EVOLUTIONARY COMBINATORIAL OPTIMIZATION ON THE GRAIN MIXING PROBLEM

by

Md Asaduzzaman Noor

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

May 2022

DEDICATION

This thesis is dedicated to my wonderful parents who have raised me and supported me to pursue my dream abroad, my loving wife for encouraging me throughout the whole process, and my brothers and friends for keeping faith in me.

ACKNOWLEDGEMENTS

First of all, I am extremely grateful to my advisor Dr. John Sheppard for his invaluable advice, continuous support, and patience during my study. His immense knowledge and plentiful experience have encouraged me in all the time of my academic research.

I would also like to thank my committee members: Dr. Mike Wittie and Dr. Sean Yaw, for guiding me and helping me throughout the process. My special thanks to Dr. Sean Yaw for helping me with the NP-Hardness proof and the mathematical formulation of the problem, and Dr. Binhai Zhu for sharing his ideas and helping me out with the complexity proof.

Finally, I would like to thank the members of the Numerical Intelligent Systems Laboratory (NISL) at Montana State University - specifically, Jordan Schupbach, Na'Shea Wiesner, Amy Peerlinck, Georgio Morales, and Kyle Webster for their insightful comments and suggestions.

TABLE OF CONTENTS

TABLE OF CONTENTS – CONTINUED

TABLE OF CONTENTS – CONTINUED

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

ABSTRACT

Combinatorial optimization is an important area in computer science that uses combinatorics to solve discrete optimization problems. In this thesis, we considered a combinatorial optimization problem in the wheat supply chain known as grain mixing. The grain mixing problem involves mixing two or more collections of grain with different protein content to produce collections of grain with a weighted average protein content that improves the overall profit to the farmer. The presence of non-linearity in the objective function and some of the constraints in the grain mixing problem makes the problem difficult to solve exactly using linear programming (LP) or mixed-integer LP models. First, we presented an NP-Hardness proof for the grain mixing problem to justify the use of approximation algorithms. Then we explored several approaches to solve the grain mixing problem. For the approximation algorithms, we adapted two evolutionary approaches (EA) for the grain mixing problem: Genetic Algorithm (GA) and Differential Evolution (DE). We developed a pseudo-permutation-based representation for the EAs for which the conventional crossover operator for GA and mutation operator for DE had to be adapted to fit our problem representation. Specifically, we adapted two crossover operators for GA: Ordered Crossover (OX) and Partially Mapped Crossover (PMX), and two discrete mutation operators for DE: Relative Position Indexing (RPI) and Global Best Perturbation (GBP). Moreover, we introduced and compared three baseline approaches: *no* mixing, *greedy* mixing, and *random* mixing to evaluate the solution quality of the proposed EA approaches. The experimental results demonstrate that solutions obtained from the evolutionary approaches consistently provided a higher overall profit compared to the non-evolutionary baseline methods for both real and simulated datasets. It also suggests that grain mixing is beneficial to improve farmers' overall wheat selling profitability.

CHAPTER ONE

INTRODUCTION

Optimization is the process of making the best choice out of many options. It has a wide range of applications arising in different disciplines such as computer science and engineering, operations research, economics, etc. In mathematical terms, optimization or mathematical programming is the process of finding the solution to some objective function (e.g., cost, time, profit) out of all other alternatives that either minimizes or maximizes the given objective under some criteria (or constraints). The input variables in the optimization domain can be finite/discrete or continuous. The problem for which the variables only contain a finite set of discrete values are often modeled as combinatorial optimization problems (COP). The traveling salesperson problem (TSP), bin packing, and job-shop scheduling are well-known examples of COPs [10].

There exists a variety of techniques for solving COPs. Linear programming (LP) model can be used for certain special classes of COPs, however, it requires the objective function and the constraints to be linear [48]. Some example problems that fall into this category are the shortest path, maximum flow, spanning tree, and matching problems [43]. For many real-world COPs, all/some decision variables may be restricted to be only integers where Integer Programming (IP) or Mixed IP models are often used to solve the problem. Restricting variables to contain only integer values often makes IP/MIP NP-Hard [39].

Since the integer constraints destroy the convexity of the feasible regions of the objective function, MIP solvers often use an approximation called LP-relaxation. However, for some problems, relaxing the integrity constraints often lead to an LP solution that is far from the actual integer solution. The solution quality of the MIP also depends on the mathematical

representation of the problem as there may be different representations for the same problem. Moreover, for many real-world problems, the objective function and the constraints may have non-linearity which cannot be modeled efficiently by only using LP relaxations.

Dynamic Programming (DP) is another approach for solving some COPs that can provide an exact solution. For example, it can be used to solve COPs such as 0–1 Knapsack, Subset–Sum, and Partition problems [68] that runs in pseudo-polynomial time. However, to be represented by DP, a problem should have an optimal substructure and overlapping subproblems [68]. Therefore, not all problems can be modeled efficiently using DP. There exist some approximation algorithms that can provide approximation-guaranteed suboptimal solutions for many intractable COPs. For example, greedy, randomized, or local search algorithms [74, 76] can be used to solve problems such as TSP, job-shop scheduling, and satisfiability.

Meta-heuristics are another approximation approach for solving intractable COPs but generally fail to provide any optimality guarantees [7]. They are a well-known and promising approach for hard COPs (especially for large-scale problem instances) to find a quality solution with decent computational time. Evolutionary algorithms (EA) such as Genetic Algorithm (GA) [28] and Differential Evolution (DE) [70] fall into the class of meta-heuristics that uses the biological principle of evolution to explore the search space for finding optimal or near-optimal solutions. EAs are an iterative stochastic approach that starts with an initial population of randomly generated solutions and uses recombination operators (i.e., crossover and mutation) to improve the solution quality. Although EA approaches depend on the particular representation of a COP, they do not put any restrictions on how the objective function is defined. Therefore, one possible benefit over MIP models is that no linearization of the objective is required. Consequently, for some real-world COPs with non-linearity, they are the most feasible approach.

1.1 Problem Definition

In this thesis, we considered an important optimization problem in the wheat supply chain, referred to as grain mixing. The wheat considered in our case is mainly winter wheat that accounts for approximately 70% of the wheat production in the US [9]. The lifecycle of winter wheat starts with planting that takes place from mid-August through October. Then there is a dormant period from November to March. Harvest takes place from mid-May to mid-July of the following year. The farmers store the harvested grain into several bins and wait until either the moisture level is perfect or the elevator price is right. Then they transport the grain via trucks in batches to sell the wheat to multiple local grain elevators depending on the price. The elevators then send the wheat to either domestic or foreign mills where the grain is processed into flour and eventually to baked goods for food.

Several factors come into play when determining the profit from wheat production. Here, we only considered the distribution of wheat from a local farm site to the local grain elevators. Usually, farmers load trucks with grain in batches and head to the nearest elevator to sell the grain. The elevator then weighs each truck and performs a quality check of the grain in that truck. The quality check involves checking several attributes such as moisture level, the protein content of wheat, etc., and based on the quality check, assigns a price grade for each truck.

The end-use functionality and the commercial value of wheat heavily depend on the grain protein content (GPC, %) in the wheat [77]. Therefore, in some states, such as Montana, the price the farmers get for selling wheat at the local grain elevators depends heavily on the protein content of the wheat. However, a combination of environmental factors such as temperature during the growing season, soil nitrogen levels, plant genetics, timing, and precipitation affect the grain protein content in crop fields [69]. Due to these factors, protein content in wheat changes not only from year to year but also from crop to

crop. In fact, there can even be significant protein variation within a field [4, 69]. Integration of remote sensing and proper crop planning may help improve the overall protein content in the field [77], however, maintaining a uniform protein content across the field remains a challenge. Therefore, after harvesting, the farmers end up storing grains of varying protein content in their bins.

Due to the advent of agricultural technology, it is now possible to monitor the protein content of wheat during harvest [81]. It is also possible to mix different quality wheat to change the average protein content. When selling wheat to a local grain elevator, the price per bushel depends on a range of protein levels. Therefore, by mixing different quality wheat, it might be possible to improve the price grade of lower quality wheat without hampering the price of higher quality wheat. However, the protein tracking device is expensive (usually cost around $4000 to $5000 US dollars), making it inaccessible to several small farmers. Moreover, mixing grain to change the average protein content also involves a mixing cost, and it is often hard to come up with a quality mixing plan that will increase the overall profit. Consequently, most wheat producers end up taking their harvest to the closest elevators and collecting whatever amount is paid to them. Therefore, the grain mixing problem can be represented as an optimization problem where the objective is to maximize the overall wheat selling profitability while minimizing the associated cost considering the real-world constraints.

There are many constraints involved in the grain mixing problem. For example, mixing grain from grain bins incurs an additional mixing cost, and mixing is restricted to two bins at a time due to farmers' physical limitations. Additionally, the mixing cost between bin pairs differs based on the physical location of the grain bins. Furthermore, grain bins contain varying amounts of bushels, and fixed capacity trucks are used to transport grain into the elevators that involve a delivery cost depending on the distance of the elevator. In addition, the elevator protein cost function (i.e., the objective function) is a non-linear step function

with an inflection point at the center that makes the problem even harder to find the optimal mixing plan using polynomial-time solvers such as Linear Programming models.

1.2 Contribution

For the grain mixing problem, the solutions can be represented as a set of bin-pair combinations with the appropriate mixing ratio to load each truck without violating the capacity constraints, and the objective is to find the optimal set of combinations for which the total profit is maximized. As the number of storage bins used by the farmers is finite, the grain mixing problem can be modeled as a combinatorial optimization problem (COP). In this thesis, we explored several approaches to find a quality mixing plan for the grain mixing problem. The major contributions of this thesis are as follows.

First, we provide a formal mathematical programming specification for the grain mixing problem. On the surface, it may appear that the mathematical formulation of the problem can be solved using MILP solvers. However, the inherent non-linearity in the objective function and some of the constraints make the problem difficult to solve exactly using MILP solvers. Therefore, as our second contribution, we provide a theorem to show that the grain mixing problem studied here is NP-Hard. The theorem helps us assess that using exact algorithms to find the optimal mixing plan may not be feasible, and we may need to rely on meta-heuristic algorithms such as evolutionary algorithms to find (near)-optimal solutions.

For the meta-heuristic algorithms, we utilize constrained evolutionary approaches, specifically a Genetic Algorithm (GA) and Differential Evolution (DE), to obtain a quality mixing plan for the grain mixing problem. We introduce a novel permutation-based representation of the grain mixing problem for the evolutionary algorithms. It allows some of the constraints of the problem not to be violated in the first place. For example, restricting the mixing to two bins at a time. For permutation-based COPs such as the traveling salesman problem and job-shop scheduling, usually, the problem representation contains the exact

permutation of cities or jobs [30, 60]. However, our grain mixing problem representation does not follow a strict permutation of the bin pair and mixing ratio. Therefore, the conventional permutation-based crossover operators in GA and the differential mutation operators in discrete DE do not fit directly to our problem representation.

Consequently, our third contribution is an adaptation of two well-known permutation-based crossover operators - Ordered Crossover (OX) [16] and the Partially Mapped Crossover (PMX) [29] operator for GA for our pseudo-permutation-based representation. Similarly, for DE, we adapted the Relative Position Indexing (RPI) [46] mutation operator and introduced another discrete mutation operator called the Global Best Perturbation (GBP). We argue that the adaptation of these operators may be beneficial for solving similar COPs whose representation does not follow strict permutation.

To evaluate the performance of our proposed evolutionary approaches (EA) for the grain mixing problem, we employed three baseline algorithms. The first one is called the *NoMix* algorithm that provides the baseline profit the farmers get without any grain mixing. This algorithm helps us assess if grain mixing is beneficial in the first place to improve the overall profitability. Then we introduce a deterministic *greedy* mixing approach to obtain a mixing plan that greedily selects bin pair combinations based on the maximum profit. This method was chosen to assess if the added complexity of the EAs is advantageous for improving the solution quality. Finally, we used a stochastic *Random* search algorithm that selects bin pair combinations for loading trucks randomly and returns the best solution. The *Random* algorithm helps us assess if the proposed EAs are exploring the search space efficiently to improve the solution quality rather than just random search.

Given the above, we hypothesize that the *greedy* mixing and *Random* search approaches will yield a higher profit than the baseline *NoMix* profit as the expectation is that mixing grain bins should provide a better solution. We further hypothesize that the GA and DE algorithms with different recombination operators will yield higher profit than the

deterministic *greedy* and the stochastic *Random* search approaches as they provide a more comprehensive search of the search space. For the proposed GA and DE algorithm, both algorithms have been adapted to fit the problem definition (constraint validation), and we hypothesize that there will be no significant differences in the solution quality obtained from different variations of the algorithms.

1.3 Organization

The remainder of the thesis is outlined as follows. Chapter 2 provides an overview of general combinatorial optimization problems (COPs) and possible solution approaches for solving COPs. Then, an overview of the Genetic Algorithm (GA) is presented in the context of solving COPs with the example of two crossover operators (i.e., OX and PMX) for traveling salesperson problem representation followed by the Differential Evolution (DE) with two discrete differential mutation operators (RPI and GBP).

Chapter 3 provides an overview of the grain mixing problem with a simple example to show how mixing grain can be beneficial to improve the overall wheat selling profit and how it creates the scope for optimization. Then we provide a mathematical formulation for the grain mixing problem and show why the problem is not straightforward to be used directly in a MILP solver. Then we prove that the grain mixing problem is NP-Hard following a reduction from the 3-Dimensional Matching problem. For the proof, we assume that each elevator has a fixed number of bushels that they will accept. Although this assumption is not present in the general grain mixing problem, the proof provides an idea of the hardness of the more general grain mixing problem studied here. In addition, we provide the dataset description with the non-linear elevator protein cost functions that were used across all of the experiments. Finally, we provide an overview of the existing approaches that were used in the literature to solve related optimization problems.

In Chapter 4, we present our approach for calculating the baseline *NoMix* profit followed

by the deterministic *greedy* mixing and the stochastic *Random* search approaches. Then we show example solutions along with a detailed profitability analysis between these approaches.

Chapter 5 presents our constrained GA approach for the grain mixing problem. In this chapter, we explain our specific implementations of the permutation-based problem representation, fitness function, and the adapted crossover operators (i.e., OX, and PMX). Then we show examples of the best solutions obtained by different crossover operators and provide a comparative analysis of the results with the baseline approaches along with a detailed analysis of the GA population diversity.

The permutation-based problem representation and the fitness function for the DE approach are similar to the GA. However, due to the differences in search mechanism, the differential mutation, crossover, and selection operators are different in DE. The specific implementations of these DE operators are discussed in Chapter 6. In addition to presenting example results for the DE variations and diversity analysis, we provide a complete profitability analysis of all the approaches used for grain mixing in this chapter. Finally, we conclude our thesis in Chapter 7 by summarizing our contributions and discussing the extent to which our hypothesis has been supported or refuted, followed by a discussion of possible future works.

CHAPTER TWO

BACKGROUND

In this chapter, we present the necessary concepts and background for the methodology used in this study.

2.1 Combinatorail Optimization Problem (COP)

Combinatorial optimization is the process of maximizing or minimizing an objective function of a discrete, large configuration space domain [7, 53, 58]. The objective function is often subject to some equality and/or inequality constraints and integrity restrictions on many variables. For clarity, We assume a maximization problem without loss of generality unless specified otherwise. More formally, a COP, $P = (\mathbf{S}, f)$ can be expressed by a set of variables $\mathbf{X} = \{x_1, ..., x_n\}$ with variable domains $\{\mathbf{D_1}, ..., \mathbf{D_n}\}$, a set of constraints among variables, and an objective function f to be maximized such that $f : \mathbf{D_1} \times ... \times \mathbf{D_n} \rightarrow \mathbb{R}$. The set of all feasible solutions is expressed as

$$\mathbf{S} = \{\mathbf{s} = \{v_1, ..., v_n\} | v_i \in \mathbf{D_i} \text{ and } \mathbf{s} \text{ satisfies all the constraints}\}$$

where $\mathbf{S}$ is known as the solution space. The objective is to find the best set $\mathbf{s}^*$ that maximizes the objective function value such that $f(\mathbf{s}^*) >= f(\mathbf{s}) \quad \forall \mathbf{s} \in \mathbf{S}$ and $\mathbf{s}^*$ is called the globally optimal solution for $P = (\mathbf{S}, f)$.

Combinatorial optimization is used to represent a variety of problems in many fields. In operations research, it is used to solve problems like efficient distribution of goods, machine sequencing, and production scheduling [12, 53]. It has also been used for planning problems such as portfolio analysis, determining facility locations, and design problems like an automated production system, VLSI circuit design, etc [7, 80]. In mathematics and

computational complexity theory, several applications in the domain of combinatorics, graph theory, and logic are represented by combinatorial optimization problems [58]. The feasible solutions in the combinatorics domain are often expressed as sets, subsets, combinations, or permutations of the variables whereas, in graph theory, they are expressed as vertices, edges, cliques, paths, cycles, or cuts.

There exist many combinatorial optimization problems in the literature that can be solved using polynomial-time algorithms. For example, some such problems can be modeled as linear programming problems and can be solved exactly in polynomial time [48]. Some example problems that fall into this category are the shortest paths, maximum flows, spanning trees, matching, and matroid problems [43]. However, for many real-world problems, the space of possible solutions is often too large to search exhaustively using brute force methods, and their formulation does not fit the Linear Programming framework. For some problems, they are modeled as integer or mixed-integer linear programming problems [53] and can be solved exactly using Branch and Bound algorithms [52]. For other problems, where the exact solution is not feasible, often approximation algorithms are used that have polynomial running time to input size and provide optimal or near-optimal solutions. An approximation algorithm's performance depends on how quickly it can provide a solution and what is the best solution quality it can guarantee. Example approximation algorithms that can provide approximation-guaranteed suboptimal solutions for some intractable problems include greedy, sequential, and local search algorithms [74, 76] and dynamic programming algorithms [68]. Heuristic/Metaheuristic approaches have also been used as approximation algorithms for solving NP-Complete COPs [7]. Some of the most used metaheuristic approach include random-restart hill-climbing [66], simulated annealing [42], evolutionary algorithms [40], and tabu search [25]. Example problems that fall into the NP-complete COPs are the Traveling Salesperson Problem (TSP), Bin-packing, Job-Shop Scheduling, Boolean Satisfiability, etc [23].

The grain mixing problem studied here can be viewed as a permutation-based combinatorial optimization problem. In general, permutation-based problems are those where the solutions are encoded as permutations and the goal is to find the best set from all possible solutions for which a specific objective function is maximized. In the grain mixing problem, the solutions can be represented as a set of bin pairs and mixing ratio combinations to load trucks, and the objective is to find the optimal set of combinations for which the total profit is maximized. The order in which the bin-pair are loaded into a truck matters due to the delivery cost, which is what makes it a permutation-based problem. Other well-known permutation-based COPs are TSP, Flow Shop Scheduling Problem (FSSP), Quadratic Assignment Problem (QAP), etc. [10]. Most of these problems are known to be NP-complete and rely on approximation algorithms rather than exact algorithms. Evolutionary algorithms such as genetic algorithms (GA) and differential evolution (DE) have shown to be efficient algorithms for solving permutation-based COPs in the literature [8, 54, 63]. We utilized the evolutionary approach for solving the grain mixing problem. The next two sections provide the background for GA and DE for solving permutation-based COPs.

2.2 Genetic Algorithm

A Genetic Algorithm (GA) is a stochastic search algorithm that mimics the evolutionary process of natural selection and the "survival of the fittest." GA was first introduced by John Holland [34] and belongs to the larger class of evolutionary algorithms (EA). In the theory of evolution, individuals with better adaptation to the surrounding environment have a higher chance of survival and reproduction, while the less fit individuals will be eliminated in the process. By sharing good characteristic genes through the process of mating, two individuals may produce even more fit offspring for the next generation. Over the course of several generations, the species become more and more adapted to the environment.

A GA applies the same process for solving an optimization problem. It starts with an

initial population of individuals where each individual represents a potential solution to the problem. The fitness of an individual is evaluated based on the objective function of the problem. Then a selection process takes place where the highest fit individuals have a better chance to participate in the mating pool for producing offspring for the next generation. A crossover operator is applied to create new offspring (i.e., children) that share some common gene characteristics from its parents. A mutation operator is then applied that alters some genes in the individuals that help GA to prevent converging in the local optima and provides exploration in the search space. The process of fitness evaluation, selection, crossover, and mutation continues until a satisfactory solution is found or some termination criterion is met. Figure 2.1 shows the classical GA flowchart.

The search mechanism of GA consists of the following components. 1) Population representation, 2) Fitness evaluation, 3) Parent selection, 4) Genetic operators such as crossover and mutation. In the following sections, we discussed each component of GA in the context of solving COPs.

2.2.1 Population Representation (Encoding)

GA starts the search process with an initial population of individuals. Each individual represents a potential solution to the optimization problem that it is trying to solve. In the context of COP, the solution may be represented as a set of parameters/variables that only takes discrete values. For example, let us consider an objective function $f(x_1, x_2, ..., x_m)$ of M discrete parameters that we want to maximize. An individual solution will be the set of values of these M parameters. Each parameter in the individual solution is known as a gene and the string of genes is known as the chromosome. The goal of GA is to find the individual with the optimal set of parameters that maximizes the objective function f.

The chromosome representation often depends on the nature of the COP. In classical GA, strings of binary values (0–1) are usually used to represent the chromosomes. The same

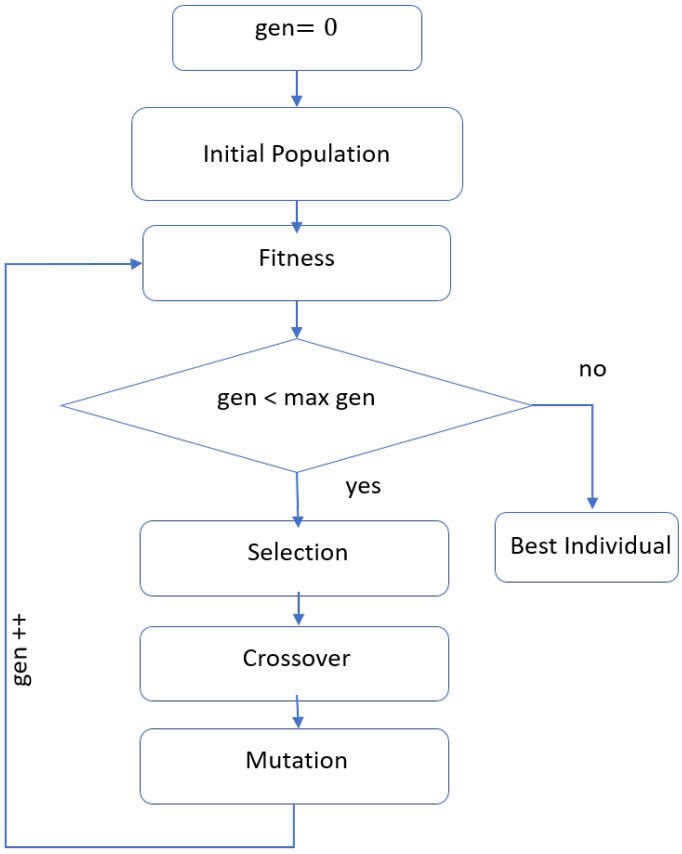


Figure 2.1: Genetic Algorithm Flowchart

representations can be applied to COPs with 0–1 Integer Programming (IP) formulations such as the 0–1 knapsack problem [67]. However, the binary representation is not always suitable for many COPs. For permutation-based COPs, the permutation of a finite set of variables is often used to represent the chromosome. For example, in the Traveling Salesperson Problem (TSP), the permutation of n cities is usually used to design the chromosome [30]. The initial population is randomly generated most of the time, however, heuristic approaches can also be used to find a better set of initial solutions that speeds up the search process for GA.

2.2.2 Fitness Function

In GA, the fitness function evaluates how good an individual's chromosome representation is for solving the given optimization problem. It provides a single numerical score (i.e., fitness) that reflects the 'quality' of the solution in the whole solution space. An individual with a higher fitness score will have a better chance of reproducing and surviving in the next generation. For optimization problems, a simple way to define the fitness function is to use the objective function. In that case, the fitness of the individual can be calculated by assigning the chromosome values to each variable of the objective function.

For COPs with one or more constraints, the objective function alone may not be suitable for representing the 'true' fitness of an individual. Due to the constraints, most of the solutions in the search space may be infeasible. Therefore, a modified fitness function is used to deal with the infeasible solution. Several approaches have been proposed in the literature for handling constraints in GA [78]. One approach is to introduce a penalty function that penalizes the infeasible individuals based on the magnitude of the constraint violation [35]. Another approach is to ensure feasibility in the chromosome representation [49]. To do that, a 'repair' method is applied that modifies the structure of the infeasible individuals to make them feasible.

2.2.3 Parent Selection

The selection of parents plays a major role in directing the GA's search process. Selection is the process of choosing individuals from the current population for mating to produce offspring for the next generation. The idea of selection relates to the fitness of an individual. It reduces the genotypic diversity in the population by selecting particular individuals based on fitness thus narrowing the search space and eventually converging in a particular region of the search space.

