King Saud University
College of Computer and Information Science
Computer Science Department

2nd Semester - 1445

Optimal Portfolio Allocation

CSC311 Project Implementation Report

Name	ID
Yara Alfouzan	443203087
Lamees Alturki	443200862

Table of Contents

1. Brute Force

1.1 Pseudocode

```
function findOptimalAllocationBF(assets, totalInvestment, riskTolerance):
    optimalPortfolio = null
    maxReturn = 0
    portfolio = Portfolio(0, 0, 0)

    for i = 0 to 100:                   // 0% to 100%
        for j = 0 to (100 - i):         // 0% to 100 - i%
            remaining = 100 - i - j     // remaining%

            portfolio = Portfolio(i, j, remaining)

            if ((i / 100.0) * totalInvestment <= assets[0].units
                && (j / 100.0) * totalInvestment <= assets[1].units
                && (remaining / 100.0) * totalInvestment <= assets[2].units):

                portfolio = Portfolio(i, j, remaining)
                portfolio.calculatePortfolioEfficiency(assets)

                if (portfolio.risk <= riskTolerance && portfolio.expectedReturn > maxReturn):
                    maxReturn = portfolio.expectedReturn
                    optimalPortfolio = portfolio

    return optimalPortfolio
```

1.2 Explanation

The program consists of two classes: Asset and Portfolio. The Asset class represents individual assets and stores information related to each asset. Also there is the Portfolio class which represents a client's investment portfolio and contains an allocation array for his assets, expected return, and risk.

In the main method, asset's information are being read from a file; each line of the file corresponds to an asset, and the information is split and stored in the assets list. Note that the last two lines of our input file are the total investment amount and the risk tolerance values.

1. Initialize the variables optimalPortfolio and maxReturn to store the best portfolio found so far. Then create a Portfolio object to represent different allocation combinations. To allocate, nested loops are used to iterate through all possible asset allocation percentages; the outer loop iterates from 0% to 100%, which

is the allocation percentage for the first asset, and the inner loop iterates from 0% to (100 - i)% second asset allocation percentage, then it calculates the remaining percentage for the third asset after allocating percentages for the first two assets.

2. Create a Portfolio object with the current allocation percentages. then validate the following:
   - If the units allocated to each asset don't exceed the available units.
   - If the allocations are valid:
     - Calculate the portfolio's return and risk using the calculatePortfolioEfficiency function.
     - Check if the portfolio's risk is within the specified tolerance and if its expected return is higher than the best return found so far.

3. If the portfolio meets the risk tolerance and has a higher expected return than the current maximum, update maxReturn and optimalPortfolio with the current portfolio's allocation values.
4. After all allocations are checked, return the optimal portfolio found, which represents the best (highest return) combination of assets within the risk tolerance. If no valid allocation is found, optimalPortfolio remains null.

1.3 Sample Run on the Provided Cases

```
Optimal Allocation:
AAPL: 300 units (30% of investment)
GOOGL: 500 units (50% of investment)
MSFT: 200 units (20% of investment)
Expected Portfolio Return: 0.0630
Portfolio Risk Level: 0.0240
macbook@Lameess-MacBook-Air Algorithms-
```

```
Optimal Allocation:
AMZN: 117 units (13% of investment)
TSLA: 396 units (44% of investment)
FB: 387 units (43% of investment)
Expected Portfolio Return: 0.0789
Portfolio Risk Level: 0.0380
macbook@Lameess-MacBook-Air Algorithms-
```

