SET11508 Optimisation of a Fantasy Football Team

1 Approach

1.1 Representation

Binary representations are not always the most appropriate ones. Even though the binary representation can be used in this case, dealing with constraints may prove difficult. Real-valued representations are used when we want an integer to represent a particular value [1]. The representation scheme used in the initialisation, crossover and mutation is that each individual is represented by a vector of size 11 containing integers corresponding to the index of the player's position in the Excel sheet dataset. Motivation for this representation scheme was taken from: [2], where N indexes represent the position of a team; [3], where each player corresponds to the position in the Excel sheet; [4], where a code vector is used where again each index corresponds to a position of the team and other papers dealing with team formation, resource allocation and combinatorial optimisation where no constraints have to be broken [5] [6] [7]. This ensures that the team size is always eleven and decreases issues within crossover and mutation.

1.2 Algorithm

Evolutionary algorithms are stochastic search methods inspired by the natural biological evolutionary process that address various optimisation problems. Grouping problems, specifically Team Formation (TF), are a particular combinatorial optimisation problem. Given that the solution process required represents high complexity, NP-hard. Multiple papers [8] [9] find that Genetic Algorithms (GAs) have demonstrated the best performance in most cases. The GA is genetic and can be implemented differently according to the problem. Since the TF problem can have global optima search space, in this case, it is unknown whether there is only one single best solution to the problem or multiple. Multiple solutions can have the same points and costs because the team consists of points and costs. Furthermore, a more comprehensive range of solutions is evaluated using a population. The diversity in the population has been used as an advantage as it acts like a local search procedure [10].

2 Algorithm/Operator Design

Algorithm: The generational evolutionary algorithm has been used where the offspring replaces the whole population. The generational evolutionary algorithm is used because it has been found through testing that it generates individuals with the highest fitness scores, even though they exceed the cost of 100. Furthermore, the algorithm itself was modified to store the best individual without the cost constraint. The check is done in each generation, and the best individual from the previous generation is compared with the best individual in the current generation; whichever is the best is stored – this process is repeated until the end. This feature has been implemented into the algorithm because while generating only feasible solutions, it breaks the cost constraint even though the proportional evaluation function is used. The best solution can be lost when individuals are passed to the next generation due to mutation and crossover. The algorithm stops when 150 generations are reached, the individuals that do not exceed the cost of 100 are lost in the process. Furthermore, the hall of fame stores the best individuals of the size of the population, but that does not guarantee that a feasible solution will not be replaced by a solution that breaks the cost constraint.

Initialisation: During initialisation, the first eight positions of the vector are filled by the minimum amount of players in a position to satisfy the constraints. The other three players in the vector are chosen randomly as long as the number of players' positions does not break a constant (Table 1). This ensures that the algorithm's initialisation process produces feasible solutions, reduces the search space, and moves the search closer to the feasible region where the only constraint that may be broken is the cost [11]. This vector is transformed to an individual of a binary array of length 523, where a bit set to 1 indicates the player is included in the team. This binary transformation also ensures that no player is included twice [12].

Initialising a population of feasible solutions is a standard way to deal with constraints [13]. Others [14] referenced the penalty function for infeasible solutions. However, there is no easy and appropriate way to determine the penalty for each constraint as a solution can be infeasible in many ways and without prioritising one constant over another, it is an inappropriate method except if the death penalty method is used. However, [15] concludes that it is not well suited for Evolutionary Algorithms.

DEF	DEF	DEF	DEF	MID	MID	MID	MID	STR	STR	GK

Table 1. Feasible Individual (Light (yellow) – randomly selected positions, Dark (purple) – hard-coded positions)

Selection: The reproduction of better individuals is achieved with a selection mechanism. Choosing a selection mechanism has been carefully considered regarding maintaining diversity in the population and selective pressure. Because the initialisation is done randomly, the proportion of individuals with high fitness and low fitness is unknown. [16] and [17] conclude that good genetic diversity is preserved as the Stochastic Universal Sampling prevents individuals with exceptionally high fitness values from saturating the next generation. Tournament selection seems to perform better in that it compares individuals' fitness and chooses the best one and because individuals can break the cost constraint, the fitness function tries to penalise such individuals, so this could be why the tournament is working better in experiments. Stochastic Universal Sampling found the third-best record over many runs, and it was the primary selection method until it was compared to Tournament – Tournament selection was then tested, and over a few runs, it found two individuals that outperformed the other selection method in fewer runs. Similar problems have also been solved with the Tournament selection [3].