The original selection scheme proposed by Holland is commonly known as the 'Roulette-

Wheel' method [34] where the selection of parents is proportional to the actual fitness values of the individuals. One downside of this method is that if two parents have high fitness values (but not optimal) compared to the other individuals then the selection would be heavily biased towards the fittest individuals leading to premature convergence. To overcome the problem with extreme individuals, another approach known as the 'Ranking-based' method [5, 26] is proposed where the individuals were ranked according to their fitness, and probabilities were assigned based on the ranking instead of the exact fitness. Although it loses some information in the process, it is a simple and efficient selection procedure.

Another alternative to strict fitness-proportionate selection scheme is the 'Tournament' selection [50]. A k-tournament selection operator randomly selects k individual from the current population, and the individual with the highest fitness (tournament winner) is selected as a parent for reproduction. The process continues until u parents are selected. For a comparative analysis of these methods along with other selection methods, please see [27]. As the selection is related to the individuals' fitness, these selection methods can be applied directly for solving COPs.

2.2.4 Crossover

The crossover operator replaces some of the genes of one parent with corresponding genes from another parent to create a new offspring. The idea is that combining good genes from the parents will yield offspring providing a better solution over time. In a traditional GA, crossover operators such as one-point crossover, two-point crossover, and uniform crossover are often used [15]. In one-point crossover, a random crossover point is selected in the parent string and then segments of the two parent strings are swapped to produce one or two child strings. In two-point crossover, two crossover points are selected randomly and crossover is applied accordingly. In uniform crossover, each gene in the child string is created by copying the corresponding gene from one or the other parent following

a binary random number generator.

For permutation-based COPs such as TSP, traditional crossover operators like one-point crossover, two-point crossover, and uniform crossover are not often suitable. Therefore, crossover variants such as ordered crossover, partially mapped crossover, cyclic crossover, etc., have been proposed for handling permutation-based problems [36, 60]. For this thesis, we used the ordered and partially mapped crossover operators, which are similar to those used with the TSP representation; however, we adapted both to fit our problem representation.

2.2.4.1 Ordered Crossover (OX) The OX operator was first introduced by Davis [16] as an alternative operator for solving the 2-D bin packing problem. The mechanism of the OX operator has been shown to be an effective operator for solving permutation-based problems such as TSP [55]. To demonstrate OX crossover, we will use the most common TSP representation where the cities are represented as integer vertices, and a legal tour is represented by a sequence of vertices. For example, let us consider a TSP instance with 8 cities. If $4 \rightarrow 1 \rightarrow 2 \rightarrow 5 \rightarrow 8 \rightarrow 6 \rightarrow 7 \rightarrow 3 \rightarrow 4$ is a legal tour, then the chromosome can be represented as (4 1 2 5 8 6 7 3).

Figure 2.2 demonstrates generating new offspring using the OX operator. First, it selects a random crossover point between two chosen parents. In the next step, it creates two empty offspring and copies the gene of the parents until the crossover point. In this example, offspring 1 copies genes from parent 1 and offspring 2 copies genes from parent 2. Then offspring 1 marks the genes in parent 2 that are already present in the sequence and copies the rest of the genes from parent 2 in the order they occur in the sequence (excluding the marked ones). Therefore, the OX operator respects the relative orders of the genes from parents when generating offspring. It also creates the offspring 2 analogously.

2.2.4.2 Partially Mapped Crossover (PMX) The PMX operator was first introduced by Goldberg and Lingel [29], and for some problems, it provides better performance than

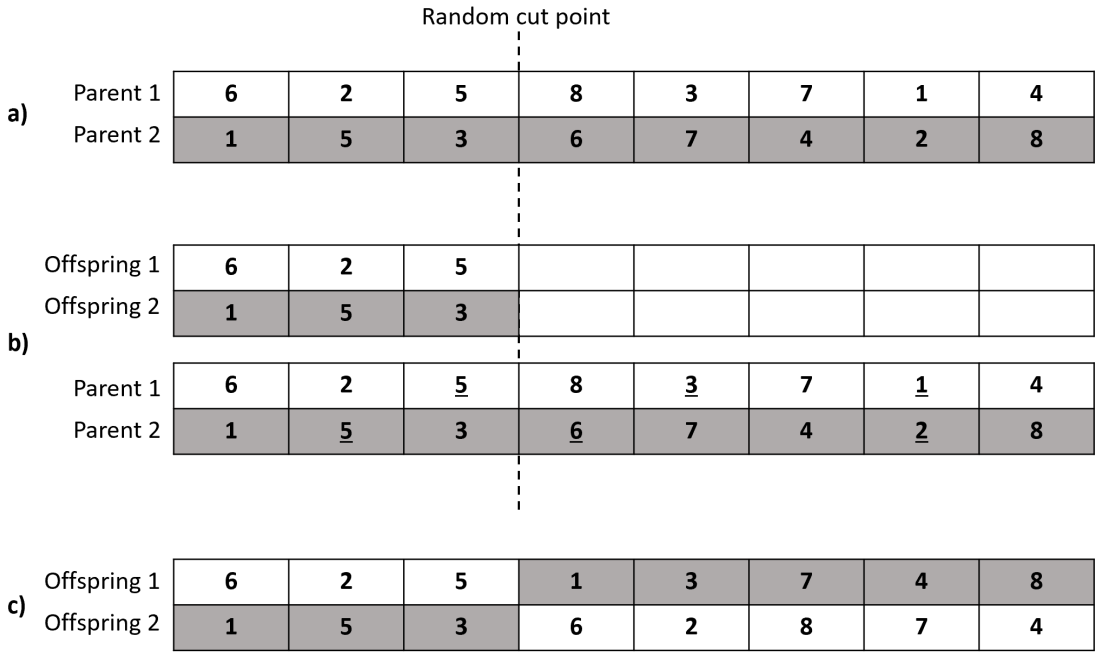


Figure 2.2: Order crossover example on two TSP parents

other crossover operators. Figure 2.3 demonstrates the offspring generation using the PMX operator. First, it selects two random cut points in the parent's chromosome that forms a substring. For the next step, offspring 1 copies the substring of parent 2 (and offspring 2 copies the substring of parent 1) and creates a partial mapping of the substring. Then, offspring 1 further fills the sequence with genes from parent 1 that do not have any conflict. Finally, it uses the mapping to resolve the conflicts to generate legal offspring. For example, in Figure 2.3(b), genes 6 and 7 are conflicting in offspring 1, so the mapping ($8 \leftrightarrow 6$) is used to replace gene 6 with gene 8. Similarly, gene 5 replaces gene 7 (following the double mapping $3 \leftrightarrow 7, and\ 5 \leftrightarrow 3$). The mapped values are represented with underlined vertices in the final offspring. It then creates the offspring 2 analogously.

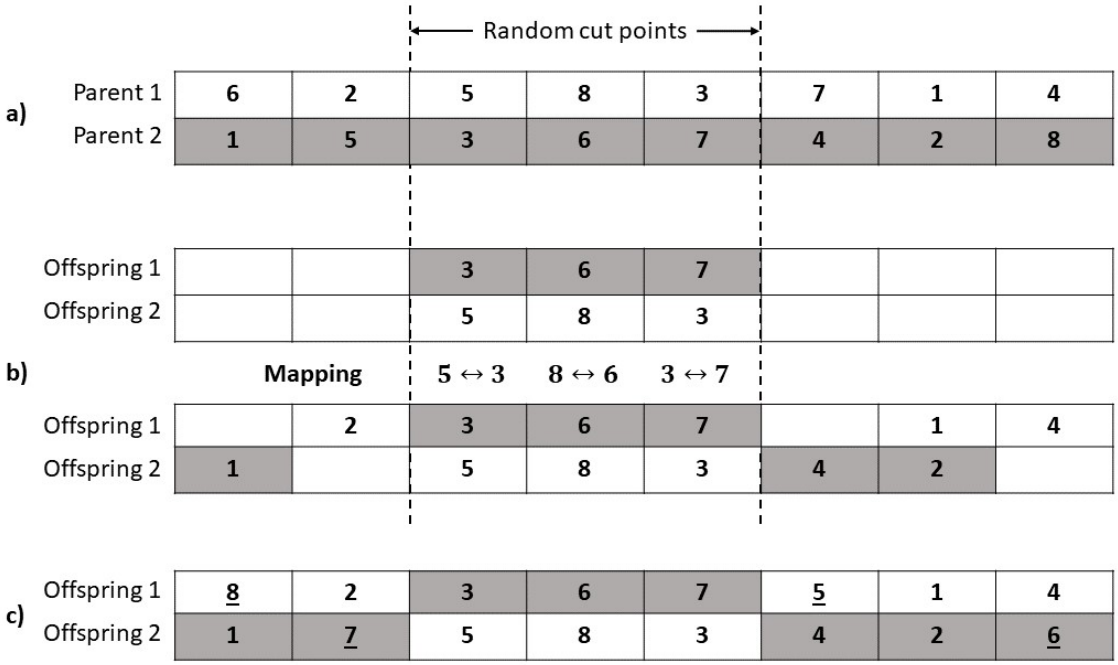


Figure 2.3: Partially mapped crossover example on two TSP parents

2.2.5 Mutation

The mutation procedure helps GA to maintain and introduce population diversity. It relates to the term 'Exploration' in the search space and randomly tweaks/alter some genes in the chromosome to get a new solution. For a binary representation, the mutation operator flips each bit in the chromosome with a small probability.

For COPs, applying traditional mutation operators may not always be meaningful. Moreover, it may lead to infeasible solutions. Therefore, different variants of the mutation operator have been proposed in the literature for COPs. For example, for permutation-based encoding, swap mutation is a common approach that swaps the positions of the two randomly selected genes in a chromosome [60]. Other mutation operators for permutation-based representation include Insert, Inversion, Scramble, Translocation, etc [19].

2.3 Differential Evolution (DE)

DE a population-based search mechanism introduced by Storn and Price [70] for solving continuous optimization problems. DE is computationally simple and a competitive search method for many real-world test problems. The canonical DE was designed for solving problems defined in continuous space. However, this is somehow limiting to DE, and researchers have proposed various techniques to adapt the standard DE to solve permutation-based COPs over the years. First, we will briefly describe the search mechanism of canonical DE for solving continuous optimization problems. Like GA, DE also starts with a set of random individuals that represent a potential solution to the problem. The flowchart for the canonical DE is shown in Figure 2.4. Let x_i^g represents the i^{th} individual in generation g. To introduce new individuals in the population, DE applies the differential mutation, crossover, and selection operators.

2.3.1 Differential Mutation

The differential mutation operator creates a mutant vector v_i^g by first taking the difference between two donor vectors, chosen randomly from the population. Then, the difference is scaled and added to a third donor vector (i.e., base vector) to obtain the final mutant vector. The equation for the mutant vector is defined as:

$$v_i^g = x_{r1}^g + F \times (x_{r2}^g - x_{r3}^g)$$

where, $r1, r2, and\ r3$ represents the population indices of the three donor vectors and are mutually exclusive (i.e., $r1 \neq r2 \neq r3$), and $F \in (0, \infty)$ represents the differential weight (i.e., a scaling factor).

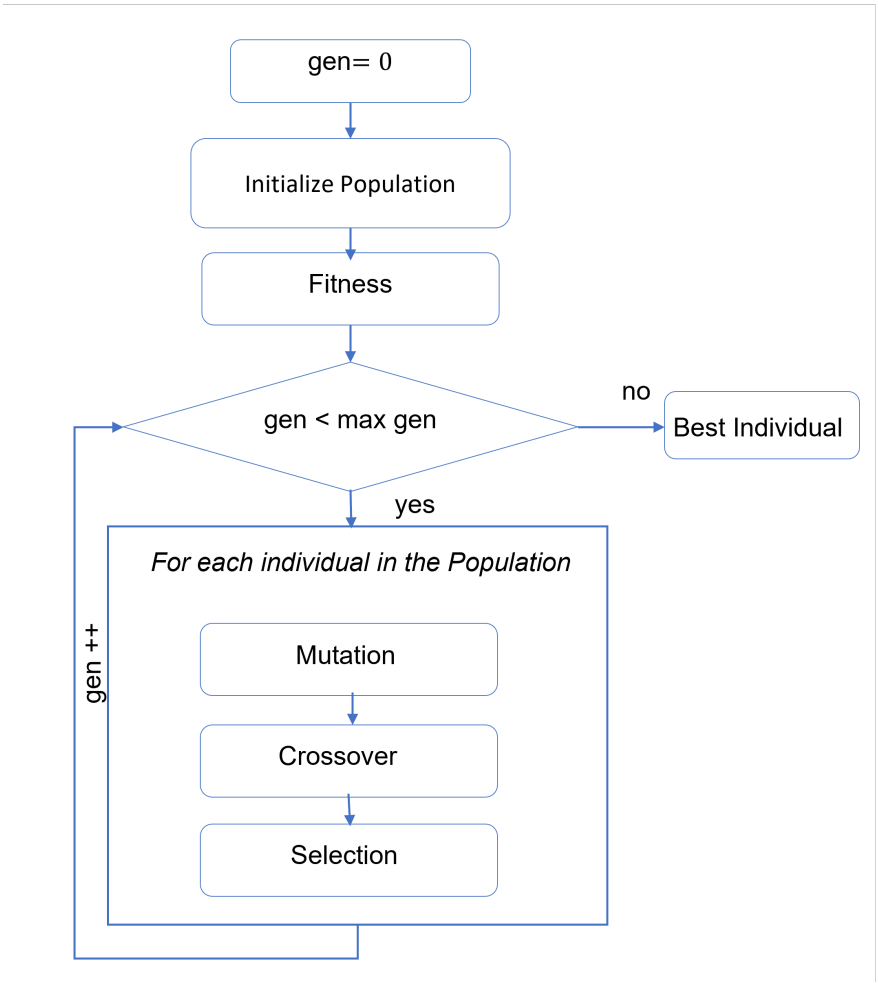


Figure 2.4: Differential Evolution Flowchart

2.3.2 Crossover

After the mutation step, a crossover between the current individual (target vector) x_i^g and the mutant vector v_i^g takes place to create a trial vector u_i^g. There are mainly two crossover variants for the standard DE: binomial crossover and the exponential crossover [47, 61]. The binomial crossover operator is defined as follows:

$$u_{i,j}^g = \begin{cases} v_{i,j}^g, & \text{if } rand(0,1) \leq CR \text{ or } j = j_{rand} \\ x_{i,j}^g, & \text{otherwise} \end{cases}$$

where CR represents the crossover rate and j_{rand} represents a random value in the vector dimension $[1, D]$. The trial vector $u_{i,g}$ takes the j^{th} real-value of either the target vector or mutant vector based on the condition.

2.3.3 Selection

Finally, a selection procedure selects the better individual between the trial vector and the current individual based on their fitness values for the next generation as follows:

$$x_i^{g+1} = \begin{cases} u_i^g, & \text{if } f(u_i^g) \leq f(x_i^g) \\ x_i^g, & \text{otherwise} \end{cases}$$

Thus, if the trial vector has better fitness, it replaces the current individual; otherwise, the current individual survives to the next generation. The process of mutation, crossover, and selection continues for each individual in the population for each generation until the termination condition is satisfied, and the global best individual is returned as the final solution.

2.3.4 DE for COPs

Simple DE works best for numerical optimization problems with real-valued chromosome representations. However, for permutation-based representations, the arithmetic operations to create the new mutant vector are not meaningful. Therefore, many differential mutation operators have been proposed in the literature to deal with permutation-based problems for DE. Application areas where DE for permutation-based COPs showed competitive results include scheduling problems (e.g., Flow shop, Job shop, etc.), TSP, Knapsack, Vehicle Routine Problem, and many others [14]. Some proposed differential mutation operators involve the Permutation Matrix, Adjacency Matrix, Relative Position Indexing, and Forward-Backward Transformation approaches [54]. For the grain mixing

problem, we adapt two different techniques for DE mutation. The first is adapted from the Relative Position Indexing (RPI) operator, and the second is based on perturbing the generation best individual chromosome with random individual chromosomes in the population.

2.3.4.1 Relative Postition Indexing (RPI) The RPI approach was adapted from the original differential mutation and is only applicable for permutation-based problems [46]. It has been applied successfully in COPs like permutation flow-shop scheduling [57, 62]. RPI first transforms the integer permutation vectors into floating-point vectors and then applies the standard DE mutation on the floating-point vectors. The mutated floating-point vector is then transformed back into an integer permutation vector using the relative position indexing.

For example, let us consider three random donor vectors in the current population, x_{r1}, x_{r2}, and x_{r3} with the following representation where each vector represents a legal TSP tour.

$$x^g_{r1} = \begin{bmatrix} 1 & 5 & 3 & 6 & 7 & 4 & 2 & 8 \end{bmatrix}$$
$$x^g_{r2} = \begin{bmatrix} 6 & 2 & 5 & 1 & 3 & 7 & 4 & 8 \end{bmatrix}$$
$$x^g_{r3} = \begin{bmatrix} 8 & 2 & 3 & 6 & 7 & 5 & 1 & 4 \end{bmatrix}$$

To convert these integer vectors into floating point vectors with each entry in the half interval $(0, 1]$, one approach is to divide each integer value with the highest value in the vector. After

conversion, the floating-point vectors become:

$$x(f)^g_{r1} = \begin{bmatrix} 0.125 & 0.625 & 0.375 & 0.750 & 0.875 & 0.500 & 0.250 & 1.000 \end{bmatrix}$$
$$x(f)^g_{r2} = \begin{bmatrix} 0.750 & 0.250 & 0.625 & 0.125 & 0.375 & 0.875 & 0.500 & 1.000 \end{bmatrix}$$
$$x(f)^g_{r3} = \begin{bmatrix} 1.000 & 0.250 & 0.375 & 0.750 & 0.875 & 0.625 & 0.125 & 0.500 \end{bmatrix}$$

Then, the standard DE mutation operator can be applied to the resulting vectors to obtain the floating mutation vector. With $F = 0.4$, the floating mutant vector would be:

$$v(f)^g_i = x(f)^g_{r1} + F \times (x(f)^g_{r2} - x(f)^g_{r3})$$
$$v(f)^g_i = \begin{bmatrix} 0.025 & 0.625 & 0.475 & 0.500 & 0.675 & 0.600 & 0.400 & 1.200 \end{bmatrix}$$

The final step is to convert the floating mutant vector back to the integer permutation representation. To do that, the RPI is used, where the smallest floating-point value is replaced by the smallest integer value and the next smallest floating-point by the next smallest integer value and so on until all the floating points are converted. After conversion, the mutant vector becomes:

$$v^g_i = \begin{bmatrix} 1 & 6 & 3 & 4 & 7 & 5 & 2 & 8 \end{bmatrix}$$

The above mutant vector transformation is a legal tour; however, it may not always be the case if two or more floating-point values happen to be the same. In that case, the trial vector is repaired or discarded for that individual in the current generation.

2.3.4.2 Generation Best Perturbation (GBP) The second differential mutation operator that we used involves perturbing the chromosome of the generation best individual. Similar approaches have been proposed in the literature for solving application-specific problems

using discrete DE [31, 71]. For the permutation flow-shop problem, Tasgetiren *et al.* [72] and Pan *et al.* [56] introduced a novel discrete DE algorithm where they represented the solution based on discrete job permutation. In their approach, the target individual $x_i = \{\pi_1, \pi_2, ..., \pi_n\}$ in the population is represented by the discrete permutation of the jobs, and, they perturbed the genes of the previous generation's best individual to obtain the mutant vector for the current generation. Therefore, the differential mutation, in this case, is achieved in the form of perturbations. Since the perturbation procedure is stochastic, it is expected that the new mutant individual is going to be distinct from the other individuals in the population. Their proposed mutation vector creation can be expressed as follows:

$$v_i^g = \begin{cases} F_k(x_{best}^{g-1}) & \text{if } r < P_m \\ Insert(x_{best}^{g-1}) & \text{otherwise} \end{cases}$$

where x_{best}^{g-1} specifies the best individual from the previous generation and F_k is the perturbation procedure with a perturbation strength of k. The perturbation function swaps the genes of the best individual based on the value of k. If k is too high, there will be excessive randomness in the algorithm and if k is too low, the algorithm might get stuck to local minima. P_m represents the perturbation/mutation probability. $Insert()$ is the procedure for inserting random genes in the mutant vector. For a uniform random number $r \in [0, 1]$, if r is less than P_m then the mutant vector is simply the perturbation of the previous best individual. Otherwise, the mutant vector is the perturbation of the best individual with random insertion moves.

For this study, we formulate the grain mixing problem as a permutation-based combinatorial optimization problem. In this chapter, we presented a brief overview of the COPs along with some of the existing approaches for solving COPs. We also presented a brief overview of the Genetic Algorithm and Differential Evolution and how they were used

in the literature for solving permutation-based COPs. We adapted these two algorithms for solving the grain mixing problem,

CHAPTER THREE

THE GRAIN MIXING PROBLEM

In this chapter, we introduce the grain mixing problem with a simple example showing how mixing grain can be beneficial to improve the overall profitability of the farmers when selling wheat to local grain elevators. Then we provide a formal mathematical formulation for the grain mixing problem studied in this thesis, followed by a theorem to show that the grain mixing problem is NP-Hard. Finally, we provide descriptions of the datasets used for the study followed by a review of some existing approaches that were used to solve similar optimization problems in the literature.

3.1 Grain Mixing Example

One of the goals of this study is to develop methods for farmers to maximize their overall wheat distribution profit when selling wheat at multiple local grain elevators. In Montana, when selling wheat, the price per bushel of grain varies based on a range of protein levels. Each elevator has a base protein range for which a base price is paid and follows a premium-dockage curve where the price/bushel is increased for a higher protein level and decreased for a lower protein level. For example, if the protein level is $[10, 11)\%$, the price might be \$3 per bushel where it might increase to \$4/bu with protein levels of $[12, 13)\%$. The payment schedule can be viewed as a step function of the protein level with prices per bushel (which we discuss further in dataset description in Section 3.4). There are many cases where the protein level of a bin is short of reaching a higher price range or maybe above the minimum protein requirement having no added benefit in terms of price. Therefore, it might be possible to mix grain from a high-quality bin with a low-quality bin to change the average protein level where the low-quality bin reaches the next step in the price range and the high-quality

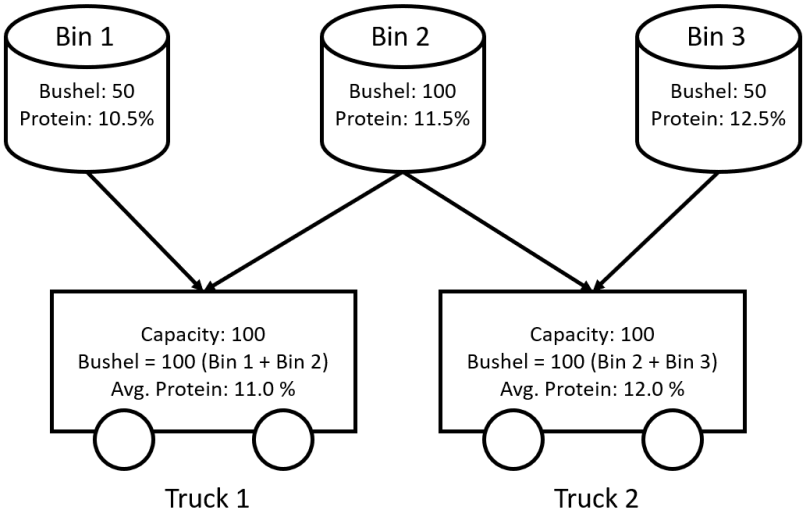


Figure 3.1: A simple grain mixing example

Table 3.1: Elevator price for grain mixing example

Protein range (%)	**Price/Bushel ($)**
$[10, 11)$	3
$[11, 12)$	4
$[12, 13)$	6

bin remains in the same step in the price range making a better overall profit. The profit of a truck depends on the number of bushels and the average protein content of the grain in that truck.

Figure 3.1 illustrates an example of how mixing grain could be useful for making a better overall profit. In the example, there are three bins with different numbers of bushels and protein levels. Table 3.1 represents elevator prices for different quality grains. Many small farmers who do not track the protein level of their wheat would load each truck with grain from a single bin and sell it to the elevators. A truck has a maximum bushel capacity it can carry which is 100 bushels in our example. The price they will get without

any grain mixing by loading three trucks separately with grain from each bin would be $(50 \times \$3 + 100 \times \$4 + 50 \times \$6) = \$850$. However, if they were to track the protein content and mix grains as shown in the figure; load truck one with 50 bushels from bin one and 50 bushels from bin two, and load truck two with 50 bushels from bin two and 50 bushels from bin three, the price they will get would be $(100 \times \$4 + 100 \times \$6) = \$1000$. Therefore, mixing grain in this scenario increases the profit by \$150.

A key challenge in the grain mixing problem is to find the optimal bin-pair combination and bushels drawn from each bin to load trucks that will yield maximum profit. For example, in Figure 3.1, there are also other ways to mix the grain. If we load truck one by mixing 50 bushels from bin one and 50 bushels from bin three (average protein level 11.5), and truck two by taking 100 bushels from bin two without any mixing, the total price would be $(100 \times \$4 + 100 \times \$4) = \$800$ which yields a profit less than taking all the grains separately.

