Chapter 4
Genetic Algorithms

Kumara Sastry, David E. Goldberg, and Graham Kendall

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 of simple GAs and the associated terminologies. 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 (TSP), 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.

K. Sastry
University of Illinois, Urbana-Champaign, IL, USA
Current Affiliation: Intel Corp.

D.E. Goldberg
ThreeJoy Associates, Inc. and University of Illinois, Urbana-Champaign, IL, USA

G. Kendall (✉)
Automated Scheduling, Optimization and Planning Research Group, School of Computer Science, University of Nottingham, Nottingham, UK

Automated Scheduling, Optimization and Planning Research Group, School of Computer Science, University of Nottingham, Semenyih, Malaysia
e-mail: Graham.Kendall@nottingham.edu.my

E.K. Burke and G. Kendall (eds.), *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, DOI 10.1007/978-1-4614-6940-7_4,

Another important concept of GAs is the notion of population. Unlike traditional search methods, GAs 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 GAs. 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 in the generation of the initial population.
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 to solutions with better 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 bits and pieces 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 achieving competent performance depends on getting the recombination mechanism designed properly; but the primary idea to keep in mind is that 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 that are 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 one or more stopping criteria are met.

Goldberg (1983, 1999a, 2002) has likened GAs to mechanistic versions of certain modes of human innovation and has shown that, although these operators when analyzed individually are ineffective, when combined together they can work well. This aspect has been explained with the concepts of *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 selec-

tion and recombination to *innovation* (cross-fertilizing). These analogies have been used to develop a design decomposition methodology and so-called *competent* genetic algorithms — GAs that solve hard problems quickly, reliably, and accurately — both of which are discussed in subsequent sections.

This chapter is organized as follows. The next section provides details of the individual steps of a typical GA and introduces several popular genetic operators. Section 4.3 presents a principled methodology of designing competent GAs based on decomposition principles. Section 4.4 gives a brief overview of designing principled efficiency enhancement techniques to speed up genetic and evolutionary algorithms.

4.2 Basic GA Operators

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

4.2.1 Selection Methods

Selection procedures can be broadly classified into two classes (Goldberg 1989): fitness proportionate selection and ordinal selection.

4.2.1.1 Fitness Proportionate Selection

This includes methods such as roulette-wheel selection and stochastic universal selection. 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 larger slot sizes 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 an 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.

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 $q_2 = 0.484 < 0.585 \leq q_3 = 0.631$.

4.2.1.2 Ordinal Selection

Ordinal selection includes methods such as tournament selection (Goldberg et al. 1989) 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 s 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.

4.2.2 Recombination Operators

After selection, individuals from the mating pool are recombined (or crossed over) to create new, hopefully better offsprings. In the GA literature, many crossover methods have been designed (Goldberg 1989; Booker et al. 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 is briefly discussed in Sect. 4.3.

In most recombination operators, two individuals are randomly selected and are recombined with a probability p_c, called 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 1989, 2002).

k-point crossover: One-point and two-point crossovers are among the simplest and most widely applied crossover methods. In one-point crossover, illustrated

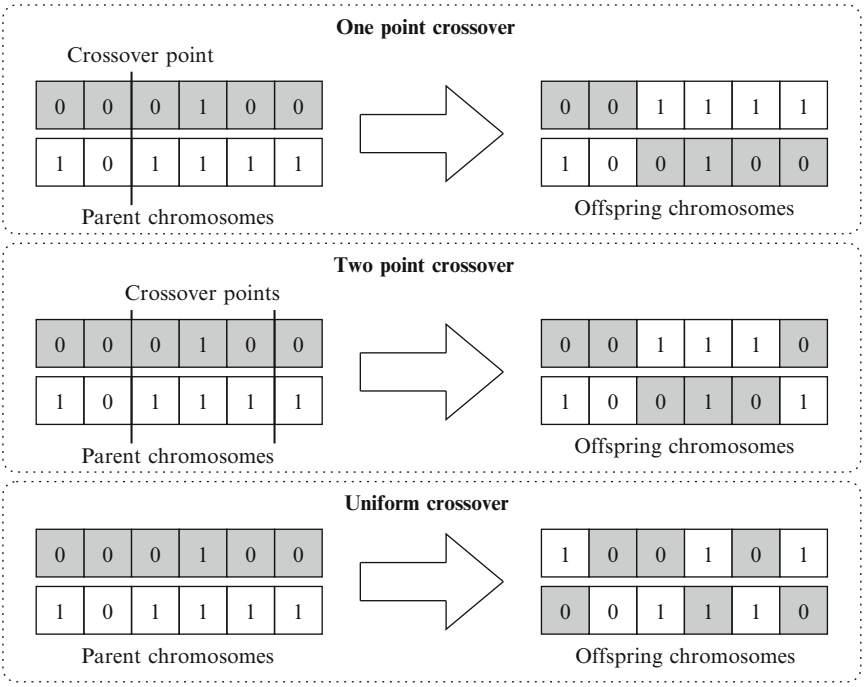

Fig. 4.1 Illustration of one-point, two-point, and uniform crossover methods

in Fig. 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 illustrated in Fig. 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 Fig. 4.1, every allele is exchanged between 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 those used in the TSP. They often create offsprings that represent invalid solutions for the search problem. Therefore, 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.

