PART 1

THEORY OF MODERN HEURISTIC OPTIMIZATION

CHAPTER 1

Introduction to Evolutionary Computation

DAVID B. FOGEL

1.1 INTRODUCTION

Darwinian evolution is intrinsically a robust search and optimization mechanism. Living organisms demonstrate optimized complex behavior at every level: the cell, the organ, the individual, and the population. The problems that biological species have solved are typified by chaos, chance, temporality, and nonlinear interactivities. These are also characteristics of problems that have proved to be especially intractable to classic methods of optimization and appear routinely in the area of power systems. The evolutionary process can be applied to these problems, where heuristic solutions are not available or generally lead to unsatisfactory results. As a result, evolutionary algorithms have recently received increased interest, particularly with regard to the manner in which they may be applied for practical problem solving.

Evolutionary computation, the term now used to describe the field of investigation that concerns all evolutionary algorithms, offers practical advantages to the researcher facing difficult optimization problems. These advantages are multifold, including the simplicity of the approach, its robust response to changing circumstance, its flexibility, and many other facets. This chapter summarizes some of these advantages, offers a brief review of some parts of evolutionary computation theory, and introduces a new optimization technique that models swarming behavior in insects or schooling in fish. The reader who wants to further review the basic concepts of evolutionary algorithms is referred to Fogel [1–3], Bäck [4], and Michalewicz [5].

Modern Heuristic Optimization Techniques. Edited by K. Y. Lee and M. A. El-Sharkawi

1.2 ADVANTAGES OF EVOLUTIONARY COMPUTATION

1.2.1 Conceptual Simplicity

A primary advantage of evolutionary computation is that it is conceptually simple. The main flowchart that describes every evolutionary algorithm applied for function optimization is depicted in Fig. 1.1. The algorithm consists of initialization, which may be a purely random sampling of possible solutions, followed by iterative variation and selection in light of a performance index. This figure of merit must assign a numeric value to any possible solution such that two competing solutions can be rank ordered. Finer granularity is not required. Thus, the criterion need not be specified with the precision that is required of some other methods. In particular, no gradient information needs to be presented to the algorithm. Over iterations of

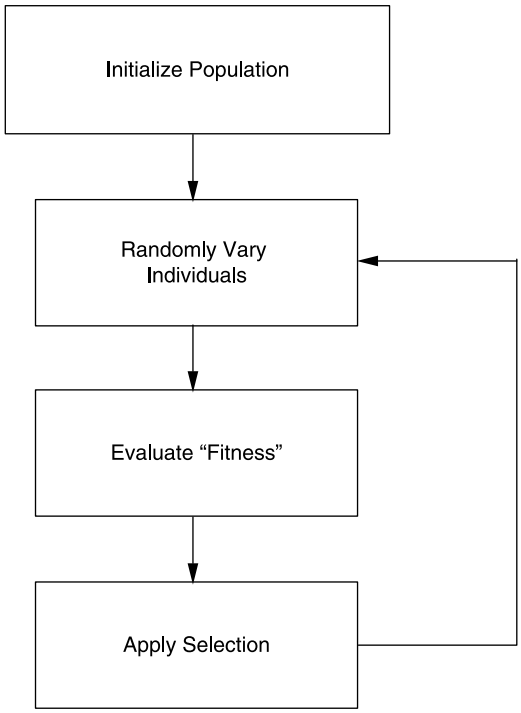

FIGURE 1.1 The main flowchart of the vast majority of evolutionary algorithms. A population of candidate solutions to a problem at hand is initialized. This often is accomplished by randomly sampling from the space of possible solutions. New solutions are created by randomly varying existing solutions. This random variation may include mutation and/or recombination. Competing solutions are evaluated in light of a performance index describing their "fitness" (or equivalently, their error). Selection is then applied to determine which solutions will be maintained into the next generation, and with what frequency. These new "parents" are then subjected to random variation, and the process iterates.

random variation and selection, the population can be made to converge asymptotically to optimal solutions (Fogel [6], Rudolph [7], and others).

The evolutionary search is similar to the view offered by Wright [8] involving "adaptive landscapes." A response surface describes the fitness assigned to alternative genotypes as they interact in an environment (Fig. 1.2). Each peak corresponds with an optimized collection of behaviors (phenotypes), and thus one or more sets of optimized genotypes. Evolution probabilistically proceeds up the slopes of the topography toward peaks as selection culls inappropriate phenotypic variants. Others (Atmar [9], Raven and Johnson [10], pp. 400–401) have suggested that it is more appropriate to view the adaptive landscape from an inverted position. The peaks become troughs, or "minimized prediction error entropy wells" (Atmar [9]). Such a viewpoint is intuitively appealing. Searching for peaks depicts evolution as a slowly advancing, tedious, and uncertain process. Moreover, there appears to be a certain fragility to an evolving phyletic line; an optimized population might be expected to quickly fall off the peak under slight perturbations. The inverted topography leaves an altogether different impression. Populations advance rapidly, falling down the walls of the error troughs until their cohesive set of interrelated behaviors is optimized. The topography is generally in flux, as a function of environmental erosion

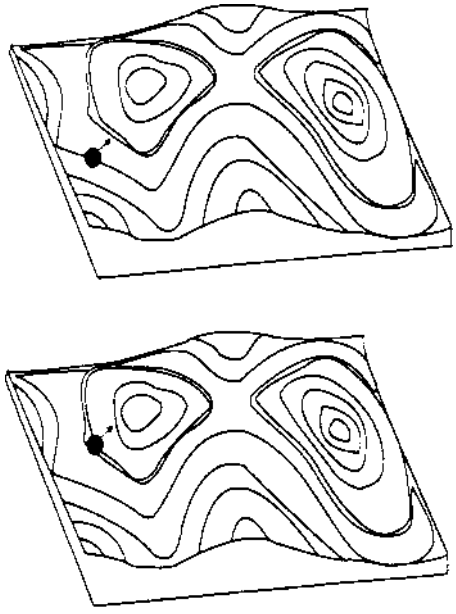

FIGURE 1.2 Evolution on an inverted adaptive topography. A landscape is abstracted to represent the fitness of alternative phenotypes and, as a consequence, alternative genotypes. Rather than viewing the individuals or populations as maximizing fitness and thereby climbing peaks on the landscape, a more intuitive perspective may be obtained by inverting the topography. Populations proceed down the slopes of the topography toward valleys of minimal predictive error.

and variation, as well as other evolving organisms, and stagnation may never set in. Regardless of which perspective is taken, maximizing or minimizing, the basic evolutionary algorithm is the same: a search for the extrema of a functional describing the objective worth of alternative candidate solutions to the problem at hand.

