Ollscoil na Gaillimhe
University of Galway

Assignment One CT421 - Evolutionary Search (GAs)

Student Name: David Bohan **Student ID**: 20436904 **Programme**: CT421

Overview

Genetic algorithms have been used as a search mechanism to identify solutions to a range of problems, where the use of brute force is impractical due to its exponential nature. They are based on the study of adaptive processes occurring in natural systems and imitate the process of natural selection.

Genetic algorithms begin with a population of individual specimens, with each specimen a unique genome that encodes a potential solution to the problem as a binary bit-string. The set of all current solutions at a given point in the algorithm is called the Generation. At Generation 0, we have a population of randomly initialised genomes.

To evaluate the effectiveness of each genome, a problem specific fitness function is used to determine how good a that genome is in terms of solving the problem. Generally, higher fitness scores are better for "reproducing" and are selected to act as parent specimens to the next generation.

Two parents are selected and "cut" at random indexes in the genome – allowing for the exchange of genetic material from A --> B and B --> A. This process is known as single point crossover and it results in two new genomes for the subsequent generation. We repeat this process until a new Generation is fully formed.

Due to the fact that the selection and crossover functions are governed by randomness, there is no way to guarantee we don't accidentally destroy our globally optimal genome(s). A process called Elitism is used to account for this, whereby we preserve our top N solutions by directly copying them into the next generation without any changes.

During the mutation phase, a random bit in the genome's binary sequence is altered based on a pre-defined hyperparameter, ie. 0 --> 1 or 1 --> 0. Mutations are used to help uncover new solutions which otherwise would not have been possible based on the initial population.

The genetic algorithm will continue to loop until it meets the desired criteria or else it hits a fixed limit on the number of generations.

One-max Problem

When working with a real-world genetic algorithm, it is typically impossible for us to know if a generated solution is the best and optimal solution. In the case of evolving a bit-string that contains all 1s in every location, we simply know the optimal solution. This allows us to perform analysis which may not always be possible.

Results

- The optimal solution found after approixamtely 27-37 generations on the majority of runs, however one run observed iterated 66 generations – showing the effect of randomness in genetic algorithms.
- The best fitness in the first generation was approaximately 19-22 on most runs. Best fitness increased non-linearly, with a fast start and slower end.

- Average fitness improved approxiately 1.54% from one generation to the next.

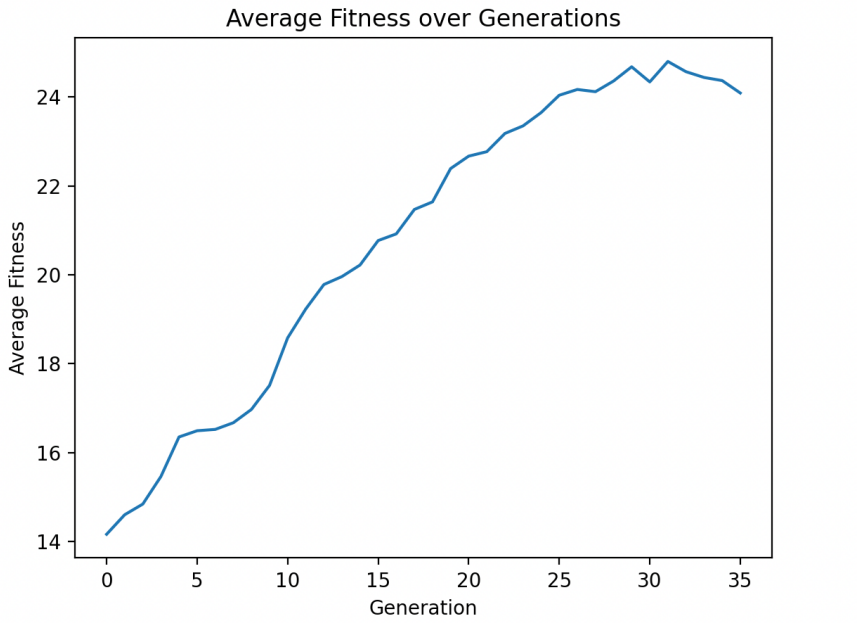


Figure 1: Average fitness of the population versus the generations passed

Evolving to a Target String

The task of evolving the target bit-string shares fundamental similarities with the above example of evolving a bit-string of all 1s, with the primary distinction being in the fitness function. We are now assessing the alignment between the current genome and the target string by comparing each corresponding bit. In theory, the dynamics of the evolution ie. the number of generations required to find the optimal solution, the average and best fitness scores—should be equivalent to the previous problem, given their similarity in complexity. We may observe a very minor difference in runtime due to the increased computational overhead associated with the added comparison operation in the fitness function.

