Chapter 1
Evolutionary Computation

1.1 Introduction

Charles Darwinian evolution in 1859 is intrinsically a so bust search and optimization mechanism. Darwin's principle "Survival of the fittest" captured the popular imagination. This principle can be used as a starting point in introducing evolutionary computation. Evolved biota demonstrates optimized complex behavior at each level: the cell, the organ, the individual and the population. Biological species have solved the problems of chaos, chance, nonlinear interactivities and temporality. These problems proved to be in equivalence with the classic methods of optimization. The evolutionary concept can be applied to problems where heuristic solutions are not present or which leads to unsatisfactory results. As a result, evolutionary algorithms are of recent interest, particularly for practical problems solving.

The theory of natural selection proposes that the plants and animals that exist today are the result of millions of years of adaptation to the demands of the environment. At any given time, a number of different organisms may co-exist and compete for the same resources in an ecosystem. The organisms that are most capable of acquiring resources and successfully procreating are the ones whose descendants will tend to be numerous in the future. Organisms that are less capable, for whatever reason, will tend to have few or no descendants in the future. The former are said to be more *fit* than the latter, and the distinguishing characteristics that caused the former to be fit are said to be *selected for* over the characteristics of the latter. Over time, the entire population of the ecosystem is said to *evolve* to contain organisms that, on average, are more fit than those of previous generations of the population because they exhibit more of those characteristics that tend to promote survival.

Evolutionary computation (EC) techniques abstract these evolutionary principles into algorithms that may be used to search for optimal solutions to a problem. In a search algorithm, a number of possible solutions to a problem are available and the task is to find the best solution possible in a fixed amount of time. For a search space with only a small number of possible solutions, all the solutions can be examined in a reasonable amount of time and the optimal one found. This *exhaustive search*, however, quickly becomes impractical as the search space grows in size. Traditional search algorithms randomly sample (e.g., *random walk*) or heuristically sample (e.g., *gradient descent*) the search space one solution at a time in the hopes

of finding the optimal solution. The key aspect distinguishing an evolutionary search algorithm from such traditional algorithms is that it is *population-based*. Through the adaptation of successive generations of a large number of individuals, an evolutionary algorithm performs an efficient directed search. Evolutionary search is generally better than random search and is not susceptible to the hill-climbing behaviors of gradient-based search.

Evolutionary computing began by lifting ideas from biological evolutionary theory into computer science, and continues to look toward new biological research findings for inspiration. However, an over enthusiastic "biology envy" can only be to the detriment of both disciplines by masking the broader potential for two-way intellectual traffic of shared insights and analogizing from one another. Three fundamental features of biological evolution illustrate the range of potential intellectual flow between the two communities: particulate genes carry some subtle consequences for biological evolution that have not yet translated mainstream EC; the adaptive properties of the genetic code illustrate how both communities can contribute to a common understanding of appropriate evolutionary abstractions; finally, EC exploration of representational language seems pre-adapted to help biologists understand why life evolved a dichotomy of genotype and phenotype.

1.2 The Historical Development of EC

In the case of evolutionary computation, there are four historical paradigms that have served as the basis for much of the activity of the field: genetic algorithms (Holland, 1975), genetic programming (Koza, 1992, 1994), evolutionary strategies (Recheuberg, 1973), and evolutionary programming (Forgel et al., 1966). The basic differences between the paradigms lie in the nature of the representation schemes, the reproduction operators and selection methods.

1.2.1 Genetic Algorithms

The most popular technique in evolutionary computation research has been the *genetic algorithm*. In the traditional genetic algorithm, the representation used is a *fixed-length bit string*. Each position in the string is assumed to represent a particular feature of an individual, and the value stored in that position represents how that feature is expressed in the solution. Usually, the string is "evaluated as a collection of *structural* features of a solution that have little or no interactions". The analogy may be drawn directly to genes in biological organisms. Each gene represents an entity that is structurally independent of other genes.