The procedure may be written as the difference equation

$$\mathbf{x}[t+1] = s(v(\mathbf{x}[t])), \tag{1.1}$$

where $\mathbf{x}[t]$ is the population at time t under a representation $\mathbf{x}$, v is a random variation operator, and s is the selection operator (Fogel and Ghozeil [11]). There are a variety of possible representations, variation operators, and selection methods (Bäck et al. [12]). Not more than about 10–12 years ago, there was a general recommendation that the best representation was a binary coding, as this provided the greatest "implicit parallelism" (more detail is offered later in this chapter, and see also Goldberg [13]). But this representation was often cumbersome to implement (consider encoding the solution to a traveling salesman problem as a string of symbols from $\{0, 1\}$), and empirical results did not support any necessity, or even benefit, to binary representations (e.g., Davis [14], Michalewicz [15], Koza [16]). Moreover, the suggestions that advantages would accrue from recombining alternative solutions through crossover operators and amplifying solutions based on their relative fitness also did not obtain empirical support (e.g., Fogel and Atmar [17], Bäck and Schwefel [18], Fogel and Stayton [19], and many others). Recent mathematical results have proved that there can be no best choice for these facets of an evolutionary algorithm that would hold across all problems (Wolpert and Macready [20]), and even that there is no best choice of representation for any individual problem (Fogel and Ghozeil [21]). The effectiveness of an evolutionary algorithm depends on the interplay between the operators s and v as applied to a chosen representation $\mathbf{x}$ and initialization $\mathbf{x}[0]$. This dependence provides freedom to the human operator to tailor the evolutionary approach for their particular problem of interest.

1.2.2 Broad Applicability

Evolutionary algorithms can be applied to virtually any problem that can be formulated as a function optimization task. It requires a data structure to represent solutions, a performance index to evaluate solutions, and variation operators to generate new solutions from old solutions (selection is also required but is less dependent on human preferences). The state space of possible solutions can be disjoint and can encompass infeasible regions, and the performance index can be time varying or even a function of competing solutions extant in the population. The human designer can choose a representation that follows his or her intuition. In this sense, the procedure is representation independent, in contrast with other numerical techniques that might be applicable for only continuous values or other constrained sets. Representation should allow for variation operators that maintain a behavioral link between parent and offspring. Small changes in the structure of a parent should

lead to small changes in the resulting offspring, in order to facilitate an understanding of the problem space, and likewise large changes should engender gross alterations. A continuum of possible changes should be allowed such that the effective "step size" of the algorithm can be tuned, perhaps online in a self-adaptive manner (discussed later). This flexibility allows for applying essentially the same procedure to discrete combinatorial problems, continuous-valued parameter optimization problems, mixed-integer problems, and so forth.

1.2.3 Outperform Classic Methods on Real Problems

Real-world function optimization problems often (1) impose nonlinear constraints, (2) require payoff functions that are not concerned with least-squared error, (3) involve nonstationary conditions, (4) incorporate noisy observations or random processing, or include other vagaries that do not conform well to the prerequisites of classic optimization techniques. The response surfaces posed in real-world problems are often multimodal, and gradient-based methods converge rapidly to local optima (or perhaps saddle points), which may yield insufficient performance. For simpler problems, where the response surface is, say, strongly convex, evolutionary algorithms do not perform as well as traditional optimization methods (Bäck [4]). But this is to be expected as these traditional techniques were designed to take advantage of the convex property of such surfaces. Schwefel [22] has shown in a series of empirical comparisons that in the obverse condition of applying classic methods to multimodal functions, evolutionary algorithms offer a significant advantage. In addition, in the often-encountered case of applying linear programming to problems with nonlinear constraints, this offers an almost certainly incorrect result because the assumptions required for the technique are violated. In contrast, evolutionary computation can directly incorporate arbitrary linear and nonlinear constraints (Michalewicz [5]).

Moreover, the problem of defining the payoff function for optimization lies at the heart of success or failure: Inappropriate descriptions of the performance index lead to generating the right answer for the wrong problem. Within classic statistical methods, concern is often devoted to minimizing the squared error between forecast and actual data. But in practice, equally correct predictions are not of equal worth, and errors of identical magnitude are not equally costly. Consider the case of correctly predicting that a particular customer will demand 10 units of energy in a particular time period. This is typically worth less than correctly predicting that the customer will demand 100 units of energy, yet both predictions engender zero error and are weighted equally in classic statistics. Further, the error of predicting the customer will demand 10 units and having them actually demand 100 units is not of equal cost to the energy supplier as predicting the customer will demand 100 units and having them demand 10. One error leaves a missed opportunity cost and the other leaves a 90-unit oversupply. Yet again, under a squared error criterion, these two situations are treated identically. In contrast, within evolutionary algorithms, any definable payoff function can be used to judge the appropriateness of alternative

behaviors. There is no restriction that the criteria be differentiable, smooth, or continuous.

1.2.4 Potential to Use Knowledge and Hybridize with Other Methods

It is always reasonable to incorporate domain-specific knowledge into an algorithm when addressing particular real-world problems. Specialized algorithms can outperform unspecialized algorithms on a restricted domain of interest (Wolpert and Macready [20]). Evolutionary algorithms offer a framework such that it is comparably easy to incorporate such knowledge. For example, specific variation operators may be known to be useful when applied to particular representations (e.g., 2-OPT on the traveling salesman problem). These can be applied directly as mutation or recombination operations. Knowledge can also be implemented into the performance index, in the form of known physical or chemical properties (e.g., van der Waals interactions: Gehlhaar et al. [23]). Incorporating such information focuses the evolutionary search, yielding a more efficient exploration of the state space of possible solutions.

Evolutionary algorithms can also be combined with more traditional optimization techniques. This may be as simple as the use of a conjugate-gradient minimization used after primary search with an evolutionary algorithm (e.g., Gehlhaar et al. [23]), or it may involve simultaneous application of algorithms (e.g., the use of evolutionary search for the structure of a model coupled with gradient search for parameter values; Harp et al. [24]). There may also be a benefit to seeding an initial population with solutions derived from other procedures (e.g., a greedy algorithm; Fogel and Fogel [25]). Furthermore, evolutionary computation can be used to optimize the performance of neural networks (Angeline et al. [26]), fuzzy systems (Haffner and Sebald [27]), production systems (Wilson [28]), and other program structures (Koza [16], Angeline and Fogel [29]). In many cases, the limitations of conventional approaches (e.g., the requirement for differentiable hidden nodes when using back propagation to train a neural network) can be avoided.

