15

Coevolutionary Systems

In most of this book we have been concerned with problems where the quality of a proposed solution can be relatively easily measured in isolation by some externally provided fitness function. Evaluating a solution may involve an element of random noise, but does not particularly depend on the *context* in which it is done. However, there are two obvious scenarios in which this set-up does not really hold. The first occurs when a solution represents some strategy or design that works in opposition to some competitor that is itself adapting. The most obvious example here would be adversarial game-playing such as chess. The second comes about when a solution being evolved does not represent a complete solution to a problem, but instead can only be evaluated as part of a greater whole, that together accomplishes some task. An example might be the evolution of a set of traffic-light controllers, each to be sited on a different junction, with fitness reflecting their joint performance in reducing congestion over a day's simulated traffic. Both of these are examples of **coevolution**. This chapter gives an overview of the types of scenarios where coevolution might be usefully applied, and of some of the issues involved in designing a successful application.

15.1 Coevolution in Nature

Previously in this book we made extensive use of Wright's analogy of the adaptive landscape, where an evolving population is conceptualised as moving on a surface whose points represent the set of possible solutions. This metaphor ascribes a vertical dimension to the search space that denotes the fitness of a particular solution, and the combined effects of selection and variation operators move the set of points into high-fitness regions.

While an attractive metaphor, this can also be profoundly misleading when we consider the adaptation of a biological species. This is because it tends to lead to the implicit notion that solutions have a fitness value *per se*. Of course, in life the adaptive value (that is, fitness) of an organism is entirely determined by the environmental niche in which it finds itself. The characteristics of this niche are predominantly determined by the presence and character of other organisms from the same and, in particular, different species.[1]

The effect of other species in determining the fitness of an organism can be positive – for example, the pollination of plants by insects feeding on their nectar – or negative – for example, the eating of rabbits by foxes. Biologists tend to use the terms **mutualism** and **symbiosis** to refer to the coadaptation of species in a mutually beneficial way, and the terms **predation** or **parasitism** to refer to relationships in which one species has a negative effect on the survival and reproductive success of the other (antagonism).

If all of the other species in an environmental niche remained the same, and only one species was evolving, then the notion of a fixed adaptive landscape would be valid for that species. However, since evolution affects all species, the net effect is that the landscape 'seen' by each species is affected by the configuration of all the other interacting species, i.e., it will move. This process is known as coevolution. To give a concrete example, the adaptive value to a rabbit of being able to run at, say, 20 km/h depends entirely on whether the fox that preys on it has a maximum top speed of 15 km/h or 30 km/h. The height on the landscape of a 20-km/h phenotype is reduced over time from a high value to a low value as the fox evolves the ability to run faster.

Despite the additional complications of coevolutionary models, they hold some significant advantages that have been exploited within EAs to aid the generation of solutions to a range of difficult problems. One that we have already described in Sect. 6.5 is the coevolution of a population of partial models in Michigan-style LCS — these may be thought of as co-operating to provide a complete model of a problem. Another very well known example is the modelling and evolution of game-playing strategies. In this case evolving solutions play against each other to get their fitness, i.e., only one species is used and the model is competitive in the sense defined in Sect. 15.3. Since computers provide the freedom to use a number of different models, and biology is serving as an inspiration rather than a strict blueprint, a number of different models have been used successfully. Coevolutionary EAs have been implemented using both cooperation and competition, and both single and multiple-species models, as we shall now describe.

15.2 Cooperative Coevolution

Coevolutionary models in which a number of different species, each representing part of a problem, cooperate in order to solve a larger problem have been successfully applied many times. Among many examples of this are high-dimensional function optimisation [342] and job shop scheduling [225].

The advantage of this approach is that it permits effective function decomposition; each subpopulation is effectively solving a much smaller, more

[1] With the possible exception of some extremely simple organisms.

tractable problem. The disadvantage is that it relies on the user to subdivide the problem which may not be obvious from the overall specification. In nature, mutually beneficial relationships have as their ultimate expression so-called **endosymbiosis**, where the two species become so interdependent that they end up inextricably physically linked – for example, the various gut bacteria that live entirely within a host's body and are passed from mother to offspring. The equivalent in EA optimisation is where the different parts of a problem are so interdependent that they are not amenable to division.

Bull [68] conducted a series of more general studies on cooperative coevolution using Kauffman's static NKC model [244] in which the amount of effect that the species have on each other can be varied systematically. In [69] he examined the evolution of coevolving symbiotic systems that had the ability to evolve linkage flags denoting that solutions from different populations should stay together. He showed that the strategies that emerge depend heavily on the extent to which the two populations affect each other's fitness landscape, with linkage preferred in highly interdependent situations.

