**Genetic Algorithm (GA)**

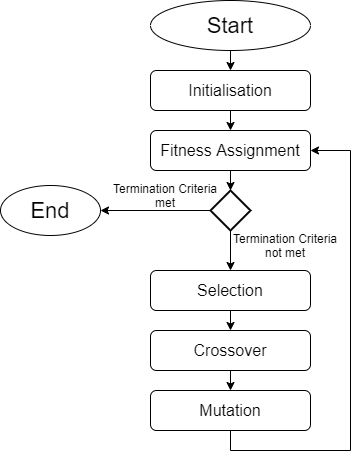
In this Evolutionary Algorithm, each candidate has an identifying ‘chromosomal genome’, which I represented as an n-dimensional vector that encodes each solution, and their ‘fitness’ is encoded as the vector’s function value.

Figure 1. Flowchart of Genetic Algorithm

The basic process of the implementation of GA is detailed in Fig. 1; A population of candidate solutions is initialised by random sample from a uniform distribution. Each candidate is then assigned a fitness and the population is sorted by their fitness ascending. The top candidates are selected as the ‘fittest’, and the recombination operators Crossover and Mutation are applied. This process iteratively repeats until a termination criterion is met.

I have chosen the ‘K-point crossover’ operator as detailed in Genetic Algorithms Reference Vol.1 (Gwiazda and Lomianki, 2006), and for mutation I applied my own ‘Decaying operator’ whereby the candidate has the probability MR to be mutated by adding a noise vector sampled from a normal distribution of mean 0 and variance M. MR and M both decay every iteration by the formula:

Figure 2. Decaying Operator

Where given at a scaled rate according to the maximum number of iterations . It is common to control mutation or crossover rate via a decaying factor (Kühn, Severin and Salzwedel, 2013), in order to aid GA converge to a minima, and in addition this operator is applied in some later algorithms as well. Additional parameters chosen were , following de Jong’s guidelines on having a middle ground between long term and initial response (de Jong, 1975), as well as , and .

GA holds the most ‘faithful’ to the Evolutionary Algorithm format, implementing recombination with the strongest parallels to processes in nature, such as Survival of the Fittest, as well as the distinct processes of crossover and mutation. However, GA does not scale well with dimensionality, uses a relatively high number of function calls and tends to converge towards local minima. In Skiena’s *Algorithm Design Manual* (Skiena, 2010) he states:

*“The pseudobiology [in GA] adds another level of complexity between you and your problem […] [T]he analogy with evolution—where significant progress require [sic] millions of years—can be quite appropriate […] Stick to simulated annealing for your heuristic search voodoo needs”*

However, this quote most likely relates to the use of combinatory optimisation, as was discussed earlier on in that chapter, and therefore I will be testing Skiena’s claim on continuous optimisation.

**Differential Evolution (DE)**

Although DE holds a similar structure to GA in that it is an Evolutionary Algorithm, there are a few major differences. Instead of utilising a top percentile of the population as parents, a ‘mutant’ solution is generated for each candidate; if it is fitter then it replaces the candidate. Also, in place of GA’s mutation and crossover schema, DE implements a more complex recombination operator, by mutating a candidate by the vector difference of two other random candidates. I employed the ‘rand1bin’ variant of the recombination process, as detailed below:

Figure 3. DE “rand1bin” recombination process

where CR is the Crossover Rate and F is the Differential Weight. The above process means that initially, as the random population has a high variance, candidates are mutated by a greater magnitude, and as the candidates are converging to a minima, the magnitude of mutation decreases.

I also employ dithering, a technique detailed in *Differential Evolution: A Practical Approach to Global Optimization* (Price, Storn, Lampinen, 2005) whereby F is selected uniformly randomly in the range [0.5,1.0] for each generation, which “improves convergence behaviour significantly, especially for noisy functions” (Storn, 2013). Other parameters chosen were , .

DE utilises a mathematical representation of a solution in place of working around a chromosomal representation as in GA. It proves to be typically more computationally efficient and is able to converge more precisely to a minima than GA, and it also performs well on Additively Separable Functions, ie. due to axis-orthogonal steps produced in crossover (Sutton, Lunacek and Whitley, 2007). However, like GA, it has many parameters to tune optimally, and requires a large population.

**Covariance Matrix Adaption Evolution Strategy (CMA-ES)**

Although it can be loosely classed as an Evolutionary Algorithm due to its vague structure, as well as its use of population and notion of evolution, CMA-ES holds little similarities to traditional Evolutionary Algorithms in that it is based almost entirely mathematically rather than a nature-inspired metaphor, and operates with different mechanisms.

