DSA AAM Homework 6 Group 11

Alex Nozka

Marshall Baldwin

QUESTION 1:

Part a:

Method 1: RandomText

Description automatically generated

This method provides a random solution to the knapsack problem. For this method, it will randomly select items within the [0,n) distribution, and add them together. The function does this until the weight limit of the knapsack is met, where the initial solution will be returned. Logic in the code prevents duplicates of the same item from being used, and it only adds the item to the list if the addition of the item keeps the knapsack under its weight. This method will be useful for generating multiple random solutions, which support hill climbing methods like stochastic and random restart hill climbing.

Method 2: Max Value

Text

Description automatically generated

This method will take the highest value items from the list of highest value items and put them into the knapsack until the knapsack weight limit is met or exceeded, in which the list is returned. This method is not randomized as the first method is, however, this method provides more of a “first best guess” method for generating the solution. Might work best for hill climbing methods that do not require multiple random starting positions, like best choice, first choice, and random walk hill climbing.

Part b:

Method 1: j-shift

Text

Description automatically generated

This method is based fairly similarly to the first neighborhood design, the 1-flip method. In this method, rather than flipping the ith value, the goal is to shift the value to be the i+jth value. This again is designed to add a little randomness, but the j value is kept the same throughout the experiment so that all neighborhoods are uniformly constructed. The method loops, so if i+j >= n, then the value is looped around the beginning of the array. Similarly to the 1-flip neighborhood method that was constructed in the code base, the neighborhood size is n2 (or n n-sized neighbors).

Method 2: j-flips

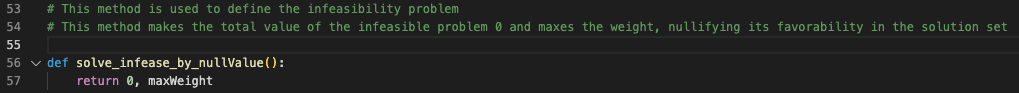
Text

Description automatically generated

This method is somewhat based on the previous neighborhood design, however the goal is to make more flips per neighbor, and for this we are flipping a j amount of items at random for each neighbor within the neighborhood. This neighborhood is also an n2 neighborhood, with n neighbors size n.

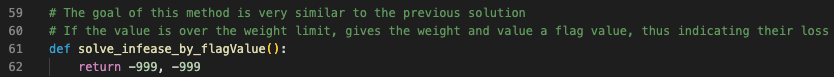
Part c:

Method 1: Null/0 value



This version of solving for the infeasible problem sets the value of the solution to 0, which will not be favored by the algorithm. Additionally, the infeasible solution is set to max weight, since this is an accepted weight value. So not only are we giving the algorithm 0 value but also max weight, which will disincentivize this solution within the search.

Method 2: Flag value



The goal of this version of solving the infeasible solution is very similar to the first, except instead is given a flag value for the both the value of the solution and the weight of the solution. The benefit of using this would be that if the solution is caught into the final solution that it would be very easy to identify that something went wrong in the search algorithm and that this solution is not viable.

Text

Description automatically generated

This subsection of code shows both methods in the evaluate function, under where the infeasible solution should be processed.

QUESTION 2:

A picture containing background pattern

Description automatically generated

The above screenshot displays the solutions for each of the four combinations for the initial solution development and the neighborhood strategies. The first two sets denote the random solution initialization with the j-flips neighborhood strategy on top and j-shift on the bottom. The bottom two follow this ordering of neighborhood strategies, except with the max value initialization. I noticed that the j-values are the same, despite being a random number. I attribute this to the use of the random seed, which I presume generates random numbers, however the first number will always be the same to keep consistency across initializations.

In this we see that the j-shift neighborhood outperformed the j-flips neighborhood in both initialization types. This was quite interesting to me, because I thought that a more randomized neighborhood would have given the algorithm more chances of seeing a better solution. The argument that I would probably make for this not being the case is that it is “harder” for the algorithm to “build” off of what it has “learned”, thus the more randomized neighborhood strategy would not allow for the model to make as many corrections without finding a local maximum in this neighborhood strategy. This was the case, as seen in the screenshot above, the heuristic only tested 150 neighbors in the j-flips neighborhood but made 300 in the j-shift neighborhood.