1.2.5 Parallelism

Evolution is a highly parallel process. As distributed processing computers become more readily available, there will be a corresponding increased potential for applying evolutionary algorithms to more complex problems. It is often the case that individual solutions can be evaluated independently of the evaluations assigned to competing solutions. The evaluation of each solution can be handled in parallel, and only selection (which requires at least pairwise competition) requires some serial processing. In effect, the running time required for an application may be inversely proportional to the number of processors. Regardless of these future advantages, current desktop computing machines provide sufficient computational speed to generate solutions to difficult problems in reasonable time (e.g., the evolution of a neural network for classifying features of breast carcinoma involving more than 5 million separate

function evaluations requires only about 3 hours on a 200-MHz 604e PowerPC, Fogel et al. [30], or equivalently one third of an hour on a 2-GHz PC).

1.2.6 Robust to Dynamic Changes

Traditional methods of optimization are not robust to dynamic changes in the environment and often require a complete restart in order to provide a solution (e.g., dynamic programming). In contrast, evolutionary algorithms can be used to adapt solutions to changing circumstance. The available population of evolved solutions provides a basis for further improvement and in most cases it is not necessary, nor desirable, to reinitialize the population at random. Indeed, this procedure of adapting in the face of a dynamic environment can be used to advantage. For example, Wieland [31] used a genetic algorithm to evolve recurrent neural networks to control a cart-pole system comprising two poles (Fig. 1.3). The degree of difficulty depended on the relative pole lengths (i.e., the closer the poles were to each other in length, the more difficult the control problem). Wieland [31] developed controllers for a case of one pole of length 1.0 m and the other of 0.9 m by successively controlling systems where the shorter pole was started at 0.1 m and incremented sequentially to 0.9 m. At each increment, the evolved population of networks served as the basis

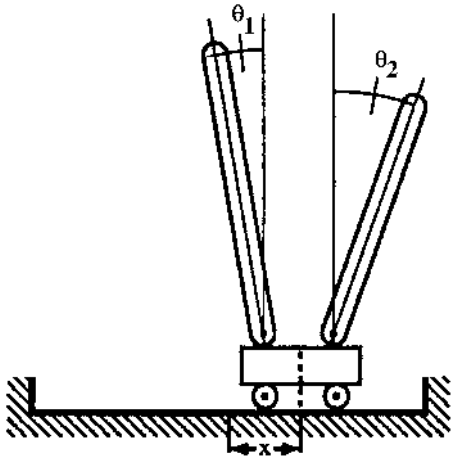

FIGURE 1.3 A cart with two poles. The objective is to maintain the cart between the limits of the track while not allowing either pole to exceed a specified maximum angle of deflection. The only control available is a force with which to push or pull on the cart. The difficulty of the problem is dependent on the similarity in pole lengths. Wieland [31] and Saravanan and Fogel [32] used evolutionary algorithms to optimize neural networks to control this plant for pole lengths of 1.0 m and 0.9 m. The evolutionary procedure required starting with poles of 1.0 m and 0.1 m and iteratively incrementing the length of the shorter pole in a series of dynamic environments. In each case, the most recent evolved population served as the basis for new trials, even when the pole length was altered.

for a new set of controllers. A similar procedure was offered in Saravanan and Fogel [32] and Fogel [33].

The ability to adapt on the fly to changing circumstance is of critical importance to practical problem solving. For example, suppose that a particular simulation provides perfect fidelity to an industrial production setting. All workstations and processes are modeled exactly, and an algorithm is used to find a "perfect" schedule to maximize production. This perfect schedule will, however, never be implemented in practice because by the time it is brought forward for consideration, the plant will have changed: machines may have broken down, personnel may not have reported to work or failed to keep adequate records of prior work in progress, other obligations may require redirecting the utilization of equipment, and so forth. The "perfect" plan is obsolete before it is ever implemented. Rather than spend considerable computational effort to find such perfect plans, a better prescription is to spend less computational effort to discover suitable plans that are robust to expected anomalies and can be evolved on the fly when unexpected events occur.

1.2.7 Capability for Self-Optimization

Most classic optimization techniques require appropriate settings of exogenous variables. This is true of evolutionary algorithms as well. However, there is a long history of using the evolutionary process itself to optimize these parameters as part of the search for optimal solutions (Reed et al. [34], Rosenberg [35], and others). For example, suppose a search problem requires finding the real-valued vector that minimizes a particular functional $f(\mathbf{x})$, where $\mathbf{x}$ is a vector in n dimensions. A typical evolutionary algorithm (Fogel [1]) would use Gaussian random variation on current parent solutions to generate offspring:

$$x_i' = x_i + \sigma_i N(0, 1), \tag{1.2}$$

where the subscript indicates the ith dimension, and σ_i is the standard deviation of a Gaussian random variable, denoted by $N(0, 1)$. Setting the "step size" of the search in each dimension is critical to the success of the procedure (Fig. 1.4). This can be accomplished in the following two-step fashion:

$$\sigma_i' = \sigma_i \exp(\tau N(0, 1) + \tau' N_i(0, 1)) \tag{1.3}$$

$$x_i' = x_i + \sigma_i' N_i(0, 1), \tag{1.4}$$

where $\tau \propto (2n)^{0.5}$ and $\tau' \propto (2n^{0.5})^{0.5}$ (Bäck and Schwefel [18]). In this manner, the standard deviations are subject to variation and selection at a second level (i.e., the level of how well they guide the search for optima of the functional $f(\mathbf{x})$). This general procedure has also been found effective in addressing discrete optimization problems (Angeline et al. [36], Chellapilla and Fogel [37], and others). Essentially,

