CSE431: HOMEWORK 3

BRECKEN MCGEOUGH

Problem 1. *My problem (based off of fractional knapsack problem): Given stocks where some kth stock has a potential profit* $\theta_k \in \mathbb{R}$*, a volatility measure* $\beta_k \in \mathbb{R}^+$*, and a price* $p_k \in \mathbb{R}$*. When choosing stocks, we are able to buy a fraction of a stock for a fraction of its price, but also for a fraction of the profit, while the risk remains the same. The constraints are:*

Maximize $\sum_{i=0}^{n-1} x_i\theta_i$ *where* $x_k \in [0,1]$ *and* $n = |stocks|$

while keeping $\sum_{i=0}^{n-1} x_i p_i \leq P \wedge \sum_{i=0}^{n-1} \beta_i \leq R$ *where* P *is the maximum amount of money we can spend on stocks and* R *is the total risk we are willing to take onto our portfolio. We will start by sorting all of the stocks by their profit/price ratio from greatest to least.*

Given the set of stocks $S = \{s_0, s_1, ..., s_{n-1}\}$ *where each stock* s_k *is a 3-tuple* (θ_k, β_k, p_k)

The sorted list $S' = \{\forall_{k\in[0:n]} s_k \mid \theta_k/p_k > \theta_{k+1}/p_{k+1}\}$

Then we will iterate through the sorted list, buying as much of a stock as possible while still obeying the constraints until we've either used all our money or reached our limit for volatility and risk. Our algorithm will buy as much of a stock as possible while obeying the constraints in order to maximize the profits, since we sorted the stocks by their profit/price ratio.

Pseudocode:

```
procedure greedy_stocks(Stocks[0:n],P,R):
    S'<- set of stocks Stocks[0:n] sorted in decreasing order by their theta/price ratio

    sumP = 0
    sumR = 0
```

```
    total = 0


    for i in S':
        fraction = 1
        if sumP+price_i > P or sumR+beta_i > R: (negation of (sumP <= P and sumR <= R) by deM
            fraction = (P-sumP)/price_i


        total = total + (theta_i * fraction)
        sumP = sumP + (price_i * fraction)
        sumR = sumR + beta_i


    return total
```

In order to derive the fraction to make the amount we buy plus sumP equal P:

if $sumP + p_i > P \Rightarrow sumP + x_i p_i = P \Rightarrow x_i p_i = P - sumP \Rightarrow x_i = (P - sumP)/p_i, x_i \in [0, 1]$

Theorem 1. *Prove the Greedy Choice Property: Globally optimal solution is produced by making locally optimal choice for each step.*

Proof. Let k be the index of the element that has the highest ratio ($k = 0$ since we sorted the stocks by ratio in decreasing order), θ_k/p_k. By the definition of our algorithm, we will buy as much of the kth stock as possible. Let $A = \{a_0, a_1, ..., a_{n-1}\}$ be the set of fractions of the amount of a stock we buy for all stocks for our algorithm, where a_0 is the first greedy choice made by our algorithm as it is the highest possible fraction of the kth stock price that can be taken. Assume our solution is not the optimal solution. Then assume there exists an optimal solution that does not buy as much of stock k as possible. The set of fractions of the optimal solution is $O = \{o_0, o_1, ..., o_{n-1}\}$. Therefore, $o_0 < a_0$ (the first element in all of the solutions fraction set maps to the largest ratio since we sorted the elements by their ratios in

decreasing order, a_0, o_0 maps to the highest ratio, a_1, o_1 maps to the second highest ratio, etc.) since our algorithm buys more of the stock with the highest ratio θ_k/p_k than the optimal solution. There exists an element with index l with the second largest ratio ($l = 1$) and thus $\theta_l/p_l < \theta_k/p_k$, therefore we can create a new solution O' such that $O' = \{\forall_{m \in |O|} o_m \mid m \neq 0 \wedge m \neq 1\} = \{o'_0, o'_1, o_2, o_3, ..., o_{n-1}\}$ where we bought a little more of the kth stock and a little less of the lth stock such that $\sum_{i=0}^{n-1} o'_i p_i \leq P$ while keeping every other fraction the same from the optimal solution. Since taking different amounts of a stock doesn't change its risk, $\sum_{i=0}^{n-1} \beta_i \leq R$. Therefore, $o'_0 = o_0 + \delta$ and $o'_1 = o_1 - c\delta$ where c is a scaling constant that ensures O' still obeys the constraint $\sum_{i=0}^{n-1} o'_i p_i \leq P$ and δ is a very small perturbation of the amount of stock. We can solve for c such that it obeys the constraint:

$o'_0 p_0 + o'_1 p_1 + \sum_{i=2}^{n-1} o'_i p_i \leq P$ where $\sum_{i=2}^{n-1} o'_i p_i = \sum_{i=2}^{n-1} o_i p_i$. Therefore,

$(o_0 + \delta)p_o + (o_1 - c\delta)p_1 + \sum_{i=2}^{n-1} o_i p_i = p_0 o_0 + p_1 o_1 + \delta p_0 - \delta p_1 c + \sum_{i=2}^{n-1} o_i p_i \leq P$ and therefore $\delta p_0 - \delta p_1 c = 0 \Rightarrow \delta p_0 = \delta p_1 c \Rightarrow c = p_0/p_1$

