Chapter 4

GENETIC ALGORITHMS

Kumara Sastry, David Goldberg
University of Illinois, USA

Graham Kendall
University of Nottingham, UK

4.1 INTRODUCTION

Genetic algorithms (GAs) are search methods based on principles of natural selection and genetics (Fraser, 1957; Bremermann, 1958; Holland, 1975). We start with a brief introduction to simple genetic algorithms and associated terminology.

GAs encode the decision variables of a search problem into finite-length strings of alphabets of certain cardinality. The strings which are candidate solutions to the search problem are referred to as *chromosomes*, the alphabets are referred to as *genes* and the values of genes are called *alleles*. For example, in a problem such as the traveling salesman problem, a chromosome represents a route, and a gene may represent a city. In contrast to traditional optimization techniques, GAs work with coding of parameters, rather than the parameters themselves.

To evolve good solutions and to implement natural selection, we need a measure for distinguishing good solutions from bad solutions. The measure could be an *objective* function that is a mathematical model or a computer simulation, or it can be a *subjective* function where humans choose better solutions over worse ones. In essence, the fitness measure must determine a candidate solution's relative fitness, which will subsequently be used by the GA to guide the evolution of good solutions.

Another important concept of GAs is the notion of population. Unlike traditional search methods, genetic algorithms rely on a population of candidate solutions. The population size, which is usually a user-specified parameter, is one of the important factors affecting the scalability and performance of genetic algorithms. For example, small population sizes might lead to premature

convergence and yield substandard solutions. On the other hand, large population sizes lead to unnecessary expenditure of valuable computational time.

Once the problem is encoded in a chromosomal manner and a fitness measure for discriminating good solutions from bad ones has been chosen, we can start to *evolve* solutions to the search problem using the following steps:

1 *Initialization.* The initial population of candidate solutions is usually generated randomly across the search space. However, domain-specific knowledge or other information can be easily incorporated.

2 *Evaluation.* Once the population is initialized or an offspring population is created, the fitness values of the candidate solutions are evaluated.

3 *Selection.* Selection allocates more copies of those solutions with higher fitness values and thus imposes the survival-of-the-fittest mechanism on the candidate solutions. The main idea of selection is to prefer better solutions to worse ones, and many selection procedures have been proposed to accomplish this idea, including roulette-wheel selection, stochastic universal selection, ranking selection and tournament selection, some of which are described in the next section.

4 *Recombination.* Recombination combines parts of two or more parental solutions to create new, possibly better solutions (i.e. offspring). There are many ways of accomplishing this (some of which are discussed in the next section), and competent performance depends on a properly designed recombination mechanism. The offspring under recombination will not be identical to any particular parent and will instead combine parental traits in a novel manner (Goldberg, 2002).

5 *Mutation.* While recombination operates on two or more parental chromosomes, mutation locally but randomly modifies a solution. Again, there are many variations of mutation, but it usually involves one or more changes being made to an individual's trait or traits. In other words, mutation performs a random walk in the vicinity of a candidate solution.

6 *Replacement.* The offspring population created by selection, recombination, and mutation replaces the original parental population. Many replacement techniques such as elitist replacement, generation-wise replacement and steady-state replacement methods are used in GAs.

7 Repeat steps 2–6 until a terminating condition is met.

Goldberg (1983, 1999a, 2002) has likened GAs to mechanistic versions of certain modes of human innovation and has shown that these operators when analyzed individually are ineffective, but when combined together they can

work well. This aspect has been explained with the concepts of the *fundamental intuition* and *innovation intuition*. The same study compares a combination of selection and mutation to *continual improvement* (a form of hill climbing), and the combination of selection and recombination to *innovation* (*cross-fertilizing*). These analogies have been used to develop a design-decomposition methodology and so-called *competent* GAs—that solve hard problems quickly, reliably, and accurately—both of which are discussed in the subsequent sections.

This chapter is organized as follows. The next section provides details of individual steps of a typical genetic algorithm and introduces several popular genetic operators. Section 4.1.2 presents a principled methodology of designing competent genetic algorithms based on decomposition principles. Section 4.1.3 gives a brief overview of designing principled efficiency-enhancement techniques to speed up genetic and evolutionary algorithms.

4.1.1 Basic Genetic Algorithm Operators

In this section we describe some of the selection, recombination, and mutation operators commonly used in genetic algorithms.

4.1.1.1 Selection Methods. Selection procedures can be broadly classified into two classes as follows.

Fitness Proportionate Selection This includes methods such as roulette-wheel selection (Holland, 1975; Goldberg, 1989b) and stochastic universal selection (Baker, 1985; Grefenstette and Baker, 1989). In roulette-wheel selection, each individual in the population is assigned a roulette wheel slot sized in proportion to its fitness. That is, in the biased roulette wheel, good solutions have a larger slot size than the less fit solutions. The roulette wheel is spun to obtain a reproduction candidate. The roulette-wheel selection scheme can be implemented as follows:

1 Evaluate the fitness, f_i, of each individual in the population.

2 Compute the probability (slot size), p_i, of selecting each member of the population: $p_i = f_i / \sum_{j=1}^{n} f_j$, where n is the population size.

3 Calculate the cumulative probability, q_i, for each individual: $q_i = \sum_{j=1}^{i} p_j$.

4 Generate a uniform random number, $r \in (0, 1]$.

5 If $r < q_1$ then select the first chromosome, x_1, else select the individual x_i such that $q_{i-1} < r \leq q_i$.

6 Repeat steps 4–5 n times to create n candidates in the mating pool.

To illustrate, consider a population with five individuals ($n = 5$), with the fitness values as shown in the table below. The total fitness, $\sum_{j=1}^{n} f_j = 28 + 18 + 14 + 9 + 26 = 95$. The probability of selecting an individual and the corresponding cumulative probabilities are also shown in the table below.

Chromosome #	1	2	3	4	5
Fitness, f	28	18	14	9	26
Probability, p_i	28/95 = 0.295	0.189	0.147	0.095	0.274
Cumulative probability, q_i	0.295	0.484	0.631	0.726	1.000

Now if we generate a random number r, say 0.585, then the third chromosome is selected as $q_2 = 0.484 < 0.585 \leq q_3 = 0.631$.

Ordinal Selection This includes methods such as tournament selection (Goldberg et al., 1989b), and truncation selection (Mühlenbein and Schlierkamp-Voosen, 1993). In tournament selection, s chromosomes are chosen at random (either with or without replacement) and entered into a tournament against each other. The fittest individual in the group of k chromosomes wins the tournament and is selected as the parent. The most widely used value of s is 2. Using this selection scheme, n tournaments are required to choose n individuals. In truncation selection, the top $(1/s)$th of the individuals get s copies each in the mating pool.

4.1.1.2 Recombination (Crossover) Operators. After selection, individuals from the mating pool are recombined (or crossed over) to create new, hopefully better, offspring. In the GA literature, many crossover methods have been designed (Goldberg, 1989b; Booker et al., 1997; Spears, 1997) and some of them are described in this section. Many of the recombination operators used in the literature are problem-specific and in this section we will introduce a few generic (problem independent) crossover operators. It should be noted that while for *hard* search problems, many of the following operators are not scalable, they are very useful as a first option. Recently, however, researchers have achieved significant success in designing scalable recombination operators that adapt linkage which will be briefly discussed in Section 4.1.2.

In most recombination operators, two individuals are randomly selected and are recombined with a probability p_c, called the crossover probability. That is, a uniform random number, r, is generated and if $r \leq p_c$, the two randomly selected individuals undergo recombination. Otherwise, that is, if $r > p_c$, the two offspring are simply copies of their parents. The value of p_c can either be set experimentally, or can be set based on schema-theorem principles (Goldberg, 1989b, 2002; Goldberg and Sastry, 2001).

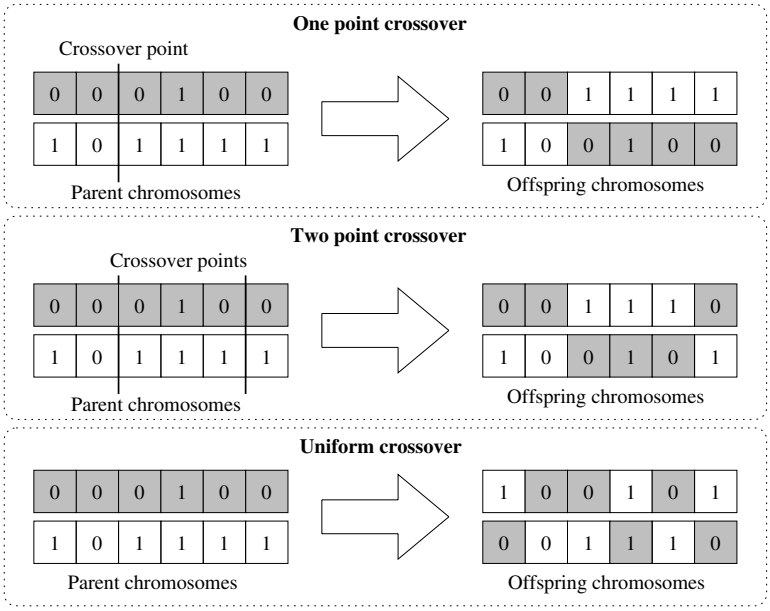

Figure 4.1. One-point, two-point, and uniform crossover methods.

***k*-point Crossover** One-point, and two-point crossovers are the simplest and most widely applied crossover methods. In one-point crossover, illustrated in Figure 4.1, a crossover site is selected at random over the string length, and the alleles on one side of the site are exchanged between the individuals. In two-point crossover, two crossover sites are randomly selected. The alleles between the two sites are exchanged between the two randomly paired individuals. Two-point crossover is also illustrated in Figure 4.1. The concept of one-point crossover can be extended to k-point crossover, where k crossover points are used, rather than just one or two.

Uniform Crossover Another common recombination operator is uniform crossover (Syswerda, 1989; Spears and De Jong, 1994). In uniform crossover, illustrated in Figure 4.1, every allele is exchanged between the a pair of randomly selected chromosomes with a certain probability, p_e, known as the swapping probability. Usually the swapping probability value is taken to be 0.5.

Uniform Order-Based Crossover The k-point and uniform crossover methods described above are not well suited for search problems with permutation codes such as the ones used in the traveling salesman problem. They often create offspring that represent invalid solutions for the search problem. Therefore,

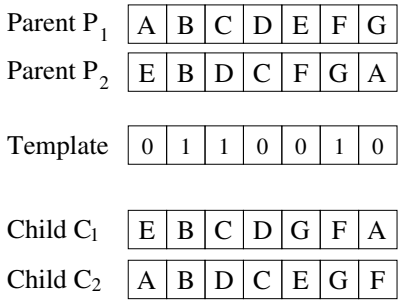

Figure 4.2. Illustration of uniform order crossover.

when solving search problems with permutation codes, a problem-specific repair mechanism is often required (and used) in conjunction with the above recombination methods to always create valid candidate solutions.

Another alternative is to use recombination methods developed specifically for permutation codes, which always generate valid candidate solutions. Several such crossover techniques are described in the following paragraphs starting with the uniform order-based crossover.

In uniform order-based crossover, two parents (say P_1 and P_2) are randomly selected and a random binary template is generated (see Figure 4.2). Some of the genes for offspring C_1 are filled by taking the genes from parent P_1 where there is a one in the template. At this point we have C_1 partially filled, but it has some "gaps". The genes of parent P_1 in the positions corresponding to zeros in the template are taken and sorted in the same order as they appear in parent P_2. The sorted list is used to fill the gaps in C_1. Offspring C_2 is created by using a similar process (see Figure 4.2).

Order-Based Crossover The order-based crossover operator (Davis, 1985) is a variation of the uniform order-based crossover in which two parents are randomly selected and two random crossover sites are generated (see Figure 4.3). The genes between the cut points are copied to the children. Starting from the second crossover site copy the genes that are not already present in the offspring from the alternative parent (the parent other than the one whose genes are copied by the offspring in the initial phase) in the order they appear. For example, as shown in Figure 4.3, for offspring C_1, since alleles C, D, and E are copied from the parent P_1, we get alleles B, G, F, and A from the parent P_2. Starting from the second crossover site, which is the sixth gene, we copy alleles B and G as the sixth and seventh genes respectively. We then wrap around and copy alleles F and A as the first and second genes.

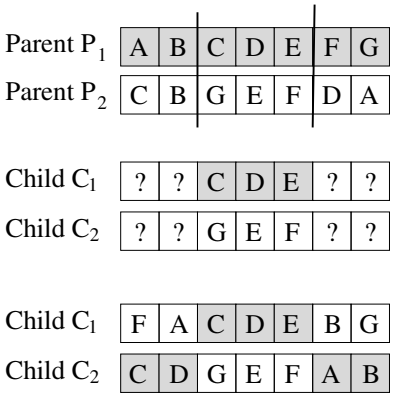

Figure 4.3. Illustration of order-based crossover.

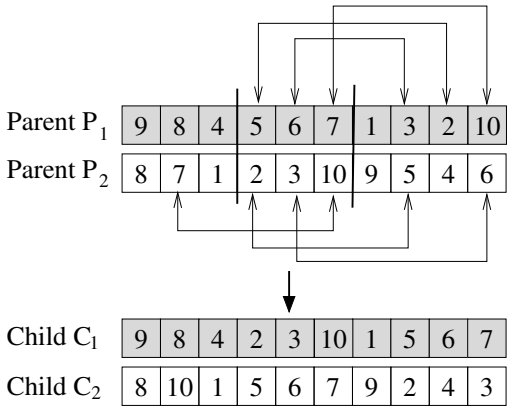

Figure 4.4. Illustration of partially matched crossover.

Partially Matched Crossover (PMX) Apart from always generating valid offspring, the PMX operator (Goldberg and Lingle, 1985) also preserves orderings within the chromosome. In PMX, two parents are randomly selected and two random crossover sites are generated. Alleles within the two crossover sites of a parent are exchanged with the alleles corresponding to those mapped by the other parent. For example, as illustrated in Figure 4.4 (reproduced from Goldberg (1989b) with permission), looking at parent P_1, the first gene within the two crossover sites, 5, maps to 2 in P_2. Therefore, genes 5 and 2 are swapped in P_1. Similarly we swap 6 and 3, and 10 and 7 to create the offspring C_1. After all exchanges it can be seen that we have achieved a duplication of the ordering of one of the genes in between the crossover point within the opposite chromosome, and vice versa.

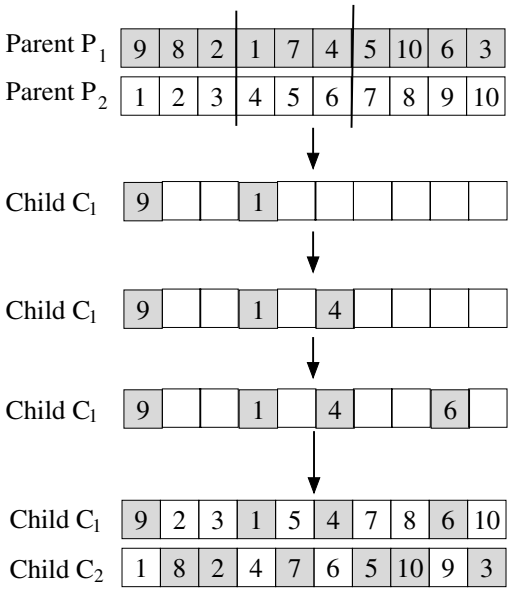

Figure 4.5. Illustration of cycle crossover.

Cycle Crossover (CX) We describe cycle crossover (Oliver et al., 1987) with help of a simple illustration (reproduced from Goldberg (1989b) with permission). Consider two randomly selected parents P_1 and P_2 as shown in Figure 4.5 that are solutions to a traveling salesman problem. The offspring C_1 receives the first variable (representing city 9) from P_1. We then choose the variable that maps onto the same position in P_2. Since city 9 is chosen from P_1 which maps to city 1 in P_2, we choose city 1 and place it into C_1 in the same position as it appears in P_1 (fourth gene), as shown in Figure 4.5. City 1 in P_1 now maps to city 4 in P_2, so we place city 4 in C_1 in the same position it occupies in P_1 (sixth gene). We continue this process once more and copy city 6 to the ninth gene of C_1 from P_1. At this point, since city 6 in P_1 maps to city 9 in P_2, we should take city 9 and place it in C_1, but this has already been done, so we have completed a cycle; which is where this operator gets its name. The missing cities in offspring C_1 is filled from P_2. Offspring C_2 is created in the same way by starting with the first city of parent P_2 (see Figure 4.5).

4.1.1.3 Mutation Operators. If we use a crossover operator, such as one-point crossover, we may get better and better chromosomes but the problem is, if the two parents (or worse, the entire population) has the same allele at a given gene then one-point crossover will not change that. In other words, that gene will have the same allele forever. Mutation is designed to

overcome this problem in order to add diversity to the population and ensure that it is possible to explore the entire search space.

In evolutionary strategies, mutation is the primary variation/search operator. For an introduction to evolutionary strategies see, for example, Bäck et al. (1997). Unlike evolutionary strategies, mutation is often the secondary operator in GAs, performed with a low probability. One of the most common mutations is the bit-flip mutation. In bitwise mutation, each bit in a binary string is changed (a 0 is converted to 1, and vice versa) with a certain probability, p_m, known as the mutation probability. As mentioned earlier, mutation performs a random walk in the vicinity of the individual. Other mutation operators, such as problem-specific ones, can also be developed and are often used in the literature.

4.1.1.4 Replacement. Once the new offspring solutions are created using crossover and mutation, we need to introduce them into the parental population. There are many ways we can approach this. Bear in mind that the parent chromosomes have already been selected according to their fitness, so we are hoping that the children (which includes parents which did not undergo crossover) are among the fittest in the population and so we would hope that the population will gradually, on average, increase its fitness. Some of the most common replacement techniques are outlined below.

Delete-all This technique deletes all the members of the current population and replaces them with the same number of chromosomes that have just been created. This is probably the most common technique and will be the technique of choice for most people due to its relative ease of implementation. It is also parameter-free, which is not the case for some other methods.

Steady-state This technique deletes n old members and replaces them with n new members. The number to delete and replace, n, at any one time is a parameter to this deletion technique. Another consideration for this technique is deciding which members to delete from the current population. Do you delete the worst individuals, pick them at random or delete the chromosomes that you used as parents? Again, this is a parameter to this technique.

Steady-state-no-duplicates This is the same as the steady-state technique but the algorithm checks that no duplicate chromosomes are added to the population. This adds to the computational overhead but can mean that more of the search space is explored.

4.1.2 Competent Genetic Algorithms

While using innovation for explaining the working mechanisms of GAs is very useful, as a design metaphor it poses difficulty as the processes of innovation are themselves not well understood. However, if we want GAs to successfully solve increasingly difficult problems across a wide spectrum of areas, we need a principled, but mechanistic way of designing genetic algorithms. The last few decades have witnessed great strides toward the development of so-called *competent* genetic algorithms—GAs that solve hard problems, quickly, reliably, and accurately (Goldberg, 1999a). From a computational standpoint, the existence of competent GAs suggests that many difficult problems can be solved in a scalable fashion. Furthermore, it significantly reduces the burden on a user to decide on a good coding or a good genetic operator that accompanies many GA applications. If the GA can adapt to the problem, there is less reason for the user to have to adapt the problem, coding, or operators to the GA.

In this section we briefly review some of the key lessons of competent GA design. Specifically, we restrict the discussion to selectorecombinative GAs and focus on the cross-fertilization type of innovation and briefly discuss key facets of competent GA design. Using Holland's notion of a building block (Holland, 1975), Goldberg proposed decomposing the problem of designing a competent selectorecombinative GA (Goldberg et al., 1992a). This design decomposition has been explained in detail elsewhere (Goldberg, 2002), but is briefly reviewed below.

Know that GAs Process Building Blocks The primary idea of selectorecombinative GA theory is that genetic algorithms work through a mechanism of *decomposition* and *reassembly*. Holland (1975) called well-adapted sets of features that were components of effective solutions *building blocks* (BBs). The basic idea is that GAs (1) implicitly identify building blocks or sub-assemblies of good solutions, and (2) recombine different sub-assemblies to form very high performance solutions.

Understand BB Hard Problems From the standpoint of cross-fertilizing innovation, problems that are hard have BBs that are hard to acquire. This may be because the BBs are complex, hard to find, or because different BBs are hard to separate, or because low-order BBs may be *misleading* or *deceptive* (Goldberg, 1987, 1989a; Goldberg et al., 1992b; Deb and Goldberg, 1994).

Understand BB Growth and Timing Another key idea is that BBs or notions exist in a kind of competitive *market economy of ideas*, and steps must be taken to ensure that the best ones (1) grow and take over a dom-

inant market share of the population, and (2) the growth rate can neither be too fast, nor too slow.

The growth in market share can be easily satisfied (Goldberg and Sastry, 2001) by appropriately setting the crossover probability, p_c, and the selection pressure, s, so that

$$p_c \leq \frac{1 - s^{-1}}{\epsilon} \tag{4.1}$$

where ϵ is the probability of BB disruption.

Two other approaches have been used in understanding time. It is not appropriate in a basic tutorial like this to describe them in detail, but we give a few example references for the interested reader.

Takeover time models, where the dynamics of the best individual is modeled (Goldberg and Deb, 1991; Sakamoto and Goldberg, 1997; Cantú-Paz, 1999; Rudolph, 2000).

Selection-intensity models, where approaches similar to those in quantitative genetics (Bulmer, 1985) are used and the dynamics of the average fitness of the population is modeled (Mühlenbein and Schlierkamp-Voosen, 1993; Thierens and Goldberg, 1994a, 1994b; Bäck, 1995; Miller and Goldberg, 1995, 1996a; Voigt et al., 1996).

The time models suggest that for a problem of size ℓ, with all BBs of equal importance or salience, the convergence time, t_c, of GAs is given by Miller and Goldberg (1995) to be

$$t_c = \frac{\pi}{2I}\sqrt{\ell} \tag{4.2}$$

where I is the selection intensity (Bulmer, 1985), which is a parameter dependent on the selection method and selection pressure. For tournament selection, I can be approximated in terms of s by the following relation (Blickle and Thiele, 1995):

$$I = \sqrt{2\left(\log(s) - \log\left(\sqrt{4.14\log(s)}\right)\right)} \tag{4.3}$$

On the other hand, if the BBs of a problem have different salience, then the convergence time scales-up differently. For example, when the BBs of a problem are exponentially scaled, with a particular BB being exponentially better than the others, then the convergence time, t_c, of a GA is linear with the problem size (Thierens et al., 1998) and can be

represented as follows:

$$t_c = \frac{-\log 2}{\log\left(1 - I/\sqrt{3}\right)}\ell \tag{4.4}$$

To summarize, the convergence time of GAs is $\mathcal{O}\left(\sqrt{\ell}\right)$–$\mathcal{O}(\ell)$ (see Chapter 1, Introduction, for an explanation of the $\mathcal{O}$ notation).

Understand BB Supply and Decision Making One role of the population is to ensure adequate *supply* of the raw building blocks in a population. Randomly generated populations of increasing size will, with higher probability, contain larger numbers of more complex BBs (Holland, 1975; Goldberg, 1989c; Goldberg et al., 2001). For a problem with m building blocks, each consisting of k alphabets of cardinality χ, the population size, n, required to ensure the presence of at least one copy of all the raw building blocks is given by Goldberg et al. (2001) as

$$n = \chi^k \log m + k\chi^k \log \chi \tag{4.5}$$

Just ensuring the raw supply is not enough, decision making among different, competing notions (BBs) is *statistical* in nature, and as we increase the population size, we increase the likelihood of making the best possible decisions (De Jong, 1975; Goldberg and Rudnick, 1991; Goldberg et al., 1992a; Harik et al., 1999). For an additively decomposable problem with m building blocks of size k each, the population size required to not only ensure supply, but also ensure correct decision making is approximately given by Harik et al. (1999) as

$$n = -\frac{\sqrt{\pi}}{2}\frac{\sigma_{BB}}{d}2^k\sqrt{m}\log\alpha \tag{4.6}$$

where d/σ_{BB} is the signal-to-noise ratio (Goldberg et al., 1992a), and α is the probability of incorrectly deciding among competing building blocks. In essence, the population-sizing model consists of the following components:

- *Competition complexity*, quantified by the total number of competing building blocks, 2^k.
- *Subcomponent complexity*, quantified by the number of building blocks, m.
- *Ease of decision making*, quantified by the signal-to-noise ratio, d/σ_{bb}.
- *Probabilistic safety factor*, quantified by the coefficient $-\log\alpha$.