1.4 Source Code

```
public class InvestmentFirm {
    private static class Asset {
        String id;
        double expectedReturn;
        double individualrisk;
        double units;

        public Asset(String id, double expectedReturn, double risk, double units) {
            this.id = id;
            this.expectedReturn = expectedReturn;
            this.individualrisk = risk;
```

```
            this.units = units;
        }
    }
    // Portfolio class representing each client
    private static class Portfolio {
        double[] allocation; // Percentage how much of each asset in total investment
        double expectedReturn;
        double risk;

    public Portfolio(int asset1, int asset2, int asset3) {
        allocation = new double[3];
        allocation[0] = asset1 / 100.0;
        allocation[1] = asset2 / 100.0;
        allocation[2] = asset3 / 100.0;
    }

    public void calculatePortfolioEfficiency(List<Asset> assets) {
        expectedReturn = 0.0;
        risk = 0.0;
        for (int i = 0; i < assets.size(); i++) {
            expectedReturn += allocation[i] * assets.get(i).expectedReturn; //return of each asset in
portfolio allocation
            risk += allocation[i] * assets.get(i).individualrisk; //risk of each asset in portfolio
allocation
        }}}
    public static void main(String[] args) throws IOException {

     List<Asset> assets = new ArrayList<>();
        int totalInvestment = 0;
        double riskTolerance = 0;
        try {
            File file = new File("Example1.txt");
            Scanner scanner = new Scanner(file);
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                String[] values = line.split(" : ");
                if (values.length == 4) {
                    String id = values[0];
                    double expectedReturn = Double.parseDouble(values[1]);
                    double riskLevel = Double.parseDouble(values[2]);
                    int quantity = Integer.parseInt(values[3]);
                    assets.add(new Asset(id, expectedReturn, riskLevel, quantity));
                } else if (values.length == 1) {
                    String[] words = values[0].split(" ");
                    if (words[0].equals("Total")) {
                        totalInvestment = Integer.parseInt(words[3]);

                    } else if (words[0].equals("Risk")) {
                        riskTolerance = Double.parseDouble(words[4]);
                    }}}
            scanner.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }

        Portfolio optimalPortfolio = findOptimalAllocationBF(assets, totalInvestment, riskTolerance);
        // solution
        System.out.println("Optimal Allocation:");
        for (int i = 0; i < assets.size(); i++) {
            System.out.printf("%s: %.0f units (%.0f%% of investment)\n",
                    assets.get(i).id, optimalPortfolio.allocation[i] * totalInvestment,
                    optimalPortfolio.allocation[i] * 100);
        }
        System.out.printf("Expected Portfolio Return: %.4f\n", optimalPortfolio.expectedReturn);
        System.out.printf("Portfolio Risk Level: %.4f\n", optimalPortfolio.risk);
}
```

2. Dynamic Programming

2.1 Pseudocode

```
function findOptimalAllocationDP(assets, totalInvestment, riskTolerance):
    optimalPortfolio = null
    maxReturn = 0
    dp[101][101][101]

    for i from 0 to 100:
        for j from 0 to (100 - i):
            remaining = 100 - i - j

            if ((i / 100.0) * totalInvestment <= assets[0].units &&
                    (j / 100.0) * totalInvestment <= assets[1].units &&
                    (remaining / 100.0) * totalInvestment <= assets[2].units):

                portfolio = Portfolio(i, j, remaining)

                if i > 0 and dp[i-1][j][remaining] is not null:
                    portfolio.expectedReturn += dp[i - 1][j][remaining].expectedReturn
                    portfolio.risk += dp[i - 1][j][remaining].risk
                if j > 0 and dp[i][j-1][remaining] is not null:
                    portfolio.expectedReturn += dp[i][j - 1][remaining].expectedReturn
                    portfolio.risk += dp[i][j - 1][remaining].risk
                if remaining > 0 and dp[i][j][remaining-1] is not null:
                    portfolio.expectedReturn += dp[i][j][remaining - 1].expectedReturn
                    portfolio.risk += dp[i][j][remaining - 1].risk

                portfolio.calculatePortfolioEfficiency(assets)

                if portfolio.risk <= riskTolerance and portfolio.expectedReturn > maxReturn:
                    maxReturn = portfolio.expectedReturn
                    optimalPortfolio = portfolio

                dp[i][j][remaining] = portfolio

    return optimalPortfolio
```

2.2 Explanation

dp is a 3D array used for memoization. It stores precalculated portfolio information for different combinations of asset percentages, the size is 101x101x101 to represent 0% to 100% of each asset. A Portfolio object is created with the current weight distribution (i, j, remaining) but the algorithm checks for previously calculated portfolios in the dp array for similar percentage combinations, and if a valid portfolio exists, it adds its expected return and risk to the current portfolio instead of recalculating. This makes use of previously computed values to avoid redundant calculations. Then use calculatePortfolioEfficiency function to calculate the current portfolio's expected return and risk. If the current portfolio's risk falls within the tolerance and its expected return is higher than maxReturn, maxReturn is updated and optimalPortfolio is replaced with the current portfolio.

The final crucial step of dynamic programming is that the algorithm stores the current portfolio information in the corresponding dp array slot for future reference. This allows the algorithm to avoid recalculating the same portfolio multiple times. Finally, the function returns the optimalPortfolio containing the best asset allocation found.

