Multi-Meal, Multi-Constraint Recommender System to Optimize Grocery Budget and Waste

by

Andrei Mereuta

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on 27th June 2025 at 11:00 AM

Project duration: November 2024 – June 2025

Thesis committee:

Dr. Anna L.D. Latour,	TU Delft, supervisor
Dr. Neil Yorke-Smith,	TU Delft, supervisor
Catalin Ștefan Cernat,	Picnic, supervisor

Author contact:
student number: 5230527

Delft University of Technology

"Nah, I'd win."

– Gojo Satoru, *Jujutsu Kaisen*

Multi-Meal, Multi-Constraint Recommender System to Optimize Grocery Budget and Waste

Master's Thesis in Computer Science

Algorithmics group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Andrei Mereuta

23rd June 2025

Author
Andrei Mereuta

Title
Multi-Meal, Multi-Constraint Recommender System to Optimize Grocery Budget and Waste

MSc presentation
27th June 2025

Graduation Committee

Dr. Anna L.D. LATOUR	Delft University of Technology
Dr. Christoph LOFI	Delft University of Technology
Dr. Neil YORKE-SMITH	Delft University of Technology

Abstract

Grocery delivery company Picnic has identified affordable meal planning, especially in the context of recipe-based shopping, as an ongoing challenge faced by its customers. While recipes enhance customer experience and operational efficiency, Picnic currently lacks an algorithmic system to recommend recipes in a cost-effective way. This research addresses this gap by proposing a scalable optimization approach that minimizes the total cost of grocery products across a set of selected and recommended recipes.

To address this challenge, the paper formulates the problem as a Mixed Integer Linear Program (MILP), chosen for its ability to guarantee a globally optimal solution. The MILP model selects a combination of recipes and corresponding products that promote ingredient reuse and bulk purchasing to minimize overall cost. To complement this, the study implements two meta-heuristic models: a standalone Genetic Algorithm (GA) and a hybrid GA+MILP model, where GA selects recipes and MILP optimally assigns products. The optimization process also incorporates real-world constraints such as minimizing food waste and ensuring cuisine consistency.

Experimental results show that the MILP formulation consistently achieves substantial cost savings compared to a naive baseline, which selects the same set of recipes but does not optimize product choices across them. Gurobi outperforms CPLEX in solve time while maintaining identical solution quality. The standalone GA yields near-optimal solutions in seconds, while the hybrid GA+MILP model improves accuracy further, albeit with increased computational cost. When constraints such as waste reduction are added, the system remains effective, with one multi-objective variant reducing waste at minimal cost increase.

The findings confirm that cost-aware recipe recommendation is feasible, efficient, and adaptable. The proposed system offers a foundation for future extensions toward sustainability, personalization, and real-world deployment at Picnic.

Acknowledgements

This thesis marks the final milestone in my journey toward obtaining a Master's degree in Computer Science at TU Delft. It has been a challenging, rewarding, and transformative experience, and I am sincerely grateful to everyone who supported me along the way.

I would like to express my general gratitude to all those who, in one way or another, contributed to this journey—through advice, encouragement, collaboration, or simply by being present during both the highs and the lows.

I am especially thankful to my academic supervisors, Professor Anna L.D. Latour and Professor Neil Yorke-Smith, for agreeing to supervise this thesis and for their thoughtful guidance and support throughout the process. I also extend my sincere thanks to Professor Christoph Lofi, the third member of my thesis committee, for dedicating time and effort to reviewing and evaluating my work.

This project would not have been possible without the support of Picnic, the company that provided me with the topic and the opportunity to apply my skills to a real-world problem. I am grateful not only for this project but also for the support I received from Picnic long before the thesis began. I feel incredibly fortunate to have met and worked with such inspiring people there. In particular, I would like to thank Catalin Ștefan Cernat, who supported me throughout the thesis with countless ideas and insights that shaped this project, and Phil Werli, who was always available to offer help and perspective when needed.

Finally, I would like to thank my friends and family for being my emotional anchor. Your encouragement and belief in me have been a constant source of strength and motivation, and I could not have completed this journey without you.

Andrei Mereuta

the Hague, The Netherlands
23rd June 2025

Contents

Chapter 1

Introduction

Picnic is one of the largest online supermarkets in the Netherlands, operating entirely through a digital infrastructure that supports everything from inventory management to customer interaction. One feature within its platform is a recipe-based shopping experience, where customers can choose full meals directly through the app. While this functionality enhances convenience and personalization, it also introduces a challenging computational problem with practical relevance.

At its core, recipe recommendation in an online grocery setting involves selecting combinations of meals and corresponding products in a way that aligns with both user preferences and operational efficiency. From a scientific perspective, this opens up a rich optimization problem. The search space—formed by all possible recipe combinations and their compatible product selections—is large and combinatorial. A naive approach that evaluates each combination individually is computationally infeasible, especially as the number of recipes, ingredients, and constraints grows. Moreover, the structure of the problem lends itself naturally to extension: one can incorporate additional objectives (e.g., minimizing food waste) or constraints (e.g., dietary requirements, cuisine consistency), further increasing complexity.

This work approaches the problem from an optimization and decision-making standpoint. It investigates whether modern algorithmic techniques can be used to efficiently recommend recipes and select associated products in a way that minimizes total cost. In particular, we developed a *Mixed Integer Linear Programming (MILP)* formulation to exploit ingredient reuse and bulk purchasing—both of which are key levers for reducing grocery expenses. In addition to exact methods, we implemented a *Genetic Algorithm (GA)* to explore the space of recipe combinations more flexibly and with lower computational overhead.

Building on the strengths of both approaches, this work also proposes a hybrid model that combines the exploratory power of GA with the precision of MILP. In this setup, the GA operates over sets of recipes, while a MILP solver is used internally to evaluate each candidate solution by computing the optimal product assignment. This hybrid approach preserves the flexibility and diversity of GA, while leveraging MILP's exactness for product-level optimization. It is particularly well-suited for cases where users want varied recommendations without sacrificing cost efficiency.

The primary aim is to evaluate the practical feasibility of this optimization approach under realistic conditions. The model must scale to thousands of recipes and products, handle real-world constraints, and provide solutions within acceptable computational time. These requirements offer a meaningful benchmark for studying the trade-offs between solution quality and performance in large-scale combinatorial optimization problems.

By addressing this problem, the paper contributes to the broader field of applied optimization and algorithmic decision-making, demonstrating how theoretical tools can be used to tackle practical challenges in digital commerce environments.

1.1 Motivation

Recipe-based meal planning promises to add value for both customers and the platform. According to Picnic's internal analysis, offering customers a way to plan meals around complete recipes improves shopping convenience, boosts satisfaction, and increases the likelihood of repeat purchases.

However, Picnic currently lacks an algorithmic system for recommending recipes that balance cost, variety, and practicality. Without an optimization layer, customers must select meals manually, which often leads to duplicated ingredients, underutilized products, and missed opportunities for savings through shared items or bulk purchases. This results in unnecessary spending and inefficiencies in shopping behavior. Furthermore, customers receive no guidance in aligning their choices with personal preferences such as cuisine type, dietary restrictions, or existing selections, limiting the potential value of the recipe feature.

From an operational perspective, these inefficiencies also reduce opportunities to streamline logistics, manage stock more effectively, and reduce food waste. Introducing algorithmic support for recipe planning is therefore not only beneficial to users, but also strategically important for the platform.

1.2 Research Question

The main research question guiding this thesis is:

> *How can we recommend recipes in a way that minimizes the total cost of products, while remaining computationally efficient and adaptable to real-world constraints?*

To address this question, we consider the following sub-questions:

- **RQ1:** To what extent can a Mixed Integer Linear Programming approach minimize the total cost of recipe recommendations while remaining computationally feasible?
- **RQ2:** How do different MILP solvers (e.g., CPLEX and Gurobi) compare in terms of solve time, solution quality, and scalability?
- **RQ3:** Can a Genetic Algorithm, alone or combined with MILP, provide near-optimal cost-efficient solutions more efficiently than standalone MILP solvers, and under what conditions is it preferable?
- **RQ4:** How can waste be reduced as a secondary objective, and what is the impact of strategies like multi-objective optimization and cuisine-based selection on cost, waste and performance of the model?

1.3 Thesis Outline

This thesis is organized as follows:

- **Chapter 2 – Background:** Describes Mixed Integer Linear Programming and Genetic Algorithms, and explains the choice of MILP solvers (CPLEX and Gurobi) used in this research.
- **Chapter 3 – Problem Statement:** Defines the recipe optimization problem addressed in this research, outlines the practical motivation, formalizes the optimization goals, and introduces the data models used to structure recipes, ingredients, and products.
- **Chapter 4 – Related Work:** Reviews prior research on recipe recommendation and meal planning optimization. It highlights existing solutions, their limitations, and the research gap this work aims to address.
- **Chapter 5 – Methodology:** Describes the design and implementation of the proposed optimization models. This includes the formulation of the MILP model, its extension to handle constraints, the development of a Genetic Algorithm as a heuristic alternative, and a hybrid GA+MILP model that combines recipe-level exploration with exact product-level optimization.
- **Chapter 6 – Experimental Setup:** Outlines the experimental configurations used to evaluate the system. It describes the computational environment, dataset selection, test scenarios, solver comparisons, and constraint-based variants.
- **Chapter 7 – Results:** Presents and analyzes the experimental results. It provides insights into the model's effectiveness, runtime performance, and trade-offs between cost, waste, and scalability in relation to the research questions.

- **Chapter 8 – Discussion and Limitations:** Critically analyzes the results and reflects on the limitations of the proposed approach. It discusses the areas where the system could be improved or extended.
- **Chapter 9 – Conclusions and Future Work:** Summarizes the key findings of the research, answers the research questions, and outlines possible directions for future research and development.

Chapter 2

Background

This chapter introduces the foundational concepts and tools required to understand the methods and experiments presented in this thesis.

Section 2.1 presents Mixed Integer Linear Programming, including its general formulation, practical applications, and techniques for handling logical constraints. It also introduces the solving methods used in this research, with a focus on the CPLEX and Gurobi solvers.

Section 2.2 covers Genetic Algorithms, describing their key components, initialization strategies, termination criteria, and the balance between exploration and exploitation. The section also discusses parameter sensitivity and provides pseudocode for the implemented GA framework.

2.1 Mixed Integer Linear Programming

This introduction to Mixed Integer Linear Programming is loosely based on the tutorial by Smith and Taskin [2008], to which we refer the reader for a more details. MILP is a widely-used mathematical optimization technique that models problems using linear relationships, where some decision variables are constrained to take integer values. It is a powerful tool for solving a broad range of discrete optimization problems, including scheduling, routing, and resource allocation.

2.1.1 General Formulation

A general MILP problem is formulated as:

$$\begin{aligned} \min_{x} \quad & c^\top x \quad \text{(Objective Function)} \\ \text{s.t.} \quad & Ax \leq b \quad \text{(Constraints)} \\ & x_i \in \mathbb{Z} \quad \forall i \in I \quad \text{(Integer Variables)} \\ & x_j \in \mathbb{R} \quad \forall j \notin I \quad \text{(Continuous Variables)} \end{aligned}$$

In this formulation, x is the vector of decision variables, c is the coefficient vector of the objective function, A is the constraint matrix, and b is the constraint bound vector. The objective function $c^\top x$ defines the goal of the optimization; in this thesis, we focus on the minimization case, where the objective is to minimize the total cost. The constraints $Ax \leq b$ represent the linear relationships and limitations imposed by the problem. The decision variables x include both continuous variables $x_j \in \mathbb{R}$ and integer variables $x_i \in \mathbb{Z}$, which introduce combinatorial complexity. MILP extends the classical Linear Programming (LP) model by incorporating these integrality constraints, which makes the problem notably harder to solve.

2.1.2 Handling Logical Implications in MILP

Many real-world optimization problems require the modelling of logical conditions such as "if a product is selected, then at least a minimum quantity must be assigned," or "a recipe must be considered only if all its ingredients are fulfilled." These types of logical implications cannot be directly expressed in linear programming and must be reformulated using auxiliary variables and constraints. Two common techniques for expressing such logic in MILP models are the ***Big-M method*** and the use of a small positive constant ε.

Big-M Constraints The Big-M method introduces an arbitrarily large constant M to control the activation or deactivation of continuous variables based on binary decisions [Cococcioni and Fiaschi, 2021]. It is often used to model implications of the form:

$$y \leq Mz, \quad y \geq \varepsilon z, \quad z \in \{0, 1\}, \quad y \in \mathbb{R}_{\geq 0} \tag{2.1}$$

In this formulation, when $z = 0$, the constraint enforces $y \leq 0$, effectively deactivating the variable y. When $z = 1$, the constraint bounds y between ε and M. This construct is commonly used to enforce selection-dependent behavior, such as allocating nonzero quantities only when a product is chosen or an ingredient is active.

Choosing an appropriate value for M is critical: if M is too small, feasible solutions may be inadvertently excluded; if M is too large, it can introduce numerical instability and degrade solver performance [Vielma, 2015, Wolsey and Nemhauser, 1999]. In practice, M is selected based on known upper bounds of the dependent variable.

Small-ε Activation Constraints To enforce that a continuous variable y becomes strictly positive when its associated binary variable z is active, a small constant $\varepsilon > 0$ is introduced:

$$y \geq \varepsilon z \tag{2.2}$$

This is useful to break symmetry or avoid degenerate solutions where $z = 1$ but $y = 0$, which may be theoretically valid but practically undesirable (e.g., selecting a product but allocating zero quantity). The value of ε is typically small (e.g., 10^{-3} or 10^{-5}) and must be chosen carefully to avoid infeasibility due to rounding or precision limits in floating-point solvers [Conforti et al., 2014].

2.1.3 Practical Applications

Mixed Integer Linear Programming has proven to be highly versatile across a wide range of industrial and academic domains. In supply chain optimization, MILP models are used to determine optimal production schedules, inventory levels, and distribution plans while minimizing overall cost [Rodriguez et al., 2014, Gumus et al., 2009]. These problems often involve complex trade-offs between production constraints and storage limitations, and require modeling large-scale systems with numerous interdependent variables.

In scheduling and timetabling, MILP supports applications such as job-shop scheduling and university exam planning [Gestrelius et al., 2017, Andersson et al., 2015]. These tasks present challenges related to resource contention, precedence constraints, and fairness. Techniques such as time-indexed formulations and disjunctive constraints are frequently employed to capture these complexities.

Vehicle routing problems, common in logistics and delivery services, use MILP to determine efficient routes for a fleet of vehicles [Dondo and Cerdá, 2006, Madankumar and Rajendran, 2019]. These problems are often NP-hard and involve additional constraints like time windows, fuel limitations, or driver regulations, making solution quality and runtime critical.