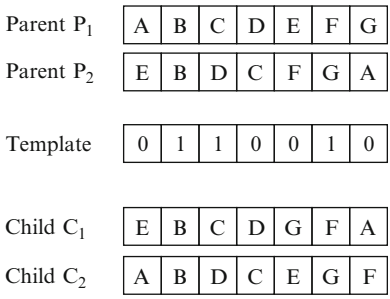

Fig. 4.2 Illustration of uniform-order crossover

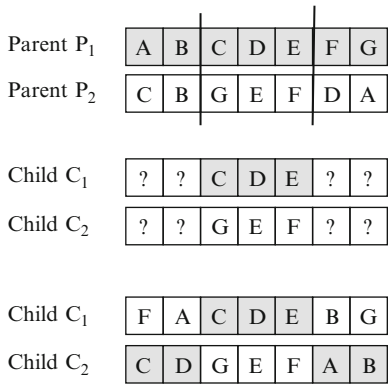

Fig. 4.3 Illustration of order-based crossover

In uniform order-based crossover, two parents (say P_1 and P_2) are randomly selected and a random binary template is generated (see Fig. 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 using a similar process (see Fig. 4.2).

Order-based crossover: The order-based crossover (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 Fig. 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 Fig. 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.

Partially matched crossover (PMX): As well as always generating valid offspring, the PMX operator (Goldberg and Lingle 1984) 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 Fig. 4.4 (reproduced with permission from Goldberg 1989), 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.

Cycle crossover (CX): We describe cycle crossover (Oliver et al. 1987) with the help of a simple illustration (reproduced with permission from Goldberg 1989). Consider two randomly selected parents P_1 and P_2 as shown in Fig. 4.5 that are solutions to a TSP 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 Fig. 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 9th 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 are filled from P_2. Offspring C_2 is created in the same way by starting with the first city of parent P_2 (see Fig. 4.5).

4.2.3 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) have the same allele at a given gene then one-point crossover will not

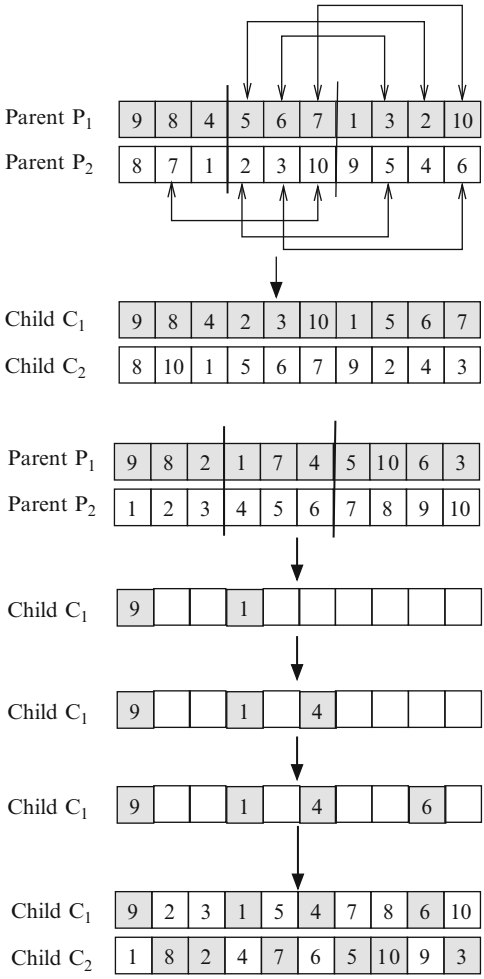

Fig. 4.4 Illustration of partially matched crossover

Fig. 4.5 Illustration of cycle crossover

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 evolution strategies, mutation is the primary variation/search operator. For an introduction to evolution strategies see, for example, Bäck (1996). Unlike evolution strategies, mutation is often the secondary operator performed with a low probability in GAs. One of the most common mutations is the bit-flip mutation. In bit-flip 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 those that swap genes and problem-specific ones can also be developed and are often used in the literature.

4.2.4 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 their fitness. Some of the most common 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 those listed below.

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.

4.3 Competent GAs

While using innovation for explaining 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 GAs. The last few decades have witnessed great strides, not only 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. Using Holland's notion of a building block (Holland 1975), the first author proposed decomposing the problem of designing a competent selectorecombinative GA (Goldberg et al. 1992). This design decomposition has been explained in detail elsewhere (Goldberg 2002), but is briefly reviewed in what follows.

