Bio-inspired Computing in R

Deliverable 1: Final Year Dissertation

Ryan Porteous

BSc (Hons) Computer Science

Supervisor: Dr Michael Lones

Second Reader: Dr Katrin Lohan

Declaration of own work

I, Ryan Porteous confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed:

R. Porteous

Ryan Porteous

Date: 26/11/17

Abstract

R has become the first-choice language for data scientists. However, it is typically not the first choice for people developing and implementing bio-inspired algorithms. Consequently, it can be hard for data scientists to make use of bio-inspired methods. This project will look at the current availability of bio-inspired algorithms in R, identify holes in the provision, and develop a package to fill in one of these holes.

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to give an overview of the domain of the project; to identify and describe the motivations and objectives of the project; and to give initial plans of how to overcome foreseeable problems. Supporting technical literature relevant to the project domain is also discussed.

1.2 Aims and Motivations

Bio-inspired computing is a field which takes inspiration for its algorithms from a variety of nature's systems such as evolution, and the way populations of animals interact with each other in an environment. This collection of algorithms can be applied to various areas and can also be used as a searching method to solve NP-hard problems due to the way the search space can be explored. R is a programming language which has become more popular in recent years as can be seen in the TIOBE Index (TIOBE, 2017). It is among the first-choice of languages for statisticians and data miners, with competition being mainly from Python, but is not the first-choice for those who are interested in implementing bio-inspired algorithms. Due to this, it can be hard for R users to apply bio-inspired methods to problem due to their limited availability.

The primary aims of this project can be defined as:

1. **Investigate the availability of bio-inspired algorithms in R**
   I will begin by identifying the main areas of bio-inspired computing and searching for implementations of these algorithms available through the Comprehensive R Archive Network (CRAN). CRAN is a network which provides a central platform for R users to upload the software packages that they have developed and provide access to them for other users.

2. **Identify implementations to be improved or built upon, and areas where no solution exists**

   From the implementations found in the previous step, I will assess how the solution has been implemented, what the solution provides and if it can be improved upon. Where no solution exists, this will be identified as an area that can be developed.

3. **To learn the fundamentals of R**

   As mentioned previously R is among the first-choice of languages for data mining which is a field I am interested in. Thus, I aim to learn the fundamentals of the R language and gain practical experience of using them.

4. **Produce an R Package to improve the availability of bio-inspired tools for R**

   A package will be developed to improve upon an existing solution or to provide a solution where no solution exists. I will aim to follow the best practices when creating this package to maintain a high standard of code quality and maintainability.

5. **Release the package on CRAN**

   I aim to release the package on CRAN so that the package will be freely available for other R users to make use of.

6. **Evaluate the functionality of the package and identify areas for improvement**

   Evaluation is an important stage of software development and I plan to incorporate it into this project. I will assess what the created package provides and what could be added to it. I will also assess the performance of the package and suggest possible improvements.

2 Bio-inspired Computing

2.1 Genetic Algorithms

Genetic Algorithms (GAs) are an evolutionary search heuristic which takes inspiration from the process of natural selection (Darwin & Wallace, 1998). The algorithm uses a population of solutions to the given problem where each solution is given a fitness value which defines how suitable this solution is in this domain. The fitness value which can either be maximised or minimised is given from a fitness function which is defined depending on the scenario. This fitness value is used in the selection process which mimics natural selection (Darwin & Wallace, 1998). Each solution has a probability relative to their fitness value of being chosen as a parent. Parent is a term used to refer to a solution from the current generation which will be used in the crossover process to produce a child. A child is a solution that will carry over to the next generation. Crossover is a process, or operator, where two parents are used to generate a child solution. The goal is to combine both parents while removing the negative characteristics of the parents so that the child will have an improved fitness. Another operator which is used is mutation. This randomly alters the child solution and can help to explore the search space quickly (Sivanandam & Deepa, 2008). One implementation of this according to (Moon et al. 2002) is to choose two random values in the solution and to swap them. This process is repeated until a pre-defined number of generations have completed, or a set number of generations have passed with no improvement.

Genetic Algorithms are used for solving optimisation problems which are problems that involve finding the optimal solution in a search space of all possible solution. It can be difficult to find the globally optimal solution due to the search landscape itself having many local maxima, noise or from other constraints according to (Kramer, 2017). The performance of GA’s are reduced significantly in problems which have very high dimensions and where the evaluation of the fitness function becomes very computationally intensive (Kar, 2016).

2.2 Cellular Automata

Although Cellular Automata (CA) were originally outlined by von Neumann and Stanislaw Ulam with the motivation of modelling biological self-reproduction (Wolfram, 1983) they did not gain widespread interest until John Conway's "Game of Life" was revealed in 1970 (Adamatzky, 2010). CA are mathematical models consisting of simple components with local interactions (Navid & Bagheri, 2013) which are made up a lattice consisting of cells. The lattice can be defined as an n-dimensional list of cells where the cells have two states, black or white. To evolve or update the lattice of cells we use discrete time where time 0 is the initial state of the lattice. In each generation, a set of rules is applied to each cell. In a one-dimensional list, the colour of a given cell at each step is dictated by the rules which consider the colour of the cell and it's left and right neighbouring cells on the previous step (Wolfram, 2002). Thus, a simple rule may be defined as if the given cell and all surrounding cells were black in the previous step, then turn the given cell white. In a one-dimensional lattice, the lines of cells can be layered to provide a visual representation of their behaviour over time which is an important characteristic of CA. Of course, there is no reason why this definition cannot be expanded into using more than 2 states for each cell, or defining a cells neighbourhood as all surrounding cells such as in the Moore neighbourhood which applies to two-dimensional automata and is defined as the 8 cells surround a given cell (Adamatzky, 2010). Another common neighbourhood is the von Neumann neighbourhood (Weisstein, 2003) which uses the cells directly above, below, to the left, and to the right of a given cell. The boundaries of the lattice need conditions to handle the problem where a cell's neighbourhood is out of bounds. A common way of handling this problem is to wrap the lattice at the edges.

Cellular automata can be used for the modelling of different processes. One such process is the spread of forest fires (Ghisu et al. 2015). Another is using them to generate random numbers that can be used in encryption (Sarkar, 2000).

2.3 Artificial Neural Networks

Inspired by biological neural networks, Artificial Neural Networks (ANN) are one of the most widely used bio-inspired techniques. McCulloch and Pitts (1943) are credited with the writing of the article which marked the beginning of Neurocomputing (Yadav et al. 2015). In the article they created a computational model for neural networks and showed that any arithmetic or logical function could be computed by a simple neural network. According to (Yadav et al. 2015) an artificial neural network is an information processing system that has performance characteristics also present in biological neural networks. Russell and Norvig (2009) formally define them as collections of nodes, or neurons, connected by directed links. Each link has a continuous weight value which governs the strength and sign of the link. Each node computes the weighted sum of its inputs and then applies an activation function to produce an output value. The activation function works as a threshold which allows a network to represent nonlinear functions. Russell and Norvig (2009) also explain how this node definition can be connected to form a network. There are two main options which are feed-forward networks and recurrent networks. A feed-forward network's connections form a directed acyclic graph as the nodes can only send information forward. Nodes in a recurrent network receive their output values as inputs which allows them to support short-term memory. Figures 2-1 and 2-2 show examples of a feed-forward and recurrent network respectively.

