Using a Genetic Algorithm to Solve Crossword Puzzles

Kyle Williams

April 8, 2009

Abstract

In this report, I demonstrate an approach to solving crossword puzzles by using a genetic algorithm. Various values for the population size and the genetic operators have been considered, and the performance of each of these variables has been measured. It was found that the values of the various variables considered, have a significant effect on the performance of the genetic algorithm. It is shown that solving a crossword puzzle using a genetic algorithm is possible, and can easily be done. However, it is noted that the genetic algorithm quickly converges to a sub-optimal solution, and then over time, through mutation, eventually finds an optimal solution.

0.1 Introduction

In this project, I set out to make use of a genetic algorithm for solving crossword puzzles. The problem being investigated is: given a solution to a crossword puzzle, can a genetic algorithm, through random generation of letters, find the correct solution? The problem is hard because the genetic algorithm needs to be able, a) to select the correct letters, and b) to arrange them in the correct order.

The goal of the project is to demonstrate the ability of a genetic algorithm in solving a crossword puzzle, as well as its efficiency in doing so. A crossword puzzle is used to demonstrate the larger concept, of the ability of a genetic algorithm to come up with a measurably correct solution to a problem. In this sense, if a genetic algorithm is able to come up with a solution for a problem for which the solution is known, and the fitness of a solution can be measured, then, given a similar problem for which the solution is not known, if the fitness of a solution can be measured, a genetic algorithm should be able to come up either with, a) the correct solution, or b) a sub-optimal solution which is close to the correct solution.

Instead of making use of the genetic algorithm tools and libraries which are available, I instead opted to code all operations and functions of the genetic algorithm myself. The reason for doing this was that it allowed me to gain a better understanding as to exactly how a genetic algorithm is implemented, and more importantly, what aspects of a genetic algorithm need to be considered before and during implementation.

To demonstrate the progress of the genetic algorithm, a graphical user interface (GUI) is made use of. The GUI shows the convergence of the genetic algorithm towards a solution by highlighting correctly placed letters in green. The GUI is updated for every new generation, and allows the user to see changes in subsequent generations taking place in real time. Once the correct solution has been found, the output is a completely green coloured crossword puzzle.

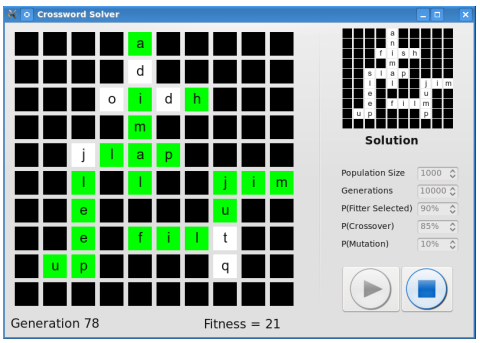

Figure 0.1: Crossword Solver GUI

0.2 Crossword Puzzles

A crossword puzzle is a word game in which the goal is to create words or phrases by filling white squares with letters. Crossword puzzles are created to have one single solution, and it is the task of the user to make use of a set of clues to find that solution. Words and phrases are represented in the puzzle either from left to right, or from top to bottom. There are different way of representing a crossword puzzle, the main and most common ones being: American style, British style, Japanese style and German style[1] - the details of these representations of crossword puzzles are not relevant to this discussion and therefore will not be discussed here.

The crossword puzzle used in this project differs slightly from a traditional crossword puzzle. The genetic algorithm does not make use of clues to come up with words, but rather uses random letters. Therefore, clues are omitted from the crossword puzzle used in this project.

0.3 The Genetic Algorithm

The classical implementation of a genetic algorithm is used to demonstrate the ability of a genetic algorithm to solve a crossword puzzle.

0.3.1 Encoding

Solutions are encoded as a binary string and each element in the solution is initialised with an equal probability of being either 0 or 1. For a crossword puzzle for which there are n letters in the solution, a binary string of size $5n$ is generated. That is, 5 bits are used to represent each letter.

In order to determine the character value of the 5 bits which make up a letter, a decoding process takes place. First the 5 bits are converted to a decimal number i. Then a fixed value of 97 is added to i to form v, and the Unicode value of v is found.