The main reproduction operator used is *bit-string crossover*, in which two strings are used as parents and new individuals are formed by swapping a sub-sequence between the two strings (see Fig. 1.1). Another popular operator is *bit-flipping mutation*, in which a single bit in the string is flipped to form a new offspring string

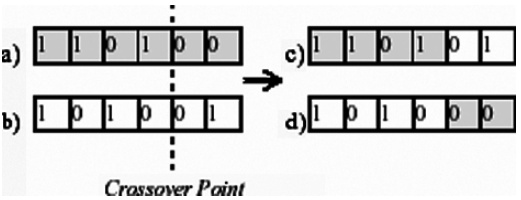

Fig. 1.1 Bit-string crossover of parents a & b to form offspring c & d

(see Fig. 1.2). A variety of other operators have also been developed, but are used less frequently (e.g., *inversion*, in which a subsequence in the bit string is reversed). A primary distinction that may be made between the various operators is whether or not they introduce any new information into the population. Crossover, for example, does not while mutation does. All operators are also constrained to manipulate the string in a manner consistent with the structural interpretation of genes. For example, two genes at the same location on two strings may be swapped between parents, but not combined based on their values. Traditionally, individuals are selected to be parents *probabilistically* based upon their fitness values, and the offspring that are created replace the parents. For example, if N parents are selected, then N offspring are generated which replace the parents in the next generation.

1.2.2 Genetic Programming

An increasingly popular technique is that of *genetic programming*. In a standard genetic program, the representation used is a variable-sized tree of functions and values. Each leaf in the tree is a label from an available set of value labels. Each internal node in the tree is label from an available set of function labels.

The entire tree corresponds to a single function that may be evaluated. Typically, the tree is evaluated in a leftmost depth-first manner. A leaf is evaluated as the corresponding value. A function is evaluated using arguments that is the result of the evaluation of its children. Genetic algorithms and genetic programming are similar in most other respects, except that the reproduction operators are tailored to a tree representation. The most commonly used operator is *subtree crossover*, in which an entire subtree is swapped between two parents (see Fig. 1.3). In a standard genetic program, all values and functions are assumed to return the same type, although functions may vary in the number of arguments they take. This *closure* principle (Koza, 1994) allows any subtree to be considered structurally on par with any other subtree, and ensures that operators such as sub-tree crossover will always produce legal offspring.

Fig. 1.2 Bit-flipping mutation of parent a to form offspring b

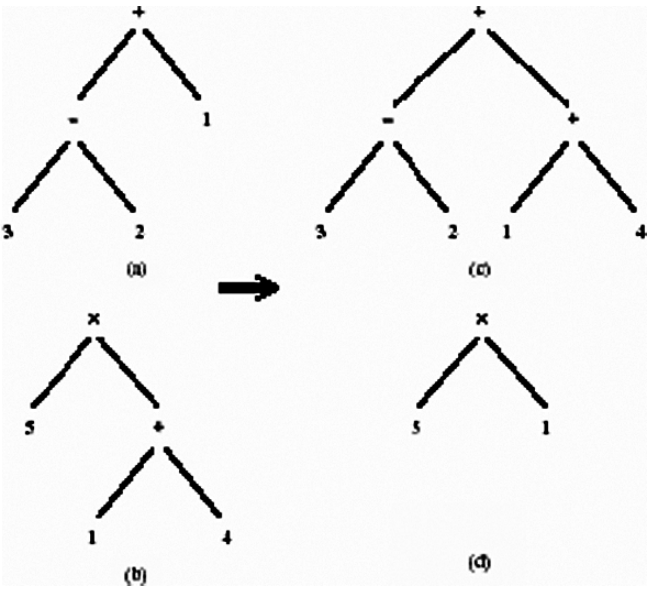

Fig. 1.3 Subtree crossover of parents a & b to form offspring c & d

1.2.3 Evolutionary Strategies

In evolutionary strategies, the representation used is a fixed-length real-valued vector. As with the bitstrings of genetic algorithms, each position in the vector corresponds to a feature of the individual. However, the features are considered to be behavioral rather than structural. "Consequently, arbitrary non-linear interactions between features during evaluation are expected which forces a more holistic approach to evolving solutions" (Angeline, 1996).