Therefore,

$\sum_{i=0}^{n-1} o'_i \theta_i = \theta_0 o'_0 + \theta_1 o'_1 + \sum_{i=2}^{n-1} o'_i \theta_i = \theta_0(o_0 + \delta) + \theta_1(o_1 - (p_0/p_1)\delta) + \sum_{i=2}^{n-1} o'_i \theta_i = \theta_0(o_0 + \delta) + \theta_1(o_1 - (p_0/p_1)\delta) + \sum_{i=2}^{n-1} o_i \theta_i = \theta_0\delta - (p_0/p_1)\delta\theta_1 + \theta_0 o_0 + \theta_1 o_1 + \sum_{i=2}^{n-1} o_i \theta_i = \theta_0\delta - (p_0/p_1)\delta\theta_1 + \sum_{i=0}^{n-1} o_i \theta_i = \delta(\theta_0 - (p_0/p_1)\theta_1) + \sum_{i=0}^{n-1} o_i \theta_i > \sum_{i=0}^{n-1} o_i \theta_i$ since $\theta_0 - (p_0/p_1)\theta_1 > 0$ and $\delta > 0$, therefore O is not an optimal solution since O' has a greater total profit while still obeying the constraints. We can repeat this process; keep taking away from other stocks and buying more of the kth stock until we get the greedy choice (when it equals a_0) and thus the first greedy choice will be in the optimal solution. Therefore, our algorithm produces the optimal solution. □

Theorem 2. *Prove Optimal Substructure: Every solution for sub-problem is optimal. If optimal, then the induced solution is optimal.*

Proof. Assume that P is the optimal solution to the kth sub-problem $I[k : n]$ with maximum profit P_f and containing the first greedy choice p_0. Let $P' = P - \{p_0\}$ and

$I' = I[k+1:n]$ and assume that P' is not an optimal solution to I'. Assume there exists P'' that is the optimal solution to I' with $P''_f > P_f$. Therefore, $P'' \cup \{p_0\}$ is a solution to I with $P''_f > P'_f$ and thus □

Problem 2.

Problem 3. *Given an $n \times n$ matrix M, where $\forall_{i,j \in |M|} C[i][j] \in \mathbb{Z}^+$ where $C[i][j]$ is the number in the ith row of the jth column. Since we can only move the right one and down one, in order to calculate the maximum number we can recursively take the sum of the maximum of the numbers to the left and up of the current cell and add it to the value of the current cell until we reach the beginning cell in order to calculate the maximum total number at the current cell. Mathematically, the recurrent relationship is written like this:*

$MaxNum[i][j] = max(MaxNum[i-1][j], MaxNum[i][j-1]) + C[i][j], \forall_{i,j \in |M|} i, j > 0$

where we initially have $MaxNum[0][0] = C[0][0], MaxNum[0][1] = C[0][0] + C[0][1], MaxNum[1][0] = C[0][0] + C[1][0]$.

```
Psuedocode:


procedure dp_path(M): where M[i][j]=C[j][j] since it is the number at those indicies


for i in [1,|M|]:
     for j in [1,|M[0]|]:
         M[i][j]<-max(M[i-1][j],M[i][j-1])+M[i][j]


 return M[|M|-1][|M[0]|-1]
```

Where we have $C[i][j]$ number for a given i, j index in M is stored at $M[i][j]$.

Proof. Proof of correctness:

The recursive invariant $MaxNum[i][j] = Max(MaxNum[i-1][j], MaxNum[i][j-1]) + C[i][j], \forall_{i,j \in |M|} i, j > 0$ will be that the total number at the ith row and jth

column is the maximum total number. We will use induction on i without loss of generality for some jth column.

Base case: Before the first iteration

$max(M[0][j]) = max(\sum_{l=0}^{j} C[0][l])$ which is true since $max(c_1) = c_1$

Assume:

$MaxNum[k][j]$ is the maximum total at k with $MaxNum[k][j] = max(MaxNum[k-1][j], MaxNum[k][j-1]) + C[k][j]$

Prove for $k+1$:

$MaxNum[k+1][j] = max(MaxNum[k][j], MaxNum[k+1][j-1]) + C[k+1][j]$

Case 1: $MaxNum[k][j] > MaxNum[k+1][j-1]$

$MaxNum[k+1][j] = MaxNum[k][j] + C[k+1]$ which makes $MaxNum[k+1][j]$ the maximum total number because it is larger than the previous maximum total number $M[k][j]$ by adding the current largest number.

Case 2: $MaxNum[k][j] < MaxNum[k+1][j-1]$

$MaxNum[k+1][j] = MaxNum[k+1][j-1] + C[k+1]$ which makes $MaxNum[k+1][j]$ the maximum total number because it is the sum of the current number and a number that was larger than the previous largest total number $M[k][j]$.

Therefore, $MaxNum[i][j]$ is the maximum total at the ith and jth indicies. $\square$