Background pattern

Description automatically generated with medium confidence

I ran both initialization strategies on the provided 1-flip neighborhood strategy and found the results. For this we see that the max value initialization local maximum was the same between the j-shift and 1-flip neighborhoods, however the random solution initialization was able to have more searches on the 1-flip neighborhood and was able to actually find a best solution out of all 6 runs of this scenario.

QUESTION 3:

Part a:

A picture containing background pattern

Description automatically generated

Using a similar format in the image above, we compared the different neighborhood strategies and initializations to determine the best solution set. In this image above, the first two solutions used the j-shift neighborhood strategy while using the random (top) and max value (bottom) initializations. The bottom two strategies both used the j-flips neighborhood strategy with the random (top) and max value (bottom) initializations. We see that using the first improvement approach, that our best answers were often able to go through many more improvements. It is important to note that for the previous best-improvement hill climbing algorithm, that it is very likely that for a given number of solutions checked the first-improvement algorithm has seen many more iterations. We know this because for the best-improvement hill climbing the number of iterations that the algorithm achieved was , where n=150. This means that the solutions checked would be 150, 300, 450, etc. for each 1,2,3,etc. iterations. In contrast, the first improvement could see any number of solutions between [1,n] in each iteration, which is how we can see numbers like 201 which are not factors of n.

What is interesting is that, unlike in problem 2, the j-flip method had very similar results that were seen by the j-shift neighborhood strategy, suggesting that this strategy is somewhat viable for the first-improvement hill climbing heuristic technique. We see that for both methods, the random initialization was more beneficial to the final answer than the max value initialization, which was not the case for the best ascent hill climbing technique.

For this section, we found that the best value achieved by the heuristic search is 14070.899 and weighed 2472.7 and carried 29 items.

Below I attached a screenshot of using the 1-flip neighborhood strategy, with either of the initialization strategies. We see similar results to that seen above. The value assessed was much greater than in the best-improvement hill climbing for both initialization types. Something that is interesting is that this neighborhood definition did worse than the j-shift neighborhood and performed very similarly to the j-flips neighborhood.

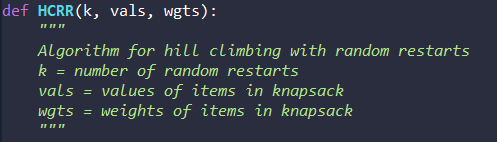
A picture containing background pattern

Description automatically generated

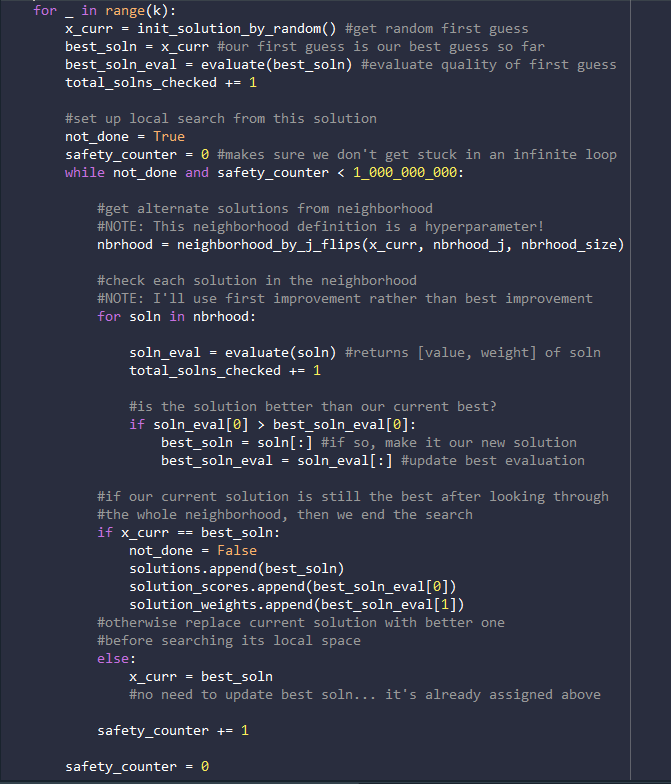
The difference between first-improvement and best improvement is implied in the name, but when the algorithm finds its first improvement, finds the first improvement possible then moves to the next iteration. For this, I put a break statement in the for loop that evaluates all the members of the neighborhood. When a solution in the neighborhood, the algorithm establishes this solution as the current best, and then the for-loop is broken and a new neighborhood is generated.

