**Implement a random solution to the Knapsack problem:**

You must use the Item class that we created in class.

Create a Chromosome class in python

* Write \_\_init\_\_()

The \_\_init\_\_() method will need to take a list of Items and a maxCapacity for your knapsack.

Other instance variables that will be helpful would be a list to represent what is “packed”, the value of what is “packed”, and the weight of what is packed.

In this method, randomly “pack” items. To do this, create a list that is the length of how many items you have, and have each index randomly assigned zero or one. One signifies packed, zero signifies not packed. For example if you were given a list of items A,B,C, D, E with the following properties:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Item** | **A** | **B** | **C** | **D** | **E** |
| **Value** | 3 | 5 | 6 | 8 | 10 |
| **weight** | 2 | 4 | 5 | 7 | 9 |

The list [0, 1, 1, 0, 1] would signify that item B, C, and E were packed.

Based on what is “packed” you can find the total weight of the chromosome (weight of all the items packed that are represented by ones), and the total value of the chromosome (value of all the items packed that are represented by zeros). In the above example, the total weight would be 4+5+9 = 18 and the total value would be 5+6+10 = 21.

You should guarantee the chromosome you made does not exceed the maxCapacity of the knapsack. If it does, then remove items until it complies. Make sure that you adjust the weight and value as you take items out.

* Write \_\_comp\_\_(), \_\_lt\_\_(), \_\_gt\_\_(), and \_\_eq\_\_() using the value of the packed items to determine ranking. Example, if this chromosome’s value is less than the other, then this is less than the other.
* Write \_\_repr\_\_() to return a string that is informative to you (prints the chromosome list, its weight, its value)
* Write a method that returns a list of the Items that are packed based on the list representation of what items are packed.
* Write a crossover method that accepts another chromosome and a crossover rate.

In this method, randomly generate a floating-point number between 0 and 1, and if that number is less than the given crossover rate –it will randomly pick a position in the list representation of your chromosome to “slice” this chromosome and the other chromosome, then it will swap the tails.

Example: if the random position is 3.

This chromosome = [1, 0, 1, 1, 0]

Other chromosome = [0, 0, 0, 0, 1]

Would alter the chromosomes to this:

This chromosome = [1, 0, 1, 0, 1]

Other chromosome = [0, 0, 0, 1, 0]

Remember to recalculate the weights and values of the chromosomes after swapping tails.

* Write a point mutation method that accepts a mutation rate, and it will randomly flip bits.

This method will walk the chromosome list. It will randomly generate a floating-point number between 0 and 1, and if the number is less than the given mutation rate, it will flip that bit.

Remember to recalculate the weights and values after flipping the bits.

* You can write any other methods you want.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Create a python file called **geneticKnapSack.py**

Write code that will solve the knapsack problem. I want to be able to type in the command prompt:

>python **geneticKnapSack.py**

And have it run.

For this version, select a crossover rate of 0.7 and a mutation rate of 0.001.

Use the same list of items that we have been using:

stuff =[Item("A", 3,2),Item("B", 5,4), Item("C", 6,5), Item("D", 8,7), Item("E", 10,9)]

Create list of 10 chromosomes. Ten is going to be the population size. The population size needs to be even.

Sort your population. You want to sort so that the highest fitness values are first.

We are arbitrarily going to run this for 50 generations (this can be altered).

For each generations cycle, pair off the sorted chromosomes in the population so chromosomes and index 0 and index 1 will “mate”, chromosomes and index 2 and index 3 will “mate”, chromosomes and index 4 and index 5 will “mate”, chromosomes and index 6 and index 7 will “mate”, and chromosomes and index 8 and index 9 will “mate”.

During mating, you will make copies of the parents, these copies are the children and will undergo crossover and mutation based on the crossover and mutation rates. You will need to adjust the children to make sure they do not go over the maxCapacity of the knapsack.

Once all the pairs in the population have “mated”, you will select the top 10 of the best candidates. These candidates will be selected from a pool consisting of the original parents and all the children that they spawned through mating. These top ten will then be used to seed the next generation.

When you have gone through all 50 generations, the best solution will be at index zero of your population list of chromosomes.

Return the best chromosome and report what items were packed along with its value and weight.

Run the algorithm 10 times and record the best chromosomes that were returned. Calculate an average and standard deviation.

|  |  |
| --- | --- |
| **Run with 50 generations** | **Best value returned** |
| 1 | 24 |
| 2 | 24 |
| 3 | 23 |
| 4 | 23 |
| 5 | 24 |
| 6 | 24 |
| 7 | 24 |
| 8 | 24 |
| 9 | 24 |
| 10 | 24 |
| **Average** | 23.8 |
| **Standard Deviation** | 0.4 |

Reduce the number of generations to 5. Run the algorithm 10 times and record the best chromosomes returned. . Calculate an average and standard deviation.

|  |  |
| --- | --- |
| **Run with 5 generations** | **Best value returned** |
| 1 | 23 |
| 2 | 23 |
| 3 | 24 |
| 4 | 24 |
| 5 | 24 |
| 6 | 22 |
| 7 | 23 |
| 8 | 21 |
| 9 | 18 |
| 10 | 23 |
| **Average** | 22.5 |
| **Standard Deviation** | 1.7464249 |

Increase the number of generations to 500. Run the algorithm 10 times and record the best chromosomes returned. . Calculate an average and standard deviation.

|  |  |
| --- | --- |
| **Run with 500 generations** | **Best value returned** |
| 1 | 23 |
| 2 | 24 |
| 3 | 24 |
| 4 | 23 |
| 5 | 23 |
| 6 | 23 |
| 7 | 24 |
| 8 | 23 |
| 9 | 23 |
| 10 | 24 |
| **Average** | 23.4 |
| **Standard Deviation** | 0.48989795 |

Answer these questions:

Is there any correlation between the number of generations you ran and the best value returned?

Yes, up to a point (500 was no better than 50). As the number of generations went up, then the best value returned got closer and closer to being the actual value. Also, with a higher number of generation the standard deviation and variance are lower, meaning you are more likely to get the same results between runs.

Alter the mutation rate and crossover rate. What impact does this have on your results?

More generations increase the chance that we will eventually get the fittest (best possible) child. Which will give us the exact answer to our question.