15.2.1 Partnering Strategies

When cooperating populations are used, a major issue is that of deciding how a solution from one population should be paired with the necessary others in order to gain a fitness evaluation.

Potter and De Jong [342] used a generational GA in each subpopulation, with the different species taking it in turns to undergo a round of selection, recombination, and mutation. Evaluation was performed using the current best from each of the other species.

Paredis coevolved solutions and their representations in a steady-state model using what he termed lifetime fitness evaluation (LTFE) [332]. In the most general form of LTFE a new individual undergoes 20 'encounters' with solutions selected from the other population. The fitness of the new individual is initially set as the mean fitness from these encounters. The effect of this scheme is that individuals from each population are continuously undergoing new encounters, and the fitness of an individual is given by the running average of its performance in the last 20 encounters. The benefit of this running-average approach is that it effectively slows down the rate at which each fitness landscape changes in response to changes in the composition of the other populations.

Husbands [225] solved the pairing problem and also effectively changed the rate at which the composition of the different populations are *perceived* to change by using a diffusion model EA (Sect. 5.5.7) with one member of each species located on each grid point.

Bull [69] examined the use of a range of different pairing strategies: best, random, stochastic fitness-based, joined, and distributed, as per [225]. His results showed that no one strategy performed better across the range of different interaction strengths and generational models, but random was robust in a generational GA, and distributed did best in a steady-state GA. When fitness sharing was added to prevent premature convergence, "best" became the most robust solution.

Finally, within the field of cooperative coevolution it is worth mentioning the use of **automatically defined functions** within GP [253]. In this extension of GP, the function set is extended to include calls to functions that are themselves being evolved in parallel, in separate populations. The great advantage of this is in permitting the evolution of modularity and code reuse.

15.3 Competitive Coevolution

In the competitive coevolution paradigm individuals compete against each other to gain fitness at each other's expense. These individuals may belong to the same or different species, in which case it is arguably more accurate to say that the different species are competing against each other.

As noted above, the classic example of this that generated much interest in the paradigm was Axelrod's work on the **iterated prisoner's dilemma** [14, 15], although early work can be traced back as far as 1962 [39]. This is a two-player game, where each participant must decide whether to cooperate or defect in each iteration. The payoff received depends on the actions of the other player, as determined by a matrix such as Table 15.1.

	Player B	
Player A	Cooperate	Defect
Cooperate	(3,3)	(0,5)
Defect	(5,0)	(1,1)

Table 15.1. Example payoff matrix for iterated prisoner's dilemma. Payoff to player A is first of pair

Axelrod organised tournaments in which human-designed strategies competed against each other, with strategies only allowed to "see" the last three actions of their opponent. He then set up experiments in which strategies were evolved using as their fitness the mean score attained against a set of eight human strategies. He was able to illustrate that the system evolved the best strategy (tit-for-tat), but there was some brittleness according to the set of human strategies chosen. In a subsequent experiment he demonstrated that a strategy similar to tit-for-tat could also be evolved if a coevolutionary approach was used with each solution playing against every other in its current generation in order to assess its quality.

In another groundbreaking study, Hillis [213] used a two-species model with the pairing strategy determined by colocation on a grid in a diffusion model EA. Note that this parallel model is similar to, and in fact was a precursor of, Husbands' cooperative algorithm described above. Hillis' two populations represented sorting networks, whose task it was to sort a number of inputs numerically, and sets of test cases for those networks. Fitness for the networks is assigned according to how many of the test cases they sort correctly. Using the antagonistic approach, fitness for the individuals representing sets of test cases is assigned according to how many errors they cause in the network's output. His study caused considerable attention as it found correct sorting networks that were smaller than any previously known.

This two-species competitive model has been used by a number of authors to coevolve classification systems [181, 333]. The approach of Paredis is worth noting as it solves the pairing strategy problem by using a variant of the LTFE method sketched above.

As with cooperative coevolution, the fitness landscapes will change as the populations evolve, and the choice of pairing strategies can have a major effect on the observed behaviour. When the competition arises within a single population, the most common approaches are to either pair each strategy against each other, or just against a randomly chosen fixed-size sample of the others. Once this has been done, the solutions can be ranked according to the number of wins they achieve and any rank-based selection mechanism chosen.

If the competition arises between different populations, then a pairing strategy must be chosen for fitness evaluation, as it is for cooperative coevolution. Since the NKC model essentially assigns random effects to the interactions between species, i.e., it is neither explicitly cooperative nor competitive, it is likely that Bull's results summarised above will also translate to this paradigm.

The main engine behind coevolution is sometimes called "competitive fitness evaluation". As Angeline states in [10], the chief advantage of the method is that it is self-scaling: early in the run relatively poor solutions may survive, for their competitors are not strong either. But as the run proceeds and the average strength of the population increases, the difficulty of the fitness function is continually scaled.