Algorithm 0.1 Decoding a 5 bit letter

start = 10111
i = decimal(start) = 23
v = i + 97 = 120
Unicode(v) = x

The maximum decimal value of 5 bit binary number (11111) is 31. There are 26 letters in the English alphabet, and therefore, if the decimal value of a 5 bit binary number exceeds 26, the 5 bits will not form a letter, but will rather form some other arbitrary character. In order to prevent the initial population from containing characters other than letters, a check is performed after the generation of every 5 bits in order to determine if the 5 bits represent a letter. If they do not, then the 5 bits are discarded, and a new set of 5 bits is generated.

0.3.2 Fitness

When the genetic algorithm is run, it is passed the correct solution to the puzzle. This solution is then used by the genetic algorithm to measure the fitness of each solution in each successive generation. Fitness is a measure of the extent to which a solution generated by the genetic algorithm, matches the correct solution to the puzzle. That is, for each correctly placed letter in the solution generated by the genetic algorithm, fitness increases.

0.3.3 Selection

The selection method used is tournament selection with a tournament of size two. That is, each time chromosomes are selected for the next generation, two chromosomes are selected, and the fitter of the solutions enters the next generation with probability p, and the less fit with probability *(p-1)*. Tournament selection was chosen, because by changing the value of p, the selection pressure of the genetic algorithm can be varied.

0.3.4 Crossover

Single point crossover is used, and crossover occurs with with some probability p. Single point crossover does however create a problem regarding the integrity of solutions. During the generation of the initial population, special attention was paid to the fact that a 5 bit binary number can represent a character other than a letter, and in cases where it did, the 5 bit binary number was discarded and replaced. Single point crossover can create the same problem because crossover can occur at a point which creates solutions which contain characters other than letters. The example below demonstrates this problem - for each chromo-

some the binary encoding as well as the character values of the binary encoding are shown:

Parent 1: 1000100111 - qg
Parent 2: 1000111000 - qy
Crossover at point 8
Child 1: 1000100000 - qa
Child 2: 1000111111 - q?

As can be seen in the example above, when crossover takes place at point 8, Child 2 has an illegal character (?).

Interestingly, allowing this to happen without checking if crossover results in valid children, has a minimal impact on the success of the genetic algorithm. This is largely due to the fact that most of the time, crossover results in valid offspring, and these offspring generally have a higher fitness than the invalid ones and therefore have a higher chance of making it into the next generation. For the reasons stated above, the fact that crossover occasionally results in invalid offspring is ignored.

0.3.5 Mutation

For each bit in a solution, mutation in the form of a bit flit occurs with some probability p. As is the case with crossover, mutation can result in offspring being invalid due to the possibility of the resulting 5 bit characters not being letters. Once again, this fact is ignored because of the infrequency of occurrence, as well as the fact that valid solutions have a higher probability of making it into successive generations.

0.4 Experimentation

A series of experiments were performed on a crossword puzzle which consisted of 26 letters. Since fitness is a measure of the extent to which a candidate solution matches the correct solution, a correct solution would have a fitness of 26.

The genetic algorithm allows for the following four variables to be altered, and thereby affect the outcome and success of the genetic algorithm:

- population size
- probability of fitter solution being selected in tournament selection
- probability of crossover occurring
- probability of mutation occurring

Variables were tested in isolation in order to determine the effect that different values for each variable have on the success of the genetic algorithm. All other variables except the one being tested were kept constant at their default values. The genetic algorithm was run 10 times for each value of the variable which was being tested, and the average of the 10 runs was reported. Success was measured as the number of generations it takes for the genetic algorithm to find the correct solution. The lower the number of generations required to find the correct solution, the more successful the genetic algorithm was. Outliers in the data set were found occasionally, and occurred with a frequency of about 1/50. These outliers were labeled as being statistically insignificant and therefore were removed from the results which are being reported[1]. It is worthwhile to keep in mind however, that the genetic algorithm does not always perform with the efficiency that is shown below, and this fact was demonstrated by the occurrence of outliers in the data set.

