ASSESSMENT 1

40278490

SET11508 Emergent Computing for Optimisation

Table of Contents

1 Introduction

This coursework entails the creation of a stochastic algorithm to find a fantasy football team that maximises the point score and satisfies the following seven constraints:

- Total cost for the team must not exceed £100.
- There must be 11 players on every team.
- Each player must be unique.
- There can only be one Goalkeeper (GK).
- There must be between 3 and 5 Defenders (DEF).
- There must be between 3 and 5 Midfielders (MID).
- There must be between 1 and 3 Strikers (STRI).

A CSV file containing all the player information from the 17/18 football season was provided, with columns for player name, position, cost, and score. There are 523 total players within the file, and from these the developed algorithm must select the optimal 11 players.

2 Approach

2.1 Algorithm

As the selection of a team can be viewed as a knapsack problem with constraints or an NP-hard combinatorial optimisation problem, a generational evolutionary algorithm will be used to find an optimal solution. Evolutionary algorithms (EAs) are stochastic meaning that they utilise randomisation to find an optimal solution, and they function using four main operators; select, evaluate, mate, and mutate (Smith, 2015). These operators will be explored more later in the paper. An evolutionary algorithm was chosen over a local-search algorithm as local-search requires all combinations to be known to find and identify a local-optima, which can be incredibly computationally intensive and time-consuming (Smith, 2015).

2.2 Language

Python was chosen to program the algorithm for several reasons:

- Existing examples and practical material could be used to base this solution algorithm on.
- Libraries for the creation, and implementation of evolutionary algorithms and statistics exist for Python that are extensively documented, for example DEAP[1] and Pandas[2].

As Python is an interpreted language, it can be relatively slow to run compared to compiled languages like C++ that have similar, less documented libraries and packages to deal with evolutionary algorithms, for example openGA[3]. However, run time is not a critical aspect of this algorithm, therefore, Python is more suitable for this task.

2.3 Representation

A standard representation for representing an individual, a knapsack or a chromosome is through a bit-string/binary-string, with 1s representing items taken and 0s representing items left (Lin et al., 2003; Smith, 2015). This bit-string would have the same length as the dataset and thus if a player was chosen for the team, all that is needed is to get the player index and change the 0 at said index in the bit-string to a 1. This would prevent duplicate players as the value can only set to 1 once.

[1] https://github.com/DEAP/deap
[2] https://pandas.pydata.org
[3] https://github.com/Arash-codedev/openGA

Other representations such as an array of size 11 with each element representing the index of a player could also be appropriate, however, bit-strings will be used for this algorithm.

3 Algorithm and Operator Design

3.1 Initialisation

To initialise the population, a custom function was created that generates an array of 11 random numbers between 0 and 1 and then multiplies each element by the number of players. When the elements in this array are parsed into integers, they represent the indexes of the players in the data. An array of 0s (the bit-string) is created that is the same length as the number of players (523), then the 0s are changed to 1s where the index equal to the value of each element in the randomised number array. To prevent the generation of teams with less than 11 players, each team is passed through one of the evaluation functions and a team with the correct number of players and positions is returned. This results in the initial population being composed of nearly valid individuals, consistently passing all constraints except total cost. Cost is calculated and evaluated later with the total points in the final evaluation function.

3.2 Evaluation (Fitness) Function

As there are constraints on what a valid team can be, a custom evaluation function was created for the EA based on a knapsack evaluation function. The fitness function takes a randomly created team and checks if it fulfils all constraints, returning the sum of points for the players within the team as a fitness value. However, if the team breaks one of the provided constraints (see 0), the function will return a fitness of 0 regardless of total points scored.

3.3 Select Operator

Tournament selection, also known as *tournament k* is a very popular selection operator for genetic algorithms (Lavinas et al., 2019). This type of selection takes 2 or more random individuals (teams) and selects the best ones to reproduce, allowing the total population to improve with each new generation. As the DEAP library has a built-in tournament selection operator for EAs, it will be used for this algorithm with experimentation to determine *k*, the best number of individuals to be compared.

3.4 Mate Operator

N-point crossover is a relatively popular method of mating two individuals in evolutionary algorithms, however, it is not clear whether a one-point crossover offers any advantage over a two-point crossover other than computation time (Lin et al., 2003; Smith, 2015). As both one- and two-point crossover operators are available within the DEAP library, each will be tested to see if one offers an advantage over the other.