Crossover: The representation of an individual is converted back to a vector of size 11 during the crossover. When the binary array is converted back to an ordered vector of integers, the indexes represent players of a specific position [15] (See Table 1). The crossover operator produces only feasible solutions, so the crossover is applied based on positions. No crossover is applied if selected individuals have the same genes. The available players for each position to crossover are found, and the duplicates are removed. This gives a list of players to crossover and the length of available players for each individual. After randomly choosing how many players based on the minimum amount of players for a position that can be crossed over from both individuals, each individual undergoes a crossover where a random index is swapped with one randomly chosen player of the available players that were included in the other individual excluding players that already exist in the individual. This crossover produces two new individuals out of the parents regardless of fitness. This promotes diversity as even if multiple individuals within the same generation have the same genes, the offspring will be different on the crossover. [2] The crossover operator was manually designed to create feasible solutions, given that each player in most positions could have a different number of players. Using any off-shelf crossover operators would create unfeasible solutions. The implementation was inspired by the n-Point Crossover (where a minimum number of players per position are swapped) and the Uniform Crossover (where the position to be swapped is chosen at random while still having a chance of preserving the parents' genes). [18]

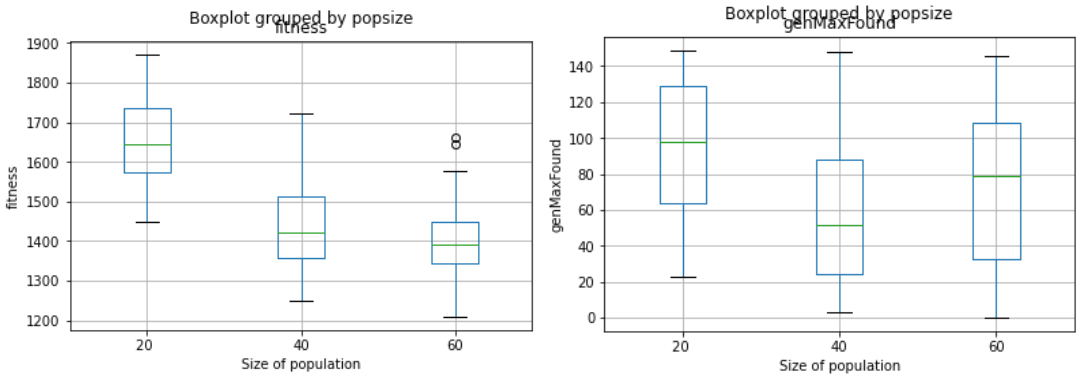
Mutation: The evolutionary algorithm has used the random resetting mutation as all values are equally likely to be chosen while still producing feasible solutions. The mutation operator uses the 11-size integer vector, and each index has a 50/50 chance of getting mutated. A gene is mutated based on the position the integer represents, so a random integer from the specific position range is chosen until it does not yet exist in an individual [18]. Using any off-shelf mutation operators might lead to unfeasible individuals, forcing the search outside of the permissible solutions.

Evaluation Function: The proportional penalty fitness function is used to try and decrease the number of individuals that break the cost constraint.

3 Experimental Design & Analysis

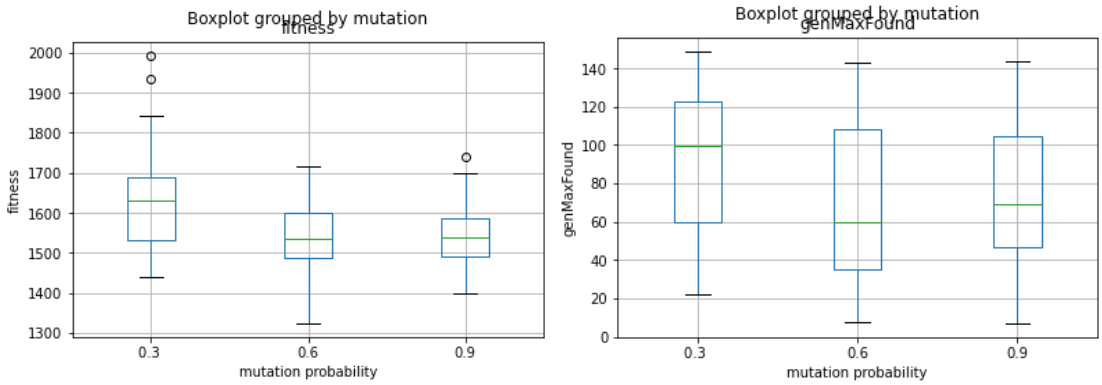
Parameter settings: population - 20, crossover probability – 0.5, mutation probability – 0.3, max generations – 150, selection proportion – population size (generational evolutionary algorithm), selection method – **stochastic universal sampling**, evaluation function - the proportional penalty fitness function. Experiments have been looped 50 times. Only specific parameters that are expected to have no effect have been changed.

