**EL3300 Machine Intelligence Genetic Algorithm Assignment**

Alvydas Juodikis G20730626 EL3300 Carl Berry

**Table of Contents**

**Introduction** 1

**Crossover** 1

**Mutation** 2-3

**Testing and Selecting Optimal Values**  3

**Conclusion** 4

**Reference List** 5

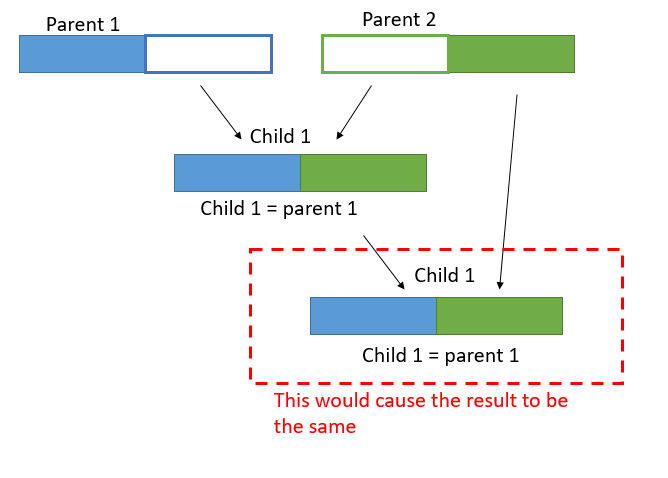
**Appendix** 6-8

Alvydas Juodikis G20730626 EL3300 Carl Berry

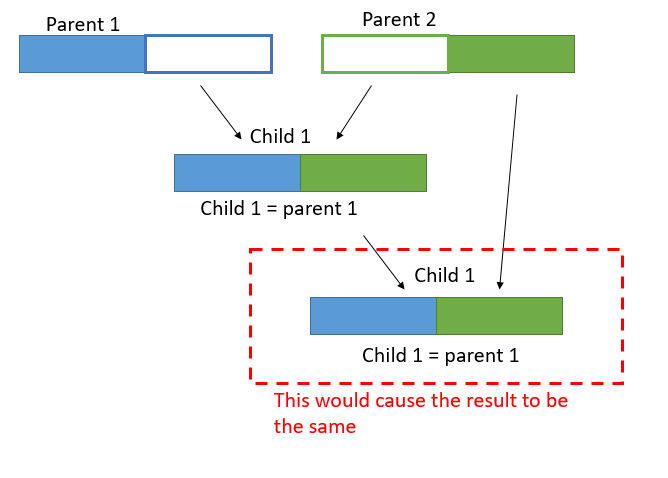
**1) Introduction**

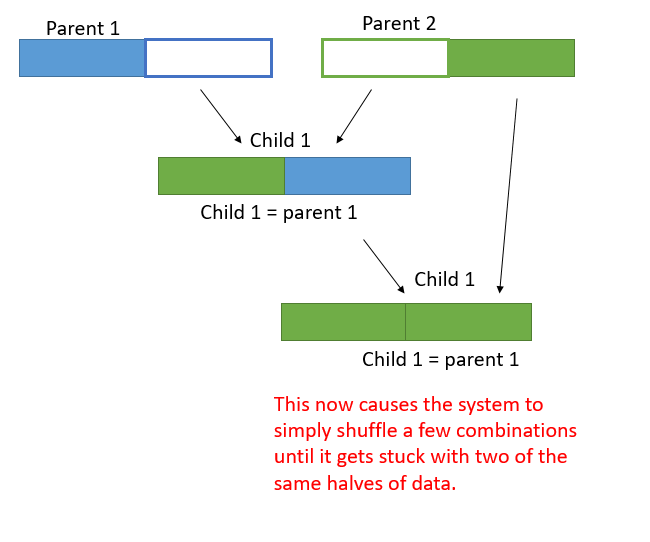
To implement the genetic algorithm two genomes of 53 genes (integer arrays of 53 elements), parent1 and parent2 were created. The main genetic algorithm code was implemented within a button and when pressed performed the crossover and mutation to find the optimum values. The desired design was to have a crossover performed on two 53 element arrays and to have a series of mutations occur at a random time. The idea behind the mutation was to have different levels of mutation, for instance a soft mutation where values in each parent array would be shifted after each crossover occurred at all times to ensure genetic variance; to implement the idea of a rare phenomenon a second child would be used to perform a crossover with the first child, exposing a different set of results adopted from each parent. This idea was adopted from nature where mutations may occur randomly at any time, however there would always be slight genetic variation with every child (Anon, 1997).

