A Report submitted in partial fulfilment of the regulations governing the award
of the Degree of
BSc (Honours) Computer Games Software Engineering
at the University of Northumbria at Newcastle

Genetic Algorithm - Eternity II

A genetic algorithm for solving packing problems based around the Eternity II board puzzle.

Mark Logan

2007/2008

Declaration of Authorship

University of Northumbria at Newcastle

School of Informatics

I, Mark Logan, confirm that this report and the work presented in it are my own achievement.

I have read and understand the penalties associated with plagiarism,

Signed:

Date:

Acknowledgements

I would like to thank Dr. D. A. Harrison B.S.c (Eng) (London) A.C.G.I. (Stirling) for his continuous support throughout the project.

Also many thanks to the IT Support team from the School of Computing, Engineering & Information Science for setting up the lab for my two days of testing.

Abstract

Eternity II is an edge matching puzzle consisting of a 16x16 game board and 256 playing pieces, each divided equally into four triangles along the diagonals. As they are placed on the game board each adjacent piece must have a matching background colour and the two equal halves of a coloured shape. The same shape can occur in different colours. There are several hard constraints within this packing problem, an example of which is that an edge piece can only be placed along the edge of the board and not in a corner or centre square. The penalty values of each constraint will be considered. This dissertation explores the use of a Genetic Algorithm (GA) to place the playing pieces correctly on the game board. The graphical user interface (GUI) is implemented in Java and the Genetic Algorithm in C++.

The prize for the first correct solution to the puzzle is $2,000,000, which is provided by Tomy UK Ltd.

Contents

Chapter 1

Introduction

1.1 Introduction

Eternity II (Tomy 2007*a*) is an edge matching board puzzle. It is the second puzzle to be released by Christopher Monckton in the Eternity series. Eternity I was based on 209 irregular shaped polygons; it is unknown why this particular number of polygons was used. The pieces had to be positioned into a large regular dodecagon (a twelve sided polygon). Eternity I was solved by two mathematicians from Cambridge just over a year after its release (Wainwright 2001). To solve the puzzle Alex Selby and Oliver Riordan used a breadth-first search (BFS) method up until a predetermined point and then an exhaustive search to position the remaining pieces into the dodecagon.

Figure 1.1: Eternity II

Eternity II is a puzzle based on a 16x16 board. There are 256 square playing pieces which are each split into four equal triangles along the diagonals. Each triangle has a background colour and

half a coloured shape drawn along the long edge. Certain shapes occur in different colours. For the pieces to be correctly placed on the board, adjacent pieces must match in background colour, shape and the shape colour. There is a single starting piece, piece number 139, which has to be positioned in square I8 as specified in the game rules. It is not stated why piece number 139 is used or why it has to be positioned in square I8.

The prize provided by Tomy UK Ltd. for the first correct solution to the puzzle is $2,000,000. The publishers are keeping all candidate solutions received unopened until the 31st December 2008 when they will be opened in order of arrival and the first correct solution found will be the winner. If no correct solution is found, the deadline will be extended until 2009 and then finally to 2010. Tomy have stated in the rule book that they may extend the final date if no correct solution is found. The rules of the game state that any correct solutions to the puzzle cannot be made public. Solutions must be kept confidential and the judges can disqualify any entry that becomes publicly known. In the official press release, it is stated that there are thousands of ways of solving Eternity II (Tomy 2007*b*). However, it is not stated how this is determined.

A brute force method of solving this problem, which would mean trying every single combination of positions for every single piece would be very slow. Not taking into consideration that the collection of playing pieces contains edge and corner pieces, the number of unique ways in which all the playing pieces can be placed on the board is 256!*4^{256} (Owen 2007).

256!*4^{256} = 1150145597068748297829844185558918847289535734095127268443012552361001622090694279390786707679310092638937938143426141647826908758675016940678798118282630999110028245022698637679006598082677135244714985216309627216978185027472968697325977719268417095542431799037585076835828332893060931249167302663185701205795993570420438017628979689679132883923583499264990803671061268089 5

5655757788055130822660925636061093552223637164895956507505729112
9730566766819588320040149109420805497846561003414626576980315893
3291635166357023361314365890301804659414182975634197604255209345
4321091341793769032610361177538560000000000000000000000000000000
0000000000000000000000000000000 $\approx 10^{661}$

This number is derived from the number of playing pieces and the number of times each piece can be rotated. Taking into consideration the starting piece, corners and edge pieces there are still 1*4!*56!*195!*4^{195} combinations (Owen 2007).

1*4!*56!*195!*4^{195} = 1115311984696224928997596089719145954209471972
7435051948772746910194651345506937767635819175854725440909825012
7814533366537520303714516904568624054580457715452150218419881603
3426898623130040291787113248494045052560522750179877345779077672
9161993899104357165807551517423390728504818872213878409929627053
4509243817733468845560076206020218546046771585537232438151046330
7935782263027804160979981707044339574468725077499533271769752898
0238829697009880225197521531101554100823012940967586668104862243
2256000000000000000000000000000000000000000000000000000000000000
$\approx 10^{557}$

This value is calculated from the starting piece, number of corner pieces, the edge pieces, and then the 195 centre pieces. Matthew Champion from Texas A&M University has done a lower limit calculation of the number of hydrogen atoms in the Universe with a result of 10^{79} (Champion 1998). Champion does not provide a higher limit estimation. This calculation shows the sheer size of the number of possible combinations within the Eternity II puzzle.

This dissertation proposes the use of a Genetic Algorithm (Mitchell et al.

1992) to place the playing pieces correctly on a game board. Given the type of puzzle and playing pieces available, there are no obvious soft constraints as adjacent pieces either match or they do not. There are several hard constraints within this packing problem and the order in which they should be evaluated and penalised will be considered. There are four smaller clue versions of the puzzle that provide the locations of known pieces in a single solution to the main puzzle (Tomy 2007*a*). It is not known if using these clue pieces limits the number of possibly solutions. These puzzles come in two sizes, 36 or 72 pieces. The algorithm will be tested on these versions first.

The user interface (GUI) will be implemented in Java (Flanagan 1996), allowing the interface to be integrated with a Genetic Algorithm written in C++ (Dale et al. 2002), a compiled language enabling quick execution of the algorithm. Genetic Algorithms (GAs) are discussed in more detail in Chapter 4.

1.2 Dissertation Structure

Chapter 2 - Packing Problems

This chapter discusses common packing problems for both regular and irregular shaped containers which includes edge-matching puzzles that are similar to Eternity II.

Chapter 3 - Heuristic Methods

A review of heuristic methods and the type of problems they are used to solve. This also includes methods to overcome local maxima and minima problems.

Chapter 4 - Genetic Algorithms

A review of Genetic Algorithms used to solve the problem of packing regular shapes.

Chapter 5 - Design

This chapter details the constraints and the penalty level that these constraints hold within the problem and includes a description of the data structure and the breeding, selection and mutation methods that are used in the algorithm. Unified modelling language (UML) (Fowler & Scott 1999) diagrams for the interface are included.

Chapter 6 - Experimentation

Details of any amendments made to the genetic algorithm whilst testing on the smaller versions of the puzzle and any changes made to the penalty levels of the constraints.

Chapter 7 - Evaluation and Recommendations

This chapter will analyse the suitability of the developed application to solving the problem. Suggestions for further work will be made and possible improvements discussed.

Appendices

The Terms of Reference is included along with a glossary. The code for the application and all other documents are included on the CD.

References

Chapter 2

Packing Problems

2.1 Packing Problems

This chapter gives a general overview of packing problems and provides a brief description of several of the most common problems. The chapter also includes a short introduction into edge-matching puzzles.

2.2 Bin Packing

A bin packing problem (de Souza et al. 2008) is the packing of multiple objects of varying sizes into a box/bin that occurs within the production, construction and distribution industries. There are other variations to the bin packing problem, where items have multiple attributes, for example, when not only the size but the weight is also of importance. There are also two-dimensional and three-dimensional packing problems, these are mentioned below in Stock Cutting. Packing the items into the bins, not exceeding the maximum weight or size of the bin and using as few bins as possible.

2.3 Knapsack Problem

The Knapsack problem (Martello & Toth 1990) is similar to the bin packing problem except that the items in a knapsack problem have both a cost that could be a size or weight and a value associated to them which, for example could be the beneficial value of the item or profit if it were to be sold. The problem is utilising no more than the space available whilst gaining maximum value. The problem is known as the 0-1 Knapsack Problem when you are working with a limited number of each item, either one or zero.

2.4 Stock Cutting

Stock cutting problems (Fister et al. 2007) arise on production lines. For example, on a cardboard or sheet metal cutting line, any moulds and templates will need positioning on the available rolls of material to minimise wastage when they are cut. The positioning of the moulds to utilise the space on the roll is important. Another example is within the clothing industry. Coats and other clothing items are made up of multiple pieces of material all stitched together. A batch of clothing will contain different sizes of many single items. All the individual pieces that make up these items will need arranging so they can be cut from the available material. Arranging all the pieces could involve mixing the pieces of a small suit with the pieces of a large suit to utilise the space on the roll effectively.

2.5 Edge-Matching

An edge-matching puzzle (Demaine & Demaine 2007) is very similar to a jigsaw puzzle. However, the fundamental difference between them is that a jigsaw has one possible solution whereas an edge-matching puzzle can have many possible

solutions. A jigsaw involves the linking together of individual pieces and making a single image. An edge-matching puzzle consists of multiple pieces which are all arranged together so that adjacent pieces match creating one of possibly many images. In an edge-matching puzzle individual pieces do not interlock. The pieces are positioned onto the game board which then defines the final shape. For example Eternity II uses a sixteen by sixteen piece board.