Although protein level in a truck plays a significant role in determining the profit from wheat distribution, we also need to consider other factors such as mixing cost and delivery cost in the profit model. When mixing grain from multiple bins to load a truck, there is an associated mixing cost that depends on the mixing difficulty level. The farmers divide their farms into multiple sites for better farm management (also known as site-specific farming [75]) and store grain in site-specific locations. The mixing difficulty depends on each bin site. For example, mixing grain from bins that are on the same site has a lower difficulty level than mixing bins that are on different sites. A difficulty level of 0 indicates no mixing and the mixing cost is also 0. However, a difficulty level of 4 indicates that the bins from which we are mixing are from different sites and farther away from each other. As an example, the mixing cost might be \$0.1 per bushel in this case. Figure 3.2 depicts a notional mixing difficulty scenario based on the site location and the Table 3.2 shows the associated mixing cost for different mixing difficulties.

The delivery cost for a truck transporting to the nearest elevators depends on the

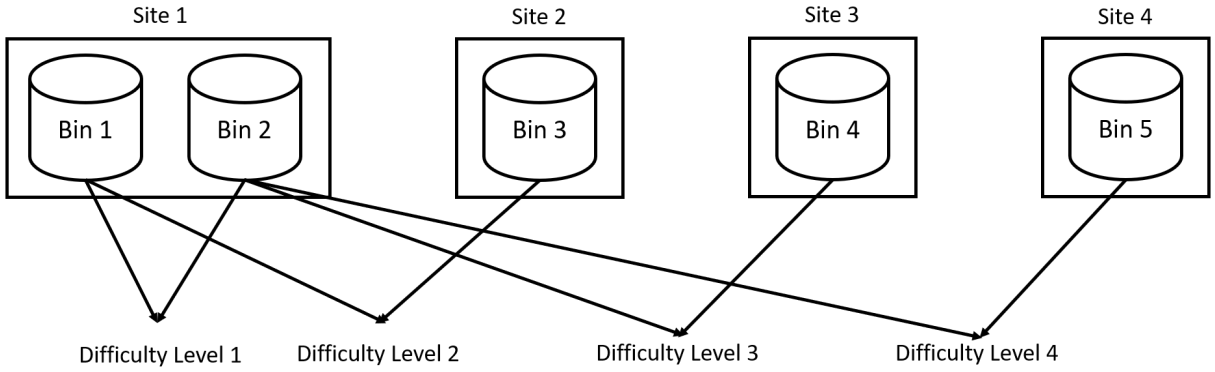


Figure 3.2: Mixing difficulty based on farm site

Table 3.2: Mixing cost based on difficulty level

Difficulty Level	**Description**	**Cost/Bushel ($)**
0	No mixing	0.000
1	Easy	0.001
2	Moderate	0.010
3	Difficult	0.050
4	Very difficult	0.100

distance between the site location and the elevators. For example, a truck moving from site one to elevator one might have a delivery cost of $0.2 per bushel whereas moving to the same elevator from site three might have a cost of $0.18 per bushel. Moreover, the delivery cost varies based on the distance between a specific site and multiple elevators.

3.2 Mathematical Model Formulation

The grain mixing problem aims to determine the optimal mix of wheat from storage bins to maximize profit when sold to a set of elevators. In this section, we present the

Table 3.3: Notation used in Grain Mixing Problem

B	Set of bins.
E	Set of elevators
b_j^{bu}	$b_j \in B$. Bushel content in bin b_j.
b_j^{pr}	$b_j \in B$. Protein content in bin b_j.
C_T	Constant Capacity of each truck.
G_e	Set of grades for elevator $e \in E$.
$g_{e,l}^{p}$	$g_{e,l} \in G_e$. Base Price/Bushel for grade $g_{e,l}$.
$g_{e,l}^{Min\_pr}$	$g_{e,l} \in G_e$. Minimum Protein requirement of grade $g_{e,l}$.
$g_{e,l}^{Max\_pr}$	$g_{e,l} \in G_e$. Maximum Protein requirement of grade $g_{e,l}$.
m_{b_j,b_k}	Mixing cost for mixing bins $b_j, b_k \in B$.
$d_{b_k,e}$	Delivery cost for transporting grain from b_k to elevator e.

Table 3.4: Decision variables in the Grain Mixing Problem

T	Set of trucks for transporting all the grain
$t_{i,j,k,e,l}$	Bushels drawn from b_j and b_k to fill t_i achieving grade $g_{e,l}$.
t_i^{pr}	Protein content of truck t_i.

mathematical specifications for the grain mixing problem studied in this thesis. The input instances of the problem are described using the notation in Table 3.3 with the decision variables represented as shown in Table 3.4.

The objective of the Grain Mixing Problem is to find a sequence of trucks that maximizes the overall profit given by:

$$\max \sum_{t_i \in T} \sum_{b_j, b_k \in B} \sum_{e \in E} \sum_{g_{e,l} \in G_e} ((g_{e,l}^{p} - (m_{b_j,b_k} + d_{b_k,e})) \times t_{i,j,k,e,l}) \tag{3.1}$$

subject to:

$$t_{i,j,k} \leq C_T \qquad \forall t_i \in T \quad \forall b_j, b_k \in B \tag{3.2}$$

$$t_{i,j,k} \geq 0 \qquad \forall t_i \in T \quad \forall b_j, b_k \in B \tag{3.3}$$

$$\sum_{t_i \in T} t_{i,j} \leq b_j^{bu} \qquad \forall b_j \in B \tag{3.4}$$

$$\sum_{t_i \in T} t_{i,j,k} \leq \sum_{b_j \in B} b_j^{bu} \tag{3.5}$$

$$t_i^{pr} \times t_{i,j,k} = b_j^{pr} \times t_{i,j} + b_k^{pr} \times t_{i,k} \qquad \forall t_i \in T \tag{3.6}$$

$$t_i^{pr} \times t_{i,j,k,e,l} < g_{e,l}^{Max\_pr} \times t_{i,j,k} \qquad \forall t_i \in T \quad \forall g_{e,l} \in G_e \tag{3.7}$$

$$t_i^{pr} \times t_{i,j,k,e,l} \geq g_l^{Min\_pr} \times t_{i,j,k} \qquad \forall t_i \in T \quad \forall g_{e,l} \in G_e \tag{3.8}$$

Note that the objective function given in Equation 3.1 reflects the nonlinearity of the premium-dockage curve with the inflection point given by the base price/bushel $g_{e,l}^p$. Equations 3.2 and 3.3 represent the linear truck capacity constraints. Bushels loaded from two bins in a truck cannot exceed the maximum capacity of that truck. Equations 3.4 and 3.5 represent linear bin content constraints. Bushels drawn from a single bin to load multiple trucks cannot exceed the initial content (bushel amount) of that bin and the sum of all bushels loaded into multiple trucks cannot exceed the total bushels of all bins. Equation 3.6 represents the weighted average protein level of a truck after mixing grain from two bins. Finally, Equations 3.7 and 3.8 are the grade protein requirements that assign a grade (price) and elevator to the truck based on elevator protein requirements and truck protein content. The last three constraints contain a product of two decision variables which introduces nonlinearity in these equations.

3.3 Grain Mixing Complexity

The formal mathematical specifications for the grain mixing problem suggest that due to the nonlinear objective function and nonlinearity in some of the constraints, it is hard to solve the problem exactly using Linear Programming or Mixed-Integer Linear Programming methods. We present and prove a theorem about the computational complexity of the grain mixing problem following a reduction from the planar 3-Dimensional Matching (3DM) problem. For the NP-Hardness theorem, we assume that each elevator has a fixed amount of bushels that they will accept. Although this assumption is not present in the general grain mixing problem, the proof provides an idea of the difficulty of the grain mixing problem studied here.

3.3.1 Planar 3-Dimensional Matching (3DM)

Planar 3DM is a more restricted version of the "standard" 3DM problem that is known to be NP-Complete [18]. In graph theory, a 3DM problem is the generalization of the bipartite graph to 3-partite hypergraphs. The standard 3DM problem is known to be NP-Complete as shown in one of Karp's NP-Complete problem lists [39] and defined as follows:

Definition 3.1. *Given* $\mathbf{T} \subset \mathbf{X} \times \mathbf{Y} \times \mathbf{Z}$*, where* $\mathbf{X}, \mathbf{Y}$*, and* $\mathbf{Z}$ *are finite sets and* $|\mathbf{X}| = |\mathbf{Y}| = |\mathbf{Z}| = \alpha$*. There is a matching set* $\mathbf{M} \subseteq \mathbf{T}$ *such that* $|\mathbf{M}| = \alpha$ *and no two elements of* $\mathbf{M}$ *agree in any coordinate [39].*

Figure 3.3 shows an example of the 3DM problem. The subfigure 3.3 (a) shows all the triples in set $\mathbf{T}$, subfigure 3.3 (b), 3.3 (c) shows two different matchings $\mathbf{M}$ where $|\mathbf{M}| = 2$. In this example, $|\mathbf{M}| = 2$ is the maximum 3DM as it is not possible to get a valid matching with size greater than 2.

A bipartite graph can be associated with the standard 3DM instance where one side of the graph consists of vertices for each element in the set $\mathbf{X}, \mathbf{Y}$, and $\mathbf{Z}$ and another side

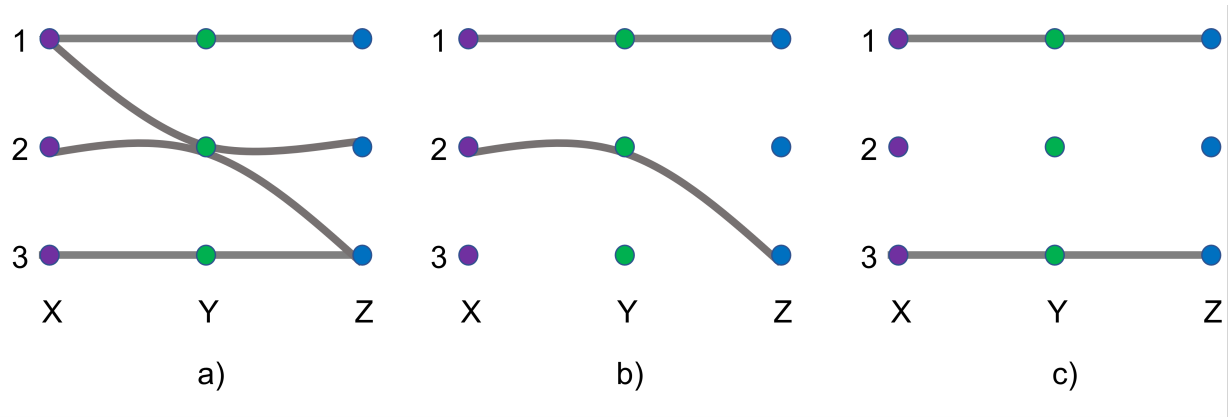


Figure 3.3: 3-Dimensional Matching example

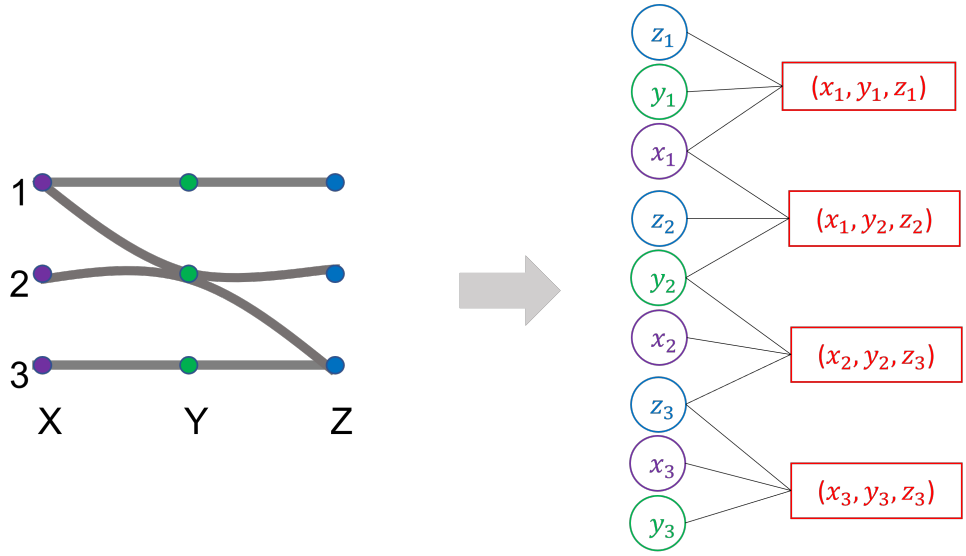


Figure 3.4: Planar 3-Dimensional Matching example

consists of vertices for each triple in **T**. There is an edge between the vertices from the sets to the vertices of the triples if and only if the set element is in the triple. If the associated bipartite graph is planar then it is referred to as planar 3DM [18]. Figure 3.4 shows an example of the planar 3DM instance from a standard 3DM instance.

The decision version of the (planar) 3DM instance is also known to be NP-Complete [39] and can be defined as *"Given a subset of triples* **T** *and an integer* α*, does there exist a*

3-*dimensional matching* $\mathbf{M} \subseteq \mathbf{T}$ *where* $|\mathbf{M}| \geq \alpha$*"*. Consequently, the optimization problem for the 3DM can be defined as *"Given a subset of triples* $\mathbf{T}$*, find the* 3-*dimensional matching* $\mathbf{M} \subseteq \mathbf{T}$ *that maximizes* $|\mathbf{M}|$*"*. As the decision problem is NP-Complete, it follows that the optimization problem will be NP-Hard [23].

3.3.2 NP-Hardness Theorem

Theorem 3.1. *The grain mixing problem is NP-Hard.*

Proof. First, given an instance of a planar 3DM problem, we convert it to an instance of a GM problem as follows:

- For each element in set $\mathbf{X}$, and set $\mathbf{Y}$:
  - Create Bin b_{x_i} and Bin b_{y_z} and assign half unit of bushels,
  - Set protein content for b_{x_i} to $(p-\epsilon)\%$ and for b_{y_z} to $(p+\epsilon)\%$ where p represents a constant base protein content and ϵ is a small positive integer,
  - Set initial mixing cost to $+\infty$ for all of the created bins.
- For each element in set $\mathbf{Z}$, create an elevator m_{z_k} and define its cost function to pay \$0 initially. Set the capacity of the elevators to accept a maximum of one unit bushel.
- For each triple $(x_i, y_j, z_k) \in \mathbf{T}$, create a truck t_i that loads grain from b_{x_i} and b_{y_z} and deliver to elevator m_{z_k}. The maximum truck capacity is also one unit bushel. Then assign the following:
  - change the mixing cost between b_{x_i} and b_{y_z} to a constant positive value C,
  - Set the delivery cost of truck t_i to C for delivering to elevator m_{z_k},
  - Update elevator m_{z_k} cost function to pay $\$R$ per unit bushel where R represents a constant positive value if the protein content of the truck is in $[p, p+2\epsilon)$.

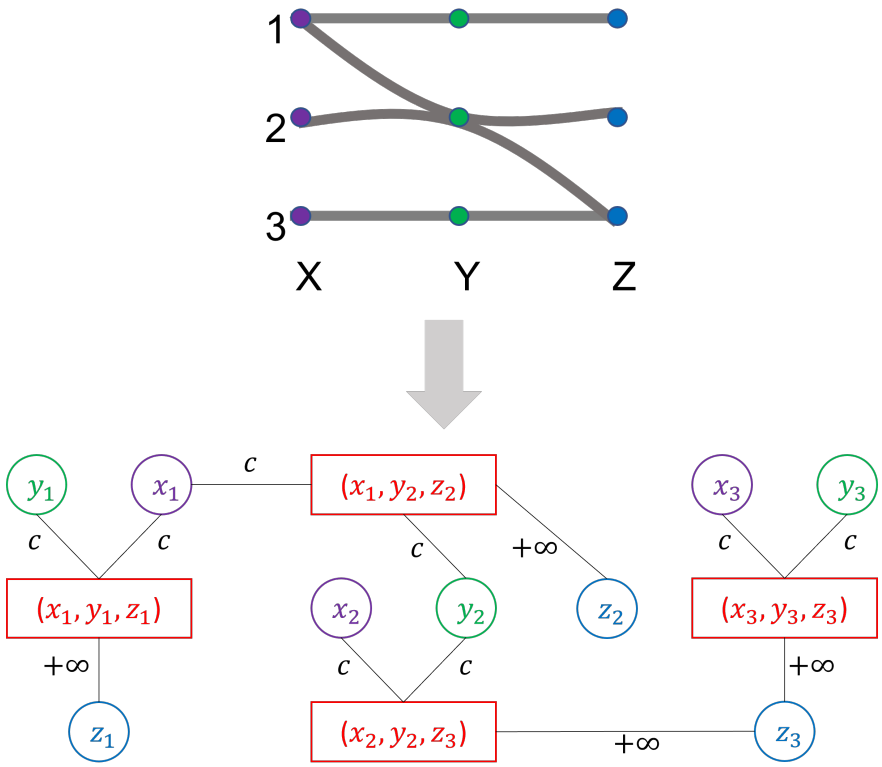


Figure 3.5: Reduction from planar 3DM to GM instance

Figure 3.5 shows the reduction from the planar 3DM to the GM instance. In the figure, the triples represented in the red box could be considered as an intermediate vertex that is connected to the elevator by an edge with cost $+\infty$ that is used to prevent flowing bushels from one elevator to another (as it is not possible for the GM scenario). The triples vertex with the associated set vertices can be thought of as a truck starting at b_{x_i} location and going to b_{y_j} location for mixing grain incurring mixing cost C and from there going to the elevator with a delivery cost of C. Although the mixing and delivery cost varies based on bins and elevators location, a constant value of C is used for both in the proof for simplicity. These costs are represented with the edge distance. We then prove the following claim: *The planar 3DM problem has a matching of size α if and only if the GM instance has a maximum revenue of $\$R \times \alpha$ and a minimum cost of $2C \times \alpha$.*

If Part: *If the planar 3DM has a matching of size α then the GM instance has a maximum revenue of $\$R\alpha$ and a minimum cost of $2C\alpha$.*

For each matching in triples $(x_i, y_j, z_k) \in \alpha$, we load trucks with half unit bushels from bin b_{x_i}, and half unit from bin b_{y_j} and send to elevator m_{z_k}. The elevator will pay $\$R$ as the protein range of the truck would be $[p, p + 2\epsilon)$ and the total mixing cost and delivery cost would be $2C$. Therefore, for α matching, the total revenue would be $\$R\alpha$ and the total cost would be $2C\alpha$.

Only If Part: *If the GM instance has a maximum revenue of $\$R\alpha$ and a minimum cost of $2C\alpha$ then the planar 3DM has a matching of size α.*

First, we argue that no two bins from the same set $\mathbf{X}$ or $\mathbf{Y}$ will be in the valid GM solution as the mixing cost between the same set of bins is set to $+\infty$. Therefore, there will be no profit from mixing from the same set of bins. Then, the same bin will not appear twice in the valid GM solution as it will either increase the overall mixing cost or decrease the overall revenue. For example, each bin has a capacity to hold half unit bushels and if a bin is delivered to multiple elevators the truck will be partially filled. For a partially filled truck the revenue would be $< \$R$ and there will be multiple mixing costs for the same bin which would be $> C$. Therefore, using the same bin to deliver grain to multiple elevators would either decrease the overall revenue or increase the overall cost.

Finally, the elevator can accept only one unit of bushels. Therefore, if it accepts a fully loaded truck (truck capacity is also one unit bushels) then it will not accept bushels from any other truck. However, two partially loaded trucks may deliver to the same elevator which will violate it to be a valid matching. We argue that partially filled trucks would not be in the valid GM solution as in that case it will either increase the overall delivery cost or decrease the overall revenue. For example, if a partially loaded truck delivers to an elevator, the elevator will pay $< \$R$ amount and in that case, another partially filled truck has to deliver to the same elevator to get an overall revenue of $\$R$. However, in this case,

multiple trucks have to go to the same elevator which will increase the delivery cost to be $> C$. Therefore, each truck in the valid GM solution will have a unique bin pair and an elevator providing a planar 3DM of size α.

The GM instance is also in NP as given a certificate consisting of triples $(b_{x_i}, b_{y_j}, m_{z_k})$ it is possible to verify if the revenue is $R\alpha$ and the cost is $2C\alpha$ in polynomial time. Therefore, the decision version of the GM problem is NP-Complete which suggests that the optimization problem will be NP-Hard. □

3.4 Grain Mixing Datasets

3.4.1 Real Dataset

The grain mixing dataset used in this project was collected from a local Montana farmer who tracks the protein level of his wheat. The dataset contains the data for wheat harvested in 2016 and 2017. All bushels of wheat were distributed among various bins. For our problem, a bin entry includes the bin id, the site number where the wheat was harvested, the average protein level of wheat in that bin, and the total number of bushels stored in that bin. The wheat data we received contains a total of 16 bins with different number of bushels and protein content in each bin. The wheat distribution statistics for both years is shown in Table 3.5, and the details on each bin for the year 2017 are shown in Table 3.6.

The data provided also include a list of elevators with their associated market prices for buying the wheat. An entry in the elevator list contains a premium-dockage curve with the base protein level, the price for the base protein level, the premium payment added to the base price for higher protein level (upPrice), and the dockage payment deducted from the base price for lower protein level (downPrice). We were provided information on three elevators for both years. The market prices for the three elevators are shown in Table 3.7.

Figure 3.6 shows the premium-dockage curves for the three elevators in 2017. The dots in each elevator's curve shows the base protein level and base price/bushel for that elevator.

Table 3.5: Wheat distribution statistics

Year	**Tot Bush**	**Max Bu/Bin**	**Min Bu/Bin**	**Avg Bu/Bin**
2016	114284.8	14836.8	1712.7	7142.8
2017	112417.5	14836.8	2985.3	7026.1
Year	**Tot Prot**	**Max Prot/Bin**	**Min Prot/Bin**	**Avg Prot/Bin**
2016	191.3	13.3	10.1	12.0
2017	190.1	13.8	10.1	11.9

As shown, the price of protein increases (or decreases) as a step function, and the step size differs based on the upProtein (or downProtein) levels from the base protein level, thus creating an inflection point at the base price. The non-linear cost function for the elevators can be expressed as

$$t_{i,j}^{Rev} = \left[e_j^{b\_price} + \left(\left\lfloor \frac{|t_i^{Pr} - e_j^{b\_Pr}|}{e_j^{up/down\_Pr}} \right\rfloor \right) \times e_j^{up/down\_price} \right] \times t_i^{Bu}$$

where, $t_{i,j}^{Rev}$ is the revenue of truck t_i going to the elevator e_j with a protein content of t_i^{Pr} and a total bushels of t_i^{Bu}. $e_j^{b\_price}$ is the base protein price of the elevator, $e_j^{b\_Pr}$ is the base protein level of the elevator, $e_j^{up/down\_Pr}$ is the up or down protein level, and $e_j^{up/down\_price}$ is the up or down price.

Multiple trucks, each with a fixed capacity of 8,000 bushels, were used to carry the wheat to the elevators. Note that there is a delivery cost involved in going to the elevators, ranging from \$960 to \$2,000 for a fully loaded truck, depending on the distance between a bin site and an elevator. Moreover, mixing the wheat from multiple bins to change the average protein content incurs a mixing cost. The mixing is restricted to two bins at a time due to farmers' physical limitations. The mixing cost also depends on the site number of the bins; mixing two bins from the same site is less expensive than mixing bins from different

Table 3.6: Farmer bin information for wheat harvested in 2017

Bin number	**Site**	**Avg Protein**	**Bushels**
1	2	12.32	14836.8
2	7	13.78	7292.5
3	2	12.71	6395.3
4	6	11.34	8525.5
5	6	10.88	5713.36
6	1	12.58	7703.67
7	7	10.35	1712.67
8	1	10.72	4192.67
9	1	13.15	6539.83
10	3	10.11	7292.5
11	3	12.38	4921.32
12	7	12.01	7089.67
13	6	10.83	5050
14	3	12.13	8657.4
15	5	13.41	2985.3
16	2	11.4	13509

sites due to the lower difficulty. Thus, for a fully loaded truck, the mixing cost varies from \$8 to \$800.

