CS 170 Final Project Write-up

Tammy Vu, Dennis Yang

Phase I: Input Files

For our 3 Phase I input files, we wrote a Python script to randomly generate large input files. We let P and M be the largest value possible (2^32 - 1) because larger constraining values allows for more freedom in choosing solution options. For N, we selected a random number between 15,000 and 20,000 because after trial and error, we found that numbers near this range allowed for files to be close to the max file size allowed of 4 MB. For C, we found pairs of C and average constraint lengths that get the input file close to 4 MB. The number of constraints were around 1000, and constraint lengths were around 450. For each item, we randomly choose item names, class, price, weight, and resale values. Similarly, for the constraints, we randomly generated lists of class numbers.

Phase II: Code

The basis of our Phase II algorithm will be greedy based on a ratio of each item's resale_value / (cost * weight) ratio. This ratio makes sense because you want the greatest proportion of how much you can sell an item for depending on how much you bought it for, while also wanting it to take up as little space as possible. To do this, we first iterate through all the items, and calculate it based on each item's characteristics.

From here, we begin our greedy algorithm by looking at the first two items with the highest value/weight ratio, unless there is only one left for consideration. Between these two items, we will randomly pick one. We introduce randomization here because as discussed in Chapter 9, it is an effective strategy to picking local optimas. For each item chosen, we check if its already been constrained by a previously bought item; if not, then we buy it, and adjust our bought classes, weight and money left. Repeat until there is no more money, weight, or items left.

One strategy we tried was using the concept of opportunity cost, or the loss in potential gain from possible alternatives. To calculate the opportunity cost for each class, we add the value/weight ratio for each item that belongs to that class. Then, we will iterate through each constraint and find all the incompatible classes for each class, and add their respective total ratio costs - this combined value for each class is its opportunity cost. Essentially what this value represents is: say you pick item *i* from class *c*, then you cannot pick any item that belongs to any incompatible class for class *i*, and thus are losing on their resale opportunity. For two items with similar value/weight ratios, we obviously want to pick the item with the lower opportunity cost. During the randomization step, randomly pick one based on a proportion of their opportunity cost, with the one with the lower opportunity cost being more likely to be chosen. However, implementation of this strategy resulted in a slightly lower score, but it is definitely something we want to explore in the future.

Runtime Analysis:

N - number of items
C - number of Constraints
L – average length of each constraint

calculating value/weight ratios - O(N)
greedily choosing each item, checking constraints for each one - O(CNL)