Chapter 4

An introduction to genetic algorithms

4.1 The inspiration

A genetic algorithm (GA) is an optimisation algorithm initially developed by John Holland and his students and colleagues as the University of Michigan through the 1960s and 1970s. Holland's 1975 book *Adaptation in Natural and Artificial Systems* is the classical book on GA's.

The GA was inspired by Darwin's theory of evolution (Darwin, 1959); a process that lends itself to optimisation quite naturally, being a mechanism by which organisms change over time in response to natural selection so as to become better adapted to their environment. The process of evolution is successful because individuals that are the 'fittest' i.e. best suited to their environment, are more likely to survive and hence pass on ideal traits to the following generation. There is also the mechanism of mutation, which introduces new traits into the gene pool ensuring diversity within the population. In this way, evolution produces organisms optimal for the environment.

4.2 The canonical GA

```
1  Create an initial population of candidate solutions.
2  REPEAT
3          EVALUATION: evaluate the fitness of each member of the population.
4          SELECTION: select members of the population to enter into the mating pool.
5          INVERSION: reorder the genes in the chromosome.
6          CROSSOVER: from the mating pool, produce new solutions.
7          MUTATION: perform random changes to a random selection of the population.
8          REPLACEMENT: members of the old population are replaced with individuals
9          from the crossover and mutation steps.
10 UNTIL stopping criteria is satisfied.
```

The GA endeavours to mimic these evolutionary mechanisms to evolve solutions to a given optimisation problem. While the mechanisms that govern the GA can vary enormously, the fundamental purpose of the operators is largely preserved. There is a lot of freedom to design each mechanism to suit the given problem but it also means that the task of designing the GA and setting the parameters for the simulation is an optimisation challenge by itself.

Candidate solutions to the problem are analogous to the individuals in a population, and are described by a set of genes that represent the problem parameters. In Darwin's survival of the fittest, the term 'fitness' is used very broadly and can encapsulate any trait that promotes survival and later reproduction. Such characteristics include: a moth's ability to camouflage, an eagle's ability to spot prey and a peacock's ability to attract a peahen. In a GA, the fitness is a measurement of the strength of a candidate solution, where typically a better solution is associated with a higher fitness score.

Better solutions are 'evolved' through the process of selection, crossover and mutation. Selection is the stage where good solutions are chosen from the population to go into a 'gene pool'. These solutions go on to produce new solutions by crossover (also referred to as recombination), in which the parameters of the descendent solution(s) are derived from those of the parent with the goal of producing better solutions. Mutation is a way

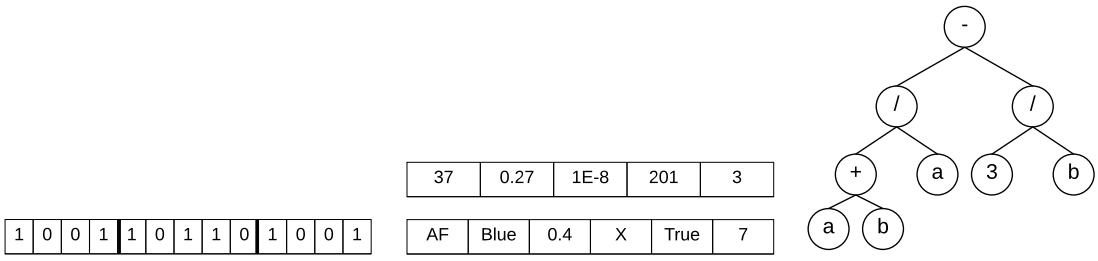


Figure 4.1: Examples of different ways of encoding a chromosome. (a) binary encoding. Thick lines partition the chromosome into genes. (b) real-valued encoding. In the first chromosome, each gene value is numeric whereas in the second there is a mixture of numeric, categorical and boolean values. (c) Tree encoded chromosome. This encodes the expression $\frac{a+b}{a} - \frac{3}{b}$. When tree encoding is used, the procedure is referred to as a genetic program.

of introducing new characteristics into the population by altering parameters as a means to further explore the solutions space. At the end of each epoch, the parent generation expires.

In the canonical GA, Holland introduced an inversion operator, which randomly changes the order of genes in chromosome to give correlated genes a better chance of staying together during crossover. However most GA's do not use this operator because it does not provide much improvement (Goldberg, 1989).

4.3 Details on the mechanisms

4.3.1 Encoding the population

Parameters of a problem are encoded in *chromosomes* as genes. Each chromosome represents a full specification of a candidate solution. The traditional form of encoding is binary encoding (Holland, 1975). This is where an individual is represented by a string of bits; sections of bits make up a gene (Figure 4.1a). More commonly used however is real-value encoding where chromosomes are an array of values, which can either be numeric or categorical (Figure 4.1b). These are more efficient (Michalewicz, 1994) and are more compatible with other optimization alogorithms, making hybridisation easier (Haupt and Haupt, 1998). Another form of encoding is tree encoding (referred to as a genetic program) where attributes are stored in a tree structure (Figure 4.1c). Since the traditional operators of selection, inversion, crossover and mutation (Holland, 1975) were designed for binary encoding, these need to be adapted if other forms of encoding are to be used.

A set of chromosomes makes up a population. It is important to have a large enough population to adequately explore the solution space. However, if it is too large, convergence is slow. Typically population sizes of order 10 to 1000 are used (Mitchell, 1998; Sivanandam and Deepa, 2007).

4.3.2 Evaluating a solution

A candidate solution is evaluated by a *fitness function*, which measures how good a solution is for the tasked problem. Formulating a good fitness function can be challenging, as different features of the ideal solution may need to be unequally weighted. (e.g. when evolving a creature that can swim the fastest, it may be more important to have bigger flippers than to be streamlined). The fitness function may also need to account for correlation between features.

While a better fitness score should be indicative of a better solution, it is not necessarily true that the optimal solution has the best fitness score. It is however important for the success of the GA to have a well chosen fitness function because the GA will optimise based on the fitness score.

4.3.3 Selection

The selection process aims to mimic natural selection where individuals that are unfit do not survive. In the selection stage, the fittest individuals selected to survive and enter into the mating pool where they will have the opportunity to reproduce. Selection schemes can be classified as a *proportional selection* where selection is proportional to the fitness score or *ordinal selection* if it depends on the rank of the fitness score.

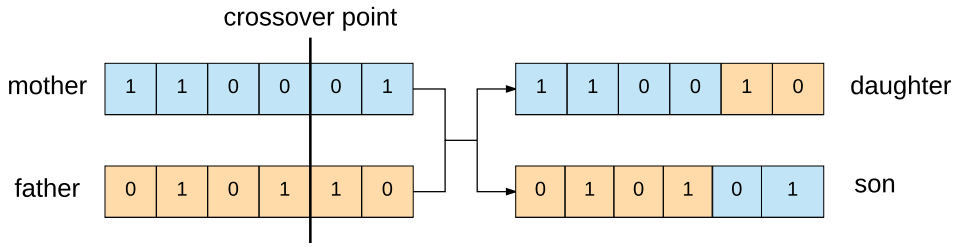


Figure 4.2: Illustration of the single point crossover method with binary encoded chromosomes.

When designing the selection mechanism *selection pressure* needs to be taken into account. The selection pressure describes the degree of bias towards fitter chromosomes. A high selection pressure highly favours the individuals with the best fitness score. While this may greatly improve the overall fitness of the following generation, the population is at risk of becoming homogenous and result in a poor exploration of the solution space. If however the selection pressure is too low, evolution can be quite slow.

4.3.4 Crossover

The crossover operator combines to chromosomes to produce an offspring, in an artificial, sexual reproduction process. This is crucial for producing variation within the population without decreasing the populations fitness (Mackay, 2003). The fundamental crossover method is the single point crossover (Holland, 1975) (Figure 4.2), which produces a daughter and son from a mother and father chromosome. Note that individuals do not have an identifiable sex, and the terms mother and father as well as daughter and son are interchangeable. In the single point crossover, a crossover point is selected at random. The genes before the crossover point are passed from mother to daughter and father to son. The genes after the crossover point are passed from mother to son and father to daughter. An extension of the single point crossover is the multi point crossover where multiple crossover points are selected.

There is a lot of room for creativity when designing a crossover method. Other forms of crossover use an averaging technique between the two parents or may use different numbers of parents and produce a different number of offspring. The crossover mechanism strongly depends on the problem and the encoding. A great resource outline different forms of crossover is Sastry et al. (2005).

4.3.5 Inversion

Inversion (Holland, 1975) is a means of shuffling genes such that related genes can be kept together. Since the ordering of the genes should not affect the fitness of the chromosome, the chromosome 11 00 01 10 can be represented as:

$$[(1,1),(2,1),(3,0),(4,0),(5,0),(6,1),(7,1),(8,0)]$$

where the first number in each pair represents the location of the gene and the second number is the allele. In inversion, two numbers are selected, and the position of the bits inside are reversed, for example, selecting 4 and 7, the chromosome above would be represented as 11 01 10 00:

$$[(1,1),(2,1),(3,0),(7,1),(6,1),(5,0),(4,0),(8,0)]$$

If the combination of genes 11**01** correspond to a high fitness, then reordering of the chromosome to 1101**** may mean that this combination is more likely to remain undisturbed by effects of crossover (e.g. less likely to be disturbed by the single point crossover). However this operator has not been shown to offer much improvement (Goldberg, 1989) and so, is not usually used.

4.3.6 Mutation

In nature, mutation occurs when new characteristics are introduced into a species by chance, and plays a crucial role in ensuring variance within the population. This is mimiced by the mutation operator in a GA, which is a means of introducing new genes into the population, thus allowing for further exploration of the solutions space. Traditionally this was done by flipping bits at random in a chromosome with some small *mutation rate* e.g. 0.001

(Mitchell, 1998). The mutation is of course different for different types of encoding. For real valued encoding, a parameter value is usually changed by some arithmetic operation.

Choosing a mutation rate can be difficult. If the mutation rate is too high, then the GA will tend to behave as a random walk. If the mutation rate is too low, the population may prematurely converge to a sub-optimal solution and may be unable to embark on further exploration.

4.3.7 Stopping criterion

There are a number of ways to terminate a GA and the chosen criteria will depend on the goal of problem.

Common termination criteria are:

- Allowing the simulation to run for a set number of epochs
- Allowing the simulation to run for a certain length of time
- Stopping the simulation once the fitness score has reached some threshold
- Stopping the simulation when the change in the fitness score over some number of epochs is below some threshold

4.3.8 Tuning the GA

A GA can be very complex in design and there are many parameters that need to be tuned to optimise the performance of the GA. Firstly, the mechanisms of each operation needs to be decided, i.e. determining how selection, crossover and mutation will be done. There are also parameters within these methods that will need to be tuned such as the selection pressure and the mutation rate. These parameters can either be static or dynamic. For example, it can be good to have a high selection pressure and low mutation rate at the start of the simulation, then, as the population starts to converge, reduce the selection pressure and increase the mutation rate to slow convergence.

