

Chapter 2

Evolutionary Algorithms and the Control of Systems

2.1 Control of Systems

Control of systems may reasonably be viewed as a branch of cybernetics as defined by Wiener: the “science of control and communication in the animal and the machine”. One of its central subjects is that of control, and particularly feedback control of autonomous robots which is one of our core areas of concern. Indeed, when Wiener gave the new discipline of Cybernetics a name in 1948 he made use of the Greek word for steersman, *κυβερνήτης*; he arrived at this term through the etymology of the word “governor”, a popular term used for the first widely used feedback device (Wiener 1948). Feedback control then is a topic of central importance in our understanding of biological and mechanical systems and in the design of robots and other machinery for the manipulation of our environment.

2.2 Recent Rapid Developments in the Field of Intelligent Robotics

The current rate of progress in the intelligent robotics field is quite astonishing and, if anything, appears to be happening at an ever-accelerating rate. Each new edition of the *IEEE Spectrum Robotics News*, an online publication of the Institute of Electrical and Electronics Engineers, showcases new developments in advanced robotics, many of which in their own way might be described as milestones in the field. For example, as of the time of writing, a robot (quadruped) has just been developed by the company Boston Dynamics (now owned by Google) which can run faster (albeit on a treadmill) than the world’s currently fastest man.

Other recent developments described in this online publication include almost fully autonomous cars capable of navigating safely through urban streets

(Google again), cooperating microrobots capable of building three-dimensional structures (SRI International), flying robots (hexrotor drones) capable of playing a symphony of musical instruments, including drums, bells, and a piano (KMeL Robotics LLC), and a hopping bionic kangaroo (Festo AG & Co. KG). These are, of course, in addition to the recent developments in the specific field of humanoid robotics, which we address in Chap. 4.

2.3 Evolutionary Algorithms

Here we give a synopsis of the main evolutionary algorithms in current use in the evolutionary robotics field. We use the term “evolutionary algorithm” after Hoffmeister and Schwefel (1990), who used the term to cover algorithms which copied some principles from organic evolution and was meant specifically to refer to both the genetic algorithms of Holland (1975) and the evolution strategies of Rechenberg (1973). We use this term to cover the broader spectrum of any algorithms deriving in some fashion from the Darwinian evolutionary process. These include genetic algorithms (GA, Holland 1975), genetic programming (GP, Koza 1992), evolutionary strategies (ES, Rechenberg 1973), evolutionary programming (EP, Fogel et al. 1966), covariance matrix adaptation (CMA, Hansen and Ostermeier 2001), neuroevolution of augmenting technologies (NEAT, Stanley and Miikkulainen 2002), and the nondominated sorting genetic algorithm II (NSGA-II, Deb et al. 2002). As the genetic algorithm is probably the commonest evolutionary algorithm currently in use in the evolutionary robotics field, we devote the most space here to this paradigm. In fact, in our survey in Chap. 6 of the state of the art in the EHR domain, over 50 % of the research applications used some variant of a genetic algorithm as their main, or as an ancillary algorithm.

2.3.1 Genetic Algorithms (GA)

Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics. They combine the notion of the survival of the fittest with a structured but randomised exchange of information between competing solutions. They also efficiently exploit historical information to improve performance over time. Genetic algorithms, or GAs as they are referred to in short, may be viewed as one of a family of algorithms operating around the same principles of natural selection and natural genetics, each with a different mode of implementation and each emphasising a different aspect of the natural process.

Genetic algorithms as proposed by Holland (1975) use three basic operators; these are reproduction, recombination or crossover, and, to a lesser extent,

mutation. One starts with an initial population of structures, each of which encodes a specific solution to the problem at hand. This population is generally, though not necessarily, chosen at random. Each individual structure may take the form of a string of bits, or some other representational mechanism.

Of course, in a binary computer everything translates into bits at the end of the day; however, using, at a higher level, nonbinary representations (e.g., real values) the mutation and the crossover operators will have to operate in a slightly different fashion. Holland’s original work demonstrated the ability of simple bit strings to encode complicated structures and also the power of simple transformations to improve dramatically the performance of these structures given sufficient time.