3.5 Mutate Operator

As each individual will be represented as a bit-string/binary-string, a standard bit-flip mutation operator will be implemented. This function is already provided by the DEAP library; therefore, it will be utilised for this algorithm with experimentation to determine the best mutation rate.

4 Experimentation

Throughout all experiments, unless being the subject of the experimentation itself, the following values were used consistently as defaults:

- Mutation Probability (MUTPB) of 0.01 (1%)
- Population (POPSIZE) of 500
- Number of Generations (NGEN) of 50
- Tournament Size of 2

- Mate Operator as two-point crossover
- Number of runs in each data set as 10

The significance tests used to justify the results given by the following boxplots are the Student's t-test (Mishra et al., 2019) for Gaussian or normal distributions and the Mann-Whitney U-test (Beatty, 2018) for non-Gaussian or non-normal distributions. Alternative tests such as the unequal variance t-test could equally be used in the case that one or both sets of data have non-normal/non-Gaussian distributions (Ruxton, 2006) or the two-tailed t-test in Gaussian samples (Smith, 2015). The two significance tests used were chosen based on their convenience as both can be found in the scipy library for Python.

4.1 Exploring the effect of population size

To see whether starting population size affects the overall fitness, the EA was run with an initial population of 500, increasing by 500 after each set of runs until it reaches 2000 initial population.

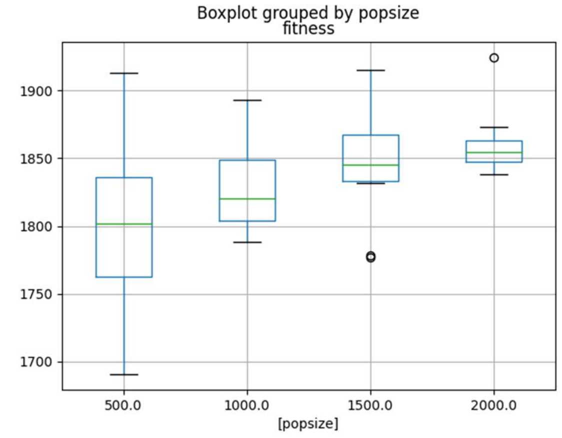
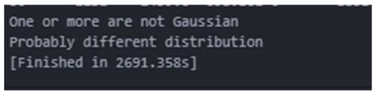


Figure 1 Boxplot showing the spread of best fitness with each population.

The results, as shown in Figure 1, suggest that increasing the population positively influences the best fitness. This can be seen in the boxplot as most of the 2000-population box is within the upper quartile of the 500-population box, however, has a lower spread than the lower population. Through running the significance test (Mann-Whiney U-test), we can reject the null hypothesis and determine that having a larger population positively influences the fitness of a population (Figure 2).



```
One or more are not Gaussian
Probably different distribution
[Finished in 2691.358s]
```

Figure 2 Results from running a significance test for an initial population of 500 vs 2000.

4.2 Exploring the effect of the number of generations

To explore the effect of increasing the number of generations on best fitness, the EA is run with a starting number of generations (NGEN) of 10, increasing by 10 after every set of runs until it reaches 50 generations.

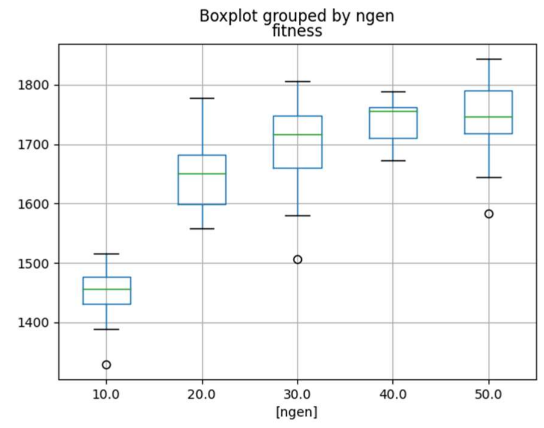
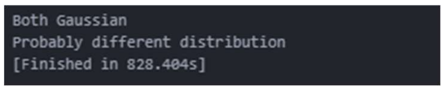


Figure 3 Boxplot showing the spread of best fitness for each number of generations.