3.4.2 Artificial Dataset

For this study, we were able to collect real wheat harvest information only for the years 2016 and 2017 from a Northwest Montana farmer. To further evaluate the performance of the

Table 3.7: Market prices for three elevators in Northwest Montana

Year	Elevator	Base Price	BaseProtein	upPrice	upProtein	downPrice	downProtein
2016	1	3.32	11.25	0.03	0.75	-0.06	0.75
	2	3.84	11.75	0.25	0.50	-0.30	0.50
	3	3.54	12.00	0.50	0.60	-0.40	0.30
2017	1	4.42	11.50	0.05	0.50	-0.10	0.50
	2	4.47	12.00	0.25	0.50	-0.30	0.50
	3	4.39	12.20	0.50	0.60	-0.40	0.30

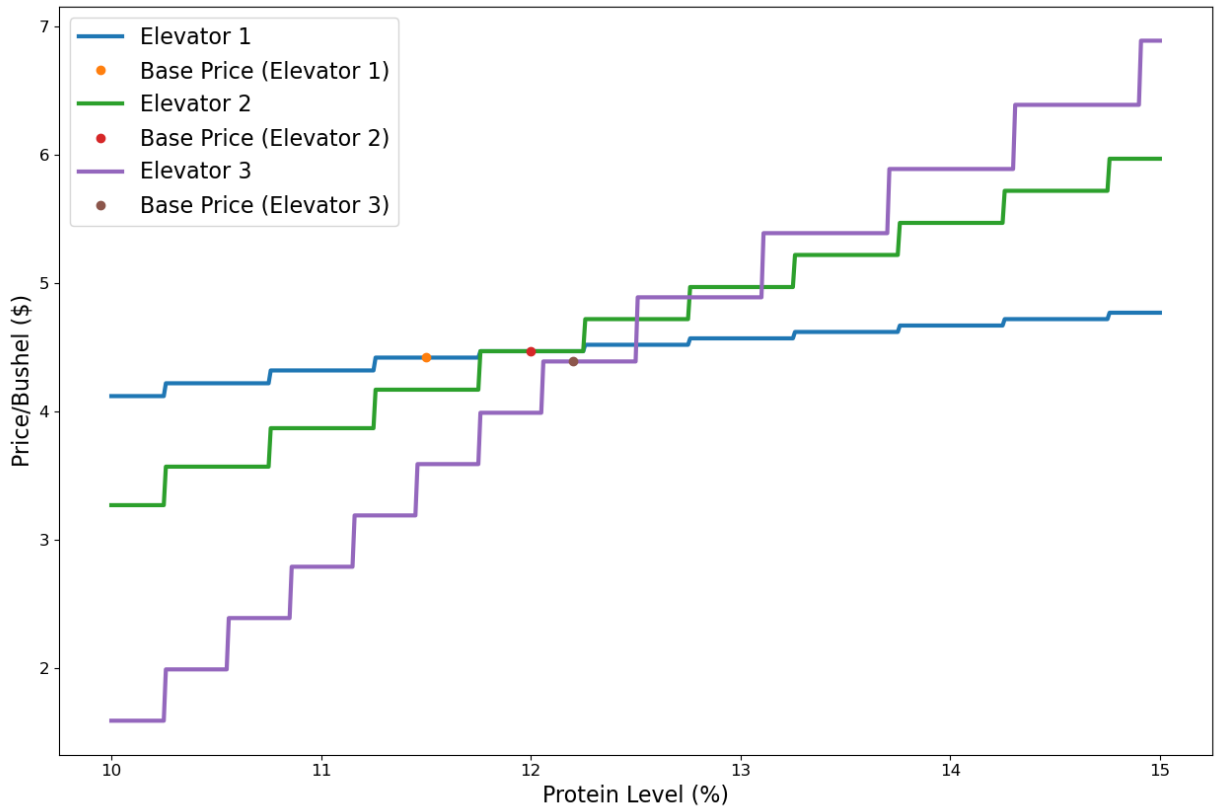


Figure 3.6: Premium-dockage curve of three elevators in Northwest Montana in 2017

mixing algorithms, we created twelve additional datasets. Among them, two datasets were created by swapping the elevator prices for both years. In other words, the bin information for

the year 2017 was used with the elevator prices from 2016 and vice-versa. These two datasets help us assess the effectiveness of the different mixing algorithms for different elevator prices.

Moreover, we created five artificial datasets for each year. For these datasets, the elevators, mixing cost, and delivery cost remained the same, however, the bin contents were altered. Looking at the real dataset for the years 2016 and 2017, we noticed that the number of bins, mixing, and delivery costs remained the same over the years. However, the number of bushels in each bin and the protein content of the bins were different. Therefore, we used the same 16 bins, and the number of bushels and the protein content in each bin was assigned randomly falling within a pre-specified lower and upper bound. For bushel content, the lower and upper bound was set between 2000 to 16000, and the protein content range was set between 10% to 14% respectively. These artificial datasets provide a rough idea of how the different mixing algorithms perform with different years' bin information.

3.5 Existing Approaches

The grain mixing problem studied in this paper relates to the classical blending optimization problem [13]. In general, the blending problem involves mixing two or more collections of grain with different characteristics to produce a single (or multiple) final product(s) that either lowers the production cost or improves the profit (or both). Next, we discuss some existing approaches from the literature that were used to solve related blending optimization problems.

3.5.1 Linear Programming

Linear Programming (LP) is an optimization tool for solving a continuous-space optimization problem. It is possible to solve the classical blending problems by LP methods if both the objective function and the constraints are linear [38]. There exist a few works that attempted to solve the decision version of the grain mixing problem (also known as wheat

blending) with linear programming. Hayka and Cakmalki utilized LP methods capable of predicting the optimal wheat blend ratio for a targeted final quality to produce a bread making flour [33]. Haas used the simplex algorithm to find the optimum blend that satisfies the customer's specific solvent retention capacities (SRC) [32]. LP has also been used for other blending optimization problems. Yoon *et al.* proposed the least cost LP model in blending various components of Surimi Seafood [79]. Eraslan *et al.* utilized LP to model the coal blending problem in a power plant that was able to determine the optimal coal blends [20]. Farghaly *et al.* used LP rules for blending origin mines and metallurgical units that minimize the total cost of production and distribution [22]. In many real-world applications, the LP formulation is often not suitable to address specific blending needs. In our case, there is no specific targeted wheat quality (each truck's protein content is determined at runtime), which makes it challenging to solve the problem by only using LP.

3.5.2 Mixed-integer linear programming

Mixed-integer linear programming (MILP) is often used to solve many real-world blending problems [59]. MILP is a variation of linear programming where some decision variables are restricted to integer values. There are exact and approximation methods to solve IP/MILP problems; however, MILP is known to be NP-hard [41]. Bilgen and Ozkarahan proposed an MILP model to optimize the cost for the wheat supply chain (blending, loading, transportation, and storage), where the model used a specific blending formula for mixing [6]. Thakur *et al.* proposed a multi-objective optimization problem for minimizing the food safety risk caused by the grain elevators' lot aggregation and cost of blending grain [73]. They utilized a mixed-integer programming model for the two objective functions and computed the Pareto optimal front for the simultaneous optimization of the two objective functions. Gholamian and Taghanzadeh proposed an integrated design of the wheat supply chain in Iran that considers different sectors of the chains including blending wheat, locating silos,

different modes of transportation, and export sectors [24]. In our problem, the truck protein content and the grade protein content constraints (product of two decision variables) makes it challenging to be used exactly in an MILP solver. Furthermore, the premium-dockage curve used by the elevator is a nonlinear function based on a baseline protein level. Although it is possible to relax the problem for MILP, we left this as future work.

MILP has also been used in other blending optimization problems. Moro and Pinto utilized an MIP model for short-term crude oil scheduling where the objective is to maintain a target crude oil load that minimizes the cost and is subject to nonlinear constraints [51]. Randall *et al.* proposed a water supply planning simulation model that uses MILP for finding a long-term plan by blending water from different sources to meet water quality standards [64]. Jia and Ierapetritou addressed the problem of gasoline blending and distribution, and used a MILP formulation for finding the optimal operation of gasoline blending, transfer to stock tanks, and the delivery schedule for satisfying orders [37]. Ashayeri *et al.* applied MILP to a blending problem in the process industry that minimizes the overall cost of manufacturing chemical fertilizers [3].

3.5.3 Meta-heuristic approaches

For complex blending problems, meta-heuristic approaches such as evolutionary and swarm-based algorithms have also been used in the literature. Li *et al.* proposed a hybrid evolutionary method to solve the wheat blending problem in Australia [45]. Their method used a GA with a heuristic method and a linear-relaxed version of the simplex algorithm to solve the blending problem. Their work closely relates to our mixing problem; however, the problem formulation differs based on the US wheat market. In our problem, there is an added constraint on the capacity a truck can carry, and the farmers have the choice of delivering wheat to multiple elevators depending on the price. Farahani *et al.* addressed the optimization problem of wheat storage and transportation system [21]. Due to variation

in climate, wheat production is not consistent around the year. The difference in wheat production and consumption rate necessitates the storage and transportation of wheat. They utilized a genetic algorithm and MIP for solving the optimization problem, however, MIP was failed to provide a solution in a reasonable time for real-size problems. Asgari *et al.* proposed a model-based software to optimize the wheat storage and transportation problem where they also utilized a genetic algorithm approach to find a quality solution [2].

Evolutionary approaches have also been used for other blending problems. Chen and Wang addressed the gasoline blending optimization problem that consists of nonlinear objective functions and nonlinear constraints [11]. They proposed a DNA-based hybrid genetic algorithm where sequential quadratic programming (SQP) was applied in the feasible search region to improve the solution quality. Adams *et al.* proposed a guided master-slave parallel GA for optimizing blending of composite laminates in panel structure [1]. Fomeni formulated blending tea optimization as a multi-objective optimization approach where one of the objectives is to minimize the total cost of the raw materials used and another objective is minimizing the violation of the desired characteristic scores of the final tea blends [17]. The multiobjective approach provides the decision-maker flexibility of balancing the raw material cost and desired blends. They used a Monte Carlo simulation approach to solve the multiobjective optimization problem.

Most of the work for the wheat blending problem in the US market has been done by the milling companies to produce client-specific bread making flour. To the best of our knowledge, our approach is the first to address the grain-mixing problem to increase profitability to the farmers.

In this chapter, we demonstrated the general grain mixing problem and provided the mathematical specifications for the grain mixing problem that reflects the nonlinearity in the objective function and some of the constraints. We then provided a theorem to show that the grain mixing problem is NP-Hard which suggests that finding an optimal mixing

plan may not be feasible and we may need to rely on approximation algorithms for finding a quality solution. We described the datasets that are used across all the experiments and also discussed some of the related works for solving similar optimization problems from the literature.

CHAPTER FOUR

BASELINE APPROACHES

In this chapter, we introduced three basic approaches for solving the grain mixing problem. We use these approaches for performing a baseline comparison to our proposed evolutionary methods. The first method is referred to as the *NoMix* approach that assumes that tracking protein content of the wheat is not available. Thus the wheat distribution profit is computed assuming no grain mixing has occurred. This approach will give us the naïve baseline profit, representing farmers who do not have the protein monitor on their harvesters. The baseline profit will then be used to compare other grain mixing approaches to assess the solution quality.

The second basic approach is referred to as the "greedy" mixing approach where the grain mix is determined based on a lookup table with combinations of bin pairs and pre-computed mixing ratios to provide the highest immediate profit, under the assumption that there is sufficient grain left to fill the trucks. This provides a naïve baseline for farmers with protein monitors but who do not apply our proposed evolutionary method. The deterministic approaches are computationally efficient compared to the evolutionary methods. The greedy solution will help us assess if the added complexity in the evolutionary approaches is beneficial to improve the overall profitability of the farmers.

Finally, we proposed a baseline stochastic *Random* search algorithm to further assess the solution quality of the evolutionary approaches. This algorithm randomly searches the possible search space and returns the best solution after a predefined maximum number of iterations. This algorithm helps us assess if the evolutionary algorithms are exploring the search space efficiently to provide a better solution than just doing a random search.

Algorithm 4.1 NoMix

```
1: Input: Set of Bins, Set of Elevators, Delivery Cost
2: Output: Overall Profit, Loaded truck assignment
3: for each bin in Bin_list:
4:    while grain in bin is not empty:
5:       Load truck fully/partially with grain from that bin
6:       if truck load makes profit: record profit
7:       else: discard grain
8: return sum of truck profits, Loaded truck list
```

4.1 No Mixing Approach

The "no mixing" (*NoMix*) approach calculates the baseline profit without any grain mixing. The algorithm starts by taking a single bin at a time and loading trucks with grain from that bin until the grain in the bin is exhausted. For a partially loaded truck (i.e., when not enough grain is left to load the truck fully), the method checks if that truck provides any profit (i.e., $revenue > delivery\_cost$) when going to an elevator. If not, the grain in that truck is discarded without any penalty, and the truck is freed up to be used to load grain from another bin. To show an example of a partially filled truck, let us consider a bin with 9000 bushels. A truck has a capacity to carry 8000 bushels which is why we will need two trucks to carry the grain from that bin. One will be a fully loaded truck, and another one would carry only 1000 bushels. The method checks if carrying that 1000 bushels is profitable or not. Finally, the total profit is calculated by summing the individual profits from all loaded trucks. The pseudocode for the *NoMix* approach is shown in algorithm 4.1.

Table 4.1: Partial lookup profit table based on a set of premium-dockage curves

ID	**BinPair_1**	**BinPair_2**	**MixRatio**	**Protein**	**Elevator**	**MaxProfit**
1	1	2	0.1	13.63	3	41992
2	1	2	0.2	13.49	3	41992
3	1	2	0.3	13.34	2	38632
4	1	2	0.4	13.20	2	38632
5	1	2	0.5	13.05	2	38632
6	1	2	0.6	12.90	3	37992
7	1	2	0.7	12.76	2	36632
⋮	⋮	⋮	⋮	⋮	⋮	⋮
2155	16	15	0.4	12.61	2	35200
2156	16	15	0.5	12.41	1	33200
2157	16	15	0.6	12.20	1	33200
2158	16	15	0.7	12.00	1	33200
2159	16	15	0.8	11.80	2	33200
2160	16	15	0.9	11.60	2	33200

4.2 Greedy Mixing Approach

The "greedy mixing" (*GreedyMix*) approach first creates a lookup profit table (LPT) by considering all the possible bin pair combinations with different mixing ratios. In our dataset, there are a total of 16 bins. Therefore, there are a total of $P(16, 2) = 16!/(16 - 2)! = 240$ ways to select two bins. Here, we used the permutation to determine the delivery cost because a truck transporting to an elevator only depends on the site number of the second bin. Therefore, the order in which a truck loads grain changes the total profit of that truck.

Algorithm 4.2 GreedyMix

```
Input: Set of Bins, Set of Elevators, Mixing Cost, Delivery Cost
Output: Overall Profit, Loaded truck assignment
Create the lookup profit table (LPT)
Sort the LPT with respect to the MaxProfit
for each entry in the sorted_LPT:
    if both bins have sufficient grain remaining:
        Load truck fully/partially based on bin pair and mixing ratio
        if truck load makes profit: record profit
        else: discard grain
if a single bin has sufficient grain left:
    Load truck and record profit
return sum of truck profits, Loaded truck list
```

Moreover, to make the LPT finite, we discretize the mixing ratio and select from the values $\alpha \in \{0.1, 0.2, 0.3, ..., 0.9\}$. Therefore, in the LPT, there are a total of $240 \times 9 = 2160$ unique entries. Each entry in the LPT also contains an additional field *MaxProfit*. This field tells us the maximum profit a truck can have when fully loaded following the bin pair and mixing ratio combination of that entry. Table 4.1 shows a subset of the LPT for a set of elevators/markets and their premium-dockage curves.

To find a solution, the algorithm sorts the profit table in descending order of *MaxProfit* and traverses it from the top. For any particular combination, if both bins have enough grain left to fill a new truck using the mixing ratio α, the algorithm fills the truck and records the profit. Otherwise, it partially fills the truck with the remaining grain from both bins and includes it in the solution if the truck provides profit (or discards the unprofitable mix otherwise without any penalty). For example, in the case of the combination $((1, 2), \alpha = 0.5)$,

if there are 5,000 bushels left in bin 1 but only 3,000 bushels left in bin 2, the algorithm will take 4,000 bushels from bin 1 (half the capacity of the truck) and 3,000 bushels from bin 2. Therefore, the total number of bushels in that truck will be 7,000, and the mixing ratio will be updated accordingly.

After the first sweep through the LPT, if there exists a single bin with sufficient grain remaining to make a profit (where other bins do not have any bushels left to make a profitable combination), a new truck is filled with the remaining grain from that bin. No mixing cost is incurred for this truck. Finally, the total profit is calculated by summing the individual profits from all loaded trucks. Algorithm 4.2 shows the pseudocode for the greedy mixing approach.

4.3 Random Search Approach

We introduced the *Random* search algorithm as a stochastic baseline method to assess the effectiveness of the proposed evolutionary approaches. The algorithm starts by randomly selecting a subset of the bin pair and mixing combinations out of 2160 total combinations, and using the subset to generate a feasible solution. The process of finding the feasible solution is similar to the *GreedyMix* approach; however, the main difference is in using a subset of the total mixing combinations. The algorithm repeats this process for a predefined maximum number of iterations and returns the best solution. The intuition behind using a subset of combinations is that it limits the possibility of filling trucks greedily based on the maximum profit in a fully loaded truck.

For example, consider a combination $((2, 5), \alpha = 0.9)$ that fills 7200 bushels from bin 2 and 800 bushels from bin 5. If the bin 2 capacity is only 8000 bushels then after loading a truck with the above combination, there will be only 800 bushels left in bin 2. Then if we have another combination $< (2, 9), \alpha = 0.8 >$, it can load 6400 bushels from bin 2, however, there will be only 800 bushels left in bin 2 providing an underfilled truck with less profit. By

Algorithm 4.3 RandomMix

1: **Input:** MaxIter, #MC (number of mixing combination)
2: **Output:** Overall Profit, Loaded truck assignment
3: $best\_profit \leftarrow 0$
4: $best\_assignment \leftarrow \emptyset$
5: **for each** generation in MaxIter:
6: $\quad mixing\_comb \leftarrow$ randomly select #MC mixing combinations
7: $\quad profit, assignment \leftarrow GreedyMix(mixing\_comb)$
8: $\quad$ **if** $profit > best\_profit$
9: $\quad\quad best\_profit \leftarrow profit$
10: $\quad\quad best\_assignment \leftarrow assignment$
11: **return** $best\_profit, best\_assignment$

using a subset of the total combinations, these particular combinations may not be present in the subset ensuring a more balanced distribution of the bushels. Algorithm 4.3 shows the pseudocode for the *Random* search method. At line 7, we use the *GreedyMix* method to generate a feasible solution but instead of using all the combinations, it only generates solutions based on the subset of the mixing combination.

The subset size parameter (i.e., the number of mixing combinations) plays an important role in finding a quality solution. It has to be selected in a way so that it contains enough bin combinations to empty all the bushels of the bins. If the subset is too small, then it may not contain an entry for a particular bin; therefore, all of the bushels of that bin may be unused in the solution. If the subset is too large, then it will perform like the *GreedyMix* algorithm with all the choices of filling a truck.

To select the subset size, we ran separate experiments considering subset size ranges from 50 to 150 (from a total of 2160 combinations), increasing by 10 on each experiment.

Then we monitor the solutions to evaluate if all of the grain was utilized or not. For a subset size of 100, more than 90% of the solutions were able to utilize all of the grain. Therefore, for the *Random* search, the subset size was fixed to 100 mixing combinations, and we used 100 iterations for all of the experiments. Increasing the number of iterations above 100 did not provide significant improvement in the solution quality, and we used 100 iteration to make the algorithm computationally efficient. The deterministic approaches always yield the same solution; however, the *Random* search algorithm is stochastic. So to evaluate its performance, we ran 10 experiments on each dataset and recorded the average overall profit.

4.4 Results and Analysis

To evaluate the performance of the baseline algorithms, we used the real datasets for the years 2016 and 2017 along with the twelve artificially created datasets. For a detailed description of the datasets, please see Section 3.4. Table 4.2 shows the solutions obtained from the *NoMix*, *GreedyMix*, and *Random* search approaches. In the table, the real datasets are represented by R_year, the flipped market datasets are represented by RF_year, and the artificial datasets that were created by randomly assigning bushels and protein contents are represented by A^*_year for each year respectively. The values represent the profit in thousands of US dollars.

The bold values represent the highest profit comparing the three baseline approaches, and the underline values represent substantial profit increase compared to the baseline *NoMix* approach. We assumed more than \$5000 profit increase as substantial profit increase because the price for purchasing a protein content tracking device cost is roughly between \$4000 and \$5000. The profit from the *Random* search algorithm represents the mean and standard deviation over 10 runs.

Based on the solutions on Table 4.2, we see that *GreedyMix* increased the overall profit for the real datasets with a substantial profit increase for the year 2017. We see

Table 4.2: Performance of the baseline algorithms (profit in thousands of dollars)

Dataset	**Random**	**Greedy**	**NoMix**
R_2016	427.5 ± 0.86	**430.5**	424.5
R_2017	489.7 ± 0.53	**491.6**	487.1
RF_2016	**500.9** ± 0.63	499.9	499.7
RF_2017	420.6 ± 0.60	**423.1**	418.0
A1_2016	**405.6** ± 1.21	394.5	395.4
A2_2016	**548.9** ± 2.24	533.1	545.2
A3_2016	568.3 ± 3.39	543.9	**569.9**
A4_2016	**556.4** ± 1.92	520.4	539.9
A5_2016	**512.7** ± 2.51	496.9	512.2
A1_2017	595.8 ± 2.28	581.7	**608.9**
A2_2017	520.1 ± 1.68	515.8	**520.6**
A3_2017	**661.8** ± 2.32	655.8	643.9
A4_2017	**442.4** ± 1.16	435.6	430.1
A5_2017	**670.9** ± 2.95	656.0	663.5

a similar substantial profit increase for the flipped market *RF*_2017 dataset. However, for most artificial datasets, *GreedyMix* failed to provide a solution better than no mixing at all (except for *A*3_2017 and *A*4_2017 where we see a substantial profit increase). The *Random* search algorithm was able to increase the profit compared to no mixing in most of the datasets except for three artificial datasets and provided a substantial profit increase in five artificial datasets. In Table 4.3, we show the complete solutions from the *NoMix* algorithm for the real 2017 dataset. Each row in the table represents a single truck. The column "Bin#" represent from which bin the truck loads grain, "Protein" is the protein content of the grain

Table 4.3: Complete Solution from *NoMix* algorithm for *R_2017* dataset

Bin#	**Protein**	**Load**	**Elv**	**Profit**
1	12.32	8000.0	1	33760.0
1	12.32	6836.8	1	28851.3
2	13.78	7292.5	3	38285.6
3	12.71	6395.3	2	28587.0
4	11.34	8000.0	1	33360.0
4	11.34	525.5	2	1591.3
5	10.88	5713.4	1	23253.4
6	12.58	7703.7	2	34820.6
7	10.35	1712.7	1	6267.5
8	10.72	4192.7	1	17273.8
9	13.15	6539.8	2	31195.0
10	10.11	7292.5	1	29461.7
11	12.38	4921.3	1	21112.5
12	12.01	7089.7	2	30698.3
13	10.83	5050.0	1	20553.5
14	12.13	8000.0	1	34320.0
14	12.13	657.4	1	2218.6
15	13.41	2985.3	3	15170.8
16	11.40	8000.0	1	33360.0
16	11.40	5509.0	1	22972.5
Total		**112417.5**		**487113.3**

in the truck, "Load" is the number of bushels in the truck (maximum is 8000), "Elv" is the elevator from which the truck gets the highest profit, and "Profit" shows the amount paid by the elevator for grains in the truck after deducting the mixing and delivery cost.

Table 4.4 shows the complete solution of the *GreedyMix* algorithm for the real 2017 dataset. It considers mixing different protein content grains to improve the overall profitability. Each row in the solution represents a truck entry. The column "Pair" identifies the pair of bins mixed to load a truck,"P1_Bu" and"P2_Bu" specifies the number of bushels taken from each bin in the pair, "Protein" shows the weighted average protein level, "Load" gives the total amount of wheat loaded in the truck, and "Elv" identifies the elevator where the truck delivers the grain. The last truck entry (i.e., with bin pair $(13, 1)$ shows that after the first sweep in the lookup profit table, grain remained only in bin 13. Therefore, the algorithm loaded a truck only with the remaining grain from bin 13 (no mixing cost was incurred for this case) to utilize all the grain and maximize the profit.

On the surface, one may suppose that the *Greedy* solution should provide the optimal results since it creates all possible combinations of mixes and selects the mix greedily based on the maximum profit. This claim might be true if there existed an infinite number of bushels in all of the bins. However, in the real world, our problem is constrained to consider a limited (and possibly different) number of bushels for different bins, and loading a truck greedily based on profit may not always be possible if there are no bushels left in a specific bin. Furthermore, the truck might be partially filled based on the remaining bushels of bins, giving a lower profit than in an infinite grain context. For example, if we look at the second truck with bin pair $(2, 5)$ in Table 4.4, it should provide a profit similar or slightly lower than the profit from truck one. However, bin 2 has only 92.5 bushels left after filling up truck one which is why truck two ended up being a partial fill truck with a profit much lower than the maximum profit in the LPT. Therefore, for all the test cases, the *Greedy* solution not only failed to provide optimal results but also gave a lower profit (in the artificial datasets) than

the baseline no mix solution.

The complexity of the grain mixing problem (Section 3.3) suggests that finding the optimal solution may not be feasible. The *GreedyMix* algorithm is simple and computationally efficient, however, the solution may not be a quality solution. In the next two chapters, we will utilize evolutionary approaches to solve the grain mixing problem. They are stochastic algorithms and more complex than the deterministic approaches. We will assess if the added complexity is beneficial to improve the overall solution quality compared to the baseline approaches.