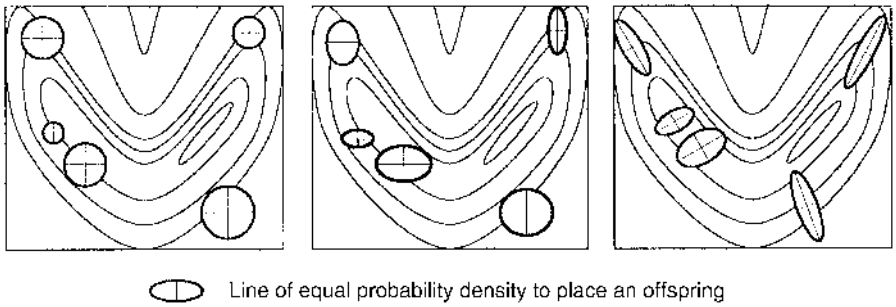

FIGURE 1.4 When using Gaussian mutations in all dimensions (as in evolution strategies or evolutionary programming), the contours of equal probability density for placing offspring are depicted above (Bäck [4]). In the left panel, all standard deviations in each dimension are equal, resulting in circular contours. In the middle panel, the standard deviations in each dimension may vary, but the perturbation in each dimension is independent of the others (zero covariance), resulting in elliptical contours. In the right panel, arbitrary covariance is applied, resulting in contours that are rotatable ellipses. The method of self-adaptation described in text can be extended to adapt arbitrary covariances, thereby allowing the evolutionary algorithm to adapt to the changes in the response surface during the search for the optimum position on the surface. Similar procedures have been offered for self-adaptation when solving discrete combinatorial problems.

the effect is much like a temperature schedule in simulated annealing; however, the schedule is set by the algorithm as it explores the state space rather than *a priori* by a human operator.

1.2.8 Able to Solve Problems That Have No Known Solutions

Perhaps the greatest advantage of evolutionary algorithms comes from the ability to address problems for which there are no human experts. Although human expertise should be used when it is available, it often proves less than adequate for automating problem-solving routines. Troubles with such expert systems are well-known: the experts may not agree, may not be self-consistent, may not be qualified, or may simply be in error. Research in artificial intelligence has fragmented into a collection of methods and tricks for solving particular problems in restricted domains of interest. Certainly, these methods have been successfully applied to specific problems (e.g., the chess program Deep Blue). But most of these applications require human expertise. They may be applied impressively to difficult problems requiring great computational speed, but they generally do not advance our understanding of intelligence. "They solve problems, but they do not solve the problem of how to solve problems" (Fogel [1]). In contrast, evolution provides a method for solving the problem of how to solve problems even in the absence of human expertise. It is a recapitulation of the scientific method (Fogel et al. [38]) that can be used to learn fundamental aspects of any measurable environment.

1.3 CURRENT DEVELOPMENTS

As indicated above, evolutionary computation has a long history with several independent beginnings. Each of these beginnings, whether they occurred in the simulation of genetic systems (Fraser [39], Bremermann [40], Holland [41]), engineering optimization (Rechenberg [42]), artificial intelligence (Fogel et al. [38]), or other areas, had specific traits that were originally unique to them individually. For example, the classic genetic algorithm of Holland [41] operated on binary strings (regardless of the problem). In contrast, the classic work of Rechenberg and Schwefel (termed *evolution strategies*) relied on real-valued representations, whereas the classic evolutionary programming work of L. Fogel used finite state machines. These issues of different representations, as well as variation operators, selection methods, and other particular aspects of evolutionary algorithms, have grown to be virtually nonexistent in current evolutionary computation practice. It now makes little sense (scientifically) to speak of these originally different methods as being currently disparate: They are all plainly similar in the extreme, all relying on diverse possible representations, with many choices for variation and selection.

This similarity has grown mainly from increased communications across the groups over the past decade, as well as a recognition that early theory in evolutionary algorithms had many flaws. Some historical and current aspects of evolutionary algorithm theory are offered here.

1.3.1 Review of Some Historical Theory in Evolutionary Computation

Some early efforts (1975–1990) in evolutionary algorithm theory focused on (1) the belief that it is possible to generate superior general problem solvers, (2) the notion that maximizing implicit parallelism is useful, (3) a schema theorem to describe the propagation of components of solutions in a population, and (4) an analysis of a two-armed bandit problem that was intended to indicate an optimal sampling plan for evolutionary algorithms. Unfortunately, the conventional wisdom regarding these four focal points from even just a decade ago has been shown to be either incomplete or incorrect. As a result, the foundations of evolutionary computation have been reconsidered, and an integrated approach to the study of evolutionary computation as a whole has been undertaken.

1.3.2 No Free Lunch Theorem

It is natural to ask if there is a best evolutionary algorithm that would always give superior results across the possible range of problems. Is there some choice of variation operators and selection mechanisms that will always outperform all other choices regardless of the problem? Sadly, the answer is no: There is no best evolutionary algorithm. In mathematical terms, let an algorithm a be represented as a mapping from previously unvisited sets of points to a single new (i.e., previously

unvisited) point in the state space of all possible points (solutions). Let $P(d_m^y|f, m, a)$ be the conditional probability of obtaining a particular sample d_m when algorithm a is iterated m times on cost function f. Given these preliminaries, Wolpert and Macready [20] proved the so-called no free lunch theorem:

Theorem 1.1 (No Free Lunch). For any pair of algorithms a_1 and a_2,

$$\sum_f P\left(d_m^y|f, m, a_1\right) = \sum_f P\left(d_m^y|f, m, a_2\right).$$

(See Appendix A of Wolpert and Macready [20] for the proof; English [43] showed that a similar no free lunch result holds whenever the values assigned to points are independent and identically distributed random variables.) That is, the sum of the conditional probabilities of visiting each point d_m is the same over all cost functions f regardless of the algorithm chosen. The immediate corollary of this theorem is that for any performance measure $\Phi(d_m^y)$, the average over all f of $P(\Phi(d_m^y)|f, m, a)$ is independent of a. In other words, there is no best algorithm, whether or not that algorithm is "evolutionary," and moreover whatever an algorithm gains in performance on one class of problems is necessarily offset by that algorithm's performance on the remaining problems.

This simple theorem has engendered a great deal of controversy in the field of evolutionary computation and some associated misunderstanding. There has been considerable effort expended in finding the "best" set of parameters and operators for evolutionary algorithms since at least the mid-1970s. These efforts have involved the type of recombination, the probabilities for crossover and mutation, the representation, the population size, and so forth. Most of this research involved empirical trials on benchmark functions. But the no free lunch theorem essentially dictates that the conclusions made on the basis of such sampling are in the strict mathematical sense limited to only those functions studied. Efforts to find the best crossover rate, the best mutation operator, and so forth, in the absence of restricting attention to a particular class of problems and methods are pointless.