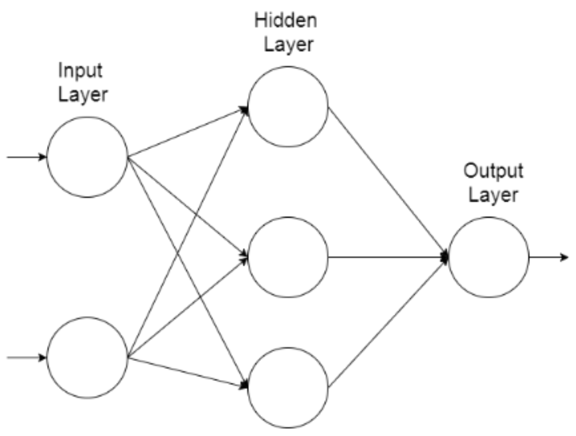


Figure 2-1: An example of a feed-forward neural network topology

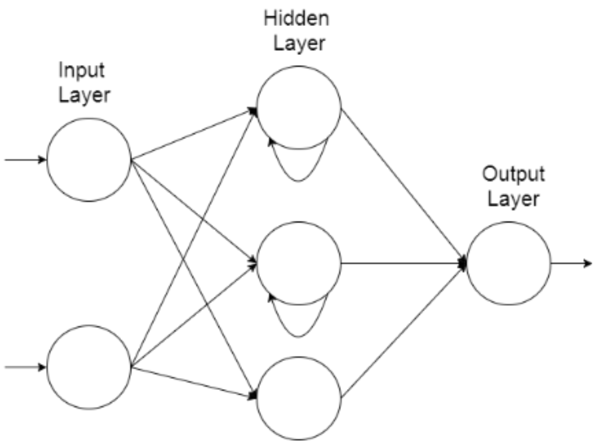


Figure 2-2: An example of a recurrent neural network topology

Due the amount of research applied to ANNs, there are many different topologies or arrangements and can be applied to a variety of problems. They are useful for identifying relationships between variables or recognising patterns within data (Zhang, 2009) and due to this are a common tool used in data mining where they have been applied to both supervised and unsupervised learning problems (Craven & Shavlik, 1997).

2.4 Swarm Intelligence

Swarm Intelligence is an area of algorithms which have gained a lot of popularity due to their versatility and their efficiency in solving nonlinear design problems (Yang & Karamanoglu, 2013). I will cover two of the main swarm intelligence algorithms, namely Ant Colony Optimisation (ACO) and Particle Swarm Optimisation.

2.4.1 Ant Colony Optimisation

This algorithm takes inspiration from real ant colonies. Ants use pheromone to mark paths leading to food to communicate its location indirectly to other ants (Khushaba et al. 2008). Khushaba et al. (2008) continue to explain the behaviour of ants foraging for food. The amount of pheromone deposited depends on the distance to the food source, and the quality and quantity of the food source. The paths that are shorter are visited more on average due to more pheromone existing on the path. After a period, the difference in the amount of pheromone between the path options is large enough so that future ants to come across the paths are likely to follow the path previously marked and reinforce the option with their own pheromone (Dorigo & Gambardella, 1997). The quality of a solution to a problem can be modelled as the concentration of pheromone on a path according to (Yang & Karamanoglu, 2013). Due to the solution being modelled this way, the algorithm generally produces routes and paths evident from their higher concentrations, thus ant algorithms are well suited to discrete optimisation problems.

The ACO was originally used to solve the Travelling Salesman Problem and was effective at finding good solutions (Khushaba et al. 2008). ACO has been applied as a searching method within feature selection problems namely in areas of face and speech recognition problems (Khushaba et al. 2008). Feature selection is the process of reducing data with high dimensions into a lower dimension while keeping as close to the same amount of information as possible (Khan & Baig, 2015).

2.4.2 Particle Swarm Optimisation

The collective behaviour of different animal species is the basis for Particle Swarm Optimisation (PSO). Behaviours such as fish schooling, insect swarming, and birds flocking are examples of such behaviour (Saka et al. 2013). Many newer algorithms that are based on swarm intelligence have taken inspiration from different areas, but still share connections to components used within PSO, thus it can be said that PSO established the foundational ideas of swarm intelligence based computation as Yang (2014) describes.

Particle Swarm Optimisation was developed by Eberhart and Kennedy (1995) and they state in this article that it can be used to solve many of the same problems as the previously described area, Genetic Algorithms, but does not suffer from some of the same difficulties. Since it can be used to solve the same types of problems, it is useful to compare the algorithms stating the differences. As mentioned previously, GA use operators known as mutation and crossover, but this is not present in PSO. Instead it uses random real-numbers and allows the particles to communicate with each other (Yang, 2014). Yang (2014) also continues to explain that PSO is easier to implement due to no encoding or decoding of the solution being used. Eberhart and Kennedy (1995) describe the PSO concept as being like a GA in that a starting population of random candidate solutions is used but differ in that each solution is given a velocity value and is then “flown” through hyperspace. Solutions in PSO are referred to as particles. Each of these particles has memory, which is not a feature in a GA. This stores a value called the pbest which is the coordinates of the best solution found so far in the search space. The gbest is also stored by the particle swarm optimiser which is the best solution found by any of the population of particles. The search space is explored by the particles moving through the space, the moves are decided by referring to the particles own performance so far and the collective performance of the entire swarm (Saka et al. 2013).

Cho et al. (2011) discuss some of the challenges faced by the PSO topology defined by Eberhart and Kennedy (1995). Using gbest helps particles to converge to a solution quickly as they are attracted to move towards the global best solution found by the swarm. This is a problem as often the particles are trapped in a local maximum because not enough of the search space was explored before converging. Another topology which Cho et al. (2011) describe is lbest. In this, particles can only communicate with a select number of other particles allowing for a more thorough exploration to take place, but convergence occurs slower than gbest.

2.5 Genetic Programming

Genetic Programming (GP) is the last area of bio-inspired computing that I will cover and is an area interested in using natural selection to automatically evolve computer programs (Miller, 2011). Koza (1992) describes the structure of a GP algorithm by stating that it starts with an initial population consisting of randomly generated computer programs. These programs consist of functions and terminals defined according to the domain of the problem. Functions can be anything from arithmetic or programming operations to mathematical or programming functions. The collection of allowed functions is called the function set. These functions can branch into other functions or terminals. Terminals are the variables and constants allowed in the program. The collection of terminals is called the terminal set. Koza (1992) continues by stating that each of these programs are measured according to their fitness value, that is, how well it performs in the given problem. The algorithm performs in generation just like a GA and with each generation with the goal of improving the fitness values of the population each time.

2.5.1 Tree-based Genetic Programming

In tree-based GP, programs are expressed in the form of parse trees or abstract syntax trees. The internal nodes of the tree are elements taken from the defined function set and the leaf nodes are elements taken from the terminal set. For example, a tree built using a function set defined as $\{+,\ *,\ -,\ /\}$ and a terminal set consisting of $\{x, y, 2, 5, 1\}$ may look like the example shown in Figure 2-3 or Figure 2-4.

2.5.2 Initialisation of the Population

There are different styles of initialising the population of random programs and Poli et al. (2008) outline the full and grow methods, as well as a combination of both known as ramped half-and-half.

In the full and grow methods, a user set maximum depth parameter is chosen and the random individuals of the population are generated so that they do not exceed this depth. Poli et al. (2008) define this depth as the number of edges that need to be traversed to reach a specific node from the tree's root node. The full method is appropriately named as it generates full trees, meaning nodes are generated from the function set until the maximum depth is reached, and all the leaves are at the same depth. Each of these leaves may only be a terminal as choosing a function would cause the tree to exceed the maximum depth. The grow method allows for trees with more variation in the shapes and sizes then the full method. It differs by allowing any function or terminal to be selected until the user defined depth is reached.

