|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Iterations | # Items Selected | Weight | Objective |
| Hill Climb – Best Improvement | 2,550 | 18 | 2,495.9 | 27,152 |
| Hill Climb – First Improvement | 395 | 20 | 2,497.9 | 13,814.2 |
| Hill Climb with Random Restart – Best Imp, k=1000 | 1,463,550 | 23 | 2,498.5 | 30,629.7 |
| Hill Climb with Random Walk – Best Imp, p=.8 | 3 | 2,400 | 17 | 2,497.2 |

**1a)** We know our knapsack’s only limitation is weight and our goal is to maximize value. So, a good strategy for determining an initial solution to the knapsack problem is adding items based off their *value to weight* ratio until our weight limit is reached. This will ensure we include the items that provide the most value for the weight they are taking up in the knapsack. (Logic pictured below)Text

Description automatically generated

Note: This will obviously not work for our random restart algorithm. Also, it resulted in most of our neighborhoods not being able to find a more optimized solution – resulting in uninteresting results (as far as seeing differences in the techniques are involved). So, we will be using the random initial solution for the given solutions. (Pictured below)

Text

Description automatically generated

Further, here are the results for all the problems (excluding random restart) with the ratio-based initial solution:

Iterations: 150

Items: 35

Value: 41,456.6

Weight: 2484.2

**1b)** Structure 1: 1-Flip neighborhood. This neighborhood is made up of n lists of size n where the ith value of the ith list is flipped from 1 to 0 or 0 to 1. It’s size is n^2.

Text

Description automatically generated

Structure 2: Swap neighborhood. This neighborhood structure is made up of solutions where for each item included in your current solution (“in” the knapsack) you swap it for each item “out” of the knapsack. This neighborhood will return k lists of n-k solutions of size k, where k is the number of items included in a solution. It’s size is k\*n\*(n-k)

Text

Description automatically generated

Structure 3: Double flip neighborhood. This neighborhood, similar to the 1-flip neighborhood, is made up of solutions where we swap both the ith and i+1 value for each index. It also returns n lists of size n so it’s size is n^2.

Text

Description automatically generated

**1c)** If a candidate solution is infeasible one solution would be to assign it a negative value, because our objective function is looking to maximize the value, it will avoid this solution.

Another solution could be to have an additional condition in our “if” statement that checks to see if we found a better solution. That is:

Change “if evaluate(s)[0] > f\_best[0]:” to “if evaluate(s)[0] > f\_best[0] and evaluate(s)[1]<=maxWeight”:

This change would prevent any infeasible solution from being set as our best solution.

**Question 2:**

To finish the given code, we needed to provide logic to generate an initial solution (Random Initial Solution pictured in 1a). We also needed to provide a way to deal with infeasible solutions. We decided to use the negative value method described in 1c. We use the 1-flip method to determine our neighborhood.

Text

Description automatically generated

**Question 3:**

To change from best improvement to first improvement we can simply add a “break” statement in our for loop that checks solutions in a returned neighborhood. This will result in no further solutions being evaluated in a given neighborhood once a single improvement is reached. We used the 1-Flip method to determine our neighborhood.

Text

Description automatically generated

**Question 4:**

For the random start we run the main loop k times, storing the best overall result in a variable outside of the loop. The current iteration is also pushed into our random initial solution equation to ensure we avoid the same starting spot due to the “pseudo-random” number generator. We used the 1-Flip method to determine our neighborhoods and the best improvement method to determine our next solution.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| K | Iterations | # Items Selected | Weight | Objective |
| 10 | 16,200 | 19 | 2,493.2 | 28,787.6 |
| 1000 | 1,463,550 | 23 | 2,498.5 | 30,629.7 |

Text

Description automatically generated

**Question 5:**

For the hill climb with random walk algorithm we set x\_curr and f\_curr (found using best improvement method) with probability p. Otherwise, we select a random solution from the neighborhood. This logic can be seen below in the “if/else” statements. We used 1-Flip method to determine our neighborhoods and used the best improvement method to determine our next solution.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| p | Walks | Iterations | # Items | Weight | Objective |
| .8 | 3 | 2,400 | 17 | 2,497.2 | 23,517.7 |
| .2 | 31 | 5,850 | 19 | 2,496.9 | 20,685.2 |

Text

Description automatically generated