3
Genetic Algorithm

3.1. OVERVIEW

A genetic algorithm (GA), is an optimisation method inspired in nature and its evolution mechanisms. Starting with a set of typically randomly generated solutions – called individuals of the initial population - the algorithm tries to breed new individuals based on the current generation using operators of selection, crossover and mutation similar to those used by biological organisms.

A GA is a metaheuristic method and, as such, the solution space to be explored is limited by various constraints such as computational capacity, algorithm parameters, problem information and time constraints. Given these limitations the optimum solution returned by the algorithm may not represent the global optimum but a satisfactory solution for the problem. Figure 3.1 shows the flow of a standard GA.

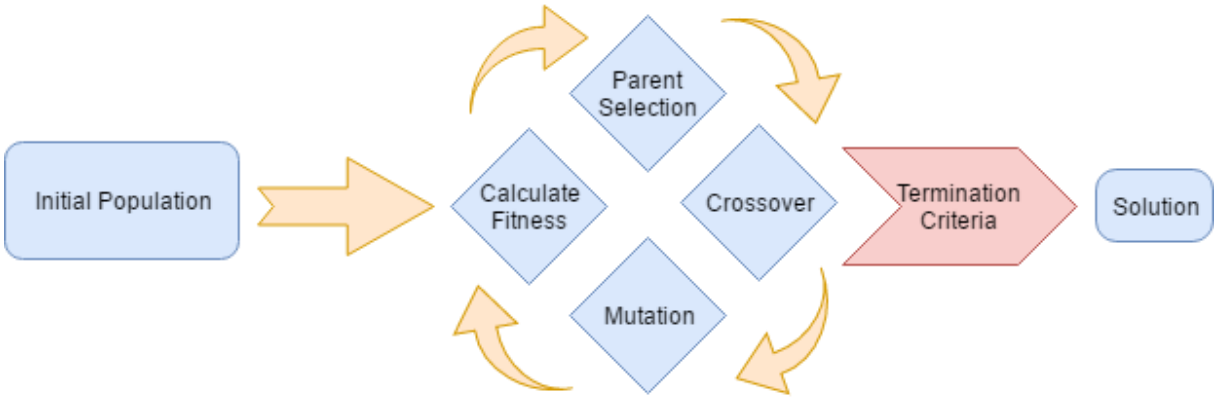

Fig. 3.1 – Flow of a standard GA

The initial population is generated and populated with individuals whose chromosome is randomly generated. Each individual represents a different solution for the problem.

A fitness function runs through every individual and awards a fitness value to each solution. It represents the merit each solution has and is used later to select parents for the next generation. A fitness function can be static or dynamic, i.e. – changing at runtime.

When every individual of the initial population has a fitness value assigned to it, an Evolve command is issued triggering a set of operations. The parents of each new individual are selected by a method

that takes into account the fitness value of each individual of the current population. Individuals with better fitness are more likely to be selected. The selection is passed on to the first genetic operator, usually a crossover function that has the task of assembling a new chromosome made up of blocks from the parents' chromosomes. At this point, the chromosome of the new individual is built, however, to preserve the diversity of solutions during the search of the optimum individual, a mutation operator is used to introduce new information into the chromosome.

The new individual is then evaluated with the fitness function, and the process is repeated until a predefined stop criteria is met. The next subchapters will detail the fundamental elements of a GA.

3.2. FUNDAMENTAL ELEMENTS OF A GENETIC ALGORITHM

3.2.1. INDIVIDUAL

In nature, a set of organisms from the same species defines a population and each organism is described by a set of encoded instructions on its DNA. Each chromosome sets specific traits such as eye colour, and each trait can have multiple values: blue, brown, green, etc.

In a GA an individual does not represent an organism but a possible solution for the optimisation problem. The way it solves the problem is encoded in a convenient datatype where each characteristic of the solution is stored, similar to DNA in natural organisms. In general, an individual shares with the others the structure of its chromosome, but the values stored change from individual to individual (Figure 3.2).