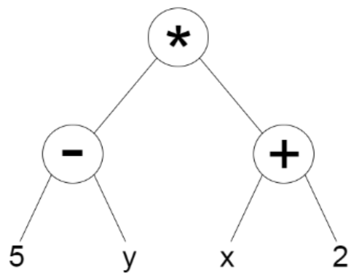


Figure 2-3: A tree built using the full method

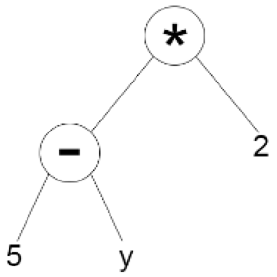


Figure 2-4: A tree built using the grow method

As Poli et al. (2008) state, the full and grow methods do not provide a wide array of tree shapes and sizes. Koza (1992) defines a method to combat this problem called ramped half-and-half. The method incorporates both the full and grow methods and is useful because often in GP the size or shape of the ideal solution is not known in advance. A maximum depth is still used but this time a range of depths from two to the maximum is used so that an equal number of trees is produced for each depth. For each value of depth, half of the trees are created using the full method and the other half are created using the grow method. Due to all full trees for a given depth having the same shape and grow trees shapes varying widely from each other, this allows the ramped half-and-half method to create a variety of sizes and shapes.

2.5.3 Operators in GP

I mentioned previously in Section 2.1 that genetic algorithms are based on natural selection and since this is also true for GP, it uses the same steps and operators although they are defined very differently in practice. Namely these operators are selection, crossover and mutation. Selection is defined the same way for GP and uses the same selection method. Poli et al. (2008) describe the other two operators at a high level as follows. In crossover for GP, a child program is created by combining parts of two selected parent programs. Mutation in GP is defined as the creation of a new child program by altering a randomly selected part of a selected parent program. These operators are used to progressively help to improve the fitness of the programs while still allowing the search space to be explored by not applying too much pressure.

2.5.4 Problems with Tree-Based Representation

Tree-based GP is one of the older methods of GP and as such has various problems associated with it. As Poli et al. (2008) state that in a high-performance environment, a tree-based representation can be too inefficient as it requires the storage and handling of many pointers. Another issue with a tree-based representation is that expressions in separate subtrees need to be re-evaluated multiple times wasting time and memory as well as adding complexity to the tree. An example of this is shown in Figure 2-5. Both the left and right subtrees have the expression $5 - y$ so it must be evaluated twice.

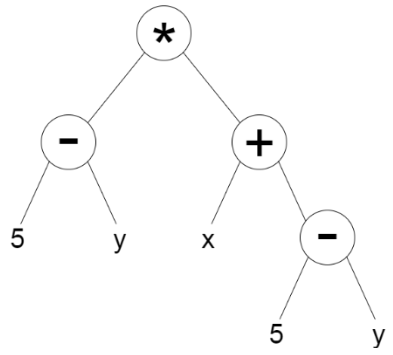


Figure 2-5: A tree with two subtrees containing $5 - y$

2.5.5 Other Types of Genetic Programming

There are other types of GP which aim to improve upon some of the problems faced in a tree-based representation. Some examples of these are Cartesian GP (Miller, 1999), Linear GP and Probabilistic GP along with a variety of others.

3 Availability of Bio-inspired Algorithms in R

3.1 Genetic Algorithms

Access to the algorithms in this area are covered through the two packages mentioned in Table 3-1.

Package Name	Implemented In	Provides	Source
GA	Entirely in R	Set of tools for implementing GA, Can define own operators Can define own fitness function	(Scrucca, 2013)
genalg	Entirely in R	Implementation of GA for multi-dimensional function optimisation	(Willighagen & Ballings, 2015)

Table 3-1: Packages providing access to genetic algorithms

3.2 Cellular Automata

The package identified in Table 3-2 provides an implementation of one-dimensional cellular automata but as mentioned in Section 2.2, multi-dimensional CA exist and there is no access to these currently within R. This is an area which can be developed.

Package Name	Implemented In	Provides	Source
CellularAutomaton	Entirely in R	one-dimensional cellular automata	(Hughes, 2013)

Table 3-2: Packages providing access to cellular automata

3.3 Artificial Neural Networks

Due to Artificial Neural Networks having a wide range of uses, there are many implementations to cover the different topologies of ANNs. Although there are many packages identified, most of them are built purely in R. These could be improved slightly by implementing the computationally heavy parts in a more efficient language such as C or Java and interfacing to them from R. This is an area which could be improved.

Package Name	Implemented In	Provides	Source
rnn	Entirely in R	Multi-Layered RNN, Gated Recurrent Unit, LSTM NN	(Quast, 2016)
rsnns	Entirely in R	An R interface to the Stuttgart Neural Network Simulator	(Mergmeir & Benitez, 2012)
neural	Entirely in R	Radial Basis Function and Multi-layer Perceptron with an attached graphical interface	(Fritsch et al., 2016)
nnet	The neural network is implemented in C and an interface to it is provided in R	Feed-forward Neural Networks with a single hidden layer	(Venables & Ripley, 2002)

Table 3-3: Packages providing access to artificial neural networks

3.4 Swarm Intelligence

Table 3-4 shows three packages which implement Particle Swarm Optimisation with "ppso" providing an optionally parallel solution. All three packages are built entirely in R meaning these could be improved slightly by writing code to perform the demanding tasks. This is an area which could be improved.

Package Name	Implemented In	Provides	Source
pso	Entirely in R	Implementation of Particle Swarm Optimisation	(Bendtsen, 2012)
psoptim	Entirely in R	Implementation of Particle Swarm Optimisation	(Ciupke, 2016)
ppso	Entirely in R	Optionally parallelised Particle Swarm Optimisation	(Francke, n.d)

Table 3-4: Packages providing access to areas of swarm intelligence

3.5 Genetic Programming

The package "rgp" provides an implementation of tree-based GP but as previously mentioned in Section 2.5.5, other types of GP exist. This is an area which can be developed.

Package Name	Implemented In	Provides	Source
rgp	R with computationally heavy parts written in C	Implementation of tree-based GP	(Flasch et al., 2014)

Table 3-5: Packages providing access to genetic programming

4 Genetic Programming

While I have identified Cellular Automata as an area that could be developed, ultimately, I have chosen to build upon Genetic Programming. The reasoning behind this choice is that access to the area is limited and I also find this area more interesting that CA. Also, GP has more practical uses than CA has within the field of data science. As mentioned in Section 2.5.5, other areas of GP exist and one of these is Cartesian Genetic Programming (CGP). This is the area I would like to develop a package to provide access to.

4.1 Cartesian Genetic Programming

Cartesian Genetic Programming is a form of graph-based GP. Graph-based GP shares some similarities with tree-based GP in the way that programs are represented but also has some important distinctions. Graphs are like trees but represent the links between nodes using arrows showing directions and allow for cycles which allows for the reuse of previously calculated subgraphs. This solves one of the issues involved with tree-based GP (Miller, 2011). Figure 4-1 and Figure 4-2 show how a graph representation can reuse previously evaluated expressions.