One aspect of the performance and efficiency of the genetic algorithm which is not considered

[1] the full set of experimental results can be found in Appendix A

in this report, is the differences in processing time required for different values of variables. It is not considered for two reasons:

1. The main goal of this project is to investigate the effect of different values for genetic operators on the success of a generation, and not the time taken to come up with a solution
2. On modern day computers, the difference in processing times for different values of the variables considered, is for the most part, largely insignificant

0.4.1 Varying population size

An experiment was carried out to determine the effect that the population size has on the success of the genetic algorithm. The size of the population was varied, while the other variables were kept constant. The sizes of the populations being tested started at 1000 and then increased in increments of 1000 up to 10000. The results of the experiment are shown in the graph below:

Figure 0.2: The Effect of Population Size on Success

In the graph above, the Y-axis measures the number of generations required to come to a solution, and the x-axis measures the population size. The most noticeable thing about the graph above is the significant drop in the number of generations required to find the correct solution as the population size increases. The number of generations required to find the correct solution for population sizes of 1000 and 2000 is 500, while for all other population sizes, it stabilises at an average of about 20 generations.

The value of 500 generations to find a solution for population sizes of 1000 and 2000 is in fact a doctored number, which has been chosen to allow the graph to presented in a manner which is easy to follow. The real average number of generations required to find a solution for population sizes of 1000 and 2000, are 8033 and 2668 respectively. In fact, for population sizes of 1000 and 2000 it was not uncommon for the genetic algorithm not to find a solution in 10000 generations.

It is clear from the results presented above, that population size does have a significant impact on the success of a genetic algorithm. However, beyond some certain population size, there ceases to be any significant improvement in the success of the genetic algorithm, with an increase in the population size.

0.4.2 Varying probability of fitter chromosome being selected

Chromosomes are selected for the next generation using a tournament selection of size two. Of the two chromosomes selected to take part in the tournament selection, the fitter of the two is selected for the next generation with probability p, and the less fit with probability p-*1*. In this experiment, various values for p were used in order to determine the effect of p on the success of the genetic algorithm. The values for p which were used in the experiment started with 0.5 and then increased in

increments of 0.1 until 1. The results of the experiment are shown below:

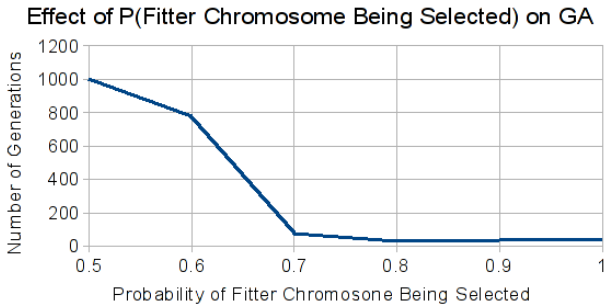

Figure 0.3: Effect of P(Fitter Chromosome Being Selected) on Success of GA

In the graph above, the Y-axis measures the number of generations required to come to a solution, and the x-axis measures the probability of the fitter chromosome being selected. Increasing the probability of the fitter chromosome being selected in tournament selection, increases the selection pressure of the genetic algorithm. The graph above clearly shows that an increase in selection pressure improves the genetic algorithm and allows it to find the solution in a smaller number of generations.

The optimal level of selection pressure exists when the probability of the fitter chromosome being selected is 0.8. Thereafter, the efficiency of the genetic algorithm decreases slightly. This decrease is due to the decrease in variation, caused by the higher selection pressure. The higher selection pressure causes the genetic algorithm to converge to a sub-optimal solution quickly, and then has to rely on mutation to find the correct solution.

As was the case with the effect of population size on the success of the genetic algorithm, one of the results here was doctored to make the graph easier to read. When the probability of the fitter chromosome being selected is equal to 0.5, the number of generation required to find a solution is not 1000. In fact, with the probability of selecting the fitter chromosome for entry into the next generation equal to 0.5, a solution was never found. This is because any improvements to a chromosome made through crossover and mutation have a high probability of being destroyed, and therefore, throughout the 10000 generations, there was almost no improvement in the population.

