model provides a finer grain of parallelism than that of the worker/farmer and island models. Its is suitable for implementation on a wide range of parallel architectures from single-bit digital array processors (DAPs) and massively parallel SIMD machines like the Connection Machine, to MIMD computers, such as Transputer networks. There are even reports in the literature of fine-grained GAs being implemented on clusters of networked workstations [55].

The basic operator to support the diffusion model is that of a local neighbourhood selection mechanism. From the examples of diffusion models presented in this Section, it is clear that this local selection results in performance superior to that of global and migration GAs with comparable population sizes. Good solutions are found faster, requiring fewer function evaluations, and different solution niches may be established in the same evolutionary cycle. Furthermore, diffusion appears to implement a more robust search in the presence of deceptive or GA-hard objective functions.

5 Conclusions

In this report, a broad survey of the current trends in GAs has been presented. Many of the variations on the original GA have been discussed, such as different representation and selection strategies, and a number of parallel implementations have been described in detail. From the material presented in this text, it should be clear to the reader that the GA is a powerful and versatile search and optimisation method applicable to a broad spectrum of activities. Further, it is apparent that parallel GAs employing population distribution with local selection and reproduction and some level of genetic mobility are superior to approaches where the population is treated globally.

Clearly, this is a particularly rewarding area for work in parallel algorithm design. Not only can we realise the algorithms efficiently on parallel architectures, but also the paradigm has revealed performance benefits which can even be realised on a sequential machine.

From the preceding discussion of migration and diffusion GAs, the following general points can be noted about parallel GAs when compared with global strategies:

- Parallel GAs typically require less function evaluations to find optimal solutions.
- They have the potential to find multiple optimal solutions.
- They may be synchronous or asynchronous.
- The implementation of parallel GAs may be adapted to efficiently exploit different parallel architectures.
- They have a higher degree of robustness.
- Parallel GAs may be made fault-tolerant.
- Parallel GAs come closer to the biological metaphor of evolution.

6 References

[1] J. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1975.

[2] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley Publishing Company, January 1989.

[3] C. L. Karr, "Design of an Adaptive Fuzzy Logic Controller Using a Genetic Algorithm", *Proc. ICGA 4*, pp. 450-457, 1991.

[4] R. B. Holstien, *Artificial Genetic Adaptation in Computer Control Systems*, PhD Thesis, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, 1971.

[5] R. A. Caruana and J. D. Schaffer, "Representation and Hidden Bias: Gray vs. Binary Coding", *Proc. 6th Int. Conf. Machine Learning*, pp153-161, 1988.

[6] W. E. Schmitendorgf, O. Shaw, R. Benson and S. Forrest, "Using Genetic Algorithms for Controller Design: Simultaneous Stabilization and Eigenvalue Placement in a Region", *Technical Report No. CS92-9*, Dept. Computer Science, College of Engineering, University of New Mexico, 1992.

[7] M. F. Bramlette, "Initialization, Mutation and Selection Methods in Genetic Algorithms for Function Optimization", *Proc ICGA 4*, pp. 100-107, 1991.

[8] C. B. Lucasius and G. Kateman, "Towards Solving Subset Selection Problems with the Aid of the Genetic Algorithm", In *Parallel Problem Solving from Nature 2*, R. Männer and B. Manderick, (Eds.), pp. 239-247, Amsterdam: North-Holland, 1992.

[9] A. H. Wright, "Genetic Algorithms for Real Parameter Optimization", In *Foundations of Genetic Algorithms*, J. E. Rawlins (Ed.), Morgan Kaufmann, pp. 205-218, 1991.

[10] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer Verlag, 1992.

[11] T. Bäck, F. Hoffmeister and H.-P. Schwefel, "A Survey of Evolution Strategies", *Proc. ICGA 4*, pp. 2-10, 1991.

[12] J. J. Grefenstette, "Incorporating Problem Specific Knowledge into Genetic Algorithms", In *Genetic Algorithms and Simulated Annealing*, pp. 42-60, L. Davis (Ed.), Morgan Kaufmann, 1987.

[13] D. Whitley, K. Mathias and P. Fitzhorn, "Delta Coding: An Iterative Search Strategy for Genetic Algorithms", *Proc. ICGA 4*, pp. 77-84, 1991.