For an algorithm to perform better than even random search (which is simply another algorithm), it must reflect something about the structure of the problem it faces. By consequence, it mismatches the structure of some other problem. Note too that it is not enough to simply identify that a problem has some structure associated with it: that structure must be appropriate to the algorithm at hand. Moreover, the structure must be specific. It is not enough to say, as is often heard, "I am concerned only with real-world problems, not all possible problems, and therefore the no free lunch theorem does not apply." What is the structure of "real-world" problems? Indeed, what is a real-world problem? The obvious vague quality of this description is immediately problematic. What constitutes a real-world problem now might not have been a problem at all, say, 100 years ago (e.g., what to watch on television on a Thursday night). Regardless, simply narrowing the domain of possible problems without identifying the correspondence between the set of problems considered and

the algorithm at hand does not suffice to claim any advantage for a particular method of problem solving.

One apt example of how the match between an algorithm and the problem can be exploited was offered in De Jong et al. [44]. For a very simple problem of finding the two-bit vector $\mathbf{x}$ that maximizes the function

$$f(\mathbf{x}) = \text{integer}(\mathbf{x}) + 1,$$

where integer($\mathbf{x}$) returns 0 for [00], 1 for [01], 2 for [10], and 3 for [11], De Jong et al. [44] employed an evolutionary algorithm with (1) one-point crossover at either a probability of 1.0 or 0.0, (2) a constant mutation rate of 0.1, and (3) a population of size 5. In this trivial example, it was possible to calculate the exact probability that the global best vector would be contained in the population as a function of the number of generations. In this case, the use of crossover definitely increased the likelihood of discovering the best solution, and mutation alone was better than random search.

In this example, the function f assigned values $\{1, 2, 3, 4\}$ to the vectors $\{$[00], [01], [10], [11]$\}$, respectively, but this is not the only way to assign these fitness values to the possible strings. In fact, there are $4! = 24$ different permutations that could be used. De Jong et al. [44] showed that, in this case, the performance obtained with the 24 different permutations falls into three equivalence classes, each containing eight permutations that produce identical probability of success curves. In the second and third equivalence classes, the use of crossover was seen to be detrimental to the likelihood of success, and in fact random search outperformed evolutionary search for the first 10–20 generations in the third equivalence class. In the first equivalence class, crossover could combine the second- and third-best vectors to generate the best vector. In the second and third equivalence classes, it could not usefully combine these vectors: the structure of the problem did not match the structure of the crossover operator. Any specific search operator can be rendered superior or inferior simply by changing the structure of the problem.

Intrinsic to every evolutionary algorithm is a representation for manipulating candidate solutions to the problem at hand. The no free lunch theorem establishes that there is no best evolutionary algorithm across all problems. The fundamental result is twofold: (1) claims that evolutionary algorithms must rely on specific operators to be successful (e.g., the less-often-heard but still occasional claim that crossover is a necessary component of a successful evolutionary algorithm, as found in [13]) are not correct, and (2) efforts to make generally superior evolutionary algorithms are misguided.

1.3.3 Computational Equivalence of Representations

Holland ([41], p. 71) suggested that alternative representations in an evolutionary algorithm could be compared by calculating the number of schemata (subset templates, see below) processed by the algorithm. In order to maximize this *intrinsic*

parallelism, it was recommended that representations should be chosen with the fewest "detectors with a range of many attributes." In other words, alphabets with low cardinality were to be favored because they generate more schemata. For example, six elements, each with a range of 10 values, can generate 1 million distinct representations, which is about the same as 20 elements with a range of 2 values ($2^{20} = 1{,}048{,}576$). But the number of schemata processed is 11^6 ($=1{,}771{,}561$) versus 3^{20} ($=3.49 \times 10^9$). This increased number of schemata was suggested to give a "larger information flow" to reproductive plans such as genetic algorithms. Emphasis was therefore placed on binary representations, this offering the lowest possible cardinality and the greatest number of schemata.

To review, a schema is a template with fixed and variable symbols. Consider a string of symbols from an alphabet $\mathbf{A}$. Suppose that some of the components of the string are held fixed while others are free to vary. Following Holland [41], define a wild card symbol, $\# \in \mathbf{A}$, that matches any symbol from $\mathbf{A}$. A string with fixed and/or variable symbols defines a schema, which is a set denoted by a string over the union of $\{\#\}$ and the alphabet $\mathbf{A} = \{0, 1\}$. Consider the schema [01##], which includes [0100], [0101], [0110], and [0111]. Holland ([41], pp. 64–74) offered that every evaluated string actually offers partial information about the expected fitness of all possible schemata in which that string resides. That is, if string [0000] is evaluated to have some fitness, then partial information is also received about the worth of sampling from variations in [####], [0###], [#0##], [#00#], [#0#0], and so forth. This characteristic was termed *intrinsic parallelism* (or *implicit parallelism*), in that through a single sample, information is gained with respect to many schemata.

Antonisse [45] offered a different interpretation of the wild card symbol # that led to an alternative recommendation regarding the cardinality of a chosen representation. Rather than view the # symbol in a string as a "don't care" character, it can be viewed as indicating all possible subsets of symbols at the particular position. If $\mathbf{A} = \{0, 1, 2\}$, then the schema [000#] would indicate the sets {[0000] [0001]}, {[0000] [0002]}, {[0001] [0002]}, and {[0000] [0001] [0002]} as the # symbol indicates the possibilities of (a) 0 or 1, (b) 0 or 2, (c) 1 or 2, and (d) 0, 1, or 2. When schemata are viewed in this manner, the greater implicit parallelism comes from the use of more, not fewer, symbols.

Fogel and Ghozeil [21] showed that, in contrast with the arguments offered in both Holland [41] and Antonisse [45], there can be no intrinsic advantage to any choice of cardinality: Given some weak assumptions about the structure of the problem space and representation, equivalent evolutionary algorithms can be generated for any choice of cardinality. Moreover, the theorems in Ref. 21 indicate that there can be no intrinsic advantage to using any particular two-parent search operator (e.g., a crossover) as there always exists an alternative two-parent operator that performs the equivalent function, regardless of the chosen representation. The proofs carry considerable notation, and the reader is recommended to Ref. 21 to review them if interested.

These theorems from Ref. 21 provide an extension of the result offered in Battle and Vose [46] where it was shown that isomorphisms exist between alternative

instances of genetic algorithms for binary representations (i.e., the operations of crossover and mutation on a population under a particular binary representation in light of a fitness function can be mapped equivalently to alternative similar operators for any other binary representation). They also extend the results of Vose and Liepins [47] and Radcliffe [48], where it was shown that there can be no general advantage for any particular binary representation. Although particular representations and operators may be more computationally tractable or efficient than others in certain cases, or may appeal to the designer's intuition, under the conditions studied in Ref. 21, no choice of representation, or one-point or two-point variation operator, can offer a capability not found in another choice of representation or analogous operator.