In the energy sector, MILP is used for power grid management tasks, including generation scheduling, load balancing, and maintenance planning [Liu et al., 2022, Zaree and Vahidinasab, 2016]. The main challenges involve capturing nonlinear behaviors using piecewise-linear approximations and balancing multiple competing objectives such as cost, stability, and reliability.

A domain more directly related to this thesis is diet and meal planning, where MILP is applied to select food combinations that meet nutritional requirements while minimizing cost or waste [Buisman et al., 2019, Sklan and Dariel, 1993]. These models resemble the one developed in this thesis, as they involve a fixed set of items (recipes or foods), quantitative constraints (nutrient or ingredient weights), and cost minimization. However, unlike previous work that typically focuses on individual meals or nutritional adequacy, this thesis also introduces recipe recommendation as part of the optimization process, expanding the decision space and introducing combinatorial aspects typically handled by heuristics such as Genetic Algorithms.

Overall, MILP's adaptability to domain-specific constraints and objectives makes it a powerful tool. However, challenges such as scalability, solver efficiency, and solution interpretability remain critical, motivating the hybrid and heuristic approaches explored in this thesis.

2.1.4 Solving Techniques: Branch-and-Bound

MILP problems are NP-hard, meaning that solving them exactly requires exponential time in the worst case. However, modern solvers implement advanced techniques that enable solving large MILPs efficiently in practice. The foundational technique among these is *Branch-and-Bound* [Clausen, 1999, Huang et al., 2021].

Overview

Branch-and-Bound (B&B) is an exact algorithm that systematically explores the solution space of an MILP. It operates by recursively dividing the problem into subproblems (branching) and eliminating parts of the search space that cannot yield better solutions (bounding).

Steps in Branch-and-Bound

1. **Relaxation**: Solve the LP relaxation of the MILP (ignoring integer constraints).

2. **Branching**: If the solution contains fractional values for integer variables, create two subproblems with tighter bounds (e.g., $x_i \leq \lfloor v \rfloor$ and $x_i \geq \lceil v \rceil$).

3. **Bounding**: Use the solution to prune branches. If the LP relaxation gives a worse objective than the current best integer solution (incumbent), discard the subproblem.

4. **Termination**: Continue until all branches have been explored or pruned.

This method guarantees that the optimal solution will be found, although the number of branches can grow exponentially. To improve performance, solvers employ additional enhancements such as:

- **Cutting Planes**: Adding linear constraints to tighten the LP relaxation.

- **Heuristics**: Finding good feasible solutions quickly to improve pruning.

- **Presolve Techniques**: Simplifying the problem before optimization begins.

- **Parallelism**: Solving multiple subproblems concurrently.

Branch-and-Bound forms the backbone of commercial MILP solvers such as CPLEX and Gurobi, which incorporate decades of engineering and optimization theory.

2.1.5 CPLEX

IBM ILOG CPLEX is a commercial optimization solver known for its high performance in solving linear, integer, and quadratic programming problems. CPLEX uses sophisticated branching strategies, presolve routines, and cutting-plane generation to accelerate convergence [IBM CPLEX, 1987]. It supports Python through the `docplex` library, which provides an intuitive interface for modelling optimization problems.

CPLEX offers fine-grained control over solving parameters such as node selection strategies, emphasis settings (e.g., feasibility vs. optimality), and parallelism levels. It also provides detailed logs and solution pools for analyzing alternate optima or near-optimal solutions.

CPLEX's built-in conflict refiner helps identify infeasibility causes in models, which can be especially helpful during iterative model design.

2.1.6 Gurobi

Gurobi is another state-of-the-art MILP solver, frequently benchmarked alongside CPLEX [Meindl and Templ, 2012, Jablonskỳ et al., 2015]. It offers competitive performance, supports multi-threaded parallelism, and provides detailed diagnostics during optimization.

Gurobi is known for its strong performance on large-scale problems and offers various tuning tools for parameter optimization. It supports a wide range of programming interfaces including `Python`, `Java`, `C++`, and`MATLAB`, with a consistent modelling interface.

Gurobi's Python API is tightly integrated with `NumPy` and `Pandas`, making it especially useful for data-driven optimization workflows, such as recipe-based product selection.

2.2 Genetic Algorithms

Genetic Algorithms are a class of population-based, stochastic optimization methods inspired by the principles of natural selection and evolution [Forrest, 1996]. They are especially useful for solving complex combinatorial problems where exact methods like MILP become computationally expensive. GAs search for approximate solutions by evolving a population of candidate solutions over successive generations.

2.2.1 Key Components

A typical Genetic Algorithm operates over a population of individuals called *chromosomes*, each representing a potential solution to the problem. The main components are as follows:

- **Chromosome Representation**: A chromosome encodes a candidate solution.
- **Selection**: Parents are chosen from the current population based on their fitness. Common selection strategies include tournament selection, roulette wheel selection, and rank-based selection. Selection promotes fitter individuals while maintaining diversity.
- **Crossover**: Pairs of parents produce offspring by exchanging parts of their chromosomes. One-point, two-point, and uniform crossover are standard techniques.
- **Mutation**: Random changes are introduced into offspring to maintain genetic diversity.
- **Fitness Function**: Evaluates how well a chromosome performs in solving the problem.

2.2.2 Algorithm Dynamics

This section discusses key dynamic aspects of Genetic Algorithms, including how the population is initialized, when the algorithm terminates, how it balances exploration and exploitation, how sensitive it is to parameter tuning, and its overall workflow.

The initial population is typically generated randomly, ensuring sufficient diversity to explore the solution space. In some cases, heuristic-based or partially greedy initialization can be used to bias the search toward promising regions.

A Genetic Algorithm can be terminated in several ways, depending on the problem and performance goals. One common approach is to stop the algorithm after a fixed number of generations, which ensures a predictable runtime. Alternatively, the algorithm may terminate early if no improvement in the best fitness score is observed over a number of consecutive generations, indicating convergence. Another strategy involves stopping once a satisfactory or optimal fitness level has been reached.

GAs must balance *exploration* (diversity and searching new areas) and *exploitation* (refining known good solutions). Crossover and selection favor exploitation, while mutation introduces randomness for exploration. Striking the right balance is critical to avoiding premature convergence or excessive randomness.

The performance of a Genetic Algorithm is highly dependent on its *hyperparameters*. These include the population size, which determines the diversity and search capacity; the mutation rate, which controls the degree of randomness introduced in each generation; and the crossover probability, which influences how often offspring inherit traits from both parents. Selection pressure also plays a role in how strongly better-performing individuals are favored. Poorly tuned parameters can result in premature convergence or inefficient exploration of the solution space. For this reason, parameter tuning is often necessary to balance exploration and exploitation effectively.

To illustrate the overall structure and flow of a Genetic Algorithm, we provide the following pseudocode:

Algorithm 1 Basic Genetic Algorithm

```
1: Initialize population $P$ with $N$ individuals
2: for each generation do
3:     Evaluate fitness for all individuals in $P$
4:     Select parent pairs from $P$ based on fitness
5:     Create offspring via crossover
6:     Apply mutation to offspring
7:     Evaluate fitness of offspring
8:     Select individuals for the next generation (e.g., via elitism)
9:     $P \leftarrow$ new population
10: end for
11: return best individual found
```

Chapter 3

Problem Statement

This chapter defines the core problem addressed in this research. We clarify the optimization objectives, motivate the need for diversity in recipe recommendations, and explain why solving this problem is non-trivial in an operational context like Picnic's. We conclude the chapter with a formal description of the data structures used in our models, followed by a placeholder for the mathematical problem formulation.

3.1 Optimization Objectives

The goal of this research is to develop a system that recommends a set of recipes in a way that minimizes the total grocery cost, respects customer preferences such as cuisine type or dietary restrictions, delivers recommendations quickly, and reduces unnecessary product waste.

These requirements introduce several challenges: the solution must be cost- optimal, adaptable to diverse constraints, scalable to a large recipe and product database, and responsive to real-world customer needs.

3.2 Need for Diverse Recommendations

We address the recipe recommendation task using two complementary optimization approaches: Mixed Integer Linear Programming and Genetic Algorithms.

MILP provides globally optimal results and guarantees reproducibility: the same input always leads to the same output. This property ensures transparency and consistency. However, it becomes a limitation when a customer dislikes the suggested recipes but does not modify their inputs—MILP will keep returning the same result.

To overcome this rigidity, we incorporate Genetic Algorithms. GAs introduce controlled randomness, enabling the generation of multiple high-quality alternatives from the same input. Their probabilistic nature allows us to explore a wider solution space and generate diverse yet cost-effective recommendations. By combining these two methods, we balance optimality with adaptability.

3.3 Picnic Recipe Data Models

The data used in this research originates from the Picnic dataset, which includes detailed information on recipes, their corresponding ingredients, and the selling units (products) available for purchase.

Recipes and Ingredients Each recipe is defined as a collection of ingredients, where each ingredient has a specified weight requirement—the amount necessary to prepare the dish. Formally, a recipe r is represented as:

$$r = \{(i, w_{r,i}) \mid i \in I_r\} \tag{3.1}$$

where I_r is the set of ingredients required for recipe r, and $w_{r,i}$ is the required weight of ingredient i in recipe r.

Ingredient-to-Selling Unit Mapping Each ingredient can be fulfilled by one or more corresponding products, referred to as *selling units*. A selling unit represents a specific product available in the Picnic store. These can differ not only in type (e.g., cow milk, oat milk, almond milk), but also in packaging or quantity (e.g., 0.5 liter, 1 liter, or 1.5 liter). This mapping is formalized as:

$$S_i = \{s \in \mathcal{S} \mid s \text{ can fulfill ingredient } i\}, \quad \forall i \in I \tag{3.2}$$

where S_i is the set of selling units that can fulfill ingredient i, and each selling unit s_k corresponds to a specific product variant listed in the Picnic catalog.

Selling Units (Products) A selling unit s represents an individual product that can be purchased. Each selling unit has associated properties:

$$s = (w_s, p_s) \tag{3.3}$$

where w_s is the weight (or number of pieces) per unit of the product, and p_s is its price.

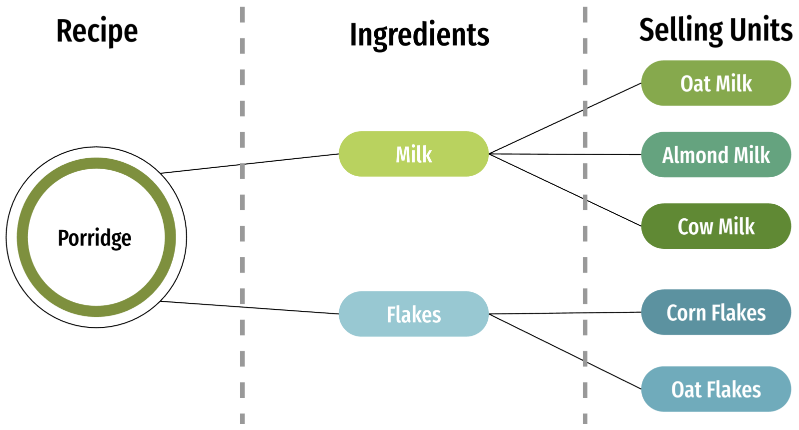

Figure 3.1: Simplified Example of Recipe Model

These structured data representations form the foundation of our optimization models. As shown in Figure 3.1, solid lines between the recipe and its ingredients denote which ingredients belong to the recipe. Solid lines between ingredients and selling units indicate which products can potentially fulfill each ingredient requirement.

Optimization Goal Illustration Figure 3.2 visualizes the algorithm's goal: selecting selling units that minimize total grocery cost across multiple recipes. The red edges indicate the selling units chosen by the algorithm. In this example, the system reuses *oat flakes* to satisfy the needs of both porridge and pancakes, rather than buying separate products for each dish. By choosing combinations that satisfy ingredient requirements at the lowest cost—such as bulk or shared items—the optimizer reduces the total expenditure. The total price of such a recommendation is computed as:

$$\text{TotalCost} = price_{\text{oat milk}} + 3 \cdot price_{\text{oat flakes}} + price_{\text{ABC Eggs}} = 1 + 3 \cdot 0.5 + 2 = 4.5$$

where the oat flakes are used in both recipes and require a combined quantity of $0.5 + 0.6 = 1.1$ kg. Since the product is sold in units of 0.5 kg, the algorithm selects 3 units to satisfy the total demand.

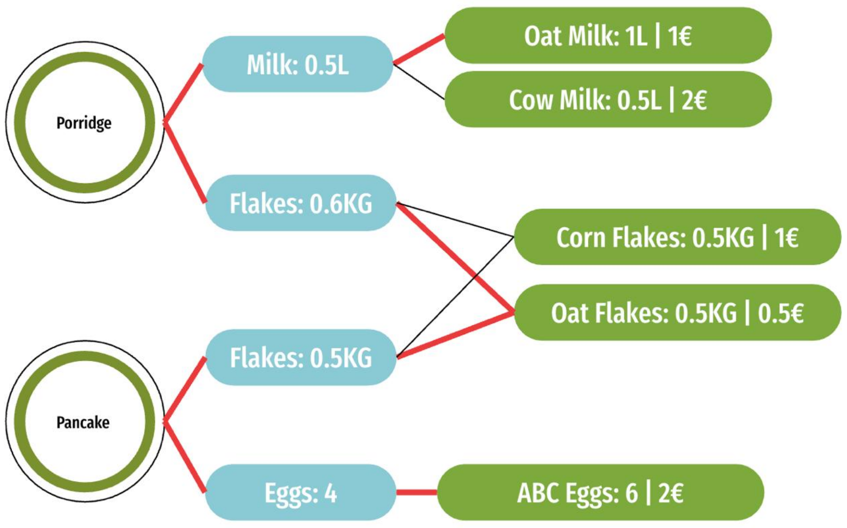

Figure 3.2: Optimal selling unit selection across two recipes. Circular nodes represent recipes. Blue nodes denote required ingredients with specific quantities. Green nodes indicate available product options, labeled with packaging size and price. Black edges show compatible product options for each ingredient. Red edges highlight the selected products that minimize total cost, promoting reuse across recipes.

3.4 Formal Problem Definition

We formally define four distinct optimization problems addressed in this research. Each problem assumes access to a set of recipes $\mathcal{R}$, a universe of ingredients $\mathcal{I}$, and a set of available selling units (products) $\mathcal{S}$. Each recipe $r \in \mathcal{R}$ consists of ingredients $i \in I_r \subseteq \mathcal{I}$, where each ingredient i requires weight $w_{r,i}$. For every ingredient i, a corresponding set of selling units $S_i \subseteq \mathcal{S}$ is available, where each selling unit $s \in S_i$ has unit weight w_s and price p_s.