15.4 Summary of Algorithmic Adaptations for Context-Dependent Evaluation

As the discussion above shows, the choice of context, or equivalently the pairing strategy, can have a significant effect on how well EAs perform in this type of situation, and many successful approaches attempt to reduce this effect by awarding fitness as an average of a number of contexts encountered.

The second major algorithmic adaptation is the incorporation of some kind of history into the evaluation — often in the form of an archive of historically 'good solutions against which evolving solutions are periodically tested. The reason for this is to avoid the problem of cycling: a phenomenon where evolution repeatedly moves through a series of solutions rather than making advances. To illustrate this consider as an example the simple 'rock–scissors–paper game. In a population which had converged to a 'rock strategy, a mutant individual displaying 'scissors would be disadvantaged, whereas a 'paper strategy would do well and come to dominate the population. This in turn provides the conditions to favour a chance mutation into a 'scissors strategy, and later back to 'rock', and so on. This form of cycling can be avoided by the use of an archive of different solutions, which can provide impetus for the evolution of more complex strategies, and has been demonstrated in a range of applications.

15.5 Example Application: Coevolving Checkers Players

In [81], which is expanded into a highly readable book [168] and further summarised in [169], Fogel charts the development of a program for playing the game of checkers (a.k.a. draughts), a board game that is also highly popular on the Internet. In this two-player game a standard 8×8 squared board is used, and each player has an (initially fixed) number of pieces (checkers), which move diagonally on the board. A checker can 'take' an opponent's piece if it is adjacent, and the checker jumps over it into an empty square (both players use the same-coloured squares on the board). If a checker reaches the opponent's home side, it becomes a 'king' in which case it can move backwards as well as forwards. Human checker players regularly compete against each other in a variety of tournaments (often Internet-hosted), and there is a standard scheme for rating a player according to their results.

In order to play the game, the program evaluates the future value of possible moves. It does this by calculating the likely board state if that move is made, using an iterative approach that looks a given distance ('ply') into the future. A board state is assigned a value by a neural network, whose output is taken as the 'worth' of the state from the perspective of the last player to move.

The board state is presented to the neural network as a vector of length 32, since there are 32 possible board sites. Each component comes from the set $\{-K, -1, 0, 1, K\}$, where the minus sign presents an opponent's king or piece, and K takes a value in the range [1.0, 3.0].

The neural network thus defines a "strategy" for playing the game, and this strategy is evolved with evolutionary programming. A fixed structure is used for the neural networks, which has a total of 5046 weights and bias terms that are evolved, along with the importance given to the kings K. An individual solution is thus a vector of dimension 5047.

The authors used a population size of 15, with a tournament size 5. When programs played against each other they scored $+1$, 0, -2 points for a win, draw, and loss, respectively. The 30 solutions were ranked according to their scores over the 5 games, then the best 15 became the next generation.

The mutation operator used took two forms: the weights/biases were mutated using the addition of Gaussian noise, with lognormal adaptation of the step sizes *before* mutation of the variables, i.e., using standard self-adaptation with $n = 5046$ strategy parameters. The offspring king weightings were mutated accorded to $K' = K + \delta$, where δ is sampled uniformly from $[-0.1, 0.1]$, and the resulting values of K' are constrained to the range $[1.0, 3.0]$. Weights and biases were initialised randomly over the range $[-0.2, 0.2]$. K values were initially set to 2.0, and the strategy parameters were initialised to 0.05.

The authors proceeded by having the neural networks compete against each other for 840 generations (6 months) before taking the best evolved strategy and testing it against human opponents on the Internet. The results were highly impressive: over a series of trials the program earned an average ranking that put it in the "expert" class, and better than 99.61% of all rated players on the website. This work is particularly interesting in the context of artificial intelligence research for the following reasons:

- There is no input of human expertise about good short-term strategies or endgames.
- There is no input to tell the evolving programs that in evaluating board positions a negative vector sum (that is, the opponent has a higher piece-count) is worse than a positive vector sum.
- There is no explicit credit assignment'mechanism to reward moves that lead to wins; rather a 'top-down' approach is taken that gives a single reward for an entire game.
- The selection function averages over five games, so the effects of strategies that lead to wins or losses are blurred.
- The strategies evolve by playing against themselves, with no need for human intervention!

In the spirit of this chapter, we can say that the approach taken is rather similar to Paredis' Life-time fitness evaluation — to even out the effect of a specific context (combination of opponents) fitness is awarded as the average performance across a number of contexts.

For exercises and recommended reading for this chapter, please visit `www.evolutionarycomputation.org`.