1.3.4 Schema Theorem in the Presence of Random Variation

The traditional method of selection in genetic algorithms (from, say, Holland [41]) requires solutions to be reproduced in proportion to their fitness relative to the other solutions in the population (sometimes termed *roulette wheel selection* or *reproduction with emphasis*). That is, if the fitness of the ith string in the population at time t, $\mathbf{x}_i^t$, is denoted by $f(\mathbf{x}_i^t)$, then the probability of selecting the ith string for reproduction for each available slot in the population at time $t+1$ is given by

$$P(\mathbf{x}_i^{t+1}) = \frac{f(\mathbf{x}_i^t)}{\sum_{i=1}^{n} f(\mathbf{x}_i^t)}, \tag{1.5}$$

where there are n members of the population at time t. This procedure requires strictly positive fitness values.

The implementation of proportional selection leads to the well-known variant of the schema theorem (Holland [41]):

$$EP(H, t+1) = P(H, t)\frac{f(H,t)}{\bar{f}_t}, \tag{1.6}$$

where H is a particular schema (the notation of H is used to denote the schema as a hyperplane), $P(H, t)$ is the proportion of strings in the population at time t that are an instance of schema H, $f(H, t)$ is the mean fitness of strings in the population at time t that are an instance of H, $\bar{f}_t$ is the mean fitness of all strings in the population at time t, and $EP(H, t+1)$ denotes the expected proportion of strings in the population at time $t+1$ that will be instances of the schema H. Equation (1.2) does not include any effects of recombination or mutation. It only describes the effects of proportional selection on a population in terms of the manner in which schemata are expected to increase or decrease over time.

The fitness associated with a schema depends on which instances of that schema are evaluated. Moreover, in real-world practice, the evaluation of a string will often include some random effects (e.g., observation error, time dependency,

uncontrollable exogenous variables). That is, the observed fitness of a schema H (or any element of that schema) may not be described by a constant value, but rather by a random variable with an associated probability density function. Selection operates not on the mean of all possible samples of the schema H but only on the fitness associated with each observed instance of the schema H in the population. It is therefore of interest to assess the expected allocation of trials to schemata when their observed fitness takes the form of a random variable. Fogel and Ghozeil [49] showed that this can result in the introduction of a bias such that the expected sampling from alternative schemata will not be in proportion to their mean fitness.

The schema theorem of Holland [41] refers only to the specific realized values of competing schemata in a population; it is not intended to handle the case when the fitness of these schemata are described by random variables. The analysis in Fogel and Ghozeil [49] (cf., Rana et al. [50], see also Gillespie [51]) indicates that the expected proportion of a particular schema H in the population at the next time step is not generally governed by the ratio of the mean of that schema H to the sum of the means of schema H and schema H' (everything that is not in H). In general, there is no *a priori* reason to expect the schema theorem to adequately describe the mean sampling of alternative schemata when the fitness evaluation of those schemata is governed by random variation. This may occur in the form of observation noise on any particular string, random selection of individual strings from a particular schema, or a number of other mechanisms. Under such conditions, reliably estimating mean hyperplane performance is not sufficient to predict the expected proportion of samples that will be allocated to a particular hyperplane under proportional selection. The fundamental result is that the schema theorem of Ref. 41 cannot, in general, be used to reliably predict the average representation of schemata in future populations when schema fitness depends on random factors.

1.3.5 Two-Armed Bandits and the Optimal Allocation of Trials

Recalling the idea of sampling from alternative schemata, in order to optimally allocate trials to schemata, a loss function describing the worth of each trial must be formulated. Holland ([41], pp. 75–83) considered and analyzed the problem of minimizing expected losses while sampling from alternative schemata. The essence of this problem can be modeled as a two-armed bandit (slot machine): Each arm of the bandit is associated with a random payoff described by a mean and variance (much like the result of sampling from a particular schema where the payoff depends on which instance of that schema is sampled). The goal is to best allocate trials to the alternative arms conditioned on the payoffs that have already been observed in previous trials. Holland ([41], pp. 85–87) extended the case of two competing schemata to any number of competing schemata. The results of this analysis were used to guide the formulation of one early version of evolutionary algorithms, so it is important to review this analysis and its consequences. This is particularly true because the formulation has recently been shown mathematically to be flawed (Rudolph [52]; Macready and Wolpert [53]).

Holland ([41], pp. 75–83) examined the two-armed bandit problem where there are two random variables, RV_1 and RV_2 (representing two slot machines) from which samples may be taken (much like schemata represent samples from a solutions space). The two random variables are assumed to be independent and possess some unknown means (μ_1, μ_2) and unknown variances (σ_1^2, σ_2^2). A trial is conducted by sampling from a chosen random variable; the result of the trial is the payoff. Suppose some number of trials has been devoted to each random variable (n_1 and n_2, respectively) and that the average payoff (the sum of the individual payoffs divided by the number of trials) from one random variable is greater than for the other. Let RV_{high} be the random variable for which the greater average payoff has been observed (not necessarily the random variable with the greater mean), and let RV_{low} be the other random variable. The objective in this problem is to allocate trials to each random variable so as to minimize the expected loss function

$$L(n_1, n_2) = [q(n_1, n_2)n_1 + (1 - q(n_1, n_2))n_2] \cdot |\mu_1 - \mu_2|, \tag{1.7}$$

where $L(n_1, n_2)$ describes the total expected loss from allocating n_1 samples to RV_1 and n_2 samples to RV_2, and

$$q(n_1, n_2) = \begin{cases} \Pr(\dot{x}_1 > \dot{x}_2) & \text{if } \mu_1 < \mu_2 \\ \Pr(\dot{x}_1 > \dot{x}_2) & \text{if } \mu_1 > \mu_2 \end{cases},$$

where x_1 and x_2 designate the mean payoffs (sample means) for having allocated n_1 and n_2 samples to RV_1 and RV_2, respectively. Note that specific measurements and the number of samples yield explicit values for the sample means.

As summarized in Goldberg ([13], p. 37), Holland ([41], pp. 77–78) proved the following (notation consistent with Holland [41]):

Theorem 1.2 Given N trials to be allocated to two random variables with means $\mu_1 > \mu_2$ and variances σ_1^2 and σ_2^2, respectively, and the expected loss function described in Eq. (1.7), the minimum expected loss results when the number of trials allocated to the random variable with the lower observed average payoff is

$$n^* \cong b^2 \ln\left[\frac{N^2}{8\pi b^4 \ln N^2}\right], \tag{1.8}$$

where $b = \sigma_1/(\mu_1 - \mu_2)$. The number of trials to be allocated to the random variable with the higher observed average payoff is $N - n^*$.