Know that GAs process building blocks (BBs): The primary idea of selectorecombinative GA theory is that GAs work through a mechanism of *decomposition* and *reassembly*. Holland (1975) identified well-adapted sets of features that were components of effective solutions (BBs). The basic idea is that GAs (1) implicitly identify BBs 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, 2002).

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 dominant market share of the population, and (2) the growth rate is neither too fast nor too slow.

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

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

where ε is the probability of BB disruption.

Two other approaches have been used in understanding time. It is not appropriate in a basic text 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; Bäck 1994; Sakamoto and Goldberg 1997; Cantú-Paz 1999; Rudolph 2000).
- Selection-intensity models, where the dynamics of the average fitness of the population is modeled (Mühlenbein and Schlierkamp-Voosen 1993; Bäck 1995; Thierens and Goldberg 1994; Miller and Goldberg 1995, 1996a).

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

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

where I is the selection intensity (Bulmer 1985), which is a parameter dependent on the selection method and selection pressure.

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

$$t_c = \frac{-\log 2}{\log\left(1 - I/\sqrt{3}\right)}\ell. \tag{4.3}$$

To summarize, the convergence time of GAs scale up as $\mathrm{O}\left(\sqrt{\ell}\right) - \mathrm{O}(\ell)$ (see Chap. 1 for an explanation of the O notation).

Understand BB supply and decision making: One role of the population is to ensure an 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 et al. 2001). For a problem with m building blocks, each consisting of k alphabets of cardinality χ, the population size required to ensure the presence of at least one copy of all the raw building blocks is given by Goldberg et al. (2001)

$$n = \chi^k \log m + k\chi^k \log \chi. \tag{4.4}$$

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 (Goldberg et al. 1992; 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)

$$n = -\frac{\sqrt{\pi}}{2}\frac{\sigma_{BB}}{d}2^k\sqrt{m}\log\alpha, \tag{4.5}$$

where d/σ_{BB} is the signal-to-noise ratio (Goldberg et al. 1992), 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 BBs, 2^k
- *Subcomponent complexity*, quantified by the number of BBs, 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 BBs are exponentially scaled, the population size scales as (Thierens et al. 1998; Goldberg 2002)

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

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

To summarize, the population size required by GAs scales up as $\mathrm{O}\left(2^k\sqrt{m}\right) - \mathrm{O}\left(2^k m\right)$.

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 a *facetwise* modeling approach (Goldberg et al. 1993b; Thierens 1999; Sastry and Goldberg 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 into 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 principled design of effective BB identification and exchange mechanisms have led to the development of competent GAs. Competent GAs are a class of GAs that 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 a GA 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. 1993a). Since those early scalable results, a number of competent GAs have been constructed using different mechanism styles. We categorize these approaches and provide some references for the interested reader, but a detailed treatment is beyond the scope of this chapter:

- *Perturbation techniques* such as the messy GA (mGA) (Goldberg et al. 1989), fast messy GA (fmGA) (Goldberg et al. 1993a), gene expression messy GA (GEMGA) (Kargupta 1996), linkage identification by nonlinearity check/linkage identification by detection GA (LINC/LIMD GA) (Munetomo and Goldberg 1999; Heckendorn and Wright 2004), and dependency structure matrix driven genetic algorithm (DSMGA) (Yu 2006).
- *Linkage adaptation techniques* such as linkage learning GA (LLGA) (Chen 2004; Harik and Goldberg 1997).
- *Probabilistic model building techniques* (Pelikan et al. 2006) such as population-based incremental learning (PBIL) (Baluja 1994), the univariate model building algorithm (UMDA) (Mühlenbein and Paass 1996), the compact GA (CGA) (Harik et al. 1998), extended compact GA (eCGA) (Harik 1999), the Bayesian

optimization algorithm (BOA) (Pelikan et al. 2000), the iterated distribution estimation algorithm (IDEA) (Bosman and Thierens 1999) and the hierarchical Bayesian optimization algorithm (hBOA) (Pelikan 2005).

4.4 Efficiency Enhancement of Genetic Algorithms

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 $O(\ell^2)$ 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 1 hour. It requires about 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 effective for a particular problem. The literature contains a large number of papers which discuss enhancements of GAs. Once again, a detailed discussion is well beyond the scope of the chapter, 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 2000). Evolutionary algorithms are by *nature* parallel, and many different parallelization approaches such as a simple master–slave, coarse-grained fine-grained or hierarchical architectures can be readily used. Regardless of how parallelization is done, 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 an 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 of GAs. 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 GA 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 widely. For more details about memetic algorithms, 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 (Burke et al. 1998; Fleurent and Ferland 1993; Watson et al. 1999);
- Repairing infeasible solutions into legal ones (Ibaraki 1997);
- Developing specialized heuristic recombination operators (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 (Burke and Newell 1999).

