Alicia Vargas-Morawetz  
CPSC 490  
Senior Project Proposal  
Advisor: Professor Angluin

**Knapsack Problem with Category Constraint**

The Knapsack Problem is a well-known problem within the Computer Science field. It involves selecting items from a list which have all been prescribed a given weight and value. The solution to the problem would be the selection of items with the highest cumulative value, but whose cumulative weight is below an upper bound. One way to think about the problem is as a collection of objects with values that you are trying to fit into a knapsack that can only hold up to a certain weight. This problem can be extended to many different situations such as packing things in a suitcase, buying items on a budget, or practically any type of resource allocation problem.

The Knapsack problem has been studied extensively over the years. One important book on the subject, titled *Knapsack Problems* by Hans Kellerer, Ulrich Pferschy, and David Pisinger, delves into the complexities of the problem, as well as multiple algorithms – some approximate and some more precise – that can be used to solve it (e.g., greedy algorithms, dynamic programming). In addition the book examines extensions of the Knapsack Problem such as a situation in which there are multiple knapsacks and items can only be placed in certain knapsacks.\(^1\) Other study of added constraints to this problem includes the Multi-Dimensional Knapsack Problem where there other factors in addition to weight, like volume, that have to be considered when trying to fit items into the knapsack.\(^2\) In addition, the multi-dimensional problem has been studied as it specifically applies to grid resource

---

scheduling of applications running on the same machine (in this paper it is titled the “Temporal Knapsack Problem”).³

A typical way to solve this problem when given a set of items, with their values and weights, and a threshold capacity for the knapsack is to use 2D dynamic programming. In this way you can find the trivial solution for when you have just the first item and then continually add the other items, while re-computing the solution after each item is added. However, this algorithm does not take into account what the final collection of items will be. In real-life applications of this problem it is possible to imagine other kinds of constraints on the solution besides the weight threshold. For instance, if you were packing a suitcase and needed to fit in items in order to get maximum value, it could happen that all the most-valued, but least-weight items would be shirts, so the end result would be a collection of shirts, which does not make sense if you’re trying to pack a full wardrobe for a trip. Thus we can imagine a category constraint on the solution to the Knapsack problem. If each item in the list is given a category (e.g., shirts, pants, socks, etc.) then you could add a requirement that the solution contain approximately equal numbers of each type of item. The way you restrict the categories could vary, but the basic idea is to require that the solution to the problem has some amount of items in each category, but also fits into the weight threshold and has the highest value possible (given the constraints).

The example I imagine using for this problem is when someone is trying to come up with a reading list that they can complete over a limited time frame, such as a vacation. The inputs would be the amount of time they have to read (the total number of days) and a list of books. Each book would have a weight (the number of days it would likely take to read it) and a value (the popularity of the book). In addition, there could be a category constraint on genre if the person wants to make sure they read a diverse array of books. I will use this example to illustrate what extra considerations have to be made when a category restraint is added to the

---
Knapsack problem. I will compare different ways the solution could be modified and evaluate their space requirements and their runtimes. In addition I will run my different solutions on sets of example data in order to see if they perform as expected.

One idea I have for modification of the dynamic programming algorithms that solves the Knapsack Problem simply checks at each step if a given possible solution meets the category restraint, and discards it if it does not. Another idea I have requires splitting up the items into their categories and building up solutions within each category that can be combined together at the end. This strategy could potentially be applied to both dynamic programming algorithms as well as greedy algorithms to solve the knapsack problem. I aim to test the performance of these different modified algorithms in order to evaluate them against each other generally. In addition, I will try to determine if certain algorithms are better suited for certain category situations (e.g., many categories, few categories, disproportionate category requirements).

From this senior project I hope to learn what actually goes into implementing a dynamic programming algorithm that I have learned about in a theoretical sense in the past. This project would also require me to think creatively about different ways a given algorithm can be modified to work when it is given an additional constraint. Additionally, I would try to figure out the best way to generate random data in order to test my proposed solutions to the problem. This project would consist of both theoretical problem solving and actual implementation, and would allow me to learn about the most important considerations that should be made when working with these two sides of Computer Science together.

**Deliverables:**

- Code written to implement and test the algorithms
- Paper describing and analyzing the results
References:

