University of Babylon, College of science for women
Dept. of Computer science

Evolutionary Computing

Forth year

Dr. Salah Al-Obaidi

Lecture #6: Fitness and Selection Fall 2024

Contents

4. Fitness and Selection

As explained in lecture #5, there are two fundamental forces that form the basis of evolutionary systems: variation and selection. In this lecture, we discuss the EA components behind the second one.

4.1 Population Management Models

In the previous lecture, we focused on the way that potential solutions are represented to give a population of diverse individuals, and on the way that variation (recombination and mutation) operators work on those individuals to yield offspring. These offspring will generally inherit some of their parents' properties but also differ slightly from them, providing new potential solutions to be evaluated. We now turn our attention to the second important element of the evolutionary process – the differential survival of individuals to compete for resources and take part in reproduction, based on their relative fitness.

Two different models of population management are found in the literature: the **generational model** and the **steady-state model**. For the generational model:

- In each generation we begin with a population of size $\boldsymbol{\mu}$, from which a mating pool of parents is selected.

- Every member of the pool is a copy of something in the population, but the proportions will probably differ, with (usually) more copies of the 'better' parents.
- Next, $\boldsymbol{\lambda}$ offspring are created from the mating pool by the application of variation operators, and evaluated.
- After each generation, the whole population is replaced by $\boldsymbol{\mu}$ individuals selected from its offspring, which is called the next generation.

In the model typically used within the **Simple Genetic Algorithm**, the population, mating pool and offspring are all the same size, so that each generation is replaced by all of its offspring.

In the steady-state model, the entire population is not changed at once, but rather a part of it. In this case, $\boldsymbol{\lambda}(\leq \boldsymbol{\mu})$ old individuals are replaced by $\boldsymbol{\lambda}$ new ones, the offspring. The proportion of the population that is replaced is called the **generational gap**, and is equal to $\boldsymbol{\lambda}/\boldsymbol{\mu}$.

4.2 Parent Selection

4.2.1 Fitness Proportional Selection

Fitness proportional selection (FPS) is used for selecting potentially useful solutions for recombination. For each choice, the probability that an individual $\boldsymbol{i}$ is selected for mating depends on its absolute fitness value compared to the absolute fitness values of the rest of the population. Observing that the sum of the probabilities over the whole population must equal $\boldsymbol{1}$. If $\boldsymbol{f_i}$ is the fitness of individual $\boldsymbol{i}$ in the population, its probability of being selected using **FPS** is $\boldsymbol{P_{FPS} = f_i / \sum_{j=1}^{\mu} f_j}$, where $\boldsymbol{\mu}$ is the number of individuals in the population.

There are some problems with this selection mechanism:

- Outstanding individuals take over the entire population very quickly. This tends to focus the search process and makes it less likely that the algorithm will thoroughly search the space of possible solutions, where better solutions may exist. This phenomenon is often observed in early generations, when many of the randomly created individuals will have low fitness, and is known as **premature convergence**.

- When fitness values are all very close together, there is almost no **selection pressure**, so the selection is almost uniformly random, and having a slightly better fitness is not very 'useful' to an individual. Therefore, later in a run, when some convergence has taken place and the worst individuals are gone, it is typically observed that the mean population fitness only increases very slowly.

- The mechanism behaves differently if the fitness function is transposed.

This last point is illustrated in Table 4.1, which shows three individuals and a fitness function with $\boldsymbol{f(A) = 1}$, $\boldsymbol{f(B) = 4}$, and $\boldsymbol{f(C) = 5}$. Transposing this fitness function changes the selection probabilities, while the shape of the fitness landscape, and hence the location of the optimum, remains the same.

Table 4.1: Transposing the fitness function changes selection probabilities for fitness-proportionate selection.

Individual	Fitness for f	Sel. prob. for f	Fitness for $f+10$	Sel. prob. for $f+10$	Fitness for $f+100$	Sel. prob. for $f+100$
A	1	0.1	11	0.275	101	0.326
B	4	0.4	14	0.35	104	0.335
C	5	0.5	15	0.375	105	0.339
Sum	10	1.0	40	1.0	310	1.0

4.2.2 Ranking Selection

Rank-based selection is another method that was inspired by the observed drawbacks of fitness proportionate selection. It preserves a constant selection pressure by sorting the population on the basis of fitness, and then allocating selection probabilities to individuals according to their rank, rather than according to their actual fitness values. Let us assume that the ranks are numbered so that an individual's rank notes how many worse solutions are in the population, so the best has rank $\boldsymbol{\mu - 1}$ and the worst has rank 0. The mapping from rank number to selection probability can be done in many ways, for example, linearly or exponentially decreasing. As with FPS above, and any selection scheme, we insist that the sum over the population of the selection probabilities must be unity – that we must select one of the parents.

The usual formula for calculating the selection probability for linear ranking schemes is parameterised by a value $\boldsymbol{s}$ $\boldsymbol{(1 \leq s \leq 2)}$. In the case of a generational EA, where $\boldsymbol{\mu = \lambda}$, this can be interpreted as the expected number of offspring allotted to the fittest individual. Since this individual has rank $\boldsymbol{\mu - 1}$, and the worst has rank $\boldsymbol{0}$, then the selection probability for an individual of rank $\boldsymbol{i}$ is:

$$P_{lin-rank}(i) = \frac{(2-s)}{\mu} + \frac{2i(s-1)}{\mu(\mu-1)} \tag{4.1}$$

Note that the first term will be constant for all individuals. Since the second term will be zero for the worst individual (with rank $i = 0$), it can be thought of as the 'baseline' probability of selecting that individual.

In Table 4.2 we show an example of how the selection probabilities differ for a population of $\mu = 3$ different individuals with fitness proportionate and rank-based selection with different values of s.

Table 4.2: Fitness proportionate (**FP**) versus linear ranking (**LR**) selection.

Individual	Fitness	Rank	P_{selFP}	$P_{selLR}(s=2)$	$P_{selLR}(s=1.5)$
A	1	0	0.1	0	0.167
B	4	1	0.4	0.33	0.33
C	5	2	0.5	0.67	0.5
Sum	10		1.0	1.0	1.0

4.2.3 Implementing Selection Probabilities

The description above provides two alternative schemes for deciding a probability distribution that defines the likelihood of each individual in the population being selected for reproduction. In an ideal world, the mating pool of parents taking part in recombination would have exactly the same proportions as this selection probability distribution. This would mean that the number of any given individual would be given by its selection probability, multiplied by the size of the mating pool. However, in practice, this is not possible because of the finite size of the population, i.e., when we do this multiplication, we find typically that some individuals have an expected number of copies which is noninteger – whereas of course in practice we need to select complete individuals. In other words, the mating pool of parents is sampled from the selection probability distribution, but will not in general accurately reflect it.

The simplest way of achieving this sampling is known as the **roulette wheel algorithm**. Conceptually this is the same as repeatedly spinning a one-armed roulette wheel, where the sizes of the holes reflect the selection probabilities. In general, the algorithm can be applied to select λ members from the set of μ parents into a mating pool.

To illustrate the workings of this algorithm, we will assume some order over the population (ranking or random) from 1 to μ, so that we can calculate the cumulative probability distribution, which is a list of values $[a_1, a_2, \cdots, a_\mu]$ such that $a_i = \Sigma_1^i P_{sel}(i)$, where $P_{sel}(i)$ is defined by the selection distribution - fitness proportionate or ranking. Note that this implies $a_\mu = 1$. The outlines of the algorithm are given in Figure 4.1.

Despite its inherent simplicity, it has been recognised that the roulette wheel algorithm does not in fact give a particularly good sample of the required distribution. Whenever more than one sample is to be drawn from the distribution - for instance λ - the use of the **stochastic universal sampling (SUS)** algorithm is preferred. Conceptually, this is equivalent to making one spin of a wheel with λ equally spaced arms, rather than λ spins of a one-armed wheel. Given the same list of cumulative selection probabilities $[a_1, a_2, \cdots, a_\mu]$, it selects the mating pool as described in Figure 4.2.

```
BEGIN
  /*  Given the cumulative probability distribution a */
  /*  and assuming we wish to select $\lambda$ members of the mating pool */
  set $current\_member = 1$;
  WHILE ( $current\_member \leq \lambda$ ) DO
    Pick a random value $r$ uniformly from $[0,1]$;
    set $i = 1$;
    WHILE (  $a_i < r$  ) DO
      set $i = i + 1$;
    OD
    set mating_pool[$current\_member$] = parents[$i$];
    set $current\_member = current\_member + 1$;
  OD
END
```

Figure 4.1: Pseudocode for the roulette wheel algorithm.