[14] K. A. De Jong, *Analysis of the Behaviour of a Class of Genetic Adaptive Systems*, PhD Thesis, Dept. of Computer and Communication Sciences, University of Michigan, Ann Arbor, 1975.

[15] J. E. Baker, "Adaptive Selection Methods for Genetic Algorithms", *Proc. ICGA 1*, pp. 101-111, 1985.

[16] J. E. Baker, "Reducing bias and inefficiency in the selection algorithm", *Proc. ICGA 2*, pp. 14-21, 1987.

[17] L. Booker, "Improving search in genetic algorithms," In *Genetic Algorithms and Simulated Annealing*, L. Davis (Ed.), pp. 61-73, Morgan Kaufmann Publishers, 1987.

[18] W. M. Spears and K. A. De Jong, "An Analysis of Multi-Point Crossover", In *Foundations of Genetic Algorithms*, J. E. Rawlins (Ed.), pp. 301-315, 1991.

[19] G. Syswerda, "Uniform crossover in genetic algorithms", *Proc. ICGA 3*, pp. 2-9, 1989.

[20] W. M. Spears and K. A. De Jong, "On the Virtues of Parameterised Uniform Crossover", *Proc. ICGA 4*, pp.230-236, 1991.

[21] R. A. Caruana, L. A. Eshelman, J. D. Schaffer, "Representation and hidden bias II: Eliminating defining length bias in genetic search via shuffle crossover", In *Eleventh International Joint Conference on Artificial Intelligence*, N. S. Sridharan (Ed.), Vol. 1, pp. 750-755, Morgan Kaufmann Publishers, 1989.

[22] H. Mühlenbein and D. Schlierkamp-Voosen, "Predictive Models for the Breeder Genetic Algorithm", *Evolutionary Computation*, Vol. 1, No. 1, pp. 25-49, 1993.

[23] H. Furuya and R. T. Haftka, "Genetic Algorithms for Placing Actuators on Space Structures", *Proc. ICGA 5*, pp. 536-542, 1993.

[24] C. Z. Janikow and Z. Michalewicz, "An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms", *Proc. ICGA 4*, pp. 31-36, 1991.

[25] D. M. Tate and A. E. Smith, "Expected Allele Convergence and the Role of Mutation in Genetic Algorithms", *Proc. ICGA* 5, pp.31-37, 1993.

[26] L. Davis, "Adapting Operator Probabilities in Genetic Algorithms", *Proc. ICGA 3*, pp. 61-69, 1989.

[27] T. C. Fogarty, "Varying the Probability of Mutation in the Genetic Algorithm", *Proc. ICGA 3*, pp. 104-109, 1989.

[28] K. A. De Jong and J. Sarma, "Generation Gaps Revisited", In *Foundations of Genetic Algorithms 2*, L. D. Whitley (Ed.), Morgan Kaufmann Publishers, 1993.

[29] D. Whitley, "The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocations of Reproductive Trials is Best", *Proc. ICGA 3*, pp. 116-121, 1989.

[30] R. Huang and T. C. Fogarty, "Adaptive Classification and Control-Rule Optimization Via a Learning Algorithm for Controlling a Dynamic System", *Proc. 30th Conf. Decision and Control*, Brighton, England, pp. 867-868, 1991.

[31] T. C. Fogarty, "An Incremental Genetic Algorithm for Real-Time Learning", *Proc. 6th Int. Workshop on Machine Learning*, pp. 416-419, 1989.

[32] N. Dodd, D. Macfarlane and C. Marland, "Optimization of Artificial Neural Network structure Using Genetic Techniques Implemented on Multiple Transputers", *Transputing '91*, P.Welch, D. Stiles, T. L. Kunii and A. Bakkers (Eds.) Vol. 2 pp. 687-700, IOS Press, 1991.

[33] P. B. Grosso, *Computer Simulation of Genetic Adaptation: Parallel Subcomponent Interaction in a Multilocus Model*, PhD Thesis, University of Michigan, 1985.

[34] C. B. Petty, M. R. Leuze and J. J. Grefenstette, "A Parallel Genetic Algorithm", *Proc. ICGA 2*, pp. 155-161, 1987.

[35] S. W. Wilson, "Knowledge Growth in an Artificial Animal", *Proc. ICGA 1*, pp. 16-23, 1985.