To deal with each of the basic operators in turn, mutation (random) provides background variation and occasionally introduces beneficial modifications into a structure. Mutation generally just involves changing a bit in the bit string from 0 to 1 or vice versa. Mutation is not assigned the same importance in genetic algorithms as in some of the other evolutionary algorithms, and so the probability of mutation is generally kept low. The crossover operation is generally looked on as the key to the power of the genetic algorithm. Crossover is probably best illustrated by a simple example (Fig. 2.1).

Assume these two strings encode different solutions to a particular problem (x^1, \dots, x^6 and y^1, \dots, y^6 taking either the values 0 or 1). If these two strings are selected for crossover the first thing to do is to select a crossover point or points. This crossover point is generally chosen at random, for a single-point crossover on a bit string of length l an integer position k is selected randomly in the interval $1, \dots, l-1$. We can then create two new strings by swapping over all bits between the positions $k + 1$ and l inclusive.

Returning to our example, assuming a crossover point of 2, Fig. 2.2 shows how the strings (offspring) after crossover would look.

Consider two strings X and Y, each of length six bits:

X =	x ¹	x ²	x ³	x ⁴	x ⁵	x ⁶
Y =	y ¹	y ²	y ³	y ⁴	y ⁵	y ⁶

Fig. 2.1 Two simple chromosomes prior to crossover

X' =	x ¹	x ²	y ³	y ⁴	y ⁵	y ⁶
Y' =	y ¹	y ²	x ³	x ⁴	x ⁵	x ⁶

Fig. 2.2 Chromosomes after crossover

One entire bit string is sometimes known as a chromosome, with individual bits being known as genes. Crossover allows for genetic material to be passed from two chromosomes, which may be termed the parent chromosomes, to create two new chromosomes, the offspring. This process allows the offspring to combine beneficial material from both parents. While some of the other evolutionary algorithms that we will discuss shortly emphasise mutation as the principal genetic operator, crossover is the main operator for genetic algorithms (Holland 1975). Without crossover, for an individual to acquire a beneficial trait requiring two separate mutations, neither of which on its own is beneficial, one of these mutations must happen to the parent, and then the second mutation must happen to one of that parent's offspring. This is an unlikely occurrence as the probability of survival of the first mutation will be low given that it will probably not have any immediate beneficial effect.

The final operator, reproduction, may be viewed simply as a process by which individual strings are copied according to their fitness. Very fit individuals receive a large number of copies; poor individuals possibly receive none. We start off with an initial population of strings, then, by the application of these basic genetic operators we hope to obtain populations increasing in overall utility. Summarising, a basic genetic algorithm to solve a particular problem will have the following components, many of which are shared with the other evolutionary algorithms which we will discuss shortly:

- a method of representing solutions to the problem in chromosomes
- a method of creating an initial population of chromosomes
- an evaluation or fitness function related directly to the problem environment for deciding the reproductive capability of individual chromosomes
- basic genetic operators for generating new solutions
- general parameters for the genetic algorithm such as population size, number of generations, etc.

Figure 2.3 gives an outline of a simple genetic algorithm (SGA) employing the so-called “roulette wheel” selection method, which simply involves the selection of an individual with a probability proportional to its fitness.

We are very conscious of the arguments put forward by Matarić and Cliff (among others) that if the amount of time and energy expended on the design of a GA for a particular problem, including tuning parameter sets and hand-crafting a fitness function for the particular problem domain and performing multiple experiments (necessary, given the inherently stochastic nature of evolutionary algorithms), exceeds the time required to hand-produce a particular control algorithm (or body design), then the use of evolutionary algorithms may not, indeed, be appropriate (Matarić and Cliff 1996).

With this caveat in mind, the author has been struck by the power of the simple GA to eke out solutions in complex problem domains. It may well be that using, for example, real-value encoding, tournament selection or some other selection procedure, or adaptive mutation or crossover operators will result in a more efficient search, but the power and beauty of the simple GA is that it works sufficiently well

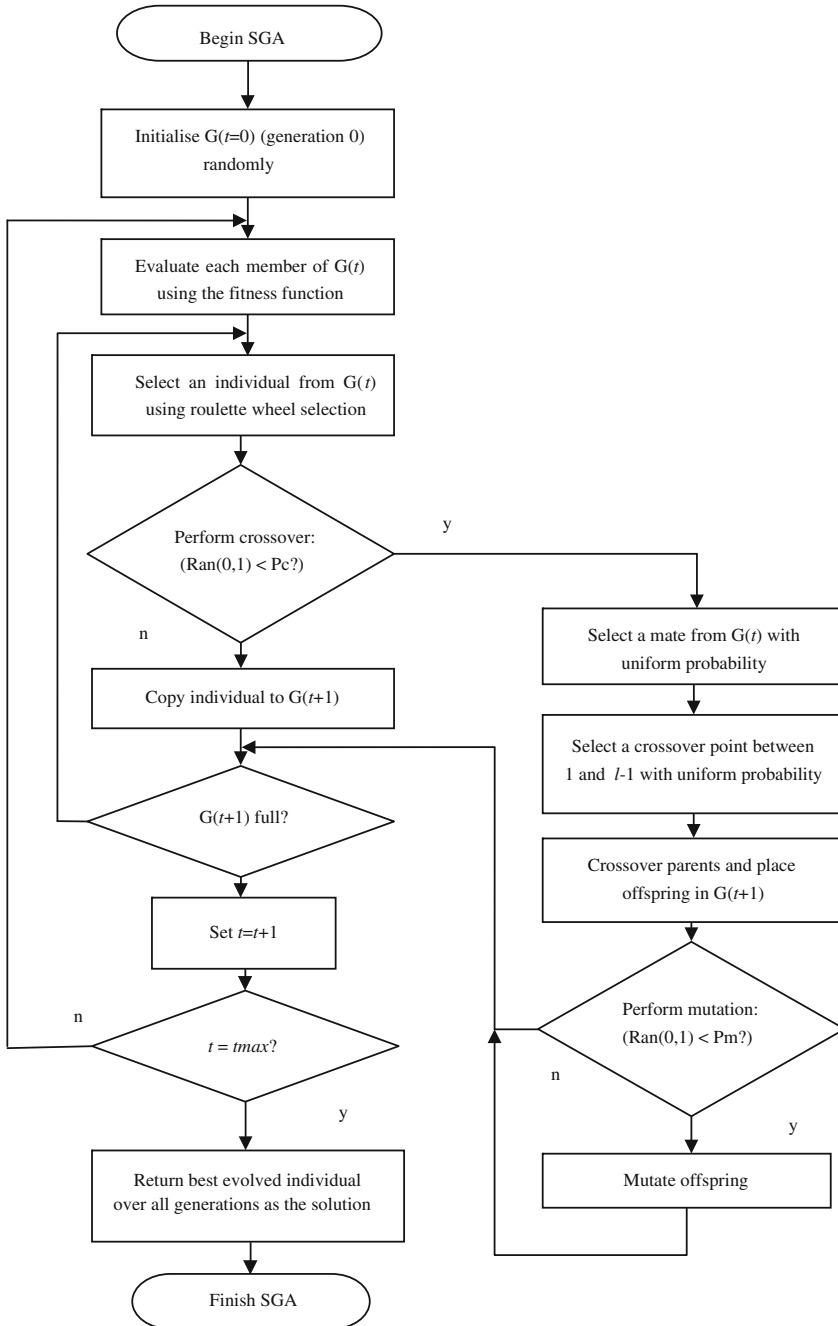


Fig. 2.3 Simple genetic algorithm (SGA) flowchart

in many circumstances. In many difficult problem domains, as in nature, we may not wish to find optimal solutions; indeed we may find it difficult to define exactly what it is we mean by optimal. In nature the core issues revolve around adaptation and survival of an organism in its environment, allowing for the transfer of an entity's genes from one generation to the next, or facilitating the transfer of a member of one's close genetic neighbors through to the next generation. In nature, the environment, to an extent, defines the fitness function.

Also, in nature the optimum shifts constantly in response to a changing environment, and perhaps in a more rapid fashion in response, for example, to the introduction of a new predator into a prey's environment. Indeed, in this case a finely tuned organism adapted specifically to a particular environment, and perhaps predator, may find itself at a loss (and probably quickly eaten!) when these changes occur. However, an organism with perhaps a more sloppy encoding system and with many redundant genes may find it easier to adapt through a system of neutral networks.