3.1 Changing the population size has no effect on fitness and genMaxFound



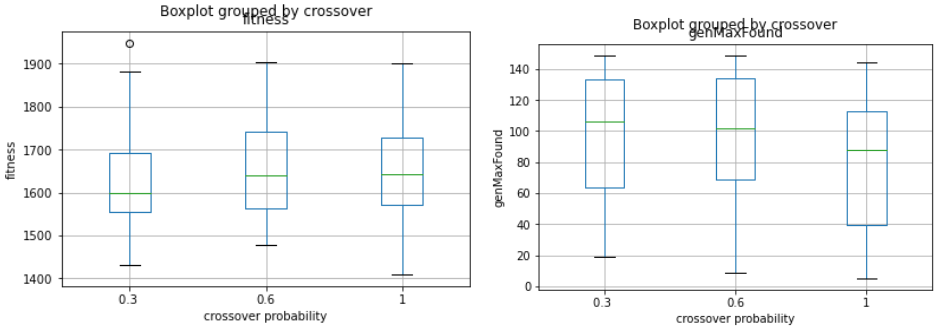
This null hypothesis is rejected for popsize=20 and popsize=40; popsize=20 and popsize=60 but cannot be rejected for popsize=40 and popsize=60. It can be observed that having a smaller population promotes the generation of individuals with a higher fitness score and decreases as the population increases and decreases the time to run the algorithm.

3.2 Changing the mutation probability has no effect on fitness and genMaxFound



This null hypothesis is rejected for mutation=0.3 and mutation=0.6; mutation=0.3 and mutation≈0.9 but cannot be rejected for mutation=0.6 and mutation≈0.9. Interestingly, it can be observed that having a minor mutation promotes the generation of individuals with a slightly higher fitness score in this experiment but is found after generation 20.

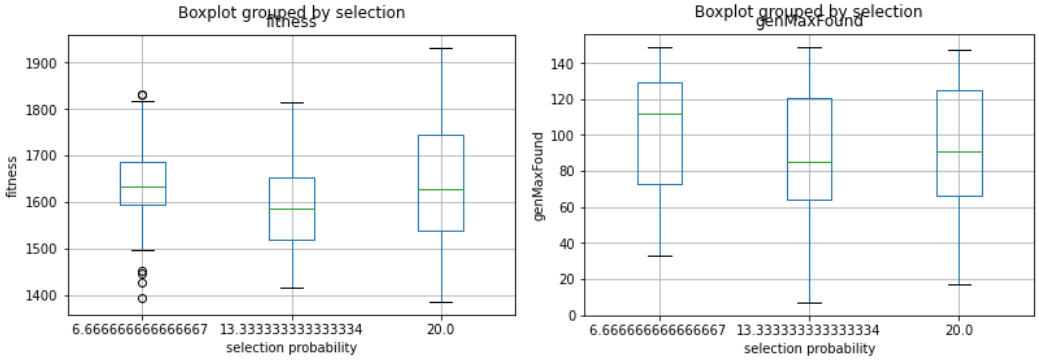
3.3 Changing the crossover probability has no effect on fitness and genMaxFound



This null hypothesis cannot be rejected for its effect on fitness as there is no significant difference for all crossover probabilities.
On the other hand, the null hypothesis for genMaxFound when crossover≈0.3 and crossover=1.0; crossover≈0.6 and crossover=1.0 can be rejected as it has an effect when the best individual is found during the runtime of the algorithm as having a mutation rate of 1.0 seems to be able to generate individuals with higher fitness early on in the runtime. However, it is not clear which mutation probability is best to use.