Another consideration is the size of the population. It needs to be large enough for adequate exploration of the solution space, but if it's too large, evolution will be slow (Sivanandam and Deepa, 2007). Some GAs even change the size of their population dynamically so that initially the population is quite large so as to inspect the solution space, and then reduces the population once exploration becomes confined to regions of high fitness. Furthermore the proportion of the new generation that are survivors of the old generation or are produced by crossover or mutation need to be decided, as well as whether mutant individuals are spawned from the original or replace it.

4.4 Comparison with other algorithms

There is no general scheme that can specify which optimisation algorithm will best suit any given problem. An exhaustive search is practical for a small solution space and a small cost function provided the search space can be sampled at small enough intervals (Haupt and Haupt, 1998). For a landscape that is both smooth and unimodal a local optimiser such as a steepest-ascent hill climb will be much more efficient (Mitchell, 1998). A GA is typically used when the search space is large, not perfectly smooth, not unimodal and not well understood (Mitchell, 1998).

GA's are a stochastic process and are not necessarily guaranteed to find the global optimum. Their performance will depend on the tuning parameters. GA's are not necessarily guaranteed to find the best solution, but are well trusted to find a good solution (Sivanandam and Deepa, 2007). GA's are very general so any specialised algorithm is almost guaranteed to outperform them (Sivanandam and Deepa, 2007). However, GA's can be hybridised with other specialised techniques to improve performance. Generally this is done by making use of a local optimiser in the vicinity of the optimum found by the GA.

Probably what is most attractive about GA's are that the fitness scores can be evaluated in parallel, which cannot be easily achieved with a Markov Chain Monte Carlo, a nested sampler or in simulated annealing. There are a number ways parallelisation can be achieved. Probably the most simple to implement is the master-slave arrangement (Goldberg, 1989). In this architecture, one computer, called the master, controls the evolutionary operators of selection, crossover and mutation and designates the task of evaluating the fitness score (generally the computationally expensive operator) to other computers called 'slaves'.

The main disadvantage of GA's is that they do not estimate uncertainties as they do not sample the underlying probability distribution function (PDF). However, an optima located by a GA can be used to initialise a Markov

Chain Monte Carlo (MCMC), which can then explore the parameter space more efficiently. The hybridisation of the two avoids an unnecessarily long burn-in with a pre-informed optima and incorporates probability, allowing the uncertainties to be obtained.

4.5 Applications of GAs in astrophysics

GA's are very robust and outperform most other optimisation algorithms on complex and ill-behaved workspaces (Goldberg, 1989). It happens that there are many challenging optimisation problems in astrophysics typically involving data fitting and error minimisation. While GA's have been widely in computer science and artificial intelligence, the application of GA's to astrophysics-related problems has only recently become popular (Charbonneau, 1995).

A popular package is PIKAIA (Charbonneau, 1995), which implements a general purpose genetic algorithm written in FORTRAN-77. PIKAIA has been used in areas such as helioseismology (Charbonneau and Tomczyk, 1997), modelling pulsating white dwarves (Metcalfe et al., 2000) and modelling dust spectra (Baier et al., 2011, 2010) where it was used in conjunction with the radiative transfer code DUSTY. GA's have also been used for finding planetary companions to pulsars (Lazio, 1997), fitting magnetosphere models to the solar minimum corona (Gibson and Charbonneau, 1998), modelling galaxies (Wahde, 1998; Wahde and Donner, 2001; Attia et al., 2005) and modelling protoplanetary discs (Hetem and Gregorio-Hetem, 2007). It is important to note that GA's can be hindered by degeneracies (Hetem and Gregorio-Hetem, 2007) or insufficient tuning parameters (Baier et al., 2011).

Chapter 5

Implementation of the GA

The code developed during this thesis is available at a GitHub repository at `https://github.com/awon8465/awongHonsThesis`*. I invite interested readers to examine this and apply it to their own scientific questions.*

5.1 Generalising the GA

A GA should be easily extendable to a number of optimisation problems. The operations and mechanisms of the GA should be independent of the problem; the only difference between the application of GAs to different problems is the encoding of the candidate solutions into chromosomes and the method of evaluating the fitness score.

Our GA was implemented in PYTHON3. An object oriented approach was used because it provided an intuitive way to create the individuals of the population and to track their attributes. Individuals are real-value encoded. As long as the representation of the chromosome is consistent, the GA should be easy to adapt to solve problems other than the one in this thesis. The user only needs to initialise the chromosomes and write an evaluation function.

Since there are lots of different parameters that control the GA, there is a settings file where all these settings are specified. This compartmentalising allows the GA to be run without the user modifying much of the code. The GA has also been written to accomodate both maximisation and minimisation problems, which is specified in the settings file by one of the following:

```
problem type,        minimise
problem type,        maximise
```

The structure of the implemented GA is given in Figure 5.1 with details provide in Section 5.3.

5.2 The individual

Candidate solutions are defined by their chromosome set, but it's often useful to associate other values with the chromosome. Hence we defined a class *individual* (Figure 5.2). The individual consists of a chromosome, which is a set of instance attributes assigned the parameter values. These names of these parameters are defined in the settings file. For example, the following gives the GA the 3 parameters `dust_mass`, `rin` and `radius`.

```
parameter list, dust_mass, rin, radius
```

Users also need to provide the lower and upper bound values for these parameters, which defines the parameter space (in standard MCFOST units). Note that these need to be in the same order as the parameter list.

```
lower bounds,         1.0E-8, 1.0, 1
upper bounds,         3.0E-5, 10.0, 800
```

The initial population is initialised with chromosomes randomly generated within this parameter space. Chromosomes are later generated by operations of the GA.