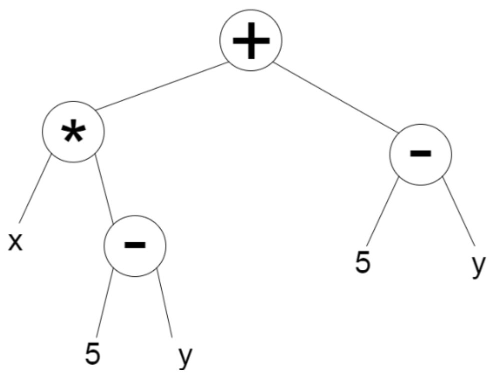


Figure 4-2: A tree with repeated subtrees of 5-y

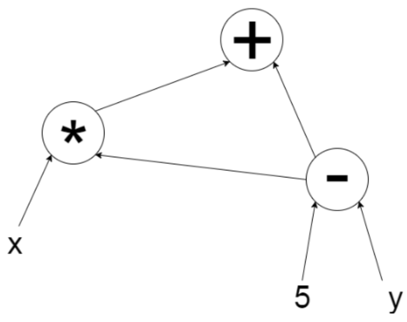


Figure 4-1: A graph reusing the 5-y subgraph

One way to represent a program in CGP is to use a list of integers referred to as a genome. A genome in CGP has a set size which stops this method of GP suffering from bloat which is a common problem in other methods (Turner & Miller, 2017). This set size is a maximum length of the genome and phenotypes of any size less than this can be produced allowing varied sizes of phenotypes. As Turner and Miller (2017) describe, a genome is composed of function genes, connection genes and output genes. Function genes contain integers which are corresponding entries in a function look-up-table. This look-up-table is similar to the function set described previously except that each function is mapped to an integer. Connection genes specify where the inputs for the function come from, and output genes specify where the output of the genome comes from. The structure of a genome for the graph in Figure 4-2 can be seen in Figure 4-3. The first value in each group is the function gene and the others are connection genes. The connection genes can either specify the inputs to the program or other groups. This can be seen in group 1 where “x” is an input to the program and “0” is group 0. The last group containing a single value is an output gene which specifies that the output of the genome comes from group 2. The function table for this genome can be specified as subtract (0), multiply (1), add (2).

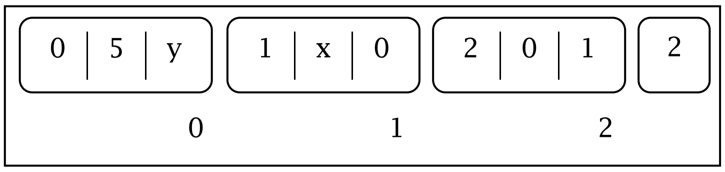


Figure 4-3: A genome representing the graph from Figure 4-2.

Another way of representing programs in CGP which is more visual is shown in Figure 4-4. In this representation the grid imposes a restriction on the maximum size of the program, but this grid still needs to be large enough to still allow for expressiveness and evolvability. The squares in each column are not allowed to be connected to other nodes in the same column as Miller and Smith (2006) describe. Miller and Smith (2006) also describe an additional parameter on the representation that defines how many columns away a column can connect to, this is known as level-back. The squares on the right show the input values to the program. The squares in the two-dimensional grid show a random selection of functions from a function set. The square on the right marked as outputs has the same meaning as it does in the representation show in Figure 4-3, it marks which parts of the program the output result can be taken from. The arrows mark the direction and destination of the values being passed. The grid contains many functions which were unused in this execution which are represented by the greyed-out squares. During evolution subgraphs can also become greyed-out through mutation and become active at any point. Overestimating the size of the grid allows for more subgraphs to fade in and out of being used. This has been shown to be very beneficial in the evolutionary search involved in CGP (Miller & Smith, 2006).

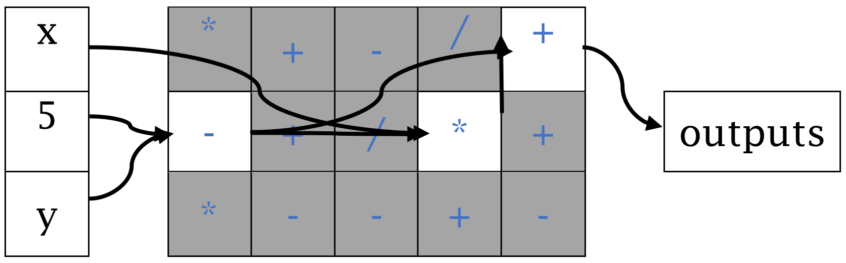


Figure 4-4: A two-dimensional grid representing the graph from Figure 4-2.

5 Implementations of CGP in Other Languages

5.1 JCGP

JCGP is an implementation of CGP in Java created by Eduardo Pedroni (2013). It implements the most commonly used evolutionary strategies as well as mutation methods while also allowing the user to define their own. It also allows the user to select whether the function nodes contain mathematical operators used for Symbolic Regression or whether they contain logic gates for a digital circuit representation. Again, the user can define their own set of function nodes by either selecting a subset of the functions provided by JCGP or to provide a completely different set. The key feature of JCGP is that it has an attached Graphical User Interface (GUI) to allow the user to view the chromosomes as they evolve. The ability to manually change the connections or function of each node in the chromosome is also supported. An interesting feature of JCGP is that it supports the ability to step through the generations of the evolutionary process which makes it clear to see how the chromosome changes over time. Another feature is that it allows the user to save the parameters they have set to a file which can then be loaded and used again later. Files containing the structure of the chromosomes can also be loaded but they architecture of the chromosome must be the same as the parameters in the program.

5.2 CGP-Library

CGP-Library is a cross-platform implementation of CGP written in C (Turner, 2014). It implements classic CGP as well as Recurrent CGP and the ability to use them with Artificial Neural Networks. It provides a C source and header file to be compiled along with the user's own source code allowing the user to use the library. This library also features the ability to view the structure of a chromosome by using the open source Graphvis (Graphviz.org, n.d) software which displays the structure of the chromosome in a graph format. User defined functions, selection operators and reproduction operators can also be added easily.

5.3 CGP for ECJ

ECJ is an evolutionary computation system written in Java which supports a variety of evolutionary computation techniques. Although its origins are in Genetic Programming, it only provides Tree-Based GP. ECJ also supports plugins and there is a plugin developed to provide Cartesian Genetic Programming called CGP for ECJ (Oranchak, 2009). It supports real-valued and integer representations of the genomes. The plugin also includes a collection of sample problems and associated scripts and parameters files to run them. The plugin also includes three symbolic regression problems which can be run using either the integer representation or the real-valued representation. Classification problems are also included where each problem uses the real-valued representation. The classification problems are configured by default to split the dataset into a test set of 30% of instances and a training set of 70% of instance but this parameter can be altered.

6 Requirements Analysis

As the project is a personal project, the primary user will be myself although I do plan to release the project to the public if I believe it has been completed to a high standard.

Even though the project is aimed towards users from a technical background, it is essential that the project is easy to use as they may not have knowledge in the field of Genetic Programming but still wish to use the software. The software will be provided with documentation describing the purpose and how to use each component as well as a guide for getting started with started using the software.

The requirements of the package can be split into three main categories:

1. Allow users to perform Cartesian Genetic Programming on a set of data containing the desired results.
2. Allow users to change the parameters of the program and define their own where necessary.
3. Provide results from the program through textual outputs and graphs.

The following requirements will be assessed using the MoSCoW method which is used to assign priorities to requirements. The four categories are Must have, Should have, Could have and Won't have. Each requirement will be given a Requirement ID to allow them to be easily referred to during evaluation. The first number refers to the main category identified above, and the second number is the unique number of the requirement within the category.