Let $X \subset \mathcal{R}$ denote a set of recipes pre-selected by the customer, and let $Y \subset \mathcal{R} \backslash X$ denote the set of recipes recommended by the model, with $|Y| = k$. The goal in each problem is to jointly select Y (if applicable) and the appropriate quantities of selling units $q_s \in \mathbb{Z}_{\geq 0}$ for each $s \in \mathcal{S}$, so that the overall objective is minimized and all ingredient requirements are satisfied.

Problem 1: Cost-Minimizing Recipe and Product Selection

Given a set X of customer-selected recipes, the task is to recommend an additional set Y of k recipes. The optimizer must also determine the number of units q_s to purchase for each selling unit $s \in \mathcal{S}$, such that the combined ingredient requirements of all recipes in $X \cup Y$ are fulfilled. The objective is to minimize the total cost, which equals the sum of prices of all purchased selling units.

This problem reflects the core setting of our research: optimizing both the content (recipe selection) and logistics (product selection) of a multi-meal shopping basket.

Problem 2: Product Assignment for Fixed Recipe Set
In this setting, the customer has already chosen a set X of recipes and does not want additional recommendations. The task is to determine the quantities q_s of each selling unit needed to fulfill all ingredient requirements across the recipes in X, in a way that minimizes the total cost. Since the recipe set is fixed, the problem focuses purely on cost-efficient product assignment and potential product sharing across ingredients.

Problem 3: Weight-Minimizing Recipe and Product Selection
This variant is structurally similar to Problem 1 but uses a different optimization criterion. Given a set X of user-selected recipes, the model recommends a set Y of k additional recipes. It selects selling units to fulfill the combined ingredient requirements of $X \cup Y$, but instead of minimizing the total price, it minimizes the total weight of purchased products. This approach indirectly promotes waste reduction, since it encourages the use of fewer and smaller product packages.

Problem 4: Cost-Minimizing Recommendation with Cuisine Constraints
This variant extends Problem 1 by adding personalization. The model must recommend a set Y of k recipes, but each recommended recipe must belong to a predefined set of target cuisines, denoted $\mathcal{C}_{\text{target}}$. As before, the model must select quantities q_s for each selling unit to satisfy all ingredient demands from $X \cup Y$, and it must minimize the total cost. This formulation balances cost optimization with personalized, cuisine-aware recommendations.

Chapter 4

Related Work

This chapter reviews prior research in recipe optimization and meal planning systems, focusing on work that employs combinatorial optimization techniques such as Mixed Integer Linear Programming, Integer Linear Programming (ILP), and Genetic Algorithms. These studies provide context for the novelty, relevance, and methodological direction of this thesis. We group the related work into three thematic areas: recipe-level meal planning systems, large-scale optimization models for food services, and comparative analyses of MILP and GA approaches.

4.1 Recipe-Level Optimization

Leung et al. [1995] introduced one of the earliest ILP-based systems for diet planning that operates at the recipe level rather than individual food items. Their model selected weekly meal plans from a predefined set of recipes, minimizing cost or preparation time while satisfying nutritional constraints. Although their work emphasized realistic, palatable outputs, it did not model product-level cost structures, ingredient reuse, or packaging units. These exclusions reduce the model's relevance for environments such as online supermarkets, where these factors play a important role in determining cost.

Kashima et al. [2009] applied a multidimensional 0/1 Knapsack formulation and Genetic Algorithms to generate personalized daily meal plans aimed at promoting healthier diets. Their system considered user satisfaction and nutritional goals but did not account for product costs, ingredient reuse, or packaging constraints. While their use of GAs for navigating a large search space inspired elements of our approach, the problem scope in their work remains narrow and health-centric, in contrast to our economic focus in a retail context.

4.2 Large-Scale Meal Planning and Institutional Settings

Ramos-Pérez et al. [2021] proposed a multi-objective model for school lunch planning that balances cost with dietary diversity. Using Non-dominated Sorting Genetic Algorithm II (NSGA-II) [Deb et al., 2002], Strength Pareto Evolutionary Algorithm (SPEA2) [Zitzler, 2002], and Indicator-based Evolutionary Algorithm (IBEA) [Zitzler and Künzli, 2004], they minimized both financial costs and the repetition of food groups, based on strict pediatric nutritional guidelines. Although their setting differs from ours, the study illustrates how evolutionary algorithms can handle multi-objective optimization in large combinatorial spaces.

Padovan et al. [2023] formulated a Mixed Integer Programming model to create monthly menus for institutional foodservice in Brazil, optimizing for cost, nutritional requirements, and variety. Their model also includes practical constraints like visual appeal and ingredient diversity. However, it does not model product-level packaging or leverage shared ingredients across recipes, which are central to our cost-reduction strategy.

Wang et al. [2022] developed a recipe recommender for diabetic patients using Integer Programming to minimize cost while meeting macronutrient and micronutrient requirements. Their approach integrates food composition, personal preferences, and exclusion constraints, prioritizing health outcomes over operational efficiency. Though the system is well-suited for individual-level dietary planning, it lacks mechanisms for scaling across multiple recipes and optimizing product reuse.

4.3 Comparative Studies of MILP and GA

de Souza Amorim et al. [2021] compared MILP and hybrid Genetic Algorithms in a glass production planning context. While the MILP formulation could solve small instances optimally, hybrid GAs scaled better and provided near-optimal solutions for larger instances. This study underscores the complementary strengths of both approaches and motivates our inclusion of GA as a secondary method for benchmarking and scalability testing.

Foster et al. [2013] conducted an in-depth comparison of MIP and GA methods in the domain of distributed power generation planning. They analyzed several problem formulations using different power flow models and showed that GAs offer rapid, flexible solutions, whereas MIP methods yield tighter bounds and global insights when tractable. Their findings support our hybrid approach, which balances optimization quality with practical feasibility.

Azamathulla et al. [2008] evaluated GA and Linear Programming (LP) models for real-time reservoir operation in agricultural settings. Their GA-based model demonstrated better adaptability and performance under dynamic, multi-objective conditions compared to the LP baseline. This work reinforces the relevance of GAs in planning scenarios with complex constraints and uncertain inputs, such as our recipe recommendation problem.

4.4 Summary

The reviewed literature demonstrates a strong foundation for applying optimization techniques to meal planning and related domains. While prior work has explored both exact (MILP, ILP) and heuristic (GA) approaches, few studies address product-level cost modelling, ingredient reuse, or packaging constraints in retail-scale settings. Additionally, comparisons between exact and heuristic methods are often domain-specific and rarely applied to recipe recommendation. This thesis builds upon these insights by integrating MILP and GA techniques in a hybrid framework that emphasizes economic efficiency, real-world constraints, and scalability in the context of online grocery platforms.

Chapter 5

Methods

This chapter outlines the development of the recipe recommendation system, focusing on the modelling decisions and optimization techniques used to minimize product costs under real-world constraints.

We investigate two solution approaches: Mixed Integer Linear Programming and a Genetic Algorithm. We selected MILP because it directly optimizes a cost-minimization objective, making it well-suited for our problem. In contrast, Constraint Programming (CP) focuses on finding feasible solutions that satisfy complex constraints, but does not naturally prioritize optimality with respect to an objective function. Since our goal is not just to find feasible assignments but to minimize total grocery cost, MILP offers a more appropriate modeling framework [Meng et al., 2020]. Likewise, while Machine Learning (ML) is powerful in data-rich contexts, it is less appropriate for constraint-heavy optimization without labeled training data [Popescu et al., 2022].

GA complements MILP by offering a flexible, scalable alternative for exploring near-optimal solutions in large and complex search spaces. It is particularly useful when exact methods become computationally expensive.

The rest of the chapter details the MILP model and its variations, followed by the GA design, including representation, fitness function, and genetic operators.

5.1 Mixed Integer Linear Programming Model

We selected MILP as a core optimization method for this research due to its ability to deliver globally optimal solutions for structured decision problems. The recipe recommendation task—framed as ingredient-to-product assignment with cost and selection constraints—naturally fits the MILP paradigm.

MILP supports binary and continuous variables, linear objectives, and diverse constraint types, making it ideal for modeling rule-based systems like recipe selection. Solvers such as *IBM CPLEX* and *Gurobi* efficiently handle large instances and return exact solutions within reasonable time limits.

This approach meets key methodological needs: it offers **multi-objective capability** by allowing multiple linear goals in the objective function; ensures **scalability** through advanced solver techniques like presolve, branch-and-bound, and parallelization; provides **ease of modelling** since the problem's structure (e.g., weight constraints, cuisine limits) maps naturally to linear constraints; and supports **flexibility**, enabling domain-specific rules such as cuisine diversity or soft waste penalties with minimal changes.

Overall, MILP delivers exactness, transparency, and serves as a robust baseline for evaluating heuristic methods like Genetic Algorithms. Overall, MILP delivers exactness and transparency, making it a robust baseline for evaluating heuristic methods like Genetic Algorithms. These qualities are especially important in our context, as customers are expected to add the recommended recipes/products to their carts — making solution quality and interpretability essential for user satisfaction

5.1.1 MILP Formulation

The optimization model is formulated as a Mixed Integer Linear Program with the goal of minimizing the total cost of purchasing ingredients while ensuring that the selected recipes remain complete. The model optimally assigns *selling units* (specific product packages) to *ingredients* in recipes, allowing for product sharing across multiple recipes to reduce costs.

Objective Function

The objective function minimizes the total cost of the selected selling units:

$$\min \sum_{k \in K} c_k x_k \tag{5.1}$$

where:

- $c_k \in \mathbb{N}_0$ is the price of selling unit k, expressed in cents (e.g., $c_k = 1005$ represents €10.05).
- $x_k \in \mathbb{N}_0$ is the total integer quantity of selling unit k purchased.
- K is the set of all selling units, where each $k \in K$ is a unique string identifier.

This optimization is subject to several constraints.

Ingredient Weight Satisfaction

Each ingredient has a required weight, specified either in grams, or milliliters, or pieces (e.g., avocado), depending on product's type. All weights refer strictly to the usable content and explicitly exclude packaging. Ingredients are modeled as abstract entities that can be satisfied by a variety of selling units. For example, the ingredient "milk" may be fulfilled by selling units such as cow milk, almond milk, or oat milk. The constraint ensures that weight requirement is satisfied:

$$\sum_{k \in K_{r,i}} w_k y_{r,i,k} \geq W_{r,i} \cdot z_r, \quad \forall r \in R, \forall i \in I_r \tag{5.2}$$

where:

- $K_{r,i} \subseteq K$ is the set of selling units that can be used for ingredient i in recipe r. Each $k \in K_{r,i}$ is a unique string identifier.
- I_r is the set of ingredients required for recipe r, where each ingredient $i \in I_r$ is a unique string identifier.
- R is the set of all recipes, with each $r \in R$ represented by a unique string identifier.
- $y_{r,i,k} \in \mathbb{R}_{\geq 0}$ is the non-negative decimal quantity of selling unit k allocated for ingredient i in recipe r.
- $w_k \in \mathbb{N}_0$ is the content weight or number of pieces in a single unit of selling unit k, excluding packaging.
- $W_{r,i} \in \mathbb{N}_0$ is the required weight or number of pieces for ingredient i in recipe r, also excluding packaging.
- $z_r \in \{0, 1\}$ is a binary variable indicating whether recipe r is selected.

Single Selling Unit per Ingredient

As mentioned in the previous constraint, there are multiple selling units, which can be used to fulfill ingredient requirement. But, only one can be chosen. This constraint is imposed because in multiple cases selling units cannot be combined. For example, cooking porridge with almond milk and cow milk would be inconvenient. Therefore, each ingredient in a recipe must be assigned exactly *one* selling unit:

$$\sum_{k \in K_{r,i}} p_{r,i,k} = z_r, \quad \forall r \in R, \forall i \in I_r \tag{5.3}$$

where:

- $p_{r,i,k} \in \{0, 1\}$ is a binary variable indicating whether the selling unit k is selected for ingredient i in recipe r.

Quantity Consistency

The total quantity of a selling unit must be *sufficient* to cover all its assigned uses across selected recipes:

$$x_k \geq \sum_{r \in R} \sum_{i \in I_r} y_{r,i,k}, \quad \forall k \in K \tag{5.4}$$

Pre-Selected Recipes

The model simulates a real-world scenario, where a customer selects some recipes, but cannot choose other recipes, hence Picnic attempts to help customers by proposing several more recipes. Recipes selected by a customer are called *pre-selected* and must always be included in the recommendation:

$$z_r = 1, \quad \forall r \in R_{\text{preselected}} \tag{5.5}$$

Number of Recommended Recipes

The system must recommend in total T recipes, including the *pre-selected* ones:

$$\sum_{r \in R} z_r = T \tag{5.6}$$

Domain Constraints

All quantity variables must be *non-negative*, and binary variables must be properly constrained:

$$x_k \geq 0, \quad y_{r,i,k} \geq 0, \quad z_r \in \{0,1\}, \quad p_{r,i,k} \in \{0,1\} \tag{5.7}$$

Additionally, auxiliary constraints enforce logical relationships between variables:

$$x_k \geq p_{r,i,k}, \quad y_{r,i,k} \geq \epsilon p_{r,i,k}, \quad y_{r,i,k} \leq M p_{r,i,k}, \quad y_{r,i,k} \leq M z_r \tag{5.8}$$

where:

- ϵ is a small decimal positive constant ensuring that $y_{r,i,k}$ is nonzero when $p_{r,i,k}$ is active (i.e., if a selling unit is selected, its assigned ingredient weight must be greater than zero), see Section 2.1.2.
- M is a sufficiently large constant used for enforcing big-M constraints (i.e., if a recipe or its corresponding selling unit is not selected, the associated ingredient weight must be zero), see Section 2.1.2.

5.1.2 MILP Formulation for Fixed Recipe Set

In addition to recommending recipes, the system should also support customers who have already selected their preferred set of recipes. In such cases, there is no need to generate new recipe suggestions; instead, the task becomes finding the most cost-efficient way to fulfill the ingredient requirements of the fixed recipe set. This can be achieved by optimizing the selection and sharing of products (selling units) across the given recipes.

This problem is modeled as a special case of the main MILP formulation, where the set of recipes is fixed and fully known. Therefore, the binary recipe selection variables z_r are no longer decision variables but are treated as fixed values equal to 1 for all recipes in the user-defined set. The objective remains minimizing the total cost:

$$\min \sum_{k \in K} c_k x_k \tag{5.9}$$

The same structure of constraints is preserved:

- Ingredient weight satisfaction to ensure all ingredients are sufficiently fulfilled.
- Single selling unit per ingredient to enforce consistency.
- Quantity consistency to properly aggregate total product usage.
- Domain constraints and auxiliary logic to maintain model feasibility.

This variant of the model transforms the system into a "shopping assistant", optimizing cost and product reuse for a fixed list of recipes. Additional constraints, such as budget limits or waste minimization, can be easily added to this formulation to further tailor it to user needs.

5.1.3 MILP Formulation Addressing Waste

We observed that a significant challenge during the development of the optimization system was the issue of *ingredient waste*. Waste arises when purchased selling units exceed the required amount needed across all selected recipes, resulting in leftover products. While the primary goal of this work is to minimize the total cost of ingredients, it is also desirable to minimize waste as a secondary objective. Excessive waste not only undermines cost efficiency but also reduces the practical usefulness of the recommendations.

To address this issue, we explored three modelling strategies within the MILP framework:

- **Weight Minimization:** Use total ingredient weight as a proxy for waste.

- **Cuisine-Based Selection:** Restrict recipe choices to a small number of cuisines to promote ingredient reuse.
- **Multi-Objective Optimization:** Combine cost and weight minimization into a single objective.

Each of these approaches required adjustments to the optimization model, as described below.

Weight Minimization

The simplest approach to indirectly reduce waste is to minimize the total quantity (weight) of purchased products. In this formulation, the objective function becomes:

$$\min \sum_{k \in K} w_k x_k \tag{5.10}$$

where w_k is the weight or volume of a single unit of selling unit k, and x_k is the total quantity purchased. To ensure consistency, all selling units are assigned an equivalent weight in either grams or milliliters, even if they are typically measured in pieces (e.g., avocados or eggs). This allows the model to compare and penalize excess quantity in a uniform manner. While this objective ignores prices, it discourages purchasing unnecessary weight or volume and thus indirectly addresses waste.

Cuisine-Based Selection

A more targeted way to reduce waste is to recommend recipes from the same or similar cuisines, since these often share overlapping ingredients. To achieve this, we added the following constraints to the model:

$$\sum_{r \in R_{\text{cuisine}}} z_r = C_{\text{cuisine}}, \quad \forall \text{cuisines} \tag{5.11}$$

$$z_r = 0, \quad \forall r \in R_{\text{unselected\_cuisine}} \tag{5.12}$$

Here, z_r is a binary variable indicating whether recipe r is selected, R_{cuisine} is the set of available recipes for a given cuisine, C_{cuisine} is a randomly assigned count that imitates customer behavior by specifying how many recipes to select from that cuisine, and $R_{\text{unselected\_cuisine}}$ is the set of all recipes whose cuisine was not selected.

Multi-Objective Optimization: Cost and Weight

To explicitly model the trade-off between cost and weight, we introduced a joint objective function that minimizes both total price and total weight:

$$\min\left(\sum_{k\in K} c_k x_k + \sum_{k\in K} w_k x_k\right) \tag{5.13}$$

where c_k is the cost of selling unit k. This formulation penalizes both expensive and excessive purchases, encouraging efficient solutions that balance both cost and waste considerations.

These three formulations provide distinct modelling strategies for incorporating waste reduction into the MILP-based recipe optimization framework.

5.2 Genetic Algorithms Model

While Mixed Integer Linear Programming approach guarantees optimal solutions, it can become computationally expensive when dealing with large-scale optimization problems. Given the complexity of the recipe optimization problem, where the search space grows exponentially with the number of recipes, ingredients, and selling units, we explored metaheuristic approaches that trade off optimality for computational efficiency.

A Genetic Algorithm is a nature-inspired optimization technique based on the principles of natural selection and evolution. Unlike exact solvers, GA does not guarantee an optimal solution but can efficiently explore large search spaces and converge to near-optimal solutions within reasonable time constraints. The motivation for using GA in this study is as follows:

1. **Scalability**: GA is well-suited for high-dimensional combinatorial problems and is less affected by increasing the number of constraints.
2. **Flexibility**: Unlike MILP, GA can easily incorporate additional constraints without significantly impacting performance.
3. **Computational Efficiency**: GA provides a balance between solution quality and solve time, making it ideal for cases where near-optimal solutions are sufficient.

The following sections describe the chromosome representation, genetic operators, and the overall GA workflow used in this research.

5.2.1 Chromosome Representation

In Genetic Algorithms , a chromosome represents a potential solution to the optimization problem. For the recipe optimization problem, each chromosome is structured as:

$$C = (\mathbf{R}, \mathbf{S}, \mathbf{Q}) \tag{5.14}$$

where:

- $\mathbf{R} = \{r_1, r_2, \dots, r_7\}$ is a set of selected recipes, including a **fixed** subset of pre-selected recipes.
- $\mathbf{S}$ represents the map of selected selling unit ids for each ingredient i in recipe r.
- $\mathbf{Q} = \{q_k\}$ is the mapping from each selected selling unit k to its quantity.

Each chromosome encodes a valid set of recipes and associated ingredient allocations, ensuring that all selected recipes are feasible while minimizing cost. The subset of pre-selected recipes within $\mathbf{R}$ is fixed and remains unchanged throughout the evolutionary process. However, the corresponding selling units for their ingredients ($\mathbf{S}$) may still be modified to explore cost-effective alternatives.

5.2.2 Fitness Function

The primary objective of the optimization remains to minimize the total cost while ensuring all selected recipes have the required ingredients. The fitness function is defined as:

$$\text{fitness}(C) = \sum_{k \in K} c_k q_k \tag{5.15}$$

where:

- $c_k \in \mathbb{N}_0$ is the price of selling unit k, expressed in cents (e.g., $c_k = 1005$ represents €10.05).
- $q_k \in \mathbb{N}_0$ is the integer quantity of selling unit k purchased.

Lower fitness values correspond to cheaper solutions, making cost minimization the driving force behind the selection process.

5.2.3 Genetic Operators

Genetic Algorithms evolve populations over multiple generations using selection, crossover, and mutation.

Selection: Probabilistic Tournament Selection

To choose parent solutions for crossover, a *probabilistic tournament selection* mechanism is used:

1. Randomly select k individuals from the population.
2. Sort them based on fitness (cost).
3. Select the best individual with probability p and a weaker individual with probability $1 - p$.

This method balances exploration and exploitation, ensuring that better solutions have a higher chance of survival while avoiding premature convergence.

Crossover: Uniform Crossover

The crossover operator combines two parent solutions to generate offspring. A *uniform crossover* mechanism is used, where recipes are swapped between parents with a 50% probability:

$$C_{\text{child}} = \text{Crossover}(C_{\text{parent1}}, C_{\text{parent2}}) \tag{5.16}$$

where:

- Each recipe in $\mathbf{R}$ has a 50% probability of being inherited from either parent.
- Selling unit selections $\mathbf{S}$ are updated to ensure ingredient compatibility across the offspring.

Crossover introduces diversity in the population while preserving high-quality partial solutions.

Mutation: Recipe and Selling Unit Mutation

Mutation introduces randomness to prevent stagnation in local optima. We implement two mutation operators:

Recipe Mutation A randomly selected recipe is replaced with another from the dataset:

$$r_{\text{new}} = \text{Random}(R_{\text{pool}} \setminus R) \tag{5.17}$$

This mutation encourages exploration of new recipe combinations.

Selling Unit Mutation The selling unit for a randomly chosen ingredient is replaced by a different selling unit corresponding to the chosen ingredient:

$$s_{\text{new}} = \text{Random}(S_{\text{available}}) \tag{5.18}$$

This allows for better ingredient price optimization. Mutation occurs with a predefined mutation probability p_m.

5.2.4 Algorithm Workflow

The genetic algorithm starts by generating an initial population of chromosomes, each representing a potential recipe combination that includes the fixed set of pre-selected recipes. The population evolves over a fixed number of generations through selection, crossover, and mutation.

In each generation, tournament selection is used to choose parent chromosomes based on their fitness. These parents undergo uniform crossover to produce offspring, which are then mutated to introduce variability and explore the search space. The algorithm adds offspring to the pool, and selects a new generation from the combined set of parents and children using a survivor selection strategy.

This process repeats for a fixed number of generations. Finally, the algorithm returns the best chromosome found, that is, the one with the lowest fitness value, representing the most cost-efficient and constraint-satisfying recipe recommendation.

Algorithm 2 Genetic Algorithm for Recipe Optimization

```
Require: Set of pre-selected recipe IDs $R_{\text{pre}}$, number of generations $G$, population
    size $N$
 1: Initialize population $P \leftarrow \textsc{CreatePopulation}(N, R_{\text{pre}})$
 2: for generation $g = 1$ to $G$ do
 3:     $O \leftarrow \emptyset$                                   // Offspring pool
 4:     for $i = 1$ to $N/2$ do
 5:         $p_1 \leftarrow \textsc{TournamentSelection}(P)$
 6:         $p_2 \leftarrow \textsc{TournamentSelection}(P)$
 7:         $(c_1, c_2) \leftarrow \textsc{UniformCrossover}(p_1, p_2)$
 8:         $c_1 \leftarrow \textsc{Mutate}(c_1)$
 9:         $c_2 \leftarrow \textsc{Mutate}(c_2)$
10:         $O \leftarrow O \cup \{c_1, c_2\}$
11:     end for
12:     $P \leftarrow \textsc{NextGeneration}(P, O)$
13: end for
14: return $\arg\min_{x \in P} \textsc{Fitness}(x)$
```

5.2.5 Final Notes

This GA implementation provides an alternative to MILP, offering a scalable and flexible approach to recipe optimization. While GA does not guarantee exact optimality, it is capable of exploring large search spaces and producing high-quality solutions within practical time limits.

The next chapters compare the performance of GA and MILP, analyzing tradeoffs between solution quality and computational cost.

5.3 Hybrid MILP + GA Model

To combine the strengths of both exact and heuristic methods, we introduce a hybrid model that integrates a Genetic Algorithm with a Mixed Integer Linear Programming solver. This approach aims to retain the scalability and flexibility of GA while leveraging MILP's ability to compute globally optimal product-level decisions.

In this model, the GA is responsible for exploring the space of possible recipe combinations, while the MILP solver determines the most cost-efficient product assignments for any given recipe set. Each GA chromosome encodes only the set of selected recipes—no selling units or quantities are stored or manipulated by the GA directly. Instead, the MILP model evaluates each candidate solution by optimizing the associated selling units and quantities to minimize total cost.

5.3.1 Chromosome Representation

Each chromosome represents a set of recipes and is formally structured as:

$$C = (\mathbf{R}) \tag{5.19}$$

where $\mathbf{R} = \{r_1, r_2, \ldots, r_7\}$ is a set of selected recipes, including a **fixed** subset of pre-selected recipes.

In this hybrid approach, the GA only evolves the non-fixed subset of selected recipes. The MILP model computes corresponding selling units and their quantities on demand.

5.3.2 Fitness Evaluation via MILP

The fitness of each chromosome is computed by passing the full recipe set into a MILP model, which selects selling units and their quantities to minimize the total price. This MILP model is identical to the one described in Section 5.1.2 for the fixed-recipe case.

5.3.3 Genetic Operators

All genetic operations in the hybrid model act exclusively on the mutable subset of selected recipes. The set of pre-selected recipes remains unchanged.

Crossover: Uniform Crossover We use uniform crossover on the mutable recipes as in the Section 5.2.3.

Mutation: Recipe Level A mutation operator randomly replaces one recipe in the mutable recipe set with another from the available recipe pool, excluding duplicates and pre-selected ones. The same approach was described by Equation 5.17.

Selling units and their quantities are never encoded in the chromosome. They are always computed by MILP during fitness evaluation.

5.3.4 Algorithm Workflow

The hybrid algorithm starts by generating an initial population of chromosomes. Each chromosome encodes a full recipe plan $\mathbf{R}$, which includes a fixed set of pre-selected recipes and a mutable subset of recommended recipes. The GA evolves only the mutable subset. Selling units and their quantities are never stored in the chromosome, they are computed on demand by a MILP solver.

In each generation, tournament selection identifies parent chromosomes based on their fitness, which is the total cost returned by the MILP for the full recipe set. Uniform crossover combines parent recipes to create offspring, and mutation replaces one recipe in the mutable set. After applying these operators, MILP evaluates the new chromosomes by optimizing selling units and their quantities. The new generation is formed using a survivor selection strategy.

This process continues for a fixed number of generations. At the end, the algorithm returns the chromosome with the lowest MILP-evaluated cost, representing the most cost-efficient recipe combination.

Algorithm 3 Hybrid Genetic Algorithm with MILP Evaluation

```
Require: Set of fixed pre-selected recipe IDs $R_{\text{pre}}$, number of generations $G$, population size $N$
 1: Initialize population $P \leftarrow$ CREATEPOPULATION$(N, R_{\text{pre}})$
 2: for generation $g = 1$ to $G$ do
 3:     $O \leftarrow \emptyset$                                  // Offspring pool
 4:     for $i = 1$ to $N/2$ do
 5:         $p_1 \leftarrow$ TOURNAMENTSELECTION$(P)$
 6:         $p_2 \leftarrow$ TOURNAMENTSELECTION$(P)$
 7:         $(c_1, c_2) \leftarrow$ UNIFORMCROSSOVER$(p_1, p_2)$
 8:         $c_1 \leftarrow$ MUTATE$(c_1)$
 9:         $c_2 \leftarrow$ MUTATE$(c_2)$
10:         // $c_1$ and $c_2$ are always evaluated using MILP
11:         $O \leftarrow O \cup \{c_1, c_2\}$
12:     end for
13:     $P \leftarrow$ NEXTGENERATION$(P, O)$
14: end for
15: return $\arg\min_{x \in P}$ FITNESS$(x)$
```

5.3.5 Summary

This hybrid model delegates strategic decision-making (recipe selection) to GA while outsourcing tactical optimization (product choice and quantity) to MILP. It benefits from the exploratory power of GA and the exactness of MILP, producing high-quality solutions even in large search spaces.

Chapter 6

Experimental Setup

This chapter describes the experimental setup used to evaluate the optimization models developed in this study. It includes a detailed overview of the dataset, analysis methodologies, and the evaluation procedures for each of the research questions. For each research question, a dedicated section outlines the corresponding experimental configuration, key assumptions, and metrics used to assess performance. Additionally, we include a description of the computational environment to ensure reproducibility.

6.1 Dataset Description

All experiments are based on the complete set of recipes retrievable from the Picnic platform, comprising a total of $1\ 529$ unique recipes. These recipes collectively reference $7\,914$ distinct selling units (products). Below are some statistical insights that illustrate the scale and structure of the data:

- Average number of selling units per ingredient: 9.74
- Average number of ingredients per recipe: 6.00

Given that each ingredient can be represented by approximately 9.74 selling units on average, and each recipe contains around 6 ingredients, the number of possible product combinations to represent a *single* recipe is estimated as

$$9.74^6 \approx 839\,000$$

combinations. This highlights the combinatorial complexity of the recommendation problem, even at the level of a single recipe.

6.2 Baseline Comparison Methodology

To quantify the benefit of optimization, we define a *naive cost* that estimates the total price without accounting for shared products. For a set of selected recipes

$\mathcal{R}_{\text{rec}}$, where each recipe $r \in \mathcal{R}_{\text{rec}}$ uses a set of selling units $\mathcal{S}_r$, and each unit $s \in \mathcal{S}_r$ has a price c_s and quantity $q_{r,s}$, the naive cost is computed as:

$$\text{NaiveCost} = \sum_{r \in \mathcal{R}_{\text{rec}}} \sum_{s \in \mathcal{S}_r} c_s \cdot q_{r,s}$$

This assumes each recipe is fulfilled independently, with no reuse of products. Let OptimizedCost denote the cost returned by the MILP model, which considers shared product usage across recipes. The resulting price difference is defined as:

$$\Delta_{\text{price}} = \text{NaiveCost} - \text{OptimizedCost}$$

A positive value indicates savings achieved through optimization.

6.3 GA Comparison Methodology

To evaluate the GA's solution quality, we compare its performance against the optimal prices produced by the CPLEX solver. For each of the 100 test cases, we record the final total price obtained by the GA and compute the difference from the CPLEX solution. The price difference is defined as:

$$\Delta_{\text{GA}} = \text{GA\_Total\_Price} - \text{CPLEX\_Total\_Price}$$

A smaller value of Δ_{GA} indicates that the GA is approaching the optimal solution. This difference is analyzed across nine different configurations, based on combinations of three population sizes (100, 200, 400) and three generation counts (100, 200, 400).

6.4 MILP for Cost Optimization (RQ1)

RQ1: *To what extent can a Mixed Integer Linear Programming approach minimize the total cost of recipe recommendations while remaining computationally feasible?*

We implement a baseline MILP model 5.1.1 using CPLEX to determine the lowest-cost combination of recipes and products. The model uses no additional constraints and serves as a reference point for assessing solution quality, computational feasibility, and runtime.

The choice of CPLEX was motivated by its robust MILP-solving capabilities, industry-wide adoption, and its convenient Python interface via the `docplex` library. Python was chosen for its flexibility, rich ecosystem of scientific libraries, and ease of prototyping. Additionally, Python is one of the most widely used programming languages in both academia and industry, and is also commonly used at Picnic, which aligns with the broader technological context of this research.

The experimental setup for this study consists of 100 unique test cases per configuration. In each case, we simulate a realistic scenario in which a user has selected a number of recipes, and the system either recommends additional recipes or optimizes product selection to minimize the total cost of required items.

To evaluate both optimization quality and scalability, we test the following configurations:

- **Recipe recommendation with optimization** (based on model 5.1.1):
  - 3 pre-selected recipes with 4 recommended recipes (7-day plan)
  - 2 pre-selected recipes with 5 recommended recipes (7-day plan)
  - 2 pre-selected recipes with 3 recommended recipes (5-day plan)
- **Product selection only** (based on model 5.1.2):
  - 3 fixed recipes
  - 5 fixed recipes
  - 7 fixed recipes

In all setups, the system's objective is to minimize the total price of all required products. The configurations involving recipe recommendation simulate typical meal planning scenarios, while the product-only setups assess the efficiency of product selection in the absence of recipe flexibility.

The following assumptions apply uniformly across all experiments:

- Users select only recipes—not specific selling units.
- The algorithm must select exactly one selling unit per ingredient.
- All selling units are assumed to be in stock and available.

This design enables us to benchmark how well MILP can solve cost minimization tasks under both flexible and constrained input conditions. To illustrate the scale of the problem, consider the most complex case: selecting four additional recipes from the remaining 1,526, given three pre-selected ones. The number of possible combinations is:

$$C(1526, 4) = \binom{1526}{4} \approx 2.25 \times 10^{11}$$

This combinatorial explosion underscores the importance of using efficient solvers for recipe recommendation and grocery optimization.

6.5 Solver Comparison: CPLEX vs. Gurobi (RQ2)

RQ2: *How do different MILP solvers (CPLEX and Gurobi) compare in terms of solve time, solution quality, and scalability?*

CPLEX and Gurobi were selected for this study due to their status as industry-standard solvers for MILP. Both offer state-of-the-art optimization capabilities and are widely used in academic and commercial settings for solving large-scale, complex problems. CPLEX is known for its strong support in IBM's ecosystem and integration with Python via `docplex`, while Gurobi is often recognized for its speed, scalability, and robust performance across a range of MILP formulations. Including both solvers allows for a meaningful comparison of solution quality and computational efficiency, particularly in applications with significant combinatorial complexity [Jablonskỳ et al., 2015].

The experiment setup follows the same base configuration as in RQ1 6.4, using 100 randomly generated test cases where three recipes are pre-selected and four are recommended. This allows for direct comparison of solver performance under fixed input conditions. To assess performance under varying problem sizes, two alternative configurations were also evaluated:

- Two pre-selected recipes with three recommended ones, simulating a five-day (work week) meal plan.
- Two pre-selected recipes with five recommended ones, aligning with a full week plan.

In addition to recipe-based variation, a dedicated scalability experiment was performed to analyze solver behavior as the number of available candidate recipes increases. For this experiment, both solvers were run using the 3 pre-selected, 4 recommended configuration, while systematically increasing the size of the recipe pool. The following candidate recipe set sizes were tested: 300 (4125 selling units), 600 (5235), 900 (6055), 1200 (6857), and approximately 1500 (the full dataset with 7914 selling units). For each size, the solve time was recorded across a set of randomly sampled instances. This setup isolates the effect of input scale on solver performance, highlighting the relative ability of each solver to handle growing combinatorial search spaces.

Together, these experiments provide a comprehensive view of how CPLEX and Gurobi compare across both fixed and scaling problem dimensions.

6.6 Meta-Heuristic Approach: Genetic Algorithm (RQ3)

RQ3: *Can a Genetic Algorithm, alone or combined with MILP, provide near-optimal cost-efficient solutions more efficiently than standalone MILP solvers, and under what conditions is it preferable?*

Two meta-heuristic models are evaluated to address this question: the standalone Genetic Algorithm (section 5.2) and the Hybrid MILP + GA model (section 5.3).

Both are implemented from scratch in Python and solve the same optimization problem posed in RQ1, where the goal is to recommend seven recipes (three preselected and four recommended) while minimizing the total grocery cost.

The experimental setup mirrors that of RQ1 section 6.4, using 100 randomized test cases to ensure consistency in evaluation. For each test case, the algorithms search for cost-minimizing recipe combinations, taking into account ingredient-to-product mappings and quantity requirements.

For all MILP evaluations in the hybrid model, we use the Gurobi solver. This choice is based on the findings from RQ2, which show that Gurobi consistently outperforms CPLEX in runtime across various configurations while maintaining solution quality. Its superior computational efficiency makes it the preferred solver for MILP-based components in the hybrid approach.

To explore how algorithm parameters influence performance, we evaluate the following configurations:

- **Standalone GA:** Population sizes of 100, 200, 400 crossed with generation counts of 100, 200, 400 (nine total settings).
- **Hybrid GA + MILP:** Three configurations: 100×100, 200×200, and 400×400 (population $\times$ generations).

For each configuration and test case, we measure the total solution cost and runtime. These metrics help evaluate the trade-offs between computational efficiency and solution quality across both heuristic models.

6.7 Integrating Real-World Constraints (RQ4)

RQ4: *How can waste be reduced as a secondary objective, and what is the impact of strategies like multi-objective optimization and cuisine-based selection on cost, waste and performance of the model?*

We explore three variants of the MILP model, all solved using CPLEX and under the same setup as in RQ1 section 6.4:

- A multi-objective version that minimizes both price and total weight of the selected products.
- A model constrained to recommend recipes from a single cuisine.
- A model that minimizes total weight as a primary objective, regardless of cost.

In this context, weight serves as a proxy for waste: by minimizing the total weight of purchased products, the model implicitly aims to reduce potential food waste. This design choice reflects a realistic trade-off between cost efficiency and sustainable consumption.

In the cuisine-constrained variant, one cuisine is randomly selected from the three pre-selected recipes, and the four recommended recipes are restricted to belong to this same cuisine. This setup aims to simulate a user preference or cultural consistency constraint in weekly planning.

Each model is tested under the same experimental conditions, and we observe how each constraint impacts price, waste, and solver performance.

6.8 Computational Environment

To ensure reproducibility and consistency, all experiments were conducted on a MacBook Pro (2023) equipped with an Apple M2 Pro chip and 16GB of RAM. The implementation was developed using `Python 3.10`. Depending on the optimization approach, we used different libraries:

- CPLEX: Implemented using `docplex` from IBM.
- Gurobi: Implemented using `gurobipy`.
- Genetic Algorithm: Implemented independently of third-party optimization libraries.

We executed all models on the same dataset, ensuring a fair comparison across different optimization approaches. The datasets used in these experiments were not preprocessed in any way, preserving their original structure as retrieved from Picnic.

The full implementation, including all experiment scripts and dataset files, is available on GitHub[1]. This repository contains the complete source code, configuration settings, and raw output data, enabling full reproducibility of the conducted experiments.

[1]`https://github.com/Andrew-Mereuta/Thesis`

Chapter 7

Results

This chapter presents the results corresponding to each of the research questions posed in the study. The structure of this chapter mirrors that of the experimental setup: for each research question, we summarize the key findings based on the experiments described previously. The aim is to present relevant observations from each configuration. Where appropriate, we also refer back to the hypotheses and experimental conditions introduced earlier.

7.1 MILP for Cost Optimization Results (RQ1)

This section presents the results related to RQ1 section 6.4, which investigates the effectiveness of a MILP-based approach for minimizing the total cost of a set of recipes. We evaluate performance using solve time and cost savings relative to a naive baseline. In the baseline, recipes are priced independently, without considering overlapping ingredient use. In contrast, the MILP model identifies shared selling units across recipes to reduce total cost.

7.1.1 Recipe Recommendation and Optimization

Figure 7.1 presents solve time distributions for configurations where the algorithm recommends and optimizes recipes simultaneously: 3 pre-selected with 4 recommended (3×4), 2 pre-selected with 5 recommended (2×5), and 2 pre-selected with 3 recommended (2×3). All three models complete within practical time bounds. The 2×3 configuration is the fastest on average, while the 2×5 setup shows slightly higher variance. Outliers beyond 60 seconds appear in more complex cases but remain rare.

7.1.2 Fixed Recipe Optimization

Figure 7.2 shows solve times for configurations with only pre-selected recipes (no recommendations): 3, 5, or 7 total. These runs are significantly faster than the runs from section 7.1.1, with median times under 0.06 seconds. Even with 7 recipes,

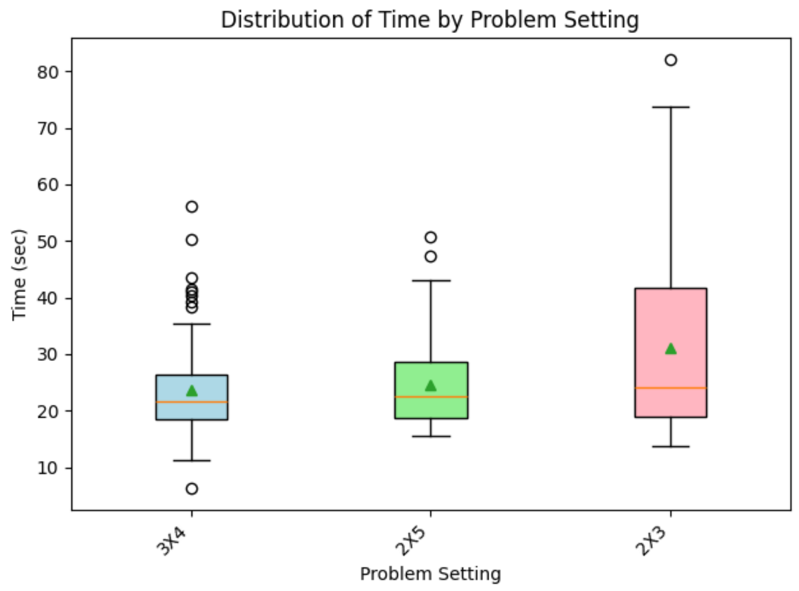

Figure 7.1: Distribution of solve times (in seconds) for three problem settings solved using the CPLEX MILP model: 3 pre-selected and 4 recommended recipes (3×4), 2 pre-selected and 5 recommended (2×5), and 2 pre-selected and 3 recommended (2×3). Each box represents the interquartile range, the orange line indicates the median, the green triangle marks the mean, and circles denote outliers. The results highlight variation in computational effort depending on the recipe combination configuration.

the MILP formulation remains highly tractable, demonstrating strong suitability for real-time grocery cost optimization when recipe choices are fixed.

7.1.3 Cost Savings Compared to Baseline

Figure 7.3 summarizes the price differences between MILP-optimized solutions and the naive baseline across all configurations using methodology from section 6.2. In all scenarios, the optimization model achieves lower costs. The largest median savings occur in the 3×4 shared model, with values around 6 euros. Even in fixed-recipe configurations with fewer optimization opportunities, the MILP model still occasionally achieves significant reductions—especially when ingredient overlap is high. These results highlight the model's consistent ability to leverage product sharing for cost minimization.

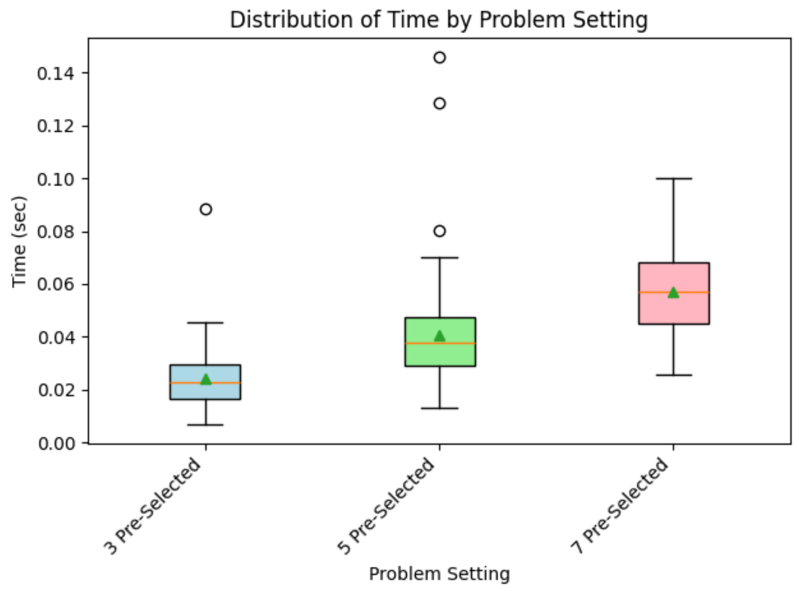

Figure 7.2: Distribution of solve times (in seconds) for three product-only optimization settings using the CPLEX MILP model. Each setting corresponds to a fixed number of pre-selected recipes: 3, 5, or 7. The orange line represents the median solve time, the green triangle denotes the mean, and circles indicate outliers. The results demonstrate how computational time increases with the number of recipes, reflecting greater problem complexity.