It is possible that individuals may evolve to explore outside the initialised parameter space. Depending on the task, it may be necessary to confine the individuals within the initial boundaries to ensure that the chromosome is valid. There are three methods of the individual class to deal with illegal exploration. The first is to create a new chromosome randomly placed inside the initial parameter space. In doing this however, we may lose good genes. So there is also the option to only reinitialise the illegal parameters either within their bounds or to the closest boundary. Alternatively, it may be desirable to allow individuals to explore beyond the boundaries if the bounds of the parameters are unknown. The validation method is selected in the settings file by specifying one of the following:

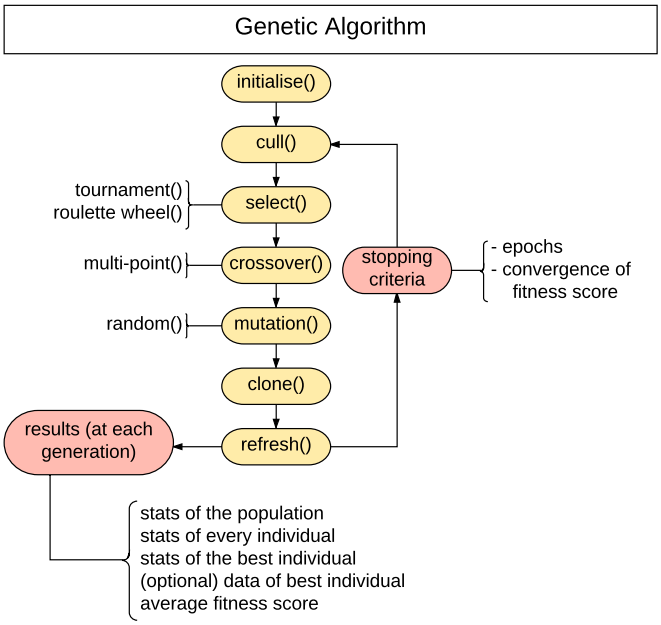


Figure 5.1: Structure of the GA. More detail in Section 5.3.

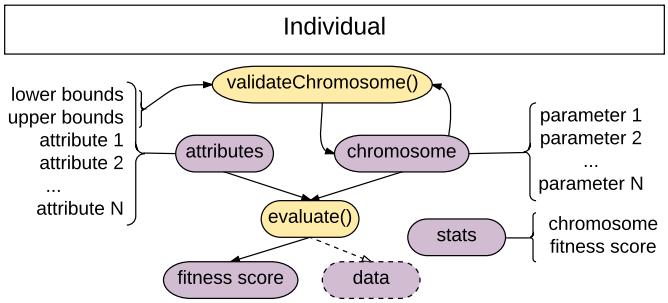


Figure 5.2: Structure of the Individual class. More detail in the Section 5.2.

```
stray, nothing
stray, para to boundary
stray, para randomise
stray, ind randomise
```

The user also has the freedom to give the individual other useful attributes. The easiest way to do this is to declare them as class attributes, which means all individuals will share these attributes.

The user will need to write an `evaluate()` function and assign it to the Individual class. The function is a class method of the `individual` class. It is required to calculate a fitness score of a given individual and save this in the instance attribute `fitScore`. While the evaluate method is primarily used to obtain fitness scores, it can also be used to inject other problem-specific operations. For example, it can be used to assert additional parameter constraints. Users can also define the instance attribute `data`, which will be passed out to the results if `save data` is `T` in the settings file. This is how attributes of the individual can be accessed after the simulation. Unless it is saved in `data`, the values will be lost once the simulation ends.

The final attribute of the individual is `stats`. This is simply an array that consists of the chromosome and the fitness score. The `stats` of the individuals are passed out to the results at each generation and can be accessed after the simulation.

5.3 The GA

5.3.1 Initialisation

There are a number of settings that need to be defined in the settings file. One of the first is to define a population size.

```
population size,        30
```

Following that we need to define a number of rates. These tell us how much of the population is produced by crossover, mutation and cloning and how many individuals are culled (more details on these operators later in this chapter). We define a crossover rate, which is the percentage of the population produced by crossover. This value is rounded up if there are mutants and rounded down if there are not.

```
crossover rate,         0.6
```

The clone rate tells us how much of the population are produced by cloning the best individuals from the previous generation. This is always rounded up.

```
clone rate,          0.2
```

The cull rate tells us how many of the least fit individuals are culled at the start of a generation. This is always rounded down.

```
cull rate,          0.3
```

The remaining individuals are produced by mutation.

5.3.2 Selection

Selection is the operation that chooses individuals from the previous generation to enter into the *mating pool* in the hope that they produce offspring that are, in general fitter. There are two methods of selection implemented; tournament selection and the roulette wheel. Tournament selection is based on the rank of an individuals fitness score so it is classed as an *ordinal selection*, while roulette wheel selection is proportionate to the fitness score of the individual, so it is a *proportional selection*. The method of selection is specified in the settings file.