Table 4.4: Complete Solution from *GreedyMix* algorithm for *R_2017* dataset

#	Pair	Mix	P1_Bu	P2_Bu	Protein	Load	Elv	Profit
1	(2, 4)	0.90	7200.0	800.0	13.54	8000.0	3	42080.0
2	(2, 5)	0.10	92.5	800.0	11.17	892.5	2	3112.8
3	(9, 12)	0.89	6539.8	800.0	13.03	7339.8	2	35378.0
4	(15, 5)	0.79	2985.3	800.0	12.88	3785.3	3	17840.9
5	(3, 12)	0.70	5600.0	2400.0	12.50	8000.0	2	36560.0
6	(3, 12)	0.33	795.3	1600.0	12.24	2395.3	2	10123.0
7	(6, 12)	0.90	7200.0	800.0	12.52	8000.0	2	36240.0
8	(1, 6)	0.61	800.0	503.7	12.42	1303.7	1	5014.4
9	(11, 12)	0.77	4921.3	1489.7	12.30	6411.0	2	27753.2
10	(1, 4)	0.20	1600.0	6400.0	11.54	8000.0	2	34552.0
11	(1, 4)	0.64	2400.0	1325.5	11.97	3725.5	2	16049.3
12	(1, 5)	0.90	7200.0	800.0	12.18	8000.0	2	34480.0
13	(1, 5)	0.64	2836.8	1600.0	11.80	4436.8	2	19122.6
14	(14, 7)	0.90	7200.0	800.0	11.95	8000.0	2	33840.0
15	(14, 7)	0.61	1457.4	912.7	11.44	2370.1	1	9278.7
16	(16, 10)	0.90	7200.0	800.0	11.27	8000.0	1	33120.0
17	(16, 10)	0.70	5600.0	2400.0	11.01	8000.0	1	33120.0
18	(16, 10)	0.31	709.0	1600.0	10.51	2309.0	1	9024.0
19	(8, 10)	0.72	4192.7	1600.0	10.55	5792.7	1	23923.7
20	(5, 10)	0.66	1713.4	892.5	10.62	2605.9	1	10407.0
21	(13, 1)	0.00	5050.0	0.0	10.83	5050.0	1	20553.5
Total						**112417.5**		**491573.0**

CHAPTER FIVE

GENETIC ALGORITHM GRAIN MIXING

The first evolutionary approach we developed to solve the grain mixing problem is based on the genetic algorithm (GA). We introduced a novel permutation-based representation for the evolutionary approaches tailored explicitly for the grain mixing problem. To be more precise, we use the permutation of grain bins to specify an order for how to load trucks so as ensure the constraints of the problem remain satisfied. And the objective function of the evolutionary approaches is to find the best sequence of loaded trucks that maximizes the overall profit through grain mixing. In this chapter, we discuss our specific implementation of the GA approach for solving the grain mixing problem and present the experimental results with a detailed analysis of the results.

5.1 Encoding

For an individual chromosome in the target population, we utilize a permuta-tion-based representation where each gene in the chromosome is a unique tuple $\langle ID, bin\_pair, \alpha \rangle$. There are a total of 16 grain bins in our dataset, and we use 9 discrete mixing ratios $\alpha \in \{0.1, 0.2, ..., 0.9\}$, which gives a total of $2,160$ unique mixes. It is similar to the lookup profit table (LPT) defined for the deterministic *GreedyMix* approach where we created a tabular form for all the mixes (see Table 4.1). The attribute ID in a gene represents an integer identifier for each combination. Therefore, each ID in the chromosome is going to appear exactly once to ensure a feasible solution in the search space. When generating a potential solution for an individual, the mixing combination in each tuple is used for loading trucks (fully/partially) as long as the bin pairs have grain left to do so. An example of the chromosome representation is shown in Figure 5.1.

1	2	3		$m-1$	m
21,(1,4),0.3	309,(3,6),0.3	2,(1,2),0.2	$\cdots$	981,(8,4),0.9	356,(3,11),0.5

Figure 5.1: Sample encoding for genetic algorithm

5.2 Initial Population

For the initial population P, we create N individuals where each individual's chromosome consists of m random mixing combinations (without any duplicate ID) sampled from the total 2160 combinations where the chromosome size m is also a tunable parameter. These m mixing combinations in the chromosome list are used for loading trucks when generating a potential solution. Therefore, the size of m should be sufficiently large to transport all of the grain to the elevator.

5.3 Fitness Function

The fitness of an individual I is calculated as follows:

$$fitness(I) = IndividualMix()$$

where the method *IndividualMix* takes the m mixing combinations from the individual's chromosome as input. To generate a feasible solution, it first calculates the maximum profit that can be achieved for each combination (assuming a fully loaded truck for that combination), and sorts the mixing combinations with respect to the maximum profit. Then, it performs two sweeps in the sorted combination list in order to load the trucks. In the first sweep, for a particular combination, if both bins in the combination have enough grain left to load a truck fully/partially and can make a profit, it loads the truck and records the

Algorithm 5.1 IndividualMix

```
1: Input: Individual chromosome of m mixing combinations (MC)
2: Output: Overall Profit, Loaded truck assignment
3: Sort MC with respect to the maximum profit
4: for each combination in the sorted_MC:
5:    if both bins have sufficient grain remaining:
6:       Load truck fully/partially based on bin pair and mixing ratio
7:       if truck makes profit: record profit
8:       else: discard grain
9: for each combination in the unused sorted MC:
10:   if single bin has sufficient grain remaining:
11:      Load truck fully/partially from the single bin
12:      if truck makes profit: record profit
13:      else: discard grain
14: return sum of truck profits
```

profit. Otherwise, it discards the truck and moves to the next combination. In the second sweep, it checks for unused combinations in the sorted list, and for a particular combination, if a single bin has enough grain left to make a profit, it loads a truck without incurring any mixing cost. The second sweep is important as the size of m is generally much lower than the total 2,160 combinations, and some bin-pair combinations might not be present in the chromosome list. if the size of m is close to the total number of mixing combinations, then it will perform similarly to the greedy approach. Finally, the method loads l trucks out of m mixing combinations (where $l < m$) ensuring all of the constraints remained satisfied and returns the total profit of all loaded trucks. The pseudocode for *IndividualMix* is shown in Algorithm 5.1.

5.4 Selection

The GA uses tournament selection [50] to select parents from the current population to generate new offspring for the next generation. A tournament consists of s randomly selected individuals from the current population, and the individual with the highest fitness (tournament winner) is added to the mating pool to generate new offspring. Several "tournaments" take place in order to select enough parents from the population and to generate new offspring for the next generation. The selection pressure that controls the GA's convergence rate depends highly on the tournament size.

5.5 Crossover

After the selection process, the GA applies a crossover operator to generate new offspring. For this study, we adapted two different crossover operators – OX and PMX – designed for handling permutation-based GA representations. A brief description of these operators on the TSP representation is given in Section 2.2.4.1 and Section 2.2.4.2, respectively. For the grain mixing representation, these operators closely follow the TSP representation. However, the major difference is that, unlike TSP, the grain mixing representation does not follow a strict permutation. In the permutation-based TSP representation, all of the cities appear in the chromosome list and, based on the order in which they are arranged, the fitness value changes. However, in the grain mixing representation, the chromosome list will utilize a subset of the 2,160 mixing combinations, scanned according to the permutation. Therefore, for two different individuals, the mixing combinations selected in the chromosome may not be exactly the same.

For example, let us consider a total of 10 mixing combinations (i.e., ID ranging from 1 through 10) with arbitrary bin pairs and mixing ratios α. Let us assume the chromosome size of an individual is 5 (i.e., we randomly draw 5 combinations out of the total 10 combinations).

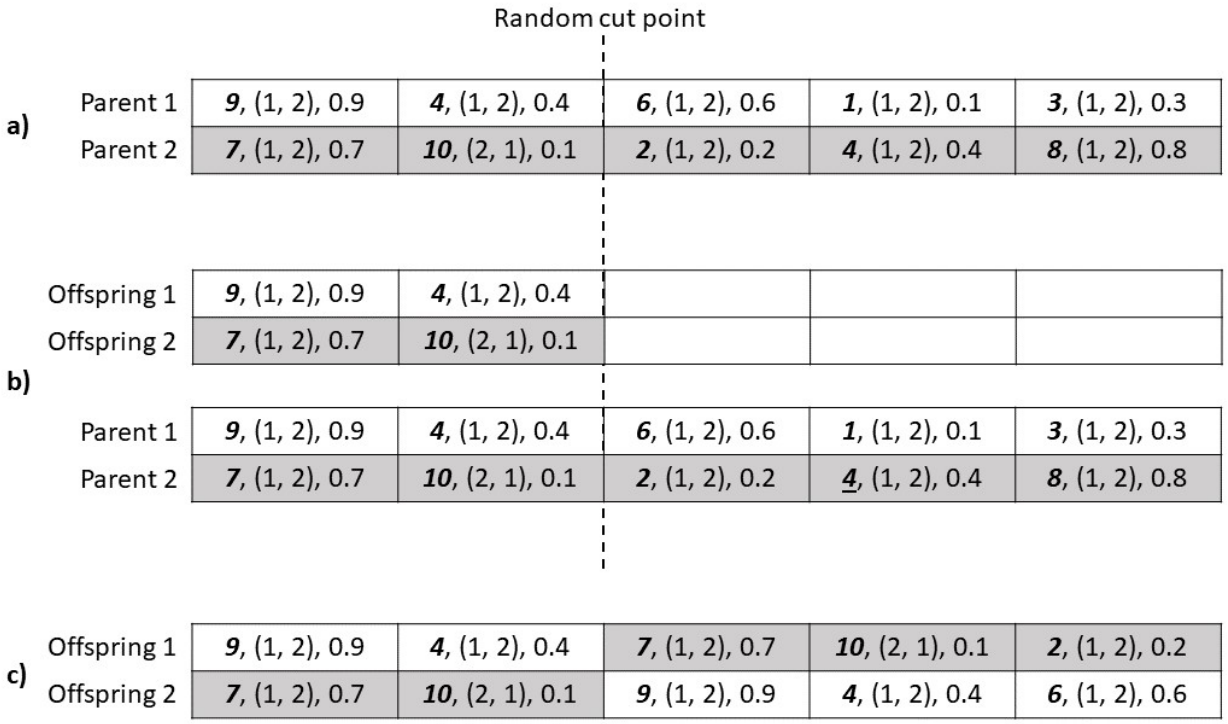


Figure 5.2: Proposed OX operator example

Figure 5.2 illustrates our proposed OX operator for generating two offspring from selected parents for the grain mixing representation. Notice that the chromosome representation in the parents does not contain the same elements unlike all the city vertices in the TSP representation. Therefore, when generating offspring with the proposed OX operator, we end up getting new offspring that contain genes from both parents and a slightly different chromosome representation with respect to the parents. Thus the OX operator maintains a feasible chromosome representation following the permutation representation and also introduces new mixing combinations in the chromosome like a traditional one-point crossover operator.

Figure 5.3 shows our proposed PMX operator on the example problem discussed above. Like the OX operator, we used the permutation-based PMX operator to enforce feasibility in the chromosome representation, and introducing new mixing combinations in the offspring

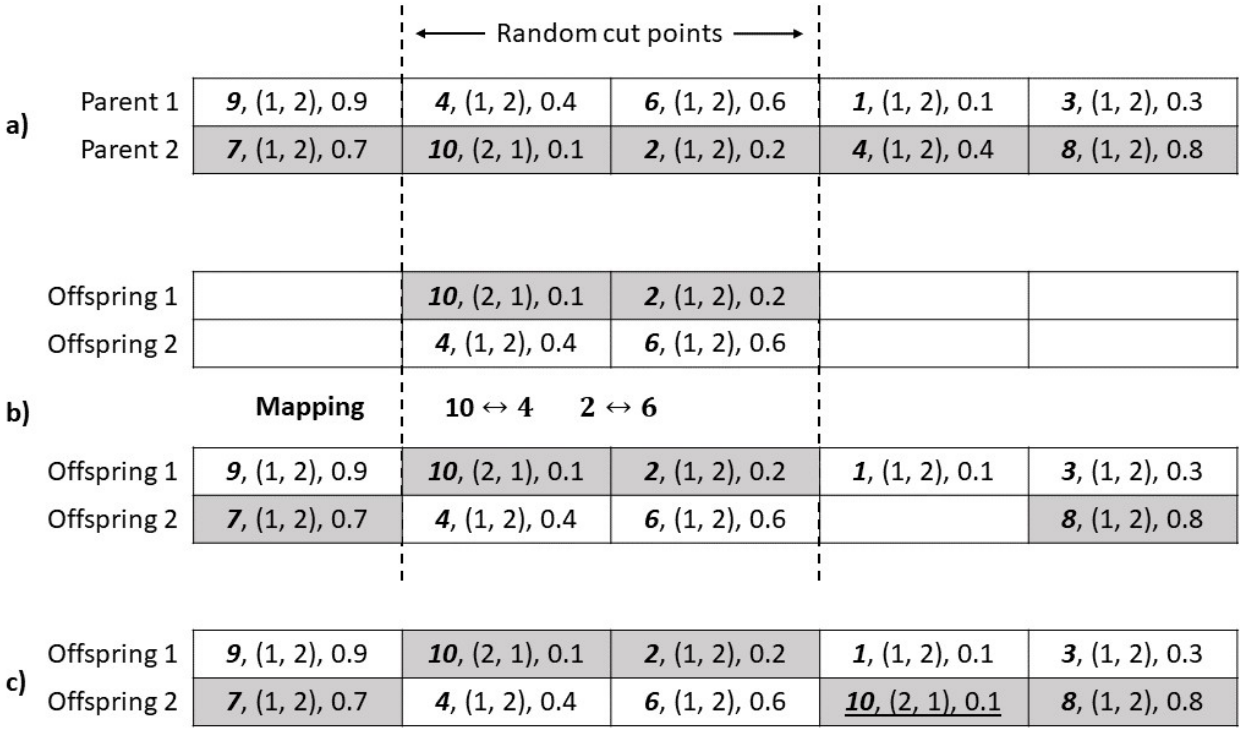


Figure 5.3: Proposed PMX operator example

can be viewed as a traditional two-point crossover operator.

Another major difference with the permutation-based crossover operator is that our fitness function sorts the mixing combinations in the chromosome based on the maximum profit. Therefore the relative order in which the combinations appear does not affect the fitness value; however, the permutation interpretation allows the problem constraints to be enforced. Then introducing new mixing combinations in the chromosome list affects the fitness value. Although a gene is represented by the tuple $\langle ID, bin\_pair, \alpha \rangle$ where the bin_pair and mixing ratio α is used to load trucks in the fitness function, only the integer combination ID is used to shuffle the genes in the proposed crossover operators.

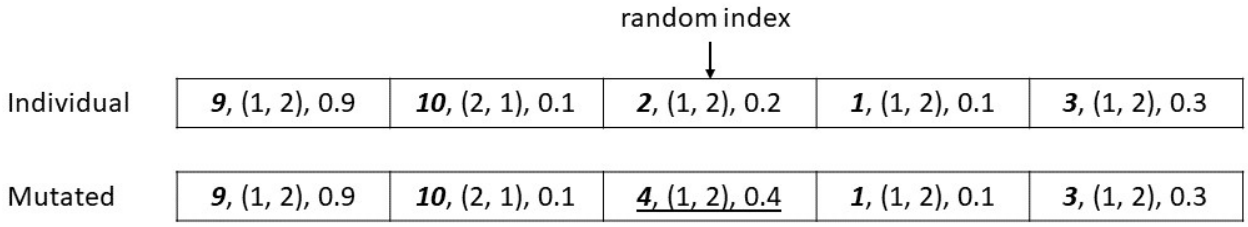


Figure 5.4: Proposed Mutation operator example

5.6 Mutation

After applying the crossover operator, the mutation operator is applied to the current population to add diversity for the next generation. Our mutation operator randomly selects a subset of individuals in the current population based on the mutation probability. Then, for each selected individual, it swaps a randomly selected mixing combination in the chromosome with a new mixing combination that is not already present in the chromosome. Finally, it recalculates the fitness of the mutated individuals to obtain a feasible solution (fitness value). Figure 5.4 shows an example of our mutation operator applied to an individual chromosome. The chromosome representation is similar to the example problem that you used for the crossover operator demonstration. In the figure, the randomly selected entry with ID 2 is replaced by the combination ID 4 after the mutation step.

5.7 Experimental Design

For this study, we were able to collect real wheat harvest information for the years 2016 and 2017 from a Northwest Montana farmer. We created twelve additional datasets to further evaluate the performance of the proposed grain mixing methods. For the detailed description of the datasets, please see Section 3.4. The GA contains many hyperparameters that need to be tuned to obtain a quality solution. Next, we discussed the hyperparameters

Table 5.1: Parameter settings for the GA algorithm

#MaxIter	**Population**	**#MC**	**Tournament**	**Mutation**
500	200	50	5	0.2

used in the GA for solving the grain mixing problem.

For the initial population, the GA implementations create random individuals by taking a small subset of the total mixing combinations to form the chromosome. The intuition was that, for a small subset of the mixing combinations, the *IndividualMix* method would return a different result than the *GreedyMix* algorithm based on the few choices of bin combinations. The goal of the GA's search mechanism is to find the best sequence from all the mixing combinations for which the total profit is maximized. Therefore, chromosome size plays an important role in the overall solution. The subset has to be selected in such a way that it contains sufficient bin combinations to empty all of the grain in the bins. If the subset is too small, then it may not contain an entry for a particular bin; therefore, all of the bushels of that bin may go unused in the solution. If the subset is too large, then it will perform like the *GreedyMix* algorithm with all of the choices for filling a truck.

The hyperparameters, along with the size of the chromosome used in the GA algorithm, were tuned manually. We performed a grid search on the hyperparameters to find a suitable set of parameters. We used a set of $\{30, 50, 100, 200, 500\}$ for the population size, a set of $\{2, 5, 10, 15\}$ for the tournament size, and a set of $\{0.05, 0.1, 0.2\}$ for the mutation probability. For the chromosome size, we used subset size (i.e., the number of mixing combinations) ranges from 50 to 100 mixing combinations with an increment of 10. As mentioned in the *Random* search approach (Section 4.3), solutions obtained using 100 mixing combinations were able to utilize all of the grain 90% of the time. For GA, it can be more flexible as the intuition is that the individual with a lower fitness will eventually be replaced by an individual with

higher fitness.

We performed the grid search on the real datasets to evaluate the solution quality and selected the hyperparameters that provided the highest average profit. Table 5.1 shows the selected parameter values used in the GA algorithm for all of the test cases. Column *#MaxIter* represents the maximum generation number, and column *#MC* represents the number of mixing combinations in the chromosome list (chromosome size).

5.8 Results and Analysis

5.8.1 Profitability Analysis

First, we show the performance of the proposed GA approach compared to the *Random* search and the deterministic baseline methods for all test datasets in Table 5.2. The column *Dataset* identifies which data was used by the algorithms to generate the results. The notation *R_year* indicates the real dataset for the respective year. *RF_year* indicates the flipped market version of the real datasets, and A^**_year* indicates the artificial datasets for each of the respective years. The values in the table show the overall profit obtained by each algorithm, expressed in thousands of US dollars. For GA, there are two versions, one for each of the different adapted operators (i.e., OX and PMX). The overall profit from the GA and the *Random* search are the mean and standard deviation over 10 runs of experiments; however, the profit did not change for the deterministic algorithms (Greedy, and NoMix). The **bold** values represent the maximum profit obtained across the algorithms for a respective dataset, and the underline values represent a sufficient profit increase from the base *NoMix* solution (which is more than $5000 in our case).

Based on the profit obtained from different methods, we note that the proposed GA leads to a higher overall profit compared to the baseline *NoMix* solution in every case. GA's adapted OX and PMX operators consistently provided a sufficient profit increase for all the real and artificial datasets. Comparing the two crossover operators of the GA, the PMX

Table 5.2: Performance of the proposed GA algorithm (Profit in thousands of dollars)

Dataset	**GA**		**Random**	**Greedy**	**NoMix**
	OX	PMX			
R_2016	435.3 ±1.04	**435.5** ±1.53	427.5 ±0.86	430.5	424.5
R_2017	495.5 ±0.59	**496.1** ±0.98	489.7 ±0.53	491.6	487.1
RF_2016	505.3 ±1.03	**506.0** ±0.54	500.9 ±0.63	499.9	499.7
RF_2017	427.6 ±1.36	**428.7** ±1.34	420.6 ±0.60	423.1	418.0
A1_2016	416.6 ±1.39	**418.2** ±2.08	405.6 ±1.21	394.5	395.4
A2_2016	562.2 ±2.44	**563.9** ±1.85	548.9 ±2.24	533.1	545.2
A3_2016	589.8 ±2.63	**592.6** ±1.91	568.3 ±3.39	543.9	569.9
A4_2016	575.3 ±1.60	**576.6** ±1.91	556.4 ±1.92	520.4	539.9
A5_2016	534.9 ±1.59	**536.4** ±1.53	512.7 ±2.51	496.9	512.2
A1_2017	621.8 ±3.38	**623.9** ±2.37	595.8 ±2.28	581.7	608.9
A2_2017	537.0 ±2.16	**539.4** ±1.98	520.1 ±1.68	515.8	520.6
A3_2017	675.3 ±1.49	**678.5** ±1.96	661.8 ±2.32	655.8	643.9
A4_2017	451.3 ±1.56	**452.7** ±0.87	442.4 ±1.16	435.6	430.1
A5_2017	684.5 ±1.58	**687.0** ±2.71	670.9 ±2.95	656.0	663.5

operator for all test cases provided a higher profit than the OX operator. The increased profit from the artificial datasets also suggests that mixing grain always improves the profitability

of the farmers.

The overall profit from the *Random* and *Greedy* algorithms are consistently lower than the GA for all test cases. Furthermore, for most of the test cases, they failed to provide a sufficient profit increase from the base solution, and for some datasets yielded a profit lower than the base solution. Based on the performance of the *Random* algorithm, it is evident that the GA is effectively exploring the search space to find a better solution, and the greedy solution suggests that the added complexity of the evolutionary approaches is beneficial for finding a better overall profit. Unfortunately, the cost associated with the production (harvesting) was not provided to us, which would be necessary to present a more complete estimate of profit.

The results obtained from GA crossover operators and *Random* search algorithms are stochastic. To that extent, we performed statistical significance testing between these algorithms to assess if the difference in results is statistically significant or not. We utilized a paired t-test [65] for the significance testing and used a threshold of $\alpha = 0.05$ for the p-value. Table 5.3 shows the significance testing between these algorithms. The header rows and columns represent which two algorithms were compared. The values represent a 3-tuple, where the first value indicates for how many datasets the row algorithm's profit was better than the column algorithm, the second value indicates for how many datasets the column algorithm's profit was better than the row algorithm, and the third value indicates for how many datasets the difference was not statistically significant. We have a total of 14 datasets, and each 3-tuple sums up to 14.

As we can see in Table 5.3, GA's OX and PMX operators significantly improved the profit compared to the *Random* search algorithm for all of the datasets. For OX and PMX operators, we notice that PMX was able to provide a better profit for 13 of the datasets compared to the OX operator, which differed from our proposed hypothesis. We hypothesized these two crossover operators to have similar solution quality.

Table 5.3: Statistical significance testing between GA and Random algorithms

	OX	**PMX**	**Random**
OX	N/A	0, 13, 1	14, 0, 0
PMX	13, 0, 1	N/A	14, 0, 0
Random	0, 14, 0	0, 14, 0	N/A

The adapted PMX operator generates offspring similar to a traditional two-point crossover operator, whereas the adapted OX operator is similar to a one-point crossover. Due to our permutation representation, it may be possible that the adapted OX operator with a single random cut point is favoring one of the parent's genes more than the other when generating new offspring. However, for the adapted PMX, which randomly selects two cut points, the generated offspring are more likely to share genes from both parents, which may explain the performance difference. We presented a population diversity analysis in Section 5.8.2 to further evaluate the performance difference between these two operators.

Next, we examine the best complete solutions obtained from the GA with the OX operator in Table 5.4 and the PMX operator in Table 5.5 for the real R_2017 dataset. In these tables, each row represents a separate truck entry. The column name "Pair" identifies the pair of bins mixed to load a truck, "P1_Bu" and "P2_Bu" specify the number of bushels taken from each bin in the pair, "Protein" shows the weighted average protein level, "Load" gives the total amount of wheat loaded in the truck (max 8,000 bu) and "Elv" identifies the elevator where the truck delivers the grain.

The mixing ratio of the last four truck entries (i.e., from 16 to 19) in the OX solution (Table 5.4) is 0, which indicates that after the first sweep in the mixing combination (chromosome list), sufficient grain remained in bin 12 and bin 16. This implies that the chromosome in the best individual does not contain any combination with bin pair $(12, 16)$.

