Hao Zhe Kok

Mr. Pettit

CS 467.01

2/2/2015

Project 1 Reflection

Project 1 (affectionately named “Knapsack Thief” by this author), is the implementation of various greedy and brute-force algorithms to solve the Knapsack Problem. Given a list of items and their respective costs and value, as well as a cost limitation, the program would return a list of items with maximum value which does not violate the cost limit.

In the first phase of implementation, we were tasked with implementing the decisions to take each item and placing it into the “knapsack” as a binary tree. Using Python (mainly due to its easy-to-work-with csv reading and writing library), I quickly came up with an implementation which stored the entire tree in memory and traversed each branch until its leaf nodes, which contained a list of every possible combination of items. The node would then be evaluated to see if it violated the cost limit, and compared with the list with the greatest value, replacing it if it was greater.

However, this brute-force and “dumb” approach led to the creation and evaluation of 2^n leaf nodes, which quickly became a problem when dealing with datasets slightly greater than 10 items, especially on more underpowered hardware.

Phase Two of our project involved sorting the set of items by three criteria (each having a time-efficiency of only n log n), best value, lowest cost and best value to cost ratio. Items were then added into the knapsack from these modified lists until the cost limit is violated. Compared to the brute-force method (and also to each other), the results of each

greedy approach were evaluated and we were able to determine upper and lower bounds for our solution.

Still, the problem of running out of memory persisted, and after consulting with Mr.Pettit, I rewrote my program so that it performed a depth-first search of the “tree” instead of loading it into memory before evaluation. This drastically cut memory usage and I was able to run the optimizations on our datasets. They involved ending the search of the tree “branch” as soon as the cost limit was exceeded, as well as keeping a “lower bound” of potential total value considering the values of items left untaken, and ending the search when the lower bound went lower than that of the best greedy algorithm.

Throughout the project I’ve appreciated the importance of optimization. Usually with the programs and projects I’ve worked on in the past, the nature of the problems have never required “smart” algorithms to solve. Therefore, I never had to think too much about optimization and memory constraints. I was surprised with how drastic the algorithm improved with the introduction of two relatively simple optimizations (less than 10 lines of code each).

All in all, I have managed to solve the problems presented and optimizing it adequately. I appreciated the impact the optimizations provided and learned a lot not only on the importance of efficient algorithms, but also depth-first traversal.