In tournament selection p participants are chosen from the population and there is a specified win probability w which is the probability that the fitter individual will win. Starting with the individual of the highest rank, each individual will 'win' the tournament with probability w. Once there is a winner, the tournament is terminated and the winner is entered into the mating pool. If no one wins, the individual of lowest rank goes into the mating pool. Individuals can be entered into many tournaments but can only enter a given tournament once. The selection pressure can be changed by changing p or w. Typically, p = 2 and w is around 0.75 (Mitchell, 1998).

In roulette wheel selection, a roulette wheel is constructed where each slice represents an individual of the population and the area is proportional to its fitness score. For a maximisation problem, the area of the individual i is calculated by taking the fitness score of the individual and dividing by the total fitness of the generation.

$$\text{Area}_i = \frac{f_i}{\sum_{k=1}^{N} f_k} \tag{5.1}$$

For minimisation problems, the fitness score first needs to be inverted. One way to invert the fitness score is to subtract the fitness score from the maximum fitness score of the generation and adding 1 (Cox, 2005).

$$f' = (f_{\text{max}} - f_i) + 1 \tag{5.2}$$

This fitness can then be used with Eqn. 5.1 to determine a corresponding area on the wheel.

To select N individuals the roulette wheel is spun N times with a single marker, and whichever individual the marker lands on enters into the gene pool. One issue with this is that usually the population of the GA is quite small, so while unlikely, it is possible that all spins select bad to mediocre individuals. *Stochastic universal sampling* (Baker, 1987) was proposed to counter this problem. Instead of spinning the wheel N times, N markers are placed uniformly around the wheel and the wheel is spun once (Figure 5.3). Each marker then selected one individual to enter into the mating pool.

The main issue with roulette wheel selection is that initially, a small percentage of the population have comparatively excellent fitness scores and so will dominate the wheel, which results in a high selection pressure. As the population converges, the selection pressure decreases because all individuals will have similar fitness scores.

The selection method is specified in the parameter file. For tournament selection, you need to specify r and then p.

```
selection method, tournament, 0.75, 2
```

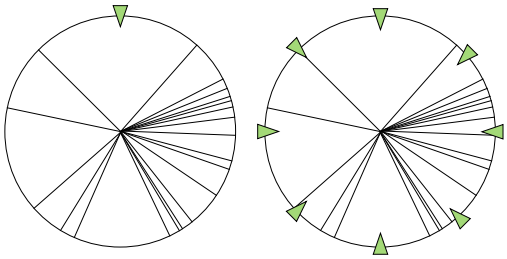


Figure 5.3: The roulette wheel selection. Each slice on the wheel represents an individual and the area corresponds to their fitness score. **Left**: Typically the roulette wheel is spun N times and the individual under the marker is selected for the mating pool. **Right**: *stochastic universal sampling* (Baker, 1987). The wheel is spun once with N uniformly distributed markers.

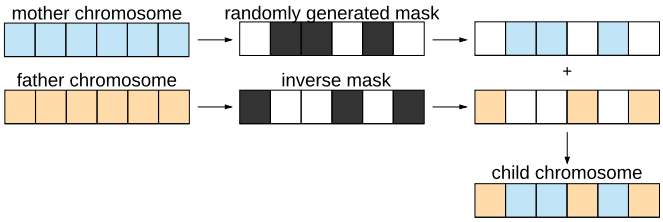


Figure 5.4: Variant of the multi-point crossover.

For roulette wheel selection, if no additional arguments are given, the wheel will be spun N times with a single marker.

```
selection method, roulette
```

For stochastic universal sampling, 'SUS' needs to be provided as an additional argument.

```
selection method, roulette, SUS
```

5.3.3 Crossover

Crossover aims to mimic sexual reproduction. Child chromosomes are produced from parent chromosomes. We have implemented a variant of the multi-point crossover. In this version a mask is randomly generated. The mask selects at minimum one parameter, and at most, all but one. The parameters selected by the mask are taken from the mother chromosome. The remaining parameters are taken from the father chromosome (Figure 5.4). In this method, no new genetic material is introduced.

Also implemented is soft brood selection. In general this is more commonly used with genetic programs (GPs) but are still applicable to GAs. Soft brood selection was proposed by Altenberg (1994) as a means to increase 'evolvability' - ability to evolve fitter individuals. In nature, not all potential offspring make it to birth or far beyond it. Some offspring are eaten, some are neglected and some are just not born at all. In brood selection b potential offspring are produced, but only s of these offspring survive. The ones that survive are the ones with a higher fitness (Figure 5.5).

Brood selection also helps to reduce the destructive tendencies of crossover due to the crossover points being random, as it increases the probability that the offspring are fitter than their parents (Tackett and Carmi, 1994).

In the parameter file, the brood number and survival number need to be given. Below specifies a brood number of 3 and survival number of 2.

```
crossover method, multi-point, 3, 2
```

5.3.4 Mutation

Mutation is the means by which new genes are introduced into the population. We use random mutation for its ease of implementation. Mutants could be produced from the gene pool or from the previous generation. The advantage of producing mutants from the gene pool is that they should, on average, have a higher fitness. This choice is made by specifying either:

```
mutation selection, prev gen
```

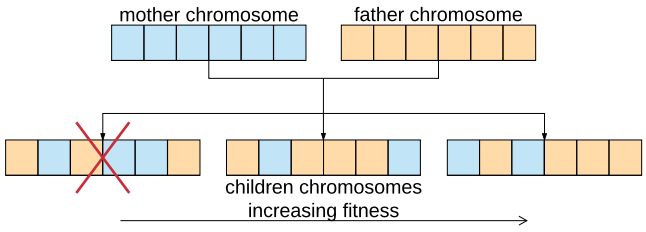


Figure 5.5: Brood selection (Altenberg, 1994) for a brood size of 3 from which 2 of the fitter children will survive.

