Quantitator

Optimizing Product Quantity

Quantitator: Your Product Quantity Solution

Ayman Ashraf Mounir
Poula said Ibrahim
Kerolos Wageh Farouk
Omar Ibrahim Ibrahim Awad
Moataz Ahmed Mohamed

Supervised By
Dr. Doaa Saleh

CAIRO UNIVERSITY
JULY, 2025

ABSTRACT

Fashion retail faces significant challenges in balancing profitability with operational constraints such as limited budget, logistics, and shelf space. This project introduces a comprehensive optimization framework to help fashion retailers determine the optimal product mix to maximize net profit. By integrating mathematical modeling, genetic algorithms (GA), ant colony optimization (ACO), and AI-driven decision support, the system assists in making strategic production decisions based on fixed pricing structures and real-world constraints.

The developed system, Quantitator, simulates production scenarios, evaluating trade-offs across production, marketing, logistics, and inventory age penalties. Our approach ensures efficient resource allocation and avoids overproduction, minimizing waste and financial risk. Data-driven methods are employed using Python-based algorithms, integrated within a Java Spring Boot backend and interactive web interface.

Through simulation and comparative analysis of GA, ACO, and GAMS implementations, the system identifies the best-performing algorithm for different problem scales. The results demonstrate notable improvements in profitability, scalability, and decision-making accuracy. The solution empowers retailers to adapt quickly to market dynamics while maintaining a sustainable and optimized inventory strategy

DECLARATION

We hereby declare that our dissertation is entirely our work and genuine / original. We understand that in case of discovery of any PLAGIARISM at any stage, our group will be assigned an F (FAIL) grade and it may result in withdrawal of our Bachelor's degree.

Group members:

Name	**Signature**
Ayman Ashraf Mounir	_____
Poula said Ibrahim	_____
Kerolos Wageh Farouk	_____
Omar Ibrahim Ibrahim Awad	_____
Moataz Ahmed Mohamed	_____

PLAIGRISM CERTIFICATE

This is to certify that the project entitled "**Quantitator**", which is being submitted here with for the award of the "**Bachelor of Computer and Artificial Intelligence Degree" in "Operations Research and Decision Support** ". This is the result of the original work by **Ayman Ashraf, Paula Said, Kerolos Wagih, Omar Ibrahim** and **Moataz Abdeen** under my supervision and guidance. The work embodied in this project has not been done earlier for the basis of award of any degree or compatible certificate or similar tile of this for any other diploma/examining body or university to the best of my knowledge and belief.

Turnitin Originality Report

Processed on 31-May-2017 00:14 PKT

ID: 300502964

Word Count: 16856

Similarity Index

10%

Similarity by Source

Internet Sources:	06%
Publications:	0 %
Student Papers:	08%

Date: 10/07/2025

Supervisor Name (Supervisor)
Dr. Doaa Saleh

Table Of Content

List Of Figures

CHAPTER 1

INTRODUCTION

1.1 Introduction

Today's retail fashion industry is characterized by fierce competition and rapid change. Companies find it difficult to strike a balance between maximizing profits and making prudent use of their resources. Our project takes on this challenge head-on by developing a mathematical model that determines the optimal product mix to produce, with the goal of maximizing profits while operating within practical constraints.

This model places all the related elements, including production costs, marketing costs, logistics costs, logistics costs and product purchases. This determines the quantity of all the products that must produce all this data. The purpose is simple, but important. Avoid maximum resources and avoid excessive inventory, inefficient use of financial resources and waste.

Profit calculations are paid by deducting all costs, including the costs of production, marketing, transport and shelves. Unlike the systems that change constantly due to market changes, the prices of the model are placed to obtain the correct amount of fasteners and focus. This allows the company to benefit from changing price policy.

The specificity of our model is to combine other operating restrictions in a single system. These customers really want on the shelf and all costs are advantageous for all costs. It also helps prevent each product from losing the cost of each product, so that the total loss cost does not exceed the sale price of the product.

When the company implements this data, it can make a wise decision to optimize the use of resources, reduce waste and optimize profits. The model also offers specific ways that can identify production under market conditions, which can reduce the risk of customers too or less or less.

1.2 Problem Statement

The The fashion industry faces significant challenges in optimizing product pricing and inventory levels due to complex factors such as fluctuating demand, diverse product attributes, and various cost constraints (production, marketing, logistics, and storage). Traditional methods often lead to suboptimal decisions, resulting in:

• Reduced Profitability: Inefficient allocation of resources and missed revenue opportunities.

• Increased Costs: Higher expenses due to overproduction, excessive marketing spend, and inefficient logistics or storage.

• Market Responsiveness Issues: Difficulty in quickly adapting to market changes and consumer preferences.

There is a critical need for an advanced, data-driven solution that can accurately determine optimal product quantities and pricing strategies, thereby maximizing profitability and operational efficiency within the dynamic fashion market.

1.3 Project Goals and Objectives

Our main goal is to create an extended optimization model that allows decoration. This model determines the mixing of the ideal product to increase the net profit, taking into account operations, logistics and financial restrictions.

Specific Objectives:

Profit Maximization

- Create a mathematical function that calculates overall net profit by subtracting all costs (production, marketing, logistics, shelf space) from fixed retail prices.

Efficient Resource Utilization

- Make a promise of budget revenues and reduce waste for production, marketing and logistics.

If you do not want to complete or underline, make sure the store is best used and the product is displayed effectively.

Demand Alignment

The level of production is suitable with the market demand (led to sales loss) and excessive production (cost increase).

Constraint Compliance

- Build in and respect operational constraints, including:
    - Keeping total production expenses within budget
    - Ensuring marketing and logistics spending doesn't exceed allocated budgets
    - Managing production quantities and inventory levels to meet demand and shelf space requirements

Sustainability and Risk Mitigation

- Reduce financial and environmental risks of making too much product, including waste and inventory costs
- Ensure every product remains profitable by verifying that total cost per unit doesn't exceed the discounted retail price

Practical Application and Scalability

- Creating a comprehensive model working on a variety of business contexts and product lines
- Provide actionable insights and strategies for production planning that companies can implement right away

Advanced Implementation Objectives:

- Development of the model of mathematical optimization: it creates a strong mathematical frame that clearly reflects the luxury of industrial production.
- Use Genetic Algorithms and Ant Colony Optimization Techniques to Efficiently Solve Complex Optimization Problems.
- Intuitive user interface: Create a responsible user friend interface design, so that business users do not need technical expertise and connect to optimized models.
- Integrate AI Capabilities: Incorporate Google Gemini AI to enhance decision support through advanced API integration.
- Enable Data-Driven Decision Making: Develop capabilities to transform raw business data into optimization model inputs.
- Manages the system's growth and adaptability by designing the system architecture for changes in system architecture lines, changes in seasonal data, and changes in key courts.

It is advisable to explain the results: the visualization and reporting version using the optimization results.

1.4 Proposed System

1.4.1 System Features

Our proposed optimization model for determining the optimal product mix includes these key features:

Data Integration and Analysis

- Uses historical performance and cost data to inform the model and support strategic production decisions
- Features intuitive Excel dashboards for data entry, result visualization, and sensitivity analysis to evaluate different scenarios

Mathematical Modeling

- We use the line of linear profit to determine the various optimal fashion items.
- Includes important restrictions, such as production budget, marketing and logistics costs, product costs, market demand and shelf space.

It takes into account the risk of market profit and volatility estimates by absorbing the unique interest or interest.

Fixed Pricing Structure

- Works with set pricing and focuses solely on optimizing production levels without changing retail prices

1.4.2 System Functionalities

Data Input and Management

- Costs associated with production, marketing, and historical data regarding logistics and shelf performance for each product

It offers a personal detection interface for pre -Larg prices, the expected market demand and budget restrictions.

Profit Optimization

- The net profit after cost and punishment is determined by the sale price.

Revenue and production can be the highest and, at the same time, the limit value of respect is increased.

Constraint Management

- Ensures production, marketing, and logistics expenses remain within allocated budgets
- Manages shelf space efficiently, preventing exceeding maximum available capacity

keeps profitability from going negative and matches production levels with market demand.

Sensitivity Analysis

- offers resources for assessing how adjustments to input variables—like demand, expenses, or budgets—affect production schedules and financial success.

Result Visualization

- Delivers comprehensive dashboards showing optimal production quantities, resource utilization, and profit breakdowns
- Highlights potential constraint violations to guide decision-makers

Decision Support Reporting

- Generates detailed reports outlining optimization results, including performance metrics, financial projections, and key recommendations
- Offers exportable formats (Excel, PDF) for stakeholder presentations and documentation

1.5 Development Methodology

We are employing an organized and iterative process to construct the optimization model in order to produce a solution that is dependable, accurate, and easy to use. Our approach blends Agile development concepts with specialised optimisation approaches.

1.5.1 Requirements Analysis

This phase focuses on understanding the problem space and defining system goals:

Identifying Core Challenges

- Examining business challenges including tight profit margins, limited shelf space, marketing budget constraints, and fluctuating consumer demand

Stakeholder Consultation

It refers to a useful concept to get a themed expert and a specific personal restriction on the model (price strateg, marketing manager and operating manager)

1.5.2 System Design

This phase translates requirements into a detailed system blueprint:

Architectural Design

- Defining key system components including user interface (Excel-based dashboards), data management structure, and mathematical optimization engine

Component Design

- Designing individual modules such as:
    - Data input interfaces for product and cost data
    - Visualization tools for displaying optimization results

Data Flow Diagrams

- Creating visual representations of data flow (cost inputs, constraints, optimization outputs) to facilitate implementation modularity and transparency

1.5.3 Implementation

The implementation follows a structured process ensuring effectiveness, usability, and seamless business integration:

Model Implementation

- Using a platform such as integration with the linear optimization model (Python, C #, Matlab)
- Calculating the optimal quantity to produce for each product type

Data Input Interfaces

Creating Excel forms or templates for cost data, product details, limitations (budgets, shelf space), and expected demand

Optimization Engine Integration

- Implementing core optimization logic to compute the best product mix that:
    - Stays within total budget
    - Respects marketing and logistics limits
    - Avoids shelf space overload
    - Maximizes total net profit (based on fixed prices and costs)

Dashboard Development

- Creating interactive dashboards displaying results including projected profits, resource utilization, product-by-product contribution, and optimized production quantities

1.5.4 Agile Development Approach

The project follows an Agile methodology to ensure flexibility, iterative progress, and continuous stakeholder feedback:

Sprint Planning

- Organizing development work into two-week sprints with clear deliverables and acceptance criteria

Daily Stand-ups

- Holding brief daily meetings to ensure team alignment and quickly resolve blockers

Sprint Reviews

- Demonstrating completed work to stakeholders for feedback at the end of each sprint

Sprint Retrospectives

- Regularly reflecting on process improvements and implementing adjustments

Continuous Integration

- Frequently integrating code changes into the main codebase with automated testing to ensure quality

User Story Mapping

- Capturing requirements as user stories with clear business value and acceptance criteria

1.5.5 Deployment

This phase focuses on delivering the finished system to end users and ensuring smooth transition to operational use:

Deployment Planning

- Creating a deployment roadmap detailing technical setup, user onboarding, data migration (if necessary), and rollout schedules

User Training

- Conducting practical training sessions and developing accessible documentation (manuals, quick guides, video tutorials) to enable retail managers and analysts to use the optimization model effectively

Maintenance and Support

- Establishing protocols for monitoring performance, addressing user issues, updating constraints or parameters, and ensuring continued system reliability

1.5.6 Iterative Improvement

Given the dynamic nature of fashion retail, the system incorporates iterative refinement to remain relevant and effective:

Feedback Collection

- Regularly gathering feedback from users (data analysts, marketing teams, retail planners) to identify system limitations or evolving business requirements

Model and System Enhancements

- Updating optimization logic to accommodate new constraints, cost considerations, or market assumptions such as demand fluctuations or seasonal product introductions

Adaptation to Market Trends

- Maintaining focus on production planning optimization while remaining responsive to retail trends including changes in consumer behavior or budget reallocations

1.6 Resource Requirements

1.6.1 Software Requirements

The project relies on these software technologies: - Backend: Java 17, Spring Boot 3.x, Maven - Frontend: HTML5, CSS3, JavaScript (ES6+) - Database: MySQL 8.0 or PostgreSQL 14+ - Integration: Python 3.9+ for data processing - Development Tools: IntelliJ IDEA, Visual Studio Code, Git - Testing: JUnit, Mockito, Selenium - Deployment: Docker, Kubernetes (optional)

1.6.2 Data Requirements

Sample Data Sets

- Historical sales data
- Product cost information
- Marketing expense records
- Logistics cost data
- Shelf space allocation metrics

Data Analysis Requirements

- Data Cleaning: Ensuring data accuracy and preparation for analysis
- Statistical Analysis: Applying statistical methods to derive actionable insights

CHAPTER 2

BACKGROUND AND EXISTING WORK

2.1 Introduction

Industries like manufacturing, logistics, and retail have long relied on optimization models to guide decision-making. These models aim to streamline operations, boost profits, and use resources efficiently within specific constraints. They've proven especially valuable in fashion retail, where limited resources and unpredictable market conditions create significant challenges.

The optimization activities during the time contain a method of accumulating the expenses, projection of the request, and control of the assets. All sorts of strategies such as straight programming, counting problems are used by science and trade specialists to find out the best levels of stocks or accommodate the arrangement of the consumers. This is to satisfy the wish of client, to regulate the costs, to filter the costs and also to any effort to avoid or even high efficiency or not so efficient.

However, in such a multi-product and complex setting as fashion retailing, where the limiting factors such as the most important shelf space, marketing budget limitations, variable and dynamic demand are highly significant, the traditional models fail to perform effectively. Individually, some of these constraints are addressed through some specific models, and there is an increasing demand in integrated solutions that can take all factors into consideration.

Our proposed project expands the current solutions since it takes into account recent breakthrough in data analysis and optimization methods to construct a broad model. With its concentration on one-set prices and optimization of production volume, our suggested system tries to cover a gap in traditional practices and provide a more viable and expandable solution in the manner of the current fashion retail market.

2.2 Literature Review

Prior research establishes that perishable goods face significant demand erosion from two interconnected factors: product aging and excess inventory.

-Ghare and Schrader (1963, p. 125) first formalized deterioration as an inventory-level-dependent process, stating: "The rate of deterioration is proportional to the inventory level: $\frac{dI}{dt} = -thetaI(t)$.

-Dobson, Pinker, and Yildiz (2017, p. 1484) later quantified age-driven demand decay, reporting: "Demand decreases linearly with product age: $(D(t) = a - b.age(t))$,where $(b > 0)$captures age sensitivity."*

-Conversely, stock levels directly suppress demand when inventory signals stagnation. Wee and Yu (1999, p. 475) demonstrated: "The demand rate is a function of the

instantaneous inventory level: $D(t) = alpha + betaI(t)$, demand decreases with inventory level $Ifbeta < 0$(e.g., due to perceived obsolescence or stagnation)."

-Feng, Chan, and Cárdenas-Barrón (2017, p. 208) empirically validated this for perishables, noting: "High stock levels $((I(t)))$ signal stagnation, reducing consumer willingness-to-pay."

-These effects compound multiplicatively. Bazrafshan, Emami, and Mashreghi (2022, Sec. 3.2) showed: "Demand follows $D(p, tau) = alphap^{-gamma}e^{-deltatau}$, where aging tau and price P interact multiplicatively to drive markdowns."

-Our penalty term $Pe_i = d_{base} \cdot \left(\frac{age_i}{age_{max}}\right) \cdot \left(\frac{s_{old,i}}{S_{max}}\right)$synthesizes these mechanisms, penalizing both aging and overstocking as empirically justified.

2.2.1 Fashion Industry Optimization Challenges

Fashion retail industry is an industry that has peculiar optimization issues because of its young and constantly changing trends, and seasonal tendencies. These issues have been looked into in some of the major studies and different ways have been suggested.

Elmaghraby and Keskinocak (2003) studied dynamic pricing strategies where the price changes according to the current levels of food inventory and need. Their studies formed a foundation on the idea of pricing mechanisms wherein production and shelf space limitations are considered and profit-maximization is put into consideration. This article is directly related to our project since it sets an important connection between the aspect of inventory management and retail profitability.

In the study developed by Li and Tian (2013), titled: Dynamic pricing and inventory control for fashion products with learning demand, published in European Journal of Operational Research, the authors concentrated on the dynamic issue of demand uncertainty and pricing within fashion retail sector. This study by them falls in line with our aim to optimize the business model via AI demand forecasting in the dynamic market. There findings indicate the importance of adaptive models that react to changes in market conditions.