QUESTION 4:

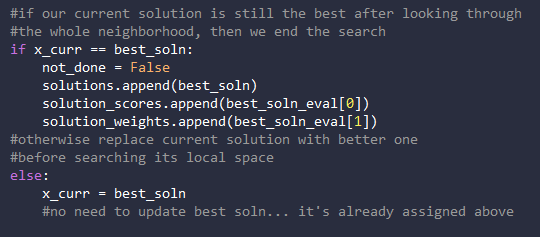
This question involves building a hill-climbing algorithm with random restarts. For this implementation in python, we created a function:



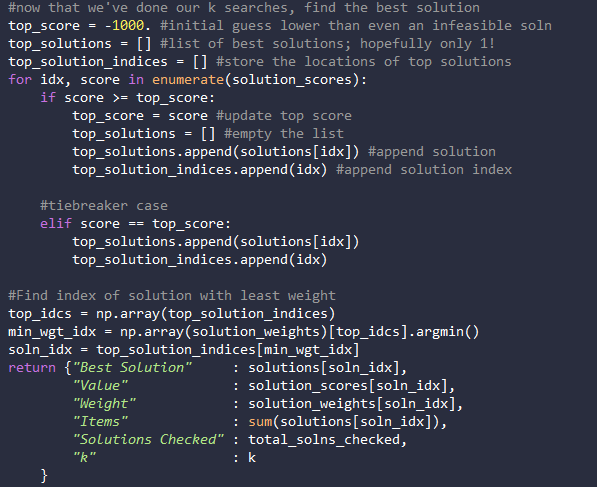
This implementation has three unstated assumptions in the docstring. First, that the 150 neighborhoods generated for each solution step come from the “j\_flips” generator, with j being equal to 18. Second, that if the solution does not converge after 1 billion steps, the hill climbing algorithm will halt and return the current solution. And third, that the algorithm will search each neighborhood in its entirety before choosing the best next move. The algorithm’s approach is to loop through the hill climbing algorithm k times, each time initializing a random initial guess. For each iteration, find the current solution’s neighborhood and check each solution to see if its evaluation score exceeds that of the current solutions’. This process is shown below.



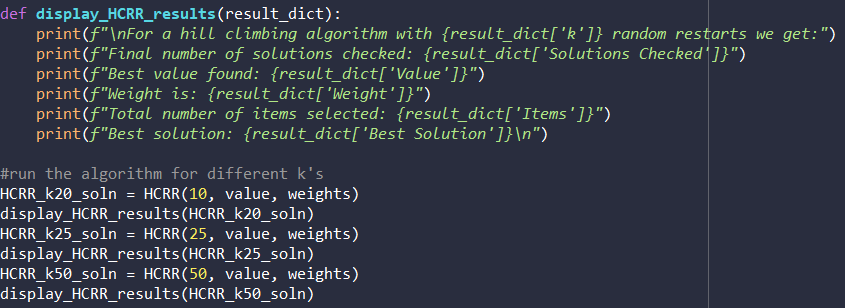
After this local search concludes, determine if the best solution is the same as the current one. If so, terminate the loop after keeping track of the best solution and its information. Otherwise set the best solution to be the current solution and then explore it’s subsequent neighborhood in the next iteration. This is shown the next block of code:



All that’s left then is to parse the solutions from each randomly initialized hill-climbing iteration by finding the highest-scoring one. If a tiebreaker is required (two solutions have the same value), then the one with the least weight is chosen. Finally the function returns a dictionary of the best overall solution and its corresponding information.