Hybrid GA and memetic approaches have demonstrated significant success in difficult real-world application areas. A small number of examples are included below (many more examples can be found in the literature):

- University timetabling: examination timetabling (Burke et al. 1996, 1998; Burke and Newell 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 searches (or the right mix of exploration and exploitation) (Goldberg and Voessner 1999; Sinha 2003);
- The effect of local search on sampling (Hart and Belew 1996);
- Hybrid GA modeling issues (Whitley 1995).

The papers cited in this section are only a tiny proportion of the literature on hybrid GAs but they should provide a starting point for the interested reader. However, although there is a significant body of literature 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 capabilities of both mutation and recombination are optimally utilized to obtain a solution of as high quality as possible with a given limited computational resource (Goldberg 1999b; Sastry and Goldberg 2004). Time utilization (or continuation) exploits the tradeoff between the search for solutions with large populations 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 BBs, then a large population with single convergence epoch is more efficient (Sastry and Goldberg 2004). On the other hand, if the BBs of the problem are of non-uniform salience, which essentially require serial processing, then a small population with multiple convergence epochs is more efficient (Goldberg 1999b). 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 population 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 2005), 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 based on parental fitnesses.

Evaluation relaxation in GAs dates back to the 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 models have since been used extensively to solve complex optimization problems in many engineering applications such as aerospace and structural engineering (Barthelemy and Haftka 1993; Dennis and Torczon 1997).

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 integrated fitness functions (Albert 2001), in simple functions of known variance or known bias (Sastry 2001) and also in fitness inheritance (Sastry et al. 2001, 2004; Pelikan and Sastry 2004).

Speed-up obtained by employing an efficiency-enhancement technique (EET) is measured in terms of a ratio of the computation effort required by a GA when the EET is used to that required by GA in the absence of the EET. That is, $\eta = T_{\text{base}}/T_{\text{efficiency-enhanced}}$. Speed-up obtained by employing even a single EET can potentially be significant. Furthermore, assuming that the performance of one of the above methods does not affect the performance of others, if we employ more than one EET, the overall speed-up is the product of individual speed-ups. That is, if the speed-ups obtained by employing parallelization, hybridization, time continuation and evaluation relaxation be η_p, η_h, η_t, and η_e respectively. If one uses all these EETs, then the overall speed-up obtained is

$$\eta_{\text{total}} = \eta_p \eta_h \eta_t \eta_e.$$

Even if the speed-up obtained by a single EET is modest, a combination of two or more EETs can yield a significant speed-up. For example, if we use a parallel GA that yields linear speed-up with 10 processors, and each of the other three EETs makes GAs 25% more efficient, then together they yield a speed-up of $10 * 1.25^3 = 19.5$. That is evaluation relaxation, time continuation and hybridization would give slightly more than 9.5 processors' worth of additional computation power. GAs designed using the decomposition principles and principled efficiency enhancement outlined in this chapter have opened doors for efficiently solving routine billion-variable optimization in the increasingly large, complex problems of science (Sastry et al. 2007).

Tricks of the Trade

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

- Start by using an "off the shelf" GA. 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 GA 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 Sect. 4.4). Perhaps the most common method is to include 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 GA (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. An alternative method is to set these parameters based on facetwise models (see Sect. 4.3). 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 following section *Additional Sources* provides many suitable references.

Sources of Additional Information

At the time of writing, a Google search for Genetic Algorithms returned over seven million hits, so it is impossible to give a comprehensive list of all potentially useful sources. Below, we have suggested a small selection of sources that you might want to look at. We have split them into three areas, web sites, books and journal articles. We apologise for any that we have missed but it is impossible to list every available source.

Web Sites

Due to the nature of the internet, these URLs may not always be available and may move. All those listed were last accessed on 11 November 2012. You should also be

wary of citing URLs as they often lack a peer review mechanism and the page could be changed or disappear altogether.