On the other hand, if the building blocks are exponentially scaled, the population size, n, scales as (Rothlauf, 2002; Thierens et al., 1998; Goldberg, 2002)

$$n = -c_o \frac{\sigma_{BB}}{d} 2^k m \log \alpha \tag{4.7}$$

where c_o is a constant dependent on the drift effects (Crow and Kimura, 1970; Goldberg and Segrest, 1987; Asoh and Mühlenbein, 1994).

To summarize, the complexity of the population size required by GAs is $\mathcal{O}\left(2^k\sqrt{m}\right)$–$\mathcal{O}\left(2^k m\right)$.

Identify BBs and Exchange Them Perhaps the most important lesson of current research in GAs is that the *identification and exchange of BBs* is the critical path to innovative success. First-generation GAs usually fail in their ability to promote this exchange reliably. The primary design challenge to achieving competence is the need to identify and promote effective BB exchange. Theoretical studies using the *facetwise* modeling approach (Thierens, 1999; Sastry and Goldberg, 2002, 2003) have shown that while fixed recombination operators such as uniform crossover, due to inadequacies of effective identification and exchange of BBs, demonstrate polynomial scalability on simple problems, they scale-up exponentially with problem size on boundedly-difficult problems. The mixing models also yield a *control map* delineating the region of good performance for a GA. Such a control map can be a useful tool in visualizing GA sweet-spots and provide insights in parameter settings (Goldberg, 1999a). This is in contrast to recombination operators that can automatically and adaptively identify and exchange BBs, which scale up polynomially (subquadratically–quadratically) with problem size.

Efforts in the principled design of effective BB identification and exchange mechanisms have led to the development of competent genetic algorithms. Competent GAs solve hard problems quickly, reliably, and accurately. Hard problems are loosely defined as those problems that have large sub-solutions that cannot be decomposed into simpler sub-solutions, or have badly scaled sub-solutions, or have numerous local optima, or are subject to a high stochastic noise. While designing a competent GA, the objective is to develop an algorithm that can solve problems with bounded difficulty and exhibit a polynomial (usually subquadratic) scale-up with the problem size.

Interestingly, the mechanics of competent GAs vary widely, but the principles of innovative success are invariant. Competent GA design began with the development of the *messy genetic algorithm* (Goldberg et al., 1989), culminating in 1993 with the *fast messy GA* (Goldberg et al., 1993). Since those early scalable results, a number of competent GAs have been constructed using different mechanism styles. We will categorize these approaches and provide

some references for the interested reader, but a detailed treatment is beyond the scope of this tutorial.

Perturbation techniques, such as the messy GA (Goldberg et al., 1989), the fast messy GA (Goldberg et al., 1993), the gene expression messy GA (Kargupta, 1996), the linkage identification by nonlinearity check/linkage identification by detection GA (Munetomo and Goldberg, 1999; Heckendorn and Wright, 2004), and the dependency structure matrix driven genetic algorithm (Yu et al., 2003).

Linkage adaptation techniques, such as the linkage learning GA (Harik and Goldberg, 1997; Harik, 1997).

Probabilistic model building techniques, such as population based incremental learning (Baluja, 1994), the univariate model building algorithm (Mühlenbein and Paaß, 1996), the compact GA (Harik et al., 1998), the extended compact GA (Harik, 1999), the Bayesian optimization algorithm (Pelikan et al., 2000), the iterated distribution estimation algorithm (Bosman and Thierens, 1999), and the hierarchical Bayesian optimization algorithm (Pelikan and Goldberg, 2001). More details regarding these algorithms are given elsewhere (Pelikan et al., 2002; Larrañaga and Lozano, 2002; Pelikan, 2005).

4.1.3 Enhancement of Genetic Algorithms to Improve Efficiency and/or Effectiveness

The previous section presented a brief account of competent GAs. These GA designs have shown promising results and have successfully solved hard problems requiring only a subquadratic number of function evaluations. In other words, competent GAs usually solve an ℓ-variable search problem, requiring only $\mathcal{O}(\ell^2)$ number of function evaluations. While competent GAs take problems that were intractable with first-generation GAs and render them tractable, for large-scale problems, the task of computing even a subquadratic number of function evaluations can be daunting. If the fitness function is a complex simulation, model, or computation, then a single evaluation might take hours, even days. For such problems, even a subquadratic number of function evaluations is very high. For example, consider a 20-bit search problem and assume that a fitness evaluation takes one hour. We will require about half a month to solve the problem. This places a premium on a variety of *efficiency enhancement techniques*. Also, it is often the case that a GA needs to be integrated with problem-specific methods in order to make the approach really effective for a particular problem. The literature contains a very large number of papers which discuss enhancements of GAs. Once again, a detailed discussion is well beyond the scope of the tutorial, but we provide four broad

categories of GA enhancement and examples of appropriate references for the interested reader.

Parallelization, where GAs are run on multiple processors and the computational resource is distributed among these processors (Cantú-Paz, 1997, 2000). Evolutionary algorithms are by *nature* parallel, and many different parallelization approaches can be used, such as a simple master–slave parallel GA (Grefenstette, 1981), a coarse-grained architecture (Pettey et al., 1987), a fine-grained architecture (Robertson, 1987; Gorges-Schleuter, 1989; Manderick and Spiessens, 1989), or a hierarchical architecture (Goldberg, 1989b; Gorges-Schleuter, 1997; Lin et al., 1997). Regardless of how parallelization is carried out, the key idea is to distribute the computational load on several processors thereby speeding-up the overall GA run. Moreover, there exists a principled design theory for developing an efficient parallel GA and optimizing the key facts of parallel architecture, connectivity, and deme size (Cantú-Paz, 2000).

For example, when the function evaluation time, T_f, is much greater than the communication time, T_c, which is very often the case, then a simple master–slave parallel GA—where the fitness evaluations are distributed over several processors and the rest of the GA operations are performed on a single processor—can yield linear speed-up when the number of processors is less than or equal to $\sqrt[3]{\frac{T_f}{T_c}n}$, and optimal speed-up when the number of processors equals $\sqrt{\frac{T_f}{T_c}n}$, where n is the population size.

Hybridization can be an extremely effective way of improving the performance and effectiveness of Genetic Algorithms. The most common form of hybridization is to couple GAs with local search techniques and to incorporate domain-specific knowledge into the search process. A common form of hybridization is to incorporate a local search operator into the Genetic Algorithm by applying the operator to each member of the population after each generation. This hybridization is often carried out in order to produce stronger results than the individual approaches can achieve on their own. However, this improvement in solution quality usually comes at the expense of increased computational time (e.g. Burke et al., 2001). Such approaches are often called Memetic Algorithms in the literature. This term was first used by Moscato (1989) and has since been employed very widely. For more details about memetic algorithms in general, see Krasnogor and Smith (2005), Krasnogor et al. (2004), Moscato and Cotta (2003) and Moscato (1999).

Of course, the hybridization of GAs can take other forms. Examples include:

- Initializing a GA population: e.g. Burke et al. (1998), Fleurent and Ferland (1994), Watson et al. (1999).
- Repairing infeasible solutions into legal ones: e.g. Ibaraki (1997).
- Developing specialized heuristic recombination operators: e.g. Burke et al. (1995).
- Incorporating a case-based memory (experience of past attempts) into the GA process (Louis and McDonnell, 2004).
- Heuristically decomposing large problems into smaller sub-problems before employing a memetic algorithm: e.g. Burke and Newall (1999).

Hybrid genetic algorithm and memetic approaches have demonstrated significant success in difficult real word application areas. A very small number of examples are included below (many more examples can be seen in the wider literature):

- University timetabling: examination timetabling (Burke et al., 1996, 1998; Burke and Newall, 1999) and course timetabling (Paechter et al., 1995, 1996).
- Machine scheduling (Cheng and Gen, 1997).
- Electrical power systems: unit commitment problems (Valenzuala and Smith, 2002); electricity transmission network maintenance scheduling (Burke and Smith, 1999); thermal generator maintenance scheduling (Burke and Smith, 2000).
- Sports scheduling (Costa, 1995).
- Nurse rostering (Burke et al., 2001).
- Warehouse scheduling (Watson et al., 1999).