6.1 Functional Requirements

Requirement ID	Description	Priority
FR1-1	The package shall implement Cartesian Genetic Programming	Must
FR1-2	The package shall allow users to run the evolutionary process on a population of chromosomes	Must
FR1-3	The package shall allow users to step through each generation in the evolutionary process	Could
FR1-4	The package shall allow users to pause and resume the evolutionary process	Could
FR1-5	The package shall allow users to load a file containing the desired results	Must
FR1-6	The package shall measure the fitness of a solution compared to the desired result	Must
FR1-7	The package shall be able to construct classifier models for a given dataset	Could
FR1-8	The package shall be able to build regression models through evolution.	Must
FR1-9	The package shall allow existing R data structures to be passed into the program	Should
FR2-1	The package shall provide a symbolic regression function set consisting of mathematical operations	Must
FR2-2	The package shall provide a logical function set consisting of logic gates	Should
FR2-3	The package shall provide the $(\mu + \lambda)$ and Tournament Selection operators to be used	Must
FR2-4	The package shall provide mutation methods to be used	Must
FR2-5	The package shall allow users to choose a function set from the included choices	Should
FR2-6	The package shall allow users to choose a selection operator from the included choices	Should
FR2-7	The package shall allow users to choose a mutation method from the included choices	Should
FR2-8	The package shall allow users to define their own function set	Should
FR2-9	The package shall allow users to select a subset of the functions from a function set	Could
FR2-10	The package shall allow users to define their own selection operators	Could
FR2-11	The package shall allow users to define their own mutation methods	Could
FR2-12	The package shall allow users to change the parameters of the program such as: • Number of columns in chromosome • Number of rows in chromosome • Number of generations to run • The levels-back parameter • The population size • Random number seed	Must
FR3-1	The package shall display the results in a textual format in the R console	Must
FR3-2	The package shall create an output file containing the results	Should
FR3-3	The package shall display the results in a plotted graph	Could
FR3-4	The package shall output an R data structure containing the results through its return value to be used elsewhere	Could
FR3-5	The package shall allow users to load previous experiments to	Could

	view the results as a graph	

Table 6-1: Functional requirements of the package

6.2 Non-functional Requirements

The following non-functional requirements are arranged into categories of:

1. Performance
2. Visualisation
3. Usability
4. Robustness

The requirements are given a Requirement ID which is used to categorise them. The first number refers to the category of non-functional requirement identified above, and the second number is the unique number of the requirement within the category.

Requirement ID	**Description**	**Priority**
NFR1-1	The packages computationally intensive parts shall will be written in C, C++ or Java to improve performance	Should
NFR1-2	Only active nodes shall be processed when decoding the chromosome to improve performance	Should
NFR1-3	The package shall stop execution when a solution has been found	Must
NFR2-1	The created graphs shall be visually clear and understandable to the user	Must
NFR2-2	The textual format shall be understandable and only contain necessary information	Must
NFR3-1	The package shall provide convenient methods of abstracting away from the lower level functionality	Should
NFR4-1	The package shall minimise errors that cause it to fail	Must

Table 6-2: Non-functional requirements of the package

7 R Packages and R Development

The availability and broad range of packages in R has been one of the main reasons for the success of R as a language. R is distributed with many standard or base packages which make up the R source code. There are more benefits to an R package than being an easy way to structure and share functions and datasets, packages can also be dynamically loaded and unloaded at runtime meaning they only occupy memory while being used.

There are different versions of packages. The diagram shown in Figure 7-1 shows the differences between the types of packages.

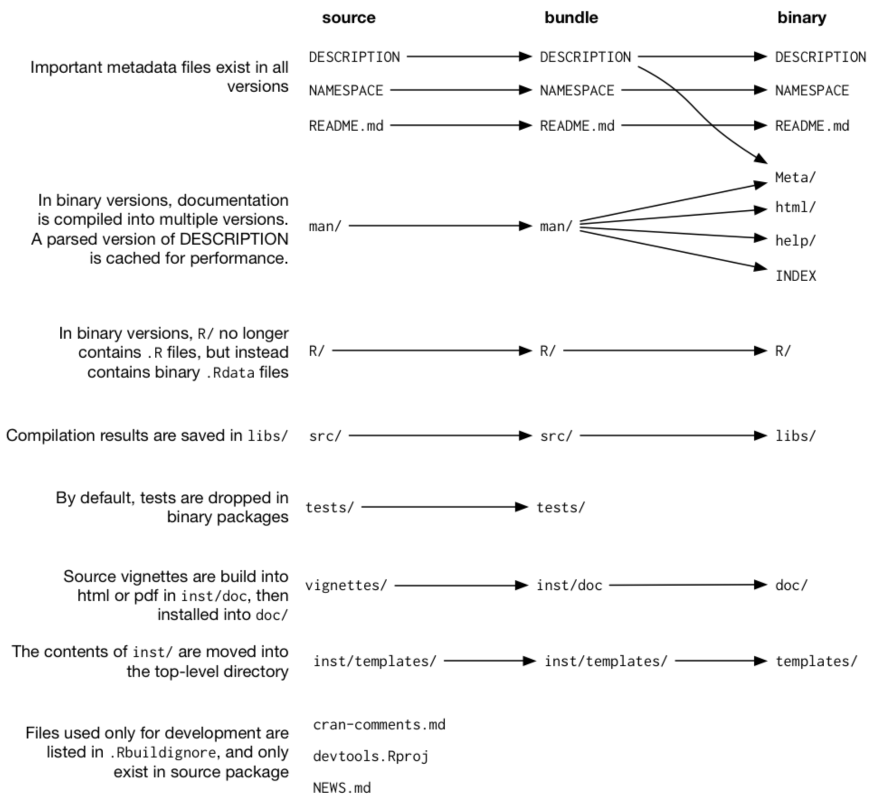

source
bundle
binary
Important metadata files exist in all versions
DESCRIPTION
NAMESPACE
README.md
Meta/
In binary versions, documentation is compiled into multiple versions. A parsed version of DESCRIPTION is cached for performance.
man/
html/
help/
INDEX
In binary versions, R/ no longer contains .R files, but instead contains binary .Rdata files
R/
Compilation results are saved in libs/
src/
libs/
By default, tests are dropped in binary packages
tests/
Source vignettes are build into html or pdf in inst/doc, then installed into doc/
vignettes/
inst/doc
doc/
The contents of inst/ are moved into the top-level directory
inst/templates/
templates/
Files used only for development are listed in .Rbuildignore, and only exist in source package
cran-comments.md
devtools.Rproj
NEWS.md

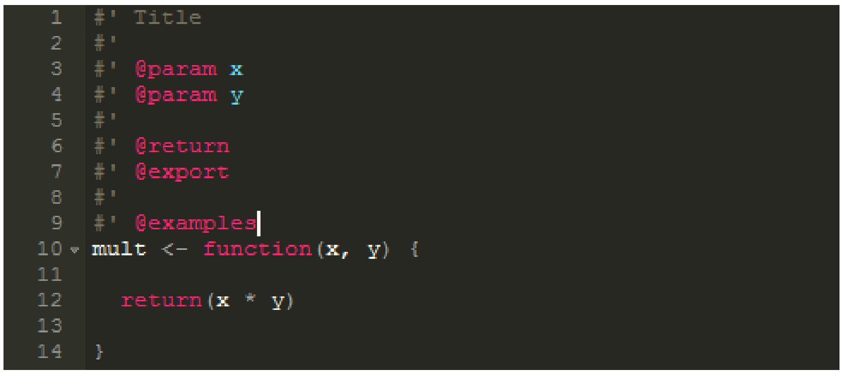
Figure 7-1: The differences between versions of R packages. Taken from R Packages (Wickham, 2015)

To create a package in RStudio there is a convenient menu to allow you to do so. After doing this the created package will contain the three minimal components. The first is a R/ directory to contain code written in R. The next is a DESCRIPTION file which describes the function of the package and contains the details of who created the package and what the dependencies are. This is the information shown on the CRAN page for a certain package. The last is the NAMESPACE file which is vital for releasing the package on CRAN as it helps to encapsulate the package and reduce the conflicts it will have with the naming conventions of other packages. It does this by specifying which functions are available from outside of the package with the export tag.

7.1 Packages and Software to aid Development

7.1.1 roxygen2

The package “roxygen2” (Wickham et al., 2017) provides a way of automatically generating documentation for a package. When used it adds a basic skeleton of keywords to the comment section above a function. Figure 7-2 shows a simple example of what happens when pressing the command “crtl+shift+alt+d” within R studio. After filling in the skeleton, entering the command “roxygen2::roxygenise()” in the console will create documentation files for each function. Updating the documentation is as easy as running the command again as it recognises which parts have changed. This also updates the NAMESPACE file saving time and reducing the chance of errors.



```r
#' Title
#'
#' @param x
#' @param y
#'
#' @return
#' @export
#'
#' @examples
mult <- function(x, y) {

  return(x * y)

}
```

Table 7-2: Example of code with a roxygen2 skeleton

7.1.2 testthat

The R package “testthat” (Wickham, 2016) aims to make testing of software as painless as possible. It provides functions that make it easy to describe what a function is expected to do and can run tests automatically as code or tests are changed. It also displays test progress visually and shows which expectations passed or failed.