2.3 Recurrence relation

$$F(i,j,k) = \begin{cases} null \; if \; any \left\{\frac{i}{100} \times totalInvestement, \frac{j}{100} \times totalInvestement, \frac{k}{100} \times totalInvestement\right\} \; exceeds \; the \; corresponding \; asset's \; units \\ protfolio(i,j,k) + dp[i-1][j][k] + dp[i][j-1][k] + dp[i][j][k-1] \quad otherwise \end{cases}$$

2.4 Sample Run on the Provided Cases

```
Optimal Allocation:
AAPL: 300 units (30% of investment)
GOOGL: 500 units (50% of investment)
MSFT: 200 units (20% of investment)
Expected Portfolio Return: 0.0630
Portfolio Risk Level: 0.0240
macbook@Lameess-MacBook-Air Algorithms-
```

```
Optimal Allocation:
AMZN: 117 units (13% of investment)
TSLA: 396 units (44% of investment)
FB: 387 units (43% of investment)
Expected Portfolio Return: 0.0789
Portfolio Risk Level: 0.0380
macbook@Lameess-MacBook-Air Algorithms-
```

2.5 Source Code

```java
import java.io.*;
import java.util.*;

public class InvestmentFirm {
    private static class Asset {
        String id;
        double expectedReturn;
        double individualrisk;
        double units;

        public Asset(String id, double expectedReturn, double risk, double units) {
            this.id = id;
            this.expectedReturn = expectedReturn;
            this.individualrisk = risk;
            this.units = units;
        }
    }
    // Portfolio class representing each client
    private static class Portfolio {
        double[] allocation; // Percentage how much of each asset in total investment
        double expectedReturn;
        double risk;

    public Portfolio(int asset1, int asset2, int asset3) {
        allocation = new double[3];
        allocation[0] = asset1 / 100.0;
        allocation[1] = asset2 / 100.0;
```

```
        allocation[2] = asset3 / 100.0;
    }

    public void calculatePortfolioEfficiency(List<Asset> assets) {
        expectedReturn = 0.0;
        risk = 0.0;
        for (int i = 0; i < assets.size(); i++) {
            expectedReturn += allocation[i] * assets.get(i).expectedReturn; //return of each asset
in portfolio allocation
            risk += allocation[i] * assets.get(i).individualrisk; //risk of each asset in portfolio
allocation
        }
    }
}
    public static void main(String[] args) throws IOException {
double totalInvestment=0;
        double riskTolerance=0;

            // List to store the parsed Asset objects
            List<Asset> assets = new ArrayList<>();
          try{
            File file = new File("Example1.txt");
            Scanner scanner = new Scanner(file);

            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                String[] values = line.split(" : ");
                if (values.length == 4) {
                    String id = values[0];
                    double expectedReturn = Double.parseDouble(values[1]);
                    double riskLevel = Double.parseDouble(values[2]);
                    int quantity = Integer.parseInt(values[3]);
                    assets.add(new Asset(id, expectedReturn, riskLevel, quantity));
                } else if (values.length == 1) {
                    String[] words = values[0].split(" ");
                    if (words[0].equals("Total")) {
                        totalInvestment = Integer.parseInt(words[3]);

                    } else if (words[0].equals("Risk")) {
                        riskTolerance = Double.parseDouble(words[4]);
                    }
                }
            }
            scanner.close();
        }catch(FileNotFoundException e){
            e.printStackTrace();
        }
            Portfolio optimalPortfolio = findOptimalAllocationDP(assets, totalInvestment,
riskTolerance);

            System.out.println("Optimal Portfolio:");
            for (int i = 0; i < assets.size(); i++) {
                System.out.printf("%s: %.0f units (%.0f%% of investment)\n",
                        assets.get(i).id, optimalPortfolio.allocation[i] * totalInvestment,
                        optimalPortfolio.allocation[i] * 100);
            }
            System.out.printf("Expected Portfolio Return: %.4f\n", optimalPortfolio.expectedReturn);
            System.out.printf("Portfolio Risk Level: %.4f\n", optimalPortfolio.risk);
private static Portfolio findOptimalAllocationDP(List<Asset> assets, double totalInvestment, double
riskTolerance) {
    Portfolio optimalPortfolio = null;
    double maxReturn = 0;
    Portfolio[][][] dp = new Portfolio[101][101][101]; //momoization table(ranges from 0% to 100%)

    // Loop through all possible allocations
    for (int i = 0; i <= 100; i++) { // 0%-->100%
        for (int j = 0; j <= 100 - i; j++) { // 0%-->100-i%
```

```
            int remaining = 100 - i - j; // remaining%

            // Check if units assigned to each asset don't exceed available units
            if (((i / 100.0) * totalInvestment) <= assets.get(0).units &&
                    ((j / 100.0) * totalInvestment) <= assets.get(1).units &&
                    ((remaining / 100.0) * totalInvestment) <= assets.get(2).units) {

                Portfolio portfolio = new Portfolio(i, j, remaining);//new portfolio with the
current allocation

                //reuse past calculations if available
                if (i > 0&&(dp[i-1][j][remaining])!=null) {
                    portfolio.expectedReturn += dp[i - 1][j][remaining].expectedReturn;
                    portfolio.risk += dp[i - 1][j][remaining].risk;
                }
                if (j > 0&&(dp[i][j-1][remaining])!=null) {
                    portfolio.expectedReturn += dp[i][j - 1][remaining].expectedReturn;
                    portfolio.risk += dp[i][j - 1][remaining].risk;
                }
                if (remaining > 0&&(dp[i][j][remaining-1])!=null) {
                    portfolio.expectedReturn += dp[i][j][remaining - 1].expectedReturn;
                    portfolio.risk += dp[i][j][remaining - 1].risk;
                }

                portfolio.calculatePortfolioEfficiency(assets); //calculate portfolio return and
risk
                // Check if risk is within tolerance and return is higher than best so far & update
when a new optimal allocation is found
                if (portfolio.risk <= riskTolerance && portfolio.expectedReturn > maxReturn) {
                    maxReturn = portfolio.expectedReturn;
                    optimalPortfolio = portfolio;
                }
                dp[i][j][remaining] = portfolio; // Store the optimal portfolio for current asset
percentages
            }
        }
    }
    return optimalPortfolio;
}
}
```