The main reproduction operator in evolutionary strategies is *Gaussian mutation*, in which a random value from a Gaussian distribution is added to each element of an individual's vector to create a new offspring (see Fig. 1.4). Another operator that is used is *intermediate recombination*, in which the vectors of two parents are averaged together, element by element, to form a new offspring (see Fig. 1.5). The effects of these operators reflect the behavioral as opposed to structural interpretation of the representation since knowledge of the values of vector elements is used to derive new vector elements.

The selection of parents to form offspring is less constrained than it is in genetic algorithms and genetic programming. For instance, due to the nature of the representation, it is easy to average vectors from many individuals to form a single offspring. In a typical evolutionary strategy, N parents are selected uniformly randomly

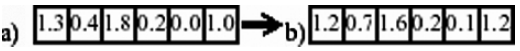

Fig. 1.4 Gaussian mutation of parent a to form offspring b

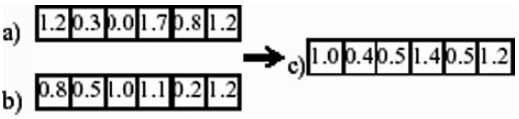

Fig. 1.5 Intermediate recombination of parents a & b to form offspring c

(i.e., not based upon fitness), more than N offspring are generated through the use of recombination, and then N survivors are selected deterministically. The survivors are chosen either from the best N offspring (i.e., no parents survive) or from the best N parents and offspring.

1.2.4 Evolutionary Programming

Evolutionary programming took the idea of representing individuals' phenotypic ally as finite state machines capable of responding to environmental stimuli and developing operators for effecting structural and behavioral change over time. This idea was applied to a wide range of problems including prediction problems, optimization and machine learning.

The above characterizations, leads one to the following observations. GA practitioners are seldom constrained to fixed-length binary implementations. GP enables the use of variable sized tree of functions and values. ES practitioners have incorporated recombination operators into their systems. EP is used for the evolution of finite state machines.

The representations used in evolutionary programming are typically tailored to the problem domain. One representation commonly used is a fixed-length real-valued vector. The primary difference between evolutionary programming and the previous approaches is that no exchange of material between individuals in the population is made. Thus, only mutation operators are used. For real-valued vector representations, evolutionary programming is very similar to evolutionary strategies without recombination.

A typical selection method is to select all the individuals in the population to be the N parents, to mutate each parent to form N offspring, and to probabilistically select, based upon fitness, N survivors from the total 2N individuals to form the next generation.

1.3 Features of Evolutionary Computation

In an evolutionary algorithm, a *representation scheme* is chosen by the researcher to define the set of solutions that form the search space for the algorithm. A number of individual solutions are created to form an *initial population*. The following steps are then repeated iteratively until a solution has been found which satisfies a pre-defined *termination criterion*. Each individual is evaluated using a *fitness function* that is specific to the problem being solved. Based upon their fitness values,

a number of individuals are chosen to be *parents*. New individuals, or *offspring*, are produced from those parents using *reproduction operators*. The fitness values of those offspring are determined. Finally, survivors are selected from the old population and the offspring to form the new population of the next *generation*.

The mechanisms determining which and how many parents to select, how many offspring to create, and which individuals will survive into the next generation together represent a *selection method*. Many different selection methods have been proposed in the literature, and they vary in complexity. Typically, though, most selection methods ensure that the population of each generation is the same size.

EC techniques continue to grow in complexity and desirability, as biological research continues to change our perception of the evolutionary process.

In this context, we introduce three fundamental features of biological evolution:

1. particulate genes and population genetics
2. the adaptive genetic code
3. the dichotomy of genotype and phenotype

Each phenomenon is chosen to represent a different point in the spectrum of possible relationships between computing and biological evolutionary theory. The first is chosen to ask whether current EC has fully transferred the basics of biological evolution. The second demonstrates how both biological and computational evolutionary theorists can contribute to common understanding of evolutionary abstractions. The third is chosen to illustrate a question of biological evolution that EC seems better suited to tackle than biology.