Like jigsaws, edge-matching puzzles have distinctive borders, so it is possible to separate the pieces into different categories of playing pieces, the categories can be corner, edge and centre pieces. A more complex variation of the edge-matching puzzle is signed edge-matching where the pieces not only have a colour but an image that needs to be matched with an adjacent piece. This usually consists of an image cut in half horizontally. Eternity II is a signed edge-matching puzzle and the playing pieces consist of a background colour and half a coloured shape that needs to be matched with another piece.

2.6 Conclusion

This chapter detailed several of the most common packing problems and provided a brief introduction into edge-matching problems. Eternity II can be considered an edge matching puzzle as there are multiple solutions to the problem. The project supervisor though insists it is a jigsaw.

Chapter 3

Heuristic Methods

3.1 Heuristic Methods

This chapter gives a general overview of five of the most commonly used heuristic methods.

Heuristic methods aim to provide a reasonable solution to a problem in a feasible period of time. An analytical method will, given enough time, find the optimal solution, after searching the complete search space. The travelling salesman problem (TSP) (Gutin & Punnen 2002) with a few thousand cities is a problem suited to a heuristic method due to the time it would take an analytical method to search through every possible tour combination. The TSP consists of multiple cities with known distances between them, the problem is finding the shortest route between all the cities, only visiting each city once. Even though the distances between each city is known at the beginning, there is no efficient way of calculating the shortest route between them all. To completely search a TSP consisting of thousands of cities would take years whereas a heuristic method could provide an acceptable solution in hours.

3.2 Hill Climbing Algorithm

Heuristic methods are used to search through problem spaces, hoping to either find the global minimum or global maximum in the space.

Searching through the problem space can be explained using two scenarios; either a hiker starting at the bottom of the valley who wants to walk to the highest peak along the mountain range or a skier dropped at a random point along the mountain range and wanting to ski down to the lowest valley. The hiker will look both left then right, decide which way looks the steepest and head a short distance in that direction, this is then repeated.

Figure 3.1: Search Space

The hiker will gradually converge upon one of the peaks along the mountain range (local/global maxima points on the graph, figure 3.1). If the hiker has arrived at one of the local maxima in the search space the search will become stuck as any progression in either direction will be seen as a deterioration in fitness of the current location. Searches stuck in local maxima and minima are a major problem that heuristic methods have to overcome.

3.3 Neural Networks

The design of a Neural Network (NN) (Dayhoff 1990) is based on the structure of the brain and so NNs are built out of many interconnected processors (also known as units or neurons).

Neural networks are used to solve many problems, for example optimisation problems such as the TSP and also vision and speech recognition problems. They are also used to find patterns in data feeds, for example stock market values and forecasting exchange rates based on previous market data (Kuan & Liu 1995).

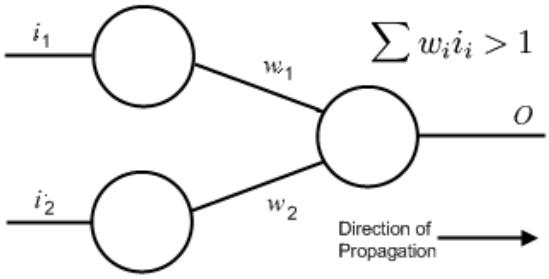

Figure 3.2: Neural Network

The neural network shown in figure 3.2 takes two inputs along i_1 and i_2, these inputs are multiplied by the weights of the link. If this new value hits the threshold of the next node along, the node will fire, sending a signal along to the next node. The new signal is multiplied by the weight of the link and is summed together with any other inputs at the next node.

Neural networks are not programmed to give certain outputs based on specific inputs, they are taught. For example training a neural network the OR function would require the teacher to feed the network two numbers via the inputs and also provide the network with the expected output, the solution to the OR. Over the learning period the network will balance the internal weights between each node, these node weights are used in calculating the network output. The nodes use all the values sent along their inputs and the weights to calculate a total, if a certain threshold is reached the node will fire a signal along

to the next node. Changing the weights of the nodes will hopefully produce a network capable of providing the correct output. The network is trained using the back propagation method, a process in which the initial node weights are generated at random. The inputs are then fed into the network and the output calculated. An error value is then calculated based on the expected output and the produced output. The error value and the learning rate are then used to calculate new node weights, which are applied to the nodes. The process is repeated until the network produces the expected output. This method is considered as supervised learning as it is possible for the teacher to step in at any time and correct any errors in the network.

Unsupervised learning is where the NN attempts to recognise any patterns in the data independently. The NN may separate the data into several different categories, but it is then up to the human to distinguish what these categories mean. The NN will be quicker than a human at separating the data in the different possible groups as the NN is faster at processing data.

It is not possible to know what a NN has learnt to recognise. For example, feeding a NN two sets of pictures of tree lines, one with objects hidden within them and one set without, you would hope that the NN would be able to distinguish between the two sets by spotting the objects in certain pictures. The NN may on the other hand pickup on other characteristics of the picture such as the weather for example and not actually if anything is hidden behind the tree line, but ultimately it is up to the human to decide why the NN has separated the pictures into different categories.

As a neural network is trained it is required to have a method of measuring how effectively it is learning. The most commonly used method is the root mean squared (RMS) error (Dayhoff 1990). The RMS value will approach zero as the NN re-balances the internal weights to produce the correct output, though this may never reach zero.

The network may get stuck in a local minimum whilst seeking the lowest

RMS error possible (the global minimum) and will need a little persuasion to continue on with the search. One method of attempting to resolve this problem is to inject varying levels of noise into the network forcing the search to jump a random distance along the graph. As it is usually unknown how far the search needs to jump to escape the local minimum, this method may or may not work.

Neural networks are often used for pattern recognition problems, for example, spotting patterns within collections of data, images and sounds.

3.4 Ant Searches

Ant Searches (Dorigo & Blumb 2005) are based on the day to day activities of some species of ants. Ants spend the majority of their time searching for food. There is a lot of luck involved in finding food as ants initially tend to walk in random directions from the nest. As ants leave the nest they deposit a trail of pheromone. On the journey back to the nest after finding food the ant will retrace their trail, depositing more pheromone. Other ants are attracted to this trail and follow it to the food, as they then return to the nest with food they also leave a trail of pheromone adding to the trail already there. This causes the pheromone concentration to increase attracting more ants, repeating the process until the food supply runs out. Some species of ant leave repellent pheromone along dead trails to discourage other ants from following them. A trail can be considered dead once the food supply at the end of it has run out.

Over time pheromone evaporates. Evaporation causes pheromone on longer trails to fade out quicker than pheromone on shorter trails given that ants on longer trails are likely to be further apart, meaning that less pheromone is deposited at any given point on the trail and thus does not take as long to evaporate. As the pheromone evaporates from longer trails the ants move towards the shorter trails where pheromone is higher, this allows the search to kill off weaker solutions. There is the possibility that an ant will wander off in a random

direction, not necessarily following a pheromone trail or other ants. This could help the search escape local maxima as this individual ant could luckily find a shorter route and therefore a better solution.

Ant searches are usually applied to route finding problems such as the TSP and travelling purchaser problem (TPP) (Bontoux & Feillet 2008). The TPP is a variation of the TSP problem, where the salesman has to visit markets at different cities to purchase a list of goods, minimising the travelling distance and overall purchasing cost.

3.5 Harmony Search

Harmony Search (HS) (Geem et al. 2001) is a heuristic method based on the process a group of musicians will go through to find a better harmony. The tuning techniques that the musicians use are replicated in the heuristic method. This process is iterated until the perfect harmony is found.

HS has been applied to route finding problems and construction problems, such as the school bus routing problem (Geem et al. 2005). The school bus routing problem is an optimisation problem based around minimising the number of buses used to transport children to school, working within a fixed time frame and without overloading the buses.

3.6 Simulated Annealing

Simulated Annealing (SA) (Kirkpatrick et al. 1983) is a heuristic method based on the heating and slow cooling of a material such as copper or glass. The process aims to minimise the imperfections within the material and increase its malleability. The material begins at its highest temperature where the atoms are active and in the most unorganised state. The material is then cooled, producing a more organised state. The process of heating and cooling is repeated, gradually

lowering the upper temperature. The best solution will be provided when the temperature has reached its lowest value. If the search did not start at a high enough temperature or is cooled at the wrong speed it may get stuck in local maxima.

SA is a meta-heuristic algorithm that employs another heuristic method to move the algorithm between different temperatures during the annealing process.

SA has been shown to work well in the design process for computer components, such as circuit boards (Kirkpatrick et al. 1983). It is possible to prioritise the order in which individual components should be positioned on the circuit board based on component requirements. This can be implemented in the annealing process by positioning important components at high temperatures and leaving the least important components until lower temperatures.

3.7 Genetic Algorithms

Genetic algorithms are discussed in chapter four.

3.8 Summary

This chapter detailed several of the most common heuristic methods and problems they are used to solve.

Chapter 4

Genetic Algorithms

4.1 Genetic Algorithms

This chapter gives an overview of GAs and the situations where they have been used.

