Popular Evolutionary Algorithm Variants

In this chapter we describe the most widely known evolutionary algorithm variants. This overview serves a twofold purpose: On the one hand, it introduces those historical EA variants without which no EC textbook would be complete together with some more recent versions that deserve their own place in the family tableau. On the other hand, it demonstrates the diversity of realisations of the same basic evolutionary algorithm concept.

6.1 Genetic Algorithms

The genetic algorithm (GA) is the most widely known type of evolutionary algorithm. It was initially conceived by Holland as a means of studying adaptive behaviour, as suggested by the title of the book describing his early research: *Adaptation in Natural and Artificial Systems* [220]. However, GAs have largely (if perhaps mistakenly – see [103]) been considered as function optimisation methods. This is perhaps partly due to the title of Goldberg’s seminal book: *Genetic Algorithms in Search, Optimization and Machine Learning* [189] and some very high-profile early successes in solving optimisation problems. Together with De Jong’s thesis [102] this work helped to define what has come to be considered as the classical genetic algorithm — commonly referred to as the ‘canonical’ or ‘simple GA’ (SGA). This has a binary representation, fitness proportionate selection, a low probability of mutation, and an emphasis on genetically inspired recombination as a means of generating new candidate solutions. It is summarised in Table 6.1. Perhaps because it is so widely used for teaching EAs, and is the first EA that many people encounter, it is worth re-iterating that many features that have been developed over the years are missing from the SGA — most obviously that of elitism.

While, the table does not indicate this, GAs traditionally have a fixed workflow: given a population of \( \mu \) individuals, parent selection fills an intermediary population of \( \mu \), allowing duplicates. Then the intermediary population is shuffled to create random pairs and crossover is applied to each consecutive
pair with probability \( p_c \) and the children replace the parents immediately. The new intermediary population undergoes mutation individual by individual, where each of the \( l \) bits in an individual is modified by mutation with independent probability \( p_m \). The resulting intermediary population forms the next generation replacing the previous one entirely. Note that in this new generation there might be pieces, perhaps complete individuals, from the previous one that survived crossover and mutation without being modified, but the likelihood of this is rather low (depending on the parameters \( \mu, p_c, p_m \)).

<table>
<thead>
<tr>
<th>Representation</th>
<th>Bit-strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recombination</td>
<td>1-Point crossover</td>
</tr>
<tr>
<td>Mutation</td>
<td>Bit flip</td>
</tr>
<tr>
<td>Parent selection</td>
<td>Fitness proportional - implemented by Roulette Wheel</td>
</tr>
<tr>
<td>Survival selection</td>
<td>Generational</td>
</tr>
</tbody>
</table>

Table 6.1. Sketch of the simple GA

In the early years of the field there was significant attention paid to trying to establish suitable values for GA parameters such as the population size, crossover and mutation probabilities. Recommendations were for mutation rates between \( 1/l \) and \( 1/\mu \), crossover probabilities around 0.6-0.8, and population sizes in the fifties or low hundreds, although to some extent these values reflect the computing power available in the 1980s and 1990s.

More recently it has been recognised that there are some flaws in the SGA. Factors such as elitism, and non-generational models were added to offer faster convergence if needed. As discussed in Chap. 5, SUS is preferred to roulette wheel implementation, and most commonly rank-based selection is used, implemented via tournament selection for simplicity and speed. Studying the biases in the interplay between representation and one-point crossover (e.g. [411]) led to the development of alternatives such as uniform crossover, and a stream of work through ‘messy-GAs’ [191] and ‘Linkage Learning’ [209, 395, 385, 83] to Estimation of Distribution Algorithms (see Sect. 6.8). Analysis and experience has recognised the need to use non-binary representations where more appropriate (as discussed in Chap. 4). Finally the problem of how to choose a suitable fixed mutation rate has largely been solved by adopting the idea of self-adaptation, where the rates are encoded as extra genes in an individuals representation and allowed to evolve [18, 17, 396, 383, 375].

Nevertheless, despite its simplicity, the SGA is still widely used, not just for teaching purposes, and for benchmarking new algorithms, but also for relatively straightforward problems in which binary representation is suitable. It has also been extensively modelled by theorists (see Chap. 16). Since it has provided so much inspiration and insight into the behaviour of evolutionary processes in combinatorial search spaces, it is fair to consider that if OneMax is the *Drosophila* of combinatorial problems for researchers, then the SGA is the *Drosophila* of evolutionary algorithms.