The graphs below show the variablilty in genetic algorithms. Figure 2 shows the convergence on the tagret string after 71 generations, whereas Figure 3 shows the same convegence after just 24 generations.

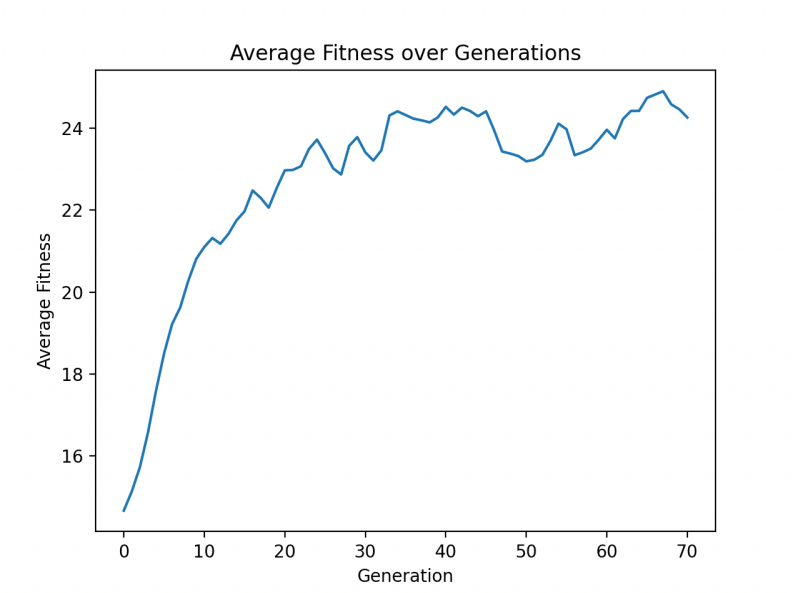


Figure 2: Average fitness of the population versus the generations passed – 71 generations

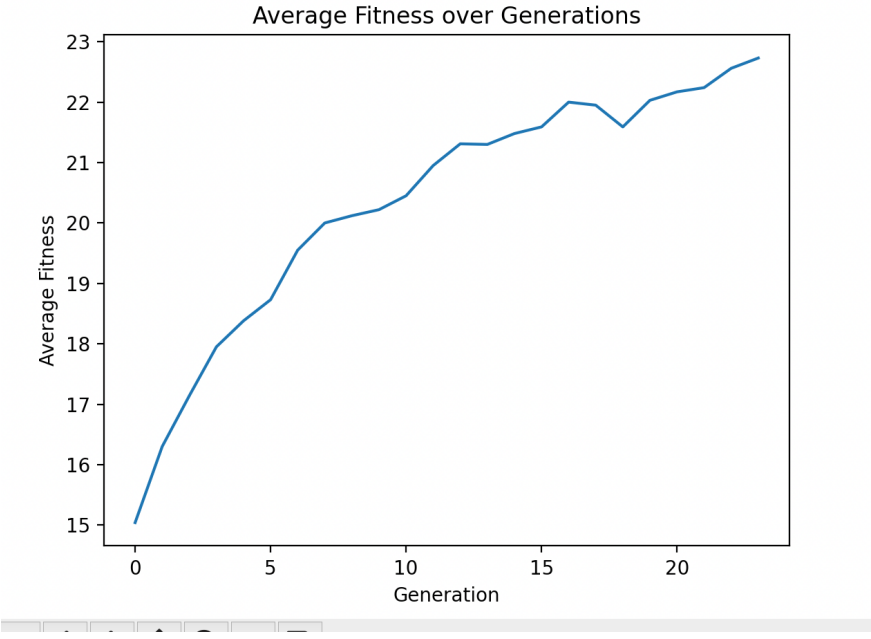


Figure 3: Average fitness of the population versus the generations passed – 24 generations

Deceptive Landscape

Deceptive problems introduce a twist in the typical evolutionary process, altering the algorithms ability to accurately predict fitness and therefore misleading the algorithm away from a globally optimal solution. In our case, the fitness function misleads the genetic algorithm through assigning an incorrectly high fitness value to a genome with an all 0 bit-string, doubling the solution's length, therefore obscuring the true optimal solution of all 1s. It results in the algorithm getting "stuck" on a local maxium of 30 because it will never encounter an all 0 string.

