**Solution to the Cloth-Cutting Problem**

Connor Johnson

October 2018

**Introduction**

The cloth-cutting problem is an optimization problem based off the Knapsack Problem. It is as follows:

You are given a rectangular piece of cloth with dimensions X by Y, where X and Y are positive integers. You are also given a list of n products to be made. Each product has dimensions ai, bi and a selling price of vi. You have a machine that can cut the cloth horizontally and vertically. Write an algorithm that finds the group of cuts that creates a group of products worth the maximum amount of money.

Like the Knapsack Problem, a brute force solution of all possible cuts is very unreasonable. The runtime of that algorithm would be at least (XY)! which is way too slow. Instead, we will use dynamic programming. Since this problem has many similarities to the Knapsack problem, we will look at the solution to that to reach our answer. That solution is the formula below:

We must change this equation to make it work with the cloth-cutting problem. To start off, the “capacity” of the cloth is based on two variables X and Y. *Optimal(M)* changes to *Optimal(i, j).* Instead of removing and adding objects as we increase the capacity like in the Knapsack Problem, we will keep finding the optimal value after making a cut. The location of the cut is represented by 1 ≤ i < X, k is an integer. Remember, cuts can be made horizontally and vertically so we will represent a horizontal cut with 1 ≤ j < Y, j is an integer. We must find the max value out of all cuts 1 ≤ i < X and then cuts 1 ≤ j < Y. Then, finding the max value out of those two maximums will give you the most optimal cut given a cloth of size x by Y. So the optimal value equation with this information is (Optimal(X,Y) is represented by C(X,Y):

This equation represents the optimal value from cutting up the cloth but it is missing the value of each sized product. We will represent this as prod(X,Y). The function prod(X,Y) is equal to the value of a piece of cloth with dimensions X by Y. The values are the n products given in the problem. If the value of X and Y do not equal any of the products, the value of prod(X,Y) is zero. With this, the final equation is:

We add prod(X,Y) to the max function because if the value of the current cloth size is higher than any value that could be gotten by continuing to cut it, we will use the current size in the final product. Now that the formula for this problem is found, we can use this in the solutions.

There are two solutions for the problem given the formula we have above. Both solutions are shown below.

**Recursive**

Optimal(x,y) {

max = 0

for i = 1 . . . x {

if (Optimal(i,y) + Optimal(x-i,y) > max){

max = Optimal(i,y) + Optimal(x-i,y) > max

}

for j = 1 . . . x {

if (Optimal(x,j) + Optimal(x,y-j) > max){

max = Optimal(x,j) + Optimal(x,y-j) > max

}

if(rect(X,Y) > max){

max = rect(X,Y);

return max

}

This algorithm recursively goes through every combination of cuts to find the optimal one. The problem with this is that it has an exponential runtime. Throughout its runtime, it continually calculates the same value repeatedly. The base case can be found in the last if statement. Once a product is found, it is returned if it is worth more than continually cutting the cloth. If the cloth is continually cut and never finds a product, the base case is hidden. It is found in max = 0. Once the cloth is 1 by 1, neither of the for loops will be used and the value of max = 0 will be returned.

**Memoized**

Optimal(c) {

knowni,j = 0 for 1 ≤ i ≤ X, 1 ≤ j ≤ Y

return Optimal(c, known)

}

Optimal(x,y, known) {

if knownx,y ≠ 0 return knownx,y

max = 0

for i = 1 . . . x {

if (Optimal(i,y, known) + Optimal(x-i,y, known) > max){

max = Optimal(i,y, known) + Optimal(x-i,y, known)

}

for j = 1 . . . y {

if (Optimal(x,j, known) + Optimal(x,y-j, known) > max){

max = Optimal(x,j, known) + Optimal(x,y-j, known)

}

if(rect(X,Y) > max){

max = rect(X,Y)

return max

}

This solution keeps a memo of all the found values of certain sized cloths. This prevents the algorithm from solving for the same one twice. Adding this memo to the recursive function reduces its runtime from exponential to a polynomial time. Because of this, this is the solution I used in my solution for the problem. I will further explain my Java implementation of the problem below.

**Implementation**

My implementation of the algorithm is almost identical to the solution described above. The main difference that should be noticed is that my optimize function does not return an integer. In my implementation, I had to solve for all the garments that would be made and the location of all the cuts. I wasn’t just solving for the potential value of the full cloth. To keep track of all the cuts and garments I made, I created a class called ClothPiece. ClothPiece represents a piece of cloth with cuts and garments(a garment is a product piece with a location value). The cuts, garments, size and value of the piece are all stored in the class. By passing this variable as the return, it made it easier for the cuts and garments to be kept track of as I traversed all the possible cut formations. Instead of just returning the optimal value of a cloth of size X by Y, my optimize function returned a piece of cloth that had information on all the cuts and garments were made to find the optimal value. This improved the usefulness of the function and made it easier to solve the overall problem. I also declared the memo as a variable of the class and not a parameter of the optimize method. This is more of a personal preference as both ways are similar in efficiency. Overall, I made a few slight changes to the memorized algorithm to work with the problem I had at hand. I also created classes to store data which are defined below

* Pattern – this represents one of the possible products and its value
* Garment – this stores a cloth that is one of the products along with its location on the cloth
* Cut- this stores the beginning and end X,Y values of a cut
* ClothPiece – this stores all the garments and cuts that are put together to create a piece of cloth that is size X,Y and has a specific value