- http://en.wikipedia.org/wiki/Genetic_algorithm/
- http://geneticalgorithms.ai-depot.com/
- http://illigal.org/
- http://lancet.mit.edu/ga/
- http://lancet.mit.edu/~mbwall/presentations/IntroToGAs/
- http://mathworld.wolfram.com/GeneticAlgorithm.html
- http://www.obitko.com/tutorials/genetic-algorithms/
- http://www.youtube.com/watch?v=ejxfTy4lI6I/

Books

- Coley, D. A. 1999. *An Introduction to Genetic Algorithms for Scientists and Engineers*. World Scientific.
- Davis, L. D. (ed) 1987. *Genetic Algorithms and Simulated Annealing*. Pitman.
- Davis, L. D. (ed) 1991. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold.
- Dawkins, R. 1976. *The Selfish Gene*. Oxford University Press.
- De Jong, K. 2002. *Evolutionary Computation: A Unified Approach*. MIT Press.
- Eiben, A. E., Smith, J. E. 2010. *Introduction to Evolutionary Computing*. Natural Computing Series, Springer.
- Falkenauer, E. 1998. *Genetic Algorithms and Grouping Problems*. Wiley.
- Fogel, D. B. 1998. *Evolutionary Computation The Fossil Record*. IEEE Press.
- Fogel, D. B. 2000. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. 2nd edn, IEEE Press.
- Goldberg, D. E. 1989. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley.
- Haupt, R. L., Haupt, S. E. 2004. *Practical Genetic Algorithms*. Wiley-Interscience.
- Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press (a second edition was published in 1992).
- Michalewicz, M. 1996. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer.
- Mitchell, M. 1998. *An Introduction to Genetic Algorithms (Complex Adaptive Systems)*. MIT Press.
- Munoz, A. R., Rodriguez, I. G. 2012. *Handbook of Genetic Algorithms: New Research*. Mathematics Research Developments. Nova Science.
- Reeves, C. R., Rowe, J. E. 2002. *Genetic Algorithms—Principles and Perspectives: A Guide to GA Theory*. Operations Research/Computer Science Interfaces Series. Springer.
- Sivanandam, S. N. 2008. *Introduction to Genetic Algorithms*. Springer.
- Vose, M. 1999. *The Simple Genetic Algorithm: Foundations and Theory (Complex Adaptive Systems)*. MIT Press.

Journal Articles

In this section, as well as providing some general references to the evolutionary computation literature, we have also provided a few suggestions for review articles in specific applications as these might be useful introductions which you may not otherwise come across.

- Carter, J. N. 2003. Chapter 3, Introduction to using genetic algorithms. *Developments in Petroleum Science*, 51, 51–76.
- Chatterjee, S., Laudato, M., Lynch, L. A. 1996. Genetic algorithms and their statistical applications: an introduction. *Computational Statistics and Data Analysis*, 22, 633–651.
- Cheng, R., Gen, M., Tsujimura, Y. 1999. A tutorial survey of job-shop scheduling problems using genetic algorithms. Part II: Hybrid genetic search strategies. *Computers and Industrial Engineering*, 36, 343–364.
- Forrest, S. 1993. Genetic algorithms: principles of natural selection applied to computation. *Science*, 261, 872–878.
- Larrañaga, P., Kuijpers, C. M. H., Murga, R. H., Inza, I., Dizdarevic, S. 1999. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, 13, 129–170.
- Salomon, R. 1996. Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions. A survey of some theoretical and practical aspects of genetic algorithms. *Biosystems*, 39, 263–278.
- Srinivas, M., Patnaik, L. M. 1994. Genetic algorithms: a survey. *Computer*, 27, 17–26.
- Zang, H., Zhang, S., Hapeshi, K. 2010. A review of nature-inspired algorithms. *Journal of Bionic Engineering*, 7, S232–S237.

References

Albert LA (2001) Efficient genetic algorithms using discretization scheduling. Master's thesis, University of Illinois at Urbana-Champaign (also IlliGAL report no 2002005)

Asoh H, Mühlenbein H (1994) On the mean convergence time of evolutionary algorithms without selection and mutation. PPSN 3, pp 98–107

Bäck T (1994) Selective pressure in evolutionary algorithms: a characterization of selection mechanisms. In: Proceedings of the 1st IEEE conference on evolutionary computation, Orlando, pp 57–62

Bäck T (1995) Generalized convergence models for tournament—and (μ, λ)—selection. In: Proceedings of 6th international conference on genetic algorithms, Pittsburgh, pp 2–8

Bäck T (1996) Evolutionary algorithms in theory and practice. Oxford University Press, New York

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-FM, Haftka RT (1993) Approximation concepts for optimum structural design—a review. Struct Optim 5:129–144

Booker LB, Fogel DB, Whitley D, Angeline PJ (1997) Recombination. In: Bäck T, Fogel DB, Michalewicz Z (eds) The handbook of evolutionary computation, chap E3.3. IOP Publishing/Oxford University Press, London/Oxford, pp C3.3:1–C3.3:27

Bosman PAN, Thierens D (1999) Linkage information processing in distribution estimation algorithms. In: Proceedings of the GECCO, Orlando, pp 60–67 (also Technical report no UU-CS-1999-10)

Bremermann HJ (1958) The evolution of intelligence. The nervous system as a model of its environment. Technical report no 1, Department of Mathematics, University of Washington

Bulmer MG (1985) The mathematical theory of quantitative genetics. Oxford University Press, Oxford