**2) Crossover**

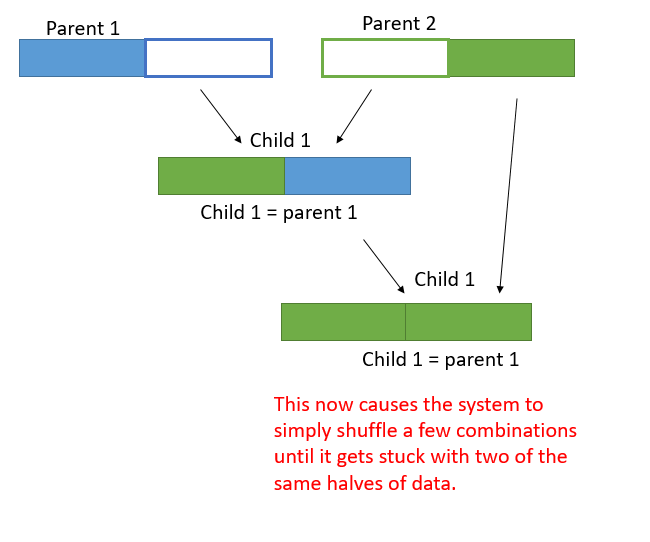
The first step was implementing the crossover which attributes from parent 1 and attributes of parent 2 are adopted by a child (Feng, 2018). This was achieved by creating two additional arrays c1 and c2 of lengths 26 and 27 and having them adopt the first half of parent 1 and second half of parent 2. The child then copies elements from both parents. This presented a large issue of that being the same result would be returned as the elements being copied to the child and then crossover performed for the next generation were the same (See Appendix A).

1) Initial Crossover



Ultimately a solution presented was to copy the 1st half of parent 1 and place that as the 2nd half of child 1 and to copy the 2nd half of parent 2 and place that as the 1st half of child 1. This only solved half the issue, as now new values were being formed, however this method neglected half of the other data from parents 1 and parents 2, also leading to the system just rearranging the same values. This without mutation, meant that the method only shuffles the same set of fitness numbers which is ineffective and does not help find the optimal solution; what is produced is a system that keeps returning a range of the same fitness numbers. The crossover would be improved within the mutation section of the code (See Appendix A).

2) Shuffling Crossover

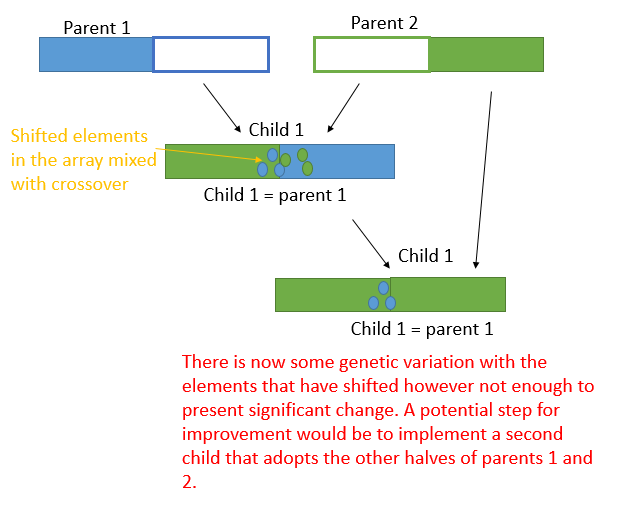


Alvydas Juodikis G20730626 EL3300 Carl Berry Page 1

**3) Mutation**

Before implementing mutation a for-loop was used which assessed the condition of the population; looping around a series of iterations based on the population size. Within another for loop that looped based on the epoch value for iterations contained a method that shifted the values in both parent1 and parent2 array by 1 place each time the system ran. This allowed for the system to keep altering values while performing the crossover to adopt new values. The system was still stuck within the issue of only dealing with the 1st half of parent 1 and 2nd half of parent 2 however this was slightly mitigated with the fact that the elements in each array are shifted 1 place each time the system is ran. This mutation would naturally occur every time the system runs, in order to mimic the idea of soft mutation in nature (See Appendix A).