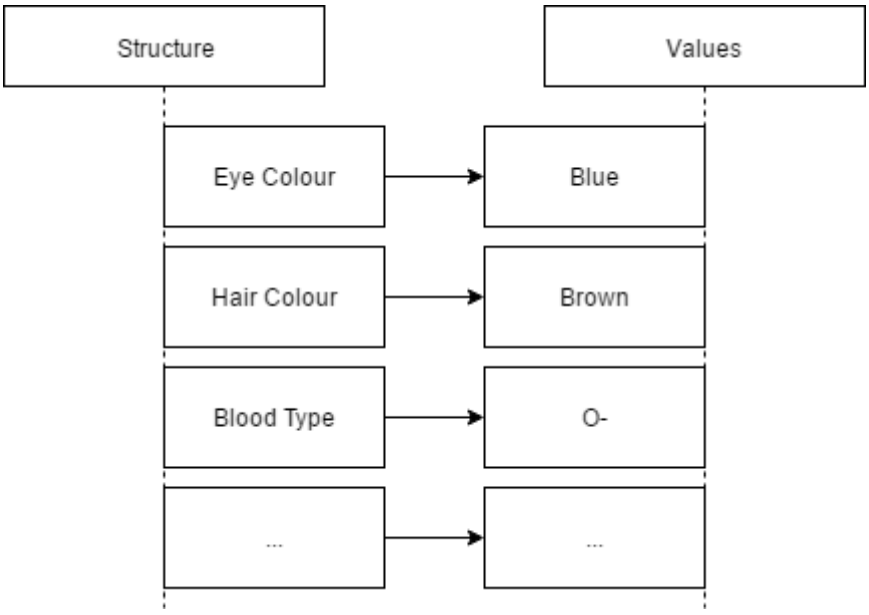

Fig. 3.2 – Chromosome structure and values

The original representation of a chromosome is a simple bit string vector – a series of 0 and 1 (Figure 3.3) – but even though it is the classical representation it is not always the most useful. For different problems, other datatypes can be better suited to store the characteristics of each individual, for example, a given problem might be better solved using individuals where the DNA is stored in a

multi-dimensional array and where the values stored do not necessarily use the base 2 (binary) numeric system.

$$[1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1]$$

Fig. 3.3 – Basic bit string chromosome

The datatype choice is problem dependant, the number of different variables that are changed during the optimisation and their relationship with each other, can exclude certain datatypes and make others more attractive for the developer. As an example (Figure 3.4), when optimising a structure's sections a vector can be used to represent the section in each bar, but if in addition to the section, the node coordinates (x,y,z) also need to be optimised, an array is probably the best way forward to represent the DNA of each individual solution.

$$\begin{bmatrix} Number & x & y & z & Section \\ 1 & 1.1 & 0.0 & 10.0 & IPE100 \\ 2 & 0.9 & 4.1 & 25.3 & IPE200 \\ 3 & 0.3 & 1.5 & 7.3 & IPE200 \\ 4 & 2.2 & 2.8 & 5.0 & L90x90x9 \\ 5 & 1.0 & 3.0 & 3.0 & IPE140 \end{bmatrix}$$

Fig. 3.4 – More complex chromosome datatype

3.2.2. Initial Population

The initial population is the beginning of the GA. If there is previous knowledge of where the optimal solution is located in the search space, the initial population can be seeded near that position to save time in the search process. When there is no information that points the developer in a specific direction to where the ideal solution is located, a random set of individuals is generated to create the initial population (Figure 3.5).

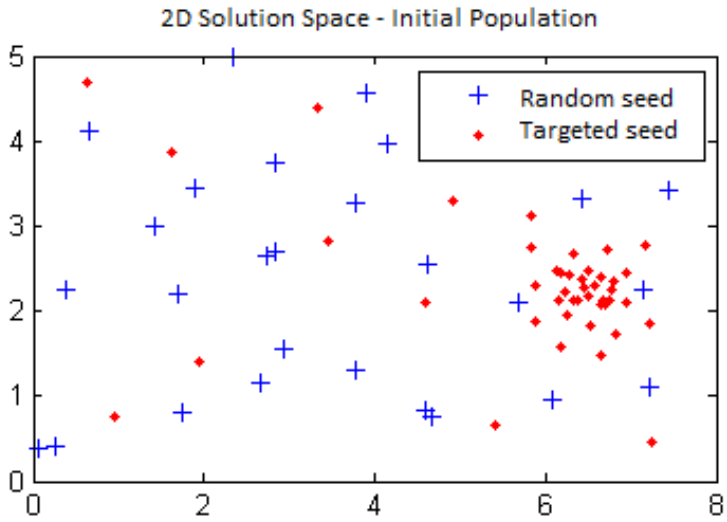

Fig. 3.5 – Random vs. Seeded population [6]

The size of the initial population usually increases with the number of genes the GA operates with, in the search of the optimum solution. Using the example referred above, the GA that only optimises the bar sections will need a smaller population size than the second GA that optimises the section and

topology of the structure, in order to populate the solution space as well as the first GA in the previous example.

3.2.3. FITNESS FUNCTION

The Fitness function is a critical part of the GA and, it is responsible for the merit evaluation of each solution so that the best individuals have a larger probability to pass on their genes. To do that successfully, it needs to evaluate each individual and give penalties for bad solutions to decrease reproduction probability.

In simple problems, when there is a single and clear objective for the optimisation, the definition of the fitness function can be straightforward. For example:

-Arrange various shapes to be cut in a fabric of WxH (Width and Height) in order to reduce material waste. Fitness function: Evaluate the wasted area of each solution, apply penalties when the shapes overlap.

For more complex problems the first challenge arises, the most common is the optimisation of more than one parameter.

When more than one parameter is evaluated by the fitness function there needs to be a way to translate that multiparameter evaluation in a single classification that represents the quality of the solution. Such that operation can be quite difficult, for example:

When optimising a structure in addition to the material usage there might be specific nodes in the structure that cannot deflect more than a given value. In this case, several questions can be raised:

- How to weight the rigidity and mass parameters of the structure?
- When deflections are checked, is every analysed node equally important?
- How to add a penalty for a structure which has acceptable stiffness but poor material usage?
- Is the penalty so harsh that it can turn the algorithm into a random search?

The second challenge, which is very common in engineering problems, is the impossibility to evaluate directly each solution. Most GAs applied to engineering problems will need to evaluate each solution in FEM (Finite element modelling) or CFD (Computer fluid dynamics) software and retrieve the results to evaluate the solution. For example, to optimise the drag coefficient of a car a call to an external CFD program needs to be made to get the drag coefficient of each solution. This requires external calls to complex software packages that can significantly delay the GA, making computational power and time an important constraint in such problems.

The fitness evaluation can be static or it can vary at runtime. As the generations increase, that change in evaluation is useful in very sensitive problems where the scale of penalties awarded initially to arrive at a near optimum are inadequate for the final adjustments needed to arrive at the final solution.

When it is impossible to define a fitness function, the adoption of interactive GAs might be an appropriate solution depending complexity of the problem. In interactive GAs, the fitness function is in fact the user that is asked to rate each individual. This is a useful approach when the algorithm is trying to optimise a problem classified with a subjective parameter such as aesthetics.

3.2.4. SELECTION FUNCTION

The selection function is used in conjunction with the fitness function to translate the Darwinian concept of survival of the fittest to the optimisation algorithm.

There are various ways to implement the selection function. The most commonly used are the roulette wheel selection and tournament selection, both are explained in the next subchapters. Even though there are clear differences between the various possible algorithms, the final goal is the same, i.e., to ensure that the individuals with better fitness classification are selected more often so that their genes are carried over to the next generation. When repeated over generations, it acts as a filter for the bad genes.