Our point here is that we see nothing sacrosanct about the SGA with a binary encoding mechanism, but rather that any evolutionary algorithm embedding the basic operators of mutation and/or crossover and selective reproduction has considerable power for the development of novel solutions to problems, especially over evolutionary time scales. These time scales may, of course, now be telescoped, using modern technology from thousands or millions of years into weeks, days, or even hours.

Kenneth De Jong, one of the foremost researchers in the field of evolutionary algorithms, puts it thus in his widely cited 2006 text *Evolutionary Computation: a Unified Approach* (De Jong 2006).

...asking what the goals and purpose of evolution are immediately raises long-debated issues of philosophy and religion which, though interesting, are ... beyond the scope of this book. What is clear, however, is that even a system as simple as EV appears to have considerable potential for use as the basis for designing interesting new algorithms that can search complex spaces, solve hard optimization problems, and are capable of adapting to changing environments.

The EV cited in this quotation refers to a simple evolutionary algorithm just employing the mutation operator (no crossover), not entirely dissimilar to the evolutionary algorithm used in the illustrative EA control example described at the end of this chapter.

Genetic algorithms are less commonly used on their own in the control of robots or other machinery, but instead are usually combined with some other (generally bioinspired) paradigm. By far and away the most common combination is to use a genetic or possibly some other evolutionary algorithm to evolve the weights and/or topology of a neural network-based controller. This neural network may be either of the feed forward or recurrent type. A recent example of this use of evolutionary algorithms to evolve neural networks of increasing complexity is the neuroevolution of augmenting technologies (NEAT) approach (Stanley and Miikkulainen 2002), which is outlined later in this chapter.

2.3.2 Genetic Programming (GP)

Unlike genetic algorithms, which in their most basic form operate on fixed-length binary strings, the genetic programming approach involves the evolution of programs of varying complexity. GP has been employed from the earliest days of research in the evolutionary robotics field. For example, in 1995 Gritz and Hahn used genetic programming for the generation of articulated figure motion, including the training of a simple simulated humanoid to perform a number of tasks, which included pointing and touching objects (Gritz and Hahn 1995). John Koza, the preeminent researcher in this field, has written several books on the subject, the first of which (a classic in its field) was published in 1992 (Koza 1992).

2.3.3 Evolutionary Strategies (ES)

The evolution strategy of Rechenberg (1973), described in (Hoffmeister and Bäck 1991), in its initial form used just mutation and selective reproduction. The probability of mutation, however, varies with time as stated in Rechenberg's 1/5 success rule to control the mutation parameter:

The ratio of successful mutations to all mutations should be 1/5. If it is greater than 1/5 increase the variance, if it is less decrease the variance.

Rechenberg then extended the basic model to the so-called multimembered evolution strategy, which includes recombination. Mutation still, however, remained the main search mechanism. In addition, in an extension to the basic algorithm, the parameters of the system may also be subjected to change by the mutation and recombination operators. See Beyer and Schwefel (2002) for a detailed introduction to evolutionary strategies. Evolution strategies have been employed in several EHR applications, and form the core of the covariance matrix adaptation evolution strategy approach, described next.

2.3.4 Covariance Matrix Adaptation (CMA) Approach

The covariance matrix adaptation evolution strategy (CMA/ES) by Hansen and Ostermeier (2001), has formed the basis for a number of recent interesting research efforts in the evolution of both the morphology and controllers for humanoid and other legged creatures, mainly in simulation. CMA operates by adaptively varying the mutation distribution employed by the evolutionary strategy in order to make successful mutation steps that were made in the past more likely to occur in the future. Depending on the test function, speed improvements of several orders of magnitude have been observed when using the evolution strategy with covariance

matrix adaptation, compared to using it without CMA. This evolutionary technique has been employed by several researchers recently in the EHR field mainly in simulation, including Al Borno et al. (2013) in the synthesis of a wide range of human movements, including walking and break dancing, Wang et al. (2012) for the synthesis of walking and running in a 30 degrees of freedom (DOF) simulated human adult, and Urieli et al. (2011) in the generation of soccer skills for a simulated Nao robot. A detailed description of this algorithm is beyond the scope of this text; the interested reader is referred to Hansen (2006) and Hansen and Ostermeier (2001) for a comprehensive description.