GAs are search techniques used in optimisation problems such as the TSP and the timetabling problem (White & Wong 1988). A GA is a heuristic method based on the process of evolution suggested by Charles Darwin (Darwin 1859). Several stages that occur in evolution are present in the algorithm. The stages of a GA are:

```
initialisePopulation();  // Initialise the population
evalutatePopulation(); // Evaluate the fitness
while(finishCriteriaNotMet)  // Loop until finishing criteria are met
{
  selectParentsToBreed();  // Select the parent chromosomes to breed
  breedParents();  // Recombine the genes of the selected parents
  mutateOffspring();  // Mutate the new offspring
  evalutateOffspring();  // Evaluate the fitness of the new offspring
  replaceChromosomes();  // Replace the new surviving offspring
}
```

The random population of chromosomes is generated before the main loop

of the algorithm begins. Once the population is created the algorithm will then loop until the finishing criteria are met. The criteria can be, for example, until a certain number of generations have occurred or a fitness value is reached. Several stages take place inside the loop when creating each new generation. First the fitness value of each chromosome is calculated using the fitness function, the fitness function is discussed in more detail in section 4.7. After the fitness of each chromosome has been calculated a selection method is used to pick which chromosomes will be used for breeding. (Selection methods are discussed in section 4.5) A new chromosome is then created using a reproduction method, which is discussed in section 4.4. Mutation is then applied to a select few new chromosomes. (Mutation is covered in section 4.6) The new chromosome then replaces a weak chromosome from the population to create a new generation. These steps all take place when creating a new generation of chromosomes.

GAs, like all other heuristic methods are prone to finding local maxima within the search space. To help a GA attempt to escape local maxima, mutation is applied to randomly selected chromosomes as they are generated, allowing a new chromosome structure to be created that would not be possible by breeding on its own. Mutation cannot always help the search escape or avoid local maxima, so other methods sometimes need to be applied such as restarting the algorithm or using a different breeding method for one generation.

4.2 Applications

Problems can fall into one of two categories, non-combinatorial and combinatorial. Non-combinatorial problems are when individual genes only affect the total fitness of the chromosome. For example, the knapsack problem can be considered a non-combinatorial problem. The genes within the chromosome represent items that are to be packed in the knapsack. The position of each gene within the chromosome does not affect the fitness of any other gene, only the

fitness of the chromosome as a whole.

In a combinatorial problem the position of each gene within the chromosome can affect the fitness of other genes. The Eternity II problem is a combinatorial problem, as each gene slot within the chromosome represents a certain position on the playing board. For a solution to be valid, adjacent playing pieces have to match, so genes are compared to other genes within the same chromosome to calculate a fitness. This means that the order in which the genes appear in the chromosome is important as this also affects the fitness.

4.3 Solution Representation

The design of the chromosome structure is one of the first problems that needs to be overcome when creating a GA. The chromosome encoding depends greatly on the problem that is being solved, a binary encoded bit string is not always suitable as to use binary encoded chromosomes it must be possible to represent solutions to the problem as strings of ones and zeros.

Chromosome 1	1001010110101011
Chromosome 2	0001001101010011
Chromosome 3	0101001101010101

Table 4.1: Binary Encoded Chromosomes

For complex problems where solutions cannot be simply represented by a list of binary digits, the chromosomes structure will need to directly represent the specific problem; this is achieved using real value chromosomes. A real valued chromosome (Adebola 1996) could be represented by a gene made of a double, a string or even a custom object representing a specific element within the solution (as shown in table 4.2).

Chromosome 1	125.21 124.15 458.78 214.57 365.98
Chromosome 2	hello because why they that free
Chromosome 3	(123) (425) (642) (647) (873) (267)

Table 4.2: Real Value Encoding

4.4 Reproduction Methods

In a GA new chromosomes are created each generation and are substituted into the population using a reproduction method. The normal method of reproduction is called crossover. The crossover process requires at least two chromosomes from the population and these chromosomes are known as the parent chromosomes. The parent chromosomes are selected based on their fitness; usually high fitness chromosomes are chosen in the hope that when combined they will produce another high fitness chromosome.

The two chromosomes are combined using N number of crossover points. If two chromosomes are combined using 1-point crossover, a single random point is chosen within the two chromosomes. This point is used to divide the chromosomes into two pieces. A new chromosome is then created taking opposite pieces from each divided parent chromosome. Single point crossover is shown in figure 4.1.

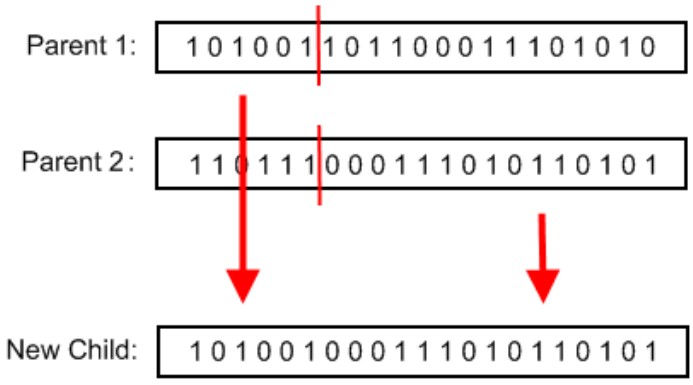

Figure 4.1: Single Point Crossover - Binary Encoding

Using a real value chromosome encoding format, crossover between two chromosomes will generate invalid chromosomes. If the chromosome can only contain unique genes, crossover between the chromosomes will generate chromosomes with duplicate and missing genes. Single point crossover using a real value encoded chromosome is shown in figure 4.2. It can be seen that single point crossover has introduced duplicate genes into the chromosome, genes 04, 05, 10 exist twice in the new chromosome and 07, 08, 09 are now missing.

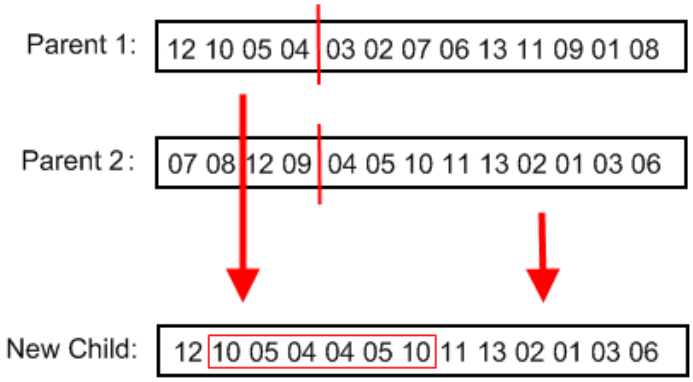

Figure 4.2: Single Point Crossover - Real Encoding

For the Eternity II problem normal crossover would not work. (Each chromosome must contain all the unique playing pieces, any chromosome that is missing any number of playing pieces could only generate an invalid solution) To resolve this problem the crossover and repair idea was considered (Mitchell 2005). However this would add numerous extra steps to the generation of each new chromosome, each new chromosome would need to be checked for duplicate genes and then repaired, adding any missing genes. This would slow the algorithm down so the decision was taken not to use this crossover and repair method as nothing would be gained from doing so.

The internal gene shuffle was an alternative idea. Any chromosome selected for breeding would technically breed with itself. Two random points are selected within the chromosome and a fixed length of genes starting at each chosen point

would simply be swapped. Using this method is was possible to avoid having to check each chromosome after breeding was still capable of generating valid solutions and did not contain duplicate genes.

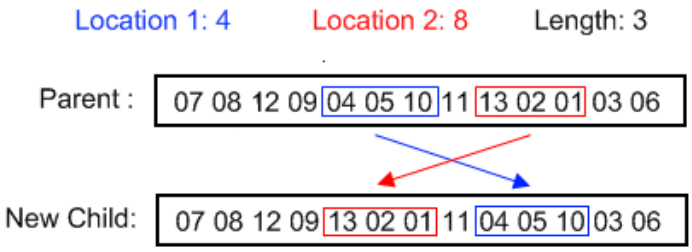

Figure 4.3: Internal Shuffle - Real Encoding

4.5 Selection Methods

Before new chromosomes can be generated and included in the population, parent chromosomes from which they are bred need to be selected. It is hoped that breeding high fitness chromosomes will produce high if not higher fitness offspring. The highest fitness chromosomes need to be selected from the population before breeding can take place, this can be done using many different methods.

Roulette wheel selection (Mitchell 1998) is based on the principle of a roulette wheel at a casino. The chromosomes in the current population are assigned a section of the wheel based on their fitness, the higher the fitness of the chromosome the larger the section thus increasing the chance of being selected. The wheel is then spun, the chromosome related to the selected section is then included in the list of parent chromosomes used to breed the next offspring. The spinning process is repeated until enough parent chromosomes are selected.

By using roulette wheel selection it is possible to miss some of the best chromosomes from the population as there is no way of ensuring that a

chromosome will be selected. If the best few chromosomes are not selected for breeding it is possible that they could be destroyed by mutation or crossover during the next generation. It is important to make sure that the GA selects a few of the best chromosomes from the population each generation which is done using a method called Elitism (Mitchell 1998). Elitism is when a select number of the fittest chromosomes are copied over to the next generation. Copying anything more than the fittest chromosome would require the population to be sorted by fitness value which will slow down the algorithm.

Tournament selection (Mitchell 1998) is when two chromosomes are chosen from the population, this can be via random selection or another selection method. The two chromosomes are then compared against each other and the chromosome with the higher fitness is selected to be a parent. This process is repeated until enough parent chromosomes have been selected. If each chromosome from the population must be selected once for tournament selection, the selection method will have built in elitism of at least the fittest chromosome from the population.

4.6 Mutation

After each new chromosome is generated, there is a possibility that mutation is applied to it. Mutation alters the chromosome, to a format that crossover cannot generate. In a binary encoded chromosome, if mutation is successful the selected bit will be swapped, a 0 becoming a 1 and a 1 becoming a 0. In the Eternity II problem, mutation is used to rotate the playing pieces on the board by either 90, 180 or 270 degrees, which is randomly chosen each time. This cannot be accomplished by crossover, as crossover simply swaps genes round the chromosome. The pieces are not moved around the board during mutation, this is done during crossover and when the GA has become stuck for a fixed number of generations. Only rotating the genes and not moving them around

during mutation may limit the development of a chromosome.