7.1.3 devtools

The aim of “devtools” (Wickham & Chang, 2017) is to make package development easier by providing functions to simplify many tasks. This package also requires that the two packages I have already mentioned are used. It provides convenient functions for installing and building packages, as well as incorporating documentation and testing into it. Using this package, it is easy to create a simple workflow for when changes are made to code so that documentation is always updated, and unit tests are performed.

7.1.4 lintr

I will be using the “lintr” (Hester, 2017) package which checks that code written is compliant with Google’s R style guide and highlights wherever something has been missed. This will help to increase the quality and readability of the code I write.

7.1.5 RStudio

RStudio (RStudio, 2016) is a free open-source integrated development environment (IDE) for the R programming language. It includes an R console, a code editor, tools for debugging, and provides an interface displaying which functions and variables are stored within the current environment. It also provides the features that can be expected from an IDE such as code prediction and direct support to version control facilities.

8 Evaluation Strategy

To evaluate this project, I will compare what I have achieved to the initial aims that I set out in Section 1.2. I will also compare the functionality of my implementation to the requirements specified in Section 6, identifying any requirements that were not met and explaining the reasoning behind this. Areas that could be improved or added in future iterations of the project will also be identified at this stage.

I currently plan to do a small user study once the main features have been implemented where users will work through a short guide instructing them how to use the basic functionality of the software. Through doing this I will be able to refine the requirements allowing me to focus on developing the features that are important to the package and get an idea of what information to include in the guide distributed with the software. After the second implementation phase, I will carry out another small user study with the focus on gathering information on the quality of the user guide. The user guide is important as this is what introduce users to using the basics of the package.

I have also allocated time within my Project Plan to develop unit tests for the software as this will ensure that my program works the way I expect and helps to prevent from changes made later, affecting earlier developed sections of the software. As I covered in Section 7.1, I will be using the "testthat" R package to make this as straightforward as possible.

Measuring the performance of my implementation is also an important part of evaluation and I will assess the time taken, or generations taken, for my implementation to find a solution to a given problem compared to other implementations.

9 Project Management

9.1 Project Schedule

9.1.1 Work Breakdown Structure

Key	
Colour	**Meaning**
🟧	Large Tasks
🟦	Sub Tasks

Task ID	Task	Start Date	Deadline
1	First Deliverable	25-Sep-17	27-Nov-17
2	Draft Deliverable to Supervisor	19-Nov-17	20-Nov-17
3	Ethics Form submitted	22-Nov-17	23-Nov-17
4	First Deliverable submitted	26-Nov-17	27-Nov-17
5	Interview with Supervisor and Second Reader	14-Dec-17	15-Dec-17
6	Development of Package	27-Nov-17	19-Feb-18
7	Design Program Structure and Features	27-Nov-17	06-Dec-17
8	Implementation of Main Features	06-Dec-17	05-Jan-18
9	Develop Unit Tests for Main Features	05-Jan-18	09-Jan-18
10	Documentation of Main Features	09-Jan-18	14-Jan-18
11	Carry out first User Study	14-Jan-18	17-Jan-18
12	Refine Requirements	17-Jan-18	19-Jan-18
13	Implementation of New Main Features	19-Jan-18	30-Jan-18
14	Implementation of Less Important Features	30-Jan-18	06-Feb-18
15	Develop Unit Tests for Final Version	06-Feb-18	10-Feb-18
16	Carry out final User Study	10-Feb-18	13-Feb-18
17	Documentation of Final Version	13-Feb-18	19-Feb-18
18	Write Final Dissertation	19-Feb-18	23-Apr-18
19	Write Draft of Dissertation	19-Feb-18	11-Mar-18
20	Draft Dissertation to Supervisor	11-Mar-18	12-Mar-18
21	Improve from feedback given	12-Mar-18	22-Apr-18
22	Dissertation submitted	22-Apr-18	23-Apr-18
23	Poster Deliverable	20-Apr-18	07-May-18
24	Poster Design	20-Apr-18	22-Apr-18
25	Create Poster	22-Apr-18	25-Apr-18
26	Send Poster for Printing	25-Apr-18	26-Apr-18
27	Prepare for Session	26-Apr-18	03-May-18
28	Poster Session	03-May-18	04-May-18
29	Poster Submitted	04-May-18	07-May-18

Table 9-1: Work Breakdown Structure of the project

9.1.2 Project Timetable

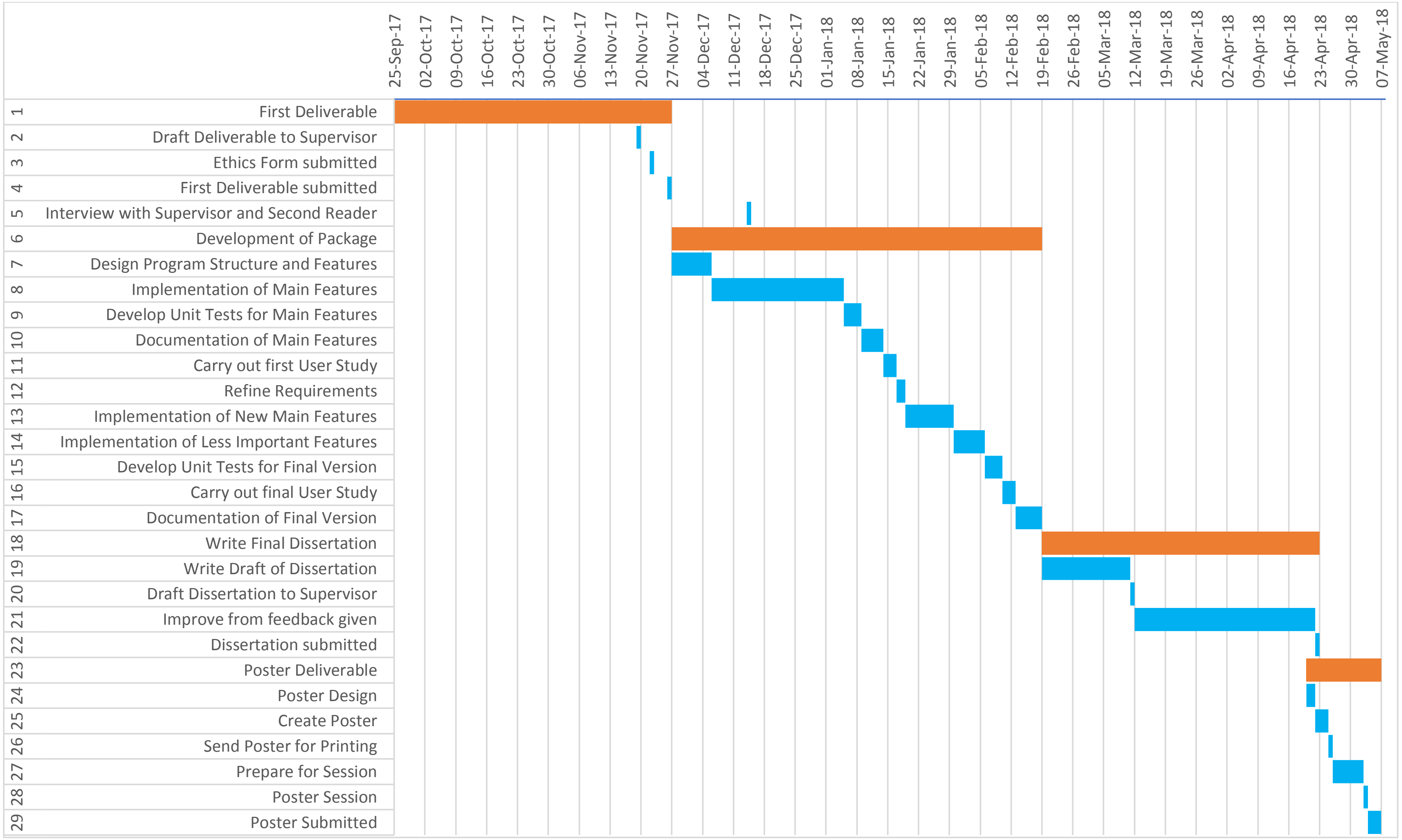


Figure 9-1: Project Timetable in the form of a Gantt Chart

9.2 Risk Analysis

9.2.1 Risk Identification

The risks associated with this project are assigned a probability of how likely they are to occur. They are also assigned an impact measured on a scale of tolerable, serious and catastrophic. Finally, they are given a colour based on the how quickly the risk should be handled depending on the impact and probability of the risk.