If the assumptions associated with the proof (Holland [41], pp. 75–83) held, and if the mathematical framework were correct, the above analysis would apply equally well to a k-armed bandit as to a two-armed bandit.

Unfortunately, as offered in Macready and Wolpert [53], the error in Holland ([41], pp. 77–85) stems from considering the unconditioned expected loss for allocating $N - 2n$ trials after allocating n to each bandit, rather than the expected loss conditioned on the information available after the $2n$ pulls. Macready and Wolpert [53] showed that a simple greedy strategy that pulls the arm that maximizes the payoff for the next pull based on a Bayesian update from prior pulls outperforms the strategy offered in Holland ([41], p. 77). The trade-off between exploitation (pulling the best Bayesian bandit) and exploration (pulling the other bandit in the hopes that the current Bayesian information is misleading) offered in Holland [41] is not optimal for the problem studied. Macready and Wolpert [53] remarked that the algorithms proposed in Holland [41] "are based on the premise that one should engage in exploration, yet for the very problem invoked to justify genetic exploration, the strategy of not exploring at all performs better than n^* exploring algorithms. . .."

The ramifications of this condition are important: One of the keystones that has differentiated genetic algorithms from other forms of evolutionary computation has been the notion of optimal schema processing. Until recently, this was believed to have a firm theoretical footing. It now appears, however, that the basis for claiming that the n^* sampling offered in Ref. [41] is optimal for minimizing expected losses to competing schemata is lacking. In turn, neither is the use of proportional selection optimal for achieving a proper allocation of trials (recall that proportional selection was believed to achieve an exponentially increasing allocation of trials to the observed best schemata, this in line with the mathematical analysis of the two-armed bandit). And as yet another consequence then, the schema theorem under proportional selection, which describes the expected allocation of trials to schemata in a single generation, lacks fundamental importance (cf. [13]). It is simply a formulation for describing one method of performing selection but not a generally *optimal* method.

1.4 CONCLUSIONS

Although the history of evolutionary computation dates back to the 1950s and 1960s (Fogel [3]), only within the past decade have evolutionary algorithms become practicable for solving real-world problems on desktop computers (Bäck et al. [12]). As computers continue to deliver accelerated performance, these applications will only become more routine. The flexibility of evolutionary algorithms to address general optimization problems using virtually any reasonable representation and performance index, with variation operators that can be tailored for the problem at hand and selection mechanisms tuned for the appropriate level of stringency, gives these techniques an advantage over classic numerical optimization procedures. Moreover, the two-step procedure to self-adapt parameters that control the evolutionary search frees the human operator from having to handcraft solutions, which would often be time consuming or simply infeasible. Evolutionary algorithms offer a set of procedures that may be usefully applied to problems that have resisted solution by common techniques and can be hybridized with such techniques when such combinations

appear beneficial. Moreover, the notion of modeling natural systems can be brought to fruition in other forms, including particle swarm methods as indicated here, as well as ant colony optimization and other techniques not mentioned. If this brief introduction serves to spur the imagination of the reader into finding new ways to model natural systems, particularly as they may be applied to problems in power systems, it will have succeeded immensely.

ACKNOWLEDGMENTS

The author would like to thank the editors for inviting this presentation, IEEE/Wiley for permissions to reprint sections of the author's prior publications, and T. Bäck for permission to reprint his figure from Bäck [4].

REFERENCES

1. Fogel DB. Evolutionary computation: Toward a new philosophy of machine intelligence. 2nd ed. Piscataway, NJ: IEEE Press; 2000.
2. Fogel DB. What is evolutionary computation? IEEE Spectrum 2000; February:26–32.
3. Fogel DB, ed. Evolutionary computation: The fossil record. Piscataway, NJ: IEEE Press; 1998.
4. Bäck T. Evolutionary algorithms in theory and practice. New York: Oxford University Press; 1996.
5. Michalewicz Z. Genetic algorithms + data structures = evolution programs. 3rd ed. Berlin: Springer; 1996.
6. Fogel DB. Asymptotic convergence properties of genetic algorithms and evolutionary programming: analysis and experiments. Cybern Syst 1994; 25:389–407.
7. Rudolph G. Convergence analysis of canonical genetic algorithms. IEEE Trans Neural Networks 1994; 5:96–101.
8. Wright S. The roles of mutation, inbreeding, crossbreeding, and selection in evolution. Proc. 6th Int. Cong. Genetics. Vol. 1. Genetic Society of America, Ithaca. 1932. p. 356–366.
9. Atmar W. The inevitability of evolutionary invention. 1979 (unpublished manuscript).
10. Raven PH, Johnson GB. Biology. St. Louis: Times Mirror; 1986.
11. Fogel DB, Ghozeil A. Using fitness distributions to design more efficient evolutionary computations. Proc. of 1996 IEEE Conf. on Evol. Comp. Keynote Lecture. New York: IEEE Press; 1996. p. 11–19.
12. Bäck T, Fogel DB, Michalewicz Z. eds. Handbook of evolutionary computation. New York: Oxford University Press; 1997.
13. Goldberg DE. Genetic algorithms in search, optimization and machine learning. Reading, MA: Addison-Wesley; 1989.
14. Davis L, ed. Handbook of genetic algorithms. New York: Van Nostrand Reinhold; 1991.