The mutation rate is usually very low, less than 1%, this is because if mutation is applied too often the search can just becomes a random search.

4.7 Evaluation of Chromosome Fitness

As the GA creates each new generation there needs to be a way of calculating how well the algorithm is progressing and if the fitness of any of the chromosomes is increasing. The fitness function within the algorithm calculates how well each chromosome (a solution) fits the solution requirements.

The requirements for a valid solution can be broken down into two categories, hard and soft constraints. Hard constraints are a necessity and any solution that does not satisfy them cannot be considered a valid solution. Soft constraints are not necessarily required in the solution, but if satisfied they will increase the fitness of a solution.

Each constraint is given a penalty value, these values vary depending on the importance of a particular constraint. Usually the greater the importance of a constraint, the higher the penalty value. When the fitness of the chromosome is calculated all the constraints are checked, the penalty value of any constraint that is not met is added to the chromosomes fitness value. A weaker solution that does not match many of the constraints will have a high penalty value and will therefore not be selected to be used during the crossover process. Penalty values and the fitness function are a way of enabling the algorithm to distinguish between weak and strong chromosomes.

4.8 Summary

This chapter detailed each stage required in a Genetic Algorithm. The solution presentation and reproduction methods for the Eternity II problem are discussed

further in chapter 5.

Each chromosome will be represented by a vector, the vector will consist of structs representing each gene (a gene is a playing piece).

Tournament selection will be used as this is the fastest method for this puzzle and will ensure that the fittest chromosome is always copied over to the next generation. The internal shuffle will be used for the reproduction method as this will not create any invalid genes and no repair function will need to be applied to each chromosome. Mutation will be applied to randomly selected chromosomes to rotate the playing pieces.

Chapter 5

Application Design

This chapter details how the GA was developed, including the problem constraints and fitness functions.

An algorithmic design for the GA was not required as this can be found in most GA books (Mitchell 1998). A simple GA structure is explained in section 4.1:

```
initialisePopulation();  // Initialise the population
evalutatePopulation(); // Evaluate the fitness
while(finishCriteriaNotMet)  // Loop until finishing criteria are met
{
  selectParentsToBreed();  // Select the parent chromosomes to breed
  breedParents();  // Recombine the genes of the selected parents
  mutateOffspring();  // Mutate the new offspring
  evalutateOffspring();  // Evaluate the fitness of the new offspring
  replaceChromosomes();  // Replace the new surviving offspring
}
```

The Eternity II puzzle has been broken down into two smaller puzzles due to the difficulty the algorithm had trying to solve Eternity II as a whole (discussed in chapter 6). The border puzzle and the centre puzzle. These two puzzles are solved in the same population, using the same chromosomes. A single

chromosome contains the same number of genes as the puzzle board contains squares, this allows each gene to represent a position on the board.

5.1 Playing Pieces

The Eternity II problem consists of 256 unique playing pieces, each split into four equal triangles along the diagonals. Each triangle has a background colour and half a coloured shape drawn along the long edge. These playing pieces were placed into a data file so that they could be loaded into the GA. Each record in the data file contained fourteen integers, these represented the playing piece ID, the background colour of the triangle, the shape, the shape background colour of each triangle and the playing piece type (corner, edge or centre). These colours and shapes were indexed as numbers to allow quick comparisons to be made between different playing pieces. An example record from the data file can be seen in Table 5.1.

17 3 1 2 2 2 1 2 2 1 4 5 6 2

Table 5.1: Example data record

The record in table 5.1 makes up a single playing piece, which represents a gene within the chromosome. The playing piece represented by this data is shown in figure 5.1.

Figure 5.1: Eternity II - Playing Piece

The first figure in the record (17) is the reference number of playing piece. Each playing piece is numbered on the reverse, this number is the unique reference number. The second, third and forth figures (3 1 2) make up the bottom triangle. The 3 represents the yellow background colour, the 1 represents the star shape and the 2 represents the light blue colour of the shape. The fifth, sixth and seventh (2 1 2) represent the left hand triangle, the next 3 figures represent the top triangle and then the eleventh, twelfth and thirteenth figures represent the right hand triangle. The final figure, the 2 relates to the playing piece type, 0 is a corner piece, 1 is an edge piece and a 2 is a centre piece.

The chromosome is represented by a vector container, consisting of gene structs. Each gene struct has three members. The playing piece reference number, rotation index and the playing piece type.

The chromosome gene array is represented by a vector container as this will allow for dynamic array lengths. This enables the application to work with different sized puzzles as the array length will need to grow or shrink, to be able to allocate enough space for all the playing pieces on the board. The fitness of both the centre and the border is required as these are solved separately. Using a vector container probably slows the algorithm down slightly due to dynamic allocation during runtime.

```
struct chromosome
{
  vector<gene> geneArray;
  int iBorderFitness;
  int iCentreFitness;
};
```

Each playing piece is represented by a gene struct. The struct consists of a reference number which refers to a certain playing piece, a rotation index which is used to calculate the orientation of the playing piece. The rotation index can be either 0, 1, 2 or 3 depending on the number of times it has been rotated clockwise by 90 degrees. The last member of the struct is the playing piece type.

The playing piece type is used by the fitness function to work out if the right type of playing piece is in the right location. The piece types are 0 for a corner piece, 1 for an edge piece and 2 for a centre piece. The reference number is used as a look up figure when requesting data from the playing pieces array.

```
struct gene
{
  int iRefNumber;
  int iRotationIndex;
  int iPieceType;
};
```

The playing pieces are stored in `gamePieces` structs that hold the information from the data file. The relevant struct is accessed depending on the gene struct playing piece reference number.

```
struct gamePieces
{
int iRefNumber;
int iPieceInfo[4][3];
int pieceType;
};
```

In the `gamePieces` struct the reference number is used to relate to an actual playing piece. This figure is used when outputting a solution as it is an understandable figure that the user can relate to. The information about an individual playing piece is stored in a 2D array `iPieceInfo[4][3]`. The 2D array consists of the three pieces of information about each of the four triangles that make up a single piece (an example piece is shown in table 5.1). This includes the background colour, the shape and the shape background colour. The playing piece type is stored so that the algorithm knows if it is currently working with a corner, edge or centre piece.

5.2 Initialisation

The width of the board is known by the GA as this is read from the settings data file when the application begins. Using the length of the chromosome and the board width it is possible to calculate if a certain gene within the chromosome is in a border location. When solving the border, breeding only takes place amongst the border locations within the chromosome. The GA cannot swap pieces between the centre and the border. Solving the border separately requires that all the border and corner pieces are positioned within the border locations in the chromosome from the very beginning.

When the population is created, all the chromosome are randomly generated. This means that all the genes in the chromosomes need rearranging to position edge and corner playing pieces in the border locations within the chromosome. This is done once to every chromosome in the population.

Clue pieces that are discussed in section 5.8 can also be provided to the GA before it starts running. Clue pieces can either be requirements of the puzzle (for example the starting piece, number 139 has to be positioned in square I8 for an Eternity II solution to be valid) or other playing pieces that have known positions within a valid solution. These playing pieces are set in each chromosome within the population after the chromosome has been sorted into border and edge pieces. Once these clue pieces are set, they must not be moved or altered via breeding or mutation.

5.3 Constraints

There are only hard constraints within the Eternity II problem. Soft constraints cannot be applied to a solution as a solution is either right or wrong. As the information about each playing piece has been broken down to a low level, it is possible to apply constraints to the background colour of each triangle, the

shape in each triangle and the shape background colour of each triangle on each playing piece.

The constraints and penalty values work backwards compared to a normal GA, the higher the fitness value of a chromosome the better the solution is. If a gene satisfies any of the constraints, the chromosome with score the constraint value. There is no particular reason why penalty values were done this way. When completing a jigsaw, the corner and edge pieces are first separated from the rest. The border is then the first part of the solution to be constructed.

Hard constraints have been applied to the background colour of the triangle, the shape and the shape colour. If adjacent pieces match on any of these three criteria they score the constraint value. The constraint values are shown in table 5.2. These are example values as they were changed during experimentation. Edge and corner playing pieces also score constraint values if they are positioned in the corners and along the edge of the playing board. This enables the GA to distinguish between solutions that have pieces in the correct positions on the board (corner pieces in the corner and edge pieces along the edge) and not just solutions that have matching adjacent pieces. Constraints have also been applied to edge and corner pieces so that they are rotated to match the board edge. This again enables the GA to construct a stronger solution through crossover as a solution with a valid border will have a higher fitness and therefore more chance of being selected for breeding.

5.4 Selection Method

Tournament selection is used to select which chromosomes will be used for breeding. This process takes the unsorted array of chromosomes in the population and compares the two chromosomes at opposite ends of the population. The chromosome with the highest fitness is included in the array of chromosomes to use for breeding whereas the other chromosome is added to the array of

Corner piece in the corner	25
Corner piece match the board edge	15
Edge piece on the edge	10
Edge piece match the board edge	10
Corner piece match background colour with an adjacent piece	5
Corner piece match shape with an adjacent piece	5
Corner piece match shape colour with an adjacent piece	5
Edge piece match background colour with an adjacent piece	3
Edge piece match shape with an adjacent piece	3
Edge piece match shape colour with an adjacent piece	3
Centre piece match background colour with an adjacent piece	1
Centre piece match shape with an adjacent piece	1
Centre piece match shape colour with an adjacent piece	1

Table 5.2: Constraint Values

chromosomes not used for breeding. This process is repeated moving one chromosome in from either end of the population until the middle of the population is reached. This method ensures that the highest fitness chromosome will survive to the next generation and will be selected to breed but does not necessarily mean that the second best will also be chosen. Using this selection method enables elitism and ensures that at least the fittest chromosome is selected without having to carry out any extra calculations on the population.