Burke EK, Newell JP (1999) A multi-stage evolutionary algorithm for the timetabling problem. IEEE Trans Evol Comput 3:63–74

Burke EK, Smith AJ (1999) A memetic algorithm to schedule planned maintenance for the national grid. ACM J Exp Algor 4. doi:10.1145/347792.347801

Burke EK, Smith AJ (2000) Hybrid evolutionary techniques for the maintenance scheduling problem. IEEE Trans Power Syst 15:122–128

Burke EK, Elliman DG, Weare RF (1995) Specialised recombinative operators for timetabling problems. In: Fogarty T (ed) Evolutionary computing AISB workshop 1995. LNCS 993. Springer, Berlin, pp 75–85

Burke EK, Newall JP, Weare RF (1996) A memetic algorithm for university exam timetabling. In: Burke EK, Ross P (eds) The practice and theory of automated timetabling I. LNCS 1153. Springer, Berlin, pp 241–250

Burke EK, Newall JP, Weare RF (1998) Initialisation strategies and diversity in evolutionary timetabling. Evol Comput J 6:81–103

Burke EK, Cowling PI, De Causmaecker P, Vanden Berghe G (2001) A mimetic approach to the nurse rostering problem. Appl Intell 15:199–214

Cantú-Paz E (1999) Migration policies and takeover times in parallel genetic algorithms. In: Proceedings of the GECCO, Orlando, p 775 (also IlliGAL report no 99008)

Cantú-Paz E (2000) Efficient and accurate parallel genetic algorithms. Kluwer, Boston

Chen J-H (2004) Theory and applications of efficient multi-objective evolutionary algorithms. Doctoral dissertation, Feng Chia University, Taiwan

Cheng RW, Gen M (1997) Parallel machine scheduling problems using memetic algorithms. Comput Indust Eng 33:761–764

Costa D (1995) An evolutionary tabu search algorithm and the NHL scheduling problem. INFOR 33:161–178

Davis L (1985) Applying algorithms to epistatic domains. In: Proceedings of the international joint conference on artifical intelligence, Los Angeles, pp 162–164

De Jong KA (1975) An analysis of the behavior of a class of genetic adaptive systems. Doctoral dissertation, University of Michigan (University microfilm no 76-9381)

Dennis JE, Torczon V (1997) Managing approximation models in optimization. In: Alexandrov NM, Hussaini MY (eds) Multidisciplinary design optimization: state-of-the-art. SIAM, Philadelphia, pp 330–347

Fitzpatrick JM, Grefenstette JJ, Van Gucht D (1984) Image registration by genetic search. In: Proceedings of the IEEE southeast conference. IEEE Press, Piscataway, Louisville, KY, pp 460–464

Fleurent C, Ferland J (1993) Genetic hybrids for the quadratic assignment problem. DIMACS series in mathematics and theoretical computer science. This DIMACS workshop on Quadratic Assignment and Related Problems was held at DIMACS 16:173–188

Fraser AS (1957) Simulation of genetic systems by automatic digital computers. II. Effects of linkage on rates under selection. Aust J Biol Sci 10:492–499

Goldberg DE (1983) Computer-aided pipeline operation using genetic algorithms and rule learning. Doctoral dissertation, University of Michigan

Goldberg DE (1987) Simple genetic algorithms and the minimal deceptive problem. In: Davis L (ed) Genetic algorithms and simulated annealing, chap 6. Morgan Kaufmann, Los Altos, pp 74–88

Goldberg DE (1989) Genetic algorithms in search optimization and machine learning. Addison-Wesley, Reading

Goldberg DE (1999a) The race, the hurdle, and the sweet spot: lessons from genetic algorithms for the automation of design innovation and creativity. In: Bentley P (ed) Evolutionary design by computers, chap 4. Morgan Kaufmann, San Mateo, pp 105–118

Goldberg DE (1999b) Using time efficiently: genetic-evolutionary algorithms and the continuation problem. In: Proceedings of the GECCO, Orlando, pp 212–219 (also IlliGAL report no 99002)

Goldberg DE (2002) Design of innovation: lessons from and for competent genetic algorithms. Kluwer, Boston

Goldberg DE, Deb K (1991) A comparitive analysis of selection schemes used in genetic algorithms. Foundations of genetic algorithms. Morgan Kaufmann, pp 69–93

Goldberg DE, Lingle R (1984) Alleles, loci, and the TSP. In: Proceedings of the 1st international conference on genetic algorithms, Pittsburgh, pp 154–159

Goldberg DE, Sastry K (2001) A practical schema theorem for genetic algorithm design and tuning. In: Proceedings of the GECCO, San Francisco, pp 328–335 (also IlliGAL report no 2001017)