2.3.5 Neuroevolution of Augmenting Technologies (NEAT)

The neuroevolution of augmenting technologies (NEAT) approach was originally developed as an approach to solving complex control and sequential decision tasks and has in recent times has been used in the evolution of biped locomotion (Lehman and Stanley 2011; Allen and Faloutsos 2009a, b). NEAT works on the basis of starting the evolutionary process with a small number of relatively simple neural networks, and over time the complexity of the network topologies increases leading to the creation of separate species of networks as the number of generations increases. Further, steps are taken to ensure that a level of diversity is maintained as evolution progresses. Detailed discussion of the NEAT approach is outside the scope of this text; however, the interested reader is referred to Stanley and Miikkulainen (2002, 2004) and Stanley et al. (2005) for comprehensive overviews of the algorithm.

2.4 Evolutionary Algorithms for Control—A Simple Example

Here I describe one of my initial experiments with evolutionary algorithms, leading to control as an area of application. This work was initially stimulated by visiting the first International Joint Conference on Neural Networks (IJCNN) in Washington DC in 1989, where a poster presentation described a system using neural networks assumed to be contained in small “animals” moving in a two-dimensional world containing “food” elements (Cecconi and Parisi 1989).

In their system, the world in which the creatures reside is a 10×10 matrix of square cells, holding just a simple creature and a single food element. Four actions are possible by the creature at any time step: move forward one step, turn 90° right, turn 90° left, or do nothing. The creature receives information about the position of the food in terms of the distance of the food from the animal and the angle formed by the line connecting the creature and the food with some reference. Using the

supervised learning back-propagation procedure, the creature was taught to predict the next position (angle, distance) of the food after each action.

In another part of the experiment the goal was to train the creature to approach and find the food. An 80×80 grid was used in this case, containing from 500 to 700 food elements. The creature was initially allowed to roam the environment selecting actions at random. Whenever it stopped on a food cell it was deemed to have “eaten” it and the food disappeared from the environment. Each time the creature stepped on a food cell the spontaneous activity was stopped. The creature recorded the sequences of actions that led it to the food cell and it was restarted in the position it had been in eight actions before, and these actions were repeated. This time, however, back-propagation was applied with the neural network receiving a training input on the output units coding the action selected by the animal. The training input was the action which had previously been selected at that point, as this had been a successful action leading the animal to the food. Cecconi and Parisi then went on to correlate ability at predicting the location of food to the ability to find and eat it. However, the section of most interest to me was at the end where they mentioned the case:

in which the food approaching ability evolved in networks through genetic selection and random mutation

Details of these experiments were not provided, but my interest was by now sufficiently aroused to begin experimentation. My initial formulation concerned a simulated “beast no. 1” which would roam, as in the previous experiments, a two-dimensional world of squares containing a single food element. The size of the world was 16×16 squares, and 4 actions were possible, as in Cecconi and Parisi’s experiments: to move one square in the direction being faced, to turn left, to turn right, or to do nothing. Sensory input to the beast was very simple, consisting of one of four inputs, either the food is forward (of the facing direction of the beast) to the left, to the right, or behind (Fig. 2.4).

The overall controlling program “Reality” called the two main routines, “Beast”, for controlling the simulated creature, and “Genops” the genetic operators sub-routine controlling the production of new generations of “creatures” by selective reproduction (“Newgen” and “Select”) and the application of the genetic operator of mutation. No other genetic operators were allowed, which was in keeping with the description of Cecconi and Parisi (1989).

Each beast was designed to operate as a stochastic automaton where the network weights represented the probability of transition from one state to the next. However, rather than updating the probabilities after each action (which assumes some sort of teacher or supervised learning, we assume that updating only takes place after a sequence of actions leading to either a successful (food is found) or unsuccessful (no food) outcome. As we are using an evolutionary algorithm to update the networks’ weights, we are faced with the task of the encoding of the network structure in the chromosome. We chose a fixed chromosome structure of 32 bits arranged as in Fig. 2.5 (Eaton 1993b).