Slight variations can be applied to tournament selection (Mitchell 1998), Mitchell suggests selecting two random chromosomes from the population. After calculating which one should be used for breeding they are both returned to the population enabling them to possibly be selected again during another tournament. Returning the chromosomes to the population does not enable elitism, there is nothing to ensure that any of the fittest chromosomes have been selected and will be included in the next generation, so this method was not used as there are no obvious benefits.

5.5 Breeding Method

After all the chromosomes have been separated into the two groups, those that will breed and those that will not by the selection method as described in section 5.4, those that have been selected to breed are bred individually with themselves. If they were bred with other chromosomes via the normal crossover method duplicate and missing genes would be introduced. Duplicate and missing genes are discussed in section 4.4.

A slightly different breeding method is applied to the border and the centre of the puzzle. When breeding the border part of the chromosome, two random border locations within the chromosome are selected. The genes at these locations are then simply swapped. It is not checked to see if swapping the genes will improve the fitness, as the puzzle is designed in such a way that just because two pieces match does not mean they are in the right position. Therefore the border may be temporarily made worse, to then be made better. When breeding the centre part of the board after the border has been solved, an extra step is included. After a gene within the centre of the board has been selected to be swapped, it is checked that the chosen gene is not a clue piece. Clue pieces are discussed in section 5.8. These pieces are set in each chromosome during the initialisation of the population and cannot be moved or mutated by the algorithm.

Once the border has been solved, the structure is copied from the solution chromosome to all the other chromosomes within the population. The GA then moves on and begins working on the centre part of the puzzle. Another version of the algorithm was developed that solved the border on pairs of chromosomes within the population. The border had to be solved using a pair of chromosomes given that at least two chromosomes are required for selection. Solving the border this way allowed for many possible border arrangements to be present within the first generation of the population as each pair of chromosomes have the possibility of having a unique border. Many of these border arrangements

may possibly be lost during crossover as the weaker chromosome is over written by the fitter solution.

5.6 Mutation

After a new generation of chromosomes have been created, all the newly created child chromosomes have the chance of mutation being applied to them. This chance is calculated by generating a random number between 1 and 100. If the random number generated is less then the mutation rate, mutation will take place on the chromosome.

```
iRandOne = rand();
if((iRandOne%100)+1 < appSettings.mutationRate)   // Apply Mutation
```

If mutation is being applied to the border section of the puzzle and one of the four corner gene locations within the chromosome is selected, the playing piece in this location is randomly rotated either, 90, 180 or 270 degrees. Randomly rotating the piece was the initial idea and this method happened to work well so was not changed. If an edge location is selected, the playing piece is rotated until it has the correct orientation of the edge section of the playing piece matching the edge of the board. This is done to help the GA solve the border. The edge pieces were not automatically rotated to always be matching the border as the application was capable of solving the border when only rotating a piece if it is selected.

When mutation is being applied to the centre section of the puzzle, genes are selected by generating a random number between 0 and the chromosome length-1. The randomly generated number is then checked to see if it represents a border position within the chromosome. If the number does not represent a border location it is then checked to see if it represents the position of any of the clue pieces.

```
iRandThree = rand();
iChromosome = (iRandThree%appSettings.chromosomeLength);
if(isBorderLocation(iChromosome, appSettings.borderSize,
   appSettings.chromosomeLength)==1)
{
  bValidRotate = false;
}

for(int k=0; k<appSettings.clueCount; k++)
{
  if(cluePiecesData[k].iArrayPosition==iChromosome)
  {
    bValidRotate = false;
  }
}
```

If a gene is found that is in the centre of the board and is not a clue piece it is rotated by 90, 180 or 270 degrees, this rotation value is randomly selected.

5.7 Fitness Function

As the puzzle is broken down into the smaller border and centre puzzles, each puzzle requires its own fitness function. Both fitness functions check the fitness of each playing piece (gene) in the same way, but they check different genes within the chromosome.

The fitness function for the border only checks pieces around the edge of the board and in the corners. The playing pieces around the edge are all compared against each other but are not compared against pieces positioned on their inner edge. The border fitness function does not take into consideration any of the centre pieces. Figure 5.2 shows which edges are checked by the border fitness function. Edges that are checked are marked with a dash.

The centre fitness function checks all of the centre pieces against each other including the edges which touch border pieces. As the border has already been

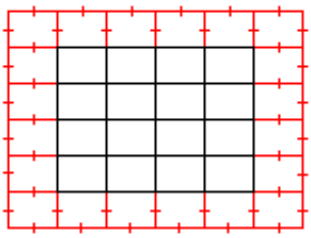

Figure 5.2: Border Fitness Function

solved a centre arrangement that matches all of the border can be deemed a valid solution.

5.8 Clue Pieces

Clue pieces can be provided to the algorithm to help the GA solve the puzzle. They can also be requirements of a solution, such as a starting piece. Any clue pieces are stored in a data file and are loaded by the GA. An example record is shown in table 5.3.

135 139 2

Table 5.3: Clue Structure

This is a record of a single clue piece, the playing piece with reference number 135 has to be positioned in the gene array in index 139 and it has a rotation of 2. This piece cannot be moved by either breeding or mutation.

The GA does not work with clue pieces which are positioned one row in from any edge.

A clue piece provided one row in from the edge could cause the GA to never be able to solve the puzzle. The border to the puzzle is generated first, and when calculating the fitness of the border, none of the centre pieces are taken into consideration, the border pieces are only compared against themselves and the

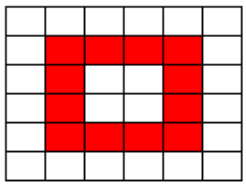

Figure 5.3: 6x6 Border - Invalid Clue Locations

edge of the board. If a clue piece is provided one row in and this piece does not match the generated border the fitness for the board as a whole will never reach 1 as the GA is not allowed to move clue pieces. The GA can only move pieces surrounding clue pieces and once a border is created and the centre of the puzzle is being worked on, the border cannot be changed.

Luckily in Eternity II none of the clue pieces are located on the first row in from any of the border. However, if clue pieces were positioned in such a location, the problem could be overcome by programming the GA to check for these clue pieces when solving the border. Checking an extra row for any clue pieces would only marginally increase the time taken to solve border.

5.9 Summary

This chapter detailed the design stages for the reproduction, selection and mutation methods and the solution presentation. The design of the reproduction method is important in the Eternity II problem, as it is essential that no invalid chromosomes are introduced into the population.

Chapter 6

Experimentation

This chapter gives details of experimentation that was carried out on the GA. Experimentation includes alterations made to the algorithm and any changes to constraint values and the effects of these changes.

6.1 Genetic Algorithm Testing

The GA was first tested on the 36 piece clue version of the puzzle. The 36 piece version was used for testing as it was solvable by a human so should be easy for an application to solve. As there was a known solution for the 36 piece version, this was provided to the algorithm as a complete chromosome to test that all the playing pieces had been recorded correctly in the data file. The GA should have run for 1 generation and recognised the chromosome within the population as being a valid solution and then stop. Initially this was not the case, the GA continued to run outputting a fitness of 0.96 for the chromosome containing the solution, meaning that there was an error in the data file. All the playing pieces were then rechecked against the data stored in the pieces data file and the GA was tested again. The same test was done with the 72 clue piece version as there was a known solution for this puzzle. Once it was known that the data files were

error free it was possible to test the efficiency of the GA at solving the puzzle.

The first version of the GA was set running using the data for the 36 piece puzzle. The constraint values were all initially set to the same value. Figure 6.1 illustrates the output of the algorithm, it can be seen that after a few hundred generations the GA fails to produce any improvement on the best chromosome in the population.

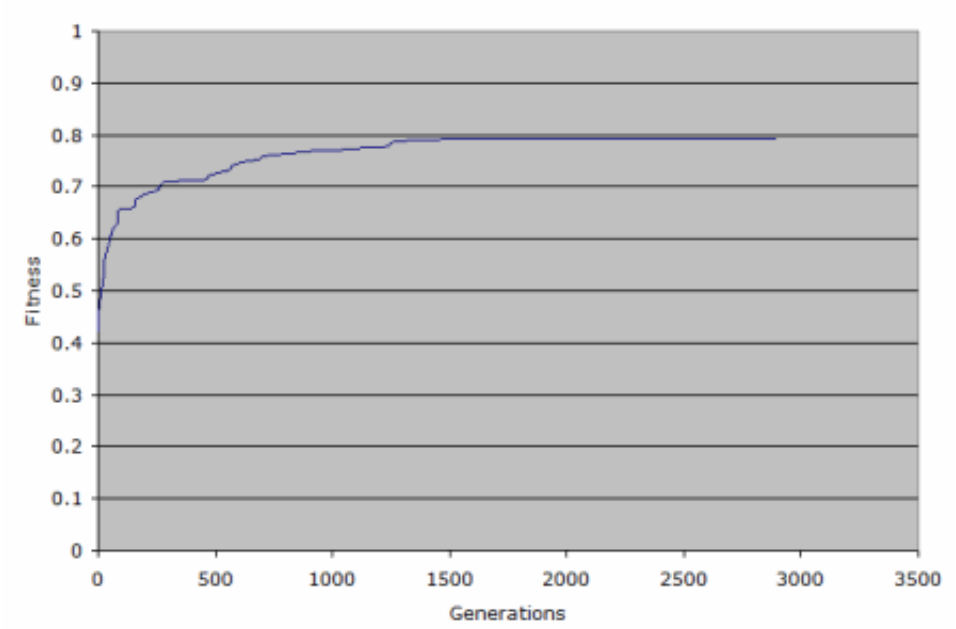

Figure 6.1: First Algorithm Design

Restarting the algorithm produced the same effect, the fitness value would constantly hit a peak somewhere around 80% fitness.