Goldberg DE, Segrest P (1987) Finite Markov chain analysis of genetic algorithms. In: Proceedings of the 2nd international conference on genetic algorithms, Cambridge, MA, USA, pp 1–8

Goldberg DE, Voessner S (1999) Optimizing global-local search hybrids. In: Proceedings of the GECCO, Orlando, pp 220–228 (also IlliGAL report no 99001)

Goldberg DE, Korb B, Deb K (1989) Messy genetic algorithms: motivation, analysis, and first results. Complex Syst 3:493–530 (also IlliGAL report no 89003)

Goldberg DE, Deb K, Clark JH (1992) Genetic algorithms, noise, and the sizing of populations. Complex Syst 6:333–362 (also IlliGAL report no 91010)

Goldberg DE, Deb K, Kargupta H, Harik G (1993a) Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In: Proceedings of the international conference on genetic algorithms, Urbana, pp 56–64 (also IlliGAL report no 93004)

Goldberg DE, Thierens D, Deb K (1993b) Toward a better understanding of mixing in genetic algorithms. J Soc Instrum Contr Eng 32:10–16 (also IlliGAL report no 92009)

Goldberg DE, Sastry K, Latoza T (2001) On the supply of building blocks. In: Proceedings of the GECCO, San Francisco, pp 336–342 (also IlliGAL report no 2001015)

Grefenstette JJ, Fitzpatrick JM (1985) Genetic search with approximate function evaluations. In: Proceedings of the international conference on genetic algorithms and their applications, Pittsburgh, pp 112–120

Harik G (1999) Linkage learning via probabilistic modeling in the ECGA (IlliGAL report no 99010). University of Illinois at Urbana-Champaign

Harik G, Goldberg DE (1997) Learning linkage. Foundations of genetic algorithms 4, pp 247–262 (also IlliGAL report no 96006)

Harik G, Lobo F, Goldberg DE (1998) The compact genetic algorithm. In: Proceedings of the IEEE international conference on evolutionary computation, Piscataway, pp 523–528 (also IlliGAL report no 97006)

Harik G, Cantú-Paz E, Goldberg DE, Miller BL (1999) The gambler's ruin problem, genetic algorithms, and the sizing of populations. Evol Comput 7:231–253 (also IlliGAL report no 96004)

Hart WE, Belew RK (1996) Optimization with genetic algorithm hybrids using local search. In: Belew RK, Mitchell M (eds) Adaptive individuals in evolving populations. Addison-Wesley, Reading, pp 483–494

Heckendorn RB, Wright AH (2004) Efficient linkage discovery by limited probing. Evol Comput 12:517–545

Holland JH (1975) Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor

Ibaraki T (1997) Combinations with other optimixation problems. In: Bäck T, Fogel DB, Michalewicz Z (eds) Handbook of evolutionary computation. Institute of Physics Publishing and Oxford University Press, Bristol/New York, pp D3:1–D3:2

Jin Y (2005) A comprehensive survey of fitness approximation in evolutionary computation. Soft Comput J 9:3–12

Kargupta H (1996) The gene expression messy genetic algorithm. In: Proceedings of the international conference on evolutionary computation, Nagoya, pp 814–819

Krasnogor N, Smith JE (2005) A tutorial for competent memetic algorithms: models,taxonomy and design issues. IEEE Trans Evol Comput 9:474–488

Krasnogor N, Hart W, Smith JE (eds) (2004) Recent advances in memetic algorithms. Studies in fuzziness and soft computing, vol 166. Springer, Berlin

Louis SJ, McDonnell J (2004) Learning with case injected genetic algorithms. IEEE Trans Evol Comput 8:316–328

Miller BL, Goldberg DE (1995) Genetic algorithms, tournament selection, and the effects of noise. Complex Syst 9:193–212 (also IlliGAL report no 95006)

Miller BL, Goldberg DE (1996a) Genetic algorithms, selection schemes, and the varying effects of noise. Evol Comput 4:113–131 (also IlliGAL report no 95009)

Miller BL, Goldberg DE (1996b) Optimal sampling for genetic algorithms. Intelligent engineering systems through artificial neural networks, ASME Press, New York 6:291–297

Moscato P (1989) On evolution, search, optimization, genetic algorithms and martial arts: towards memetic algorithms. Technical report C3P 826, California Institute of Technology

Moscato P (1999) Part 4: Memetic algorithms. In: Corne D, Dorigo M, Glover F (eds) New ideas in optimization. McGraw-Hill, New York, pp 217–294

Moscato P, Cotta C (2003) A gentle introduction to memetic algorithms. In: Glover F, Kochenberger G (eds) Handbook of metaheuristics, chap 5. Kluwer, Norwell

Mühlenbein H, Paass G (1996) From recombination of genes to the estimation of distributions I. Binary parameters. PPSN 4, pp 178–187

