Group 15: Ryan Astor, Manjusha Chava, Cole Noreika

CS4341: Assignment 2 Genetic Algorithms

**Our Approach:**

Fitness Function: Our fitness function simply calculates the score that the particular organism whether a bin or a tower, and calculates its score and uses that number to determine fitness

Selection: To do selection, we calculate the total fitness of all members of a population and then sum them all together for a total fitness. Finally, we divide each single fitness by the total fitness to get a percentage it shares of the total space for the selection. In the case of bins where negative scores exist, we shift all numbers by the lowest negative plus a constant.

Child Generation: To make children, we can use elitism to save the best parents. For the most part, we use selection to make a probability selection for the parents and then use crossover to make many new children.

Crossover: To do crossover, we have one or more cutpoints to determine where an organism is split to create the children. This can either be swapping bins or swapping entire parts of a tower.

Mutation: To mutate, we roll a number to determine if a mutation would happen. If it will, then we randomly select two different parts of the organism, numbers is different bins or pieces in the tower, and then swap them

Population Size: We used a population size of 10

Example: For the example, we will have a small population of smaller than usual parents in the bin problem for simplicity

Parent 1:

Bin 1: -3.5, -7.2

Bin 2: 4.3 2.6

Bin 3: 9.8 2.3

Bin 4: -2.4, -5.9

Parent 2:

Bin 1: 9.8, 4.3

Bin 2: 2.3, 2.6

Bin 3: -3.5, -2.4

Bin 4:-7.2, -5.9

*Eval*

Parent 1 Fitness: (-3.5 \* -7.2) + (4.3 + 2.6) + (9.8 - 2.3) = 39.6

Parent 2 Fitness: (9.8 \* 4.3) + (2.3 + 2.6) + (-2.4 - -3.5) = 48.14

*Selection*

Total Fitness = 39.6 + 48.14 = 87.74

Parent 1 Selection Limit: 39.6 / 87.74 = 45.13% -> 0 <= Random Range <= 45.13%

Parent 2 Selection Limit: 48.14 / 87.74 = 54.87% -> 45.13% < Random Range <= 100%

*Generation*

In normal cases we use selection to find parents but as only two exist in this example, they are the parents and will be crossed over to do a generation. In this example, we spap bins 2 and 3 one-for-one.

Parent 1: Parent 2:

Bin 1: -3.5, -7.2 Bin 1: 9.8, 4.3

Bin 2: 4.3 2.6 ← → Bin 2: 2.3, 2.6

Bin 3: 9.8 2.3 ← → Bin 3: -3.5, -2.4

Bin 4: -2.4, -5.9 Bin 4:-7.2, -5.9

Then make new bins, however, duplicates exist as highlighted:

Child 1: Child 2:

Bin 1: -3.5, -7.2 Bin 1: 9.8, 4.3

Bin 2: 2.3, 2.6 Bin 2: 4.3 2.6

Bin 3: -3.5, -2.4 Bin 3: 9.8 2.3

Bin 4: -2.4, -5.9 Bin 4:-7.2, -5.9

To fix and make final children, we just swap the duplicates to make the children complete:

Child 1: Child 2:

Bin 1: 9.8, -7.2 Bin 1: -3.5, -2.4

Bin 2: 2.3, 2.6 Bin 2: 4.3 2.6

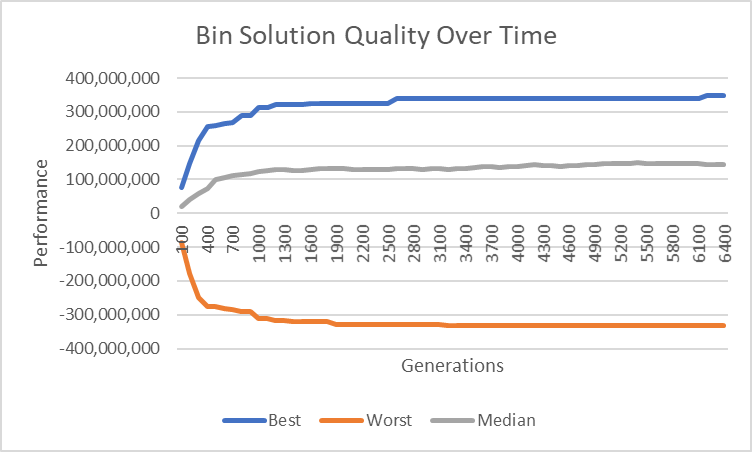
Bin 3: -3.5, 4.3 Bin 3: 9.8 2.3

Bin 4: -2.4, -5.9 Bin 4:-7.2, -5.9

Illegal Children from Crossovers: To prevent crossovers from leaving children with duplicate numbers in them, we had our program scan both children for duplicates. A dictionary is created with every key as the number and the value is one of its positions and how many times it occurs. If any are found to exist more than once, then we simply swap all duplicates found one at a time with the other child till all have been swapped back. We can easily do this as both children are guaranteed to have the same number of duplicates.