In an attempt to resolve this the constraint values were changed. Initially the constraints were ranked in order of importance, with the corner pieces being in the corner holding the highest value. This was done to encourage the GA to solve the border first. The constraint values that were used can be seen in table 6.1. Ranking the constraints in order of importance produced no change, the GA progressed as before and got stuck at a similar point and there was no obvious pattern to why this was happening.

Corner piece in the corner	15
Corner piece match the board edge	10
Edge piece on the edge	10
Edge piece match the board edge	8
Corner piece match background colour with an adjacent piece	5
Corner piece match shape with an adjacent piece	5
Corner piece match shape colour with an adjacent piece	5
Edge piece match background colour with an adjacent piece	3
Edge piece match shape with an adjacent piece	3
Edge piece match shape colour with an adjacent piece	3
Centre piece match background colour with an adjacent piece	1
Centre piece match shape with an adjacent piece	1
Centre piece match shape colour with an adjacent piece	1

Table 6.1: Constraint Values

After several test runs which produced the same output, it was decided that simply ordering the constraints by importance was not going to be enough to help the GA progress. A second idea was to randomly change the constraint values when the GA made no progress for a fixed number of generations, this number was initially 1000 generations. Whenever this value was reached, the constraint values would all be randomly changed. Randomly changing the constraint values caused the fitness to peak near the same peak fitness value reached by using ranked constraints, this can be see in figure 6.2. Neither ranking the constraints or randomly changing them helped the GA progress further.

More information was required about those chromosomes that were getting stuck between 70% and 80% fitness. The information was needed to see if there were any obvious patterns that were appearing, such as edge pieces not being placed in the border positions within the chromosome, or if the shapes on the centre pieces were not matching up with adjacent pieces for example. The GA was slightly modified to output extra information each time the GA became stuck. Such information included the fitness of the chromosome, the number of corner pieces that were positioned in a corner, the number of edge pieces

positioned along the edges, the number of centre pieces in the centre and the number of background colours, shape colours and shapes that match on adjacent pieces. The GA was left to run, producing over 450 outputs (see appendix B).

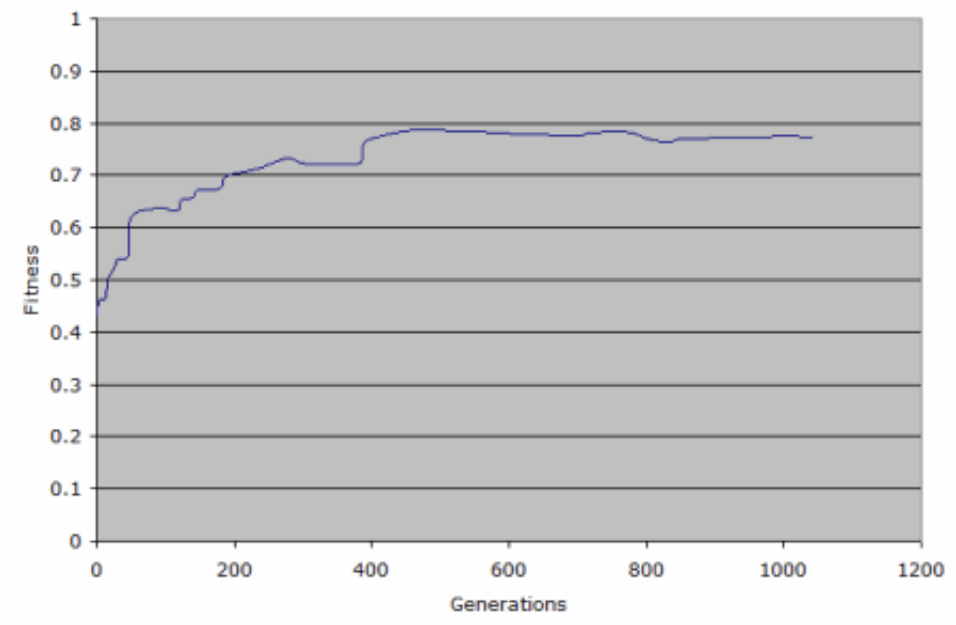

Figure 6.2: Random Constraint Values

Fitness (1)	Corners (4)	Edges (16)	Centre (16)	Bg. Colour (120)	Shape Colour (120)	Shape (120)
0.76098	4	16	16	79	100	78
0.77027	3	16	16	90	94	88
0.782095	3	15	15	86	100	86
0.744932	3	14	14	82	90	82
0.766047	3	15	15	81	103	81
0.760135	2	16	16	88	102	88
0.765203	3	13	12	82	96	82
0.714527	3	14	13	80	100	80
0.772804	2	13	12	86	101	86
0.759291	3	14	14	90	97	88
0.774493	3	15	15	79	99	79
0.76098	2	14	13	81	97	81

Table 6.2: Chromosome Details

A selection of the outputs can be see in table 6.2, these figures represent the number of matching edges. There were no obvious patterns in the data. Some chromosomes can be seen to have all four corner pieces and the edge pieces in the right position but then few matching centre pieces. However, there are other chromosomes that do not have all the border pieces in the right positions. The output data did not provide any obvious pattern in the chromosomes that could be fixed. Therefore, it was decided that when the GA becomes stuck, mutation at a much higher rate would be applied to each chromosome. A higher rate of mutation was applied to hopefully either bump the fitness along to a higher level or drop the fitness down slightly as to allow the GA to try again. The 'desperation mutation' as it was named, caused the highest fitness to drop lower, the GA simply then climbed back up and got stuck again.

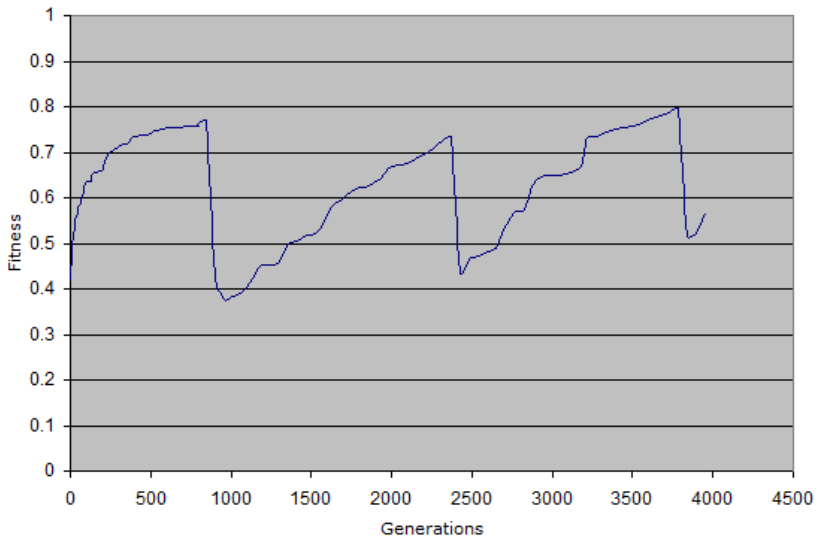

Figure 6.3: Desperation Mutation

Figure 6.3 shows the output from three desperation mutation attempts. The fitness never climbed higher than previously; this process was the equivalent of restarting the algorithm each time.

As there was a known solution to the puzzle, it was decided to provide the

algorithm with the completed border. By providing the completed border it was hoped that this would increase the highest fitness reached by the algorithm. The completed border was quickly destroyed by the algorithm. The border was being destroyed because the reproduction method was taking a random length of genes from the whole chromosome and swapping them to another position. The random length of genes could be anything from one gene to half of the chromosome. Anything of length six or over would mean at least two of the border pieces were being moved. Figure 6.4 illustrates in red the border locations within the 36 piece puzzle chromosome. A completed border is unlikely to survive one generation without being destroyed.

Figure 6.4: Border Locations - 36 piece

Restricting the length of genes that could be selected would reduce the chances of border pieces being moved. Selection was limited to moving one gene at a time. Limiting selection to one gene at a time produced the same results as moving a random length of genes and simply took longer to reach a similar maximum point. When providing the algorithm with a completed border, the border would still be broken it would just take slightly longer. There was nothing to stop border pieces being swapped with centre pieces.

As the problem was split into two smaller problems it was possible to work out which positions within the chromosome represented border locations on the board. Using crossover and mutation on border locations only, the border was solved on the 36 piece version, swapping one gene around at a time and with the normal low mutation rate. The border for the 36 piece version could be solved, although the same algorithm could not solve the border on the 256 piece version. Examples of the border for the 256 piece version can be seen in appendix C.

The GA may have eventually found a solution to the 256 piece border as it is capable of solving the 36 piece version. Extra help was given to the GA to speed

up this process. The edge pieces were rotated so that they are aligned with the border correctly, as this is a requirement of a solution.

Whenever mutation or crossover took place on a playing piece it was initially randomly rotated by either 90, 180 or 270 degrees. The random rotation was later changed so that the playing piece would rotate until it was correctly aligned with the border. Rotating all the pieces that were selected for crossover and mutation enabled the GA to create a border for all three versions of the puzzle.

Once the border is solved in any of the chromosomes, the structure is copied over to all the other chromosomes within the population. Copying the border to all the other chromosomes could restrict the number of solutions available. However, after running the algorithm more than twenty times on the 36 piece version with the GA always being capable of solving the centre part of the puzzle, it was decided that for every border solution there was a centre solution.