15. Michalewicz Z. Genetic algorithms + data structures = evolution programs. Berlin: Springer; 1992.
16. Koza JR. Genetic programming. Cambridge, MA: MIT Press; 1992.
17. Fogel DB, Atmar JW. Comparing genetic operators with Gaussian mutations in simulated evolutionary processes using linear systems. Biol Cybern 1990; 63:111–114.
18. Bäck T, Schwefel H-P. An overview of evolutionary algorithms for parameter optimization. Evol Comp 1993; 1:1–24.
19. Fogel DB, Stayton LC. On the effectiveness of crossover in simulated evolutionary optimization. BioSystems 1994; 32:171–182.
20. Wolpert DH, Macready WG. No free lunch theorems for optimization. IEEE Trans Evol Comp 1997; 1:67–82.
21. Fogel DB, Ghozeil A. A note on representations and variation operators. IEEE Trans Evol Comp 1997; 1:159–161.
22. Schwefel H-P. Evolution and optimum seeking. New York: John Wiley & Sons; 1995.
23. Gehlhaar DK, Verkhivker GM, Rejto PA, Sherman CJ, Fogel DB, Fogel LJ, Freer ST. Molecular recognition of the inhibitor AG-1343 by HIV-1 protease: Conformationally flexible docking by evolutionary programming. Chem Biol 1995; 2:317–324.
24. Harp SA, Samad T, Guha A. Towards the genetic synthesis of neural networks. In: Schaffer JD, ed. Proc. of the 3rd Intern. Conf. on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann; 1989. p. 360–369.
25. Fogel DB, Fogel LJ. Using evolutionary programming to schedule tasks on a suite of heterogeneous computers. Comp Oper Res 1996; 23:527–534.
26. Angeline PJ, Saunders GM, Pollack JB. An evolutionary algorithm that constructs recurrent neural networks. IEEE Trans Neural Networks 1994; 5:54–65.
27. Haffner SB, Sebald AV. Computer-aided design of fuzzy HVAC controllers using evolutionary programming. In: Fogel DB, Atmar W, eds. Proc. of the 2nd Ann. Conf. on Evolutionary Programming. La Jolla, CA: Evolutionary Programming Society; 1993. p. 98–107.
28. Wilson SW. Classifier fitness based on accuracy. Evol Comp 1995; 3:149–175.
29. Angeline PJ, Fogel DB. An evolutionary program for the identification of dynamical systems. In: Rogers SK, Rock D, eds. Aerosence 97, Symp. on Neural Networks. Vol. 3077. Orlando, FL: SPIE 1997. p. 409–417.
30. Fogel DB, Wasson EC, Boughton EM, Porto VW. A step toward computer-assisted mammography using evolutionary programming and neural networks. Cancer Lett 1997; 119:93–97.
31. Wieland AP. Evolving controls for unstable systems. In: Touretzky DS, Elman JL, Sejnowski TJ, Hinton GE, eds. Connectionist models: Proceedings of the 1990 Summer School. San Mateo, CA: Morgan Kaufmann; 1990. p. 91–102.
32. Saravanan N, Fogel DB. Evolving neurocontrollers using evolutionary programming. In: IEEE Conf. on Evol. Comp. Vol. 1. Piscataway, NJ: IEEE Press; 1994. p. 217–222.
33. Fogel DB. A 'correction' to some cart-pole experiments. In: Fogel LJ, Angeline PJ, Bäck T, eds. Evolutionary programming VI. Cambridge, MA: MIT Press; 1996. p. 67–71.
34. Reed J, Toombs R, Barricelli NA. Simulation of biological evolution and machine learning. J Theor Biol 1967; 17:319–342.

35. Rosenberg R. Simulation of genetic populations with biochemical properties. Ph.D. dissertation, University of Michigan, Ann Arbor. 1967.
36. Angeline PJ, Fogel DB, Fogel LJ. A comparison of self-adaptation methods for finite state machines in a dynamic environment. In: Fogel LJ, Angeline PJ, Bäck T, eds. Evolutionary programming V. Cambridge, MA: MIT Press; 1996. p. 441–449.
37. Chellapilla K, Fogel DB. Exploring self-adaptive methods to improve the efficiency of generating approximate solutions to traveling salesman problems using evolutionary programming. In: Angeline PJ, Reynolds RG, McDonnell JR, Eberhart R, eds. Evolutionary Programming VI. Berlin: Springer; 1997. p. 361–371.
38. Fogel LJ, Owens AJ, Walsh MJ. Artificial intelligence through simulated evolution. New York: John Wiley & Sons; 1996.
39. Fraser AS. Simulation of genetic systems by automatic digital computers. I. Introduction. Australian J Biol Sci 1957; 10:484–491.
40. Bremermann HJ. Optimization through evolution and recombination. In: Yovits MC, Jacobi GT, Goldstein GD, eds. Self-organizing systems—1962. Washington, DC: Spartan Books; 1962. p. 93–106.
41. Holland JH. Adaptation in natural and artificial systems. Ann Arbor, MI: Univeristy of Michigan Press. 1975.
42. Rechenberg I. Cybernetic solution path of an experimental problem. Royal Aircraft Establishment; Library Translation 1122, 1965.
43. English TM. Evaluation of evolutionary and genetic optimizers: no free lunch. In: Fogel LJ, Angeline PJ, Bäck T, eds. Evolutionary programming V: Proc. of the 5th Annual Conference on Evolutionary Programming. Cambridge, MA: MIT Press; 1996. p. 163–169.
44. De Jong KA, Spears WM, Gordon DF. Using Markov chains to analyze GAFOs. In: Whitley LD, Vose MD, eds. Foundations of genetic algorithms 3. San Mateo, CA: Morgan Kaufmann; 1995. p. 115–137.
45. Antonisse J. A new interpretation of schema notation that overturns the binary encoding constraint. In: Schaffer JD, ed. Proceedings of the Third Int. Conf. on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann; 1989. p. 86–91.
46. Battle DL, Vose MD. Isomorphisms of genetic algorithms. Artificial Intelligence 1993; 60:155–165.
47. Vose MD, Liepins GE. Schema disruption. In: Belew RK, Booker LB, eds. Proceedings of the Fourth International Conference on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann; 1991. p. 237–240.
48. Radcliffe NJ. Non-linear genetic representations. In: Männer R, Manderick B, eds. Parallel problem solving from nature. 2. Amsterdam: North-Holland; 1992. p. 259–268.
49. Fogel DB, Ghozeil A. Schema processing under proportional selection in the presence of random effects. IEEE Trans Evol Computat 1997; 1(4): p. 290–293.
50. Rana S, Whitley LD, Cogswell R. Searching in the presence of noise. In: Voigt H-M, Ebeling W, Rechenberg I, Schwefel HP, eds. Parallel problem solving from nature—PPSN IV. Berlin: Springer; 1996. p. 198–207.
51. Gillespie H. Natural selection for variances in offspring numbers: a new evolutionary principle. Am Naturalist 1977; 111:1010–1014.

52. Rudolph G. Reflections on bandit problems and selection methods in uncertain environments. In Bäck T, ed. Proc. of 7th Intern. Conf. on Genetic Algorithms, San Francisco: Morgan Kaufmann; 1997. p. 166–173.
53. Macready WG, Wolpert DH. Bandit problems and the exploration/exploitation tradeoff. IEEE Trans Evol Computat 1998; 2(1):2–22.