# Introduction

The challenge of finding the minimal solution piqued my curiosity and fueled my drive to discover such a minimum. At first look, processing all possible combinations appeared impracticable, but further analysis revealed data structure with minimal footprint allowing for exploration of entire search space. We discovered the three minimal solutions which satisfy the conditions of the problem statement where both groups have the same number of nodes, yet the bisection cost is minimal, and in this case, the absolute minimum – a bisection cost of 125 edges.

Implementing the GA was a straight forward exercise with the addition of the penalty terms to retard the selection of unfeasible solutions. Our GA was never able to yield the minimal bisection cost, but came really really close – 126 edges. After many failed attempts for the algorithm to bring the bisection cost below 140, adjusting the value of the coefficients of the penalty terms allowed it to find better minimums within a reasonable number of generations.

# The Best Solutions (Extra Credit)

There are three solutions which yield the best bisection cost of 125 listed below. We implemented an exhaustive algorithm which examined every possible *feasible* solution in the search space and discovered these absolute minimal solutions. It took about 10 hours to compute 137,846,528,820 [40!/(20! 20!)] solutions. The time could easily be brought down by reducing the search space to half of the number below since the same solution is discovered twice (when the set of nodes are in V1 in the first half of the search space vs in V2 in the second half of the search space). Even though further algorithmic optimization could yield even better time, this method quickly becomes untenable as the size of nodes doubles or triples. The GA algorithm came very close to this optimal solution with a much reasonable time of computation.

V1 = 3, 4, 5, 6, 7, 11, 16, 17, 21, 23, 25, 27, 29, 32, 33, 34, 35, 36, 37, 39

V2 = 1, 2, 8, 9, 10, 12, 13, 14, 15, 18, 19, 20, 22, 24, 26, 28, 30, 31, 38, 40

V1 = 2, 8, 9, 10, 12, 13, 14, 15, 18, 19, 20, 22, 24, 26, 30, 31, 33, 36, 38, 40

V2 = 1, 3, 4, 5, 6, 7, 11, 16, 17, 21, 23, 25, 27, 28, 29, 32, 34, 35, 37, 39

V1 = 2, 8, 9, 10, 12, 13, 14, 15, 18, 19, 20, 22, 24, 26, 28, 30, 31, 36, 38, 40

V2 = 1, 3, 4, 5, 6, 7, 11, 16, 17, 21, 23, 25, 27, 29, 32, 33, 34, 35, 37, 39

# GA Implementation

The algorithm used for this project is a standard GA with a constant population size of 200 using one point cross over as a recombination operator and bit-flip for a mutation operator. One thousand generations were generated for twenty runs where the best fitness of the runs is plotted. For the problem at hand, each solution (individual) has a genotype of a 40-bit string where the value of the bit (0 or 1) is a group identifier representing one of the two groups that contains a specific node (i.e., 0 for V1 and 1 for V2). On a 64-bit processing unit, the entire solution is contained within a single 64-bit integer and the entire population of 200 is represented by an array of 64-bit integers.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Node** | Unused | | | 40 | 39 | ∙∙∙ | 3 | 2 | 1 |
| **Bit** | 63 | 62 | ∙∙∙ | 39 | 38 | ∙∙∙ | 2 | 1 | 0 |

**Genotype**: lower 40 bits of a 64-bit integer (in gray)

The problem specifies a set of 40 nodes to be separated in two equal groups where each group contains 20 nodes while having the minimum number of edges between the two groups, *i.e., minimum bisection cost.* It is evident from the selection of a bit string representation for the genotype that there will be instances of the genotype that are not *feasible* solutions. Such instances will result from the initial random seeding of the population as well as the result of mutation or recombination. In the last project, we chose to correct for such inadequacies of the solution through a *repair* algorithm guaranteeing validity of individuals in the population at all times. For this project, we infuse the fitness function with *correction* terms, which retard the fitness of *unfeasible* solutions by a rate proportional to their distance from a feasible solution. The *B*∙ term of the fitness function shown below will reduce the fitness of the solution where *B* is a constant and is a function of the distance from a feasible solution.



Fitness is also reduced by the number of edges connecting the two groups as indicated by the *A*∙*c* term. The higher the number of edges connecting the two groups the lower the fitness; hence, solutions with lower number of connections have higher chance of selection for reproduction with the fitness proportional roulette-wheel selection algorithm.

The following parameters were used for implementation:

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| ***Number of runs*** | 1000 |
| ***Population size*** | 200 |
| ***|E|*** | 323 (from graph.txt) |
| ***A*** | 0.1 |
| ***B*** | 0.25 |
| ***Pm*** | 0.05 |
| ***Pr*** | 0.45 |
|  | |*NoOfBitsEqualOne*  - 20|2 |
| ***c*** | Bisection cost (Number of edges between two groups) |
| ***Mutation*** | Bit flip with Pm probablity |
| ***Recombination*** | One Point Crossover |
| ***Selection*** | Roulette Wheel |

Multiple runs of the GA yielded a solution of 126 bisection cost. One such solution is listed below:

V1 = 3, 4, 5, 6, 7, 11, 16, 17, 21, 23, 25, 27, 29, 31, 32, 33, 34, 35, 37, 39  
V2 = 1, 2, 8, 9, 10, 12, 13, 14, 15, 18, 19, 20, 22, 24, 26, 28, 30, 36, 38, 40

# Observation and Analysis:

The figures below show the Bisection cost and the corresponding fitness for the best of our runs for 1000 generations. The choice of our parameters which allowed our algorithm to discover our best GA solution resulted in small range, [308,310] of fitness values and yielded what appears to be sharp transitions. The Bisection cost graph is a bit smoother as the range of values, [126,155], is larger and the transitions are smaller.

For the sake of experimentation, we modified some of the parameters of the GA to understand the impact of such changes upon the speed and probability of convergence. With a B and A values higher than 0.75, the algorithm could not find bisection costs below 140 regardless of the number of runs we used. We also lowered the recombination probability from 0.6 to 0.45 as it yielded lower bisection costs than a value of 0.6 or higher. The mutation probability had a positive impact on the outcome where it allowed the algorithm to yield bisection costs below the 140 barrier mentioned above, but when we lowered the constants A and B and adjusted the recombination operator, Pm at 0.25 become the limiting factor. Finally lowering this parameter to 0.05 started to yield the best results reported herein.

As we compare the GA method to the exhaustive search method we realize the power of a GA for this class of problems. For a relatively small number of nodes, 40 in this case, the exhaustive search was able to find all perfect solutions within a tolerable amount of time; few hours. The GA, on the other hand, was able to find *close* solutions within a considerably time (once we tuned the parameters), few minutes. However, as the problem size increases, the GA becomes the only reasonable choice for solving such problems as the exhaustive search method quickly falls in the clutches of NP-completeness, and hence, impossible to compute. A GA on the other hand would be able to reach better minimums, and if it takes few hours of tuning the GA parameters, it still is better to find a minimal solution than none at all.