The idea that there was always a solution for the centre if the border had been solved was later disregarded. Limiting the whole population to one border solution may greatly reduce the chance of a solution being found. The algorithm was therefore modified to solve the border multiple times in each population before trying to solve the centre of the puzzle. Little testing has been performed on the efficiency of this change as it was made towards the end of the project, the testing that was done has proven that the GA is still capable of solving the 36 piece version.

The algorithm was capable of solving the 36 piece version and solutions are provided (see appendix D). The 72 piece version has reached 0.964583 fitness after running for two days; for these solutions see appendix E. The algorithm was also left to run for two days on 20 machines trying to solve the 256 piece version. The output from this run can be seen in figure 6.5. All twenty machines progressed at a similar rate, and it may be just a matter of time for the algorithm to solve the puzzle.

The text outputs from these runs have been included (see appendix F). The

GA is currently running on the University Unix machine (as of Thursday 17^{th} April 2008) and is set to run for as long as possible, this will hopefully provide a better solution to the 256 piece puzzle.

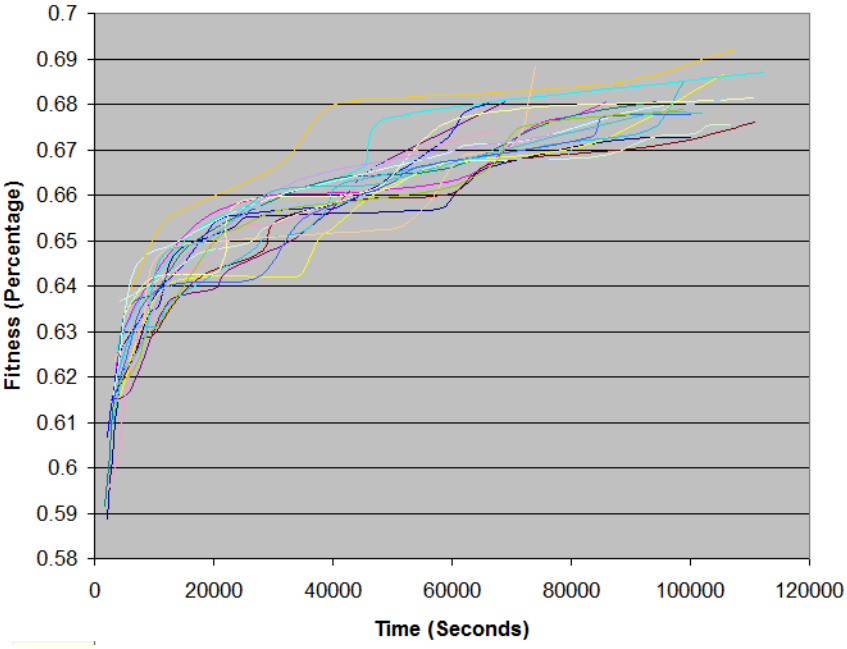

Figure 6.5: 2 day run - 256 piece

6.2 Summary

This chapter covered the experimentation carried out on the GA during the development process. The algorithm was modified to be able to solve the 36 piece puzzle, has come close to solving the 72 piece puzzle and shows signs of solving the 256 piece puzzle. To solve the 256 piece version it may just be a matter of time or there could be a fundamental flaw in the algorithm that has not been noticed yet.

Chapter 7

Evaluation and Recommendations

This chapter details the efficiency of the GA and the overall performance at solving the puzzle. Suggestions for further work for the GA are also discussed.

7.1 Genetic Algorithm

The Genetic Algorithm was capable of solving the 36 piece puzzle. Experimentation also suggests that the GA is capable of solving the 72 piece version with results currently at a fitness of approx 97%. If the algorithm can be left to run for a longer period of time, the 72 piece version could probably be solved. If the 72 piece version can be solved by the algorithm, it should theoretically also be possible to find a solution to the 256 version.

The initial few versions of the algorithm made weak attempts at solving the puzzle. The problems that these versions encountered, which for example included, destroying borders and failing to keep correct borders were overcome, producing a GA capable of solving the smallest puzzle.

The GA requires that each chromosome within the population is sorted so

that the border pieces are in border locations before the algorithm can begin, this could be avoided by breaking the puzzle into two separate populations. The process of sorting the genes within the chromosome is the equivalent of sorting the edge pieces when doing a jigsaw.

7.2 Further Work

A possibility for further work would be implementing the graphical front end so the user has a visual output of the progress the GA is making. Currently the GA has a limited GUI, this is for performance as the application does not have to render anything. The console and text output is a minimum requirement but if the application were developed into a screensaver, it could be set to only run when the computer is idle and display a visual output of the highest fitness solution.

If the GA is stopped it cannot currently be resumed from the point is has reached. The only output the user will have is the current highest chromosome, the rest of the population is unknown. A population dump option would be useful, where the user at a key press will get a dump file of the whole population to the disk. The application would be modified to take an optional extra command line parameter of a population dump, and instead of generating a new population would read the previous dump file. The option to resume the GA from a previous point would be useful because the GA has to run for days at a time and the machine it is running on may need to be turned off. Any progress that the GA has made is currently lost as the application will have to start again with a new randomly created population.

Another alteration to the algorithm has been considered where the puzzle is solved one row at a time, working in from the edge. The border is solved first, then the next row in, this is repeated until a row cannot be solved where the algorithm will then backtrack to the previous row and try again. Breaking the puzzle down into even smaller puzzles (rows) may bring to light problems not

yet noticed.

7.3 Personal Reflection

The research was an ongoing process throughout the whole of the project. Research was continuous from the first week until the last and from this a much greater understanding of the problem has been gained. By reflecting and building on original ideas, new and improved ideas came to light. When something did not work as expected it usually produced a result that initiated a new idea. The process of trying an idea and noticing something new was repeated through-out the whole project and ultimately produced the GA capable of solving the 36 piece version.

The fitness function was initially designed during the three week period timetabled, but as the GA was constantly being changed, the fitness function also needed modifying to fit the new algorithm.

The GUI was never implemented for the applications, as this would have slowed down the algorithm. Simple information was output to the console and important information was written to the disk. Initially this seemed to be a good idea as it would provide a method of watching the algorithm progress, but when it was realised that the GA would need to run for long periods of time (days at a time for the larger puzzles), a GUI could not feasibly be watched by the user.

The implementation and testing were merged into one large process. It was impossible to design a successful GA from the very beginning, with much of the process being a case of trial and error. The GA was implemented and then this version was tested and modifications made. This process was repeated multiple times throughout the project.

Overall the project process was fairly successful. An application was written that could solve the smallest version of the puzzle and has come very close to solving the 72 piece version. Over the next few weeks the progress of the GA

trying to solve the 256 piece version will be watched, hopefully with positive results, if not there is another unnoticed problem, which is most likely.

7.4 Summary

This chapter detailed the efficiency of the GA and covered suggestions for further work on the GA. The project process was also evaluated.

Appendix A

Terms of Reference

Project Title

Using a Genetic Algorithm to solve Eternity II

Background to project

Eternity II is the second board puzzle in the series released by Christopher Monckton. The first version of Eternity was released in June of 1999. It consisted of 209 irregular shaped polygons that had to be placed into a large dodecagon. Two mathematicians from Cambridge were the first to solve the problem just over a year after it was released (Wainwright 2001). Eternity II is a 256 square piece board puzzle. Each playing piece is divided diagonally into 4 right angled isosceles triangles. Each triangle has a coloured background of which there are 8 possible colours and half a coloured shape touching the long edge of the triangle of which there are 11 different shapes. The pieces have to be placed onto the game board so each adjacent piece matches in colour and shape. The game board has a grey edge and all edge and corner pieces have to match up respectively. There is a single starting piece, piece number 139, which has to be placed in square I8, this is specified in the rules of the game. A solution can only be valid with this starting piece in place. Eternity II could be considered a signed edge-

matching puzzle (Demaine & Demaine 2007) given that adjacent pieces have to match both colour and pattern.

The prize for the first correct solution to the puzzle is US$2,000,000. All solutions are to be posted to the publishers and will be kept unopened until December 31st 2008. After this date they will be opened in the order that they were received. The first correct solution will be the winner. If no solutions are found, the deadline will be extended until 2009 and finally 2010. When designing the fitness function for the genetic algorithm it will be taken into consideration that there are no obvious soft constraints. For this type of puzzle all the constraints are hard constraints and must be met for the solution to be feasible.

Not taking into consideration the starting piece or the fact that there are edge pieces, there are 256!*4256 possible combinations as each piece can be rotated four times. Including the starting piece and the edge pieces there are 1*4!*56!*195!*4195 positions. Given the possible number of piece combinations a brute force method for solving this puzzle would be very slow.

Aims of project

Investigate into methods of solving the Eternity II puzzle. Design and implement an application to attempt to solve the Eternity II puzzle.

Objectives

The following objectives will make up parts of the final project report and application.

- Research on Genetic Algorithms.
- Design the fitness function.
- Application and GUI design.

- Implementation and GUI integration.
- Application testing.
- Product evaluation.
- Write dissertation.

A description of all the expected product items

The project application will be designed to run in Windows. The Genetic Algorithm will be coded in C++ and the GUI in Java.

Relationship to the course

The project will require the use of material covered in the following modules:

- Programming for Games 1
- Programming for Games 2
- Programming for Games 3
- Programming for Games 4
- Advanced Programming Issues for Games
- Object Oriented Design
- AI for Computer Games

Sources of information / bibliography

Throughout the project several books will be required as sources of information and reference,

Dale N, Weems C & Headington M [2002]

``Programming and Problem Solving with C++``
Jones and Bartlett Publishers, Inc.