1.3.1 Particulate Genes and Population Genetics

Mainstream thinking of the time viewed the genetic essence of phenotype as a liquid that blended whenever male and female met to reproduce. It took the world's first professor of engineering, Fleming Jenkin (1867), to point out the mathematical consequence of blending inheritance: a novel advantageous mutation arising in a sexually reproducing organism would dilute itself out of existence during the early stages of its spread through any population comprising more than a few individuals. This is a simple consequence of biparental inheritance. Mendels' theory of particulate genes (Mendel, 1866) replaced this flawed, analogue concept of blending inheritance with a digital system in which the advantageous version (allele) of a gene is either present or absent and biparental inheritance produces diploidy. Thus natural selection merely alters the proportions of alleles in a population, and an advantageous mutation can be selected into fixation (presence within 100% of individuals) without any loss in its fitness. Though much has been written about the Neo-Darwinian Synthesis that ensured from combining Mendelian genetics with Darwinian theory, it largely amounts to biologists' gradual acceptance that the particulate nature of genes alone provided a solid foundation to build detailed, quantitative predictions about evolution.

Indeed, decision of mathematical models of genes in populations as "bean bag genetics" overlooks the scope of logical deductions that follow from particulate genetics. They extend far beyond testable explanations for adaptive phenomena and into deeper, abstract concepts of biological evolution. For example, particulate genes introduce stochasticity into evolution. Because genes are either present or absent from any given genome, the genetic makeup of each new individual in a sexually reproducing population is a probabilistic outcome of which particular alleles it inherits from each parent. Unless offspring are infinite in number, their allele frequencies will not accurately mirror those of the parental generation, but instead will show some sampling error (genetic drift).

The magnitude of this sampling error is inversely proportional to the size of a population. Wright (1932) noted that because real populations fluctuate in size, temporary reductions can briefly relax selection, potentially allowing gene pools to diffuse far enough away from local optima to find new destinations when population size recovers and selection reasserts itself. In effect, particulate genes in finite populations improve the evolutionary heuristic from a simple hill climbing algorithm to something closer to simulated annealing under a fluctuating temperature. One final property of particulate genes operating in sexual populations is worthy of mention. In the large populations where natural selection works most effectively, any novel advantageous mutation that arises will only reach fixation over the course of multiple generations. During this spread, recombination and diploidy together ensure that the allele will temporarily find itself in many different genetic contexts. Classical population genetics (e.g., Fisher, 1930) and experimental EC systems (e.g., O'Reilly, 1999) have focused on whether and how this context promotes selective pressure for gene linkage into "co-adapted gene complexes". A simpler observation is that a novel, advantageous allele's potential for negative epistatic effects is integral to its micro-evolutionary success. Probability will favor the fixation of alleles that are good "team players" (i.e., reliably imbue their advantage regardless of genetic background. Many mainstream EC methods simplify the population genetics of new mutations (e.g., into tournaments), to expedite the adaptive process. This preserves non-blending inheritance and even genetic drift, but it is not clear that it incorporates population genetics' implicit filter for "prima donna" alleles that only offer their adaptive advantage when their genetic context is just so. Does this basic difference between biology and EC contribute anything to our understanding of why recombination seems to play such different roles in the two systems?

1.3.2 The Adaptive Code Book

Molecular biology's Central Dogma connects genes to phenotype by stating that DNA is transcribed into RNA, which is then translated into protein.

The terms transcription and translation are quite literal: RNA is a chemical sister language to DNA. Both are polymers formed from an alphabet of four chemical letters (nucleotides), and transcription is nothing more than a process of complementing DNA, letter by letter, into RNA. It is the next step, translation

that profoundly influences biological evolution. Proteins are also linear polymers of chemical letters, but they are drawn from a qualitatively different alphabet (amino acids) comprising 20 elements. Clearly no one-to-one mapping could produce a genetic code for translating nucleotides unambiguously into amino acids, and by 1966 it was known that the combinatorial set of possible nucleotide triplets forms a dictionary of "codons" that each translate into a single amino acid meaning. The initial surprise for evolutionary theory was to discover that something as fundamental as the code-book for life would exhibit a high degree of redundancy (an alphabet of 4 RNA letters permits $4\times4\times4 = 64$ possible codons that map to one of only 20 amino acid meanings). Early interpretation fuelled arguments for Non-Darwinian evolution: genetic variations that make no difference to the protein they encode must be invisible to selection and therefore governed solely by drift. More recently, both computing and biological evolutionary theory have started to place this coding neutrality in the bigger picture of the adaptive heuristic. Essentially, findings appear to mirror Wright's early arguments on the importance of genetic drift: redundancy in the code adds selectively neutral dimensions to the fitness landscape that renders adaptive algorithms more effective by increasing the connectedness of local optima.

At present, an analogous reinterpretation is underway for a different adaptive feature of the genetic code: the observation that biochemically similar amino acids are assigned to codons that differ by only a single nucleotide. Early speculations that natural selection organized the genetic code so as to minimize the phenotypic impact of mutations have gained considerable evidential support as computer simulation enables exploration of theoretical codes that nature passed over. However, it seems likely that once again this phenomenon has more subtle effects in the broader context of the adaptive heuristic. An "error minimizing code" may in fact maximize the probability that a random effects on both traits defines a circle of radius around the organism.

The probability that this mutation will improve fitness (i.e., that the organism will move within the white area) is inversely proportional to its magnitude, mutation produces an increase in fitness according to Geometric Theory of gradualism (Fig. 1.6). Preliminary tests for this phenomenon reveal an even simpler influence: the error minimizing code smoothes the fitness landscape where a random genetic code would render it rugged. By clustering biochemically similar amino acids within mutational reach of one another it ensures that any selection towards a specific amino acid property (e.g., hydrophobicity) will be towards an interconnected region of the fitness landscape rather than to an isolated local optimum.

1.3.3 The Genotype/Phenotype Dichotomy

Implicit to the concept of an adaptive genetic code is a deeper question that remains largely unanswered by biology: why does all known life use two qualitatively different polymers, nucleic acids and proteins, with the associated need for translation? Current theories for the origin of this dichotomy focus on the discovery that RNA can act both as a genetic storage medium, and as a catalytic molecule. Within the

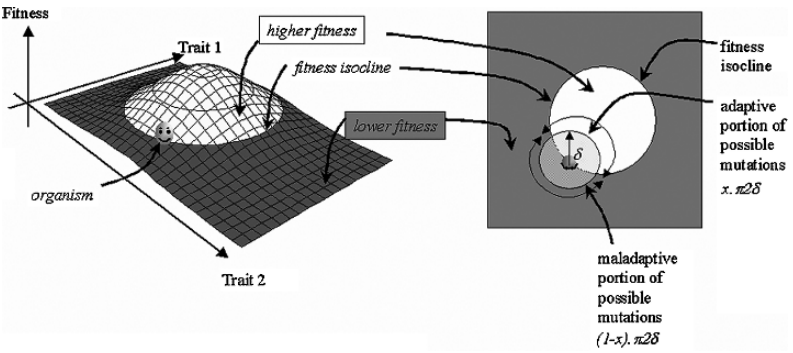

Fig. 1.6 The fitness landscape for an organism of 2 phenotypic traits: (**a**) for any organism, we may define an isocline that connects all trait combinations of equal fitness; (**b**) (the fitness landscape from above): a random mutation of magnitude that has tradeoff

most highly conserved core of metabolism, all known organisms are found to use RNA molecules in roles we normally attribute to proteins (White, 1976).

However, the answer to *how* the dichotomy evolved has largely eclipsed the question of *why* RNA evolved a qualitatively different representation for phenotype. A typical biological answer would be that the larger alphabet size of amino acids unleashed a greater catalytic diversity for the replicators, with an associated increase in metabolic sophistication that optimized self-replication. Interestingly, we know that nucleic acids are not limited to the 4 chemical letters we see today: natural metabolically active RNA's utilize a vast repertoire of posttranscriptional modifications and synthetic chemistry has demonstrated that multiple additional nucleotide letters can be added to the genetic alphabet even with today's cellular machinery. Furthermore, an increasing body of indirect evidence suggests that the protein alphabet itself underwent exactly the sort of evolutionary expansion early in life's history.

Given the ubiquity of nucleic acid genotype and protein phenotype within life, biology is hard-pressed to assess the significance of evolving this "representational language". The choice of phrase is deliberate: clearly the EC community is far ahead of biology in formalizing the concept of representational language, and exploring what it means. Biology will gain when evolutionary programmers place our system within their findings, illustrating the potential for biological inspiration *from* EC.

1.4 Advantages of Evolutionary Computation

Evolutionary computation, describes the field of investigation that concerns all evolutionary algorithms and offers practical advantages to several optimization problems. The advantages include the simplicity of the approach, its robust response to changing circumstances, and its flexibility and so on. This section briefs some of

these advantages and offers suggestions in designing evolutionary algorithms for real-world problem solving.

1.4.1 Conceptual Simplicity

A key advantage of evolutionary computation is that it is conceptually simple. Figure 1.7 shows a flowchart of an evolutionary algorithm applied for function optimization. The algorithm consists of initialization, iterative variation and selection in light of a performance index. In particular, no gradient information needs to be presented to the algorithm. Over iterations of random variation and selection, the population can be made to converge to optimal solutions. The effectiveness of an evolutionary algorithm depends on the variation and selection operators as applied to a chosen representation and initialization.

1.4.2 Broad Applicability

Evolutionary algorithms can be applied to any problems that can be formulated as function optimization problems. To solve these problems, it requires a data structure to represent solutions, to evaluate solutions from old solutions. Representations can be chosen by human designer based on his intuition. Representation should allow for variation operators that maintain a behavioral link between parent and offspring. Small changes in structure of parent will lead to small changes in offspring, and similarly large changes in parent will lead to drastic alterations in offspring. In this case, evolutionary algorithms are developed, so that they are tuned in self adaptive

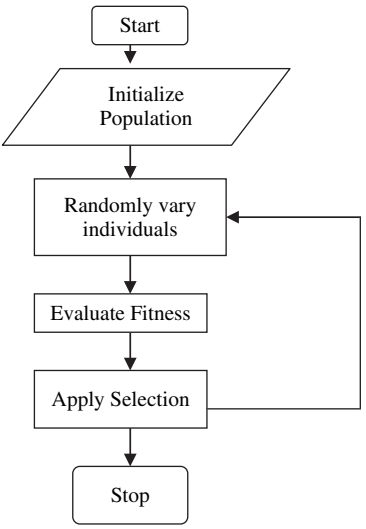

Fig. 1.7 Flowchart of an evolutionary algorithm

manner. This makes the evolutionary computation to be applied to broad areas which includes, discrete combinatorial problems, mixed-integer problems and so on.

1.4.3 Hybridization with Other Methods

Evolutionary algorithms can be combined with more traditional optimization techniques. This is as simple as the use of a conjugate-gradient minimization used after primary search with an evolutionary algorithm. It may also involve simultaneous application of algorithms like the use of evolutionary search for the structure of a model coupled with gradient search for parameter values. Further, evolutionary computation can be used to optimize the performance of neural networks, fuzzy systems, production systems, wireless systems and other program structures.

1.4.4 Parallelism

Evolution is a highly parallel process. When distributed processing computers become more popular are readily available, there will be increased potential for applying evolutionary computation to more complex problems. Generally the individual solutions are evaluated independently of the evaluations assigned to competing solutions. The evaluation of each solution can be handled in parallel and selection only requires some serial operation. In effect, the running time required for an application may be inversely proportional to the number of processors. Also, the current computing machines provide sufficient computational speed to generate solutions to difficult problems in reasonable time.

1.4.5 Robust to Dynamic Changes

Traditional methods of optimization are not robust to dynamic changes in the environment and they require a complete restart for providing a solution. In contrary, evolutionary computation can be used to adapt solutions to changing circumstances. The generated population of evolved solutions provides a basis for further improvement and in many cases, it is not necessary to reinitialize the population at random. This method of adapting in the face of a dynamic environment is a key advantage. For example, Wielaud (1990) applied genetic algorithm to evolve recurrent neural networks to control a cart-pole system consisting of two poles as shown in Fig. 1.2.