or by

```
mutation selection, gene pool
```

Currently, the only option is for random mutation.

```
mutation method, random
```

The random mutation used was taken from Tang and Tseng (2013). When producing a mutant, all the parameters in the chromosome are changed. For a parameter p_i, the mutated parameter is:

$$p_i' = p_i + \Delta(r - 0.5) \tag{5.3}$$

where r is a random number chosen uniformly between 0 and 1, and

$$\Delta = \max[2(p_i - pi^{LB}), 2(p_i^{UB} - p_i)] \tag{5.4}$$

pi^{LB} and p_i^{UB} are the smallest and greatest values of the parameter p_i of the current generation. Note that our mutants do not replace the original individuals, rather they are spawned.

5.3.5 Culling

It is often convenient to remove highly unfit individuals to prevent their genetic material infecting the rest of the population. With a high enough selection pressure the least fit individuals will not be selected for reproduction. However, one method of guaranteeing they are not selected is to cull the least fit individuals.

To cull, specify a percentage of the population with the lowest fitness scores. This is always rounded down.

```
cull rate, 0.3
```

5.3.6 Elitism

In the elitist model (De Jong, 1975) the best individual is saved in each generation and in general significantly improves the performance of the GA (Mitchell, 1998). The act of 'saving' an individual ensures that good genes are preserved so the GA doesn't waste time locating them again. However, elitism improves local search at the expense of a global search (De Jong, 1975). Supposing that the protected individual is situated at a local optima, at each generation it will have the highest chance of any individual to pass on its genetic material via crossover, so the population will progress towards the local optima it has located. The GA implements elitism by specifying a percentage of the population that should be cloned at each generation. The number of individuals s that are cloned is always rounded up and the s fittest individuals are copied. Note that these individuals are not re-evaluated.

```
clone rate,          0.05
```

5.3.7 Stopping criterion

There are two stopping criteria. The first is to stop the simulation once a maximum number of generations/epochs have been reached. Note that this includes the initial generations. For example, 5 epochs means the initial population and 4 following generations.

```
stopping criteria, epochs, 50
```

The second stopping criteria is based on the minimum change in fitness of the best chromosome of each generation, i.e. the best solution is not getting much better. A threshold and a minimum number of generations needs to be specified. For example, a threshold of 0.01 over 10 generations would be given as:

```
stopping criteria, best, 0.01, 10
```

This checks the fitness score of the best chromosome in the current generation compared with that 10 generations previous. If the change in fitness is less than the threshold of 0.01, the simulation ends. Note that the simulation will run for at minimum of 10 generations.

5.3.8 Refresh

At the refresh stage, the previous population is overwritten with the current population and the current population gets wiped.Also at this time, a number of values are saved to a results folder, which is specified in the parameter file.

```
results folder, <folder name>
```

All the results are saved with the extension `.npy` which means it is in the form of a numpy array. At each generation the following are saved:

folder	file name	description
allIndividuals	ind_i.npy	all the individuals *created* and evaluated in this generation. Includes individuals that have not survived brood selection.
popAtEachGen	pop_i.npy	all the individuals that make up the current population. Includes those that have been cloned but not those that did not survive brood selection.
-	bestInd.npy	the stats of the best individual of the generation.
data	data_i.npy	(optional) a user defined attribute of the best individual of the generation.
-	aveFitScore.npy	the average fitness score of the generation.

Table 5.1: Results saved at each generation.

Note that saving 'data' is optional. Data is only saved if the save data option is set to true.

```
save data, T
```

Also saved into the results folder is a copy of the settings file and the simulation information, which tells you population size and the number of individuals produced by crossover, cloning and mutation as well as the number of individuals culled.

There is a results class, which will load in all the data saved in the specified results folder. Provided are some in-built display functions such as plotting the average and best fitness scores by each generation, plotting a 2D or 3D errorspace by passing in the parameters for each axis and using colour to show the fitness score. Similarly, instead of showing the errorspace you can plot the progression of the population where colour shows generation instead of the fitness. Other visual representations can be written by the user.

5.3.9 Add-ons

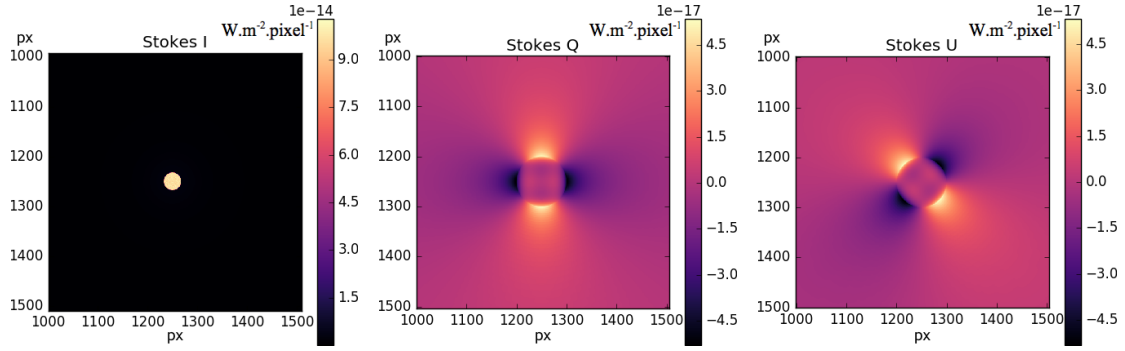
The GA has specifically been written to make it easy to add in new mechanisms for each operator. For example, a new selection method can be added as long as it follows the same structure as the other selection methods. It simply needs to create a list of individuals called `self.genePool`, which is of length `self.genePoolN`. Similarly, as long as the structure is consistent, new mechanisms for crossover and mutation can be easily integrated in.