3.2.4.1. Roulette wheel

The roulette wheel algorithm (Figure 3.6) is the most commonly used method in traditional GAs. The key operations are:

- Sort the individuals according to their fitness value;
- Add the fitness values of each individual in the population and store the value:

$$Total = \sum fitness \tag{3.1}$$

- Assign to each individual a selection probability:

$$p = \frac{fitness}{Total} \tag{3.2}$$

- Randomly generate a number between 0.0 and 1.0;
- Sum the probability of each individual until the generated number is smaller than that sum;
- The last added individual is one of the parents of the next generation.

The implementation is straightforward and is effective in problems where convergence is easily attainable [7]. When there is a need for more control on the selection function, the tournament selection is a better option.

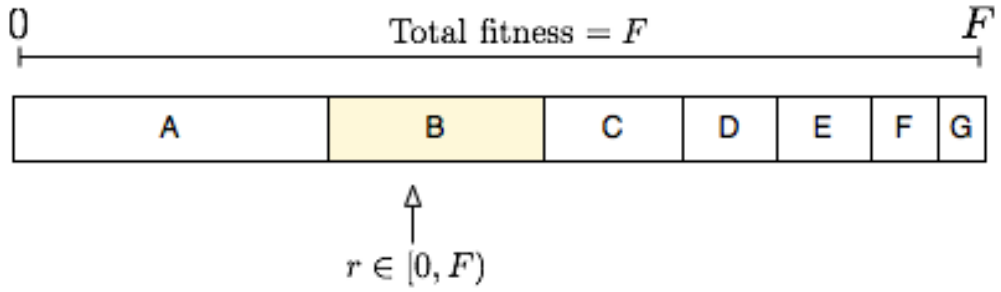

Fig. 3.6 – Roulette wheel: A is better than G therefore selected more often

3.2.4.2. Tournament selection

This method as the name implies is based on a tournament between individuals. A pool of randomly selected individuals from the population is created, and their finesses are compared. That tournament can return the best individual or a probability can be set for the selection of the best individuals, for example: the best individual is selected 70% for the time, the second 20% and the third 10% (Figure 3.7).

By having two adjustable parameters it is possible to fine-tune the selection algorithm to better suit the problem. An increase in the pool of individuals increases the selection pressure, decreasing the probability of the worse individuals being selected and by changing the probability of selection among the best individuals, early convergence is avoided.

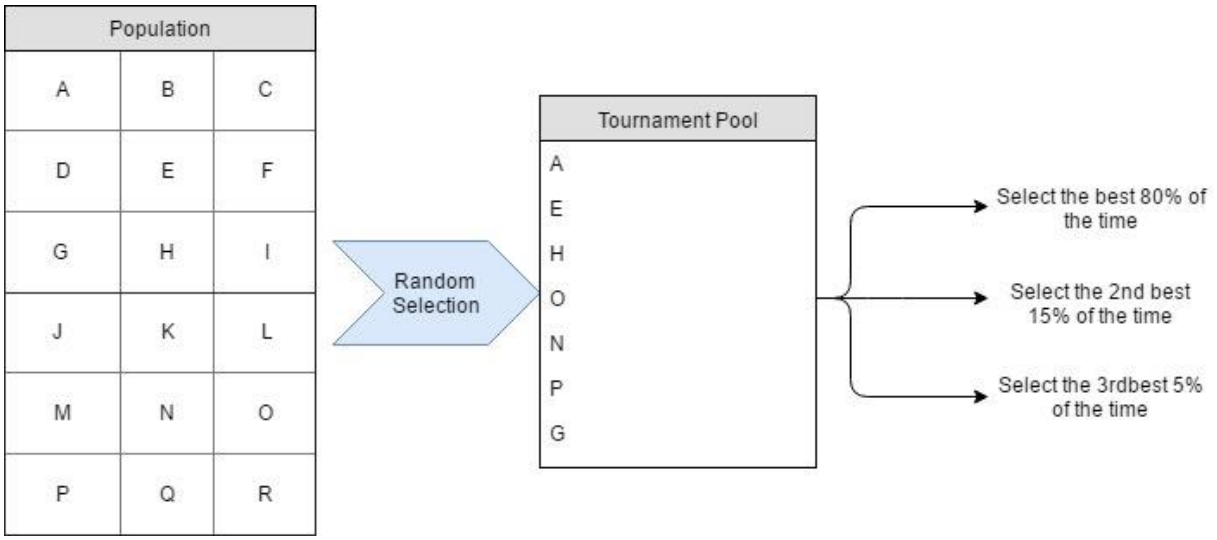

Fig. 3.7 – Tournament selection

3.2.5. GENETIC OPERATORS

Genetic operators receive the chromosomes from the parent individuals, and with manipulation of genes return a new individual inspired on its parents. In many cases the crossover operator is an acceptable way of solving a problem using GAs. However, it is common to add a mutation operator to ensure the diversity of the solutions[8].

3.2.5.1. Crossover

The crossover operator, receives the gene blocks from the parents and assembles them to create a new chromosome made of gene blocks from each parent that is assigned to the new individual.

There are several methods to create a new chromosome, the most common are the single and double point crossover that will be detailed next. The uniform crossover is another method that widens the search space of the GA. Due to this property, it is only suited for simple problems where the starting search space is not too large or when constrains such as computational capacity and time are not relevant.

The single and double point methods (Figure 3.8) are very similar, the chromosome of one parent is cut at a specific point and from that point onwards the chromosome is replaced by genes from the

other parent. The difference between single and double point is in the final point of mutation. Whilst in the single point crossover the genome uses genes from the second parent until the end of the genetic code, in the double point crossover, the replacement of genes is conducted until a generated end point.

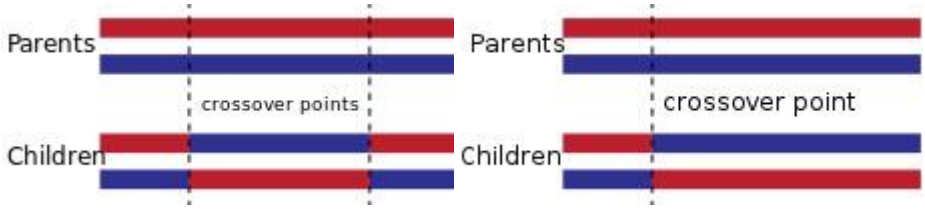

Fig. 3.8 – Double and single point crossover

3.2.5.2. Mutation

The mutation operator adds new information to the chromosome that is received from the crossover function.

This operator changes information randomly on some genes of the chromosome that is received from the crossover function. The objective is to preserve diversity in the solution space so that early convergence to a local optimum is prevented.

The operation is defined as a probability that each gene has of being changed to a random value. That probability needs to be low enough to ensure that the search does not degrade into a random search of the optimum.

The way the mutation is achieved depends on the datatype used to store the chromosome. For the classic bit string format, the mutation is achieved by a simple random change from 0 to 1 in some places of the string.

Mutation when the gene values are not in base 2 (binary) needs to be carried out with caution to avoid excessively drastic value changes that are particularly bad in the final phases of the GA when the fine-tuning of the solution is underway. To have that additional control, a method like the Gaussian mutation with the mean in the current value of the gene is desirable. This algorithm has been adopted in the present work and will be detailed in Chapter 4.

3.2.6. TERMINATION CRITERIA

There needs to be a way of informing the algorithm to stop the search. Such instruction is issued by the termination criteria. The developer has some leeway when defining this block of code but some circumstances might need to be considered. Termination examples are:

- Maximum number of generations;
- Found a sufficiently good solution;
- No recorded improvement over the last X generations;
- Computational time limit exceeded.