Babiker Babiker
bbabiker@csu.fullerton.edu
4/25/2025
CPSC 335

Report

Pseudocode:

Algorithm 1 (Exhaustive Optimization):

```
FUNCTION stock_maximization_exhaustive(M, items):
  max_stocks = 0
  best_combo = empty list
  n = length of items

  FOR mask FROM 1 TO (2^n - 1):
    current_combo = empty list
    total_value = 0
    total_stocks = 0

    FOR i FROM 0 TO n-1:
      IF (mask BITWISE-AND (1 LEFT-SHIFT i)) is not zero:
        current_combo.append(i)
        total_value = total_value + items[i][1]
        total_stocks = total_stocks + items[i][0]

    IF total_value <= M AND total_stocks > max_stocks:
      max_stocks = total_stocks
      best_combo = current_combo
```

Babiker Babiker
bbabiker@csu.fullerton.edu
4/25/2025
CPSC 335

```
  RETURN [max_stocks, best_combo]
```

Algorithm 2 (Dynamic Programming):

```
FUNCTION stock_maximization_dp(M, items):
  n = length of items
  dp = 2D array with (n+1) rows and (M+1) columns, all set to 0

  FOR i FROM 1 TO n:
    stocks = items[i-1][0]
    value = items[i-1][1]
    FOR w FROM 1 TO M:
      IF value > w:
        dp[i][w] = dp[i-1][w]
      ELSE:
        dp[i][w] = MAXIMUM(dp[i-1][w], dp[i-1][w-value] + stocks)

  w = M
  selected = empty list
  FOR i FROM n DOWNTO 1:
    IF dp[i][w] not = dp[i-1][w]:
      selected.append(i-1)
      w = w - items[i-1][1]

  RETURN [dp[n][M], selected]
```

Babiker Babiker
bbabiker@csu.fullerton.edu
4/25/2025
CPSC 335

Report:

The exhaustive search algorithm systematically evaluates all possible subsets of stocks to find the combination that maximizes the number of stocks purchased without exceeding the investment budget M. It uses a bitmask approach to generate all 2^n possible subsets, calculates the total value and number of stocks for each subset, and keeps track of the optimal solution. This approach has a time complexity of O(n2^n) and space complexity of O(n), making it suitable only for small numbers of items due to its exponential growth. In contrast, the dynamic programming algorithm constructs a 2D table to store intermediate results, building up the solution by considering each item and all possible remaining budgets from 1 to M. It has a time and space complexity of O(nM), which is more efficient for larger inputs when M is reasonably bounded. The DP approach leverages optimal substructure and overlapping subproblems to avoid redundant calculations. While both methods guarantee optimal solutions, the exhaustive search becomes impractical for n > 20 items, whereas the dynamic programming approach can handle much larger inputs efficiently, provided M isn't excessively large. The choice between them depends on the problem size, with exhaustive search being simpler to implement for small instances and dynamic programming offering superior scalability for larger problems while still maintaining quality.