5.4 Using the GA to fit MCFOST models to the R Leo data

The GA was used to fit the R Leo data obtained using the VAMPIRES instrument (Figure 3.8). The individual was defined by a set of parameters defining an MCFOST stellar model (Section 5.4.1). Artificial data corresponding to the VAMPIRES data was generated (Section 5.4.2) and the fitness score was defined using the reduce χ^2 error of the R Leo data (with the bias removed) and the model data.

5.4.1 Making models: MCFOST

Radiative transfer models can be produced using the software MCFOST (Pinte et al., 2006, 2009). MCFOST runs a Monte Carlo radiative code used for modelling circumstellar environments. We produced 3 dimensional



a b c

Figure 5.6: Example output of the Stokes I, Q, U images from MCFOST. The original map size was 2501 × 2501px but has been cropped to 1000:1500 in both axes. The inner radius of the dust shell is clear in the Stokes Q and U images, as a large amount of light is scattered here. The dust shell extends beyond the depicted region.

MCFOST parameters	Units	Values
stellar radius	$R_\odot$	150
	mas	6.98
distance	pc	100
temperature	K	3000
mass	$M_\odot$	1.0
dust mass	$M_\odot$	3.5E-6
inner radius of dust shell (r_{in})	AU	2.0
	mas	20.0
outer radius of dust shell (r_{out})	AU	300
	mas	3000
grid	px	2501
map size	AU	150
	mas	1500
x	AU	0
y	AU	0
z	AU	0

Table 5.2: Some of the MCFOST parameter values used for the production of Figure 5.6

envelope models using a spherical geometry. Scattering is calculated using Mie theory (Mie, 1908) i.e. spherical grains. The inner and outer radius of the dust shell could be controlled and had a hard edge. It is assumed that both radiative and local thermal equilibrium is reached although the gas and dust are not in hydrostatic equilibrium. The star is modelled aa a blackbody corresponding to the specified effective temperature. For a star centred at the origin, the image has vertical, point and axial symmetry. To create asymmetries, the star is offset from the origin while the dust remains centred. In this case there are no assumed symmetries.

MCFOST reads in properties of a star from a parameter file and observables are made using ray tracing methods. MCFOST outputs polarised images corresponding to the Stokes parameters I, Q, U and V. Note that only the Stokes I, Q and U images were required in this project because the Stokes V component arising from scattering is negligible. A typical set of images are shown in Figure 5.6 and the corresponding MCFOST parameters are given in Table 5.2.

5.4.2 Taking measurements

Polarised visibility ratios (PVR)

Images of the horizontally and vertically polarised light (H and V) and images of light polarised in the $+45°$ and $-45°$ (A and B) can be obtained from the I, Q and U images as follows:

$$H = I + Q \quad V = I - Q \quad A = I + U \quad B = I - U$$

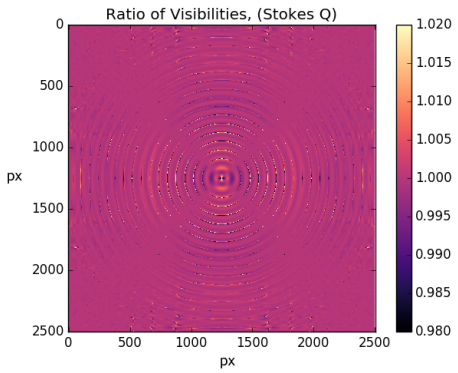


Figure 5.7: PVR_Q corresponding to the images produced in Figure 5.6

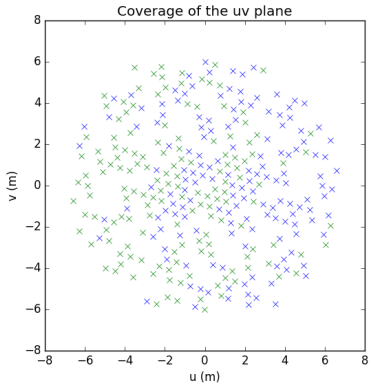


Figure 5.8: Coverage of the uv plane by an 18 hole aperture mask generating 153 baselines. Note that the coverage and PVR is inversion-symmetric as a pair of holes forms two baselines corresponding to the direction the vector of separation points but is counted as one since only one piece of information is recovered. This u,v coverage corresponds to the VAMPIRES 18 hole nudged mask (Figure 3.6a)

For each of these, we obtain the power spectrum (PS) by taking the absolute value of the square of the two dimensional Fourier transform.

$$PS(u,v) = |FFT2(\vec{x})^2|$$

These are then re-centered so that the 0 baseline (brightest point) is at the center and then normalised by dividing through by the maximum value. The two following polarised visibility ratios (PVR) corresponding to Figure 5.6 were produced.

$$PVR_Q = \frac{\text{normalise}(\text{center}(PS_H))}{\text{normalise}(\text{center}(PS_V))} \quad PVR_U = \frac{\text{normalise}(\text{center}(PS_A))}{\text{normalise}(\text{center}(PS_B))}$$

Sampling the PVRs

To produce data from the model that corresponds to that obtained from the telescope, the PVRs need to be sampled at points to achieve the same coverage as the aperture mask in the uv-plane; in this case, the VAMPIRES 18 hole nudge mask (Figure 3.6). Note that the units of these co-ordinates are baseline length divided by observation wavelength (750 nm). In order to correctly sample the PVRs we multiplied the co-ordinates by the wavelength so that co-ordinates are given in meters, then determined the scale of the PVRs in meters.