7.2 Solver Comparison: CPLEX vs. Gurobi Results (RQ2)

This section presents results related to RQ2 section 6.5, which investigates how two state-of-the-art MILP solvers, CPLEX and Gurobi, compare in terms of solve time and scalability. Both solvers were applied to the same optimization problem and dataset under identical conditions, allowing for a controlled and fair evaluation.

Three configurations were tested to observe the solvers' behavior across different problem sizes: 2 pre-selected with 3 recommended recipes, 2 pre-selected with 5 recommended recipes, and 3 pre-selected with 4 recommended recipes. The primary metric analyzed in this section is solve time, as both solvers produced identical solutions in terms of cost, given sufficient time.

The results highlight how runtime varies with problem complexity and solver choice, helping determine which approach is more efficient and scalable in real-world applications.

7.2.1 Solver Runtime Comparison

Figure 7.4 summarizes the distribution of solve times for both CPLEX and Gurobi across three test configurations. Across all cases, Gurobi consistently outperforms

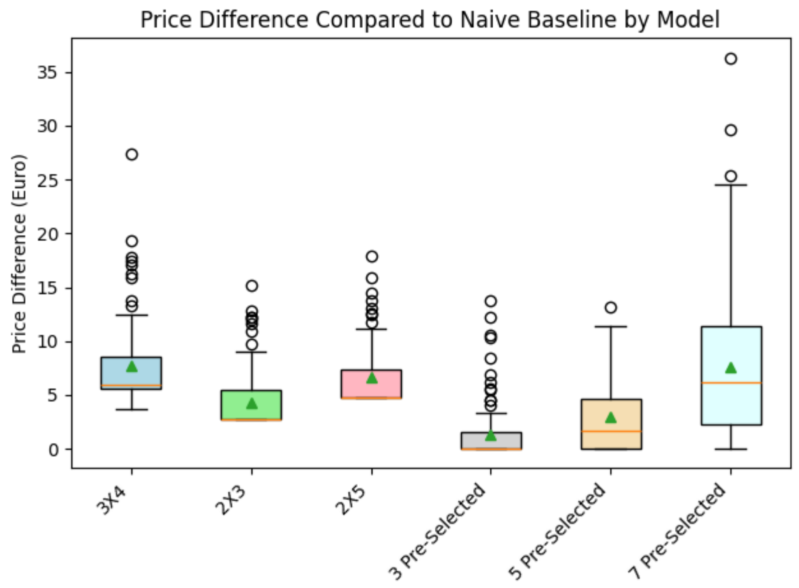

Figure 7.3: Price difference (in euros) between the naive baseline and the optimized solution across various problem settings solved by the CPLEX MILP model section 6.2. The first three groups represent configurations with recipe recommendation (e.g., 3×4 = 3 pre-selected and 4 recommended recipes), while the last three involve fixed sets of 3, 5, or 7 pre-selected recipes. The orange line shows the median savings, the green triangle represents the mean, and outliers are indicated by circles. Larger price differences reflect greater cost savings achieved through product optimization.

CPLEX in terms of runtime. The differences are particularly visible in the 2×3 and 2×5 configurations, where Gurobi's median solve time is roughly half that of CPLEX. In the 3×4 configuration, the performance gap is still present but less dramatic.

For each setting:

- **2×3 configuration:** CPLEX solve times vary widely, with some instances exceeding 80 seconds, while Gurobi consistently solves all instances under 25 seconds. Gurobi's median is notably lower, with tighter variance.
- **2×5 configuration:** CPLEX again shows higher variability and longer tail values, with solve times reaching up to 50 seconds. Gurobi maintains a narrower distribution with a lower median solve time.
- **3×4 configuration:** Both solvers perform more similarly here, though Gurobi still maintains a speed advantage. The variance for CPLEX is visibly higher with several outliers.

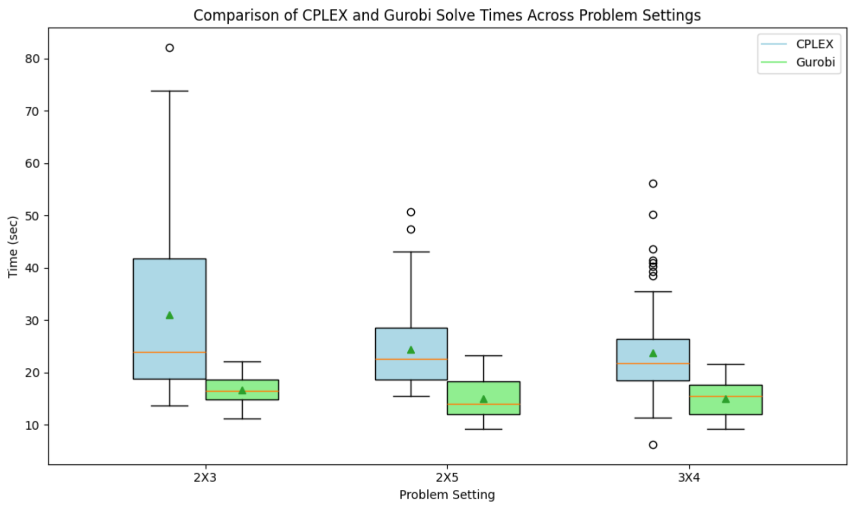

Figure 7.4: Solve time distribution (in seconds) for three recipe recommendation configurations—2×3, 2×5, and 3×4—comparing performance of the CPLEX and Gurobi MILP solvers. The orange line represents the median, the green triangle denotes the mean, and circles indicate outliers. Across all settings, Gurobi consistently achieves lower median and average solve times than CPLEX, demonstrating superior computational efficiency.

7.2.2 Scalability with Recipe Pool Size

To further evaluate solver scalability, both CPLEX and Gurobi were tested using the same 3×4 configuration while systematically increasing the total number of available candidate recipes. The tested recipe pool sizes include 300, 600, 900, 1200, and approximately 1500 recipes. For each pool size, solve times were recorded over a set of randomized instances.

Figure 7.5 presents the distribution of solve times for both solvers across the different recipe pool sizes. Gurobi consistently solves instances faster than CPLEX, with the performance gap widening as the recipe pool grows. For small recipe pools (300–600), Gurobi completes most runs in under 2 seconds. In particular, for the 300-recipe configuration, Gurobi achieves a mean solve time of just 0.72 seconds, with all runs completing in under a second. In contrast, CPLEX shows higher variability and longer tails even at smaller scales.

As the pool increases to 1200 and 1500 recipes, both solvers experience increased runtimes, but Gurobi maintains lower medians and tighter distributions than CPLEX. These results confirm Gurobi's superior scalability, especially in large-scale applications where the number of candidate recipes significantly affects the search space. CPLEX remains competitive at moderate sizes but becomes notably slower on larger instances.

These results demonstrate that Gurobi is the more efficient solver in this con-

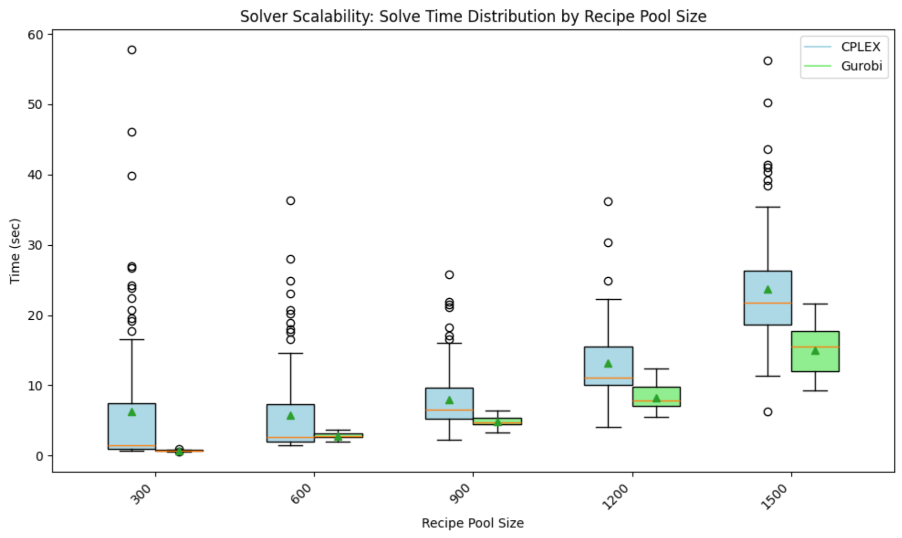

Figure 7.5: Solve time distribution (in seconds) for CPLEX and Gurobi MILP solvers across five recipe pool sizes: 300, 600, 900, 1200, and 1500 recipes. Each configuration involves selecting 3 pre-selected and 4 recommended recipes from the available pool. As the problem size increases, both solvers show rising computation times, but Gurobi consistently achieves lower median and average solve times, especially for larger datasets. Circles indicate outliers, and the orange line and green triangle represent the median and mean, respectively.

text, especially when runtime is a critical factor. The consistency of its performance makes it a strong candidate for real-time or near-real-time recommendation systems.

7.3 Heuristic Approach: Genetic Algorithm Results (RQ3)

This section presents results related to RQ3 section 6.6, which explores whether a Genetic Algorithm, alone or combined with MILP, can offer cost-efficient solutions more efficiently than standalone MILP solvers, and under what conditions these heuristic approaches become preferable.

7.3.1 Plain GA Results

Figure 7.6 illustrates how solution quality evolves across configurations, according to methodology from section 6.3. For each generation count, increasing the population size improves the quality of the solution. Larger populations explore a broader search space and are therefore more likely to find lower-cost combinations. For example, at 100 generations, the median price difference drops significantly as the population increases from 100 to 400. The same pattern holds for 200 and 400 generations.

However, improved solution quality comes at the cost of increased runtime, as shown in Figure 7.7. As expected, higher populations and longer generation runs both contribute to longer execution times. The combination of 400 generations and population size 400 leads to solve times exceeding 10 seconds on average, while the leanest configuration (100×100) consistently completes in under 2 seconds.

These results highlight a clear trade-off between runtime and solution quality. In time-sensitive scenarios, smaller configurations may offer acceptable performance with modest price differences. On the other hand, when optimality is prioritized and time constraints are relaxed, larger configurations yield solutions closer to MILP-optimal results.

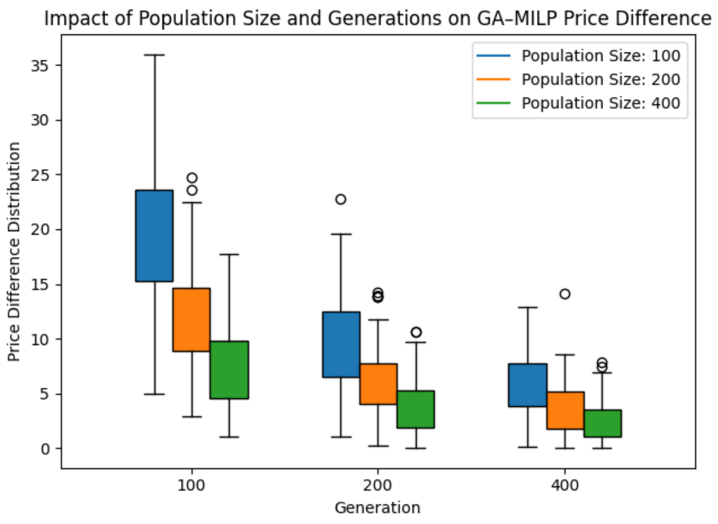

Figure 7.6: Distribution of price differences between the GA and the optimal MILP solution across different generation counts (100, 200, 400), with colors representing population sizes (100 = blue, 200 = orange, 400 = green). The y-axis shows how far each GA configuration deviates from the MILP benchmark in terms of total solution cost, see section 6.3. Results indicate that increasing either the population size or the number of generations leads to improved convergence toward the MILP-optimal solution. Circles indicate outliers.

7.3.2 Hybrid GA + MILP Results

To further explore RQ3, we evaluate a hybrid GA model that integrates exact MILP-based product selection into the evolutionary loop. In this setup, the GA operates only on the space of recipe combinations, while the MILP model computes the optimal product selection and total cost for each candidate. This hybrid approach combines the flexibility of GA with the precision of MILP.

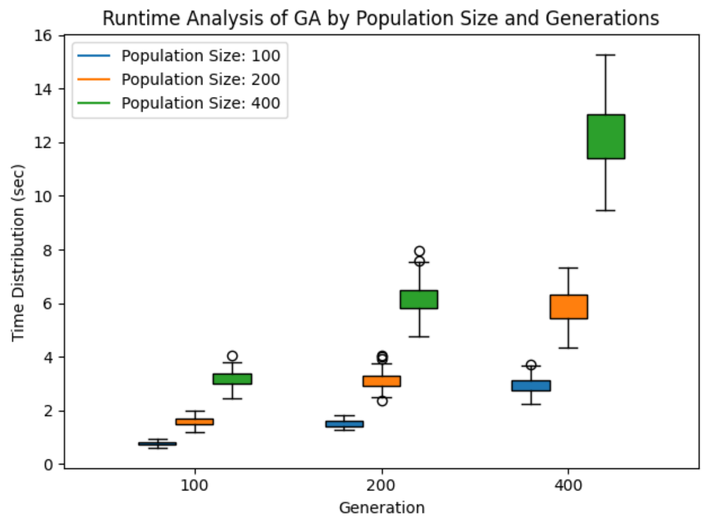

Figure 7.7: Solve time distribution (in seconds) for the GA across different generation counts (100, 200, 400) and population sizes (100 = blue, 200 = orange, 400 = green). Each box shows the variation in runtime for a given configuration, with the orange line indicating the median and circles denoting outliers. Results show that runtime increases non-linearly with both population size and number of generations, highlighting the trade-off between solution quality and computational cost in GA-based optimization.

As with the plain GA, we test three configurations: 100×100, 200×200, and 400×400 (population size × generations), using the same 100 test cases from RQ1. We compare the performance of the hybrid method to that of the plain GA along two dimensions: solution quality and runtime.

Figure 7.8 shows the price difference between the GA-derived solutions and Gurobi-optimal baselines. The hybrid GA consistently achieves lower price differences across all configurations. At the 100×100 setting, the median price difference drops from approximately 17 euros (plain GA) to under 5 euros in the hybrid. The 400×400 configuration closes the gap even further, with some instances achieving near-zero deviation from the optimal price.