While GA practitioners have often understood that real-world or commercial applications often require hybridization, there has been limited effort devoted to developing a theoretical underpinning of genetic algorithm hybridization. However, the following list contains examples of work which has aimed to answer critical issues such as

- the optimal division of labor between global and local searchers (or the right mix of exploration and exploitation) (Goldberg and Voessner, 1999);
- the effect of local search on sampling (Hart and Belew, 1996);
- hybrid GA modeling issues (Whitely, 1995).

The papers cited in this section are only a tiny proportion of the literature on hybrid genetic algorithms but they should provide a starting point for the interested reader. However, although there is a significant body of literature existing on the subject, there are many research directions still to be explored. Indeed, considering the option of hybridizing a GA with other approaches is one of the suggestions we give in the *Tricks of the Trade* section at the end of the chapter.

Time continuation, where the capabilities of both mutation and recombination are utilized to obtain a solution of as high quality as possible with a given limited computational resource (Goldberg, 1999b; Srivastava and Goldberg, 2001; Sastry and Goldberg, 2004a, 2004b). Time utilization (or continuation) exploits the tradeoff between the search for solutions with a large population and a single convergence epoch or using a small population with multiple convergence epochs.

Early theoretical investigations indicate that when the BBs are of equal (or nearly equal) salience and both recombination and mutation operators have the linkage information, then a small population with multiple convergence epochs is more efficient. However, if the fitness function is noisy or has overlapping building blocks, then a large population with a single convergence epoch is more efficient (Sastry and Goldberg, 2004a, 2004b). On the other hand, if the BBs of the problem are of non-uniform salience, which essentially means that they require serial processing, then a small population with multiple convergence epochs is more efficient (Goldberg, 1999b). Nevertheless, much work needs to be done to develop a principled design theory for efficiency enhancement via time continuation and to design competent continuation operators to reinitialize populations between epochs.

Evaluation relaxation, where an accurate, but computationally expensive fitness evaluation is replaced with a less accurate, but computationally inexpensive fitness estimate. The low-cost, less-accurate fitness estimate can either be (1) *exogenous*, as in the case of surrogate (or approximate) fitness functions (Jin, 2003), where external means can be used to develop the fitness estimate, or (2) *endogenous*, as in the case of *fitness inheritance* (Smith et al., 1995) where the fitness estimate is computed internally and is based on parental fitnesses.

Evaluation relaxation in GAs dates back to early, largely empirical work of Grefenstette and Fitzpatrick (1985) in image registration (Fitzpatrick et al., 1984) where significant speed-ups were obtained by reduced random sampling of the pixels of an image. Approximate evaluation has since been used extensively to solve complex optimization problems

across many applications, such as structural engineering (Barthelemy and Haftka, 1993) and warehouse scheduling at Coors Brewery (Watson et al., 1999).

While early evaluation relaxation studies were largely empirical in nature, design theories have since been developed to understand the effect of approximate surrogate functions on population sizing and convergence time and to optimize speed-ups in approximate fitness functions with known variance (Miller and Goldberg, 1996b) in, for example, simple functions of known variance or known bias (Sastry, 2001), and in fitness inheritance (Sastry et al., 2001, 2004; Pelikan and Sastry, 2004).

4.2 TRICKS OF THE TRADE

In this section we present some suggestions for the reader who is new to the area of genetic algorithms and wants to know how best to get started. Fortunately, the ideas behind genetic algorithms are intuitive and the basic algorithm is not complex. Here are some basic *tips*.

- Start by using an "off the shelf" genetic algorithm. It is pointless developing a complex GA, if your problem can be solved using a simple and standard implementation.
- There are many excellent software packages that allow you to implement a genetic algorithm very quickly. Many of the introductory texts are supplied with a GA implementation and GA-LIB is probably seen as the software of choice for many people (see below).
- Consider carefully your representation. In the early days, the majority of implementations used a *bit representation* which was easy to implement. Crossover and mutation were simple. However, many other representations are now used, some utilizing complex data structures. You should carry out some research to determine what is the best representation for your particular problem.
- A basic GA will allow you to implement the algorithm and the only thing you have to supply is an evaluation function. If you can achieve this, then this is the fastest way to get a prototype system up and running. However, you may want to include some problem specific data in your algorithm. For example, you may want to include your own crossover operators (in order to guide the search) or you may want to produce the initial population using a constructive heuristic (to give the GA a good starting point).
- In recent times, many researchers have hybridized GAs with other search methods (see Section 4.1.3). Perhaps the most common method is to in-

clude a local searcher after the crossover and mutation operators (sometimes known as a memetic algorithm). This local searcher might be something as simple as a hill climber, which acts on each chromosome to ensure it is at a local optimum before the evolutionary process starts again.

- There are many parameters required to run a genetic algorithm (which can be seen as one of the shortcomings). At a *minimum* you have the population size, the mutation probability, and the crossover probability. The problem with having so many parameters to set is that it can take a lot of experimentation to find a set of values which solves your particular problem to the required quality. A broad *rule of thumb*, to start with, is to use a mutation probability of 0.05 (De Jong, 1975), a crossover rate of 0.6 (De Jong, 1975) and a population size of about 50. These three parameters are just an example of the many choices you are going to have to make to get your GA implementation working. To provide just a small sample: which crossover operator should you use?... which mutation operator?... Should the crossover/mutation rates be dynamic and change as the run progresses? Should you use a local search operator? If so, which one, and how long should that be allowed to run for? What selection technique should you use? What replacement strategy should you use? Fortunately, many researchers have investigated many of these issues and the additional sources section below provides many suitable references.

SOURCES OF ADDITIONAL INFORMATION

Software

- GALib, http://lancet.mit.edu/ga/. If you want GA software then GALIB should probably be your first port of call. The description (from the web page) says

  > GAlib contains a set of C++ genetic algorithm objects. The library includes tools for using genetic algorithms to do optimization in any C++ program using any representation and genetic operators. The documentation includes an extensive overview of how to implement a genetic algorithm as well as examples illustrating customizations to the GAlib classes.

- GARAGe, http://garage.cps.msu.edu/. Genetic Algorithms Research and Applications Group.

- LGADOS in Coley (1999).

- NeuroDimension, http://www.nd.com/genetic/

- Simple GA (SGA) in Goldberg (1989b).
- Solver.com, http://www.solver.com/
- Ward Systems Group Inc., http://www.wardsystems.com/
- Other packages, http://www-2.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/genetic/ga/systems/0.html. This URL contains links to a number of genetic algorithm software libraries.

Introductory Material

There are many publications which give excellent introductions to genetic algorithms: see Holland (1975), Davis (1987), Goldberg (1989b), Davis (1991), Beasley et al. (1993), Forrest (1993), Reeves (1995), Michalewicz (1996), Mitchell (1996), Falkenauer (1998), Coley (1999), and Man et al. (1999).

Memetic Algorithms

There are some excellent introductory texts for memetic algorithms: see Radcliffe and Surry (1994), Moscato (1999, 2001), Moscato and Cotta (2003), Hart et al. (2004), Krasnogor et al. (2004), Krasnogor and Smith (2005).

You might also like to refer to the Memetic Algorithms Home Page at

- http://www.densis.fee.unicamp.br/∼moscato/memetic_home.html

Historical Material

An excellent work which brings together the early pioneering work in the field is Fogel (1998).

Conferences and Journals

There are a number of journals and conferences which publish papers concerned with genetic algorithms. The key conferences and journals are listed below, but remember that papers on Genetic Algorithms are published in many other outlets too.

Journals

- Evolutionary Computation, http://mitpress.mit.edu/catalog/item/default.asp?tid=25&ttype=4
- Genetic Programming and Evolvable Machines, http://www.kluweronline.com/issn/1389-2576/contents

- IEEE Transactions on Evolutionary Computation, http://www.ieee-nns.org/pubs/tec/

Conferences

- Congress on Evolutionary Computation (CEC)
- Genetic and Evolutionary Computation Conference (GECCO)
- Parallel Problem Solving in Nature (PPSN)
- Simulated Evolution and Learning (SEAL)

References

Asoh, H. and Mühlenbein, H., 1994, On the mean convergence time of evolutionary algorithms without selection and mutation, *Parallel Problem Solving from Nature III,* Lecture Notes in Computer Science, Vol. 866, pp. 98–107.