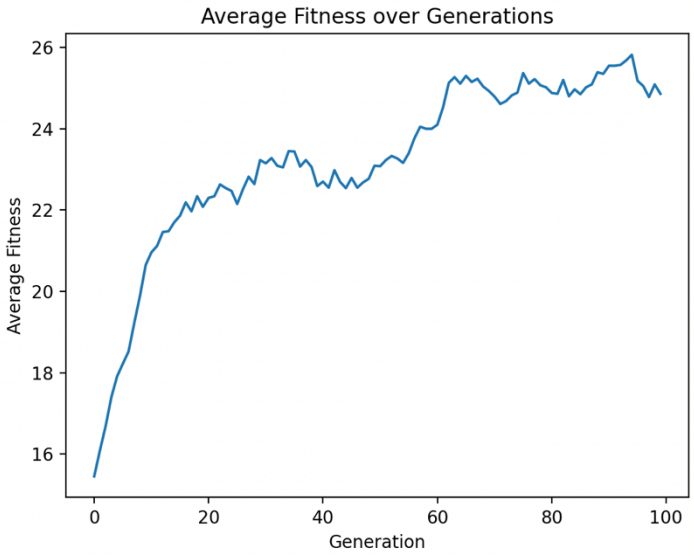


Figure 4: Average fitness of the population versus the generations passed – Deceptive Algorithm

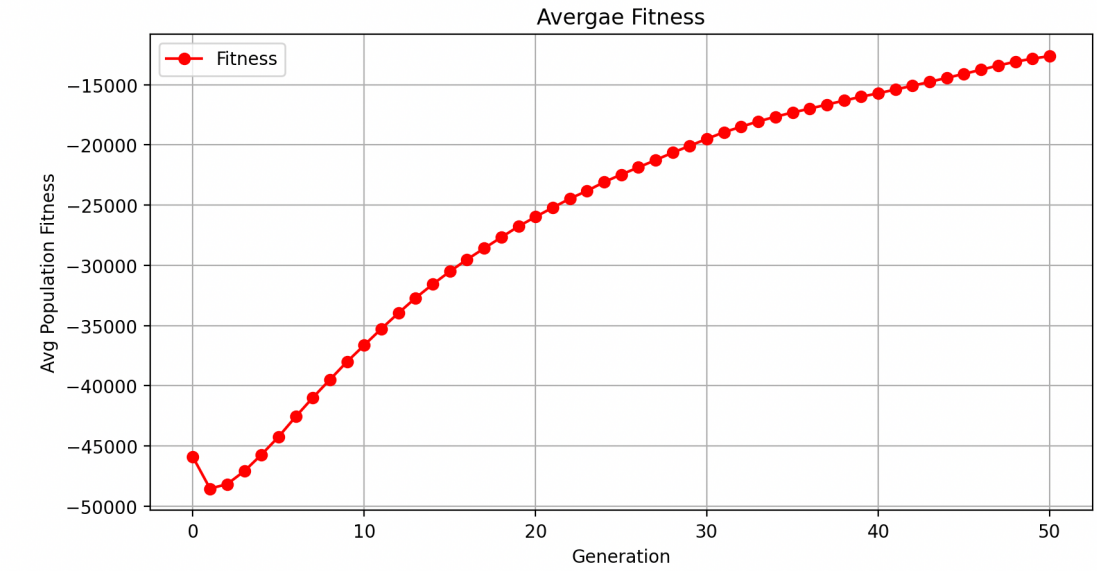
```
Generation 57: Average Fitness = 23.32 -- Best Fitness = 29
Generation 58: Average Fitness = 23.49 -- Best Fitness = 29
Generation 59: Average Fitness = 23.66 -- Best Fitness = 29
Generation 60: Average Fitness = 23.76 -- Best Fitness = 30
Generation 61: Average Fitness = 23.91 -- Best Fitness = 30
Generation 62: Average Fitness = 24.15 -- Best Fitness = 30
Generation 63: Average Fitness = 24.25 -- Best Fitness = 30
Generation 64: Average Fitness = 24.45 -- Best Fitness = 30
Generation 65: Average Fitness = 24.21 -- Best Fitness = 30
Generation 66: Average Fitness = 24.36 -- Best Fitness = 30
Generation 67: Average Fitness = 24.44 -- Best Fitness = 30
Generation 68: Average Fitness = 24.54 -- Best Fitness = 30
Generation 69: Average Fitness = 24.47 -- Best Fitness = 30
Generation 70: Average Fitness = 24.66 -- Best Fitness = 30
```