3) Shifting Crossover

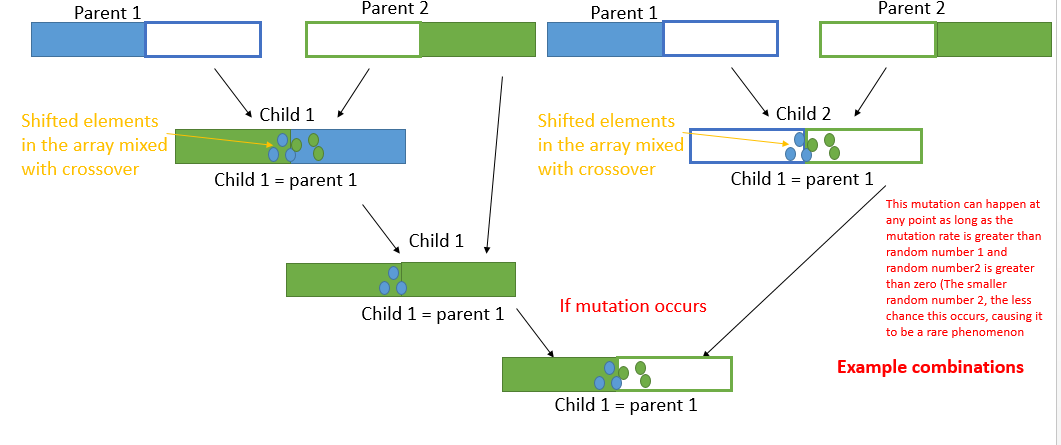


Two random numbers were created, one which provided the main condition for the system entering a mutation; if the mutation rate variable was greater than the first random number, then the possibility of any mutation happening at all was dependent of meeting this condition. If the condition was met then further if conditions were available for access. This idea was to demonstrate that a high mutation rate was not the determining factor that a mutation would occur, it only simply increased the chances and provided access to further levels of mutation.

Alvydas Juodikis G20730626 EL3300 Carl Berry Page 2

In terms of a further level of mutation, a second child was created and served the purpose of copying elements from parents 1 and parents 2 in different order (See Appendix A). Additional levels of mutation such as swapping elements that have shifted in an array where performed to further the mutation process. This meant that if the further mutation occurred child 1 would perform a crossover with child 2 and adopt unique genes. This can be regarded as a rare phenomenon due to the fact that the likeliness of this taking place involves random number 1 being less than the mutation rate, and random number 2 being greater than some such value, which was denoted as 75 to prevent this occurring too often (See Appendix A).

4) Second Child Adopts other half of attributes from each parent



**Testing and Selecting Optimum Values:**

A testing population of 60, an epoch value of 1 to monitor each iteration, a crossover probability of 0.85 and mutation rate of 80% were used. All values selected were based on trial and error testing when seeking the highest returned fitness value. The population was used when determining how many crossovers would take place each time the button was pressed and seemed to perform most effective between 50 and 70. The crossover probability needed to be high enough to ensure a crossover occurs almost every time, for this the probability of 0.85 was sufficient enough to ensure this. Similarly the mutation rate was chosen to a high value of 80% to ensure some genetic variance occurs within the system at every run; a high mutation rate did not mean that all mutations may occur, just so that the soft mutation of shifting elements would do so at the very least.

Alvydas Juodikis G20730626 EL3300 Carl Berry Page 3

**Conclusion:** Obtaining the Fitness function and interpretation of Values

Once all crossovers and mutations had taken place, the child array would be converted to a string and displayed in the tbDisplay textbox. From this a string was created in an additional button that then obtained the string of genes in the text box and ran it through the fitness function. The fitness function then returned a value in the text box (See Appendix B).

Using a testing population of 60, an epoch value of 1 to monitor each iteration and mutation rate of 80%; the best fitness value returned was 230 which corresponded to 00100 01000 00001 01000 10000 00100 01000 00100 01000 10101 100. Based on the returned string data, the following system configurations are required:

●External Temperature Medium ●Internal Temperature Low ●Cylinder Pressure Very High

●Value Opening Pressure Low●Load Torque Very Low●NOx Emissions Medium

●CO Emissions Low●HC Emissions Medium●PM Emissions Low

●UCLan developed Electronic Timing Device ON ●UCLan developed Fuel Flow Device. OFF

● UCLan developed Emissions Limiter ON ● UCLan developed Battery Management System OFF