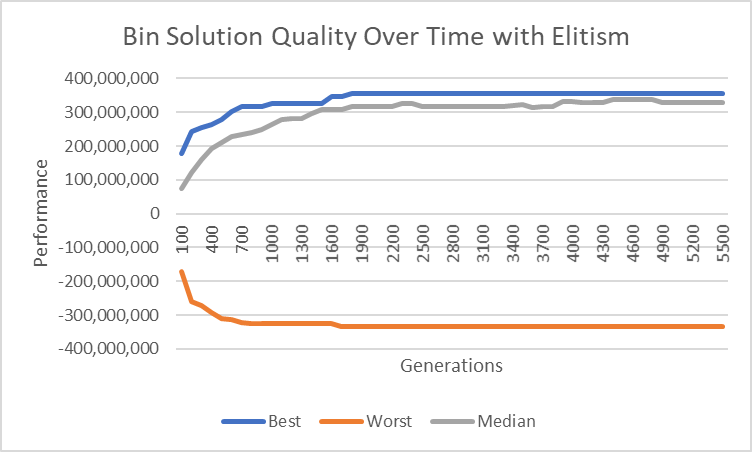
***Performance Results***

Number Allocation Over Generations:

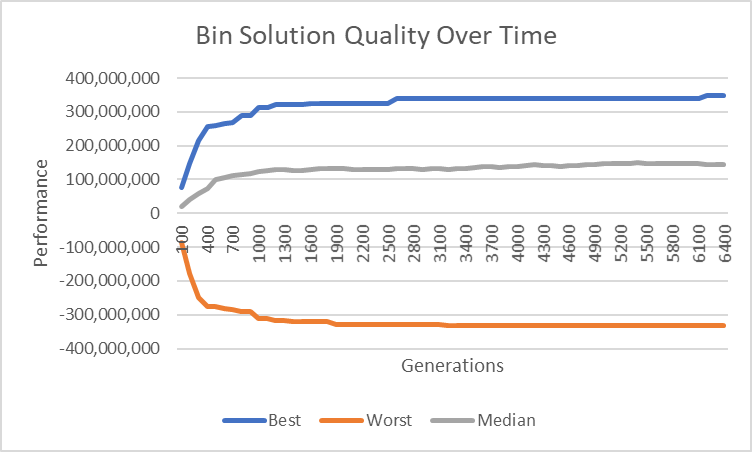


Tower Building Over Generations:

Number Allocation With Elitism:



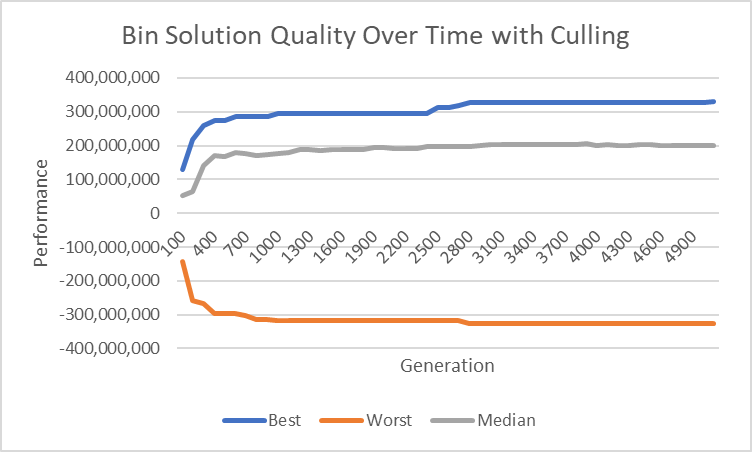
Number Allocation Without Elitism (Same as one above):



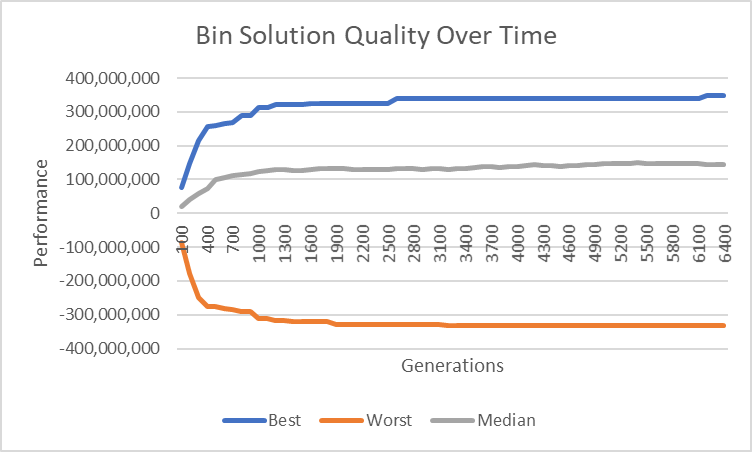
Tower Building With Elitism:

Tower Building Without Elitism:

Number Allocation With Culling:



Number Allocation Without Culling (Same as one above):



Tower Building With Culling:

Tower Building Without Culling:

How do Elitism and Culling Impact our Metrics: In terms of impact, both seem to have some effect. Elitism seems to make the best score converge faster even if it reaches the same level across enough generations, greatly increases the median score, and has no obvious impact on the worst score. Culling seems to reach the best solution level slower than without it, likely due to a smaller population size. However, the median value has a significant increase in performance while the worst case also seems not affected.

How we used Elitism and Culling: Our approaches to elitism and culling were fairly straight forward. For elitism, we simply assessed the scores of every parent and then selected the n best parents and copied them to the list to be filled with children without removing them from the parent pool. For culling, we also found the scores again, but then selected the n worst parents and removed them from the list of potential parents. We took this route as it seemed the easiest and most obvious way to implement these two concepts