Populations are sampled from multivariate normal distributions with mean and covariance matrix updated as to maximise the likelihood of successful candidates in the next generation (Hansen, 2016); it achieves this in essence by calculating the variance of the top percentile of candidates whilst utilising the mean of the current generation. The algorithm also records its “Evolution Path” to track the algorithm’s progress and alter the course of search accordingly, e.g. If the solutions head in the same direction every step, then step size is increased to cover more distance in less time, reducing time taken to reach a minima.

CMA-ES is mainly used in reinforcement policy search within the field of machine learning (Gomez et al., 2008), and according to Hansen performs well on “difficult functions” or in higher dimensionality (Hansen, 2010), perhaps due to the fact that the population used is relatively small, so less time consuming fitness evaluations occur, as well as having fewer parameters to choose. In addition, the updating of these parameters has been found to emulate a form of Natural Gradient Descent (Akimoto et Al, 2010), which factors in uncertainty in solution when updating, reducing the chance of premature convergence.

For implementation, I utilised and modified the ask-and-tell interface in Hansen’s own ‘pycma’ module, release 3.03 (Hansen, 2020), due to time constraints and the complexity of the algorithm, with starting .

**Particle Swarm Optimisation (PSO)**

PSO utilises a population of ‘particles’ which move around the search space according to mathematical formulae governing their motion, ie. their position and velocity. Each particle moves in a weighted magnitude of the directions of their personal best position and the global best-known position, which are updated each iteration and this has the overall effect of the particles ‘swarming’ towards the minima. The iterative formulae of motion are shown below:

Figure 4. PSO formulae of motion

where i refers to the ith particle, w, cp and cg are weights for velocity, personal best and generational best influence respectively, and rb and rg are uniformly random sampled weights in the interval [0,1]. For implementation I chose the following parameters: population size 60, w=0.5 and

PSO was first intended for simulating social behaviour (Kennedy, 1997) but was later adapted for optimisation purposes, this can be seen in Kennedy and Eberhart’s book *Swarm Intelligence* where the creators of the algorithm, write:

*“A very simple sociocognitive theory underlies … particle swarms”*

Kennedy further elaborates on the psychological and philosophical roots of PSO in the chapter, mentioning inspiration from their Adaptive Culture model and Festinger’s social comparison theory; it is clear that the algorithm is more heavily inspired by the metaphor of swarm intelligence compared to a mathematical basis. In practice, PSO has a relatively low convergence rate and tends to converge to local minima; more concretely the algorithm has been proven that it is not guaranteed to locate a global optimum (Bonvadi and Michalewicz, 2014).

**Bat Algorithm (BA)**

BA utilises a population of microbats flying with random velocities that adjust the frequency of echolocation pulses depending on their proximity to a solution. In addition to this, new solutions for each bat are generated using random walk with magnitude equal to the echolocation’s amplitude, if a uniformly random variable is greater than the ‘mutation rate’ r. Similar to mutation in Evolutionary Algorithms, the new bat is accepted if its fitness is lower than the original solution, and another uniformly random variable is greater than the amplitude. The corresponding formulae are shown below:

Figure 5. BA formulae of motion and mutation

where and are uniformly random vectors in the interval [0,1], is the current global best location and is the average amplitudes at a time step. I also varied A and r via the following processes as seen in Yang’s later paper (Yang and He, 2013):

Figure 6. BA A and r variation

where and are constants. I have utilised the paper’s recommendation . Other parameters chosen include , , , and population size 30.

BA can be viewed as an extension or generalisation of PSO, where BA becomes PSO *“when we replace the variations of the frequency by a random parameter, and setting = 0 and = 1”* (Yang, 2014). Unlike PSO, BA utilises Frequency tuning, Automatic zooming to a promising region in the search space and Parameter control, and has proven guaranteed global convergence (Huang, Zhao and Lu, 2013). BA remains relatively faithful to its source of inspiration, mimicking real-life processes and behaviour of bats, however these processes are formulated and modified specifically to adapt the algorithm for optimisation.

**Simulated Annealing (SA)**

Simulated Annealing borrows inspiration from annealing in metallurgy, where the concept of a material slowly being cooled is interpreted as a slowly reducing probability of worse solutions being accepted. The temperature iteratively cools to 0 via the Decaying operator as shown in Figure 1., a random “neighbour’ of the current solution is selected and if the transition probability , where r is a uniformly random variable, the neighbour becomes the new current solution, and the process repeats. For neighbour selection I employed a similar process to GA mutation, ie. a new solution is generated via adding a vector sampled from a normal distribution of mean 0 and variance M to the current solution, where M varies by the Decaying operator in Figure 1.

The process for decreasing p is modified from a Boltzmann distribution (Schulte, Horst, Eckhard 2017), shown below:

Figure 7. SA p and energy update

where is the number of accepted solutions. Other parameters selected include , . SA sticks close to the metaphor of annealing but also has flexibility in the interpretation of “cooling” schedules for temperature and transition probability, however it tends to be computationally expensive due to the slow speed of cooling requiring a large amount of function calls.