However, this accuracy comes at a computational cost. Figure 7.9 shows that each fitness evaluation in the hybrid model runs a full MILP solver, significantly increasing total solve time. For the largest configuration (400×400), runtimes reach over 200 seconds on average, compared to under 15 seconds for plain GA.

These results confirm the hybrid model's ability to produce high-quality, near-optimal solutions. When runtime is not a critical concern, it offers a practical middle ground, providing much better prices than plain GA while retaining flexibility in recipe selection. However, in high-throughput or real-time scenarios, plain GA remains preferable because of its substantially lower runtime.

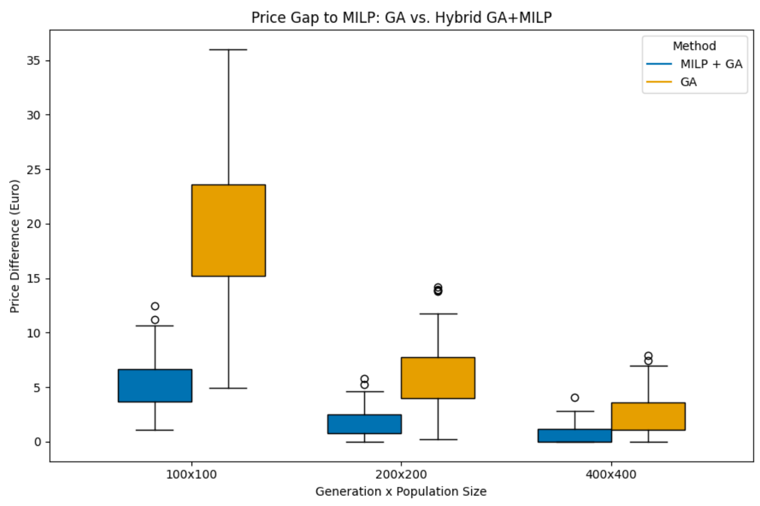

Figure 7.8: Distribution of price differences (in euros) between the MILP-optimal solution and two alternative approaches—standalone GA and Hybrid GA+MILP—across three configurations (100×100, 200×200, 400×400) representing population size × number of generations. The GA method (orange) shows larger and more variable deviations from the MILP optimum, while the hybrid approach (blue) consistently yields solutions closer to optimal, especially in higher configurations. This demonstrates the benefit of combining GA with exact optimization to improve cost efficiency. Circles denote outliers.

7.4 Integrating Real-World Constraints Results (RQ4)

This section presents results related to RQ4 section 6.7, which examines the impact of incorporating real-world constraints—such as minimizing food waste or enforcing cuisine consistency—on the model's price, waste, and runtime performance.

We compare four MILP-based model variants:

- **Price Min** (Baseline): Minimizes only the total cost of selected products, as described in the core MILP formulation section 5.1.1.
- **Weight+Price Min**: A multi-objective model that minimizes the sum of total cost and total weight section 5.1.3.
- **Cuisine Price Min**: Minimizes cost while requiring all recommended recipes to belong to the same cuisine section 5.1.3.
- **Weight Min**: Minimizes total product weight without considering price section 5.1.3.

Price Impact. Figure 7.10 shows the distribution of price differences between each model and the *Price Min* baseline. As expected, *Weight Min* leads to the

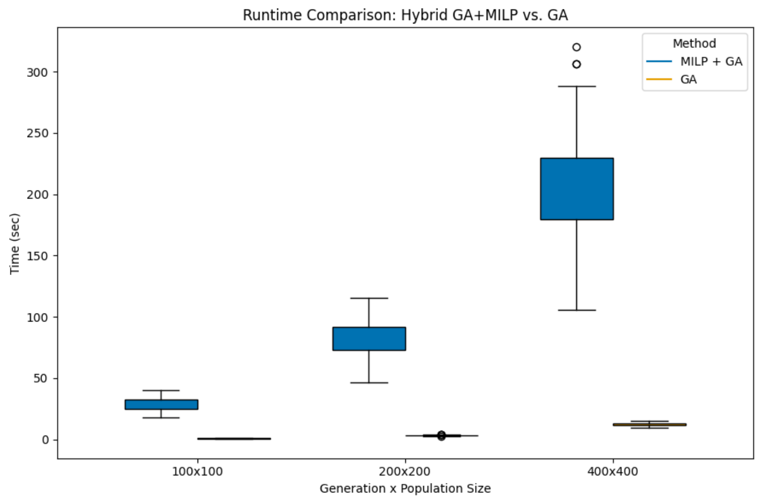

Figure 7.9: Runtime distribution (in seconds) for the Hybrid GA+MILP model (blue) and GA (orange) across three configurations: 100×100, 200×200, and 400×400 (population size × number of generations). While the hybrid model achieves better solution quality (see previous figure), it incurs significantly higher computational costs, especially at larger scales. In contrast, the standalone GA maintains consistent and low runtime across all settings. This highlights a clear trade-off between accuracy and efficiency when combining meta-heuristics with exact optimization.

highest price increase, with a median difference of over 30 euros and outliers surpassing 45 euros. *Cuisine Price Min* performs relatively close to the baseline, while *Weight+Price Min* consistently maintains low price differences—typically within 5 euros—indicating that it achieves meaningful waste reductions at minimal financial cost.

Waste Reduction. As shown in Figure 7.11, models that incorporate weight minimization (*Weight Min* and *Weight+Price Min*) achieve the lowest levels of waste. Median waste drops by 5–8 percents compared to the baseline. In contrast, the *Cuisine Price Min* model does not significantly reduce waste, suggesting that cuisine-level consistency alone is not a strong proxy for ingredient reuse.

Runtime. Figure 7.12 displays the distribution of solve times. The addition of constraints increases solver complexity: both *Weight+Price Min* and *Cuisine Price Min* exhibit higher average runtimes compared to the baseline. Nonetheless, most instances complete within a practical window, confirming the feasibility of applying these models in semi-real-time applications.

In conclusion, *Weight+Price Min* offers the best trade-off between sustainability and cost, achieving substantial waste reductions with only minor increases in total price and runtime. It presents a balanced and realistic approach for incorporating environmental goals into recipe recommendation systems.

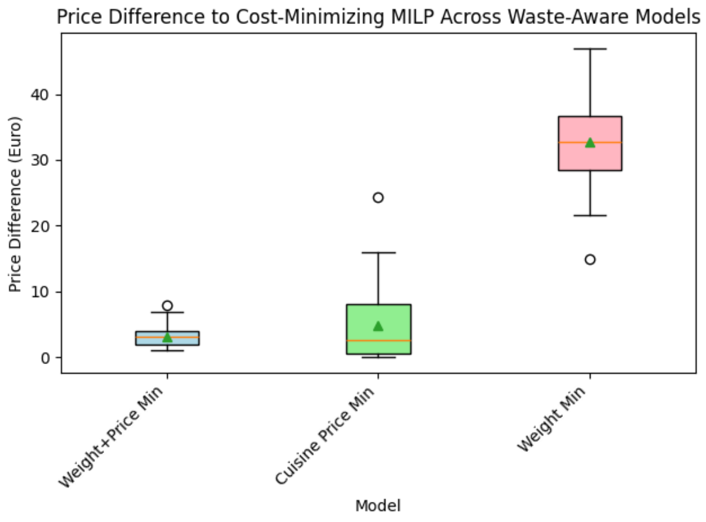

Figure 7.10: Distribution of price differences (in euros) between three waste-aware MILP formulations and the baseline MILP model that minimizes cost. The models compared are: multi-objective optimization of both price and weight (Weight+Price Min), price minimization with a cuisine constraint (Cuisine Price Min), and weight-only minimization (Weight Min). The Weight+Price Min model stays closest to the cost-optimal solution, while the Weight Min model shows the highest deviation, indicating a stronger trade-off between reducing waste and maintaining low grocery costs. Medians are marked by orange lines, means by green triangles, and outliers by circles.

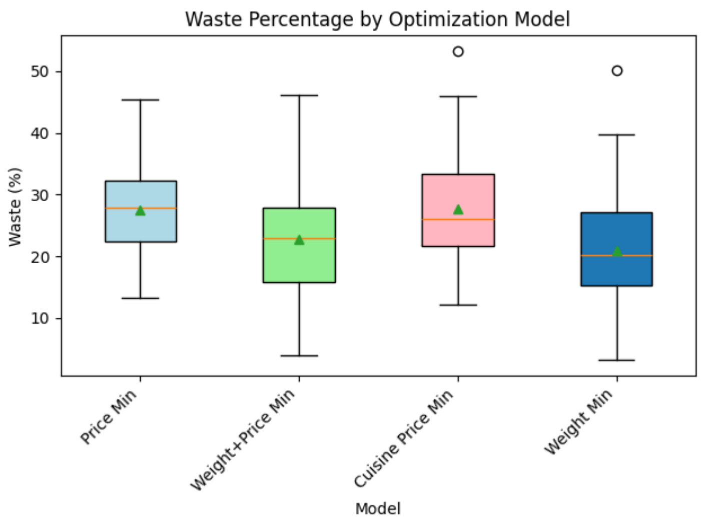

Figure 7.11: Distribution of waste percentages across four MILP formulations. Waste is defined as the proportion of purchased product quantity that exceeds the total required amount across all selected recipes. The models include: price-only minimization (Price Min), joint minimization of price and weight (Weight+Price Min), cuisine-constrained price minimization (Cuisine Price Min), and weight-only minimization (Weight Min). Models that explicitly account for weight—either as a primary or secondary objective—exhibit lower median waste levels compared to the price-only baseline. Medians are shown by orange lines, means by green triangles, and outliers as circles.

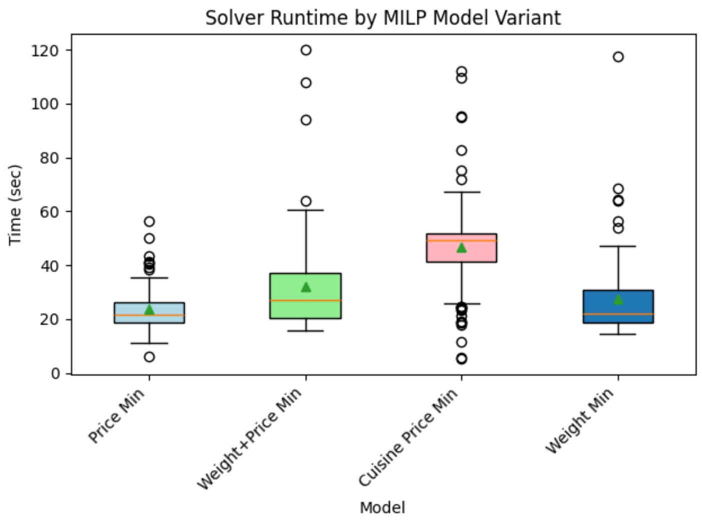

Figure 7.12: Distribution of solve times (in seconds) for four MILP model variants: cost-only minimization (Price Min), joint cost and weight minimization (Weight+Price Min), cuisine-constrained cost minimization (Cuisine Price Min), and weight-only minimization (Weight Min). While all models solve the same underlying problem structure, those with additional constraints or objectives—particularly the multi-objective and cuisine-based models—generally require longer solve times. Outliers are marked as circles, with medians shown as orange lines and means as green triangles.

Chapter 8

Discussion and Limitations

This chapter interprets and critically analyzes the results presented in Chapter 7. The focus is on deriving meaningful insights from the experimental findings and understanding the limitations inherent in the proposed approaches. Each section below addresses one of the core research questions and reflects on both the practical implications and caveats of the findings.

8.1 Effectiveness of MILP-Based Optimization (RQ1)

The MILP-based formulation consistently achieved significant cost reductions across all configurations compared to the naive baseline. This validates the central hypothesis that sharing selling units across recipes results in a more cost-effective grocery plan. In the largest setup (3 pre-selected, 4 recommended), median savings exceeded 6 euros, with some extreme cases approaching 27 euros. Even in small configurations (e.g., 2 pre-selected, 3 recommended), the MILP model produced consistent savings, emphasizing the robustness of the formulation.

The solve time analysis shows that while the setups require more computation (median 20–23 seconds), the system remains usable in real-world applications, especially in scenarios where real-time feedback is not strictly required. For fixed-recipe plans, the model demonstrates sub-second runtimes, further confirming its practical viability.

Key takeaway. The results demonstrate that even small degrees of flexibility in recipe selection can yield substantial cost savings. The MILP model is particularly powerful when recipe redundancy and ingredient overlap exist. Additionally, as the number of pre-selected recipes increases, the opportunity for ingredient sharing also grows, resulting in greater absolute savings.

Limitations. A key limitation of the MILP-based approach is its runtime, although acceptable for offline use, the model is not yet suitable for real-time interaction in its current form. However, many practical concerns such as product availability, dietary restrictions, or user preferences are not fundamental limitations of the method. On the contrary, the MILP formulation is inherently flexible and can be extended with additional constraints to incorporate such requirements. This extensibility makes it well-suited for adapting to real-world deployment scenarios, even though these aspects were not explicitly addressed in the current experiments.

8.2 Solver Performance Comparison (RQ2)

Comparing CPLEX and Gurobi highlighted significant differences in runtime efficiency. Gurobi consistently solved the same MILP formulations faster and with lower variance, particularly in configurations with a higher number of recipes. This finding is critical for practical deployments, where solver speed can be a bottleneck.

Key takeaway. Gurobi appears to be the preferable choice when optimizing for runtime, especially in larger problem instances. It provides faster and more stable performance, making it more suitable for near real-time applications.

Limitations. This comparison is based solely on runtime under controlled conditions. Licensing constraints, integration complexity, and hardware-specific performance variations were not explored and could influence solver choice in practice. Also, solver performance might vary for other MILP formulations or datasets.

8.3 Heuristic Optimization using Genetic Algorithm (RQ3)

The Genetic Algorithm approach produced near-optimal solutions, with performance improving significantly as population size and generation count increased. Larger configurations (e.g., 400 generations, 400 individuals) produced median results within a few euros of the MILP optimum. However, these improvements came at a cost: solve time grew to over 10 seconds on average.

Key takeaway. The GA approach offers a tunable trade-off between quality and runtime. For interactive settings where an exact optimum is not necessary, the GA can yield sufficiently good solutions in under 2 seconds using lighter configurations (e.g., 100×100). This flexibility makes it a practical complement to MILP methods, especially when compute resources are limited.

To improve solution quality while preserving GA's flexibility, a hybrid model was introduced that uses GA to select recipes and MILP to compute the optimal product assignments. This model consistently outperformed the standalone GA, with smaller configurations (e.g., 100×100) yielding price differences—defined as the model's price minus the MILP optimal price—of less than €2. The hybrid setup combines GA's exploratory power with MILP's exact cost minimization, offering a practical solution for batch-processing scenarios where high accuracy is important.