The original images (Figure 5.6) had a field of view of 150 AU and were 2501 pixels across. Given that the distance to the image was 100 pc, 1 pixel corresponded to ≈ 0.600 mas. By the Nyquist criterion, the highest frequency in the PVRs will be $\theta \approx 1.200\,\text{mas} \approx 5.815 \times 10^{-9}$ rad. It can be derived from the double slit experiment that $\theta = \dfrac{\lambda}{B}$ where λ is the wavelength of observation and B is the baseline length. Hence, the maximum baseline length the PVR can represent is $B = \dfrac{\lambda}{\theta} = \dfrac{750\,\text{nm}}{5.815 \times 10^{-9}\,\text{rad}} \approx 129.0\,\text{m}$. This maximum baseline is the radius of the largest circle that can be drawn inside the PVRs images. The sampling was then done using the `scipy interpolate` module with a 2d cubic interpolation. The sampling of the visibility ratios are shown in Figure 5.9.

5.4.3 Fitness Score

The fitness score used was the reduced χ^2 error, so the problem becomes a minimisation problem. The χ^2 error is defined as:

$$\chi^2_{\text{reduced}} = \frac{1}{N-p}\sum_{i=0}^{N}\frac{(R_i - m_i)^2}{e_i^2} \tag{5.5}$$

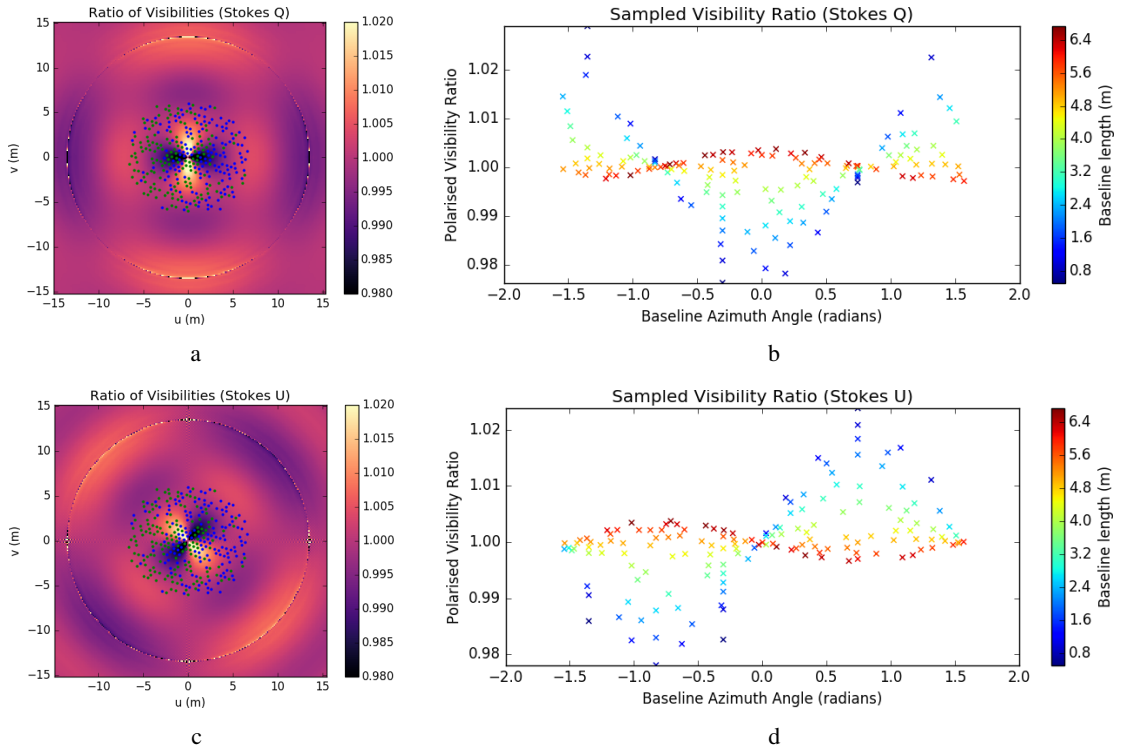


Figure 5.9: Artificial data taken from a model produced by MCFOST (Figure 5.6). Each cross in the sampled visibility ratio corresponds to a point in the ratio of visibilities image.

where N is the number of data points, p is the number of free parameters, R_i is a data point in the R Leo data (Figure 3.8), m_i is a data point from the model (Figure 5.9) and e_i is the error in R_i.

5.4.4 Constraint: inner radius of the dust shell within the stellar radius

There was one constraint that needed to be addressed and that was dealing with models where the inner radius of the dust shell (`rin`) would fall inside the stellar radius, which would be non-physical. This was achieved by expanding the inner radius of the dust shell to lie just outside the stellar radius by an arbitrary small value (0.015 AU). Note that the co-ordinates of the star also needed to be taken into account if the star was not centred within the dust shell but located at the position (x, y, z).

The expanded `rin` was calculated using the following:

$$\text{rin}_{\text{expanded}} = \text{stellar radius} + \sqrt{x^2 + y^2 + z^2} + 0.015 \tag{5.6}$$

Where all units are in AU.