Figure 3 suggests that there is a drastic difference in changing the number of generations that the EA runs as the box plots for the two extremes are mutually exclusive, including outliers. This is further shown in Figure 4, where the results of the significance test suggests that both sets of data belong to separate distributions, therefore, it can be said that having 50 generations is probably better than having 10 generations. This is likely because it allows the algorithm to reach a peak and for the population to converge.



```
Both Gaussian
Probably different distribution
[Finished in 828.404s]
```

Figure 4 Results from running a significance test for NGEN = 10 vs 50.

4.3 Exploring the effect of mutation rate

To test whether mutation probability (MUTPB) affected fitness, the EA was run with mutation probabilities of 1% through to 5%.

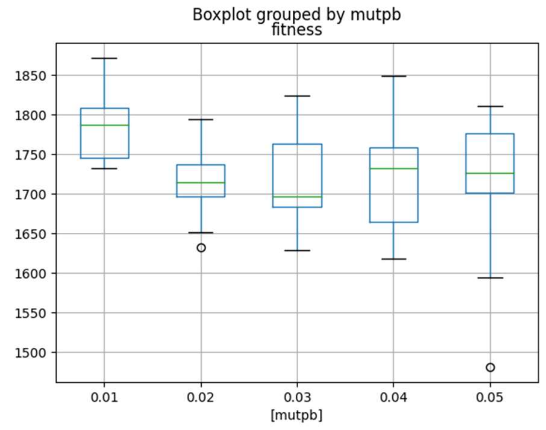
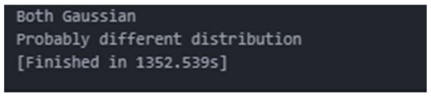


Figure 5 Boxplot showing spread of fitness for a mutation probability of 1% through to 5%.

Figure 5 suggests that there is a noticeable difference between a 1% mutation chance and a 5% mutation chance, with 1% probability producing better fitness overall, as shown by the lowest quartile of the 1% box being above the median of the 5% box. This distinction is further reinforced by running the significance test on the two sets of data, the results of which can be seen in Figure 6. From this, we can reject the null hypothesis and conclude that having a lower mutation probability likely produces a better fitness.

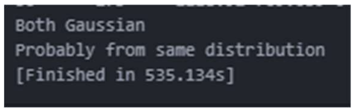


```
Both Gaussian
Probably different distribution
[Finished in 1352.539s]
```

Figure 6 Results from significance test for Mutation Probability.

4.4 Exploring the effect of mate operators (Crossover Types)

To determine whether crossover function affected the spread of best fitness, the EA was run separately using a one-point crossover and a two-point crossover. The distributions of best fitness are similar in both operators, suggesting that the crossover operator does not greatly influence overall fitness. Shapiro testing indicates that both sets of data probably follow Gaussian distribution. Further significance testing (Student's t-test) suggests that both operators return the same distribution, thus, it is likely that choosing either operator will not produce a significant difference (Figure 7).



```
Both Gaussian
Probably from same distribution
[Finished in 535.134s]
```

Figure 7 Results from significance test for Mate Operators

4.5 Exploring the effect of tournament size

To evaluate the effect of tournament size on the fitness of a child team, the EA was run with 2, 6 and 10 tournament sizes. While there is not enough evidence to suggest that tournament size affects the spread of best fitness, it is clear that when plotted against the generation where the best value was found (Figure 8), the EA can reach a best value in far fewer generations when the tournament size is higher.

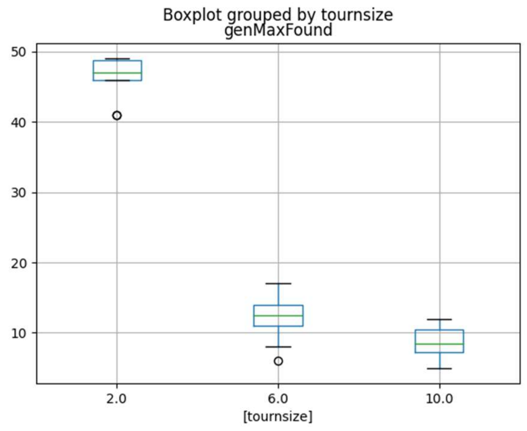


Figure 8 Boxplot showing the spread of generations where the max value is found for each tournament size.

5 Best Solution

Based on the results of the experimentation, the final algorithm uses the following values for the variables tested:

- Mutation Probability (MUTPB) of 0.01 (1%) as lower mutation probability gave a better fitness.