Key	
Colour	**Risk Priority**
🟩	Low
🟨	Medium
🟥	High

Risk ID	Risk	Probability	Impact
🟩 R1	I become ill (common flu)	Medium	Tolerable
🟨 R2	I become seriously ill	Low	Serious
🟩 R3	A feature being more difficult than planned	Medium	Tolerable
🟨 R4	Project Plan being too optimistic	Medium	Serious
🟨 R5	Work is lost due to backup failure or file corruption	Low	Serious
🟥 R6	Requirements of the package missing and must be added during development	Medium	Serious
🟨 R7	Supervisor leaves the University or is unavailable for a long period	Low	Serious
🟨 R8	I am impacted by personal issues and must take time away from the project	Low	Serious
🟥 R9	Major problems in the software produced are discovered late in development	Low	Catastrophic
🟨 R10	Not enough participants in user study	Low	Serious
🟩 R11	The project requires a higher level of technical expertise	Medium	Tolerable
🟩 R12	The CRAN website is unavailable when I try to submit my package	Low	Serious

Table 9-2: Risks associated with the project

9.2.2 Risk Management

Plans will be made to lower the impact each medium or high priority risk will have on the project.

R2 I become seriously ill

- If this risk occurs I will contact my supervisor to inform them that I am ill and ask them for my best plan on action for when I recover. If I end up majorly behind my schedule, then I will reassess which functionality I aim to implement.

R4 Project Plan being too optimistic

- If I am falling behind on my schedule I will assess what I have completed so far with what should have been done by this stage and contact my supervisor with a revised plan.

R5 Work is lost due to backup failure or file corruption

- I make regular backups of my work to GitHub to reduce the impact that this risk will have on the project, but it is still possible to lose work in-between backups. To combat this, I will make sure to back everything up whenever a substantial amount of work has been carried out.

R6 Requirements of the package missing and must be added during development

- Depending on the importance of the missing requirement and when it was identified this can have a varied impact on the project. A large important feature identified late in development will have to be supported in a revised or future version of the software.

R7 Supervisor leaves the University or is unavailable for a long period

- Should this risk arise, I will contact the director of the year who will give me the best plan of action to combat this.

R8 I am impacted by personal issues and must make time away from the project

- If this risk arises I will contact my supervisor to notify them of the issues and if I end up majorly behind my schedule, then I will reassess which functionality I aim to implement.

R9 Major problems in the software produced are discovered late in development

- The probability of this risk is low as I am planning to develop unit tests to ensure sections of my software perform the same way as they did when I developed them. If an issue truly is catastrophic to the software, then I will extend the amount of development time and reduce the functionality I expect to implement to allow for time to remedy the issue.

R10 Not enough participants in user study

- To combat this risk, I will extend the time for gathering data from surveys and update my project plan. I will also highlight the short-comings in my analysis.

9.3 Professional, Legal, Ethical and Social Issues

9.3.1 Professional Issues

This project is a personal project meaning I am working alone. Thus, there will be no conflicts of interest between me and another party.

I will also conduct my user study at a professional level ensuring that test subjects know they are free to quit the study at any time and their data will be anonymised. The code that I write will also follow a style guide to ensure a high standard of code readability and professionalism.

9.3.2 Legal Issues

There should be no legal issues with my project. I plan to release my software under the MIT License so that my code can be reused in other people’s future projects as long as they provide attribution to me and do not hold me liable for any problems with the software. I will also be using only free software or software that I have a licence to use.

Any participants in the small user study I plan to perform will have their data anonymized.

9.3.3 Ethical Issues

My project will have no ethical issues as I will ensure I am avoiding any issues relating to the Data Protection Act by anonymising any user data if I perform a user study. If a user study is performed, test subjects will be explained the protocol of the experiment and asked to sign a consent form.

9.3.4 Social Issues

This project does not have social issues as it is not developing a tool or product that can potentially replace a human within a job making the job position obsolete. It is developing a tool that can be used to benefit other software developers and data miners alike.

10 References

Adamatzky, A. (2010). Game of life Cellular Automata. London: Springer, pp.19-25.

Bendtsen, C. (2012). pso: Particle Swarm Optimization.

Bergmeir, C. and Benitez, J. (2012). Neural Networks in R Using the Stuttgart Neural Network Simulator: RSNNS. Journal of Statistical Software, [online] 46(7), pp.1-26. Available at: http://www.jstatsoft.org/v46/i07/ [Accessed 12 Nov. 2017].

Ciupke, K. (2016). psoptim: Particle Swarm Optimization.

Cho, H., Kim, D., Olivera, F. and Guikema, S. (2011). Enhanced speciation in particle swarm optimization for multi-modal problems. European Journal of Operational Research, 213(1), pp.15-23.

Craven, M. and Shavlik, J. (1997). Using neural networks for data mining. Future Generation Computer Systems, [online] 13(2-3), pp.211-229. Available at: http://www.sciencedirect.com.ezproxy1.hw.ac.uk/science/article/pii/S0167739X970002 28 [Accessed 3 Nov. 2017].

Darwin, C. & Wallace, J., 1998. The origin of species / Charles Darwin., Ware: Wordsworth Editions.

Dorigo, M. and Gambardella, L. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation, [online] 1(1), pp.53- 66. Available at: http://ieeexplore.ieee.org.ezproxy1.hw.ac.uk/document/585892/ [Accessed 21 Oct. 2017].

Eberhart, R. and Kennedy, J. (1995). A new optimizer using particle swarm theory. MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science. [online] Available at: http://ieeexplore.ieee.org.ezproxy1.hw.ac.uk/document/494215/ [Accessed 5 Nov. 2017].

Flasch, O., Mersmann, O., Bartz-Beielstein, T., Stork, J. and Zaefferer, M. (2014). rgp: R genetic programming framework.

Francke, T. (n.d.). ppso - Particle Swarm Optimization and Dynamically Dimensioned Search, with parallel option.

Fritsch, S., Guenther, F., Suling, M. and Mueller, S. (2016). neuralnet: Training of Neural Networks.

Ghisu, T., Arca, B., Pellizzaro, G. and Duce, P. (2015). An Improved Cellular Automata for Wildfire Spread. Procedia Computer Science, [online] 51, pp.2287-2296. Available at: http://www.sciencedirect.com/science/article/pii/S1877050915011965 [Accessed 10 Nov. 2017].

Hester, J. (2017). lintr: A 'Linter' for R Code. [online] Cran.r-project.org. Available at: https://cran.r-project.org/web/packages/lintr/index.html [Accessed 23 Nov. 2017].

Hughes, J. (2013). CellularAutomaton: One-Dimensional Cellular Automata.