Bäck, T., 1995, Generalized convergence models for tournament—and (μ, λ)—selection, *Proc. 6th Int. Conf. on Genetic Algorithms,* pp. 2–8.

Bäck, T., Fogel, D. B. and Michalewicz, Z., 1997, Handbook of Evolutionary Computation, Oxford University Press, Oxford.

Baker, J. E., 1985, Adaptive selection methods for genetic algorithms, *Proc. Int. Conf. on Genetic Algorithms and Their Applications,* pp. 101–111.

Baluja, S., 1994, Population-based incremental learning: A method of integrating genetic search based function optimization and competitive learning, *Technical Report* CMU-CS-94-163, Carnegie Mellon University.

Barthelemy, J.-F. M. and Haftka, R. T., 1993, Approximation concepts for optimum structural design—a review, *Struct. Optim.* **5**:129–144.

Beasley, D., Bull, D. R. and Martin, R. R., 1993, An overview of genetic algorithms: Part 1, fundamentals, *Univ. Comput.* **15**:58–69.

Blickle, T. and Thiele, L., 1995, A mathematical analysis of tournament selection, *Proc. 6th Int. Conf. on Genetic Algorithms*, pp. 9–16.

Booker, L. B., Fogel, D. B., Whitley, D. and Angeline, P. J., 1997, Recombination, in: *The Handbook of Evolutionary Computation,* T. Bäck, D. B. Fogel, and Z. Michalewicz, eds, chapter E3.3, pp. C3.3:1–C3.3:27, IOP Publishing and Oxford University Press, Philadelphia, PA.

Bosman, P. A. N. and Thierens, D., 1999, Linkage information processing in distribution estimation algorithms, *Proc. 1999 Genetic and Evolutionary Computation Conf.*, pp. 60–67.

Bremermann, H. J., 1958, The evolution of intelligence. The nervous system as a model of its environment, *Technical Report* No. 1, Department of Mathematics, University of Washington, Seattle, WA.

Bulmer, M. G., 1985, *The Mathematical Theory of Quantitative Genetics,* Oxford University Press, Oxford.

Burke, E. K. and Newall, J. P., 1999, A multi-stage evolutionary algorithm for the timetable problem, *IEEE Trans. Evol. Comput.* **3**:63–74.

Burke, E. K. and Smith, A. J., 1999, A memetic algorithm to schedule planned maintenance, *ACM J. Exp. Algor.* **4**1, www.jea.acm.org/1999/BurkeMemetic/ ISSN 1084-6654.

Burke, E. K. and Smith, A. J., 2000, Hybrid Evolutionary Techniques for the Maintenance Scheduling Problem, *IEEE Trans. Power Syst.* **15**:122–128.

Burke, E. K., Elliman, D. G. and Weare, R.F., 1995, Specialised recombinative operators for timetabling problems, in: *Evolutionary Computing: AISB Workshop 1995* T. Fogarty, ed., Lecture Notes in Computer Science, Vol. 993, pp. 75–85, Springer, Berlin.

Burke, E. K., Newall, J. P. and Weare, R. F., 1996, A memetic algorithm for university exam timetabling, in: *The Practice and Theory of Automated Timetabling I*, E. K. Burke and P. Ross, eds, Lecture Notes in Computer Science, Vol. 1153, pp. 241–250, Springer, Berlin.

Burke, E. K., Newall, J. P. and Weare, R. F., 1998, Initialisation strategies and diversity in evolutionary timetabling, *Evol. Comput. J.* (special issue on Scheduling) **6**:81–103.

Burke, E. K., Cowling, P. I., De Causmaecker, P. and Vanden Berghe, G., 2001, A memetic approach to the nurse rostering problem, *Appl. Intell.* **15**:199–214.

Cantü-Paz, E., 1997, A summary of research on parallel genetic algorithms *IlliGAL Report* No. 97003, General Engineering Department, University of Illinois at Urbana-Champaign, Urbana, IL.

Cantú-Paz, E., 1999, Migration policies and takeover times in parallel genetic algorithms, in: *Proc. Genetic and Evolutionary Computation Conf.*, p. 775, Morgan Kaufmann, San Francisco.

Cantú-Paz, E., 2000, *Efficient and Accurate Parallel Genetic Algorithms,* Kluwer, Boston, MA.

Cheng, R. W. and Gen, M., 1997, Parallel machine scheduling problems using memetic algorithms, *Comput. Indust. Eng.*, **33**:761–764.

Coley, D. A., 1999, *An Introduction to Genetic Algorithms for Scientists and Engineers,* World Scientific, New York.

Costa, D., 1995, An evolutionary tabu search algorithm and the nhl scheduling problem, *INFOR* **33**:161–178.

Crow, J. F. and Kimura, M., 1970, *An Introduction of Population Genetics Theory,* Harper and Row, New York.

Davis, L., 1985, Applying algorithms to epistatic domains, in: *Proc. Int. Joint Conf. on Artifical Intelligence*, pp. 162–164.

Davis, L. D. (ed), 1987, Genetic Algorithms and Simulated Annealing, Pitman, London.

Davis, L. (ed), 1991, *Handbook of Genetic Algorithms,* Van Nostrand Reinhold, New York.

De Jong, K. A., 1975, An analysis of the behavior of a class of genetic adaptive systems, *Doctoral Dissertation,* University of Michigan, Ann Arbor, MI (University Microfilms No. 76-9381) (*Dissertation Abs. Int.* **36**:5140B).

Deb, K. and Goldberg, D. E., 1994, Sufficient conditions for deceptive and easy binary functions, *Ann. Math. Artif. Intell.* **10**:385–408.

Falkenauer E., 1998, Genetic Algorithms and Grouping Problems, Wiley, New York.

Fitzpatrick, J. M., Grefenstette, J. J. and Van Gucht, D., 1984, Image registration by genetic search, in: *Proc. IEEE Southeast Conf.,* IEEE, Piscataway, NJ, pp. 460–464.

Fleurent, C. and Ferland, J., 1994, Genetic hybrids for the quadratic assignment problem, in: *DIMACS Series in Mathematics and Theoretical Computer Science*, Vol. 16, pp. 190–206.

Fogel, D. B., 1998, *Evolutionary Computation: The Fossil Record*, IEEE, Piscataway, NJ.

Forrest, S., 1993, Genetic algorithms: Principles of natural selection applied to computation, *Science* **261**:872–878.

Fraser, A. S., 1957, Simulation of genetic systems by automatic digital computers. II: Effects of linkage on rates under selection, *Austral. J. Biol. Sci.* **10**:492–499.

Goldberg, D. E., 1983, Computer-aided pipeline operation using genetic algorithms and rule learning, *Doctoral Dissertation,*. University of Michigan, Ann Arbor, MI.

Goldberg, D. E., 1987, Simple genetic algorithms and the minimal deceptive problem, in: *Genetic Algorithms and Simulated Annealing,* L. Davis, ed., chapter 6, pp. 74–88, Morgan Kaufmann, Los Altos, CA.

Goldberg, D. E., 1989a, Genetic algorithms and Walsh functions: Part II, deception and its analysis, *Complex Syst.* **3**:153–171.

Goldberg, D. E., 1989b, *Genetic Algorithms in Search Optimization and Machine Learning,* Addison-Wesley, Reading, MA.

Goldberg, D. E., 1989c, Sizing populations for serial and parallel genetic algorithms, in: *Proc. 3rd Int. Conf. on Genetic Algorithms*, pp. 70–79.

Goldberg, D. E., 1999a, The race, the hurdle, and the sweet spot: Lessons from genetic algorithms for the automation of design innovation and creativity, in: *Evolutionary Design by Computers,* P. Bentley, ed., chapter 4, pp. 105–118, Morgan Kaufmann, San Mateo, CA.

Goldberg, D. E., 1999b, Using time efficiently: Genetic-evolutionary algorithms and the continuation problem, in: *Proc. Genetic and Evolutionary Computation Conf.*, pp. 212–219.

Goldberg, D. E., 2002, *Design of Innovation: Lessons From and For Competent Genetic Algorithms,* Kluwer, Boston, MA.