Limitations. The GA implementation was developed from scratch and is not heavily optimized, which limits its runtime efficiency and scalability. While the method shows promise, its effectiveness is highly sensitive to parameter choices such as population size, mutation rate, and number of generations. Poor tuning may lead to suboptimal solutions, and the stochastic nature of GA introduces variability between runs. Moreover, GA provides no theoretical guarantee of optimality and can converge prematurely to suboptimal regions of the search space [Mitchell, 1998, De Jong, 2017]. These limitations are not specific to this implementation but are inherent to genetic algorithms more broadly. For structured, cost-driven problems like recipe optimization, GA may require careful calibration and performance benchmarking to ensure consistent quality across use cases.

The main drawback of the hybrid approach is runtime: since each chromosome requires solving a MILP instance, the approach becomes significantly more expensive as population size and generation count grow. For example, the 400×400 configuration averaged over 200 seconds per test case, making it unsuitable for real-time applications. Additionally, the method does not benefit from intermediate feedback (e.g., gradients or constraint violations), which limits the GA's ability to efficiently learn better patterns over time. Despite these issues, the hybrid strategy offers a strong balance of accuracy and adaptability when runtime constraints are relaxed.

8.4 Impact of Real-World Constraints (RQ4)

Incorporating real-world constraints revealed trade-offs between price, waste, and computational complexity. Notably, the *Weight+Price Min* (section 5.1.3) model achieved a compelling balance—delivering notable waste reductions with minimal cost increases and acceptable runtimes. In contrast, *Weight Min* (section 5.1.3) reduced waste most effectively but incurred a significant cost penalty. The *Cuisine Price Min* (section 5.1.3) model preserved affordability but had minimal effect on waste.

Key takeaway. Multi-objective optimization that includes sustainability goals is both feasible and effective. Specifically, the *Weight+Price Min* variant demonstrates that grocery recommendations can be aligned with environmental objectives without substantially increasing total cost or runtime.

Limitations. Real-world constraints introduce complexity that may not generalize across datasets or regions. The model assumes perfect product matching and ignores edge cases such as product discontinuity or inventory shortages.

In addition, some user-relevant constraints are not yet incorporated. These include dietary restrictions (e.g., vegetarian, vegan, low-carb), allergen avoidance (e.g., gluten, nuts), and perishability-aware planning (e.g., avoiding recipes that lead to spoilage of rare ingredients). While these constraints are conceptually compatible with the MILP framework, they would require careful modelling and user interface design to be effective in practice.

8.5 Summary

Overall, the experiments show that cost-aware meal planning benefits substantially from joint optimization approaches. The MILP model offers high-quality results at acceptable computational cost, particularly when powered by efficient solvers like Gurobi. The GA provides a valuable heuristic alternative for real-time applications, offering fast and flexible solutions with tunable quality.

The introduction of a hybrid GA + MILP model further improves performance by combining the exploration power of GA with the accuracy of MILP. This approach consistently narrows the gap to optimal solutions, though at the expense of longer runtimes. It proves particularly useful for offline or batch-processing scenarios where accuracy is more important than speed.

Sustainability constraints can be incorporated with modest compromises, making the system adaptable to both economic and environmental goals. Nonetheless, generalization and integration with live systems would require addressing personalization, preference modelling, data uncertainty, and runtime scaling. These present valuable directions for future research and real-world deployment.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

In the research we investigated whether recipe recommendations can be optimized to minimize total grocery costs while remaining computationally efficient and adaptable to real-world constraints. The motivation stemmed from a gap in Picnic's current system—despite the importance of recipes to both customers and operations, no algorithmic method was in place to make cost-effective recommendations. By exploring a range of optimization techniques, this research aimed to determine the feasibility of implementing such a system in a production environment.

The results demonstrate that the proposed approach is indeed feasible. Using a Mixed Integer Linear Programming formulation, we were able to produce optimized weekly meal plans that significantly reduce total costs compared to a naive baseline. Even in small-scale configurations, consistent savings were achieved, while larger configurations showed even greater potential. Thus, we conclude that MILP is highly effective at minimizing total recipe costs, answering **RQ1**.

To ensure scalability, we evaluated two leading MILP solvers—CPLEX and Gurobi. Gurobi consistently delivered faster and more stable solve times, particularly in more complex configurations, while maintaining the same solution quality. This supports the conclusion that solver choice has a notable impact on runtime performance, and Gurobi is preferable in time-sensitive applications (**RQ2**).

Recognizing the trade-offs between accuracy and runtime, we also implemented a Genetic Algorithm as a heuristic alternative. The GA was able to approximate optimal solutions within seconds, with performance improving significantly as population size and generation count increased. To further improve cost accuracy while maintaining flexibility, we introduced a hybrid approach that combines GA-based recipe selection with MILP-based product optimization. This hybrid model consistently outperformed standalone GA in terms of solution quality, although at the cost of significantly higher runtimes. It is therefore best suited for offline or batch-mode use cases where optimality is important and execution time is less critical (**RQ3**).

Finally, we explored the impact of integrating real-world constraints, such as minimizing food waste and enforcing cuisine consistency. The results show that waste-aware optimization is possible with only minor increases in price and solve time, especially when using a combined objective function. This confirms that the approach is adaptable and can be extended to support sustainability and user preference goals without compromising practicality (**RQ4**).

In conclusion, this research confirms that cost-efficient recipe recommendation is both computationally and operationally viable. The optimization pipeline developed in this thesis provides a foundation for real-world systems that help customers save money while enabling more efficient grocery operations. Furthermore, the approach is flexible and extendable, offering a promising direction for future personalization and sustainability efforts in online grocery platforms like Picnic.

9.2 Future Work

This work lays the foundation for cost-efficient recipe recommendation, but several promising directions remain open for further exploration.

First, while the current MILP formulation yielded strong results, there is room for improving both the efficiency and scalability of the model. Future work could explore alternative MILP formulations or decomposition techniques that reduce problem size or solve time. Additionally, integrating heuristics directly into the MILP solving process—such as warm-starting with near-optimal solutions—may further accelerate runtime, especially in dynamic or personalized settings.

Second, the Genetic Algorithm implementation shows that heuristic methods can approximate optimal solutions with significantly lower computational costs. However, the current version was developed from scratch and remains relatively unoptimized. Future efforts could benefit from using dedicated evolutionary computation libraries or implementing the algorithm in a more performance-oriented language. Additional GA strategies—such as elitism, adaptive mutation rates, or hybrid methods—may improve convergence speed and solution quality, making the GA more viable for real-time applications.

In the case of the hybrid GA+MILP model, parallelizing MILP evaluations across the population is a particularly promising direction, as it could significantly reduce the total runtime without altering the optimization logic. Other performance-enhancing techniques include memoization of MILP results for previously evaluated recipe sets, which would avoid redundant computation in later generations, and warm-starting the MILP solver using solutions from earlier iterations to reduce solve time. Additionally, the incorporation of convergence detection and early stopping criteria could help eliminate unnecessary MILP evaluations in late-stage generations, thereby improving runtime efficiency without sacrificing quality. Together, these extensions would increase the scalability and practicality of the hybrid model for batch and production scenarios.

Third, the experiments involving waste minimization revealed that modelling and optimizing for waste is inherently complex. One direction for future research is to develop a dedicated model that estimates and explicitly minimizes expected product waste, either as a primary objective or through better proxy functions. Alternatively, a novel approach would be to shift focus from recipe selection to packaging recommendations—helping analysts identify alternative product sizes or bundling strategies that minimize leftover ingredients across commonly chosen recipes. Such a system could complement recipe recommendation by informing upstream decisions in product assortment and packaging design.

In all cases, extending the system to support user personalization, dietary constraints, and preference learning represents a rich area for future work. While the current system focuses primarily on cost and waste, practical deployments would benefit from a multi-objective formulation that also accounts for user satisfaction and engagement.

Bibliography

E. Andersson, A. Peterson, and J. Törnquist Krasemann. Improved railway timetable robustness for reduced traffic delays–a milp approach. In *6th international conference on railway operations modelling and analysis-RailTokyo*, 2015.

H. M. Azamathulla, F.-C. Wu, A. Ab Ghani, S. M. Narulkar, N. A. Zakaria, and C. K. Chang. Comparison between genetic algorithm and linear programming approach for real time operation. *Journal of Hydro-environment Research*, 2(3): 172–181, 2008.

M. E. Buisman, R. Haijema, R. Akkerman, and J. M. Bloemhof. Donation management for menu planning at soup kitchens. *European Journal of Operational Research*, 272(1):324–338, 2019.

J. Clausen. Branch and bound algorithms-principles and examples. *Department of computer science, University of Copenhagen*, pages 1–30, 1999.

M. Cococcioni and L. Fiaschi. The big-m method with the numerical infinite m. *Optimization Letters*, 15(8):2455–2468, 2021. doi: 10.1007/s11590-020-01644-6.

M. Conforti, G. Cornuéjols, G. Zambelli, M. Conforti, G. Cornuéjols, and G. Zambelli. *Integer programming models*. Springer, 2014.

K. De Jong. Evolutionary computation: a unified approach. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 373–388, 2017.

F. M. de Souza Amorim, M. da Silva Arantes, M. P. de Souza Ferreira, and C. F. M. Toledo. Milp formulation and hybrid evolutionary algorithms for the glass container industry problem with multiple furnaces. *Computers & Industrial Engineering*, 158:107398, 2021.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, 6 (2):182–197, 2002.

R. Dondo and J. Cerdá. An milp framework for dynamic vehicle routing problems with time windows. *Latin American applied research*, 36(4):255–261, 2006.

S. Forrest. Genetic algorithms. *ACM computing surveys (CSUR)*, 28(1):77–80, 1996.

J. D. Foster, A. M. Berry, N. Boland, and H. Waterer. Comparison of mixed-integer programming and genetic algorithm methods for distributed generation planning. *IEEE transactions on power systems*, 29(2):833–843, 2013.

S. Gestrelius, M. Aronsson, and A. Peterson. A milp-based heuristic for a commercial train timetabling problem. *Transportation Research Procedia*, 27:569–576, 2017.

A. T. Gumus, A. F. Guneri, and S. Keles. Supply chain network design using an integrated neuro-fuzzy and milp approach: A comparative design study. *Expert Systems with Applications*, 36(10):12570–12577, 2009.

L. Huang, X. Chen, W. Huo, J. Wang, F. Zhang, B. Bai, and L. Shi. Branch and bound in mixed integer linear programming problems: A survey of techniques and trends. *arXiv preprint arXiv:2111.06257*, 2021.

C. U. IBM CPLEX. Ibm ilog cplex optimization studio. *Version*, 12(1987-2018): 1, 1987.

J. Jablonskỳ et al. Benchmarks for current linear and mixed integer optimization solvers. *Acta Universitatis Agriculturae et Silviculturae Mendelianae Brunensis*, 63(6):1923–1928, 2015.

T. Kashima, S. Matsumoto, and H. Ishii. Evaluation of menu planning capability based on multi-dimensional 0/1 knapsack problem of nutritional management system. *IAENG International Journal of Applied Mathematics*, 39(3):163–170, 2009.

P. Leung, K. Wanitprapha, and L. A. Quinn. A recipe-based, diet-planning modelling system. *British Journal of Nutrition*, 74(2):151–162, 1995.

G. Liu, M. F. Ferrari, T. B. Ollis, and K. Tomsovic. An milp-based distributed energy management for coordination of networked microgrids. *Energies*, 15 (19):6971, 2022.

S. Madankumar and C. Rajendran. A mixed integer linear programming model for the vehicle routing problem with simultaneous delivery and pickup by heterogeneous vehicles, and constrained by time windows. *Sādhanā*, 44:1–14, 2019.

B. Meindl and M. Templ. Analysis of commercial and free and open source solvers for linear optimization problems. *Eurostat and Statistics Netherlands within the project ESSnet on common tools and harmonised methodology for SDC in the ESS*, 20:64, 2012.

L. Meng, C. Zhang, Y. Ren, B. Zhang, and C. Lv. Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem. *Computers & Industrial Engineering*, 142: 106347, 2020. ISSN 0360-8352. doi: https://doi.org/10.1016/j.cie.2020.106347.

M. Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

M. Padovan, F. R. de Senna, J. K. Kimura, S. T. Nascimento, A. C. Moretti, and C. D. Capitani. Optimized menu formulation to enhance nutritional goals: design of a mixed integer programming model for the workers' food program in brazil. *BMC nutrition*, 9(1):51, 2023.

A. Popescu, S. Polat-Erdeniz, A. Felfernig, M. Uta, M. Atas, V.-M. Le, K. Pilsl, M. Enzelsberger, and T. N. T. Tran. An overview of machine learning techniques in constraint solving. *Journal of Intelligent Information Systems*, 58(1):91–118, 2022.

J.-M. Ramos-Pérez, G. Miranda, E. Segredo, C. León, and C. Rodríguez-León. Application of multi-objective evolutionary algorithms for planning healthy and balanced school lunches. *Mathematics*, 9(1), 2021. ISSN 2227-7390. doi: 10.3390/math9010080.

M. A. Rodriguez, A. R. Vecchietti, I. Harjunkoski, and I. E. Grossmann. Optimal supply chain design and management over a multi-period horizon under demand uncertainty. part i: Minlp and milp models. *Computers & Chemical Engineering*, 62:194–210, 2014. ISSN 0098-1354.

D. Sklan and I. Dariel. Diet planning for humans using mixed-integer linear programming. *British Journal of Nutrition*, 70(1):27–35, 1993.

J. C. Smith and Z. C. Taskin. A tutorial guide to mixed-integer programming models and solution techniques. *Optimization in medicine and biology*, pages 521–548, 2008.

J. P. Vielma. Mixed integer linear programming formulation techniques. *Siam Review*, 57(1):3–57, 2015.

S. Wang, K. Xia, Y. Yang, R. Qiu, Y. Qi, Q. Miao, W. Xie, and T. Liu. A recommender system for healthy food choices based on integer programming. In *Applied Mathematics, Modeling and Computer Simulation*, pages 469–475. IOS Press, 2022.

L. A. Wolsey and G. L. Nemhauser. *Integer and combinatorial optimization*. John Wiley & Sons, 1999.

N. Zaree and V. Vahidinasab. An milp formulation for centralized energy management strategy of microgrids. In *2016 smart grids conference (SGC)*, pages 1–8. IEEE, 2016.

E. Zitzler. Evolutionary algorithms for multiobjective optimization. *Evolutionary Methods for Design, Optimisation, and Control, CIMNE*, pages 19–26, 2002.

E. Zitzler and S. Künzli. Indicator-based selection in multiobjective search. In *International conference on parallel problem solving from nature*, pages 832–842. Springer, 2004.