Since the value of the variable $\boldsymbol{r}$ (in Figure 4.2) is initialised in the range $[\mathbf{0}, \mathbf{1}/\boldsymbol{\mu}]$ and increases by an amount $\mathbf{1}/\boldsymbol{\mu}$ every time a selection is made, it is guaranteed that the number of copies made of each parent $\boldsymbol{i}$ is at least the integer part of $\boldsymbol{\mu} \cdot \ \boldsymbol{P_{sel}(i)}$ and is no more than one greater. Finally, we should note that with minor changes to the code, **SUS** can be used to make any number of selections from the parents, and in the case of making just one selection, it is the same as the roulette wheel.

4.2.4 Tournament Selection

The previous two selection methods and the algorithms used to sample from their probability distributions relied on a knowledge of the entire population. However, in certain situations, for example, if the population size is very large, or if the population is distributed in some way (perhaps on a parallel system), obtaining this knowledge is either highly time-consuming

```
BEGIN
  /*  Given the cumulative probability distribution a */
  /*  and assuming we wish to select $\lambda$ members of the mating pool */
  set $current\_member = i = 1$;
  Pick a random value $r$ uniformly from $[0, 1/\lambda]$;
  WHILE ( $current\_member \leq \lambda$ ) DO
    WHILE (  $r \leq a[i]$ ) DO
      set mating_pool[$current\_member$] = parents[$i$];
      set $r = r + 1/\lambda$;
      set $current\_member = current\_member + 1$;
    OD
    set $i = i + 1$;
  OD
END
```

Figure 4.2: Pseudocode for the stochastic universal sampling algorithm making selections.

or at worst impossible. Furthermore, both methods assume that fitness is a quantifiable measure (based on some explicit objective function to be optimised), which may not be valid. Think, for instance, of an application evolving game playing strategies. In this case, we might not be able to quantify the strength of a given individual (strategy) in isolation, but we can compare any two of them by simulating a game played by these strategies as opponents.

Tournament selection *is an operator with the useful property that it does not require any global knowledge of the population, nor a quantifiable measure of quality. Instead, it only relies on an ordering relation that can compare and rank any two individuals.* It is therefore conceptually simple and fast to implement and apply. The application of tournament selection to select $\boldsymbol{\lambda}$ members of a pool of $\boldsymbol{\mu}$ individuals works according to the procedure shown in Figure 4.3.

```
BEGIN
  /* Assume we wish to select $\lambda$ members of a pool of $\mu$ individuals */
  set $current\_member = 1$;
  WHILE ( $current\_member \leq \lambda$ ) DO
    Pick $k$ individuals randomly, with or without replacement;
    Compare these $k$ individuals and select the best of them;
    Denote this individual as $i$;
    set mating_pool[$current\_member$] = $i$;
    set $current\_member = current\_member + 1$;
  OD
END
```

Figure 4.3: Pseudocode for the tournament selection algorithm.

4.2.5 Uniform Parent Selection

In some dialects of EC it is common to use mechanisms such that each individual has the same chance to be selected. At first sight this might appear to suggest that there is no selection pressure in the algorithm, which would indeed be true if this was not coupled with a strong fitness-based survivor selection mechanism.

In Evolutionary Programming, usually there is no recombination, only mutation, and parent selection is deterministic. In particular, each parent produces exactly one child by mutation. Evolution Strategies are also usually implemented with uniform random selection of parents into the mating pool, i.e., for each $\boldsymbol{1 \leq i \leq \mu}$ we have $\boldsymbol{P_{uniform}(i) = 1/\mu}$.