Table 5.4: Best solution from GA(OX) for R_2017 dataset

#	Pair	Mix	P1_Bu	P2_Bu	Protein	Load	Elv	Profit
1	(9, 2)	0.50	4000.0	4000.0	13.47	8000.0	3	41600.0
2	(15, 2)	0.65	2985.3	1600.0	13.54	4585.3	3	23843.6
3	(9, 2)	0.60	2539.8	1692.5	13.40	4232.3	3	22008.1
4	(3, 12)	0.80	6395.3	1600.0	12.57	7995.3	2	36538.5
5	(1, 6)	0.10	800.0	7200.0	12.55	8000.0	2	36080.0
6	(1, 4)	0.20	1600.0	6400.0	11.54	8000.0	2	34552.0
7	(6, 4)	0.19	503.7	2125.5	11.58	2629.2	2	11126.1
8	(1, 5)	0.60	4800.0	3200.0	11.74	8000.0	2	34480.0
9	(1, 7)	0.77	5600.0	1712.7	11.87	7312.7	2	31298.2
10	(10, 11)	0.50	4000.0	4000.0	11.25	8000.0	1	33912.0
11	(1, 8)	0.56	2036.8	1600.0	11.62	3636.8	2	15274.7
12	(14, 5)	0.61	4000.0	2513.4	11.64	6513.4	2	27486.4
13	(13, 11)	0.84	4800.0	921.3	11.08	5721.3	1	23972.3
14	(14, 8)	0.64	4657.4	2592.7	11.62	7250.1	2	30232.8
15	(13, 10)	0.07	250.0	3292.5	10.16	3542.5	1	14052.2
16	(12, 9)	0.00	5489.7	0.0	12.01	5489.7	2	23770.3
17	(14, 16)	0.00	0.0	4000.0	11.40	4000.0	1	16680.0
18	(5, 16)	0.00	0.0	5600.0	11.40	5600.0	1	23352.0
19	(16, 4)	0.00	3909.0	0.0	11.40	3909.0	1	16277.8
Total						**112417.5**		**496537.0**

Therefore, it loaded the truck separately with the remaining grain without incurring any mixing cost. Without bin pair $(12, 16)$, this solution provided the highest profit for the

Table 5.5: Best solution from GA(PMX) for R_2017 dataset

#	Pair	Mix	P1_Bu	P2_Bu	Protein	Load	Elv	Profit
1	(9, 2)	0.60	2539.8	1692.5	13.40	4232.3	3	22008.1
2	(9, 2)	0.50	4000.0	4000.0	13.47	8000.0	3	41600.0
3	(3, 12)	0.70	5600.0	2400.0	12.50	8000.0	2	36560.0
4	(6, 12)	0.90	7200.0	800.0	12.52	8000.0	2	36240.0
5	(15, 2)	0.65	2985.3	1600.0	13.54	4585.3	3	23843.6
6	(3, 6)	0.61	795.3	503.7	12.66	1299.0	2	5318.1
7	(8, 12)	0.29	1600.0	3889.7	11.64	5489.7	2	23764.8
8	(1, 4)	0.30	2400.0	5600.0	11.63	8000.0	2	34552.0
9	(1, 4)	0.58	4000.0	2925.5	11.91	6925.5	2	29911.2
10	(1, 5)	0.50	4000.0	4000.0	11.60	8000.0	2	34480.0
11	(1, 7)	0.72	4436.8	1712.7	11.77	6149.5	2	26319.7
12	(10, 11)	0.60	4800.0	3200.0	11.02	8000.0	1	33912.0
13	(14, 13)	0.70	5600.0	2400.0	11.74	8000.0	2	33760.0
14	(14, 5)	0.64	3057.4	1713.4	11.68	4770.8	2	20132.6
15	(13, 11)	0.61	2650.0	1721.3	11.43	4371.3	1	18315.8
16	(16, 10)	0.80	6400.0	1600.0	11.14	8000.0	1	33120.0
17	(8, 10)	0.74	2592.7	892.5	10.56	3485.2	1	14301.1
18	(16, 1)	0.00	7109.0	0.0	11.40	7109.0	1	29644.5
Total						**112417.5**		**497783.6**

GA with the OX operator. We observed a similar scenario in the PMX solution (Table 5.5) where the last truck entry (truck 18) shows a mixing ratio of 0, and sufficient grain remained in bin 16. Both best solutions loaded grain from bin 16 separately, which implies

that mixing bin 16 with other bins might reduce the overall profit. We also observed that the best PMX solution was able to load all the grains using 18 trucks whereas the best OX solution required 19 trucks thus providing a better overall profit. A complete solution from the mixing algorithms shows a farmer how to load each truck with bushels from respective bins to generate the solution's overall profit.

5.8.2 Population Diversity

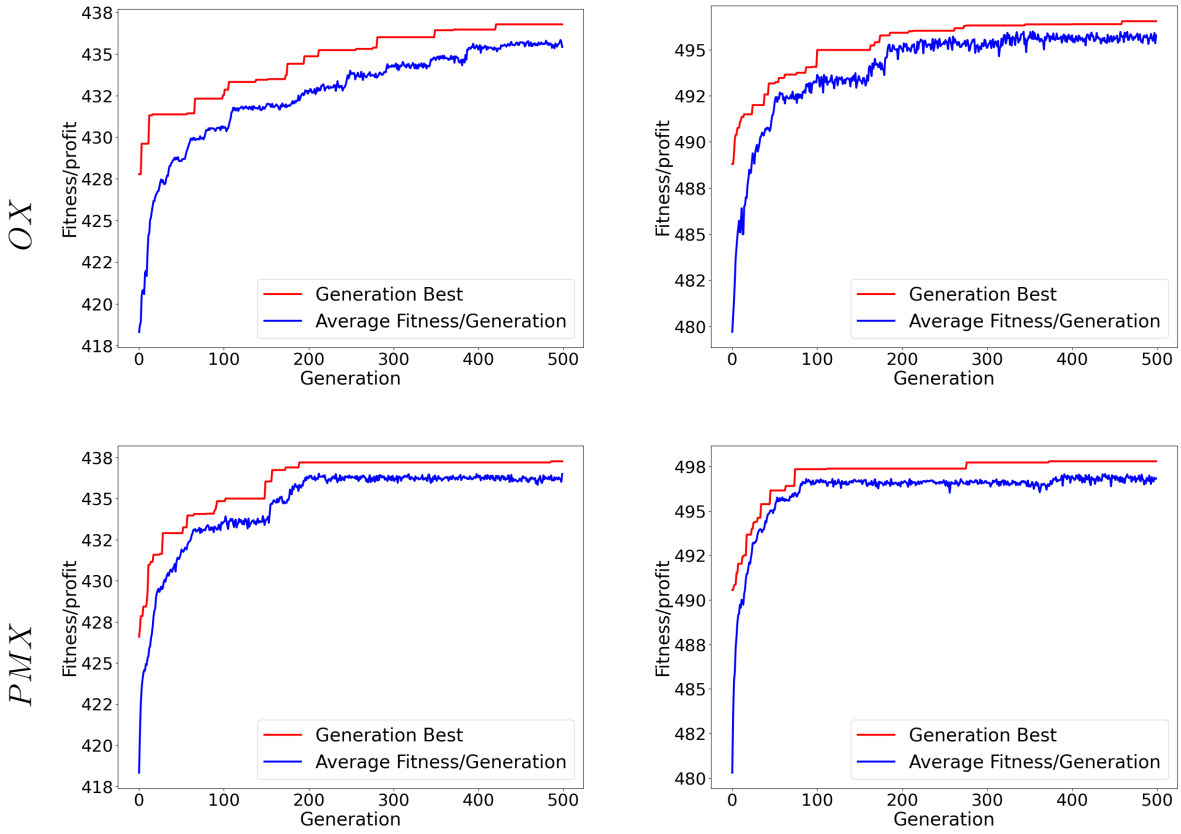
The GA often suffers from prematurely converging into a local optimum due to loss of diversity in the population. One approach to evaluate this is by considering the extent to which population diversity is lost as evolution proceeds. Therefore, we completed an analysis of population diversity for the proposed GA approach.

5.8.2.1 Fitness Diversity

One common approach to monitoring population diversity is to track the fitness values of the individual in each generation. To assess the population diversity, we tracked the fitness (profit) of the best individual and the average fitness of the population in each generation. Figure 5.5 shows the fitness diversity of the best solutions obtained from the OX and the PMX operator for the real 2016 and 2017 datasets. The y axis in the subfigures represents the profits in thousands of US dollars. The average population fitness curve (i.e., the blue line) for all of the subfigures is not smooth as GA's crossover and mutation operator may create a new individual with a lower/higher fitness providing a lower/higher average profits. We observe that the average fitness curve tends to converge to the generation best for both OX and PMX indicating a loss in the population diversity in terms of the fitness values.

5.8.2.2 Genotypic and Phenotypic Diversity

Since our permutation-based GA representation is composed of mixing combinations, for different mixing combinations the fitness might be similar. Therefore, in our case, using only fitness values to monitor diversity



a) R_2016 b) R_2017

Figure 5.5: Fitness Diversity of the OX and PMX operator

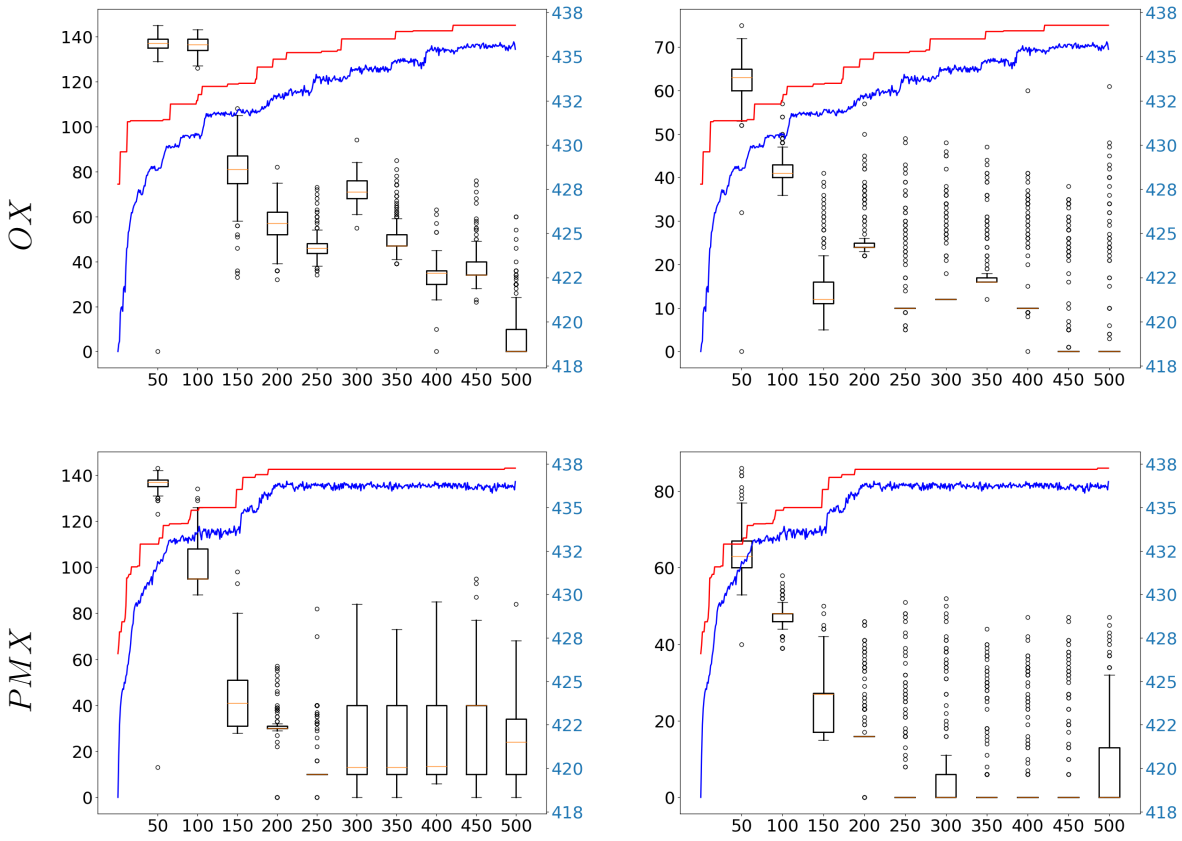
may not portray the actual scenario. Moreover, not all of the mixing combinations in the chromosome are used in the final solution returned by the *IndividualMix* method due to varying grain content in the bins. Therefore, we analyzed both 'Genotypic diversity', which compares mixing combinations in the chromosome between two individuals (i.e., structural differences), and 'Phenotypic diversity', which compares the actual mixing combinations used in the final solution (generated by *IndividualMix*) between two individuals (i.e., behavioral differences). A survey for different population diversity measures for permutation-based GA was presented in [82].

To compute diversity, we used Levenshtein distance (i.e., edit distance) [44] to compare

different mixing combinations. Levenshtein distance is used to measure the distance between two strings by determining the minimum number of insertions, deletions, and substitutions required to transform one string into another. To use Levenshtein distance for our diversity measure, we encoded the mixing combinations to make them into strings. Recall that a mixing combination is a tuple $\langle bin\_pair\ 1, bin\_pair\ 2, \alpha \rangle$. We have 16 bins in our datasets, which are encoded hexadecimally (from 0 to F), and 10 mixing ratios (including 0, meaning no mixing) that are encoded from 0 to 9. For example, the string encoding for the combination $\langle 2, 15, 0.6 \rangle$ becomes "1E6". In this way, all of the mixing combinations in a chromosome are transformed into a sequence of strings.

When generating the final solution, a mixing ratio may not be a single decimal value due to having a partially loaded truck. In that case, the value is encoded to the closest integer (e.g., 0.64 would be encoded to 6). As measuring edit distance between all pairs of individuals for all of the generations is computationally expensive, we used the generation best individual as a reference point. All other individuals in the current population were compared against the generation best individual for measuring the distance.

Figure 5.6, and Figure 5.7 shows the fitness, genotypic and phenotypic diversity of the best solutions obtained from the OX and the PMX operator for the real 2016 and 2017 datasets. Each row shows the population diversity using the proposed crossover operator for respective datasets; column (a) shows the genotypic diversity, and column (b) shows the phenotypic diversity. For each subfigure, the x axis represents the generation number, the left y axis represents the edit distance, and the right y axis represents the fitness (profit) in thousands of US dollars. The top red line shows the best individual fitness for the current generation, and the lower blue line represents the average population fitness for the current generation. The box plots show the edit distance statistics of the individuals after every 50 generation. GA uses 50 mixing combinations in the chromosome, which is why the maximum edit distance between two individuals would be 150 if all the combinations were

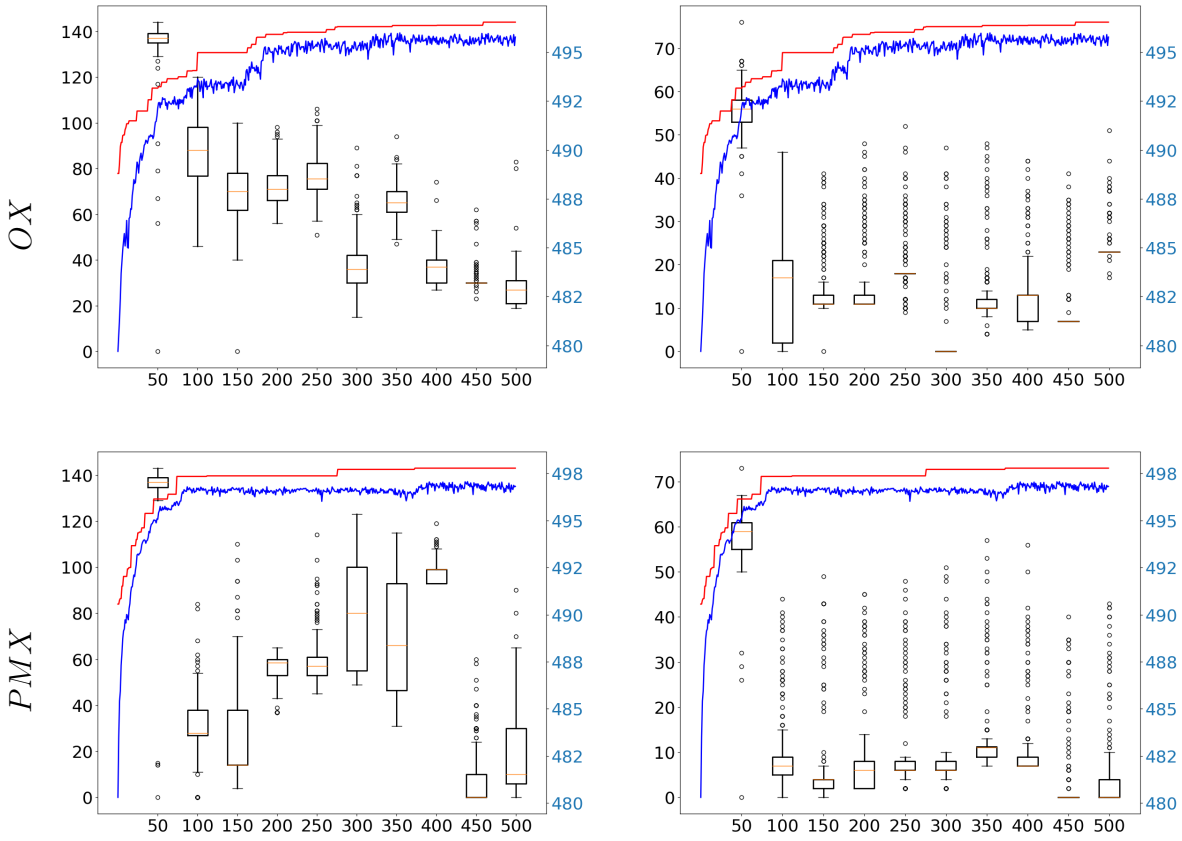


(a) Genotypic diversity (b) Phenotypic diversity

Figure 5.6: Population diversity of the OX and PMX operator in R_2016 dataset

different (since each combination is encoded by a three-character value) for the genotypic conversion. For the phenotypic diversity, the actual mixing combinations used to load the trucks range from approximately 18 to 25, and the maximum distance could be 75 or higher.

Although the average fitness curve in the diversity plot seems to converge on the generation best, the edit distance box plots show that there is diversity both in the chromosome representation (genotypic) and actual (phenotypic) solutions. For the edit distance, we observe a high variance at the beginning, and as the evolution progress, more individuals in the population tend to align with the generation best. For both datasets, the



(a) Genotypic diversity (b) Phenotypic diversity

Figure 5.7: Population diversity of the OX and PMX operator in R_2017 dataset

genotypic and the phenotypic diversity of the PMX operator seems to control the population diversity slightly better than the OX operator. This phenomenon might explain why the PMX provided a better solution for all test cases compared to OX.

In conclusion, the proposed GA approach was able to find solutions with increased overall profit for both real and artificial datasets. Moreover, for all cases, the increased profit obtained from GA variants is sufficient to invest in the protein tracking device. The experimental results suggest that the added complexity of the proposed GA approach is beneficial to improve the overall profitability compared to the baseline approaches.

CHAPTER SIX

DIFFERENTIAL EVOLUTION GRAIN MIXING

We adapted Differential Evolution (DE) as the second evolutionary approach for solving the grain mixing problem. The permutation-based representation for DE is similar to the GA approach; however, the main difference lies in the computational steps and the search mechanism. DE iteratively tries to improve the candidate solutions based on the fitness function using the recombination operators that are different compared to GA. In this chapter, we present our specific implementation of the DE algorithm for solving the grain mixing problem, the experimental results, comparative analysis of DE with GA and the deterministic approaches along with a detailed analysis of the results.

6.1 Encoding, Initial Population, and Fitness

The encoding, initial population, and fitness function used in DE are the same as in GA (see Chapter 5). The main differences lie in the 'Mutation', 'Crossover', and 'Selection' operators. Next, we discuss our specific implementation of these operators in the context of DE.

6.2 Mutation

For this study, we used two differential mutation operators. The first one is an adaptation of Relative Position Indexing (RPI) as described in Section 2.3.4.1. The chromosome in an individual consists of a subset of the total mixing combinations, and two individuals may contain different mixing combinations in the chromosome. Therefore, we need to adapt the RPI operator to fit our problem description. To illustrate the adapted RPI, let us consider three donor vectors x_{r1}, x_{r2}, and x_{r3}. Assume there are a total of 10

mixing combinations, and the vector length of each donor vector is 5 (i.e., randomly take 5 combinations out of 10). The donor vector's representation is as follows (note that the bin pair and mixing ratio are excluded for simplicity):

$$x_{r1} = \begin{bmatrix} 9 & 4 & 6 & 1 & 3 \end{bmatrix}$$
$$x_{r2} = \begin{bmatrix} 7 & 10 & 2 & 4 & 8 \end{bmatrix}$$
$$x_{r3} = \begin{bmatrix} 3 & 10 & 9 & 1 & 4 \end{bmatrix}$$

From these, we create a union vector that includes all of the different combination IDs from the three donor vectors. We need the union vector to compute the relative indexing of the mutant vector as the donor vector does not represent exact permutation. The union vector is computed as follows:

$$\begin{aligned} U &= x_{r1} \cup x_{r2} \cup x_{r3} \\ &= \begin{bmatrix} 1 & 2 & 3 & 4 & 6 & 7 & 8 & 9 & 10 \end{bmatrix} \end{aligned}$$

The next step is to transform all of the integer values to floating-point values by dividing each vector element by the largest integer in the vector list and then mapping these values back onto the original donor vectors as follows:

$$U(f) = \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 & 0.6 & 0.7 & 0.8 & 0.9 & 1.0 \end{bmatrix}$$
$$x(f)_{r1} = \begin{bmatrix} 1.0 & 0.44 & 0.67 & 0.11 & 0.33 \end{bmatrix}$$
$$x(f)_{r2} = \begin{bmatrix} 0.7 & 1.0 & 0.2 & 0.4 & 0.8 \end{bmatrix}$$
$$x(f)_{r3} = \begin{bmatrix} 0.3 & 1.0 & 0.9 & 0.1 & 0.4 \end{bmatrix}$$

Next, the standard DE mutation operator is applied to the real-valued donor vectors to obtain the real-valued mutant vector (suppose $F = 0.5$) as follows:

$$\begin{aligned} v(f) &= x(f)^g_{r1} + F \times (x(f)^g_{r2} - x(f)^g_{r3}) \\ &= \begin{bmatrix} 1.2 & 0.44 & 0.32 & 0.26 & 0.53 \end{bmatrix} \end{aligned}$$

The final step is to convert the floating mutant vector to an integer-valued mutant vector. To do that, we use the floating union vector. Each value in $v(f)$ is compared against the values in $U(f)$ and for the $U(f)$ value for which the distance is the minimum, the corresponding integer combination ID from U is selected to replace the floating-point value. The ID which has been assigned to the mutant vector is removed from the list U and $U(f)$ to avoid any duplicate entry. The procedure is illustrated as follows:

$$\begin{aligned} U &= \begin{bmatrix} 1 & 2 & 3 & 4 & 6 & 7 & 8 & 9 & 10 \end{bmatrix} \\ U(f) &= \begin{bmatrix} 0.1 & 0.2 & 0.3 & 0.4 & 0.6 & 0.7 & 0.8 & 0.9 & 1.0 \end{bmatrix} \\ v(f) &= \begin{bmatrix} 1.2 & 0.44 & 0.32 & 0.26 & 0.53 \end{bmatrix} \\ v &= \begin{bmatrix} 10 & 4 & 3 & 2 & 6 \end{bmatrix} \end{aligned}$$

The second differential mutation operator is the Generation Best Perturbation (GBP) as described in Section 2.3.4.2. Our proposed GBP operator works as follows. Two random individuals along with the generation best individual are selected from the target population (all mutually exclusive), and the differential variation is achieved by deleting some genes from the generation best individual and inserting new genes from the two random individuals depending on the mutation factor. The mutation factor controls the rate of perturbation of genes in the best individual to create the mutant vector. The modified GBP operator can

be represented as follows:

$$v_{i,j}^{g} = \begin{cases} x_{best,k}^{g} & \text{if } 0 \leq q_j \leq F \\ x_{r2,l}^{g} & \text{if } F \leq q_j \leq \left(F + \frac{(1-F)}{2}\right) \\ x_{r3,m}^{g} & \text{if } \left(F + \frac{(1-F)}{2}\right) \leq q_j \leq 1 \end{cases}$$

where, q_i is a random number between $(0, 1)$ and $F \in [0, 1]$ is the mutation factor.

The resulting mutation can be thought of as a probability measure. For an F value of 0.5, the mutant vector has a 50% probability to copy a gene from x_best and a 25% probability to copy a gene from donor vector x_{r2} and a 25% probability, to copy a gene from the donor vector x_{r2}. If the gene is already present in the mutant vector, it moves to the next index until it finds a valid gene entry.