Figure 5: Deceptive Algorithm Output

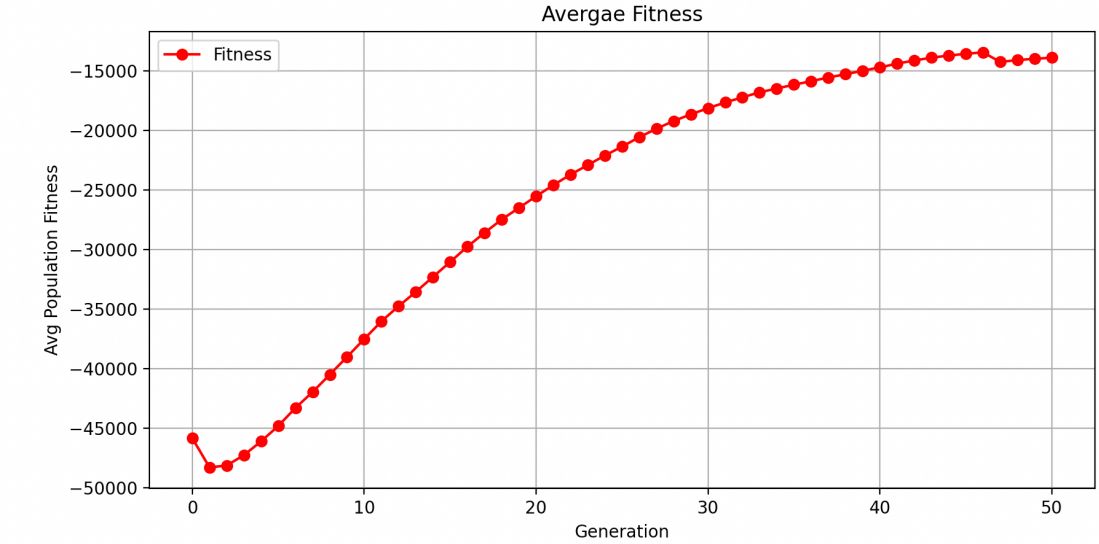
Binpacking Problem

In the bin-packing problem, we have a set of N items, each with a weight of below 1. We also have an infinite amount of bins with a maximum capacity of 1. Our goal is to pack all of the items into as few bins as possible. The fitness function evaluates the quality of a solution based on three criteria: overflow (items that don't fit within the maxium bin capacity), underflow (wasted space within bins where they are not completely full), and items omitted (not placed in any bin). Our goal is to minimise penalties, with the fitness score being a negative value where lower scores indicate better solutions. This approach means solutions aim to minimize the number of bins while maximising space utilisation and making sure all items are included.

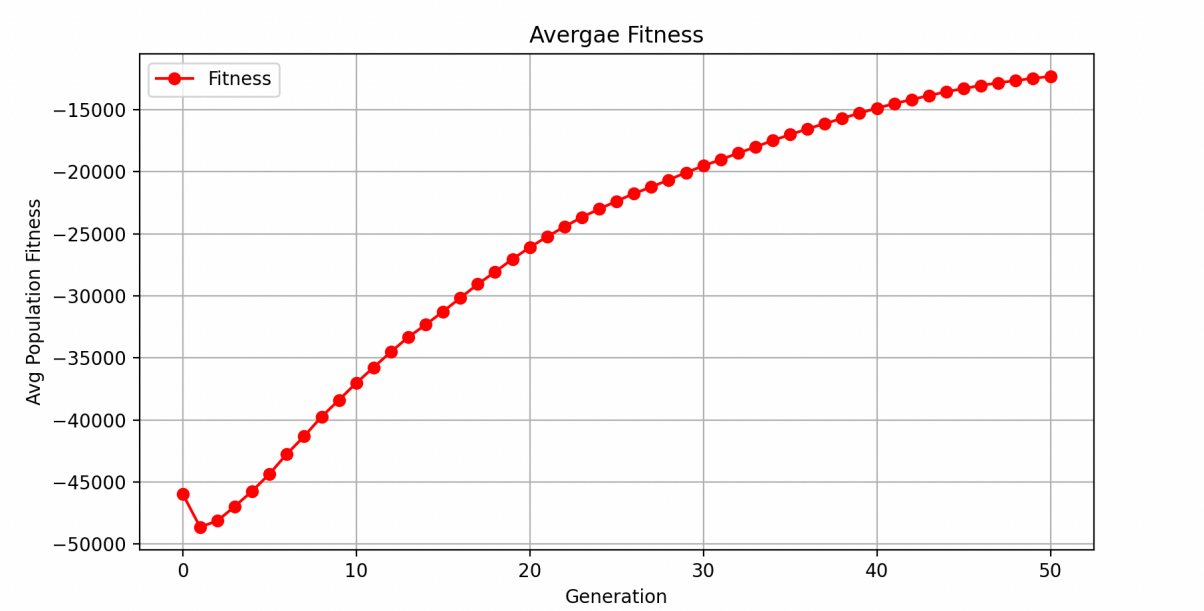
Data Set One – Best Bins 15



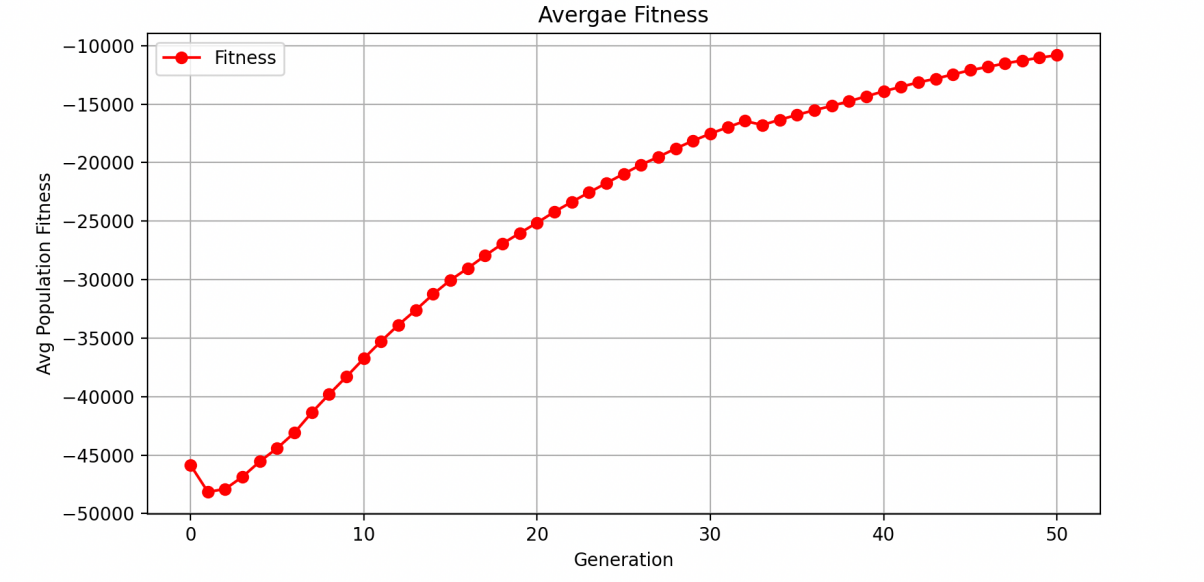
Data Set Two – Best Bins 19



Data Set Three – Best Bins 21



Data Set Four – Best Bins 15



Data Set Five – Best Bins 18