EVCO Open Assessment

Y6375144

December 16, 2013

Chapter 1

Introduction

1.1 Genetic Algorithms

For some problems it can be difficult to find an optimal solution. This is because the search space grows at a much faster rate (exponentially, for example) than the input space. The Travelling Salesperson problem is a perfect example of this - for a search space with n cities, there are $n!$ paths that the salesperson could take.

Evolutionary algorithms do not present a solution to this problem. However, by taking inspiration from biological evolution, they do offer an alternative approach that will often lead to a 'good-enough' solution in a much more reasonable number of calculations.

Genetic algorithms are a type of evolutionary algorithm that typically represent population members (i.e. candidate solutions) in bitstrings. The population of strings is repeatedly updated through the following process:

1. Select - A number of the existing population members are selected to continue through to the next round. Typically these would be the fittest members in the current population.
2. Crossover - To generate new population members, existing ones are created by crossing two (or more) population members over. See figure 1.1 for a visual representation.
3. Mutate - The population is mutated at random. This typically involves passing over every bit in a string, and with a random probability flipping the bit from a 0 to a 1 (or vice versa).
4. Evaluate - A fitness function is run on each member of the population, giving an indicator of how 'good' a solution is.

Genetic algorithms hold much potential, but in order to get the most from them you must carefully choose your representation and parameters. The representation is the bit string typically, but it can also take other forms. There are many parameters that can be fine tuned, such as the number of offspring to breed (at the crossover stage), or the type of crossover to perform.

1.2 The Problem

Morpion Solitare is a one player game that is typically played on grid paper. To start, a cross shape is drawn in the centre of the page (as shown in figure 1.2). On each turn the player must add one cross that creates a line of five crosses either vertically, horizontally or diagonally. The lines may intersect, and in this particular variant the lines may also overlap (it could be argued that this makes the game easier). Figure 1.2 shows

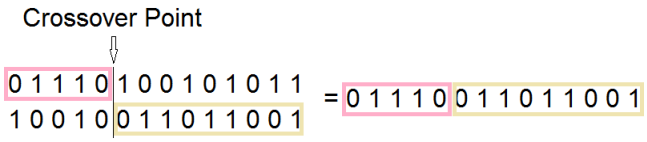

Figure 1.1: The child is made up of the top row (which represents the first parent) until the crossover point, where it then takes from the bottom row (the second parent).

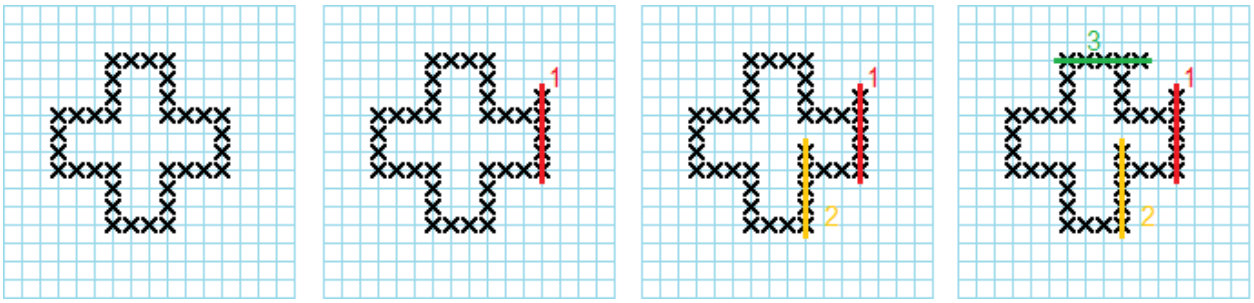

Figure 1.2: The initial state of the game on the left, and from left to right the first three moves are shown. The lines don't have to be numbered, but for clarity they are here.

a possible first three moves. Finishing at this stage would give an overall score of 3, as 3 lines of 5 crosses have been formed.

In this report I will detail my investigation into obtaining a high score through the use of a genetic algorithm. Working out the fitness of a population would be a tedious process if I have to take each population member and draw out its game from scratch. Thankfully that is not necessary as I have been provided with a computerised version of the game that is slightly faster.

When designing my genetic algorithm I will have to consider the format that the digital version takes its input by. The program represents positions on the grid by their (x, y) co-ordinate positions, where $(0, 0)$ is the bottom left of the grid. For each move, four parameters must be passed into the program in the format $x_1y_1x_2y_2$, where (x_1, y_1) is the position of one end of the line to draw, and (x_2, y_2) is the other end of that line. For the game shown in figure 1.2, the input would be:

```
linuxMsolitare.exe 12 11 12 7 9 9 9 4 6 13 10 13
```

The program accepts a maximum of 2000 inputs, or 500 moves. Given that the current best known solution is 178 [**?**], this is unlikely going to be a problem. One potential problem is that to evaluate a population of size n, the application will need to be called n times. Even with a decent processor, this could be a slow task for any sizeable population. Also, if a representation is chosen that doesn't directly map to the program's inputs is chosen, then each population member's representation will have to be converted to the correct format before the program can even be called. Again, this could be time consuming, but solutions to these problems will be considered in the next section.

Chapter 2

The Solution

2.1 Representation

2.2 Selection

2.3 Crossover

2.4 Mutation

Chapter 3

The Results

3.1 Conclusion