[36] J. D. Schaffer, "Multiple Objective Optimization with Vector Evaluated Genetic Algortithms", *Proc. ICGA 1*, pp. 93-100, 1985.

[37] R. Tanse, "Parallel Genetic Algorithm for a Hypercube", *Proc. ICGA 2*, pp. 177-183, 1987.

[38] R. Tanse, "Distributed Genetic Algorithms", *Proc. ICGA 3*, pp.434-439, 1989.

[39] T. Starkweather, D. Whitley and K. Mathias, "Optimization Using Distributed Genetic Algorithms", *Proc. Parallel Problem Solving From Nature 1*, Lecture Notes in Computer Science No. 496, pp. 176-185, Springer-Verlag, 1990.

[40] J. P. Cohoon, W. N. Martin and D. S. Richards, "A Multi-population Genetic Algorithm for Solving the K-Partition Problem on Hyper-cubes", *Proc. ICGA 4*, pp. 244-248, 1991.

[41] H. Mühlenbein, M. Schomisch and J. Born, "The Parallel Genetic Algorithm as a Function Optimizer", *Parallel Computing*, No. 17, pp. 619-632, 1991.

[42] J. C. Potts, T. D. Giddens and S. B. Yadav, "The Development and Evaluation of an Improved Genetic Algorithm Based on Migration and Artificial Selection", *IEEE Trans. Systems, Man and Cybernetics*, Vol. 24. No. 1, pp.73-86, 1994.

[43] J. D. Schaffer, R. A. Caruana, L. J. Eshelman and R. Das, "A Study of Control Parameters Affecting On-line Performance of Genetic Algorithms for Function Optimization", *Proc. ICGA 3*, pp. 36-40, 1989.

[44] S. Wright, *Evolution and the Genetics of Populations*, Vol. 2, University of Chicago Press, 1969.

[45] G. Robertson, "Parallel Implementation of Genetic Algorithms in a Classifier System", In *Genetic Algorithms and Simulated Annealing*, L. Davis (Ed.), pp. 129-140, Pitman, London, 1987.

[46] B. Manderick and P. Spiessens, "Fine-Grained Parallel Genetic Algorithms", *Proc. ICGA 3*, pp. 428-433, 1989.

[47] P. Spiessens and B. Manderick, "A Massively Parallel Genetic Algorithm: Implementation and First Analysis", *Proc. ICGA 4*, pp. 279-286, 1991.

[48] H. Mühlenbein, "Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization", In *Parallelism, Learning, Evolution*, Workshop on Evolutionary Models and Strategies, J. D. Becker, I. Eisele and F. W. Mundemann (Eds.) Lecture Notes in Artificial Intelligence, No. 565, pp398-406, Springer-Verlag, 1989.

[49] M. Gorges-Schleuter, "ASPARAGOS An Asynchronous Parallel Genetic Optimization Stratagey", *Proc. ICGA 3*, pp.422-427, 1989.

[50] Y. Davidor, "A Naturally Occurring Niche and Species Phenomenon: The Model and First Results", *Proc. ICGA 4,* pp. 257- 263, 1991.

[51] R. J. Collins and D. R. Jefferson, "Selection in Massively Parallel Genetic Algorithms", *Proc. ICGA 4*, pp. 249-256, 1991.

[52] H. -M. Voight I. Santibáñez-Koref and J. Born, "Hierarchically Structured Distributed Genetic Algorithms", In *Parallel Problem Solving from Nature 2*, R. Männer and B. Manderick, (Eds.), pp. 145-154, Amsterdam: North-Holland, 1992.

[53] S. Baluja, "Structure and Performance of Fine-Grain Parallelism in Genetic Search", *Proc. ICGA 5*, pp. 155-162, 1993.

[54] M. Georges-Schleuter, "Comparison of Local Mating Strategies in Massively Parallel Genetic Algorithms", In *Parallel Problem Solving from Nature 2*, R. Männer and B. Manderick, (Eds.), pp. 553-562, Amsterdam: North-Holland, 1992.

[55] T. Maruyama, T. Hirose and A. Konagaya, "A Fine-Grained Parallel Genetic Algorithm for Distributed Parallel Systems", *Proc. ICGA 5*, pp. 184-190, 1993.