2.2.2 UI/UX Design for Optimization Applications

User interface design in optimization applications is unusual in its own way since such systems are required to accommodate both technical aspects and business needs by users with no technical expertise in their fields.

Nielsen and Norman (2013) provided a research that defined key principles in the interface design of complex decision support systems; they focused on the point of

gradual disclosure of information with the emphasis on visualization of optimization results. These are the principles to which we worked toward the frontend development of our system.

The principles of developing interaction design aimed at creating intuitive interface design of complex systems were outlined in the article by Shneiderman, Eight Golden Rules of Interface Design (2016). The principle of minimizing the short-term memory load has a special meaning to our project as well because we have taken this into consideration by carefully structuring the optimization parameters and findings in the user interface.

2.3 Existing Optimization Models

2.3.1 Genetic Algorithms in Fashion Industry

The genetic algorithm (GA) has already been used to overcome multiple optimization issues of the fashion business. Guidelines on the application of metachevous algorithms including gene algorithms were proposed by Michalevik and Vogel (2004) in a solution they entitled, modern heuristic. The relavant comparison of the gene algorithm and the simulated radioactivity has helped in making an experimental stage.

Dev (2001) has brought evolutionary algorithms to solve complex problems that have numerous objective optimizations of the objective algorithm with evolutionary algorithms. This has informed our choice of ensuring that we bring in a gene algorithm to equalize the yield as other businesses would want to do with regard to dealing with an evolutionary algorithm to have a limit of what would be done.

We base the creation of our concept on alcoholism, stability, alcoholics and money constraints. In contrast to the past stock or price management programs, our approaches integrate these issues so as to streamline these issues.

2.3.2 Ant Colony Optimization Applications

The other meta-heuristic method of high potential in combinatorial optimization problem-solving is the Ant Colony Optimization (ACO). In fashion retail, ACO is used much less frequently than Genetic Algorithms, although it has some special advantages in particular classes of optimization tasks.

The theoretical ground on which we implemented was Dorigo and Stutzle paper on Ant Colony Optimization (2004). Their study has revealed that there exists a problem in the complicated behaviour of the capital disorders particularly in relation to ACO output and, especially, price.

Our project goes to style and provides other way of solution to compare with gene algorithm. In the relative technique that is being used, the actual optimization situation, the retailer has the capacity of utilizing the power of the two algorithms.

2.3.3 AI-Driven Decision Support Systems

New developments in artificial intelligence have overtaken decision support systems in different fields. In chapter one of the book The Elements of Statistical Learning (Hastie, Tibshirani, and Friedman, 2009), we were equipped with background knowledge that helped to develop our AI-based demand forecasting module, particularly when it comes to preprocessing and feature engineering.

Gemini AI of Google is the next step in the development of AI models with optimized and decision support capabilities. Our connection to Gemini API helps us improve the level of complexity in data patterns and offer optimization insights to the system.

2.4 Comparative Analysis of Existing Solutions

Several commercial and academic solutions exist for retail optimization, each with distinct approaches and limitations:

- Traditional Inventory Management Systems: Traditional Inventory Management Systems are generally devised around the calculation of reorder points, safety stock, they do not however contain complex capabilities to assist with decision making on what quantity to produce.
- Production quantity solutions: several available solutions tend to major on the aspects of price optimization as opposed to that of volume optimization in production. Although pricing is vital, we also proved that when there is fixed pricing solution, there is a possibility of improvement in profit through optimization in the volumes of the production.
- Supply Chain Optimization Tools These tools can be related to address more general supply chain issues, rather than specialized focus on the optimization of fashion retail production quantities which our system can address.

General-Purpose Optimization Platforms: These are general Optimization Platforms that are already powerful but usually with important customization work required and technical knowledge to enable its application to the particular fashion related retail setting.

Our solution stands out through:

- Fashion-Specific Optimization: The objectives and restrictions encountered in fashion retail have their own characteristics and our mathematical model is tailor-made.
- Multi-Algorithm Solution: Genetic Algorithm and Ant Colony Optimization might be implemented together, which enables us to present a complementaristic assortment of solutions, which can be compared to a certain business case to be chosen.

- Use of Google Gemini AI API: AI-enhanced optimization allows users to get advanced data processing capabilities to enhance optimization.
- User-Centric Design: Our system prioritizes usability for business decision-makers rather than requiring optimization expertise.

2.5 Technological Frameworks

2.5.1 Spring Boot Backend

Spring Boot provides the foundation for our backend services, offering several advantages for our optimization system:

- Dependency Injection: Makes managing and testing of components more straight forward
- Restful API Support: Supports the process of interaction between frontend and optimization services
- Strong Security: Provides protection and control over data The data is safe and secure as well as controlled Robust Security: Guarantees protection of information and control over access
- Scalability: Supports growing data volumes and user bases

2.5.2 Frontend Technologies

The frontend uses modern web technologies to create an intuitive user experience:

- HTML5/CSS3: Provides responsive layout and styling
- JavaScript (ES6+): Enables interactive features and dynamic content
- Data Visualization Libraries: Present optimization results in comprehensible formats
- PDF Export Functionality: Allows sharing and archiving of optimization reports

2.5.3 Optimization Algorithms

Our system implements two complementary optimization approaches:

Genetic Algorithm

- Population-based search technique inspired by natural selection
- Effectively handles multiple constraints and objectives
- Provides diverse solution sets for comparison
- Adaptable to changing business requirements

Ant Colony Optimization

- Inspired by foraging behavior of ant colonies
- Excels at finding paths through complex solution spaces
- Particularly effective for combinatorial optimization problems
- Offers alternative perspective on production quantity optimization

2.5.4 Google Gemini AI Integration

The integration of Google Gemini AI enhances our system's capabilities:

- Advanced Natural Language Processing: Improves interpretation of business requirements
- Pattern Recognition: Identifies trends in historical data
- Decision Support: Provides context-aware recommendations
- Continuous Learning: Improves over time with additional data

2.6 Limitations and Challenges

Our systematic way, data-driven base notwithstanding, the following limitations are inherent which can affect performance and adaptability of the system:

Data Dependency

Predictability and validity of the model provided upon optimization are subject to the quality, consistency and the quality in timeliness of fashion storage information. This contains important data, which includes, cost of production, logistics cost, and sales data of the past. Every incarnation or delay in data could influence the successful outcomes of optimization.

Scalability

The model is efficient on medium level sets but can destroy performance when used with large -scale product catalog or rapid market changes. In this regard, in addition to others that is, additional calculations or parallel processing, may be needed in this case, to support efficient operations of the system.

Fixed Pricing Assumption

The existing model projects the basic price of the product and shows all the power of optimization by the production. This approach makes the space where it is the issue simple and most of the sellers are not only in response to the method of operating the seller, but they also fail to understand the advantages of dynamic prices in some market conditions.

Market Volatility

The industry of fashion retail is exposed to an intensive evolution of trends and seasonal changes which are not easy to forecast. Although our model will use historical data, AI-based forecasting, and will be optimized, unprecedented volatility in the markets can decrease the level of prediction and the ability to find optimal solutions.

Implementation Complexity

Incorporation of numerous superior technologies (Spring Boot, Genetic Algorithms, Ant Colony Optimization, Google Gemini AI) makes up a highly qualified system that demands thorough implementation and care. Small organizations that have been weak in terms of technical resources might have difficulties exploiting all the system potentials.

CHAPTER 3

OPTIMIZATION MODELS IMPLEMENTATION

This chapter will explain how the mathematical optimization model was developed and describes how it will be done, including preparation of data and the list of proposed Genetic Algorithm.

3.1 Mathematical Model Formulation

The essence of the project is to come up with a mathematical optimization model which maximizes net profit of a fashion retail company by finding the optimal product and price combination.

3.1.1 Decision Variables

The main decision variables in this model are:

x_i: Number of units of each product type i to produce (e.g., x_1 for shirts, x_2 for dresses, x_3 for coats).

3.1.2 Objective Function: Net Profit Maximization

The objective is to maximize the total net profit (Z). The net profit for each product unit is calculated by subtracting various costs (production, marketing, logistics, and shelf space) from the retail price, adjusted in the case of any penalty existence.

The net profit per unit is defined as:

$$\text{Net Profit per unit} = \left(P_{ri} - \left(C_{pi} + C_{mi} + C_{li} + C_{si}\right)\right)$$

The overall objective function to maximize total net profit Z is given by:

$$\text{Maximize } Z = \sum_{i=1}^{n} \left(P_{ri} - \left(C_{pi} + C_{mi} + C_{li} + C_{si}\right) \cdot (1 - P_{ei})\right) \cdot x_i$$

Where:

- Z: Total net profit.
- P_{ri}: Retail price of product i.
- C_{pi}: Production cost per unit of product i.
- C_{mi}: Marketing cost per unit of product i.
- C_{li}: Delivery and logistics cost per unit of product i.
- C_{si}: Retail shelf space cost per unit of product i.
- x_i: Number of units produced for product i.

- R_i: Number of products remaining for product i.
- S_i: Shelf space required per unit of product i.
- S_{max}: Maximum shelf space available across all products.
- D_{ei}: Expected market demand for product i.
- B_P: Total production budget.
- B_M: Total marketing budget.
- B_L: Total logistics budget.
- P_{ei}: Penalty or discount rate for product i (a fraction between 0 and 1). This is calculated as:

$$P_{ei} = d_{base} \cdot \left(\frac{age_i}{age_{max}}\right) \cdot \left(\frac{s_{old,i}}{S_{max}}\right)$$

  - d_{base}: Base discount rate (e.g., 0.1 or 10%).
  - age_i: Age of product i (e.g., in years).
  - age_{max}: Maximum age before a product is heavily discounted or removed.
  - $s_{old,i}$: Shelf space occupied by old or unsold units of product i.
  - S_{max}: The maximum shelf space available (total storage capacity).

3.1.3 Constraints

A number of constraints apply to the optimization model to make it feasible and within the applicable limits of its operations.

3.1.3.1 Budget Constraints

- Production Budget: The total production cost must not exceed the allocated budget B_P:

$$\sum_{i=1}^{n} C_{pi} \cdot x_i \leq B_P$$

- Marketing Budget: The total marketing cost must not exceed the allocated budget B_M:

$$\sum_{i=1}^{n} C_{mi} \cdot x_i \leq B_M$$

- Delivery and Logistics Budget: The total logistics cost must not exceed the allocated budget B_L:

$$\sum_{i=1}^{n} C_{li} \cdot x_i \leq B_L$$

3.1.3.2 Logistics and Shelf Space Constraints

- Retail Shelf Space: The total space consumed by all products (newly produced and remaining old stock) must not exceed the maximum available shelf space S_{max}:

$$\sum_{i=1}^{n} S_i \cdot (x_i + R_i) \leq S_{max}$$

  Where S_i is the shelf space required per unit of product i, and S_{max} is the maximum shelf space available.

3.1.3.3 Profitability and Demand Constraints

- Profitability Constraint: Ensures that the total cost per unit does not exceed the discounted retail price, guaranteeing non-negative profit per unit:

$$C_{pi} + C_{mi} + C_{li} + C_{si} \leq P_{ri} \cdot (1 - P_{ei})$$

- Demand Constraints: The total number of units for each product (newly produced plus remaining old stock) must not exceed the expected market demand:

$$x_i + R_i \leq D_{ei}, \forall i$$

  Where D_{ei} is the expected market demand for product i.

- Non-Negativity Constraint: All decision variables and costs must be non-negative:

$$x_i \geq 0, \forall i$$

3.1.4 Model Validation and Verification

Model validation and verification are essential to ensure the correctness and reliability of the optimization model. The following steps were undertaken:

- Verification of Model Logic and Formulation
  The model was revised and edited multiple times to ensure that the logic, constraints, and objective function were correctly formulated and accurately represented the problem requirements.

- Scenario Testing
  The model was tested against small and large data sets and compared to solutions obtained using GAMS. Multiple parameter tweaks were applied to confirm consistency, stability, and correctness of the results across different scenarios.

3.2 Data Extraction and Preparation

Effective implementation of the optimization model depends on accurate, well-prepared data. The following measures were taken to ensure data quality and reliability.

3.2.1 Data Sources and Collection Methods

Data for the model will be sourced from various internal and external systems. Initial product data, including attributes like price, costs, age, and expected demand, can be obtained through web scraping from e-commerce platforms like Myntra.com. This direct acquisition provides real-world product information. For attributes that might be missing or not directly available through scraping, random generations are performed according to each category, followed by adjustments to ensure consistency and completeness of the dataset. This hybrid approach ensures a rich and realistic dataset for the model.

3.2.2 Data Preprocessing Techniques

Raw data often contains inconsistencies, missing values, and outliers, which can negatively impact model performance. Based on the data acquisition methods, preprocessing techniques will include:

- Data Cleaning: Handling missing values (e.g., imputation, deletion), correcting errors, and resolving inconsistencies. This is especially crucial for attributes that were randomly generated or adjusted, ensuring they align with expected ranges and formats.

- Data Transformation: Normalizing or scaling numerical features, encoding categorical variables, and aggregating data to the appropriate level of granularity, as needed for the genetic algorithm.

- Feature Engineering: Creating new variables from existing ones that might improve model performance (e.g., deriving age of product from production date, if not directly scraped).

3.3 Genetic Algorithm Implementation

To address the complex and constrained profit maximization problem, a Genetic Algorithm (GA) is implemented as the primary optimization technique. This method is effective for exploring large, non-linear search spaces while satisfying real-world constraints such as budgets, demand, and shelf space. The full implementation is provided in the GA Code.

3.3.1 Chromosome Representation

In the implementation, a chromosome is encoded as a binary string that represents a candidate solution. Each segment of the bitstring corresponds to the production quantity of a specific product. The number of bits allocated per product is denoted by bits_per_product, and the overall chromosome length is:

$$\text{Total Length} = \text{bits\_per\_product} \times \text{num\_products}$$

For example:

```
Chromosome Bitstring: '010111001010...'
Decoded Quantities: [x₁, x₂, ..., xₙ]
```

The Chromosome class includes:

- `decode()`: Decodes the bitstring into integer quantities.
- `clone()`: Creates a copy of the chromosome.
- `repair()`: Modifies the chromosome to satisfy constraints such as budgets and storage capacity.

3.3.2 Fitness Function Design

The fitness function evaluates the quality of each chromosome by computing a profit score penalized by inventory aging and storage usage. This is implemented in the fitness_and_penalties() function.

The penalty for each product i is calculated as:

$$Pe_i = d_{\text{base}} \cdot \left(\frac{\text{age}_i}{\text{age}_{\max}}\right) \cdot \left(\frac{S_{\text{old},i}}{S_{\max}}\right)$$

The penalized fitness function is defined as:

$$\text{Fitness}(x) = \sum_{i=1}^{n} (pr_i - (cp_i + cm_i + cl_i + cs_i)) \cdot (1 - Pe_i) \cdot x_i$$

Where:

- pr_i: price of product i
- cp_i, cm_i, cl_i, cs_i: costs for production, marketing, logistics, and shelf
- x_i: production quantity of product i
- Pe_i: penalty factor

The function calculateExpectedProfit() computes the unpenalized profit and is used to present the final optimal result to the user.

3.3.3 Genetic Operations

The GA uses standard genetic operators:

- Selection: Tournament selection is used with a tournament size $k = 3$, implemented in tournament_selection().
- Crossover: One-point crossover is applied via one_point_crossover(), producing offspring by swapping gene segments between two parent chromosomes.
- Mutation: Bit-flip mutation is implemented in mutate(), where each bit in the chromosome has a probability mr of flipping.

These operations ensure diversity and help explore the search space effectively.

3.3.4 Constraint Handling

Multiple real-world constraints are incorporated into the chromosome repair and evaluation process:

- Demand: Ensures that new production plus existing stock does not exceed the estimated demand D_e.
- Budgets: Chromosomes must satisfy the production (B_p), marketing (B_m), and logistics (B_L) budget constraints.
- Shelf Space: The total used shelf space, including old stock, must not exceed the maximum $S_{\max}$.

The repair() function adjusts each chromosome to conform to these constraints, and the check_constraints() function validates feasibility.