0.4.3 Varying probability of crossover occurring

Single point crossover occurs with some probability p. An experiment was carried out to investigate the effect that different values of p have on the success of the experiment. The values of p which were used in the experiment started at 0.5, and then increased in increments of 0.1 until 1. The results of the experiment are shown in the graph below:

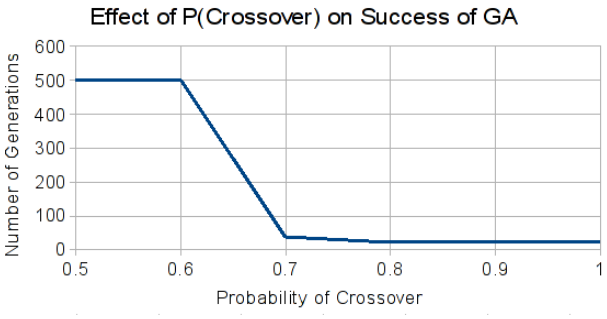

Figure 0.4: Effect of P(Crossover) on Success of GA

In the graph above, the Y-axis measures the number of generations required to come to a solution, and the x-axis measures the probability of crossover occurring. The graph shows that an increase in the probability of crossover, increases the success of the genetic algorithm. The genetic algorithm is most successful when the probability of crossover is 1 - that is, when

crossover is guaranteed to occur. The effect that an increase in the probability of crossover has on the success of the genetic algorithm stabilises at around a crossover probability of 0.8

As was the case with the previous variables investigated, two values have been doctored to allow for the graph to be read easily. The real number of generations required to find the solution when probability of crossover was 0.5 and 0.6, was 7596 and 1987 respectively.

0.4.4 Varying probability of mutation occurring

Bit flip mutation occurs on each bit in a chromosome with some probability p. An experiment was conducted to determine the effect that different values for p have on the performance of the genetic algorithm. The values of p used in the experiment were 0.01, 0.05, 0.1, 0.15 and 0.2. The results of the experiment are shown below:

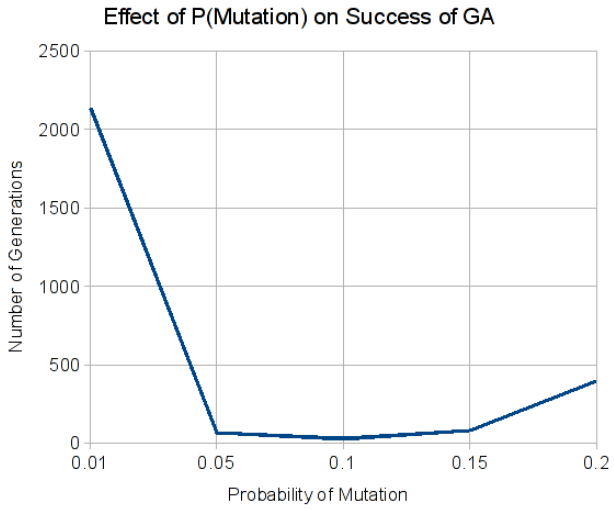

Figure: 0.5: Effect of P(Mutation) on Success of GA

In the graph above, the Y-axis measures the number of generations required to come to a solution, and the x-axis measures the probability of mutation occurring. The graph shows that when the mutation rate is too low, the genetic algorithm takes a longer time to find the correct solution. As the mutation rate increases, the success of the genetic algorithm increases, however, when the mutation rate gets too high, the success of the genetic algorithm once again decreases.

When the mutation rate is low, changes to candidate solutions occur infrequently and therefore it takes longer for the randomness in the genetic algorithm to find the correct solution. When the mutation rate is too high, any improvements which have been made in a generation, have a large chance of being lost. This shows that that mutation must be balanced in order to allow for variation to occur through randomness, but not too much so that it destroys improvements in the population. The genetic algorithm is most successful when the probability of mutation occurring is 0.1.

0.4.5 Optimal values for genetic algorithm

Two experiments were carried out in order to determine the optimal values for all the variables in the genetic algorithm.

0.4.5.1 Using optimal values identified in previous experiments

The first experiment used the optimal values which were identified in the previous experiments carried out. The motivation for this was that, if in isolation a certain value for a variable performs well, then when combined with other values for variables which perform well in isolation, the genetic algorithm should perform even better. The following values for the variables in the genetic algorithm were chosen:

- Population size: 5000

- Probability of fitter chromosome being selected: 0.9
- Probability of crossover occurring: 1
- Probability of mutation occurring: 0.1

The experiment showed that when using values which were optimal for each variable in isolation, the average number of generation required to find the correct solution was 31.3.