The function was then run for three volumes of random restarts as follows:



This outputted the following:

|  |
| --- |
| For a hill climbing algorithm with 10 random restarts we get:  Final number of solutions checked: 4210  Best value found: 14644.599999999999  Weight is: 2427.2  Total number of items selected: 30  Best solution: [0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0]  For a hill climbing algorithm with 25 random restarts we get:  Final number of solutions checked: 5725  Best value found: 15067.2  Weight is: 2473.4  Total number of items selected: 34  Best solution: [0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0]  For a hill climbing algorithm with 50 random restarts we get:  Final number of solutions checked: 11600  Best value found: 16830.9  Weight is: 2397.9  Total number of items selected: 32  Best solution: [0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1] |

As the number of random restarts increases, so does the best value! This corresponds to getting “lucky” with initial guesses that guide the algorithm to a higher local maximum. According to these heuristics, the best solution has a value of 16830.9, a weight of 2397.9, and selects 32 items.

A note: the subsequent algorithms in Q5 and Q6 use a very similar logical progression to this model. As such, I will not explain the models as thoroughly as this one; I will only make note of the main differences.

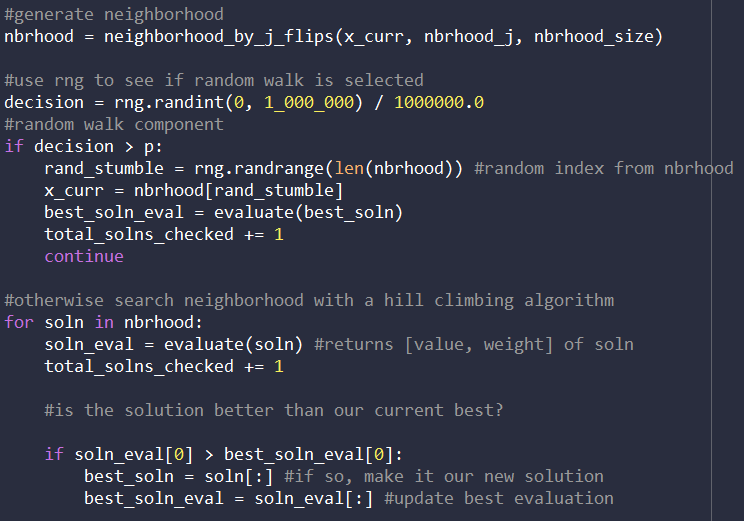
QUESTION 5:

This question involves building a hill-climbing algorithm with a random walk. For this implementation in python, we again created a function:

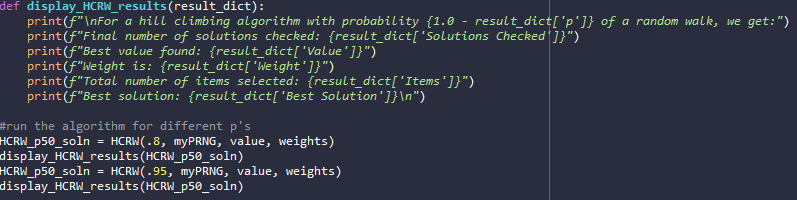
Text

Description automatically generated

This function only goes through a single iteration of hill-climbing, but it has an extra argument p. This is the probability that at a given step, the next move will be the best-scoring solution in the current solution’s neighborhood. Otherwise, the next move is chosen randomly (from a uniform distribution) out of all neighbors in the neighborhood (the random walk component). This is implemented by drawing a random integer in the range [0,1,000,000] and then dividing it by 1,000,000. If this value exceeds p, then the algorithm goes to a random walk. Otherwise, it’s hill climbing like usual. This is implemented as shown below. Note that for the random walk, feasibility of the selected neighbor is not taken into consideration. This is because we assume that the hill climber will immediately return to a feasible solution once found.