3.3.5 Parameter Configuration

The GA is configured with the following parameters in the run_ga() function:

- Population Size (pop_size): 50
- Generations (gens): 100
- Crossover Rate (cr): 0.7
- Mutation Rate (mr): 0.01
- Bits per Product (bits_per_chromosome): defined by the user input

These parameters may be tuned manually or using techniques like grid search or adaptive control to achieve an optimal balance between accuracy and computational efficiency.

3.3.6 Output and Reporting

At the end of the GA process, the best solution is decoded and the corresponding production plan is computed. The output includes:

- Optimal quantities to produce for each product
- Cost and price data
- Profit per unit and total profit

The final results are presented as structured JSON data, and the script also provides real-time progress logs and detailed debugging output.

3.3.7 Pseudocode of Genetic Algorithm

The following pseudocode summarizes the main steps of the Genetic Algorithm:

```
BEGIN GeneticAlgorithmRunner
  READ ExcelDataFile
  MAP ExcelColumnsToParameterNames
  INITIALIZE PopulationWithRandomChromosomes
  SET BestSolutionToNone
  SET BestFitnessToNegativeInfinity
  FOR each Generation in TotalGenerations DO
    FOR each Chromosome in Population DO
      DECODE ChromosomeToQuantities
      REPAIR ChromosomeToMeetConstraints
    FOR each Chromosome in Population DO
      CALCULATE FitnessOfChromosome
      STORE FitnessInList
    IDENTIFY ChromosomeWithHighestFitness
    IF ItsFitnessIsBetterThanBestFitness THEN
      SET BestSolutionToThisChromosome
      SET BestFitnessToThisFitness
    CREATE NewPopulationListStartingWithBestChromosome
    WHILE NewPopulationIsNotFull DO
      SELECT TwoParentsUsingTournamentSelection
      PERFORM CrossoverToGenerateTwoChildren
      PERFORM MutationOnChildren
      REPAIR ChildrenToMeetConstraints
      ADD ChildrenToNewPopulation
    REPLACE OldPopulationWithNewPopulation
  END FOR
  DECODE BestChromosomeToGetOptimalQuantities
  CALCULATE RealProfitFromOptimalQuantities
  FOR each ProductIndex DO
    COMPUTE UnitCost
    COMPUTE ProfitPerUnit
    COMPUTE TotalProfitAndCostForProduct
  RETURN TotalProfitAndListOfProductResults
END GeneticAlgorithmRunner
```

3.4 Ant Colony Optimization Implementation

3.4.1 Problem Representation

ACO algorithm treats a profit maximization problem that is characterized by multiple constraints. Optimal quantities of products shall be determined to maximize total profit with the constraints of available finances and available space.

A row of the Data contains such a product, and columns give us essential attributes:

Price: the price per unit which is most important in calculating revenues and profits.

Production_Cost_Per_Unit (Cp): the cost of producing one unit which is limited by the production budget (BP).

Marketing _Cost Per Unit (Cm): the cost per unit to market, subject to the marketing budget (Bm).

Logistics_Cost_Per_Unit (Cl): the expenditure on transportation and distribution as a per unit cost, limited by the logistics budget (BL).
Shelf_Space_Cost_Per_Unit (Cs): Costs that are attached to the shelf space per unit, subject to the logistics budget (BL).

Age: Product Age as value to calculate a penalty factor (Pe) that penalizes the profit of older products.

Remaining_Products (R): Quantity of existing products which is used together with Average_Expected_Demand to calculate demand_max of new production.

shelf_space (S_unit): amount of physical shelf space used per unit, which is important in imposing the maximum amount of shelf capacity (S_max).

Average_Expected_Demand (De): The expected demand of a product which is the upper limit of the number of products to be produced.

Decision Variables and Solution Structure

The variable of the main decision variable is solution array (np.zeros(n_products, dtype=int)), where solution[i] is the quantity of the product i. Such quantities are chosen out of domains, which is set of all the possible integer quantities of each product, and go between 0 and up to demand_max[i]. This provides non-negative hues in the market and production restrictions.

Objective Function

ACO algorithm tries to maximize on total profit. Calculating profit involves a penalty on the product age and quantity on hand. Calculating unit_profit through equation unit_profit = price - (Cp + Cm + Cl + Cs). To calculating net_profit_unit is net_profit_unit = unit_profit * (1 - Pe). The calculation of Pe will be Pe = d_base * (age / age_max) * (s_old_space / storage_sum). This penalizes older products or those with high stocks and it motivates to work on low inventory items.

Constraints

Checking of solution feasibility against key constraints by check_constraints() function to checks constraints feasibility:

Budget Constraint: The summation of total cost of production (np.dot(Cp, solution)), marketing (np.dot(Cm, solution)) and logistics (np.dot(Cl, solution)) costs should not exceed BP, Bm, and BL respectively.

Shelf Space Constraints: sum of all the shelf space occupied (np.dot(S_unit, solution)) should not be more than S_max.

Demand Constraints: np.any(solution > demand_max) makes sure that none of the products have not exceeded their demand_max.

check_constraints() function will return False in case any constrain is violated. Solutions that are not valid are set aside or heavily penalized. Pheromone updates are then informed by the penalized_profit so that the algorithm can explore a trade-off of high profit balanced against feasibility.

3.4.2 Pheromone Update Rules

Another step in ACO is pheromone updating, which defines the search towards the optimal solutions since the successful paths are strengthened. This is done at the end of each iteration when solutions have been built up by ants and the solution that is best in that iteration has been determined. It has two processes: evaporation and deposition.

Evaporation limits continuous build-up of pheromones that would result in premature convergence or stagnation. It simulates natural pattern of dissipation of pheromone. In the code, the evaporation rate (rho) decreases the pheromone values of all paths.

$$for i \in range(n_{products}):$$

$$pheromone[i] * (1 - rho)$$

rho (e.g. 0.3) determines the percentage of pheromone which is evaporated. The problem with avoiding local optima is therefore avoided since aged paths that are less reinforced become not attractive and exploration can thus take place successfully. It also adds forgetting yet preserves search diversity.

Deposition refers to the process through which the ants strengthen the tracks to the good solutions. The code has deposition coupled with iteration_best_sol and it is focused on successful routes.

$$deposit_{amount} = Q * sol_{profit} / (1.0 + sol_{viol})$$

Q (pheromone_coefficient): Weights the total amount of pheromone laid. More the reinforcement gets stronger, more Q get higher.

sol_profit: total profit of iteration_best_sol. More profitable solutions leave more trails.

sol_viol: sum constraint violation of iteration_best_sol. The 1.0 + sol_viol in the denominator reduce deposition of the infeasible solutions.

An amount of deposit_amount is added to the pheromone level of each selected quantity in iteration_best_sol after the calculation:

$$for i \in range(n_{products}):$$

$$idx = domains[i].index\left(sol_{\overrightarrow{[i]}}\right)$$

$$pheromone[i][idx] + deposit_{amount}$$

This selective deposition leaves only the optimal solution which is reinforced sending future ants in the right direction. the pheromone update tries to maintain the exploration (evaporation) and exploitation (deposition), hence the convergence nature of the pheromone update is towards high-quality and feasible solutions.

3.4.3 Solution Construction

Solution construction occurs in which every ant creates a full solution (products quantities). The heuristic information along with this probabilistic process follows the pheromone trails in search of profitable and feasible solutions.

The solutions are constructed a product at a time by each ant. The ant chooses a quantity solution [i] which is a weighted probability distribution of every product i.

Heuristic Information (eta)

eta (η) is a greedy term that constitutes the a priori attractiveness of a quantity with the profit potential. Eta of each quantity x in the code is founded on net_profit_unit:

$$h = [1e-6 if x = 0 else max(0.0, net\ \ [i] * x) for x \in domain] h = np.array(h)$$

$$if h.max() > 0:$$

$$h = h/h.max()$$

To estimate potentially profitable quantity, take the total net profit per one unit: net_profit_unit[i] * x. Then, so that the heuristic value is not negative, use max (0.0, ...): and to ensure that a heuristic value is considerably higher than zero even in the event of x = 0, multiply it by a small constant: 1e-6. The last step is normalizing the values to an interval between 0 and 1: h = h / h.max().

Pheromone Information (tau)

tau(τ) is the amount of social experience of a colony that means how effectively a path has been a part of good solutions. Pheromone concentration starts with uniform random values, and small level which are similar thus fostering exploration. They are updated based on solution quality forming a global memory.

Probability Calculation

A decision made by an ant is a combination of tau and eta to determine weights:

$$weights = (tau * alpha) * (eta * beta) if not np.all(eta = 0) else np.power(tau, alpha)$$

alpha (α): Controls the pheromone effect. Greater alpha instils exploitation.

beta (β): Controls heuristic influence. An increased beta promotes investor greed.

The conditional takes care of situations when eta is zero, and only tau is used in making the decisions.

Normalization is done of probabilities to weights:

$$probs = weights/np.\sum(weights) if np.\sum(weights) > 0 else np.ones(len(domains[i]))/len(domains[i])$$

Quantity Selection and Constraint Checking

An ant picks a number out of the possible ones according to probabilities based on np.random.choice. It is done for all the products. It is a prominent implementation note that the check_constraints is invoked within the product loop:

$$if \check{} \ constraints(solution, Cp, Cm, Cl, S_{unit}, BP, Bm, BL, S_{max}, demand_{max}):$$

$$valid \quad = True$$

This stepwise validation is aggressive, by branching infeasible directions at once. In case some cases of the solution is invalid the flag valid_solution_found is set. If the final solution is found not to be valid the effort of the ant is discarded. In case of valid solutions, total_profit is calculated. penalized_profit is directly assigned to total_profit, and the penalties will be computed during pheromone update rather than during evaluation of the solutions.

3.4.4 Performance Optimization

Early Stopping Mechanism

This avoids execution of the algorithm through every iteration in case there is no improvement made. The number of consecutive iterations without improvement is specified by max_no_improvement (default 5). In case iteration_best_profit breaks records of the best_profit, the counter will be increased, otherwise, the counter will be reset. The loop is left saving computational resources when no_improvement_count rises to max_no_improvement.

Vectorized Operations with NumPy

Numerical operations using NumPy are very fast when used abundantly. Using optimized NumPy functions in Python is so much faster than using loops. Examples include:

Cost and Resource Calculation: Total production, marketing, logistic and shelf space cost using np.dot.

Demand Calculation: np.maximum to perform an effective demand calculation effectively.

Profit Calculation: Element wise operations of unit_profit and net_profit_unit.

Total Profit Summation: np.sum to fasten summing.

NumPy vectorization also does not use Python loops, which results in large data speeding up significantly with NumPy.

Parameter Tuning and Sensible Defaults

The parameters are (alpha, beta, evaporation_rate, n_ants, n_iterations, pheromone_coefficient) that can be sorted out externally. Default parameters (n_ants=100, n_iteration= 10) require a compromise between the quality and computing time. Setting these parameters enables users to get the best performance regarding given problems.

Constraint Checking Efficiency (Revisited)

Although it is computationally costly to call check_constraints within the product loop, doing that serves as an extremely aggressive kind of pruning strategy that ignores any solution found within a partial computation as soon as it proved to be infeasible. This can help stop working on invalid steps towards solutions which indirectly helps in being efficient.

3.4.5 Ant Colony Optimization Pseudocode

Here's a pseudocode representation of the ant_colony_optimization function:

$$FUNCTIONant\ \ (data, parameters):$$

$$EXTRACTparameters(alpha, beta, evaporation_{rate}, num_{ants}, num_{iterations}, etc.)$$

$$INITIALIZE:$$

$$-Product - relateddata(costs, prices, shelfspace, etc.)$$

$$-Constraints(budgets \wedge shelfcapacity)$$

$$-Netprofitperunit, affectedbyproductage \wedge remainingstock$$

$$FOReachproduct:$$

$$IFproducthasnon - positivenetprofit \vee nodemand:$$

$$Restrictdomain[0]$$

$$Setheuristic \wedge pheromonedefaultvalue$$

$$ELSE:$$

$$Createdomain = [0, \ldots, max\ \ ]$$

$$CALCULATEheuristics = normalizedexpectedprofitforeachquantity$$

$$INITIALIZEpheromonevaluesrandomlyforeachquantity$$

$$SETbest_{solution} =$$

$$SETbest_{profit} = -\infty$$

$$SETno\ \ = 0$$

$$FOReachiteration(1num_{iterations}):$$

$$SETiteration\ \ =$$

$$SETiteration\ \ = -\infty$$

$$FOReachant(1num_{ants}):$$

$$CREATEanemptysolutionvector$$

$$FOReachproduct:$$

$$CALCULATEprobabilitydistributionusingpheromone \wedge heuristicvalues:$$

$$probs[i] \propto pheromone[i]^{alpha} * heuristic[i]^{beta}$$

$$RANDOMLY choose a quantity based on the probability$$

$$ADD quantity current solution vector$$

$$IF solution satisfies all constraints:$$

$$CALCULATE total profit for this solution$$

$$UPDATE best_{solution} \wedge iteration \quad if better$$

$$UPDATE pheromones(evaporation):$$

$$FOR each product:$$

$$pheromone[i] * (1 - evaporation_{rate})$$

$$IF iteration \quad \exists:$$

$$DEPOSIT new pheromones based on solution quality:$$

$$FOR each product:$$

$$pheromone[i][chosen \quad] + Q * profit/(1 + violations)$$

$$IF no improvement \in global best:$$

$$INCREMENT no$$

$$ELSE:$$

$$RESET no$$

$$IF no \quad \geq max \quad :$$

$$BREAK(early stopping)$$

$$COMPUTE final results(revenues, costs, penalties, profit)$$

$$GENERATE report \wedge save file$$

$$RETURN best_{solution}, final_{profit}, report, details$$

$$FUNCTION \check{} \ constraints(solution, Cp, Cm, Cl, S_{unit}, BP, Bm, BL, S_{max}, demand_{max}):$$

$$COMPUTE total production, marketing, logistics cost, \wedge shelf space used$$

$$IF any cost exceeds its budget \vee quantity > demand_{max}:$$

$RETURN False$

$ELSE$:

$RETURN True$

This pseudocode provides a clear overview of the ACO process, from initialization to iterative solution construction, pheromone updates, and final reporting.

3.5 GAMS Model Implementation

3.5.1 Mathematical Model in GAMS

GAMS Model Formulation

The GAMS model is formulated as a Mixed-Integer Programming (MIP) problem designed to optimize the total net profit by determining the optimal integer quantities of products to produce. The model incorporates various cost elements, product age, existing stock levels, and budget/capacity limitations.

Objective Function

The objective is to maximize the total net profit, represented by the variable obj. This is achieved by summing the product of the net profit coefficient per unit (profit_coeff(i)) and the quantity to produce (x(i)) for each product i.

Maximize:

Plain Text

obj = sum(i, profit_coeff(i) * x(i));

Constraints

The model is subject to the following budgetary and capacity constraints:

1.Production Budget (prod_budget): The total production cost must not exceed the allocated production budget (Bp).

2.Marketing Budget (market_budget): The total marketing cost must not exceed the allocated marketing budget (Bm).

3.Logistics Budget (log_budget): The total logistics cost must not exceed the allocated logistics budget (BL).

4.Shelf Space (shelf_space): The total shelf space required for new production combined with existing stock must not exceed the maximum available shelf space (S_max).

Scalar (Global Constants)

These are constant parameters applied universally across the model:

•Bp: Production budget.

•Bm: Marketing budget.

•BL: Logistics budget.

•S_max: Maximum available shelf space.

•d_base: Base penalty factor.

Derived Parameters

These parameters are calculated based on input data and other model parameters:

•age_max: The maximum age observed among all products (s_max(i, age(i))).

•storage_sum: The total sum of remaining products across all product types (sum(i, s_old(i))).

•pe(i): Penalty factor for product i, influenced by its age and existing stock. It is calculated as: d_base * (age(i)/age_max) * (s_old(i)/storage_sum).

•cost_u(i): The total unit cost for product i, derived from the sum of production, marketing, logistics, and shelf space costs: cp(i) + cm(i) + cl(i) + cs(i).

•profit_coeff(i): The net profit coefficient per unit for product i, considering all costs and penalties: (pr(i) – cost_u(i)) * (1 – pe(i)).

This formulation defines a Mixed-Integer Programming problem aimed at identifying optimal integer values for x(i) (quantities to produce) to maximize the objective function obj while adhering to all specified constraints.

3.5.2 Parameter and Variable Definition

This section outlines the parameters and variables utilized within the GAMS model, defining their roles in data input, coefficient representation, decision-making, and objective function formulation. All elements are precisely defined to ensure clarity and model integrity.