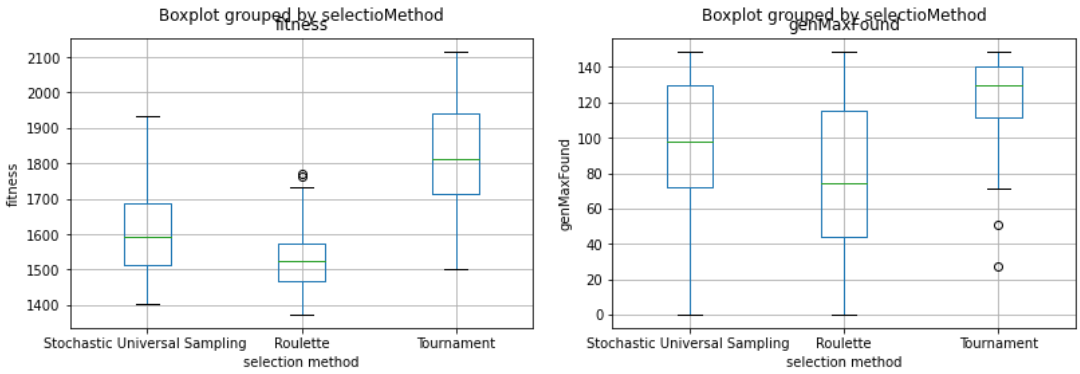
3.4 Changing the selection probability has no effect on fitness and genMaxFound

This changes the algorithm to a steady-state algorithm when selection!=20, as only the worst individuals are replaced rather than the whole population.



This null hypothesis can only be rejected for selection≈13.3 and selection=20 for fitness. Interestingly, selection≈6.7 looks similar to, but yet different than, selection≈13.3, yet the hypothesis cannot be rejected. When the selection pressure is 20 on a Stochastic Universal Sampling, it does not guarantee that it will not create individuals that break the constraint of the cost as all individuals are chosen for the repopulation (not by random), so while it generates good results, it is unknown what part of these results are below the cost of 100.

3.5 Changing the selection method has no effect on fitness and genMaxFound



The null hypothesis for all selection methods on fitness and genMaxFound is rejected. It can be seen that higher-fitness individuals are generated using the Tournament selection method, while the Roulette selection method generates individuals with lower fitness values. On the other hand, the algorithm finds these high-fitness individuals using the Tournament selection method much later in the lifetime of the algorithm. In contrast, Stochastic Universal Sampling and Roulette seem to generate individuals with high fitness values much earlier and continue throughout the life cycle of the algorithm. While it cannot be said which selection method is the best, it can be concluded that it affects the fitness and the genMaxFound.

4 Solution Quality

The best solution found consists of 1811 points and a cost of 100. [See Appendix]

5 Evaluation

Strengths: The algorithm creates feasible solutions but ignores the cost constraint. Additionally, the algorithm can find multiple optima's, giving the user a choice of feasible individuals depending on preference between cost and points. The algorithm can be reused with little changes, provided the problem is team formation and data can be arranged similarly. The algorithm can sustain diversity because of crossover and mutation operators.

Weaknesses: Solutions that break the cost constraint are generated and can replace solutions that do not. Hence solutions that do not break the cost constraint have to be saved additionally rather than in the Hall of Fame currently. The stochastic universal search reached higher results as it relies on exploration.

Future work: The algorithm could further be developed to generate or handle (repair) solutions that do not break the cost constraint. Additionally, only solutions that do not break the cost constraint can be chosen or accepted for/after operators or even kept N best individuals (cost <= 100) can be kept for the next generation. Another idea would be to populate the algorithm with already reasonable solutions, but this could decrease diversity depending on the number of individuals included and how different they are—modifying the crossover and mutation operator to act differently or choosing different methods on how they are applied overall. Another approach would be to extend the algorithm to a Multi-Objective Algorithm that handles multiple objectives, such as maximising the total cost while minimising the total cost of an individual, or a more straightforward approach to hybridising the algorithm with tabu search as repetition of individuals would minimise. Furthermore, Swarm algorithms have often been used to solve such issues [8]. Increasing the number of generations might also find solutions with higher solutions, as seen with the Tournament selection method

Overall, the algorithm provides feasible solutions, but further testing needs to be done to decide which is the best evaluation operator (that decreases the solutions that break the cost constraint) and selection method, including proportion, as seen in experiments. Even though the finesses reached in the experiments are high, they break the cost constraint by being between 101 to 110.

6 Appendix

```
total broken constraints: 0
total points: 1811.0
total cost is 100.0
selected players are [0, 4, 22, 48, 119, 180, 190, 218, 311, 376, 466]

(0, 1811.0)
```

References

[1] F. Rothlauf, *Representations for genetic and evolutionary algorithms*, 2nd ed. Heidelberg: Springer, 2006.