● UCLan developed Air Flow Management Device ON

● Fuel Injection Timings Low

Alvydas Juodikis G20730626 EL3300 Carl Berry Page 4

**Reference List:**

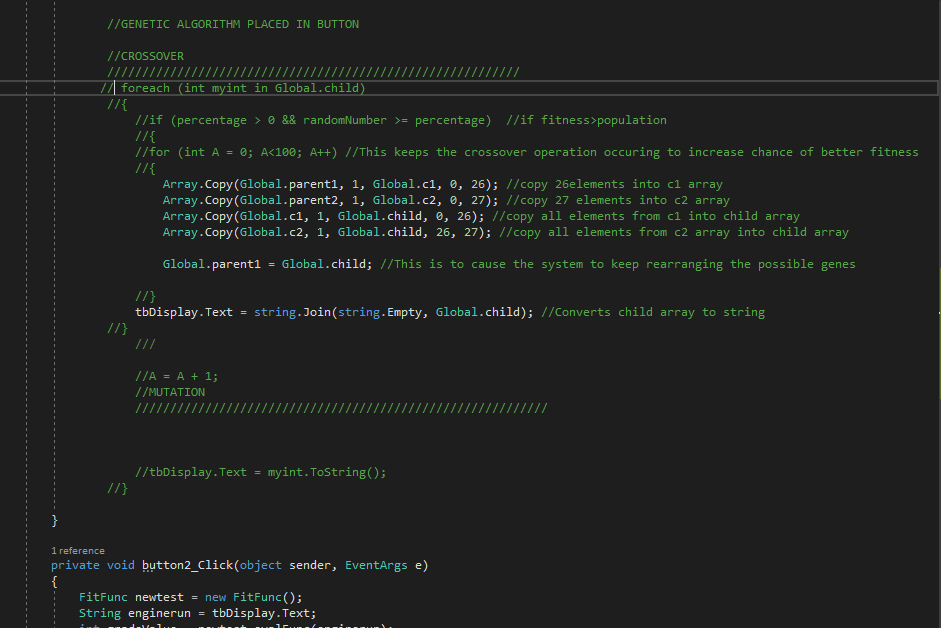
Anon (1997) 2nd International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA ’97). Place of publication not identified: Institution of Engineering & Technology.

Feng, Y. et al. (2018) A three-layer chromosome genetic algorithm for multi-cell scheduling with flexible routes and machine sharing. International Journal of Production Economics. [Online] 196269–283.

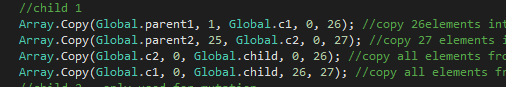
Alvydas Juodikis G20730626 EL3300 Carl Berry Page 5

**Appendix A: Development through Testing**

1) Initial Crossover

****

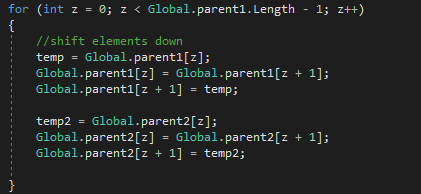
This essentially led to the same set of fitness numbers being shuffled as this method only kept moving the same set of genes around and around again.



Caused system to simply shuffle the same combinations until it got stuck with two halves of the same string.