Sets

•i 'products': Represents the set of distinct products, serving as the primary index for data and equations within the model.

Parameters (Data Inputs)

Parameters represent static information or coefficients that serve as inputs to the model.

•pr(i): Price per unit for product i.

•cp(i): Production cost per unit for product i.

•cm(i): Marketing cost per unit for product i.

•cl(i): Logistics cost per unit for product i.

•cs(i): Shelf space cost per unit for product i.

•age(i): Age of product i.

•s_old(i): Number of units remaining in stock for product i.

•S_i(i): Unit shelf space requirement for product i.

•D_e(i): Expected demand for product i.

Scalar Constants

Scalar constants are global parameters applied across the entire model.

•Bp: Total production budget.

•Bm: Total marketing budget.

•BL: Total logistics budget.

•S_max: Maximum available shelf space.

•d_base: Base penalty factor, influencing effective profit.

Computed Aggregate Parameters

These parameters are derived from input data and represent aggregate values across all products.

•age_max: The maximum age observed among all products (max(i, age(i))).

•storage_sum: The total sum of remaining products across all product types (sum(i, s_old(i))).

Computed Derived Parameters

These parameters are calculated for each individual product i and are crucial for defining the objective function and constraints.

•pe(i): Penalty factor for product i, influenced by its age and current stock, which reduces its effective profit.

•cost_u(i): The total unit cost for product i, calculated as the sum of production, marketing, logistics, and shelf space costs (cp(i) + cm(i) + cl(i) + cs(i)).

•profit_coeff(i): The net profit coefficient per unit of production for product i, considering all costs and penalties ((pr(i) – cost_u(i)) * (1 – pe(i))).

•max_x(i): The maximal permissible production level for product i, constrained by expected demand and existing stock (max(0, floor(D_e(i) – s_old(i)))).

Variables

Variables represent unknown quantities determined by the solver during the optimization process.

•x(i) integer 'quantity to produce': An integer decision variable representing the quantity of product i to be produced. Its lower bound (x.lo(i)) is 0, and its upper bound (x.up(i)) is defined by max_x(i).

•obj 'the total net profit': A continuous variable representing the overall net profit, which is the objective function to be maximized by the model.

3.5.3 GAMS Pseudocode and Solver Configuration

This section details the procedural flow of the GAMS model, from defining its core components to configuring and executing the optimization solver.

GAMS Pseudocode

The following pseudocode outlines the sequential steps involved in the GAMS model formulation:

1. Define Sets: Establish the set i representing products.

2. Define Input Parameters: Specify all data input parameters, including pr , cp , cm , cl , cs , age , s_old , S_i , and D_e .

3. Define Scalar Constants: Declare global scalar constants such as Bp , Bm , BL , S_max , and d_base .

4. Compute Aggregate Parameters: Calculate aggregate parameters like age_max and storage_sum based on input data.

5. Compute Derived Parameters: Determine product-specific derived parameters, including pe , cost_u , profit_coeff , and max_x .

6. Define Variables: Declare the decision variable x(i) as an integer representing the quantity to produce, and obj as a continuous variable for the total net profit.

7. Set Bounds for Variables: Establish lower (x.lo(i)) and upper (x.up(i))

bounds for the decision variable x(i) .

8. Define Equations: Formulate the constraints for production budget (prod_budget), marketing budget (market_budget), logistics budget (log_budget), and shelf space (shelf_space), along with the objective function (objective).

9. Solve Model: Execute the optimization process.

10. Display Results: Present key results such as x.level (optimal quantities) and obj.level (maximum total net profit).

11. Detailed Output Parameters: Calculate and display additional output parameters like unit_cost , profit_per_unit , total_gross_profit , and total_cost .

12. Display Formatted Results: Present a structured table of product details and the overall net profit.

Solver Configuration

The GAMS model leverages external solvers for optimization. The configuration within the GAMS file specifies the solver and its application:

Model Definition: The statement MODEL ga_model /all/; defines ga_model to encompass all declared equations and variables within the model.

Solver Specification: The command option mip = cplex; designates CPLEX as the solver for this Mixed-Integer Programming (MIP) problem. CPLEX is a robust and widely recognized commercial solver for complex optimization problems.

Optimization Execution: The instruction SOLVE ga_model USING mip MAXIMIZING obj; initiates the optimization. GAMS is directed to solve ga_model as a MIP, with the objective of maximizing the obj variable. The solver then proceeds to find optimal integer solutions for x(i) .

CHAPTER 4

RESULTS, EVALUATION, AND DISCUSSION

The chapter entails a comparative evaluation of optimization outcomes of the three algorithmic procedures namely; Genetic Algorithm (GA), Ant Colony Optimization (ACO), and GAMS optimization model. It includes the analysis of simulation results, performance evaluation measures, sensitivity, and a detailed discussion of findings providing the corresponding managerial implications as well.

4.1 Simulation Results

4.1.1 Optimal Product Mix and Quantity

To carry out the optimization of the algorithms, they were tested on product portfolios of different sizes (10, 50, 100, 200 and 500 products) and the most appropriate product mix configurations were identified, which could maximize profitability. Heatmap results showed different characteristics in each algorithm as they were identifying the optimal product quantities, and performance variations get more significant as the problem complexity gets larger. The problem of product mix optimization provided a solution to the basic problem of the resources sharing between the competing product lines involved in meeting the requirements of the market and the capacity constraints

On small-scale problems, the number of products, 10-100, all three algorithms tended to find the same optimal product configuration with little variation in the amount of products.

The Genetic Algorithm performed best within this range with the average profit of $17,020,804 versus ACO who had its average profit at $16,093,301 and GAMS which had its average profit at $17,006,347. This proves the usefulness of evolutionary methods on searching solution space on moderately complex product mix situations.

When the problem implementation grew to medium-scale operations (110 300 products), the algorithms started to show even more varied results in the selection of products and their quantities of. The GA had a competitive advantage, which implied a strong exploration ability within the population-based methodology. The results of GAMS in virtually all problem sizes were in close optimal solutions, proving how mathematical programming technique works to real-life problems of product mix optimization

4.1.2 Net Profit Maximization Outcomes

The profit maximization analysis showed that the three optimization methods produced significant performance disparities of different scales of operations. GAMS and GA produced the most substantial average profit by generating $82,689,688 and $82,678,005 respectively according to the complete range of 50 operational scenarios (10-500 products) followed by ACO that generated an average of $77,359,215.

These findings point out to the fact that the conventional mathematical optimization techniques, such as GAMS, have the capability of achieving the same level of profits as their newer counterparts, the metaheuristic optimizers such as GA.

When it comes to profit extremes, the GA produced the greatest value of profit, which was of $184,268,492, at the 500-product level, meaning that it can be productively applied in extreme large-scale problem situations. Such outstanding performance can be due to the capability of GA to preserve population diversity, and prevent early convergence to local optima characteristics, especially, in complex combinatorial optimization.

This is at the expense of consistency though. The use of standard deviation analysis indicated variations in profits capture as the amounts displayed with GAMS were 50194249, 46765321, and 50143610 by ACO and GA respectively indicating that there was trade-off between large profits and stable solutions.

The larger the scale of the problem (300+ products), the larger the performance gap turned out to be. GAMS and GA continued to average profits of over 133 million dollars (GAMS: 133,338,443; GA: 133,448,118) but ACO kept falling behind to average nearly 124,327,443. This stresses the significance of selecting algorithms, both in terms of scale and involved complexity of operations.

4.1.3 Resource Allocation Efficiency

The efficiency of resource allocation was tested through the capability of the algorithms to involve the maximization of the profit to relative to time of computation. Efficiency was calculated as the quotient between the highest profit and time consumed (ms). As it is

shown in the graph, GAMS is far more better than GA and ACO in terms of efficiency as it is much more efficient among all problem sizes. Also, the mean efficacy of GAMS was approximately equal to 2,000,000, whereas ACO and GA yielded results of less than 10,000, proving the former correct in terms of effective optimization of profits in a time-saving realm.

Such high performance of GAMS is explained by the fact the mathematical problem is optimized deterministically, i.e. is explicitly solved. Comparatively, ACO and GA are iterative population-based searches of any given heuristic, and thus their complexity increases with each increment in the problem.

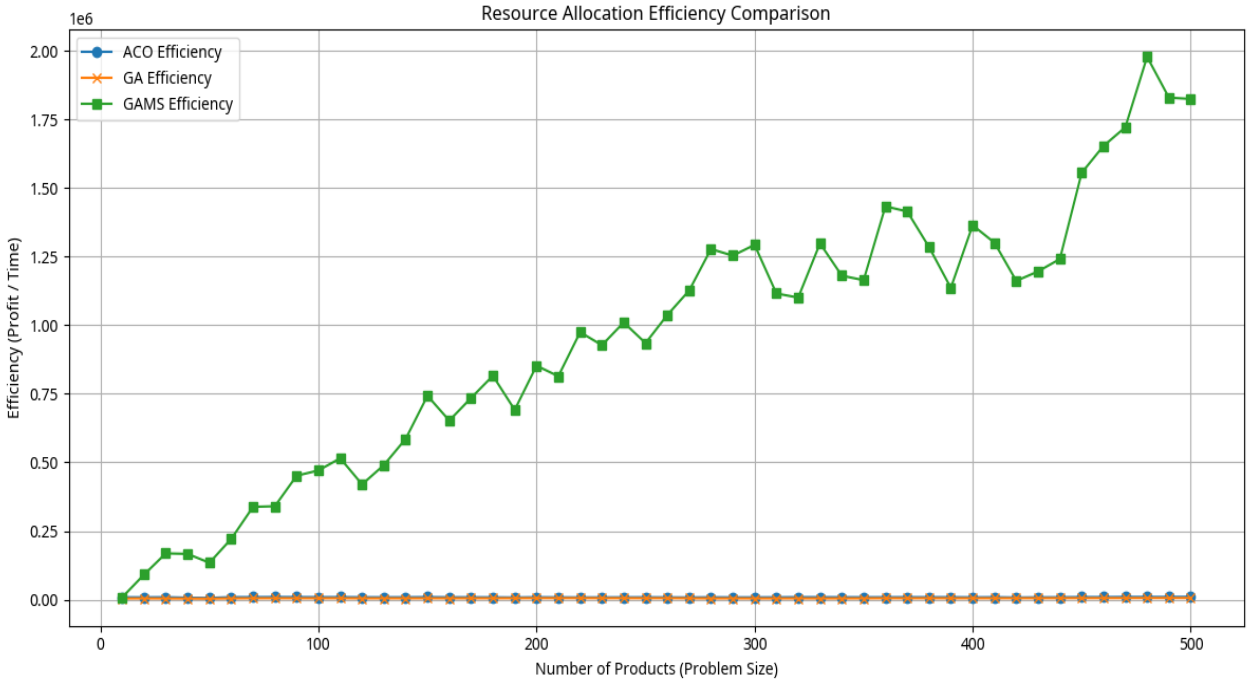

Figure 4. 1 Resource Allocation Efficiency Comparison

- GAMS has a consistently humming efficiency, even when the size of the problem grows. This uniformity brings out its scalability and accuracy in constrained conditions.

- The two methods, namely ACO and GA demonstrate acceptable results on small scale instances, but their efficiency performance curves tend to be flat or worsen with problem size. This means that they incur increasingly high overheads in their search processes.

Even now with their reduced ratios GA and ACO can produce competitive profits and thus can be applied in circumstances where only approximation solutions are acceptable or atmospheres where GAMS-like solvers are inaccessible.4.2 Performance Evaluation of Optimization Algorithms

4.2.1 Genetic Algorithm Performance Metrics

The Genetic Algorithm (GA) demonstrated strong scalability and consistent performance across varying problem sizes, as shown in Figure: GA Profit vs. Problem Size. The chart reveals a clear upward trend, with optimal profit ($) steadily increasing as the number of products grows—from approximately $5 million for small problem instances to over $180

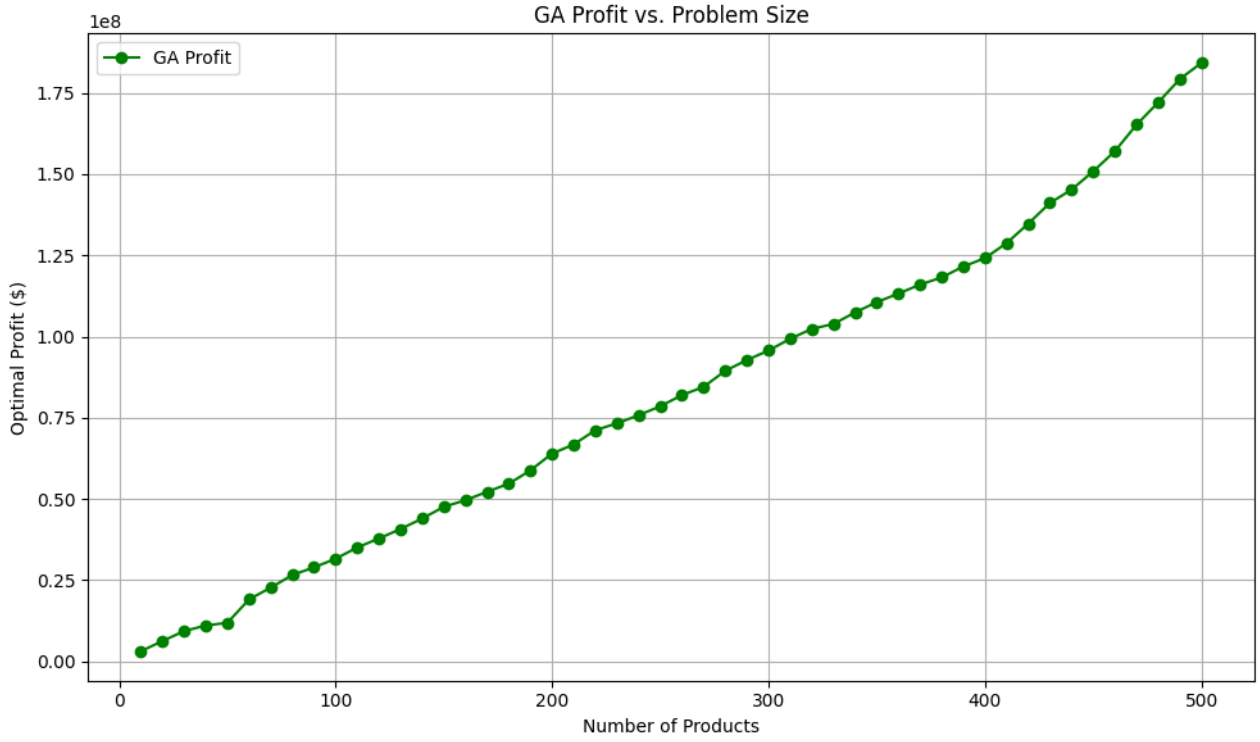

Figure 4. 2 GA Profit vs. Problem Size

million at 500 products.This linear progression indicates that GA maintains its solution quality even as complexity scales. The population-based search strategy, leveraging crossover and mutation operations, effectively explores diverse solution spaces and prevents premature convergence. These mechanisms ensure robust performance even in larger, more complex product mix configurations.

In terms of computational performance, GA exhibited an average execution time of 16,022 milliseconds, with a standard deviation of 9,035 milliseconds reflecting sensitivity to problem size and configuration. Typically requiring 100–200 generations to converge, GA's computational effort scales approximately quadratically, making it well-suited for scenarios where solution quality outweighs execution speed, such as offline strategic planning.

Overall, GA's profit curve in the graph highlights its capability to consistently deliver high-quality solutions while adapting efficiently to increasing problem scales.

4.2.2 Ant Colony Optimization Performance Metrics

The Ant Colony Optimization (ACO) algorithm demonstrated solid scalability and consistent profit performance across varying problem sizes, as illustrated in Figure: ACO Profit vs. Problem Size. The chart reveals a steadily increasing trend in optimal profit, starting from approximately $5 million for small-scale problems and reaching up to nearly $174 million for instances with 500 products.