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

Munetomo M, Goldberg DE (1999) Linkage identification by non-monotonicity detection for overlapping functions. Evol Comput 7:377–398

Oliver JM, Smith DJ, Holland JRC (1987) A study of permutation crossover operators on the travelling salesman problem. In: Proceedings of the 2nd international conference on genetic algorithms, Cambridge, pp 224–230

Paechter B, Cumming A, Luchian H (1995) The use of local search suggestion lists for improving the solution of timetable problems with evolutionary algorithms. In: Fogarty T (ed) Evolutionary computing: AISB workshop 1995. LNCS 993. Springer, Berlin, pp 86–93

Paechter B, Cumming A, Norman MG, Luchian H (1996) Extensions to a memetic timetabling system. In: Burke EK, Ross P (eds) The practice and theory of automated timetabling I. LNCS 1153. Springer, Berlin, pp 251–265

Pelikan M (2005) Hierarchical Bayesian optimization algorithm: toward a new generation of evolutionary algorithm. Springer, Berlin

Pelikan M, Sastry K (2004) Fitness inheritance in the Bayesian optimization algorithm. In: Proceedings of the GECCO 2, Seattle, pp 48–59 (also IlliGAL report no 2004009)

Pelikan M, Goldberg DE, Cantú-Paz E (2000) Linkage learning, estimation distribution, and Bayesian networks. Evol Comput 8:314–341 (also IlliGAL report no 98013)

Pelikan M, Sastry K, Cantú-Paz E (eds) (2006) Scalable optimization via probabilistic modeling: algorithms to applications. Springer, Berlin

Rudolph G (2000) Takeover times and probabilities of non-generational selection rules. In: Proceedings of the GECCO, Las Vegas, pp 903–910

Sakamoto Y, Goldberg DE (1997) Takeover time in a noisy environment. In: Proceedings of the 7th international conference on genetic algorithms, East Lansing, pp 160–165

Sastry K (2001) Evaluation-relaxation schemes for genetic and evolutionary algorithms. Master's thesis, University of Illinois at Urbana-Champaign (also IlliGAL report no 2002004)

Sastry K, Goldberg DE (2003) Scalability of selectorecombinative genetic algorithms for problems with tight linkage. In: Proceedings of the GECCO, Chicago, pp 1332–1344 (also IlliGAL report no 2002013)

Sastry K, Goldberg DE (2004) Let's get ready to rumble: crossover versus mutation head to head. In: Proceedings of the GECCO 2, Seattle, pp 126–137 (also IlliGAL report no 2004005)

Sastry K, Goldberg DE, Pelikan M (2001) Don't evaluate, inherit. In: Proceedings of the GECCO, San Francisco, pp 551–558 (also IlliGAL report no 2001013)

Sastry K, Pelikan M, Goldberg DE (2004) Efficiency enhancement of genetic algorithms building-block-wise fitness estimation. In: Proceedings of the IEEE international congress on evolutionary computation. Portland, OR, USA, pp 720–727

Sastry K, Goldberg DE, Llorà X (2007) Towards billion bit optimization via efficient estimation of distribution algorithms. In: Proceedings of the GECCO, London, pp 577–584 (also IlliGAL report no 2007007)

Sinha A (2003) Designing efficient genetic and evolutionary hybrids. Master's thesis, University of Illinois at Urbana-Champaign (also IlliGAL report no 2003020)

Smith R, Dike B, Stegmann S (1995) Fitness inheritance in genetic algorithms. In: Proceedings of the ACM symposium on applied computing. ACM, New York, pp 345–350

Spears WM, De Jong KA (1994) On the virtues of parameterized uniform crossover. In: Proceedings of the 4th international conference on genetic algorithms, San Diego, pp 230–236

Syswerda G (1989) Uniform crossover in genetic algorithms. In: Proceedings of the 3rd international conference on genetic algorithms, San Mateo, pp 2–9

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

Thierens D, Goldberg DE (1994) Convergence models of genetic algorithm selection schemes. PPSN 3, Springer, Berlin/New York, pp 116–121

Thierens D, Goldberg DE, Pereira AG (1998) Domino convergence, drift, and the temporal-salience structure of problems. In: Proceedings of the IEEE international congress on evolutionary computation, pp 535–540

Valenzuala J, Smith AE (2002) A seeded memetic algorithm for large unit commitment problems. J Heuristics 8:173–196

Watson JP, Rana S, Whitely LD, Howe AE (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: Winter G, Périaux J, Galán M, Cuesta P (eds) Genetic algorithms in engineering and computer science. Wiley, Chichester, pp 191–201

Yu T-L (2006) A matrix approach for finding extrema: problems with modularity, hierarchy, and overlap. Doctoral dissertation, University of Illinois at Urbana-Champaign (also IlliGAL report no 2007012)