In the above Fig. 1.8, 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 control available here is the force, with which pull and push action on the cart is performed. The difficulty here is the similarity in pole lengths. Few researchers used evolutionary algorithms to optimize neural networks to control this plant for different pole lengths.

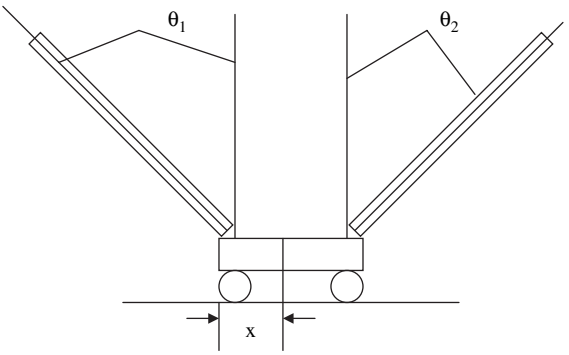

Fig. 1.8 A cart with two poles

1.4.6 Solves Problems that have no Solutions

The advantage of evolutionary algorithms includes its ability to address problems for which there is no human expertise. Even though human expertise should be used when it is needed and available; it often proves less adequate for automated problem-solving routines. Certain problems exist with expert system: the experts may not agree, may not be qualified, may not be self-consistent or may simply cause error. Artificial intelligence may be applied to several difficult problems requiring high computational speed, but they cannot compete with the human intelligence, Fogel (1995) declared artificial intelligence as "They solve problems, but they do not solve the problem of how to solve problems." In contrast, evolutionary computation provides a method for solving the problem of how to solve problems.

1.5 Applications of Evolutionary Computation

Evolutionary computation techniques have drawn much attention as optimization methods in the last two decades. From the optimization point of view, the main advantage of evolutionary computation techniques is that they do not have much mathematical requirements about the optimization problems. All they need is an evaluation of the objective function. As a result, they are applied to non-linear problems, defined on discrete, continuous or mixed search spaces, constrained or unconstrained.

The applications of evolutionary computation include the following fields:

- Medicine (for example in breast cancer detection).
- Engineering application (including electrical, mechanical, civil, production, aeronautical and robotics).
- Traveling salesman problem.
- Machine intelligence.
- Expert system

- Network design and routing
- Wired and wireless communication networks and so on.

Many activities involve unstructured, real life problems that are difficult to model, since they require several unusual factors. Certain engineering problems are complex in nature: job shop scheduling problems, timetabling, traveling salesman or facility layout problems. For all these applications, evolutionary computation provides a near-optimal solution at the end of an optimization run. Evolutionary algorithms are thus made efficient because they are flexible, and relatively easy to hybridize with domain-dependent heuristics.

1.6 Summary

The basics of evolutionary computation with its historical development were discussed in this chapter. Although the history of evolutionary computation dates back to the 1950s and 1960s, only within the last decade have evolutionary algorithms became practicable for solving real-world problems on desktop computers. The three basic features of the biological evolutionary algorithms were also discussed. For practical genes, we ask whether Evolutionary computation can gain from biology by considering the detailed dynamics by which an advantageous allele invades a wild-type population. The adaptive genetic code illustrates how Evolutionary computation and biological evolutionary research can contribute to a common understanding of general evolutionary dynamic. For the dichotomy of genotype and phenotype, biology is hard-pressed to assess the significance of representational language. The various advantages and applications of evolutionary computation are also discussed in this chapter.

Review Questions

1. Define Evolutionary computation.
2. Briefly describe the historical developments of evolutionary computation.
3. State three fundamental features of biological evolutionary computation.
4. Draw a flowchart and explain an evolutionary algorithm.
5. Define genotype and phenotype.
6. Mention the various advantages of evolutionary computation.
7. List a few applications of evolutionary computation.
8. How are evolutionary computational methods hybridized with other methods?
9. Differentiate: Genetic algorithm and Genetic Programming.
10. Give a description of how evolutionary computation is applied to engineering applications.