**Problem 1:**

To determine the initial temperature, we take the first neighborhood and extract the max and min evaluations for its solutions. With this information we can then calculate a starting temperature that will select even the worst solutions with probability p (we decided to use .8). Note: We simply use algebraic manipulation of our probability function to solve for t. The logic for both functions can be seen below.

Text

Description automatically generated

Below you can see the logic for the main function. We use a min\_t as our stopping criteria which will be met once T has been “cooled” enough by our cooling schedule, a percentage we use to lower our temperature after M iterations in our nested while loop. Inside the nested while loop we have the logic to track our best answer, current answer, and accept a worse answer with probability P (above) based on the current temperature.

Also, below is a table of different problem instances and the variables/methods used.

Text

Description automatically generated

Blue = Control Row, Yellow = Changed Variable

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Initial T | Cooling | M | # of Temps | Iterations | Items Selected | Weight | Value |
| 81,462.4 | .8\*Tk-1 | 20 | 82 | 1,640 | 27 | 2,493.8 | 25,610.8 |
| 172,529.5 | .8\*Tk-1 | 20 | 85 | 1,700 | 22 | 2,498.5 | 21,274 |
| 81,462.4 | .8\*Tk-1 | 100 | 82 | 8,200 | 30 | 2,486.0 | 30,680.3 |
| 81,462.4 | .95\*Tk-1 | 20 | 356 | 7,120 | 26 | 2,497.8 | 19,613.9 |
| 81,462.4 | T0/(1+.9\*k) | 20 | 50,000 | 1,000,000 | 30 | 2,499.5 | 27,251.3 |

Note: For the final row our stopping criterion was changed to k >= 50000

Evaluation:

Text

Description automatically generated

**Problem 2:**

**Text

Description automatically generatedcreateChromosome()**

To generate chromosomes we pick a random int, i, to decide how many items to add to our knapsack. Then another random int, i times, that will be the indices we are adding to the knapsack.

**crossover()**

In our crossover function we “crossover” if a randomly generated int between 0 and 1 is less than our crossOverRate, set earlier in the code.

If crossover is performed we get a random index where we will “split” the parent chromosomes. Then each offspring gets a section from each parent, opposite of the other offspring.

Otherwise, we just return the parents.**Text

Description automatically generated**

**evaluate()**

In our evaluate function as long as a solution is feasible we return the total value. Otherwise, we take the maxWeight-totalWeight which will give us a negative value (which our maximize objective will want to avoid) but we also divide by the totalValue so higher value solutions will give us smaller negatives. i.e. If we have 2 solutions 1 pound over our max but one is worth 20,000 and another 200 the solution worth 20,000 is closer to a better feasible solution than the solution worth 200 and the 20,000 value solution will return a smaller negative value.

**Text

Description automatically generated**

**rouletteWheel()**

In the rouletteWheel function we select a mating pool from the current population. Our function gives a possibility of selection based on rank positions and allows repeats.

Text

Description automatically generated

**mutate()**

Our mutate function is a simplified version of the 1-flip neighborhood but instead of generating an entire neighborhood it selects a random index to flip before returning the mutated solution.

This all happens with probability = mutationRate, set earlier in the code.

Text

Description automatically generated

**insert()**

Our insert function generates a new population by selecting the best solution from the current population with probability p or otherwise selecting the best option from the generated offspring. This is continued until our new population is full.

Text

Description automatically generated

**Results:**

Blue = Control Row, Yellow = Changed Variable

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Generations | Pop Size | Crossover | Mutation | Elitism | # Items | Weight | Value |
| 100 | 150 | .8 | .05 | .1 | 28 | 2,493.7 | 33,055.6 |
| 1000 | 150 | .8 | .05 | .1 | 30 | 2,490.8 | 32,706.0 |
| 100 | 1000 | .8 | .05 | .1 | 30 | 2,495.1 | 36,082.8 |
| 100 | 150 | .95 | .05 | .1 | 30 | 2,496.2 | 32,849.9 |
| 100 | 150 | .8 | .15 | .1 | 27 | 2,469.3 | 32,351.6 |
| 100 | 150 | .8 | .05 | .01 | 29 | 2499.9 | 31,649.7 |
| 1000 | 1000 | .7 | .02 | .2 | 31 | 2471.2 | 36,374.8 |