We then ran the following code which produced the following output:



|  |
| --- |
| For a hill climbing algorithm with probability 0.19999999999999996 of a random walk, we get:  Final number of solutions checked: 301  Best value found: 14644.599999999999  Weight is: 2427.2  Total number of items selected: 30  Best solution: [0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0]  For a hill climbing algorithm with probability 0.050000000000000044 of a random walk, we get:  Final number of solutions checked: 301  Best value found: 13808.7  Weight is: 2424.2000000000003  Total number of items selected: 32  Best solution: [0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0] |

In these two cases, the one that hill climbs 80% of the time outperformed the one that hill climbs only 50% of the time. They both underperformed compared to the random restart methods.

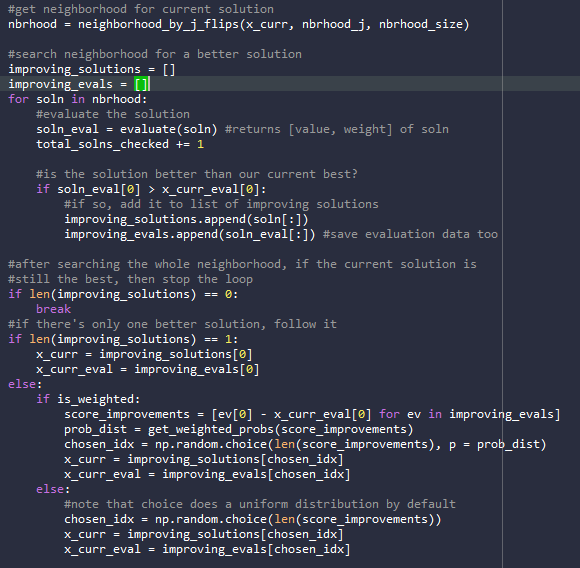
QUESTION 6:

This question involves building a stochastic hill-climbing algorithm. For this implementation in python, we again created a function:

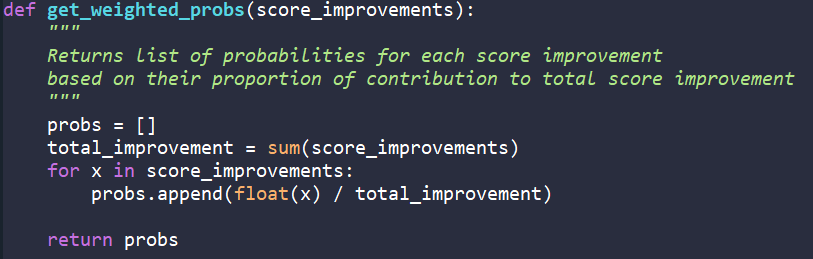
Text

Description automatically generated

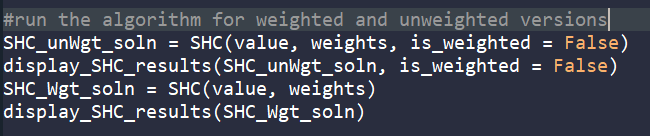
This function again goes through a single iteration of the algorithm (terminating at a local maximum). The wrinkle this time is that the function collects all improving solutions from a current solution’s neighborhood and chooses randomly from this list to make the next move. This is implemented as follows:



There are two probability distributions that the solutions are picked from. The first is an unweighted, uniform distribution. The second is calculated based on the proportion of total score improvement generated by a solution. For example, if three solutions gave improvements over the current solution’s of 100, 75, and 25 respectively, then they would have probabilities of 50%, 37.5%, and 12.5% respectively. This is implemented as a helper function:



Finally, running the following code gives the following output:



|  |
| --- |
| For a stochastic hill climbing algorithm with a uniform choice distribution, we get:  Final number of solutions checked: 301  Best value found: 14644.599999999999  Weight is: 2427.2  Total number of items selected: 30  Best solution: [0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0]  For a stochastic hill climbing algorithm with a weighted choice distribution, we get:  Final number of solutions checked: 751  Best value found: 14718.9  Weight is: 2448.8999999999996  Total number of items selected: 33  Best solution: [0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1] |

The two runs resulted in similar solutions and slightly outperformed the random walker. However, I believe the success of the random resets algorithm speaks to the sensitivity to initial condition that hill climbing algorithms face with this formulation of the problem. Using some human ingenuity by noticing that items with a high value to weight ratio, we manually found a solution of over 24,000.