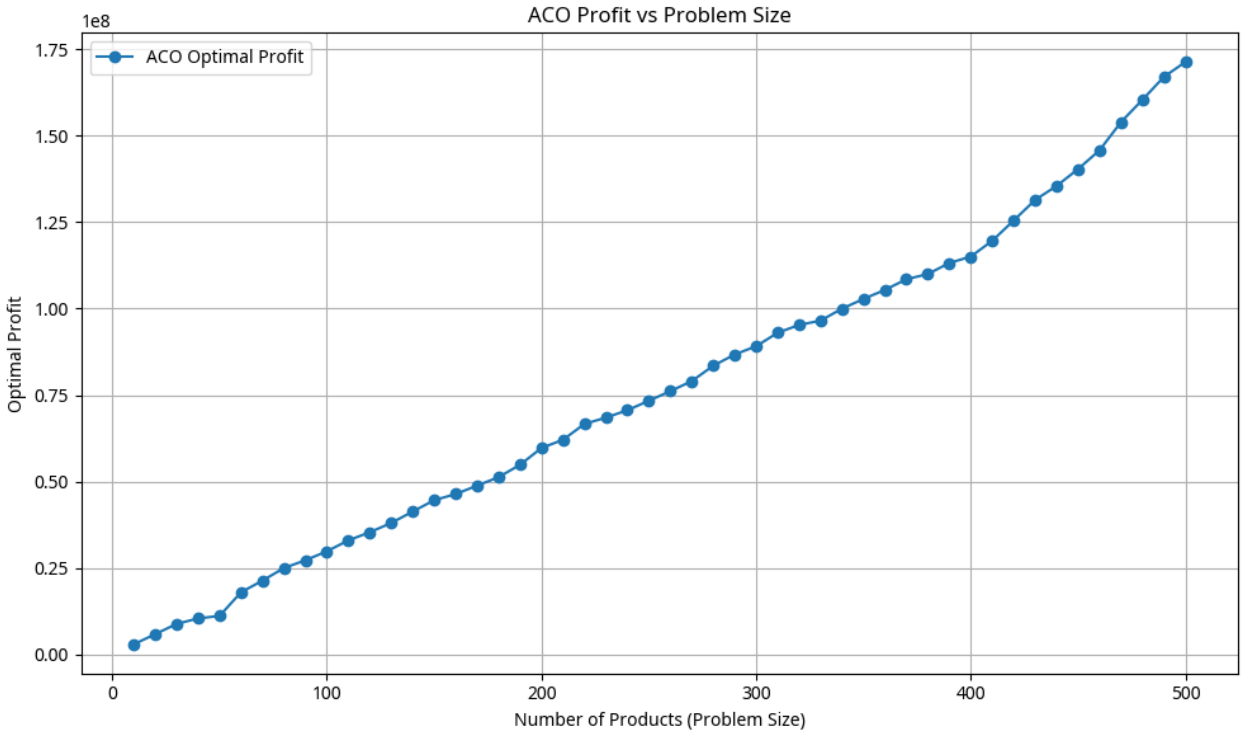

Figure 4. 3 ACO Profit vs Problem Size

ACO optimized the computation time against solution quality where the average run time of 8,679 milliseconds is located between GA and GAMS in regard to the computational load of the algorithm. Although it tended to make slightly lower profits as compared with GA, ACO was able to produce quality solutions with a consistent behavior on different instances of problems.

The advantage of the algorithm is attributed to its foundation on swarm intelligence where there are artificial pheromone lines to induce construction of the solution into high performing areas. Whereas convergence is rather slow when compared to GA, one advantage of the algorithm is that it performs well in medium-scale problems, since the exploration and exploitation balance minimizes the chance of getting stuck in the local optimum.

Also, ACO had the lowest standard deviation in execution performance (681.20 ms), which means that this is a reliable function, and it behaves in a predictable manner. This determinism is one of the reasons why ACO is well adapted to the use cases where stable performance is more important than maximizing profit margins- either on time-constrained or resource-limited scenarios

4.2.3 GAMS Optimization Model Results

Time complexity analysis graph shows some striking dissimilarities with regard to computational performance in ACO, GA and GAMS algorithms. GAMS was exceptionally efficient on a computational level, employing almost the same computation times with regard to the size of the problems. As the plot (red line) indicates, GAMS needed an average of about 88.8 milliseconds, which presented a radical superiority compared to both GA and ACO. The simple difference demonstrated its high scalability and the fact that it is based on deterministic solvers that take advantage of the mathematical structure in order to achieve fast convergence.

In contrast, GA (green line) showed a clear polynomial increase in time with problem size, reaching over 30,000 milliseconds for the largest instances. ACO (blue line) also demonstrated increasing execution times, though more moderate than GA, averaging around 8,679 milliseconds.

Time complexity analysis showing computational scalability of optimization algorithms

These results emphasize that while metaheuristic approaches like GA and ACO are adaptable, they suffer from increased computational costs as problem size grows. GAMS, by comparison, provides near-instant, reproducible results, even at large scales, making it ideal for real-time and operational optimization scenarios.

4.2.4 Comparative Analysis of Algorithm Efficiency and Effectiveness

The thorough comparative study shows that each of the approaches to optimization has its own performance profile, and the choices between the efficiency of computation and the complexity of implementation are obvious. GAMS proved to be strongly ahead with profit-to-time ratios that were almost 200 times better than the metaheuristic competition. The efficiency benefit, however, has the conditions of exact mathematical modeling and is not applicable to problems that have complicated non-linear objective functions or discontinuous constraint spaces.

GA has proved to produce better profit generating ability with an average profit of 6.86 percent above the average profit of ACO with reasonable computation needs.

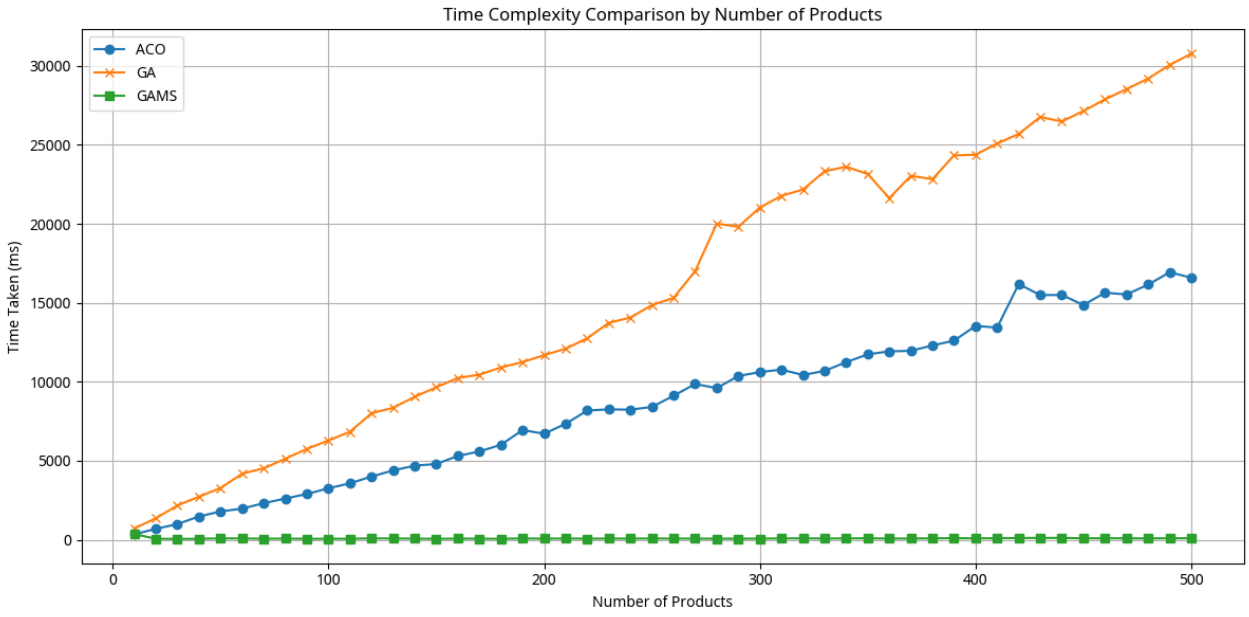

Figure 4. 4 Time Complexity Comparison by Number of Products

The evolutionary mechanisms of the algorithm were found especially well suited to search the complex solution landscapes, finding good product mix configuration. The ability of the GA to cope with diverse constraints types and different characteristics of objective functions makes it applicable to different optimization problems.

Performance scale observed between problem sizes showed that selection of an algorithm ought to be determined by certain operational needs and restrictions. In time sensitive situations that demand a fast optimization, GAMS is unrivalled in the speed of computation. GA has better exploration and robustness of solution with a complex problem where there are more than one or two conflicting goals or where there is a lack of clear definition of parameters.

4.3 Discussion of Findings

4.3.1 Managerial Implications

To conclude, the relative comparison of the optimization algorithms reveals to be an important contribution to the managerial choices of the product mix optimization and allocation of the resources. GAMS can be considered as the best solution to organizations that need real-time optimization options with deterministic result, especially those companies operating under manufacturing setting and constraint structure that is well structured and objective functions that are linear. Its functionality and computational efficiency combined with the reliability of its solution qualifies it to be a proper choice of the algorithm used by operational decision support systems in situations requiring immediate response with respect to their conditions changing.

Where the optimization problems are complex multi-objective, optimisation problems or uncertain operating environment, GA has superior exploration powers and robustness of solutions. Having the capability to consider non-linear constraints and discontinuous objective functions, the algorithm can be of value in strategic planning applications where quality of the solution is more important that speed of calculation. The population based approach of GA will inherently yield many near-optimal solutions and thus managers will have the ability to look at alternative approaches and ways to mitigate the risks.

The results indicate that coupled techniques of optimization involving the strength of mathematical programming and exploration features of metaheuristics may lead to optimal results in a complex context of organizations. Organizations are advised to select algorithms depending on type of problem, time requirements and requirements of solution quality instead of a blanket approach of applying one type of algorithm. The large discrepancies in performance levels across scale of problems are strong indications of the significance of matching the optimization technique with operational scales of complexity.

The investment on an optimization infrastructure must take into account long-term needs of scalability because, given the complex interactions between algorithms, the performance differences are intensified with increased operational complexity. The optimization methods should focus on organizations that intend expansion or diversification of the product line by focusing on preserving performance attributes along different scales..

4.3.2 Limitations of the Current Study

There are a number of shortcomings which limit the scope and applicability of the present optimization analysis. The analysis was limited to the single maximization of profits to the exclusion of multi-objective situations that may apply in practice by the organizations. When dealing with real world product mix decision, conflicting goals that include maximization of market share, reduction of risk and maximization of customer satisfaction may change the relative algorithm performance.

The supporting conditions of the experiment presupposed the deterministic nature of demand and costs parameters, whereas the real business conditions are unpredictable and follow the possibility of stochastic fluctuations. In the settings of uncertainty, algorithm behavior can be vastly different to the deterministic scenarios discussed in this paper: especially in the case of methods such as GAMS which assume knowledge of exact parameter values. Future studies have to consider the stochastic programing techniques, as well as the robust optimization techniques in order to deal with the uncertainty effects.

Computational measures taken in the study might not be able to reveal the full complexity of implementation and the difficulty of adopting in the organization. In the real world, the choice of an algorithm is not only based on its technical performance, but also staff training needs, the complexity of integrating software and the complexity of its maintenance. Such pragmatic aspects may make a difference in the viability of optimization methods in practice.

The tuning of parameters and the setup of the algorithm was made common to all test instances which did not imply the best performance concerning problem characteristics. Individual parameter optimization per algorithm and problem type may provide alternate relative performance trends, especially in case of the metaheuristic whose performance often depends on the parameter choice.

4.3.3 Recommendations for Future Research

Numerous limitations exist that restrict the usefulness and scope of the current optimization analysis. The resultant analysis even excluded the possibility of multi-objective conditions which might arise in practice on the part of the organizations through the singular maximization of profits. Relative performance of the algorithms may change depending on the conflicting goals that exist in real world product mix decision which may involve maximization of market share, minimization of risk and maximization of customer satisfaction.

The assumptions of the supporting conditions of the experiment implied the deterministic character of the parameters of demand and the cost, but the business reality, which is unpredictable, is driven by the potential of the stochastic fluctuations. Under these conditions of uncertainty, the behavior of algorithms can change compared to that under deterministic conditions considered here: in particular, there is a difference compared to methods of which the exact values of parameters are known, like GAMS. The stochastic programing methods, and the robust optimization techniques must be taken into account by the future studies in order to address the effect of uncertainty.

The study may not be able to reflect the complexity of what it takes to implement and how difficult it can be to adopt in the organization due to computational measures made in the

study. Practically, a decision to use a specific algorithm is also determined by staff training requirement, software and maintenance complexity as well as integration complexity in the real world. These pragmatic considerations can be the difference as to whether optimization strategies are viable in practice.

The parameter fitting and the initialization of the algorithm was backgrounded as shared to each of the test instances that did not suggest optimum results as involved with the nature of the problem. Optimizing the individual parameters per algorithm and per type of problem might offer different relative performance trends, in particular in the case of the metaheuristic, the behavior of which can be sensitive to parameterization

CHAPTER 5

SYSTEM ARCHITECTURE AND DESIGN

5.1 System Overview

Quantitator is a complete software that helps the fashion business optimize their products best quantities. It uses sophisticated algorithms like Genetic Algorithms (GA) and Ant Colony optimisation (ACO) to conduct analysis on a wide range of factors i.e. the cost of production, the cost of marketing, logistics, shelf space and demand. The system offers a solid platform to simulate various quantity scenarios and determine best quantity to maximize on choice of quantity and produce allocation strategies.

To the core, the system has been structures as a client-server system. The backend is built using spring boot that gives it a strong and scalable base in terms of processing the business logic, data and integration with the external services. The frontend is a web-based interface using HTML, CSS, and JavaScript, but designed to provide the user with a more natural experience of uploading their data, setting up the simulation parameters, and visualizing the results. The remarkable fact about the system architecture is its combination with the Python services to implement the sophisticated system of optimization algorithms and with the Gemini AI to achieve the functions of smart conversation.

The system in general is meant to simplify the process of quantity optimization, so that the businesses can make data-driven decision, optimize costs of operation and increase revenue. it talks about challenges of dynamic market conditions and variable product by giving a tool of strategic planning that is responsive and highly effective.

5.2 Backend Architecture

The backend of Quantitator is developed using Spring Boot framework, which is robust and highly embraced in developing enterprise quality Java application. Spring boot is based on the development of stand-alone, production-ready spring apps which you can just run. It offers an efficient and scaled method of dealing with complex business logic, data management, interaction with different services.