Goldberg, D. E. and Deb, K., 1991, A comparative analysis of selection schemes used in genetic algorithms, *Foundations of Genetic Algorithms*, G. J. E. Rawlins, ed., pp. 69–93.

Goldberg, D. E., Deb, K. and Clark, J. H., 1992a, Genetic algorithms, noise, and the sizing of populations, *Complex Syst.* **6**:333–362.

Goldberg, D. E., Deb, K. and Horn, J., 1992b, Massive multimodality, deception, and genetic algorithms, *Parallel Problem Solving from Nature II*, pp. 37–46, Elsevier, New York.

Goldberg, D. E., Deb, K., Kargupta, H. and Harik, G., 1993, Rapid, accurate optimization of difficult problems using fast messy genetic algorithms, in: *Proc. Int. Conf. on Genetic Algorithms,* pp. 56–64.

Goldberg, D. E., Korb, B. and Deb, K., 1989, Messy genetic algorithms: Motivation, analysis, and first results. *Complex Syst.* **3**:493–530.

Goldberg, D. E. and Lingle, R., 1985, Alleles, loci, and the TSP, in: *Proc. 1st Int. Conf. on Genetic Algorithms,* pp. 154–159.

Goldberg, D. E. and Rudnick, M., 1991, Genetic algorithms and the variance of fitness, *Complex Syst.* **5**:265–278.

Goldberg, D. E. and Sastry, K., 2001, A practical schema theorem for genetic algorithm design and tuning, in: *Proc. of the Genetic and Evolutionary Computation Conf.,* pp. 328–335.

Goldberg, D. E., Sastry, K. and Latoza, T., 2001, On the supply of building blocks, in: *Proc. of the Genetic and Evolutionary Computation Conf.*, pp. 336–342.

Goldberg, D. E. and Segrest, P., 1987, Finite Markov chain analysis of genetic algorithms, in: *Proc. 2nd Int. Conf. on Genetic Algorithms*, pp. 1–8.

Goldberg, D. E. and Voessner, S., 1999, Optimizing global-local search hybrids, in: *Proc. of the Genetic and Evolutionary Computation Conf.,* pp. 220–228.

Gorges-Schleuter, M., 1989, ASPARAGOS: An asynchronous parallel genetic optimization strategy, in: *Proc. 3rd Int. Conf. on Genetic Algorithms,* pp. 422–428.

Gorges-Schleuter, M., 1997, ASPARAGOS96 and the traveling salesman problem, in: *Proc. IEEE Int. Conf. on Evolutionary Computation,* pp. 171–174.

Grefenstette, J. J., 1981, Parallel adaptive algorithms for function optimization, *Technical Report* No. CS-81-19, Computer Science Department, Vanderbilt University, Nashville, TN.

Grefenstette, J. J. and Baker, J. E., 1989, How genetic algorithms work: A critical look at implicit parallelism, in: *Proc. 3rd Int. Conf. on Genetic Algorithms,* pp. 20–27.

Grefenstette, J. J. and Fitzpatrick, J. M., 1985, Genetic search with approximate function evaluations, in: *Proc. Int. Conf. on Genetic Algorithms and Their Applications,* pp. 112–120.

Harik, G. R., 1997, Learning linkage to eficiently solve problems of bounded difficulty using genetic algorithms, *Doctoral Dissertation,* University of Michigan, Ann Arbor, MI.

Harik, G., 1999, Linkage learning via probabilistic modeling in the ECGA, *IlliGAL Report* No. 99010, University of Illinois at Urbana-Champaign, Urbana, IL.

Harik, G., Cantú-Paz, E., Goldberg, D. E. and Miller, B. L., 1999, The gambler's ruin problem, genetic algorithms, and the sizing of populations, *Evol. Comput.* **7**:231–253.

Harik, G. and Goldberg, D. E., 1997, Learning linkage, *Foundations of Genetic Algorithms,* **4**:247–262.

Harik, G., Lobo, F. and Goldberg, D. E., 1998, The compact genetic algorithm, in: *Proc. IEEE Int. Conf. on Evolutionary Computation,* pp. 523–528.

Hart, W. E. and Belew, R. K., 1996, Optimization with genetic algorithm hybrids using local search, in: *Adaptive Individuals in Evolving Populations,* R. K. Belew, and M. Mitchell, eds, pp. 483–494, Addison-Wesley, Reading, MA.

Hart, W., Krasnogor, N. and Smith, J. E. (eds), 2004, Special issue on memetic algorithms, *Evol. Comput.* **12** No. 3.

Heckendorn, R. B. and Wright, A. H., 2004, Efficient linkage discovery by limited probing, *Evol. Comput.* **12**:517–545.

Holland, J. H., 1975, *Adaptation in Natural and Artificial Systems,* University of Michigan Press, Ann Arbor, MI.

Ibaraki, T., 1997, Combinations with other optimization methods, in: *Handbook of Evolutionary Computation,* T. Bäck, D. B. Fogel, and Z. Michalewicz, eds, pp. D3:1–D3:2, Institute of Physics Publishing and Oxford University Press, Bristol and New York.

Jin, Y., 2003, A comprehensive survey of fitness approximation in evolutionary computation, *Soft Comput. J.* (in press).

Kargupta, H., 1996, The gene expression messy genetic algorithm, in: *Proc. Int. Conf. on Evolutionary Computation,* pp. 814–819.

Krasnogor, N., Hart, W. and Smith, J. (eds), 2004, *Recent Advances in Memetic Algorithms*, Studies in Fuzziness and Soft Computing, Vol. 166, Springer, Berlin.

Krasnogor, N. and Smith, J. E., 2005, A tutorial for competent memetic algorithms: model, taxonomy and design issues, *IEEE Trans. Evol. Comput.*, accepted for publication.

Louis, S. J. and McDonnell, J., 2004, Learning with case injected genetic algorithms, *IEEE Trans. Evol. Comput.* **8**:316–328.

Larrañaga, P. and Lozano, J. A. (eds), 2002, *Estimation of Distribution Algorithms,* Kluwer, Boston, MA.

Lin, S.-C., Goodman, E. D. and Punch, W. F., 1997, Investigating parallel genetic algorithms on job shop scheduling problem, *6th Int. Conf. on Evolutionary Programming,* pp. 383–393.

Man, K. F., Tang, K. S. and Kwong, S., 1999, Genetic Algorithms: Concepts and Design, Springer, London.

Manderick, B. and Spiessens, P., 1989, Fine-grained parallel genetic algorithms, in: *Proc. 3rd Int. Conf. on Genetic Algorithms,* pp. 428–433.

Memetic Algorithms Home Page:
http://www.densis.fee.unicamp.br/∼moscato/memetic_home.html

Michalewicz, Z., 1996, *Genetic Algorithms + Data Structures = Evolution Programs,* 3rd edn, Springer, Berlin.

Miller, B. L. and Goldberg, D. E., 1995, Genetic algorithms, tournament selection, and the effects of noise, *Complex Syst.* **9**:193–212.

Miller, B. L. and Goldberg, D. E., 1996a, Genetic algorithms, selection schemes, and the varying effects of noise, *Evol. Comput.* **4**:113–131.

Miller, B. L. and Goldberg, D. E., 1996b, Optimal sampling for genetic algorithms, *Intelligent Engineering Systems through Artificial Neural Networks (ANNIE'96)*, Vol. 6, pp. 291–297, ASME Press, New York.

Mitchell, M., 1996, *Introduction to Genetic Algorithms,* MIT Press, Boston, MA.

Moscato, P., 1989, On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms, *Technical Report* C3P 826, Caltech Concurrent Computation Program, California Institute of Technology, Pasadena, CA.

Moscato, P., 1999, Part 4: Memetic algorithms, in: *New Ideas in Optimization*, D. Corne, M. Dorigo and F. Glover, eds, pp. 217–294, McGraw-Hill, New York.

Moscato, P., 2001, Memetic algorithms, in: *Handbook of Applied Optimization*, Section 3.6.4, P. M. Pardalos and M. G. C. Resende, eds, Oxford University Press, Oxford.

Moscato, P. and Cotta, C., 2003, A gentle introduction to memetic algorithms, in: *Handbook of Metaheuristics*, F. Glover and G. Kochenberger, eds, Chapter 5, Kluwer, Norwell, MA.