For example, let us consider the same donor vectors that we used in the RPI example. With a mutation factor of 0.5 and $q \sim U(0, 1)$, the mutation vector v can be obtained as follows:

$$\begin{aligned} q &= \begin{bmatrix} 0.57 & 0.19 & 0.32 & 0.84 & 0.07 \end{bmatrix} \\ x_{best} &= \begin{bmatrix} 9 & 4 & 6 & 1 & 3 \end{bmatrix} \\ x_{r2} &= \begin{bmatrix} 7 & 10 & 2 & 4 & 8 \end{bmatrix} \\ x_{r3} &= \begin{bmatrix} 3 & 10 & 9 & 1 & 4 \end{bmatrix} \\ v &= \begin{bmatrix} 7 & 9 & 4 & 3 & 6 \end{bmatrix} \end{aligned}$$

6.3 Crossover

After creating the mutant vector, binomial crossover is performed between the target vector x_i^g and the mutant vector v_i^g to create the trial vector u_i^g. The crossover operation is similar to the standard DE crossover, however, in our implementation, the trial vector first copies the chromosome of the mutant vector $v_{i,g}$ and replaces the gene with the target vector $x_{i,g}$ only if it is not already present in the trial vector and the crossover condition ($rand(0,1) \leq CR$ or $j = j_{rand}$) condition is satisfied.

6.4 Selection

The selection operator is the same as standard DE. It selects the better individual by comparing the fitness of the trial vector $u_{i,g}$ and the target vector $x_{i,g}$. If the trial vector has a higher fitness value than the target vector, the target vector is replaced by the trial vector for the next generation. Otherwise, the trial vector is discarded and the target vector remains in the population for the next generation.

6.5 Experimental Design

Similar to the other grain mixing approaches, we used the two real datasets along with the twelve additional datasets to evaluate the performance of the DE algorithm. The dataset descriptions are given in Section 3.4. The hyperparameters along with the chromosome size used in DE were tuned manually. Similar to the GA approach, we followed a grid search to determine the best set of parameters. We used a set of $\{30, 50, 100, 200, 500\}$ for the population size, a set of $\{0.3, 0.4, 0.5, 0.6\}$ for the mutation probability, and a set of $\{0.6, 0.7, 0.8, 0.9\}$ for the crossover rate. For the chromosome size, we used subset size (i.e., the number of mixing combinations) ranges from 50 to 100 mixing combinations with an

Table 6.1: Parameter settings for the DE algorithm

#MaxIter	**Population**	**#MC**	**CR**	**Mutation**
500	100	100	0.9	0.5

increment of 10.

We performed the grid search on the real datasets to evaluate the solution quality and selected the hyperparameters that provided the highest average profit. Table 6.1 shows the selected parameter values used in the DE algorithm for all of the test cases. Column *#MaxIter* represents the maximum generation number, column *#MC* represents the number of mixing combinations in the chromosome list (chromosome size), and *CR* represents the crossover rate.

The solution obtained from DE is compared against the GA approach to evaluate if there is any significant profit increase using the DE search technique. We also consider the solution obtained from *Random* search and other deterministic approaches (i.e., *Greedy* and *NoMix*) to assess DE's overall solution quality.

DE is also a stochastic algorithm similar to GA and baseline *Random* search algorithm. Therefore, we ran 10 experiments for each dataset and recorded the average overall profit to get a fair comparison with the other stochastic approaches.

6.6 Results and Analysis

6.6.1 Profitability Analysis

Table 6.2 shows the performance of all of the approaches (i.e., stochastic and deterministic) that we used for solving the grain mixing problem studied in this thesis. The column *Dataset* identifies which data was used by the algorithms to generate the results. The notation *R_year* indicates the real dataset for the respective year. *RF_year* indicates

the flipped market version of the real datasets, and A^*_year indicates the artificial datasets for each of the respective years. The values in the table show the overall profit obtained by each algorithm, expressed in thousands of US dollars. For DE, we adapted two differential mutation operators: RPI and GBP, and for GA we have two versions with the adapted OX and PMX crossover operators. The overall profit from the stochastic algorithms are the mean and standard deviation over 10 runs of experiments; the profit did not change for the deterministic algorithms (i.e., *Greedy* and *NoMix*) so they were run only once. The **bold** values represent the maximum profit obtained across the algorithms for a respective dataset, and the underline values represent a sufficient profit increase from the base *NoMix* solution (which is more than $5000 in our case).

Based on different algorithms' performance, it is evident that the evolutionary approaches lead to a higher overall profit compared to the *NoMix* solution in every case. For all datasets, DE's GBP operator along with the GA's OX and PMX operators consistently provided a sufficient profit increase compared to the baseline *Random* search and no mixing solutions. However, for DE, the RPI operator did not perform as well as the GBP operator. Although RPI provided a sufficient profit increase for most of the datasets (except for two), the profit from RPI is consistently lower than the other evolutionary operators for most of the test cases. And, comparing all evolutionary approaches, DE with the GBP operator performed best for the real and flipped datasets, whereas the GA with the PMX operator performed best for all the artificial datasets.

The profit obtained from the DE's RPI operator was still able to provide a better profit than the *Random* search and the *Greedy* approach for all test cases. A possible reason for why DE's RPI operator consistently provided a lower profit than the GBP operator might be in the core design of the RPI operator. The RPI operator is designed to function similarly to the original DE mutation for solving the continuous search space problem. For strict permutation problems (e.g., TSP), converting the fractional values to the integer domain may

Table 6.2: Performance of the different algorithms (*profit* in thousands of dollars)

Dataset	**DE**		**GA**		**Random**	**Greedy**	**NoMix**
	RPI	GBP	OX	PMX			
R_2016	430.9 ±0.48	**436.0** ±0.50	435.3 ±1.04	435.5 ±1.53	427.5 ±0.86	430.5	424.5
R_2017	494.2 ±0.77	**497.1** ±0.74	495.5 ±0.59	496.1 ±0.98	489.7 ±0.53	491.6	487.1
RF_2016	502.8 ±0.38	**506.2** ±0.37	505.3 ±1.03	506.0 ±0.54	500.9 ±0.63	499.9	499.7
RF_2017	424.0 ±0.96	**429.1** ±0.67	427.6 ±1.36	428.7 ±1.34	420.6 ±0.60	423.1	418.0
A1_2016	411.2 ±1.02	417.1 ±1.34	416.6 ±1.39	**418.2** ±2.08	405.6 ±1.21	394.5	395.4
A2_2016	558.0 ±1.32	562.7 ±2.04	562.2 ±2.44	**563.9** ±1.85	548.9 ±2.24	533.1	545.2
A3_2016	583.1 ±1.90	588.7 ±1.73	589.8 ±2.63	**592.6** ±1.91	568.3 ±3.39	543.9	569.9
A4_2016	566.9 ±1.96	572.8 ±1.98	575.3 ±1.60	**576.6** ±1.91	556.4 ±1.92	520.4	539.9
A5_2016	527.3 ±1.13	533.8 ±1.98	534.9 ±1.59	**536.4** ±1.53	512.7 ±2.51	496.9	512.2
A1_2017	610.2 ±2.43	620.8 ±3.31	621.8 ±3.38	**623.9** ±2.37	595.8 ±2.28	581.7	608.9
A2_2017	529.5 ±1.75	538.4 ±1.33	537.0 ±2.16	**539.4** ±1.98	520.1 ±1.68	515.8	520.6
A3_2017	668.4 ±1.37	676.1 ±1.58	675.3 ±1.49	**678.5** ±1.96	661.8 ±2.32	655.8	643.9
A4_2017	449.4 ±1.24	452.5 ±1.06	451.3 ±1.56	**452.7** ±0.87	442.4 ±1.16	435.6	430.1
A5_2017	677.0 ±1.11	686.2 ±2.36	684.5 ±1.58	**687.0** ±2.71	670.9 ±2.95	656.0	663.5

be an appropriate choice, however, in our case, we have a pseudo permutation representation for which the fractional conversion may be favoring a particular set of combinations.

Table 6.3: Statistical significance testing on the stochastic algorithms

	RPI	**GBP**	**OX**	**PMX**	**Random**
RPI	N/A	0, 14, 0	0, 14, 0	0, 14, 0	14, 0, 0
GBP	14, 0, 0	N/A	9, 3, 2	1, 9, 4	14, 0, 0
OX	14, 0, 0	3, 9, 2	N/A	0, 13, 1	14, 0, 0
PMX	14, 0, 0	9, 1, 4	13, 0, 1	N/A	14, 0, 0
Random	0, 14, 0	0, 14, 0	0, 14, 0	0, 14, 0	N/A

Table 6.3 shows the statistical significance testing between all of the stochastic algorithms. For the significance testing, we utilized a paired t-test [65] and used a threshold of $\alpha = 0.05$ for the p-value. The header rows and columns represent which two algorithms were compared. The values represent a 3-tuple, where the first value indicates for how many datasets the row algorithm's profit was better than the column algorithm, the second value indicates for how many datasets the column algorithm's profit was better than the row algorithm, and the third value indicates for how many datasets the difference was not statistically significant. We have a total of 14 datasets, and each 3-tuple sums up to 14.

As we can see in Table 6.3, similar to GA's crossover operators, DE's RPI and GBP operator significantly improved the profit compared to the *Random* search algorithm for all of the datasets. All of the proposed recombination operators provided a significant profit increase compared to the baseline *Random* algorithm as we hypothesized. However, we also hypothesized the four recombination operators to perform the same, which did not turn out to be true. DE's RPI operator consistently provided a lower profit compared to the other three recombination operators for all of the datasets. As mentioned earlier, the RPI operator transforms the integer permutation vectors into real-valued vectors, which for our pseudo-permutation representation may not be an appropriate choice. We presented a population

diversity analysis in Section 6.6.2 to further analyze RPI's behavior.

Comparing the performance of the GBP, PMX, and OX operators, we notice that for our experiments, GA's PMX operator performed the best, followed by DE's GBP operator. The GBP operator significantly performed better than the PMX operator for only one dataset (R_2017) with four other datasets where the difference was not statistically significant. Moreover, for nine of the datasets, GBP's performance was better than OX, and for two of the datasets, the difference was not statistically significant. Overall, DE's GBP operator's performance was competitive compared to the GA's PMX operator for our test cases.

We present the best solution obtained from the DE's RPI operator in Table 6.4, and GBP operator in Table 6.5 for the real R_2017 dataset. In these tables, each row represents a separate truck entry. The column "Pair" identifies the pair of bins mixed to load a truck, "P1_Bu" and "P2_Bu" specify the number of bushels taken from each bin in the pair, "Protein" shows the weighted average protein level, "Load" gives the total amount of wheat loaded in the truck (max 8,000 bu) and "Elv" identifies the elevator where the truck delivers the grain.

As can be seen from the tables, both best solutions were able to utilize all of the bushels. The profit obtained from the RPI solution is almost $3000 less than the GBP solutions. This means some of the bin pair combinations taken by the RPI solutions may not be the best choice for filling a truck. For example, in the RPI solution (Table 6.4), all of the trucks mixed bins from a bin pair including the bin 16. However, in the GBP solution (Table 6.5), we observe that the last three trucks were loaded with grain from bin 16 without any mixing. We observe a similar pattern in the GA's OX and PMX best solution where bin 16 was taken separately without any mixing (see Section 5.8.1, Table 5.4 and 5.5). From these observations, taking bin 16 separately seems to be a better choice to increase the profitability. This means that the best RPI individual did not have a set of bin pair combinations for which it can load grain from bin 16 separately. A possible reason may

Table 6.4: Best solution from DE(RPI) for R_2017 dataset

#	Pair	Mix	P1_Bu	P2_Bu	Protein	Load	Elv	Profit
1	(1, 2)	0.20	1600.0	6400.0	13.49	8000.0	3	41992.0
2	(15, 2)	0.77	2985.3	892.5	13.50	3877.8	3	20147.5
3	(3, 9)	0.11	800.0	6539.8	13.10	7339.8	2	34937.6
4	(1, 6)	0.10	800.0	7200.0	12.55	8000.0	2	36080.0
5	(1, 3)	0.30	2400.0	5595.3	12.59	7995.3	2	35659.0
6	(8, 12)	0.30	2400.0	5600.0	11.62	8000.0	2	34632.0
7	(1, 4)	0.20	1600.0	6400.0	11.54	8000.0	2	34552.0
8	(1, 4)	0.65	4000.0	2125.5	11.98	6125.5	2	26456.0
9	(1, 5)	0.58	4436.8	3200.0	11.72	7636.8	2	32914.6
10	(10, 11)	0.50	4000.0	4000.0	11.25	8000.0	1	33912.0
11	(14, 7)	0.77	5600.0	1712.7	11.72	7312.7	2	30932.6
12	(16, 12)	0.76	4800.0	1489.7	11.55	6289.7	2	26605.3
13	(16, 6)	0.91	4800.0	503.7	11.51	5303.7	2	22381.5
14	(14, 5)	0.55	3057.4	2513.4	11.57	5570.8	2	23508.6
15	(16, 11)	0.63	1600.0	921.3	11.76	2521.3	1	10172.1
16	(16, 10)	0.74	2309.0	800.0	11.07	3109.0	1	12710.9
17	(13, 10)	0.68	5050.0	2400.0	10.60	7450.0	1	30470.5
18	(8, 10)	0.95	1792.7	92.5	10.69	1885.2	1	7405.1
Total						**112417.5**		**495469.3**

be the DE with the RPI operator was converging slowly. We considered a maximum of 500 iterations for each evolutionary algorithm and maybe with more iterations, the RPI operator would eventually find a solution similar to the other operators.

Table 6.5: Best solution from DE(GBP) for R_2017 dataset

#	Pair	Mix	P1_Bu	P2_Bu	Protein	Load	Elv	Profit
1	(6, 2)	0.30	2400.0	5600.0	13.42	8000.0	3	41920.0
2	(15, 2)	0.64	2985.3	1692.5	13.54	4677.8	3	24324.6
3	(9, 12)	0.89	6539.8	800.0	13.03	7339.8	2	35378.0
4	(3, 12)	0.80	6395.3	1600.0	12.57	7995.3	2	36538.5
5	(1, 6)	0.23	1600.0	5303.7	12.52	6903.7	2	31135.6
6	(11, 7)	0.74	4800.0	1712.7	11.85	6512.7	2	28134.7
7	(10, 12)	0.25	1600.0	4689.7	11.54	6289.7	2	27171.4
8	(1, 4)	0.40	3200.0	4800.0	11.73	8000.0	2	34552.0
9	(1, 4)	0.39	2400.0	3725.5	11.72	6125.5	2	26456.0
10	(1, 5)	0.60	4800.0	3200.0	11.74	8000.0	2	34480.0
11	(1, 5)	0.53	2836.8	2513.4	11.64	5350.2	2	23059.2
12	(10, 11)	0.97	4000.0	121.3	10.18	4121.3	1	16646.0
13	(14, 13)	0.70	5600.0	2400.0	11.74	8000.0	2	33760.0
14	(14, 13)	0.54	3057.4	2650.0	11.53	5707.4	2	24085.2
15	(8, 10)	0.71	4192.7	1692.5	10.54	5885.2	1	24305.8
16	(16, 2)	0.00	4800.0	0.0	11.40	4800.0	1	20016.0
17	(16, 12)	0.00	4000.0	0.0	11.40	4000.0	1	16680.0
18	(3, 16)	0.00	0.0	4709.0	11.40	4709.0	1	19636.5
Total						**112417.5**		**498279.5**

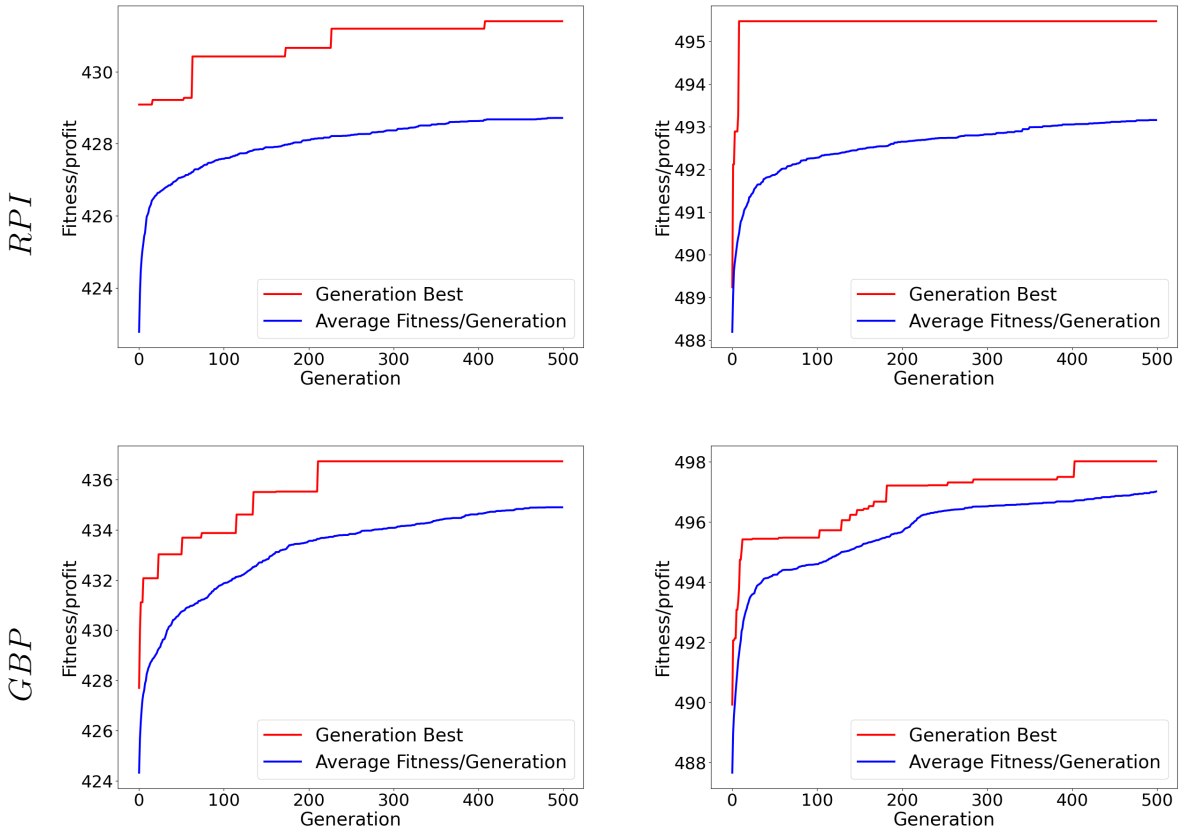
6.6.2 Population Diversity

One of the questions to be addressed is whether or not the DE variants are converging prematurely. To address this, we completed an analysis of the fitness diversity in the

population along with the genotypic and phenotypic diversity for both DE variants.

6.6.2.1 Fitness Diversity Similar to the GA fitness diversity approach, we tracked the best individual fitness (profit) and the average fitness of the population in each generation to assess the fitness diversity for the DE variants. We show the fitness diversity plot for the best solution obtained by DE's RPI and GBP operator for the real 2016 and 2017 datasets in Figure 6.1. The y axis in the subfigures specify profit in thousands of US dollars. As can be seen, unlike the GA fitness curve, the average fitness curve in DE is a smooth increasing curve portraying the search mechanism difference between these two algorithms. In DE, an individual in the current population is only replaced by a better-fit individual created by the mutation and crossover operator. Therefore, as evolution progress, more fit individuals appear in the population increasing (or may remain the same in case of loss of diversity) the average population fitness. Also, comparing the fitness curve of the RPI and GBP operator for both datasets, we notice a huge gap between the generation best and the average fitness curve for the RPI operator indicating a slow convergence rate. However, the average fitness curve of the GBP operator seems to close the gap with the generation best curve which may explain the better performance of the GBP operator.

6.6.2.2 Genotypic and Phenotypic Diversity We followed a similar approach like GA to measure the genotypic and phenotypic diversity in the DE variants. To do that, we had to convert the set of mixing combinations in an individual into a series of strings and used edit distance measures to compute how different each individual is. We encoded each mixing combination with a 3 character symbol. For genotypic diversity, we used the mixing combinations in the chromosome, and for phenotypic diversity, we used the mixing combinations used to generate the final solution. As comparing edit distance between all pairs of the individual is computationally expensive, we consider the generation best individual as the reference point. For a detailed description of the string conversion and edit distance

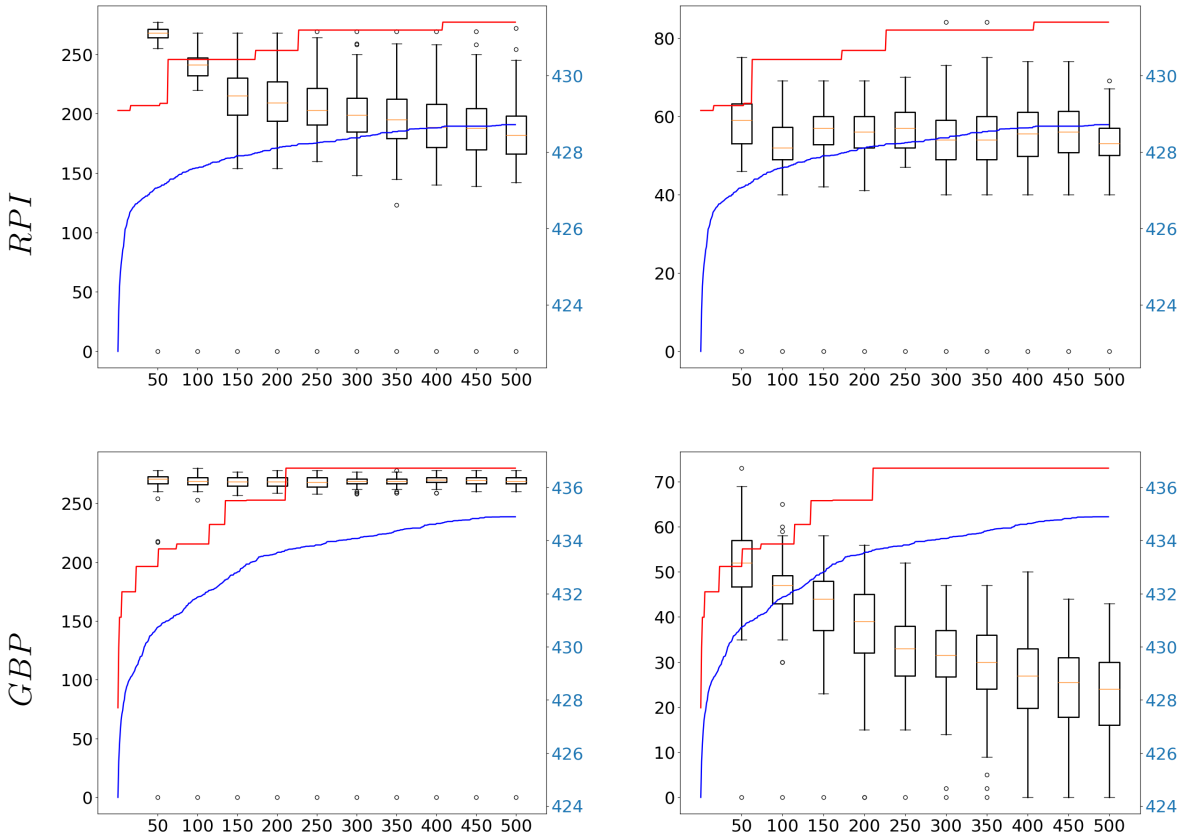


a) R_2016 b) R_2017

Figure 6.1: Fitness Diversity of the RPI and GBP operator

measure, please see Section 5.8.2.2.