0.4.5.2 Using truly optimal values

During experimentation, all values for variables were recorded, as well as records of the performance of all combinations of values of variables. The following combination of values for variables were found, on average, to find the correct solution in the least number of generations:

- Population size: 5000
- Probability of fitter chromosome being selected: 0.9
- Probability of crossover occurring: 0.9
- Probability of mutation occurring: 0.05

Using these values for variables, the average number of generations required to find the correct solution was 19.5.

0.5 Key Findings

0.5.1 Isolation vs. combination

The values for variables which were found to be best, when each variable was tested in isolation, was not the same as when the variables were all tested together.

0.5.2 Low values for population size and genetic operators result in poor results

When the population size and probability of crossover and mutation taking place are low, The genetic algorithm performs extremely poorly. It usually takes several thousand generations to find the correct solution, and very often, no solution is found within 10000 generations, at which point the genetic algorithm stops.

0.5.3 Increases in population size and genetic operators improve results

It was found that when the population size, and probability of crossover and mutation taking place are low, the genetic algorithm performs extremely poorly. Increasing the values of these variables leads to a significant improvement in the performance of the genetic algorithm. However, beyond some point, these increases lead to little to no improvement in the performance of the genetic algorithm. It was found that, in some cases, increasing the probability of genetic operators occurring beyond some point, led to a decrease in the performance of the genetic algorithm.

0.5.4 Sub-optimal solutions are found quickly

The genetic algorithm quickly finds sub-optimal solutions to the problem. The crossword puzzle which was used as a test base, had 26 letters and thus the correct solution would have a fitness score of 26. It was very common for the genetic algorithm to hover around a fitness score of 20-25 before mutation would lead to the improvements which allowed the correct solution to be found.

0.6 Conclusions

0.6.1 Variables work in combination

The success of the genetic algorithm is a result of the combination of the values of variables and the interactions between them, rather than the individual effect of each of the variables. Therefore the optimal values for variables can not simply be determined based on their performances in isolation. Instead, the optimal values for variables are more likely to be found through trial and error while experimenting on different combinations of values for variables.

0.6.2 Randomness leads to solutions

Small population sizes and low probabilities of genetic operators occurring leads to it taking a long time for solutions to be found, or to solutions not being found at all. Population size and genetic operators introduce randomness into the genetic algorithm. If these values are low, the genetic algorithm struggles to ever evolve towards a better solution. Therefore, increasing these values increases the randomness in the genetic algorithm, and thereby improves its performance. Once the genetic algorithm has reached a sub-optimal solution, randomness in the form of mutation, is what ultimately leads it to finding the correct solution.

0.6.3 Too much randomness adversely affects the genetic algorithm

Increases in population size and the probabilities of genetic operators occurring, increases the randomness in the genetic algorithm, and allows it to find solutions in a smaller number of generations. Beyond some point however, too much randomness leads to the loss of good solutions, and increases the number of generations required to find a solution. This is because, with too much randomness, good solutions are destroyed through the frequently occurring genetic operators.

0.7 Final Remarks

A crossword puzzle was an interesting way to explore the use of genetic algorithms in finding solutions. This was because the solution was known before hand, and the efficiency of the genetic algorithm could be measured by how long it took to find that solution. Experimentation allowed for investigation into the various aspects of the genetic algorithm, and how they worked in combination to increase the efficiency of the algorithm. The GUI allowed for visualisation of the genetic algorithm operating in real time, and allowed it to be seen how a sub-optimal solution was quickly found, and then how mutation would ultimately lead to the correct solution. With the optimal combination of values for variables, the genetic algorithm proved to be highly efficient in finding the correct solution, usually doing it in 10 seconds or less. However, with a bad combination of values for variables, it would often take more than 10 minutes for the genetic algorithm to complete, and more often than not, it would not find a solution. This project shows that when a genetic algorithm is well designed, and makes use of good values for variables which are used in combination, solutions can usually be found quickly and efficiently.

---

[1] Wikipedia, The Free Encyclopedia. 2009. *Crossword.* [ONLINE] http://en.wikipedia.org/wiki/Crossword (06/04/2009)