Mühlenbein, H. and Paaß, G., 1996, From recombination of genes to the estimation of distributions I. Binary parameters, in: *Parallel Problem Solving from Nature IV*, Lecture Notes in Computer Science, Vol. 1141, Springer, Berlin.

Mühlenbein, H. and Schlierkamp-Voosen, D., 1993, Predictive models for the breeder genetic algorithm: I. continous parameter optimization, *Evol. Comput.* **1**:25–49.

Munetomo, M. and Goldberg, D. E., 1999, Linkage identification by non-monotonicity detection for overlapping functions, *Evol. Comput.* **7**:377–398.

Oliver, J. M., Smith, D. J. and Holland, J. R. C., 1987, A study of permutation crossover operators on the travelling salesman problem, in: *Proc. 2nd Int. Conf. on Genetic Algorithms*, pp. 224–230.

Paechter, B., Cumming, A., Norman, M. G. and Luchian, H., 1996, Extensions to a memetic timetabling system, *The Practice and Theory of Automated Timetabling I*, E. K. Burke and P. Ross, eds, Lecture Notes in Computer Science, Vol. 1153, Springer, Berlin, pp. 251–265.

Paechter, B., Cumming, A. and Luchian, H., 1995, The use of local search suggestion lists for improving the solution of timetable problems with evolutionary algorithms, *Evolutionary Computing: AISB Workshop 1995*, T. Fogarty, ed., Lecture Notes in Computer Science, Vol. 993, Springer, Berlin, pp. 86–93.

Pelikan, M., 2005, *Hierarchical Bayesian Optimization Algorithm: Toward a New Generation of Evolutionary Algorithm,* Springer, Berlin.

Pelikan, M. and Goldberg, D. E., 2001, Escaping hierarchical traps with competent genetic algorithms, in: *Proc. Genetic and Evolutionary Computation Conf.,* pp. 511–518.

Pelikan, M., Goldberg, D. E. and Cantú-Paz, E., 2000, Linkage learning, estimation distribution, and Bayesian networks, *Evol. Comput.* **8**:314–341.

Pelikan, M., Lobo, F. and Goldberg, D. E., 2002, A survey of optimization by building and using probabilistic models, *Comput. Optim. Appl.* **21**:5–20.

Pelikan, M. and Sastry, K., 2004, Fitness inheritance in the Bayesian optimization algorithm, in: *Proc. Genetic and Evolutionary Computation Conference,* Vol. 2, pp. 48–59.

Pettey, C. C., Leuze, M. R. and Grefenstette, J. J., 1987, A parallel genetic algorithm, in: *Proc. 2nd Int. Conf. on Genetic Algorithms,* pp. 155–161.

Radcliffe, N. J. and Surry, P. D., 1994, Formal memetic algorithms, *Evolutionary Computing: AISB Workshop 1994*, T. Fogarty, ed., Lecture Notes in Computer Science, Vol. 865, pp. 1–16, Springer, Berlin.

Reeves, C. R., 1995, Genetic algorithms, in: *Modern Heuristic Techniques for Combinatorial Problems,* C. R. Reeves, ed., McGraw-Hill, New York.

Robertson, G. G., 1987, Parallel implementation of genetic algorithms in a classifier system, in: *Proc. 2nd Int. Conf. on Genetic Algorithms,* pp. 140–147.

Rothlauf, F., 2002, *Representations for Genetic and Evolutionary Algorithms,* Springer, Berlin.

Rudolph, G., 2000, Takeover times and probabilities of non-generational selection rules, in: *Proc. Genetic and Evolutionary Computation Conf.,* pp. 903–910.

Sakamoto, Y. and Goldberg, D. E., 1997, Takeover time in a noisy environment, in: *Proc. 7th Int. Conf. on Genetic Algorithms,* pp. 160–165.

Sastry, K., 2001, Evaluation-relaxation schemes for genetic and evolutionary algorithms, *Master's Thesis,* General Engineering Department, University of Illinois at Urbana-Champaign, Urbana, IL.

Sastry, K. and Goldberg, D. E., 2002, Analysis of mixing in genetic algorithms: A survey, *IlliGAL Report* No. 2002012, University of Illinois at Urbana-Champaign, Urbana, IL.

Sastry, K. and Goldberg, D. E., 2003, Scalability of selectorecombinative genetic algorithms for problems with tight linkage, in: *Proc. 2003 Genetic and Evolutionary Computation Conf.,* pp. 1332–1344.

Sastry, K. and Goldberg, D. E., 2004a, Designing competent mutation operators via probabilistic model building of neighborhoods, in: *Proc. 2004 Genetic and Evolutionary Computation Conference II,* Lecture Notes in Computer Science, Vol. 3103, Springer, Berlin, pp. 114–125.

Sastry, K. and Goldberg, D. E., 2004b, Let's get ready to rumble: Crossover versus mutation head to head, in: *Proc. 2004 Genetic and Evolutionary Computation Conf. II,* Lecture Notes in Computer Science, Vol. 3103, Springer, Berlin, pp. 126–137.

Sastry, K., Goldberg, D. E., & Pelikan, M., 2001, Don't evaluate, inherit, in: *Proc. Genetic and Evolutionary Computation Conf.,* pp. 551–558.

Sastry, K., Pelikan, M. and Goldberg, D. E., 2004, Efficiency enhancement of genetic algorithms building-block-wise fitness estimation, in: *Proc. IEEE Int. Congress on Evolutionary Computation,* pp. 720–727.

Smith, R., Dike, B. and Stegmann, S., 1995, Fitness inheritance in genetic algorithms, in: *Proc. ACM Symp. on Applied Computing,* pp. 345–350, ACM, New York.

Spears, W., 1997, Recombination parameters, in: *The Handbook of Evolutionary Computation,* T. Bäck, D. B. Fogel and Z. Michalewicz, eds, Chapter E1.3, IOP Publishing and Oxford University Press, Philadelphia, PA, pp. E1.3:1–E1.3:13.

Spears, W. M. and De Jong, K. A., 1994, On the virtues of parameterized uniform crossover, in: *Proc. 4th Int. Conf. on Genetic Algorithms*.

Srivastava, R. and Goldberg, D. E., 2001, Verification of the theory of genetic and evolutionary continuation, in: *Proc. Genetic and Evolutionary Computation Conf.*, pp. 551–558.

Syswerda, G., 1989, Uniform crossover in genetic algorithms, in: *Proc. 3rd Int. Conf. on Genetic Algorithms,* pp. 2–9.

Thierens, D., 1999, Scalability problems of simple genetic algorithms, *Evol. Comput.* **7**:331–352.

Thierens, D. and Goldberg, D. E., 1994a, Convergence models of genetic algorithm selection schemes, in: *Parallel Problem Solving from Nature III,* pp. 116–121.

Thierens, D. and Goldberg, D. E., 1994b, Elitist recombination: An integrated selection recombination GA, in: *Proc. 1st IEEE Conf. on Evolutionary Computation,* pp. 508–512.

Thierens, D., Goldberg, D. E. and Pereira, A. G., 1998, Domino convergence, drift, and the temporal-salience structure of problems, in: *Proc. IEEE Int. Conf. on Evolutionary Computation,* pp. 535–540.

Valenzuala, J. and Smith, A. E., 2002, A seeded memetic algorithm for large unit commitment problems, *J. Heuristics*, **8**:173–196.

Voigt, H.-M., Mühlenbein, H. and Schlierkamp-Voosen, D., 1996, The response to selection equation for skew fitness distributions, in: *Proc. Int. Conf. on Evolutionary Computation,* pp. 820–825.

Watson, J. P., Rana, S., Whitely, L. D. and Howe, A. E., 1999, The impact of approximate evaluation on the performance of search algorithms for warehouse scheduling, *J. Scheduling*, **2**:79–98.

Whitley, D., 1995, Modeling hybrid genetic algorithms, in *Genetic Algorithms in Engineering and Computer Science*, G. Winter, J. Periaux, M. Galan and P. Cuesta, eds, Wiley, New York, pp. 191–201.

Yu, T.-L., Goldberg, D. E., Yassine, A. and Chen, Y.-P., 2003, A genetic algorithm design inspired by organizational theory: Pilot study of a dependency structure matrix driven genetic algorithm, *Artificial Neural Networks in Engineering (ANNIE 2003),* pp. 327–332.