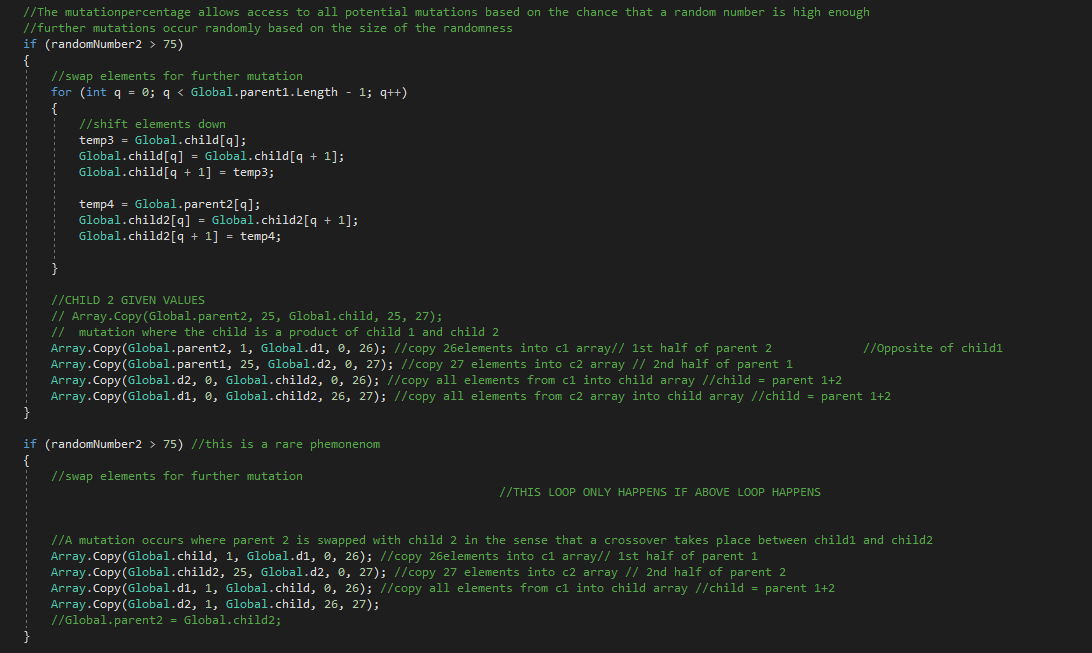
2) Shuffling Crossover

3) Shifting Crossover



This was used to shift elements of each parent down every time a crossover was performed. The same issue still was taking place with slight genetic variation and the other halves of parents were being neglected.

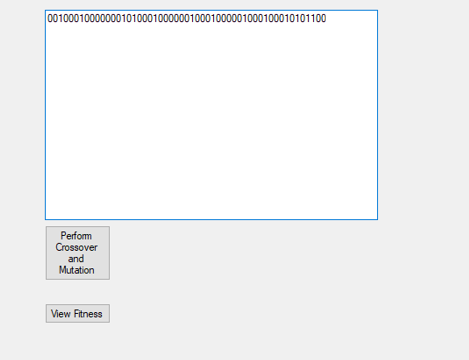
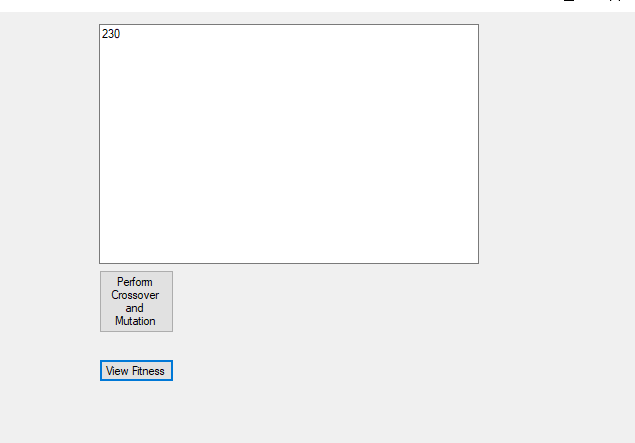
Alvydas Juodikis G20730626 EL3300 Carl Berry Page 6



4) Second Child Adopts other half of attributes from each parent

This mutation can happen at any point as long as the mutation rate is greater than random number 1 and random number2 is greater than a proposed value (The smaller random number 2, the less chance this occurs, causing it to be a rare phenomenon). For the case of testing 75 was used as a standard value that would prevent random mutation occurring too often however not too rarely.

Alvydas Juodikis G20730626 EL3300 Carl Berry Page 7

**Appendix B: Result and Fitness Value**

**String Returned:**

00100 01000 00001 01000 10000 00100 01000 00100 01000 10101 100

**Fitness Value:** 230

**Required System Configurations**

●External Temperature Medium

●Internal Temperature Low

●Cylinder Pressure Very High

●Value Opening Pressure Low

●Load Torque Very Low

●NOx Emissions Medium

●CO Emissions Low

●HC Emissions Medium

●PM Emissions Low

●UCLan developed Electronic Timing Device ON

●UCLan developed Fuel Flow Device. OFF

● UCLan developed Emissions Limiter ON

● UCLan developed Battery Management System OFF

● UCLan developed Air Flow Management Device ON

● Fuel Injection Timings Low

(00100010000000101000100000010001000001000100010101100 = 230)

Alvydas Juodikis G20730626 EL3300 Carl Berry Page 8