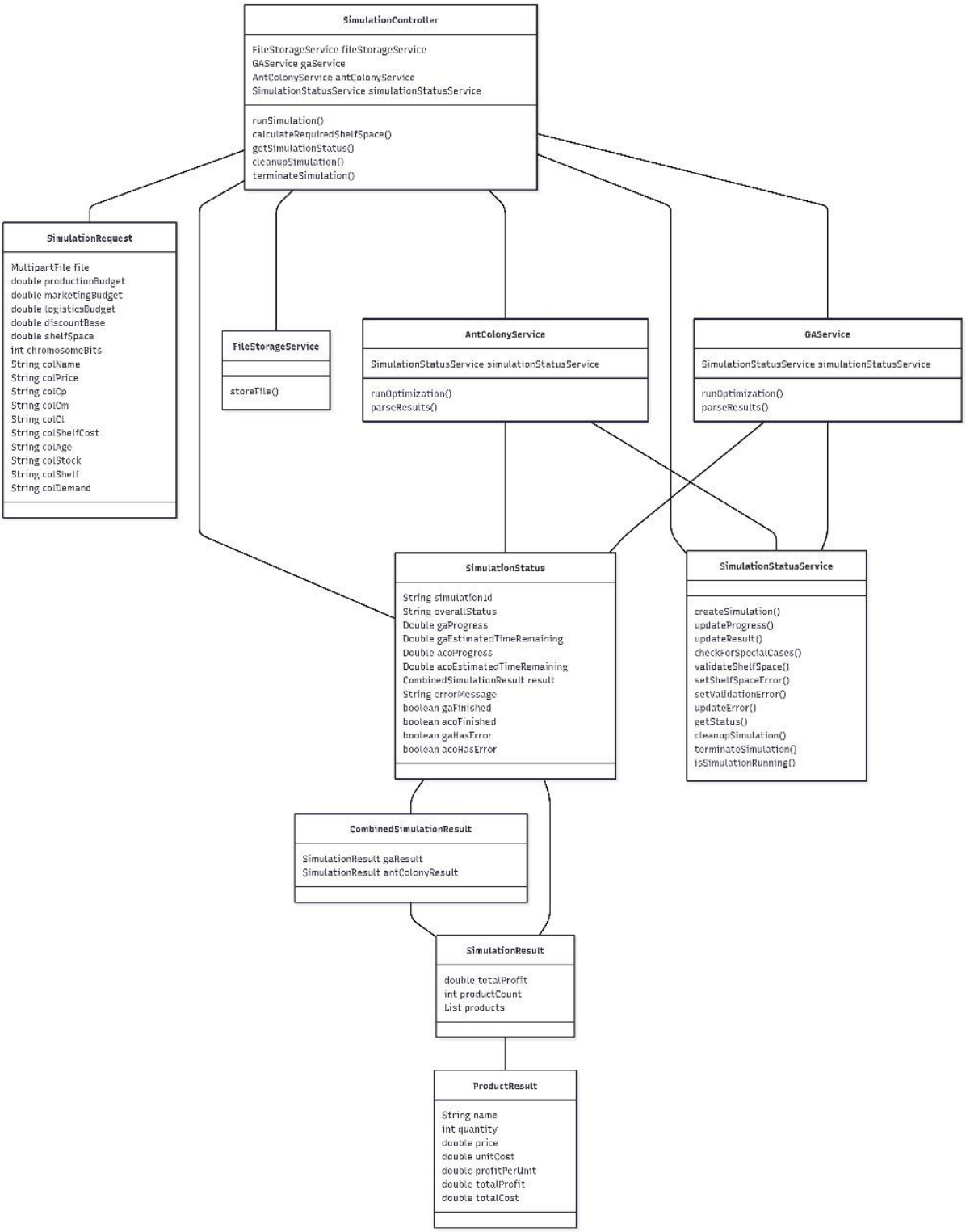

Figure 5. 1 Class Diagram

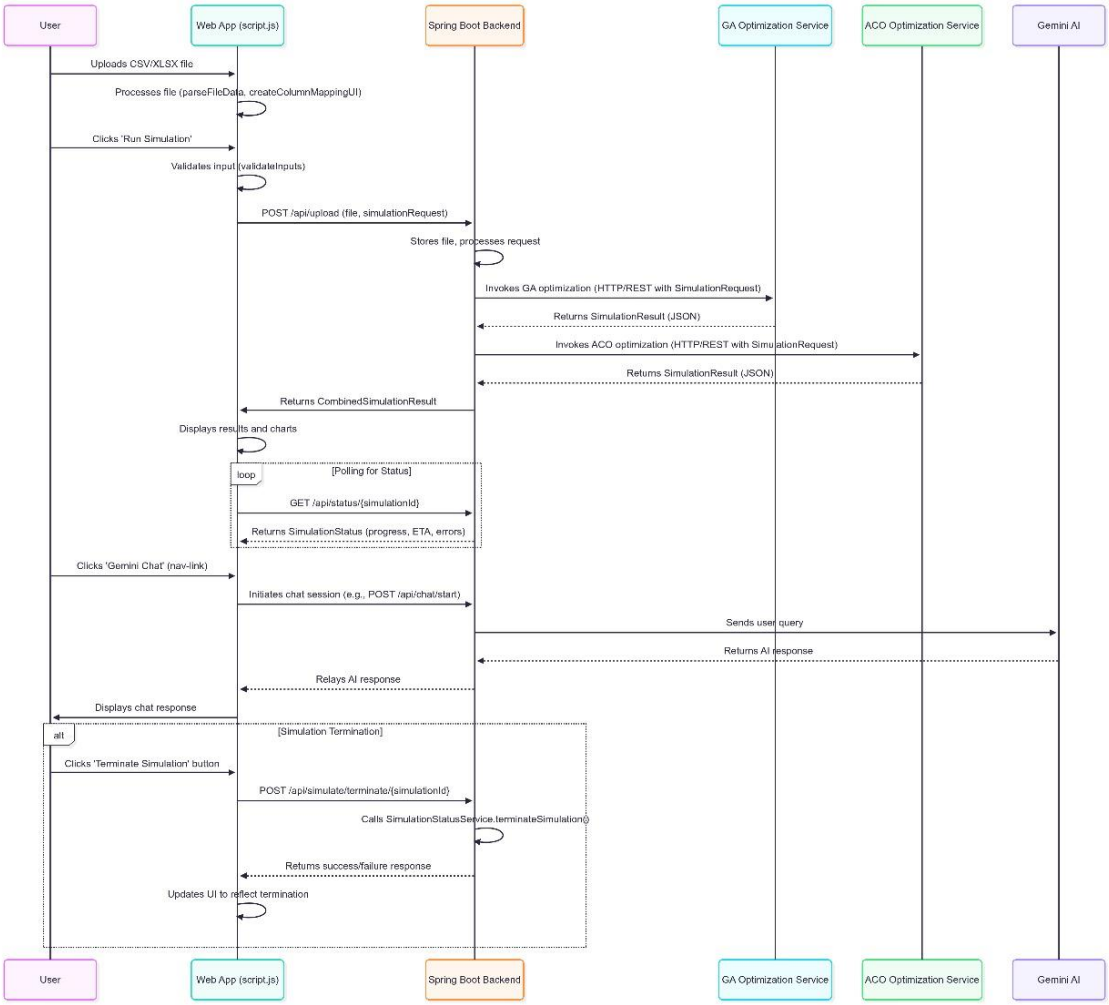

Figure 5. 2 Sequence Diagram

5.2.1 Spring Boot Application Structure

Spring Boot application adopts a common layered architecture which encourages separation of concerns and maintenance. Its main elements entail:

Controller Layer: This layer deals with any incoming HTTP requests, it handles the input and returns the input as well. A case in point is the SimulationController.java, which relays commands relating to the simulation and the responses.
Service Layer: It contains the heart of business logic. Such services as GAService.java, AntColonyService.java, FileStorageService.java, and the classes contained in SimulationStatusService.java enclose respectively, the operations concerning genetic energies, ant colony optimization, file management and the process of simulation status.
DTO (Data Transfer Object) Layer: The plain java objects that are defined here represent how to transfer the data between various layers of the app, and even between networks. Such examples are SimulationRequest.java,

SimulationResult.java, ProductResult.java, CombinedSimulationResult.java, SimulationStatus.java.
Configuration: Can be done using application.properties which makes application settings centralized like port, Gemini API key, application logging setting etc.

Such arrangement is done to make the application modular, testable, and easy to extend. Auto-configuration and convention-over-configuration principles used in Spring Boot more than eliminates the boilerplate code, therefore, letting developers concentrate on the essential operations.

5.2.2 RESTful API Design

The backend offers a series of RESTful APIs to communicate with the front end and possibly other external processing. These APIs also follow the REST principle, the standard methods of resource manipulation are based on HTTP (POST, GET). The main API endpoint is /api/simulate, and it takes simulation request and invokes simulation optimization process.
The other endpoint that is also important is /api/status/{simulationId} it lets the frontend access the real time status and results of an ongoing simulation.
The main features of RESTful API design can be identified as follows:

Resource-Oriented: APIs are resource-oriented (e.g. simulation, status) and possess the benefit of being intuitive and easy to grasp.
Statelessness: Every request of the client to the server must have all the information needed to process the request, and should not require any context of the client, as maintained by the server, beyond the request. This increases both scalability and reliability.
JSON Communication: The data is mostly transmitted as JSON (lightweight, human-readable data interchange format) so that it is usable with as many clients as possible.

The typical simulation request RESTful API endpoint diagram and sequence are presented in figure 5.3.

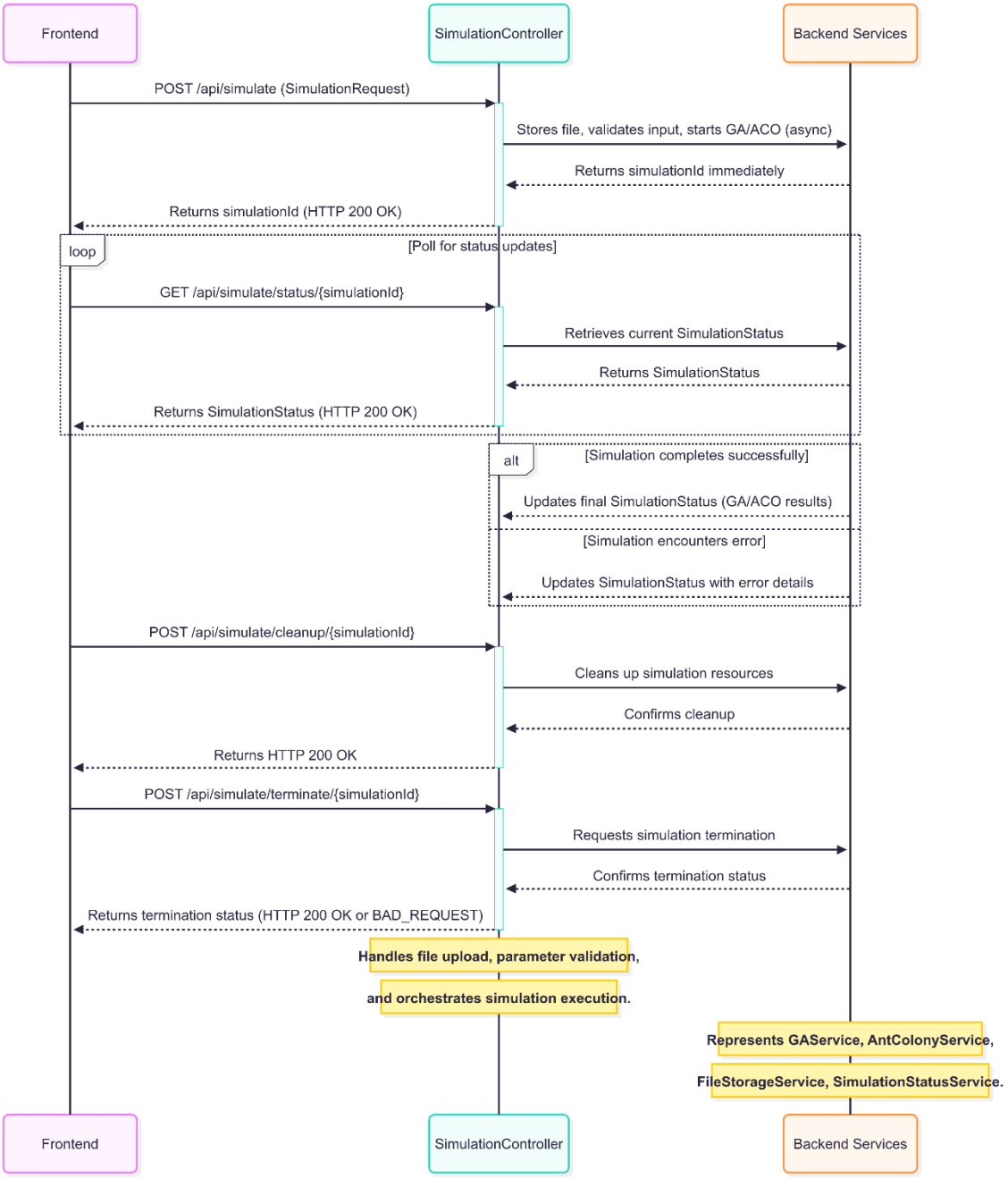

Figure 5. 3 RESTful API Endpoint Diagram Sequence Diagram

5.2.3 Service Layer Components

The backend core is the service layer which implements the business logic and also organizes the interactions among the various components. All the services are made to realize a particular set of operations, which facilitates modularity and reusability. The major parts of the service are:

GAService.java: It is a service which carries out the processing of Genetic Algorithm (GA) optimisation. It communicates with the Python script GA.py,

sends some parameters as inputs and receives results processed. It also changes the status of the simulation using SimulationStatusService.
AntColonyService.java: This is also an equivalent to the GAService and it is used to execute Ant Colony Optimization (ACO) algorithm. It interacts with the ant.py Python script, does I/O and sends the status and outcome to the SimulationStatusService.
FileStorageService.java: This service takes the responsibility of saving and loading the files sent by users (CSV/XLSX). It makes sure that files are safe. those that are saved in a temporary directory and provides the paths to the optimization services to work on it.
SimulationStatusService.java: An important service which has in real-time the status of the simulation in action. It monitors the progress, processes validation errors, updates the results of both GA and ACO, and the overall progress of a simulation lifecycle. This service is indispensable to send prompt response to the frontend.

The interactions among these components of the service layer are shown in figure 5.4.

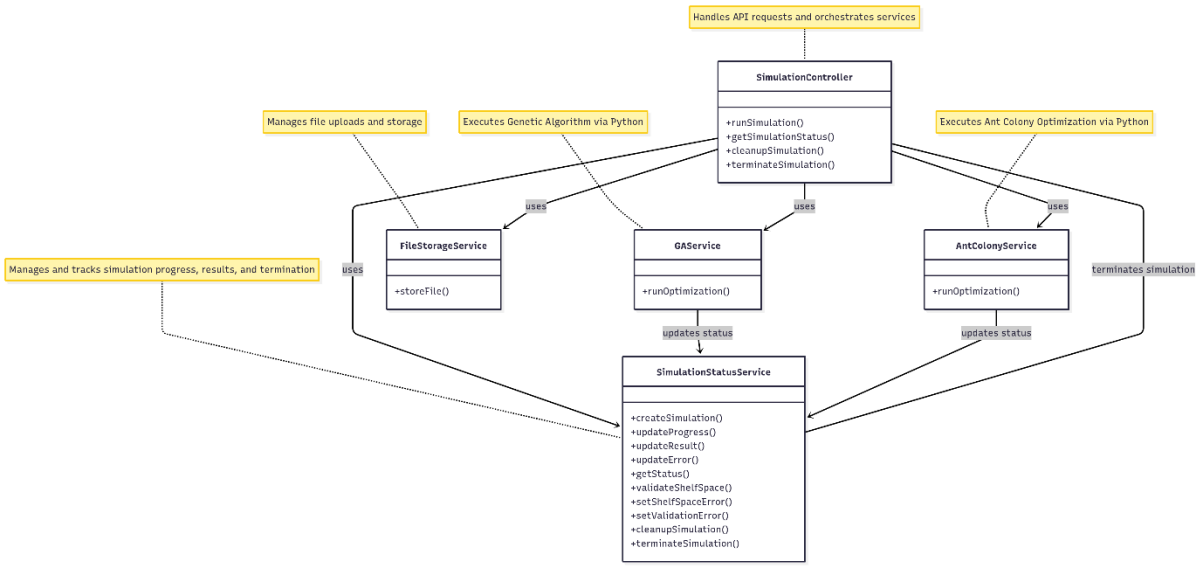

Figure 5. 4 Service Layer Interaction Diagram

5.2.4 Data Transfer Objects (DTOs)

Data Transfer Objects (DTOs) are important in determination of the structure of the data being shared between the frontend and the backend and also between different frameworks in the backend. They wrap data as a type safe and clean mechanism to exchange information that does not expose the internal domain model. This solution will provide more measures of security, easier data validation, and also consistency in APIs.
Major DTOs of the system are:

SimulationRequest.java: This DTO contains all the parameters necessary to start a simulation i.e. budget settings (product, marketing, logistics), shelfspace, base of discount, chromosome bits (in case of GA), column mappings of the uploaded product data. It makes sure that all the required input to the optimization algorithms is well formatted and verified.
SimulationResult.java: A unit of the DTO which holds the results of a single optimization process either GA or ACO. It normally provides the amount of profit realized and a list of ProductResult objects, explaining prioritized quantity and allocations on each product.
ProductResult.java: DTO contained in SimulationResult, which contains the optimized result of an individual product. Information that it has includes the name of the product, original price, optimized price, assigned quantity, and the profit contribution calculated.
CombinedSimulationResult.java: The accumulation of results of the 2 simulations, part of this DTO, so that the result of each algorithm can be displayed on the frontend together and thus compared. It has the examples of SimulationResult through GA and Ant Colony Optimization.
SimulationStatus.java: This DTO is meant to pass the status of a simulation to the frontend in real time. It provides information on barley, e.g. simulation ID, current progress (percentage of progress), estimated time left and any error messages and validation warnings. The DTO is important in producing a responsiveness user experience in long-running simulations.