3. Brute Force vs. Dynamic Algorithm

Algorithm	Best Case Complexity	Example	Worst Case Complexity	Example
Brute Force Allocation	if n=3: O(100^n-1)	Assuming the number of assets is 3, the best case is 100^2. Going through 2 loops giving each asset a different precent every time and the remaining will be assigned to the 3rd asset.	O(100^n-1)	n is the number of assets, since every percent given to an asset is derived from the asset before it, it will iterate through a new loop to allocate every asset and the remaining will go to the last asset; so if there's 8 assets, then 7 loops will be iterated through and the last asset will take the remaining percent
Dynamic Allocation	O(1)	When the risk tolerance is exceeded by the very first portfolio that satisfies the constraints. In this	O(100^n-1)	When the algorithm needs to explore all possible allocations within the given constraints to find the

		case, the algorithm would terminate early after checking only a few portfolios, as it finds the optimal portfolio quickly without fully exploring all possible allocations.		optimal portfolio. This occurs when the risk tolerance is very low, so the algorithm must iterate through all possible combinations of asset allocations. In our implementation, for simplicity we assumed only 3 asssets, but realistically if there were 10 assets we will have 9 loops

4. Challenges

Understanding how to solve the problem using brute force was tough since we had to deal with inconsistent amounts of assets, which made things tricky. To handle this, we thought about splitting the final allocation into a sum of percentages of each asset. Another challenge we faced was figuring out the best and worst complexity cases and coming up with different situations to test the solution.

The main challenge we faced lay in considering many aspects to make a decision all at once, so we fixated on some details while failing to consider others. We overcame this challenge with time, by becoming thoroughly familiar with the project.

Another challenge we faced was in teamwork where communication and collaboration can be challenging and team members may feel unmotivated or disconnected from the project, leading to a lack of participation and effort.

Overall, solving this problem needed careful planning and a good understanding of how to approach it step by step.

5. Peer Evaluation

Part 1: Team Work				
Criteria	**Student1**	**Student2**	**Student3**	**Student4**
Work division: Contributed equally to the work	1	1	-	-
Student succeeds in smoothly forming /joining group within time	1	1	-	-
Peer evaluation: Level of commitments (Interactivity with other team members), and professional behavior towards team & TA	1	1	-	-
Project Discussion: Accurate answers, understanding of the presented work, good listeners to questions				
Time management: Attending on time, being ready to start the demo, good time management in discussion and demo.				
Total/3				