[2] M. Á. Pérez-Toledano, F. J. Rodriguez, J. García-Rubio, and S. J. Ibañez, "Players' selection for basketball teams, through Performance Index Rating, using multi-objective evolutionary algorithms," *PLoS ONE*, vol. 14, no. 9, p. e0221258, Sep. 2019, doi: 10.1371/journal.pone.0221258.

[3] S. N. Omkar and R. Verma, "CRICKET TEAM SELECTION USING GENETIC ALGORITHMc," p. 9.

[4] F. Ahmed, K. Deb, and A. Jindal, "Multi-objective optimisation and decision making approaches to cricket team selection," *Appl. Soft Comput.*, vol. 13, no. 1, pp. 402–414, Jan. 2013, doi: 10.1016/j.asoc.2012.07.031.

[5] H. Bederina and M. Hifi, "A hybrid multi-objective evolutionary algorithm for the team orienteering problem," in *2017 4th International Conference on Control, Decision and Information Technologies (CoDIT)*, Apr. 2017, pp. 0898–0903. doi: 10.1109/CoDIT.2017.8102710.

[6] V. Yannibelli and A. Amandi, "A deterministic crowding evolutionary algorithm to form learning teams in a collaborative learning context," *Expert Syst. Appl.*, vol. 39, no. 10, pp. 8584–8592, Aug. 2012, doi: 10.1016/j.eswa.2012.01.195.

[7] D. Costa, "A tabu search algorithm for computing an operational timetable," *Eur. J. Oper. Res.*, vol. 76, no. 1, pp. 98–110, Jul. 1994, doi: 10.1016/0377-2217(94)90009-4.

[8] O. Ramos-Figueroa, M. Quiroz-Castellanos, E. Mezura-Montes, and O. Schütze, "Metaheuristics to solve grouping problems: A review and a case study," *Swarm Evol. Comput.*, vol. 53, p. 100643, Mar. 2020, doi: 10.1016/j.swevo.2019.100643.

[9] S. Tkatek, S. Bahti, and J. Abouchabaka, "Artificial-intelligence-for-improving-the-optimization-of-nphard-problems-A-review2020International-Journal-of-Advanced-Trends-in-Computer-Science-and-Engineering," *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 9, pp. 7411–7420, Oct. 2020, doi: 10.30534/ijatcse/2020/73952020.

[10] N. Krasnogor and J. Smith, "A Memetic Algorithm With Self-Adaptive Local Search: TSP as a case study," *Proc. Genet. Evol. Comput. Conf. GECCO-2000*, May 2000.

[11] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. doi: 10.1007/978-3-662-44874-8.

[12] A. Yee, M. Alvarado, and G. Cocho, "Team formation and selection of strategies for computer simulations of baseball gaming," vol. 1, p. 15, 2016.

[13] P. C. Chu and J. E. Beasley, "A Genetic Algorithm for the Multidimensional Knapsack Problem," *J. Heuristics*, vol. 4, no. 1, pp. 63–86, Jun. 1998, doi: 10.1023/A:1009642405419.

[14] S. Khuri, T. Bäck, and J. Heitkötter, "The zero/one multiple knapsack problem and genetic algorithms," in *Proceedings of the 1994 ACM symposium on Applied computing - SAC '94*, Phoenix, Arizona, United States, 1994, pp. 188–193. doi: 10.1145/326619.326694.

[15] C. A. Coello Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art," *Comput. Methods Appl. Mech. Eng.*, vol. 191, no. 11, pp. 1245–1287, Jan. 2002, doi: 10.1016/S0045-7825(01)00323-1.

[16] G. F. Minetti, C. Salto, H. Alfonso, and R. H. Gallard, "A study of performance of stochastic universal sampling versus proportional selection on genetic algorithms," presented at the I Workshop de Investigadores en Ciencias de la Computación, 1999. Accessed: Nov. 09, 2022. [Online]. Available: http://sedici.unlp.edu.ar/handle/10915/22220

[17] R. P. Singh, "Solving 0–1 Knapsack problem using Genetic Algorithms," in *2011 IEEE 3rd International Conference on Communication Software and Networks*, May 2011, pp. 591–595. doi: 10.1109/ICCSN.2011.6013975.

[18] A. E. Eiben and J. E. Smith, "Representation, Mutation, and Recombination," in *Introduction to Evolutionary Computing*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 49–78. doi: 10.1007/978-3-662-44874-8_4.

https://link.springer.com/chapter/10.1007/978-3-662-47011-4_7