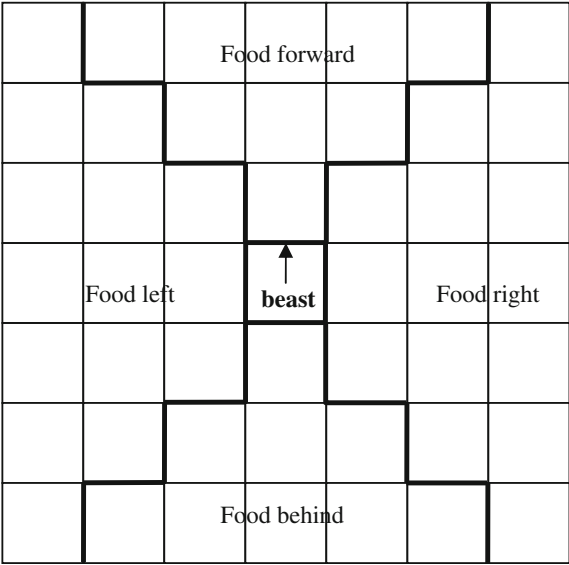


Fig. 2.4 Sensory input to “beast no. 1”

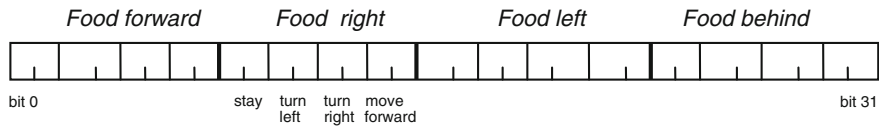


Fig. 2.5 Chromosome structure for simulated creature

In the initial experiments connections were first-order, i.e., the next action depended only on the present position of the food relative to the beast.

The chromosome encodes the probability of taking a particular action on the next time step based on a particular input. The probability of selection of individual actions were encoded two bits per weight using normal binary values which we then normalised so that the sum of the probabilities of action for a particular input was equal to one. As an example, assume that the food is currently ahead of the beast and the first eight bits of the chromosome are as in Fig. 2.6.

The probability of the beast remaining in the same position for this time step may be calculated as follows:

$$P(stay) = \frac{Val(stay)}{\sum Vals} \tag{2.1}$$

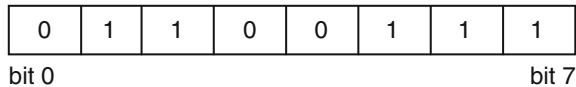


Fig. 2.6 Example chromosome section

where $Val(stay)$ is the binary equivalent of the two-bit code in the corresponding locations in the chromosome, and $\sum Vals$ is the sum over all the values for this input. Returning to our example, the probability in this case of staying in position is $1/7$ or 0.1428 . If it happens that all of the bits are zero, an action is simply chosen randomly.

A fixed population of 30 controllers was allowed to run over a number of generations, starting from a random group of chromosomes, subject to the mutation operator. The mutation rate was allowed to vary over time to implement a form of annealing, and high mutation rates apply initially, which are reduced as the creatures converge on optima. Specifically, the probability of an individual bit being mutated was

$$P(mutation) = \frac{1}{fitness} \quad (2.2)$$

where the fitness of an individual was based on the number of successful runs over a number of trials combined with the number of steps per trial. Specifically:

$$fitness = 2^N \sum_{i=1}^T \frac{1}{C_i} \quad (2.3)$$

where N is the number of successful food runs in T trials, i.e., the number of times the beast successfully located the food with the beast and food being located at different randomly chosen locations for each run. C_i is cycle time per trial, i.e., the number of steps per trial. If the beast has not located the food after 10,000 time steps the trial is terminated, and this serves as the cycle time for this trial. The number of trials, T , was set equal to 10 for all simulations described here.

Once the system had been run, over typically 60 generations, the results were collected and graphs generated of maximum and average fitness per generation together with output showing the types of paths that the creatures took in locating (or not) the food. Interesting results were obtained; in general with the creatures learning to locate food early in the evolutionary process.

It is intriguing to note that the seemingly correct deterministic solution.

Algorithm FINDFOOD

```
While food-not-found
  If food-ahead then
    move-forward
  else if food-to-right then
    turn-right
  else if food-to-left then
    turn-left
  else turn-left or turn-right
  End If
End While
```

does not, in fact, work (interested readers might like to confirm this for themselves). The author found this, to his surprise, when he decided to upstage the evolutionary algorithm with some expert knowledge!

Evolutionary Humanoid Robotics

Eaton, M.

2015, XII, 144 p. 24 illus., 17 illus. in color., Softcover

ISBN: 978-3-662-44598-3