Figure 6.2, and Figure 6.3 shows the fitness, genotypic, and phenotypic diversity of the best solutions obtained by the RPI and the GBP operator for the year 2016 and 2017 respectively. Column (a) shows the genotypic diversity, and column (b) shows the phenotypic diversity. For each subfigure, the x axis represents the generation number, the left y axis represents the edit distance, and the right y axis represents the fitness (profit) in thousands of US dollars. The top red line shows the best individual fitness for the current generation, and the lower blue line represents the average population fitness for the current generation. The box plots show the edit distance statistics of the individuals after every 50 generation. The

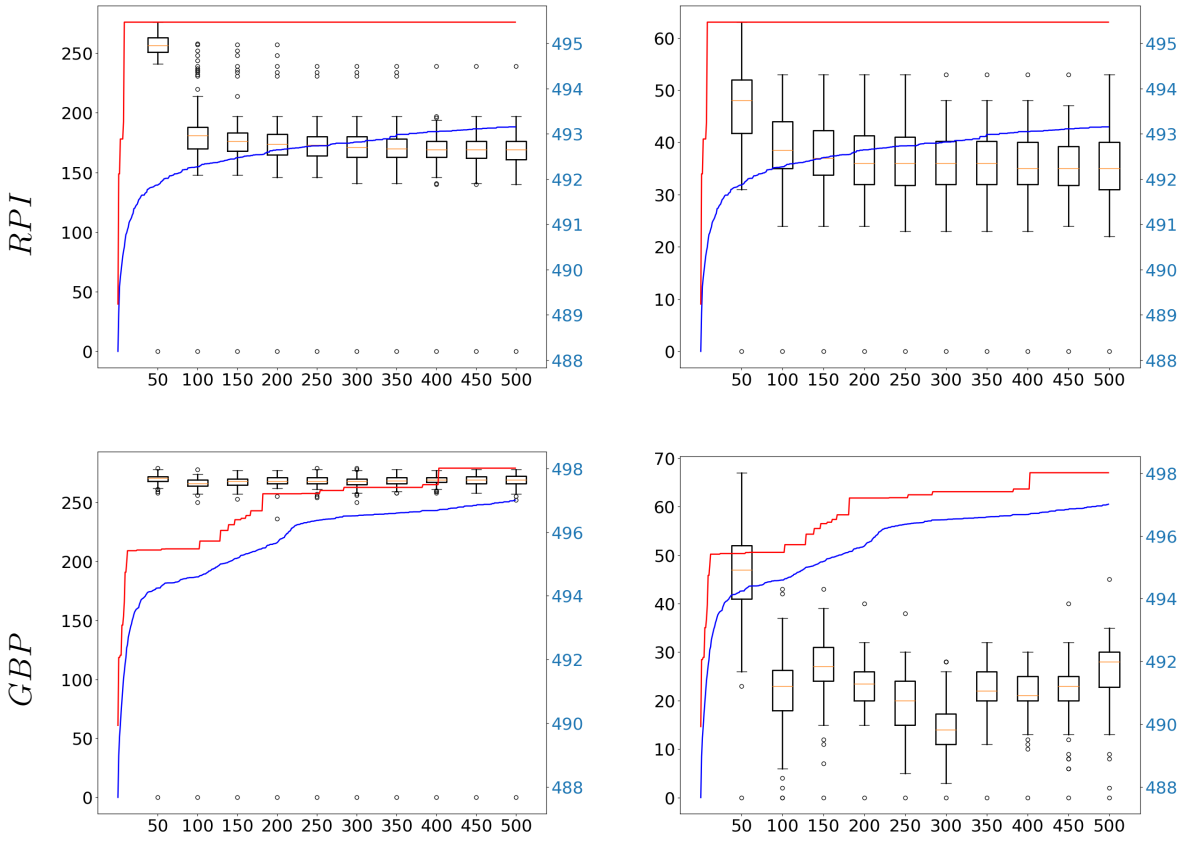


(a) Genotypic diversity (b) Phenotypic diversity

Figure 6.2: Population diversity of the RPI and GBP operator in R_2016 dataset

chromosome size used in DE is 100. Therefore, for the genotypic conversion, the maximum edit distance could be 300 if all of the combinations in the chromosome are different. For the phenotypic conversion, the actual solution consists of 18 to 25 mixing combinations, and the maximum edit distance maybe 75 or higher.

For both Figure 6.2, and Figure 6.3, we observe a high edit distance in the genotypic diversity for both the RPI and GBP operator (with GBP being the highest). However, for the GA operators genotypic diversity, we notice that as generation increases, the edit distance of the individuals (i.e., chromosomes) tends to match with the generation best



(a) Genotypic diversity (b) Phenotypic diversity

Figure 6.3: Population diversity of the RPI and GBP operator in R_2017 dataset

individual. This also shows the differences in the recombination operators for DE and GA. The operators in GA copy a subset of the genes of parents to produce offspring maintaining the order. Therefore, as evolution progresses, more individuals turn out to be similar in their chromosome structure. However, in our DE implementation, the mutation operators perturb the genes of the best individuals in random orders with genes from other individuals to create the mutant vectors. Therefore, even if the same mixing combination exists in the individual, the order in which they appear is always different providing high edit distance values.

However, for the phenotypic diversity, we sort the mixing combinations with respect to their profit to generate the final solutions. Therefore, in this case, we may expect to find more individual solutions that are closed to the generation's best solution as generation progresses. For the RPI, we notice high phenotypic diversity for both datasets. This also indicates the slow convergence rate of the RPI operator. However, for the GBP operator, we noticed more individuals with slightly different solution quality (i.e., mixing combinations) emerge as evolution progresses for both datasets. Comparing the phenotypic diversity between the proposed GA and DE approach, DE seems to maintain diversity better than GA.

In conclusion, the adapted DE approach with the modified mutation operators was able to provide increased profit compared to the baseline approaches. For DE, the adapted GBP operator provided the highest overall profit for the real datasets and showed competitive solutions compared to GA's PMX operator for the artificial datasets. The experimental results confirmed that the evolutionary approaches were consistent in providing a sufficient profit increase for all datasets depicting the benefit of grain mixing.

CHAPTER SEVEN

CONCLUSION

7.1 Discussion

Combinatorial optimization is a class of mathematical optimization that tries to find an optimal value from a finite set of values. It has many theoretical and practical applications arising in different disciplines such as computer science, operation research, business, and economics. Usually, combinatorial optimization problems (COPs) are modeled in terms of variables, constraints, and an objective function. And for COPs, some or all variables may be restricted to contain only integer values. Restricting variables to contain only integer values make COPs even harder to solve exactly using Linear Programming (LP) models. Mixed Integer Linear Programming (MIP) is a widely used technique to tackle many real-world COPs. However, MILP relies on suitable linear approximations of the non-linear objective functions and constraints which is often hard to come up with for some problems. Evolutionary approaches (EA) have also been used widely in the literature for solving COPs. The benefit of using EA over MIP is that the former does not put any restriction on how the objective function is defined. Therefore, for problems with non-linearity in the objectives and constraints, EAs are often the most suitable approach.

In this thesis, we considered a combinatorial optimization problem in the wheat supply chain referred to as grain mixing. Wheat protein content plays a critical role in determining the end-use functionality and the commercial value of wheat. Therefore, in some states, the price of a bushel of wheat depends on its protein level as measured at the elevator. Due to the presence of several environmental factors, it is quite difficult to get a uniform protein content across the fields. Consequently, the farmers end up with varying protein content grain in their storage bins. One possible approach to improve the profitability of the farmers

when selling wheat to local grain elevators is to apply some grain mixing plan that will change the average protein content in a truck. However, coming up with a quality mixing plan is not trivial as many real-world constraints come into play. Moreover, the farmers also need to invest in the protein tracking infrastructure to do grain mixing in the first place.

We have considered several approaches for determining how to maximize the profit of wheat production by mixing different numbers of bushels to change the protein level contained in a truck being delivered to an elevator. The first question that arises is how hard the grain mixing problem is for determining the optimal grain mixing plan. To answer that question, we provided a formal mathematical representation in Chapter 3 of the grain mixing problem. Due to the presence of non-linearity in the objective function and some constraints, we explained why the problem cannot be solved exactly using LP or MILP solvers. To further emphasize the hardness of the problem, we provided the NP-Hardness of the grain mixing problem following a reduction from the 3-Dimensional Matching problem, thus justifying the use of approximate algorithms for finding optimal or near-optimal solutions.

For the approximation algorithms, we utilized two evolutionary approaches: Genetic Algorithm (GA) and Differential Evolution (DE) for solving the grain mixing problem. To evaluate the performance of the evolutionary approaches, we also introduced three baseline methods in Chapter 4. The first one is the *NoMix* algorithm that provides the baseline wheat selling profit without any grain mixing. Farmers without the protein tracking infrastructure end up getting a profit close to this. This method helps us assess if grain mixing is beneficial to improve the overall profit. Then we presented the deterministic *GreedyMix* approach which greedily selects the bin pair combinations that provide the highest profit for a fully loaded truck. Due to the varying bushel content in the storage bins, it is not always possible to load each truck fully which is why *GreedyMix* might fail to provide an optimal mixing plan. The third baseline approach was the stochastic *Random* search that randomly chooses a subset of bin-pair and mixing combinations and calculates profit based on the selected

combinations. It does this several times and returns the result with the best solution.

We then showed the profitability analysis of the *GreedyMix* and *Random* search algorithms compared to the baseline no mixing profit on both real and artificial datasets to evaluate the solution quality. The expectation for the *GreedyMix* did not exactly match our hypothesis. Although *greedy* mixing provided an increased profit for the real datasets, for some artificial datasets it failed to provide a profit better than no mixing. And, for the real datasets, it often failed to provide a sufficient profit increase that would be necessary to invest in the supporting protein tracking infrastructure. On the other hand, the *Random* search algorithm provided increased profit for most of the datasets except for a few. Overall, these two methods demonstrated the potential of grain mixing for improving overall profitability.

In Chapter 5 and 6, we presented our adapted GA and DE to address the grain mixing problem. For these approaches, we formulate the problem as a permutation-based combinatorial optimization problem. Our proposed permutation-based representation only contains a subset of the permutations. Therefore, adapted two permutation-based crossover operators: OX and PMX, as alternate crossover operators for GA. Similarly, we adapted a discrete mutation operator RPI and introduced another mutation operator GBP for DE. The experimental results from both real and simulated datasets showed that the evolutionary algorithms with adapted recombination operators consistently provided an increased profit compared to no mixing, *greedy* mixing, and *random* search profits for all of the datasets.

The experimental results also showed that the evolutionary algorithms for all datasets came up with solutions with sufficiently increased profit over just doing a random search. The solution quality from these evolutionary approaches matched our hypothesis, and for all of the test datasets, we observe solutions with increased profit that were sufficient to invest in the protein tracking infrastructure. The results with the simulated datasets also confirmed that mixing grain in our experiments always improves the overall wheat selling profit, and the adapted operators are efficiently exploring the search space for improving the

solution quality.

We then presented a detailed performance analysis for each of the adapted crossover operators for GA and mutation operators for DE. The experimental results on all datasets showed that DE with the GBP operator provided the best solutions for the real datasets, and GA with the PMX operator provided the best solutions for the artificial datasets on average. GA with the OX operator did provide a sufficient profit increase for all the datasets, but the profit was always slightly lower than the PMX solution. In our GA population diversity analysis, we showed that the PMX operator maintained the diversity slightly better than the OX which may explain this behavior. On the other hand, DE with the RPI operator performed the worst comparing all other recombination operators. Although it did improve the profit from the baseline, it consistently provided a lower profit from all other recombination operators for all datasets.

In conclusion, the evolutionary approaches with the adapted recombination operators were always able to provide solutions better than the baseline approaches in our experiments. For three out of the four recombination operators, they found solutions with sufficient profit increase that needed to invest in the protein tracking infrastructure. Therefore, farmers should consider grain mixing infrastructure to improve their overall profitability when selling wheat to the nearest elevators.

Finally, to summarize our contributions, first, we provided a mathematical formulation for the grain mixing problem. The formulation reflects the non-linearity in the objective function and some constraints which makes the problem harder to solve exactly using LP or MILP solvers. Second, we proved a theorem to show that the grain mixing problem is NP-Hard. The theorem justifies the use of approximation algorithms such as GA and DE to find a quality mixing plan in a decent amount of time. For the evolutionary approaches, we followed a pseudo-permutation-based encoding for the problem representation. Therefore, as our third contribution, we adapted two crossover operators for GA and two mutation operators for

DE. The experimental results demonstrate that the adapted recombination operators were always able to find better solutions compared to the baseline grain mixing approaches and may be beneficial for other COPs with similar permutation-based representation.

7.2 Future Work

For this thesis, we made certain assumptions on the grain mixing problem. One of them was to limit the mixing to only two bins at a time for the problem representation. This was based on limits given to us by the farmers. For future work, a possible direction could be to consider if mixing more than two bins would provide a better profit or not, thereby justifying to the farmers the more complicated process of mixing from multiple bins.

For the grain mixing datasets, we were not able to collect the production or harvesting cost associated with the wheat selling process that would be necessary to provide a complete estimate of the profit. Therefore, a more realistic cost model that includes the production cost associated with harvesting, the cost of the protein tracking device, and supporting infrastructure could be considered as future work.

For the NP-Hardness proof from the 3-dimensional matching problem, we had to place an additional capacity constraint for the elevators which is not present in the actual grain mixing problem. Although the proof provides us an idea of the hardness of the general mixing problem, for future work, we would like to consider alternate reductions that would exclude the elevator capacity constraint.

It was also our intent to explore the suitability of comparing the proposed approaches to a relaxed MILP model; however, the inherent nonlinearity constraints of the problem would seem to suggest optimization quality would be limited. We also left this as possible future work. Moreover, an alternate representation of the grain mixing problem for the evolutionary approaches, and using other combinatorial optimization techniques such as swarm intelligence-based algorithms could also be considered as possible future work.

Finally, we proposed an adaptation of the two crossover operators for GA and two mutation operators for DE for solving the grain mixing problem. For possible future work, it would be interesting to explore whether the proposed adaptation is suitable for other permutation-based problems that do not follow a strict permutation rule.

REFERENCES CITED

[1] David B. Adams, Layne T. Watson, Zafer Gürdal, and Christine M. Anderson-Cook. Genetic algorithm optimization and blending of composite laminates by locally reducing laminate thickness. *Advances in Engineering Software*, 35(1):35–43, 2004.

[2] Nasrin Asgari, Reza Zanjirani Farahani, Hannaneh Rashidi-Bajgan, and Mohsen S. Sajadieh. Developing model-based software to optimise wheat storage and transportation: A real-world application. *Applied Soft Computing*, 13(2):1074–1084, 2013.

[3] J. Ashayeri, A.G.M. van Eijs, and P. Nederstigt. Blending modelling in a process manufacturing: A case study. *European Journal of Operational Research*, 72(3):460–468, 1994.

[4] Senthold Asseng and Stephen P. Milroy. Simulation of environmental and genetic effects on grain protein concentration in wheat. *European Journal of Agronomy*, 25(2):119–128, 2006.

[5] James E. Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, page 101–111, 1985.

[6] Bilge Bilgen and Irem Ozkarahan. A mixed-integer linear programming model for bulk grain blending and shipping. *International Journal of Production Economics*, 107(2):555–571, 2007.

[7] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, 2003.

[8] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, 2003.

[9] Jennifer J. Bond. Wheat sector at a glance. U.S. Department of Agriculture. `https://www.ers.usda.gov/topics/crops/wheat/wheat-sector-at-a-glance`. Online; accessed 11/24/21.

[10] Josu Ceberio, Ekhine Irurozki, Alexander Mendiburu, and JoseA Lozano. A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems. *Progress in Artificial Intelligence*, 1:103–117, 04 2012.

[11] Xiao Chen and Ning Wang. Optimization of short-time gasoline blending scheduling problem with a dna based hybrid genetic algorithm. *Chemical Engineering and Processing: Process Intensification*, 49(10):1076–1083, 2010.

[12] Yves Crama. Combinatorial optimization models for production scheduling in automated manufacturing systems. *European Journal of Operational Research*, 99(1):136–153, 1997.

[13] George B. Dantzig. Formulation a linear programming model. In *Linear programming and extensions*, pages 42–50. Princeton Univ. Press, Princeton, NJ, 1963.

[14] Swagatam Das and Ponnuthurai Nagaratnam Suganthan. Differential evolution: A survey of the state-of-the-art. *IEEE Transactions on Evolutionary Computation*, 15(1):4–31, 2011.

[15] L Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.

[16] Lawrence Davis. Applying adaptive algorithms to epistatic domains. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 1*, page 162–164. Morgan Kaufmann Publishers Inc., 1985.

[17] Franklin Djeumou Fomeni. A multi-objective optimization approach for the blending problem in the tea industry. *International Journal of Production Economics*, 205:179–192, 2018.

[18] M E Dyer and A M Frieze. Planar 3DM is *NP*-Complete. *J. Algorithms*, 7(2):174–184, 1986.

[19] A. E. Eiben and J. E. Smith. Genetic algorithms. In *Introduction to Evolutionary Computing*, pages 37–69. Springer Berlin Heidelberg, 2003.

[20] Kaan Erarslan, H. Aykul, Hamdi Akçakoca, and N. Çetin. Optimum blending of coal by linear programming for the power plant at seyitömer coal mine. In *17th International Mining Congress and Exhibition of Turkey*, 2001.

[21] R. Z. Farahani, N. Asgari, H. Hojabri, and A. A. Jaafari. Optimizing wheat storage and transportation system using a mixed integer programming model and genetic algorithm: A case study. In *IEEE International Conference on Industrial Engineering and Engineering Management*, pages 2109–2113, 2009.

[22] Mohamed G. Farghaly, Mahrous Ali, and Jong-Gwan Kim. Optimization of blending and production processes considering origin mines and metallurgical units using linear programming rules. *Geosystem Engineering*, 24(3):115–121, 2021.

[23] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman and Co., 1990.

[24] Mohammad Reza Gholamian and Abdul Hakim Taghanzadeh. Integrated network design of wheat supply chain: A real case of Iran. *Computers and Electronics in Agriculture*, 140:139–147, 2017.

[25] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[26] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[27] David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, volume 1, pages 69–93. Elsevier, 1991.

[28] David E. Goldberg and John H. Holland. Genetic algorithms and machine learning. *Machine Learning*, 3(2–3):95–99, 1988.

[29] David E. Goldberg and Robert Lingle. Alleles, loci and the traveling salesman problem. In *Proceedings of the 1st International Conference on Genetic Algorithms*, page 154–159. L. Erlbaum Associates Inc., 1985.

[30] John J. Grefenstette, Rajeev Gopal, Brian J. Rosmaita, and Dirk Van Gucht. Genetic algorithms for the traveling salesman problem. In *Proceedings of the 1st International Conference on Genetic Algorithms*, page 160–168, 1985.

[31] Qingxin Guo and Lixin Tang. Modelling and discrete differential evolution algorithm for order rescheduling problem in steel industry. *Computers & Industrial Engineering*, 130:586–596, 2019.

[32] Nikolas Haas. *Optimizing Wheat Blends for Customer Value Creation: A Special Case of Solvent Retention Capacity*. Masters Thesis, Department of Agricultural Economics, Kansas State University, USA, 2011.

[33] Mehmet Hayta and Unsal Cakmalki. Optimization of wheat blending to produce breadmaking flour. *Journal of Food Process Engineering*, 24:179–192, 08 2001.

[34] John H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[35] Abdollah Homaifar, Charlene X. Qi, and Steven H. Lai. Constrained optimization via genetic algorithms. *Simulation*, 62(4):242–253, 1994.

[36] James Alexander Hughes, Sheridan Houghten, and Daniel Ashlock. Permutation problems, genetic algorithms, and dynamic representations. In *Nature-Inspired Computing and Optimization: Theory and Applications*, pages 123–149. Springer International Publishing, 2017.

[37] Zhenya Jia and Marianthi Ierapetritou. Mixed-integer linear programming model for gasoline blending and distribution scheduling. *Industrial & Engineering Chemistry Research*, 42(4):825–835, 2003.

[38] Howard Karloff. *Linear Programming*. Birkhauser Boston Inc., 1991.

[39] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Springer, 1972.

[40] Daniel Kobler. *Evolutionary algorithms in combinatorial optimization*. Springer US, Boston, MA, 2009.

[41] Mark W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509, 1988.

[42] P. J. M. Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. Kluwer Academic Publishers, 1987.

[43] E.L. Lawler. *Combinatorial Optimization: Networks and Matroids*. Holt, Rinehart and Winston, 1976.

[44] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Cybernetics and Control Theory*, 10(8):707–710, February 1966.

[45] Xiang Li, Mohammad reza Bonyadi, Zbigniew Michalewicz, and Luigi Barone. A hybrid evolutionary algorithm for wheat blending problem. *The Scientific World Journal*, 2014:967254, 02 2014.

[46] Daniel Lichtblau. Relative position indexing approach. In *Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization*, pages 81–120. Springer Berlin Heidelberg, 2009.

[47] Chuan Lin, Anyong Qing, and Quanyuan Feng. A comparative study of crossover in differential evolution. *Journal of Heuristics*, 17(6):675–703, 2011.

[48] Jirí Matouek and Bernd Gärtner. *Understanding and Using Linear Programming*. Springer-Verlag, Berlin, Heidelberg, 2006.

[49] Z. Michalewicz and G. Nazhiyath. Genocop iii: a co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. In *Proceedings of IEEE International Conference on Evolutionary Computation*, volume 2, pages 647–651 vol.2, 1995.

[50] B. Miller and D. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex System*, 9:193–212, 1995.

[51] Lincoln F. L. Moro and José M. Pinto. Mixed-integer programming approach for short-term crude oil scheduling. *Industrial & Engineering Chemistry Research*, 43(1):85–94, 2004.

[52] David R. Morrison, Sheldon H. Jacobson, Jason J. Sauppe, and Edward C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016.

[53] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience, 1988.

[54] Godfrey Onwubolu and Donald Davendra. Differential evolution for permutation—based combinatorial problems. In *Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization*, pages 13–34. Springer Berlin Heidelberg, 2009.

[55] Abdoun Otman and Abouchabaka Jaafar. A comparative study of adaptive crossover operators for genetic algorithms to resolve the traveling salesman problem. *International Journal of Computer Applications*, 31(11):49–57, 2011.

[56] Quan-Ke Pan, Mehmet Fatih Tasgetiren, and Yun-Chia Liang. A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Computers & Industrial Engineering*, 55(4):795–816, 2008.

[57] Quan-Ke Pan, Ling Wang, and Bin Qian. A novel differential evolution algorithm for bi-criteria no-wait flow shop scheduling problems. *Comput. Oper. Res.*, 36(8):2498–2511, 2009.

[58] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover Publications, 1998.

[59] Yves Pochet and Laurence A. Wolsey. *Production Planning by Mixed Integer Programming*. Springer Publishing Company, Incorporated, 2010.

[60] J. Potvin. Genetic algorithms for the traveling salesman problem. *Annals of Operations Research*, 63:337–370, 1996.

[61] K.V. Price, R.N. Storn, and J.A. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization*. Springer, 2005.

[62] B. Qian, Ling Wang, R. Hu, Wan liang Wang, Dexian Huang, and Xiong Wang. A hybrid differential evolution method for permutation flow-shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 38:757–777, 2008.

[63] Anisha Radhakrishnan and G. Jeyakumar. Evolutionary algorithm for solving combinatorial optimization—a review. In *Innovations in Computer Science and Engineering*, pages 539–545. Springer Singapore, 2021.

[64] Dean Randall, Leasa Cleland, Catharine S. Kuehne, George W. B. Link, and Daniel P. Sheer. Water supply planning simulation model using mixed-integer linear programming "engine";. *Journal of Water Resources Planning and Management*, 123(2):116–124, 1997.

[65] Amanda Ross and Victor L. Willson. Paired samples t-test. In *Basic and Advanced Statistical Tests: Writing Results Sections and Creating Tables and Figures*, pages 17–19. SensePublishers, 2017.

[66] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 4th edition, 2020.

[67] R. P. Singh. Solving 0–1 knapsack problem using genetic algorithms. In *IEEE 3rd International Conference on Communication Software and Networks*, pages 591–595, 2011.

[68] M. Sniedovich. *Dynamic Programming: Foundations and Principles*. Taylor & Francis, 2010.

[69] Xiaoyu Song, Jihua Wang, Guijun Yang, and Haikuan Feng. Winter wheat cropland grain protein content evaluation through remote sensing. *Intelligent Automation & Soft Computing*, 20(4):599–609, 2014.

[70] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical report, TR-95-012, International Computer Science Institute, Berkeley, 1995.

[71] M. Fatih Tasgetiren, Quan-Ke Pan, and Yun-Chia Liang. A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times. *Computers & Operations Research*, 36(6):1900–1915, 2009.

[72] M. Fatih Tasgetiren, Quan-Ke Pan, Yun-Chia Liang, and P. N. Suganthan. A discrete differential evolution algorithm for the total earliness and tardiness penalties with a common due date on a single-machine. In *IEEE Symposium on Computational Intelligence in Scheduling*, pages 271–278, 2007.

[73] Maitri Thakur, Lizhi Wang, and Charles R. Hurburgh. A multi-objective optimization approach to balancing cost and traceability in bulk grain handling. *Journal of Food Engineering*, 101(2):193–200, 2010.

[74] Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001.

[75] Brett Whelan. Site-specific crop management. In *Pedometrics*, pages 597–622. Springer International Publishing, 2018.

[76] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011.

[77] Xiaobin Xu, Cong Teng, Yu Zhao, Ying Du, Chunqi Zhao, Guijun Yang, Xiuliang Jin, Xiaoyu Song, Xiaohe Gu, Raffaele Casa, Liping Chen, and Zhenhai Li. Prediction of wheat grain protein by coupling multisource remote sensing imagery and ECMWF data. *Remote Sensing*, 12(8), 2020.

[78] Özgür Yeniay. Penalty function methods for constrained optimization with genetic algorithms. *Mathematical and Computational Applications*, 10(1):45–56, 2005.

[79] Won B. Yoon, Jae W. Park, and Byung Y. Kim. Linear programming in blending various components of surimi seafood. *Journal of Food Science*, 62(3):561–564, 1997.

[80] M.Z. Zgurovsky and A.A. Pavlov. *Combinatorial Optimization Problems in Planning and Decision Making: Theory and Applications*. Springer International Publishing, 2018.

[81] Haitao Zhao, Xiaoyu Song, Guijun Yang, Zhenhai Li, Dongyan Zhang, and Haikuan Feng. Monitoring of nitrogen and grain protein content in winter wheat based on sentinel-2a data. *Remote Sensing*, 11(14), 2019.

[82] Kenny Q. Zhu and Ziwei Liu. Population diversity in permutation-based genetic algorithm. In *Machine Learning: ECML*, pages 537–547. Springer Berlin Heidelberg, 2004.