Flanagan D [1996]
``Java in a Nutshell, 5th Edition``
OReilly.

Mitchell M [1998]
``An Introduction to Genetic Algorithms``
A Bradford Book, The MIT Press.

All other books used during the project will be references in the report.

Resources - Hardware and Software Required

The project will require the use of a Windows based PC as programming will be done in Microsoft Visual Studio 2008 for Windows Vista. Visual Studio 2008 has been chosen as this is the most recent version compatible with Windows Vista. Windows Vista is available in the home environment. In the University environment Visual Studio 2005 will be used as this is the only installed version compatible with Visual Studio 2008.

The Java GUI will be coded in Visual Studio 2005 at University and JCreator in the home environment. Both applications support syntax highlighting. The GUI will be compiled through Visual Studio or JCreator using the Java SDK, which is installed at both locations.

Java will be used to implement the GUI as Visual Studio 2005 offers a layout manager, allowing easier positioning of elements and enabling simple modifications to controls.

C++ will be used for the code as it is a compiled language enabling quick execution of the algorithm.

The student is familiar with the interactive development environment that Visual Studio has to offer, having previously used it for personal and university based projects.

Structure of Report

- Introduction
- Report Chapters
  - Packing Problems - Objective 1
  - Genetic Algorithms - Objective 1
  - Design - Objectives 2, 3
  - Implementing the application - Objective 4
  - Testing - Objective 5
  - Evaluation and further reading - Objectives 6, 7
- Appendices
  - Hardware requirements
  - Terms of Reference
  - Code

Project Plan

For a successful project, time must be managed carefully. The Gantt chart (see figure A.1) shows time scheduled for each chapter of the report with times for any alterations to be made.

Marking Scheme

The following chapters of the report make up the analysis:

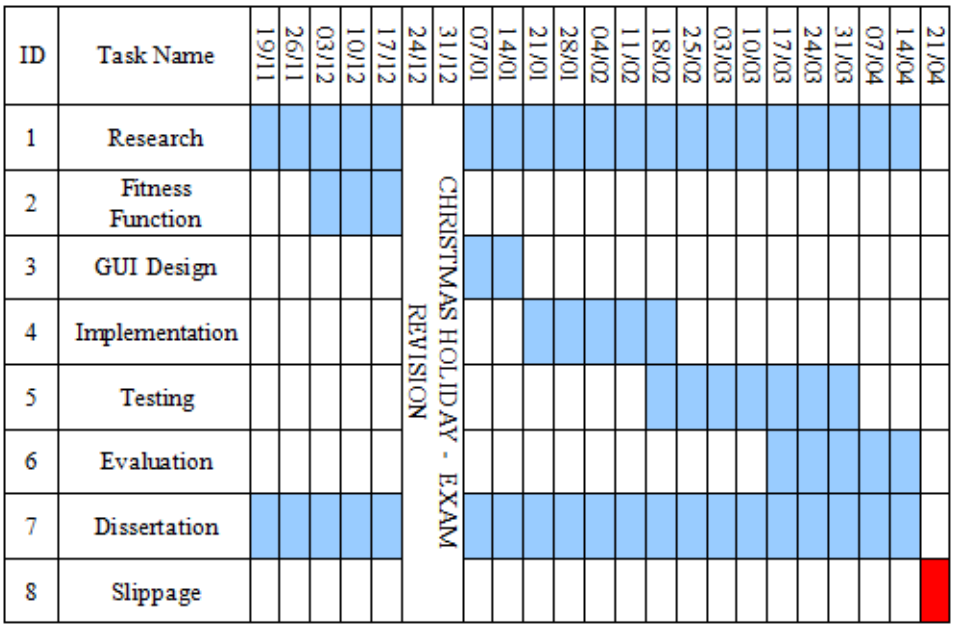

Figure A.1: Project Timetable

- Genetic Algorithms

Synthesis:

- Design
- Implementing the application

Evaluation:

- Testing
- Evaluation and further reading

Part A - Project Report	
Abstract Introduction	0 - 5 marks
Analysis	0 - 25 marks
Synthesis	0 - 25 marks
Evaluation	0 - 25 marks
Conclusions Recommendations	0 - 10 marks
Presentation	0 - 10 marks

Table A.1: Marking Scheme (Report)

Part B - Product	
Fitness for Purpose	50%
Build Quality	50%

Table A.2: Marking Scheme (Product)

Appendix B

Chromosome Information

The information about each chromosome, this includes the fitness of the chromosome, the number of corner pieces that were positioned in the corner, the number of edge pieces positioned along the edges, the number of centre pieces in the centre and the number of background colours, shape colours and shapes that match on adjacent pieces - see CD

Appendix C

256 - Border examples

Example borders produced by the algorithm when attempting to solve the 256 piece border - see CD

Appendix D

36 - Solutions

The solutions provided by the algorithm for the 36 piece version, this includes the text output files and the graphical representations - see CD

Appendix E

72 - Solutions

The partial solutions provided by the algorithm for the 72 piece version, this includes the text output files and the graphical representations - see CD

Appendix F

256 - Run outputs

The text outputs by the GA from the 20 machines trying to solve the 256 piece version - see CD

List of Abbreviations

BFS	Breadth-first search	page 1
GA	Genetic Algorithm	page 4
GUI	Graphical user interface	page 4
HS	Harmony Search	page 14
NN	Neural Network	page 11
RMS	Root mean squared	page 12
TPP	Travelling purchaser problem	page 14
TSP	Travelling salesman problem	page 9
SA	Simulated annealing	page 14
UML	Unified Modelling Language	page 5

List of Figures

List of Tables

References

Adebola A 1996

``*New methods in genetic search with real-valued chromosomes*´´

Massachusetts Institute of Technology. Dept. of Mechanical Engineering.

Bontoux B & Feillet D 2008

``*Ant colony optimization for the traveling purchaser problem*´´

Computers and Operations Research Vol. **35**, No. 2.

Champion M 1998

``*Re: How many atoms make up the universe?*´´

http://www.madsci.org/posts/archives/oct98/905633072.As.r.html.

Dale N, Weems C & Headington M 2002

``*Programming and Problem Solving with C++*´´

Jones and Bartlett Publishers, Inc.

Darwin C 1859

``*On the Origin of Species*´´

John Murray Publishers Ltd.

Dayhoff J 1990

``*Neural Network Architectures: An Introduction*´´

Van Nostrand Reinhold Company.

de Souza M C, de Carvalho C R & Brizon W B 2008

``*Packing items to feed assembly lines*´´

European Journal of Operational Research Vol. **1**84, No. 2.

Demaine E D & Demaine M L 2007

``*Jigsaw Puzzles, Edge Matching, and Polyomino Packing: Connections and Complexity*´´

Graphs and Combinatorics Vol. **2**3, No. 2.

Dorigo M & Blumb C 2005

``*Ant colony optimization theory: A survey*´´

Theoretical Computer Science Vol. **3**44, No. 2-3.

Fister I, Mernik M & Filipic B 2007

``*Optimization of markers in clothing industry*´´ *- Article in Press*

Engineering Applications of Artificial Intelligence.

Flanagan D 1996

``*Java in a Nutshell, 5th Edition*´´

O'Reilly.

Fowler M & Scott K 1999

``*UML Distilled - Second Edition - A Brief Guide to the Standard Object Modeling Language*´´

Addison Wesley.

Geem Z W, Kim J H & Loganathan G 2001

``*A New Heuristic Optimization Algorithm: Harmony Search*´´

Simulation Vol. **7**6, No. 2.

Geem Z W, Lee K S & Park Y 2005

``*Application of Harmony Search to Vehicle Routing*´´

American Journal of Applied Sciences Vol. **2**, No. 12.

Gutin G & Punnen A 2002

``*The Traveling Salesman Problems and Its Variations*´´

Kluwer Acdemic Publishers.

Kirkpatrick S, Gelatt C D & Vecchi M P 1983

``*Optimization by Simulated Annealing*´´

Science Vol. **2**20, No. 4598.

Kuan C M & Liu T 1995

``*Forecasting exchange rates using feedforward and recurrent neural networks*´´

Journal of Applied Econometrics Vol. **1**0, No. 4.

Martello S & Toth P 1990

``*Knapsack Problems - Algorithms and Computer Implementations*´´

John Wiley & Sones Ltd.

Mitchell G 2005

``*Validity Constraints and the TSP GeneRepair of Genetic Algorithms*´´

Artificial Intelligence and Applications Vol. pp. 306–311.

Mitchell M 1998

``*An Introduction to Genetic Algorithms*´´

A Bradford Book, The MIT Press.

Mitchell M, Forrest S & Holland J 1992

``*The Royal Road for Genetic Algorithms: Fitness Landscapes and GA Performance*´´

Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life, 1991

A Bradford Book, The MIT Press.

Owen B 2007

``*Eternity II*´´

http://eternityii.mrowen.net/solving.html.

Tomy 2007*a*

``*Eternity II*''

http://uk.eternityii.com/.

Tomy 2007*b*

``*Eternity Puzzle is Back This Summer With a US$2 Million Prize for the First Person to Find a Solution*''

http://www.prnewswire.co.uk/cgi/news/release?id=188486.

Wainwright 2001

``*Prize specimens*''

Plus Magazine, Millennium Mathematics Project - Issue 13

http://plus.maths.org/issue13/features/eternity/index.html.

White G M & Wong S K S 1988

``*Interactive timetabling in universities*''

Computers & Education Vol. **1**2, No. 4.