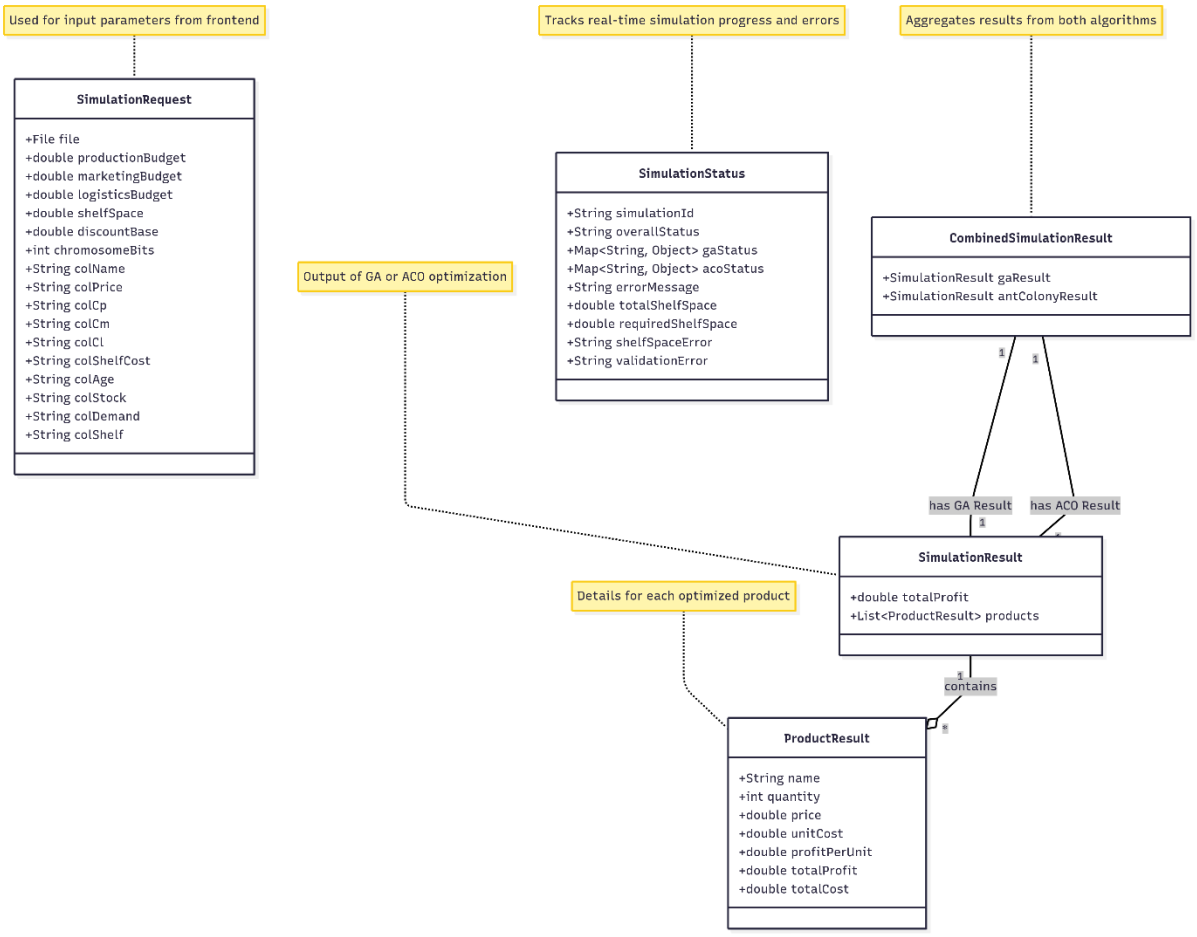

Figure 5. 5 Data Transfer Object DTO Relationships Diagram

5.3 Frontend Design

The Quantitator frontend has a user friendly and interactive interface that helps engage the backend services. It was created based on modern web technologies (HTML, CSS, and JavaScript), which guarantee a wide audience and the responsiveness of the experience in different devices. It has a user-friendly design where people can easily transfer files, set up the simulation attributes, track the progress and visualize the findings.

5.3.1 User Interface Components

The user interface consists of several main elements all of which are intended to perform certain tasks in the application workflow:

Navigation Bar: Presents a fast access to major parts of application, namely Dashboard, Simulate, and Results. It also has a separate key to use the Gemini AI chat feature and provide better support and interaction to the users.

Dashboard: This is the main page which gives long view of main metrics like projected profit, best products. This section gives direct information on the capability of the system and the possible advantage of optimization.
Simulation Panel: This is the primary interaction panel where the user posts product information (CSV / XLSX) and figuring out a range of optimization options. It comprises.
File Upload Zone: drag and drop facility to add files easily, real-time file info Plus column mapping feedback.
Parameter Input Fields: It gives the user ability to enter very essential information(s) that include production budget, marketing budget, logistical budget, base discount, operating on shelf space, and genetic algorithm specific variables like the number of chromosome bits. Every input form has a validation message that helps users to enter the correct data.
Simulation Buttons: buttons which control the starting and running of a simulation process.
Results Section: This shows the results of the GA and ACO simulation, various product level results as well as global profit amounts. This area is vital to contrasting the performance of various optimization strategies and making considerate decisions.
Gemini AI Chat Interface: A native chat window that can be used by the user to interact with the Gemini AI to receive help, explanations, or additional analysis on the quantity determining optimization.

The hierarchical structure of the user interface components is represented by figure 5.6.

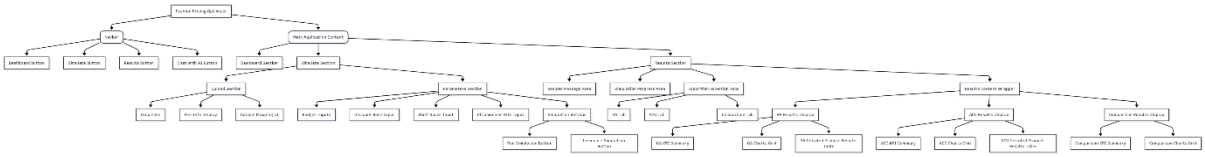

Figure 5. 6 User Interface Component Hierarchy

5.3.2 Responsive Design Implementation

The frontend has been made responsive, and it should provide an optimal viewing and interaction experience whether on a desktop computer, tablet or mobile phone. This is done by applying the style.css file with flexible grids layouts, fluid pictures and media queries. The architecture is dynamic.
To various screen sizes and orientations that would offer a consistent and user-friendly interface no matter in which device one uses.
The important features of the responsive design are:

Adaptive Layouts: The CSS Flexbox and Grid are incredibly valuable, not only because they enable flexible layouts, which means elements may adapt their size and positioning in response to the available screen, but also because they are the last major elements we still need to formulate our layouts.
Fluid Images and Media: Picture and other media assets are scaled with proportions to their containers and so do not protrude or overflow and will retain their shape on smaller screens.
Media Queries: CSS media queries are largely deployed to style various offerings depending on the nature of screen, width, and height among others. This allows to have a better control over the layout and content presentation with different breakpoints.
The touch friendly interaction: Buttons and input fields are large that they can be touched and spaces between each other are adequate so that the screen can be easily touched using the mobile devices and the web is easily used by the touchscreen device users.

This dynamic solution provides that the Quantitator can be used by a wide range of people, irrespective of the device that they want to use.

5.3.3 PDF Export Functionality

The system also has the capability of exporting a PDF to ease reporting and analysis offline. This aspect gives ability to the user to obtain a printable PDF file of the simulation output directly on the frontend. The logic behind transforming the show results into a well-formatted PDF is maintained on the client-side with the help of a pdfloms.pdf_export_feature.js script.
The process of a PDF export is normally:

Client-Side Rendering: The JavaScript library grabs the pertinent HTML information on the outcomes region of the web page.
Content Formatting: The scanned data is then formatted and made presentable so that the same is properly rendered in the PDF document such as tables, charts, etc, along with textual data.
Creation of PDFs: The formatted document is changed into a portable document format file (PDF) which the user can print or download.

This feature will make the system more usable since there will be an easy sharing and archival of the simulation results. The flowchart of the PDF export feature can be seen in figure 5.7.

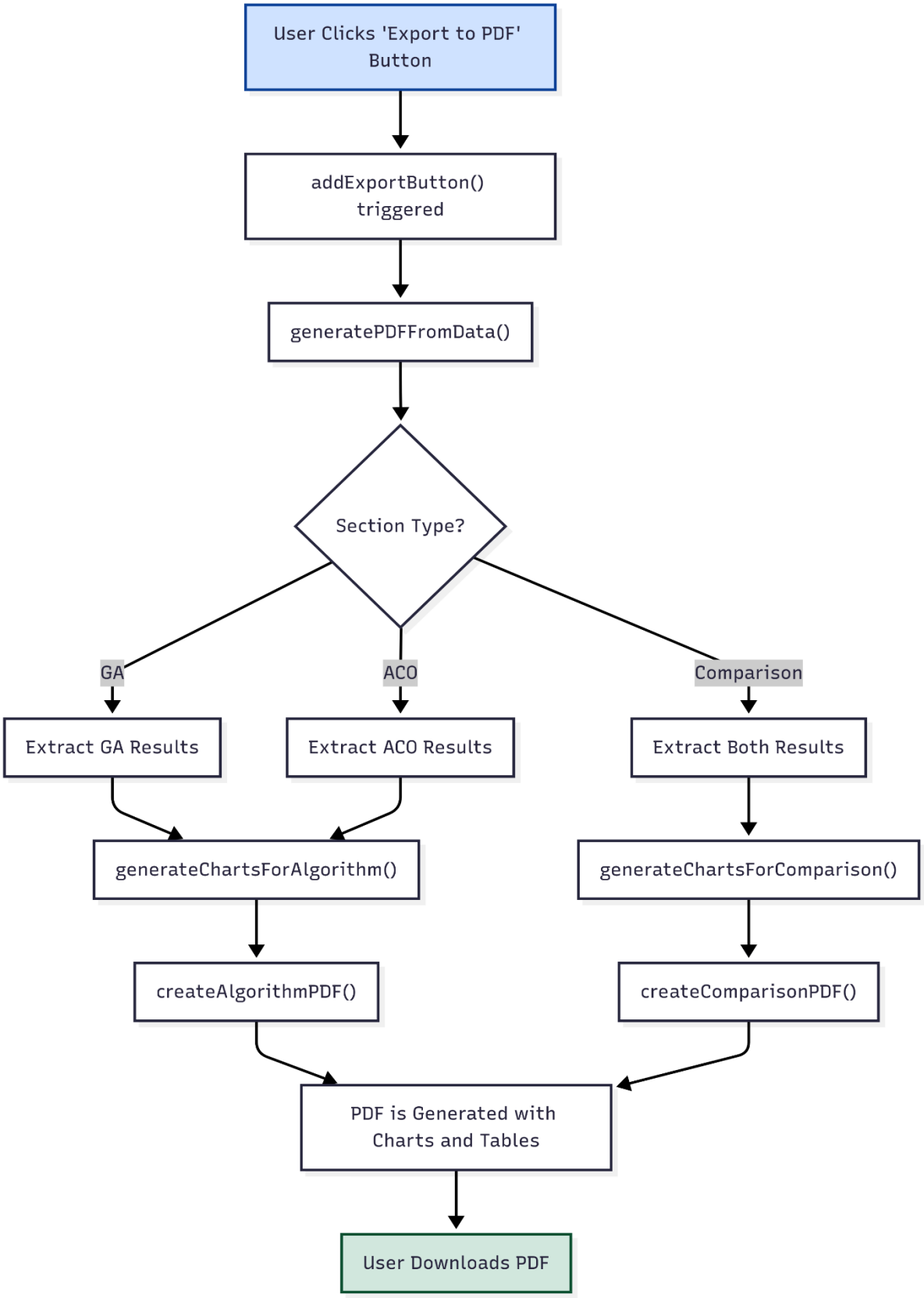

Figure 5. 7 PDF Export Functionality Flowchart

5.4 Integration Architecture

The Quantitator has a powerful integration architecture that enables it to be integrated with external services and components without any difficulties. It is essential in mobilization of dedicated features capabilities, like convoluted optimization techniques available in Python and sophisticated AI tools that come with Gemini. The integration approach makes the system modular, scalable and one that can accommodate new technologies when the need arises.

5.4.1 Python Integration Service

One of the major features of the system architecture is the possibility to connect with Python services to implement the basic optimization algorithms which are Genetic Algorithm (GA) and Ant Colony Optimization (ACO). Such interlock is realized in the backend of Spring Boot with the service wrappers: GAService.java and AntColonyService.java. The services are responsible for:

Process Management: Creation and administration of Python processes (GA.py and ant.py) by the Java app. This simply entails the generation of the suitable command-line arguments such as the path of the input files and simulation settings.
Inter-process Communication: Intercepting the results of the Python scripts including the progress updates and the complete simulation results in JSON format. This is done through reading of the standard output stream of the Python processes.
Error Handling: checking Python process error and exceptions and reporting the same to the SimulationStatusService to update the user.
Resource Management: Making sure that Python processes are efficiently executed and correctly manages system resources, such as creating a special thread pool (simulationExecutor) to run long-running simulation tasks and save the lead fashion time, wasting the least number of resources.

This is because the fact that we have chosen to use the optimization algorithms in Python, albeit the primary application being in Java, enables the system to benefit out of the deep pool of scientific computing libraries in Python as well the popularity of Python in complex algorithm implementations. A bridge between the Java backend and all these specialized Python services is the integration layer, which can enable the Java backend to both orchestrate and consume the results of these specialized services.

5.4.2 Gemini AI Integration

To enhance the user experience with interactive support, the system integrates Gemini AI directly within the frontend using JavaScript. This allows users to interact through a smart chat interface, where they can ask questions, request explanations, or seek clarification on simulation outputs and optimization workflows all without relying on a backend server.
The assimilation normally includes:

API Key Management: The gemini.api.key is securely handled on the frontend, typically through environment variables, obfuscation techniques, or client-side configuration tools during the build process. No sensitive information is hardcoded or directly exposed.
Endpoint Configuration: The gemini.api.endpoint is a configurable URL that the JavaScript frontend communicates with to send user queries to the Gemini model. It can be updated easily without server-side changes..
Request and Response Processing: All user interactions are processed in the frontend. When a user submits a message, JavaScript sends a request to the Gemini API and handles the response, which is then displayed in the application's chat interface.

The integration allows it to turn the application into an even more interactive and smarter environment in which we can receive on-demand help and make the overall experience much more embellished with the help of the application.

5.4.3 Session Management

In the Quantitator, session information can be managed mainly by the fact that their RESTful API is stateless and complies with the best practice of cloud native web applications. Each request that is being sent by the frontend to the backend has all the information required; thus, there is no need to have a session state between the backend requests. This eases the backend architecture and increases its horizontal scalability.
To control the situation of the existing simulations, a specific SimulationStatusService is used. This service also takes a role of a centralized storing point of tracking of the progress, where the result as well as any erroneous details pertaining to individual simulation runs are stored. Every simulation will be given a simulationId that will be returned to the frontend when initialized. The frontend, in turn, queries its own status by using this ID and gets updates without a need in the conventional server-side sessions at a real-time pace.
The style of managing the sessions has the following benefits:

Scalability: The API is stateless, which enables simple distributing of requests to many backend instances, because no session affinity is needed.

Reliability: Failure of a backend-instance does not halt the simulation, as each simulation runs in its own state, that is defined independently by SimulationStatusService and is uniquely identified by simulationId.
Simplicity: Simplifies the backend, as it does not make it necessary to maintain any session storage and the overhead thus incurred.

The initial interaction is stateless, although the system can implicitly use browser side user experience, e.g. local storage to store temporarily interface states, or permanent user settings. Nonetheless, critical application state particularly in case of long running processes, such as simulation, is directly controlled using the SimulationStatusService.

5.5 Performance Optimization

Optimization of performance is an essential feature of the Quantitator since it needs to be able to perform in a timely, efficient and responsive manner when it is required to service changing workloads dynamically. Due to the computational overhead of optimization algorithms, and the likely to exist large datasets, much attention has been paid in ensuring minimal response time and scalability.

5.5.1 Response Time Optimization

Response time optimization is important to achieve an efficient and pleasant user experience of working with a task that may be long-lasting, such as a simulation task. There are a number of methods that are adopted to do this:

Asynchronous Processing: The greatest optimization to achieve a quick response time is having asynchronous processing of the GA and ACO algorithms. Rather than holding up the main application thread during the execution of these algorithms, then, they are offloaded designated thread pool (simulationExecutor) located using CompletableFuture. This will allow the backend to send a simulationId back to the frontend as soon as one has been made so that the front end could get a real time update without having to wait to complete the entire simulation process.
Real-time Status Updates: SimulationStatusService is critical here, since it forces the delivery of updates between the frontend and the simulation. The Python scripts work by occasionally sending the progress (percentage time left and remaining estimated completion time) to the Java application as that process runs. This is saved and accessible at the /api/status/{simulationId} endpoint so that progress bars and progress timers can be displayed in the frontend, being able to control the expectations and design a perception of responsiveness.
Effective File Management: FileStorageService can be used to effectively manage files in terms of storing the uploaded data and retrieving them. File

operations are optimized to reduce I/O overhead by using temporary directories, and the StandardCopyOption.REPLACE_EXISTING action.
Optimized Python Script: The Python codes are optimized in terms of the algorithms they perform. Java and Python communication is simplified to reduce overhead and this progress is achieved by using outcomes and results reported through the standard output in an organised format (JSON to report results, to specify custom tags to report progress).
Lightweight DTOs: Lightweight DTOs guarantee that there is the transfer of minimum information between niches and across the network. This minimizes payload and this will make serialization/deserialization faster and also data transmission takes shorter time.

All these measures add to a very responsive system, even in case of complex and time-consuming optimization tasks.

5.5.2 Scalability Considerations

Scalability is the capacity of the system to process progressively more work or the possibility to expand the system to catch the growth. To achieve scalableness, the Quantitator has few architectural provisions, which will make it scalable:

Stateless Backend Services: The REST network interface is mostly stateless in that, each input data to the client has all the information the server requires to proceed and act on them. This enables horizontal scaling of the backend to be easy. New instances Spring Boot application can be put behind a of the Spring Boot application can be added behind a load balancer to split incoming requests, and there is no need to worry about requests with session affinity.
Simulations: to ensure that the long-running, CPU-intensive tasks of running the GA and ACO algorithms do not interfere with the main application threads, a single ExecutorService (simulationExecutor) is used to perform them. This avoids monopolization of resources that simulation task may cause and another request to the API is met with prompt response. This thread pool is configurable either in accordance with the available system resources and the anticipated concurrency.
Separation of Concerns: The proper isolations of concerns into specific layers (controller, service, and DTO) and expert services (e.g. FileStorageService, SimulationStatusService) encourage modularity. This facilitates the deployment of single items, separately. As an example, in case the file storage will become a bottleneck, it will be either optimized or replaced without impacting the business logic.
Externalization of Configuration: Sensitive content distribution and dynamic parameters, e.g. Gemini API key and server port are externalized in

application.properties. This makes it simple to change configuration without having to make changes to the code or redeploying it which is essential when it comes to deployment to new environments and scale out cases.
Python Process Management: Python integration is oriented towards management of external processes. In a very concurrent environment, resource limits, process queues, etc. would have to be carefully managed to avoid running out of resources as well as to maintain good performance.

Figure 5.8 shows the flow of the simulation process with optimization points, where performance and scaling factors have been concerned.

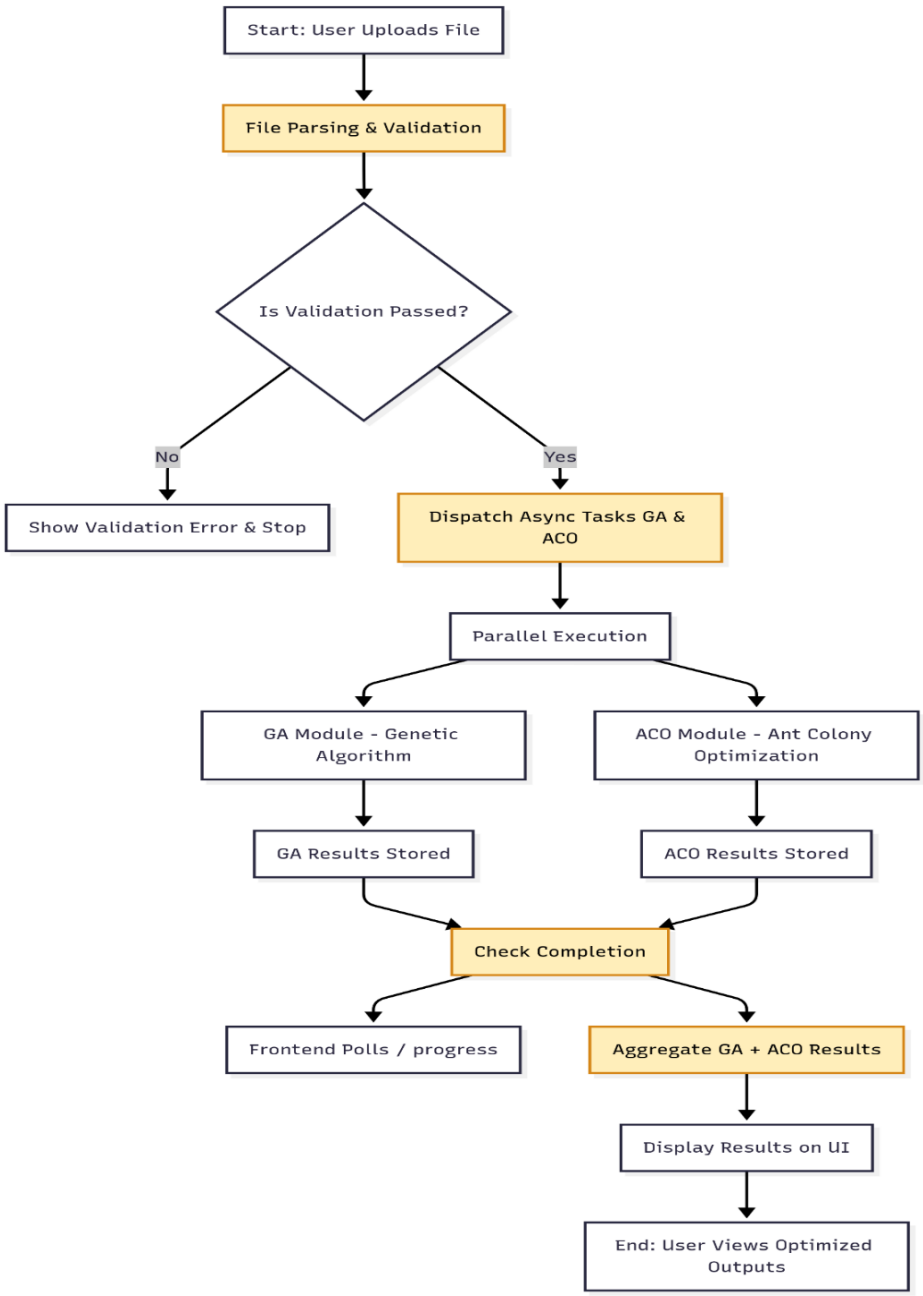

Figure 5. 8 Simulation Process Flow with Optimization Points

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Summary of Achievements

Such tools as Quantitator have enhanced the potential of fashion retail optimization greatly due to the combination of highly-developed algorithms and user-friendly interface. The most important among them are the creation of the adequately functioning system which accurately predicts product demand, optimizes the allocation of inventories, and maximizes profit margins. The main premise behind this system is that it is able to use both Genetic Algorithms (GA) and the Ant Colony Optimization (ACO) to search out complex solution spaces and find the best strategies to distribute products. One of the key areas that have been greatly emphasized through this improvement is increasing the user experience by the ability of it to offer real-time feedback, thorough error processing, and input verification among others, making the system not only great but easy and user friendly.

6.2 Evaluation of System Performance

The enhanced system has been reviewed on some of its critical dimensions in terms of responsiveness of the user-interface and the backend processing, and effectiveness of the optimization algorithms. These tests and reviews all concur with the system being effective in the provision of precise and prompt results as well as providing a pleasant users experience.

6.2.1 UI Responsiveness Metrics

The usability aspect of the application has been greatly enhanced so that it offers a very responsive and informative interface. The in-process tracking of GA and ACO algorithms became an inherent part and shows the percentage progress and approximate time left. It is done by dynamic updates reflecting real progress in the algorithm and visual progress bars to get direct feedback. Status messages on different scenarios such as completion, invisible solutions, shelf space errors and general errors are also clear and user friendly as provided in the system. This increased graphical feedback, and message categorization makes using the interface both more insightful and less frustrating, even in simulations that run far and long.

6.2.2 Backend Processing Efficiency

Its backend, which is constructed on the basis of Spring Boot and uses Python scripts to run the optimization routines has high processing performance. SimulationStatusService is responsible in controlling simulation status and checking special cases such as shelf space before any simulations are pre-validated.

invisible solutions. Error detection, reporting and an improved handling of the Python script output has been improved (it will also support parsing of progress messages) in the GAService and in the AntColonyService. SimulationController also supports checking of the available shelf space before initiating simulation and initial time estimate parameters setting, avoiding useless calculations, and increasing overall performance. The python code themselves (GA.py and ant.py) have been tuned such that they update the progress in real time and the Java backend parses that data and displays it to the frontend which results in a smooth flow of information to the frontend.

6.2.3 Optimization Algorithm Effectiveness

The project makes use of two main optimization algorithms Genetic Algorithm (GA) and Ant Colony Optimization (ACO). The two algorithms aim at identifying optimal quantities of products that would yield profit maximization by taking into consideration several constraints of budgetary nature like production budget and marketing budget as well as logistics budget and shelf space. The success of these algorithms is proved by the possibility to converge on solutions, which fulfil the complex interdependence between the prices of products, demand, and available resources. The GA.py program is a feature-rich robust GA that has chromosome representation, decoding, repair procedures (brings about demand and budget constraint), tournament selection, one-point crossover and mutation functions. This script ant.py is an ACO implementation where solutions are developed by ants which, depending on information about pheromone trails and heuristics, choose product quantities, with pheromones updates according to the quality of the solution. Both algorithm have ways to deal with edge cases, like

hindrances to infeasible inputs in the form of invisible solutions (zero or negative profit) and shelf space validation, so that the algorithms do not run. The constant refinement of these algorithms such as the improvement in the error handling and excellence reporting also incurs to the overall efficiency and decision of their being an optimum combination of such algorithms.

6.3 Limitations of the Current Implementation

Despite the significant advancements, the current implementation has certain limitations that can be addressed in future iterations. One primary limitation lies in the static nature of the input parameters once a simulation begins. While the system allows for extensive configuration prior to execution, dynamic adjustments or real-time parameter tuning during an ongoing simulation are not supported. This means that if market conditions or resource availability change mid-simulation, the current run would need to be terminated and re-initiated with updated parameters, which can be inefficient for highly volatile environments.

Another limitation is the reliance on pre-defined column mappings for input data. Although the system provides flexibility in mapping user-uploaded CSV/Excel columns to internal parameters, any deviation from the expected data types or presence of missing critical columns can lead to errors. While enhanced error handling has been implemented to report such issues, the system does not currently offer automated data cleansing or intelligent suggestions for correcting malformed inputs, requiring manual intervention from the user.

Furthermore, the current error reporting, while improved, primarily focuses on Python script errors, missing dependencies, and JSON parsing issues. While comprehensive for technical debugging, the user-facing error messages could be further refined to provide more actionable insights for non-technical users. For instance, an

error indicating an “Invisible Solution” currently suggests parameter adjustment or data review, but it could be enhanced with more specific guidance based on the input data characteristics. Similarly,

the “Shelf Space Validation” error, while preventing unnecessary runs, could benefit from suggestions on how to modify product quantities or available shelf space to resolve the issue. The system’s current architecture, while robust, might also face scalability challenges with extremely large datasets or a very high number of concurrent simulations, as the Python scripts are executed as separate processes. While not explicitly tested for extreme loads, this could become a limitation in a production environment with massive data volumes.

6.4 Recommendations for Future Enhancements

To further enhance the capabilities and usability of the Fashion Optimization Project, several key areas for future development have been identified. These recommendations aim to address the current limitations and introduce new functionalities that would make the system even more powerful and adaptable to diverse business needs.

6.4.1 Additional Optimization Algorithms

In spite of the fact that the application of both Genetic Algorithms (GA) and Ant Colony Optimization (ACO) algorithms offers quite a solid basis to address the fashion optimization issue, significant enhancements in terms of the solution quality and performance efficiency could be achieved by searching and incorporating some other metaheuristic optimization algorithms during the course of work. The Particle Swarm Optimization (PSO), Simulated Annealing (SA), or even hybrid ones, mixing the parts of other algorithms, might be explored. All these algorithms bring in their own strengths to the game of traversing search space and may actually outperform on certain types of datasets or constraints. As an example, one optimization problem can be solved more effectively with PSO (e.g., continuous problems) whereas another optimization problem can be solved more effectively in SA (e.g. local optima avoidance). It would be essential to compare these newly-developed algorithms with each other and the currently-available GA and ACO on several real-world datasets to see whether the new methodologies are effective and in which scenarios the novel methodology presents any clear advantage. Moreover, the new system, since it would be modular with the algorithmic running being performed by Python scripts, would make integration of new algorithms easier without having to perform changes on the core Java backend.

6.4.2 Enhanced AI Capabilities

The current system primarily focuses on optimization based on predefined parameters and historical data. Future enhancements could involve integrating more advanced Artificial Intelligence (AI) capabilities to enable predictive analytics and adaptive optimization. This could include:

- Machine Learning for Demand Forecasting: Instead of relying solely on historical average expected demand (D_e), machine learning models (e.g., time series analysis, neural

networks) could be employed to forecast future demand with greater accuracy, taking into account external factors such as seasonality, promotions, and economic indicators. This would provide more dynamic and realistic input for the optimization algorithms.

- Reinforcement Learning for Adaptive Parameter Tuning: The optimization algorithms (GA and ACO) rely on various parameters (e.g., population size, mutation rate, pheromone evaporation rate). Reinforcement learning agents could be trained to dynamically adjust these parameters during simulation based on the observed performance, leading to faster convergence and better solution quality. This would reduce the need for manual tuning and make the system more autonomous.

- Natural Language Processing (NLP) for User Input: Implementing NLP capabilities could allow users to provide more natural and less structured input for defining constraints or objectives. For example, instead of manually mapping columns, users could describe their data fields in plain language, and the system could intelligently infer the mappings. This would significantly improve the user experience and reduce the learning curve for new users.

- Explainable AI (XAI) for Decision Support: To increase user trust and understanding, XAI techniques could be incorporated to explain *why* a particular solution was chosen by the algorithms. This could involve visualizing the impact of different constraints, highlighting the most **influential factors in the optimization process, or providing human-readable justifications for the recommended** product quantities. This would transform the system from a black-box optimizer into a more transparent decision-support tool.

6.5 Conclusion

The Enhanced Fashion Optimization Project is a major object implemented that is to transfer advanced methods of computation to practical work and commercial aspect of fashion industry. Combining Genetic Algorithms with Ant Colony Optimization and the emphasis on the user experience via enhanced responsiveness, extensive error and input validation, the system offers an effective means of streamlining the process of distributing the products and ensuring maximum profitability. The success that was recorded in real-time progress monitoring, comprehensive error reporting, as well as effective backend processing highlight the current realization of the system.

Nevertheless, similar to every complex system, there are limitations inherent to it, especially in terms of the dynamic parameter modification during the simulation and sophisticated data cleaning. These constraints, coupled with the possible presence of scalability issues under high load levels, are the points to be developed in the future. Suggested future improvements along the proposal of implementation of more optimization algorithms, integration of more advanced AI features, including machine learning to predict demand and reinforcement learning to tune various parameters, and development of mobile application provide a strong plan on how to advance the project. These suggested improvements will enable the system to be smarter, more flexible and available, and more firmly established as an indispensable aid to strategic decision-making in fashion retail, an environment that is very mutable.

6.6 References

• Conversely, stock levels directly suppress demand when inventory signals stagnation. Wee and Yu (1999, p. 475)

• C.L. Chen et al. An application of genetic algorithms for flow shop problems European Journal of Operational Research (1995)

• J. Gottlieb, L. Paulmann. Genetic algorithms for the fixed charge transportation problem. Proceedings of the 1998 IEEE...

• Y. Nagata, S. Kobayashi. Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem,...

• J.E. Beasley et al. A genetic algorithm for the multidimensional knapsack problem Journal of Heuristics (1998)

• Z. Degraeve et al. A mixed integer programming model for solving a layout problem in the fashion industry Management Science (1998)

• Z. Degraeve, W. Gochet, R. Jans. Alternative formulations for a layout problem in the fashion industry. Research Report (1998)

• J. Martens, J. Vanhoucke. A genetic algorithm for the layout problem in the fashion industry. European Journal of Operational Research (2004)

• J. Martens, J. Vanhoucke. A genetic algorithm for the layout problem in the fashion industry. Research Report (2002)