- Population (POPSIZE) of 5000 as a higher starting population indicated a better best fitness.
- Number of Generations (NGEN) of 100 as a higher number of generations allowed the peak value to be reached.
- Tournament Size of 2 as this indicated the best fitness.
- Mate Operator as two-point crossover as there was no clear evidence of a difference between one- and two-point crossover.

After a couple of unsatisfactory runs, one run resulted in a score of 2072 and a cost of exactly £100 as can be seen in Appendix. This solution breaks none of the constraints.

6 Evaluation

A possible strength of the algorithm that has been produced is that it appears to find a valid and high scoring team, with every single run. However, the returned results produced so far cannot be held representative of the entire distribution and thus cannot be said that this algorithm is better than another, thus, this can be seen as a weakness rather than a strength. Therefore, it would be valuable to test the algorithm multiple times and compare it to other algorithm solutions by running significance tests on the produced data. The compared algorithms could follow the same algorithm template with the "worst" values for each of the variables.

Another strength can be found in the custom functions (initialisation and evaluation) that have been created for the algorithm. These allow for a higher degree of control over the algorithm; therefore, it would be beneficial for the rest of the algorithm operators to also be customised methods. This would also reduce the reliance on external libraries, a current weakness of the present algorithm.

Another weakness of the algorithm is the lack of testing data that has been taken. This can be seen in the 10 runs of the algorithm for each tested variable; however, this value was chosen as it allowed for the testing of each variable type within a reasonable timeframe. Figure 2 shows the results of the significance test on distributions for each starting population along with the time taken to complete the test (around 45 minutes). By running the algorithm 50 times, we would see a far more accurate depiction of the distributions, and therefore could make each experimental assessment with far more certainty. Thus, further justifying the values for the final run of the algorithm. Furthermore, not every variable was experimented with, as crossover probability (CXPB) testing was left out. Resulting in uncertainty around the algorithm's capabilities to produce a better distribution of results and consequently a better final team.

Lastly, a possible future experiment would be to use a different data set with the same column titles as the algorithm has been designed to handle any dataset that follows the same format as the one provided, however, this has not been tested.

7 References

Beatty, W. (2018). *Erratum to: Decision Support Using Nonparametric Statistics*. https://doi.org/10.1007/978-3-319-68264-8_11

Lavinas, Y., Aranha, C., Sakurai, T., & Ladeira, M. (2019). Experimental Analysis of the Tournament Size on Genetic Algorithms. *Proceedings - 2018 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2018*, 3647–3653. https://doi.org/10.1109/SMC.2018.00617

Lin, W. Y., Lee, W. Y., & Hong, T. P. (2003). Adapting crossover and mutation rates in genetic algorithms. *Journal of Information Science and Engineering*, *19*(5), 889–903.

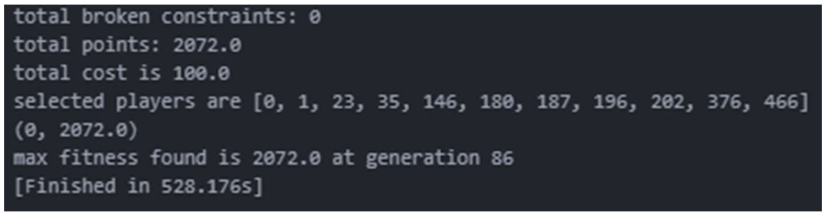
Mishra, P., Singh, U., & Pandey, C. M. (2019). *Application of Student ' s t - test , Analysis of Variance , and Covariance*. 407–411. https://doi.org/10.4103/aca.ACA

Ruxton, G. D. (2006). The unequal variance t-test is an underused alternative to Student's t-test and the Mann-Whitney U test. *Behavioral Ecology*, *17*(4), 688–690. https://doi.org/10.1093/beheco/ark016

Smith, J. E. (2015). *Introduction toEvolutionaryComputing*. papers2://publication/uuid/F189C4FB-EB51-43FE-B6FC-17BAF1BF36C8

8 Appendix

8.1 Solution



```
total broken constraints: 0
total points: 2072.0
total cost is 100.0
selected players are [0, 1, 23, 35, 146, 180, 187, 196, 202, 376, 466]
(0, 2072.0)
max fitness found is 2072.0 at generation 86
[Finished in 528.176s]
```

Figure 9 Result of the constraint checker function.