Kar, A. (2016). Bio inspired computing – A review of algorithms and scope of applications. Expert Systems with Applications, [online] 59, pp.20-32. Available at: https://www-sciencedirect-com.ezproxy1.hw.ac.uk/science/article/pii/S095741741630183X [Accessed 29 Oct. 2017].

Khan, A. and Baig, A. (2015). Multi-objective feature subset selection using mRMR based enhanced ant colony optimization algorithm (mRMR-EACO). Journal of Experimental & Theoretical Artificial Intelligence, [online] 28(6), pp.1061-1073. Available at: http://www.tandfonline.com.ezproxy1.hw.ac.uk/doi/abs/10.1080/0952813X.2015.1056240 [Accessed 4 Nov. 2017].

Khushaba, R., Al-Ani, A., AlSukker, A. and Al-Jumaily, A. (2008). A Combined Ant Colony and Differential Evolution Feature Selection Algorithm. Ant Colony Optimization and Swarm Intelligence, [online] 5217, pp.1-12. Available at: https://link-springer-com.ezproxy1.hw.ac.uk/chapter/10.1007/978-3-540-87527-7_1 [Accessed 4 Nov. 2017].

Koza, J. (1992). Genetic programming: on the programming of computers by means on natural selection. MIT Press.

Kramer, O. (2017). Genetic Algorithm Essentials. Studies in Computational Intelligence, [online] 679, pp.11-18. Available at: https://doi-org.ezproxy1.hw.ac.uk/10.1007/978-3-319-52156-5 [Accessed 29 Oct. 2017].

McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The Bulletin of Mathematical Biophysics, [online] 5(4), pp.115-133. Available at: https://link.springer.com/article/10.1007/BF02478259 [Accessed 3 Nov. 2017].

Miller, J. (1999). An empirical study of the efficiency of learning boolean functions using a Cartesian Genetic Programming Approach. In: Proceedings of the 1st Genetic and Evolutionary Computation Conference. Morgan Kaufmann, pp.1135-1142.

Miller, J. (2011). Cartesian Genetic Programming. Natural Computing Series. [online] Available at: https://link-springer-com.ezproxy1.hw.ac.uk/book/10.1007%2F978-3-642-17310-3 [Accessed 13 Nov. 2017].

Miller, J. and Smith, S. (2006). Redundancy and computational efficiency in Cartesian genetic programming. IEEE Transactions on Evolutionary Computation, [online] 10(2), pp.167-174. Available at: https://www.researchgate.net/publication/3418872_Redundancy_and_computational_efficiency_in_Cartesian_genetic_programming [Accessed 19 Nov. 2017].

Moon, C., Kim, J., Choi, G. and Seo, Y. (2002). An efficient genetic algorithm for the traveling salesman problem with precedence constraints. European Journal of Operational Research, 140(3), pp.606-617.

Navid, A. and Bagheri, A. (2013). Cellular Learning Automata and Its Applications. [online] InTech. Available at: https://www.intechopen.com/books/emerging-applications-of-cellular-automata/cellular-learning-automata-and-its-applications [Accessed 31 Oct. 2017].

Oranchak, D. (2009). Cartesian Genetic Programming for the Java Evolutionary Computing Toolkit (CGP for ECJ). [online] Oranchak.com. Available at: http://www.oranchak.com/cgp/doc/ [Accessed 24 Nov. 2017].

Pedroni, E. (2013). JCGP · Equal Parts. [online] Equalparts.eu. Available at: https://equalparts.eu/projects/jcgp/ [Accessed 21 Nov. 2017].

Poli, R., Langdon, W., McPhee, N. and Koza, J. (2008). A field guide to genetic programming. [S.l.]: Lulu Press.

Quast, B.A. (2016). rnn: a Recurrent Neural Network in R

RStudio. (2016). RStudio - Open source and enterprise-ready professional software for R. [online] Available at: https://www.rstudio.com/ [Accessed 23 Nov. 2017].

Russell, S. and Norvig, P. (2009). Artificial intelligence: A Modern Approach. 3rd ed. Pearson, pp.749-753.

Saka, M., Doğan, E. and Aydogdu, I. (2013). Analysis of Swarm Intelligence–Based Algorithms for Constrained Optimization. Swarm Intelligence and Bio-Inspired Computation, [online] pp.25-48. Available at: https://doi-org.ezproxy1.hw.ac.uk/10.1016/B978-0-12-405163-8.00002-8 [Accessed 5 Nov. 2017].

Sarkar, P. (2000). A brief history of cellular automata. ACM Computing Surveys, [online] 32(1), pp.80-107. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.4529&rep=rep1&type=pdf [Accessed 10 Nov. 2017].

Scrucca, L. (2013). GA: A Package for Genetic Algorithms in R. Journal of Statistical Software, [online] 53(4), pp.1-37. Available at: http://jstatsoft.org/v53/i04/ [Accessed 12 Nov. 2017].

Sivanandam, S. and Deepa, S. (2008). Introduction to genetic algorithms. Berlin: Springer, pp.14-39.

Tiobe.com. (2017). R | TIOBE - The Software Quality Company. [online] Available at: https://www.tiobe.com/tiobe-index/r/ [Accessed 27 Oct. 2017].

Turner, A. (2014). About - CGP-Library. [online] Cgplibrary.co.uk. Available at: http://www.cgplibrary.co.uk/files2/About-txt.html [Accessed 21 Nov. 2017].

Turner, A. and Miller, J. (2017). Recurrent Cartesian Genetic Programming of Artificial Neural Networks. Genetic Programming and Evolvable Machines, [online] 18(2), pp.185-212. Available at: https://link.springer.com/article/10.1007%2Fs10710-016-9276-6 [Accessed 13 Nov. 2017].

Venables, W. and Ripley, B. (2002). Modern Applied Statistics with S. 4th ed. New York: Springer.

Wickham, H. (2015). R Packages. Cambridge: O'Reilly & Associates.

Wickham, H. (2016). testthat: Unit Testing for R. [online] cran.r-project.org. Available at: https://cran.r-project.org/web/packages/testthat/index.html [Accessed 23 Nov. 2017].

Wickham, H. and Chang, W. (2017). devtools: Tools to Make Developing R Packages Easier. [online] cran.r-project.org. Available at: https://cran.r-project.org/web/packages/devtools/index.html [Accessed 23 Nov. 2017].

Wickham, H., Danenberg, P. and Eugster, M. (2017). roxygen2: In-Line Documentation for R. [online] cran.r-project.org. Available at: https://cran.r-project.org/web/packages/roxygen2/index.html [Accessed 23 Nov. 2017].

Willighagen, E. and Ballings, M. (2015). genalg: R Based Genetic Algorithm.

Wolfram, S., 1983. Statistical mechanics of cellular automata. Reviews of Modern Physics, 55(3), pp.601–644.

Wolfram, S. (2002). A new kind of science. 1st ed. Champaign, IL: Wolfram Media.

Yadav, N., Yadav, A. and Kumar, M. (2015). An introduction to neural network methods for differential equations. Dordrecht: Springer, pp.29-54.

Yang, X. and Karamanoglu, M. (2013). Swarm Intelligence and Bio-Inspired Computation. Swarm Intelligence and Bio-Inspired Computation, pp.3-23.

Yang, X. (2014). Particle Swarm Optimization. Nature-Inspired Optimization Algorithms, [online] pp.99-110. Available at: http://www.sciencedirect.com.ezproxy1.hw.ac.uk/science/article/pii/B9780124167438000075 [Accessed 6 Nov. 2017].

Zhang, G. (2009). Neural Networks For Data Mining. Data Mining and Knowledge Discovery Handbook, [online] pp.419-444. Available at: https://link.springer.com/chapter/10.1007/978-0-387-09823-4_21 [Accessed 3 Nov. 2017].