SN Computer Science (2022) 3:172
https://doi.org/10.1007/s42979-022-01062-8

ORIGINAL RESEARCH

Mixing Grain to Improve Profitability in Winter Wheat Using Evolutionary Algorithms

Md Asaduzzaman Noor[1] · **John W. Sheppard**[1] · **Sean Yaw**[1]

Received: 28 July 2021 / Accepted: 10 February 2022

Abstract
This paper considers an important component of the wheat distribution problem known as grain mixing (wheat blending). We represent the grain mixing problem as a permutation-based combinatorial optimization problem, which both a genetic algorithm and differential evolution are adapted and applied to solve. The proposed algorithms explore a search space that aims at finding a quality mixing of wheat from grain bins that generate the maximum profit at a grain elevator. The experimental results demonstrate that mixing bins provide more profit than not mixing and that the evolutionary approaches lead to consistently higher profits than the non-evolutionary methods.

Keywords Grain mixing · Precision agriculture · Combinatorial optimization · Genetic algorithm · Differential evolution

Introduction

Agriculture and agricultural products are essential in sustaining lives on the planet. Considerable planning is required to feed the population on the Earth efficiently. Issues, such as crop rotation or mixed cropping techniques, optimal seeding and fertilizing, proper irrigation, and efficient harvest and distribution need to be considered for agriculture to meet the needs of most people. Through the emergence of information technologies and software tools in agricultural sectors, it is now possible to collect real-time data and use that data to enhance the farming experience. Appropriately using these digital tools helps to protect the environment, increase overall profit, and reduce waste.

In this paper, we consider an important component of the food supply chain, referred to as grain mixing. The grain considered in our case is wheat, mainly winter wheat that accounts for approximately 70–80% of the wheat production in the US [5]. The lifecycle of winter wheat starts with planting that takes place from mid-August through October, followed by 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 then transport the grain via trucks to sell the wheat to the local grain markets (i.e., elevators). In some states, such as Montana, the price they get from the elevators depends on the quality (protein content) of the wheat.

Several factors play an essential role when determining the profit from wheat production. In Montana, one of the critical elements determining the price of a bushel of wheat is protein content, which is affected by several environmental factors such as temperature during the growing season, soil nitrogen levels, genetics, timing, and precipitation. Due to the resulting variations, protein content 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. The technology is available to track the protein content in a bushel of wheat on site; however, it is expensive, making it inaccessible to several smaller farmers. Consequently, most wheat producers end up taking their harvest to the closest elevators and collecting whatever amount is paid to them. Ultimately, the goal of this work was to see if it is worth

This article is part of the topical collection "Applications of bioinspired computing (to real world problems)" guest edited by Aniko Ekart, Pedro Castillo and Juanlu Jiménez-Laredo.

✉ Md Asaduzzaman Noor
mdasaduzzaman.noor@student.montana.edu

John W. Sheppard
john.sheppard@montana.edu

Sean Yaw
sean.yaw@montana.edu

[1] Gianforte School of Computing, Montana State University, Bozeman, MT 59717, USA

Published online: 24 February 2022

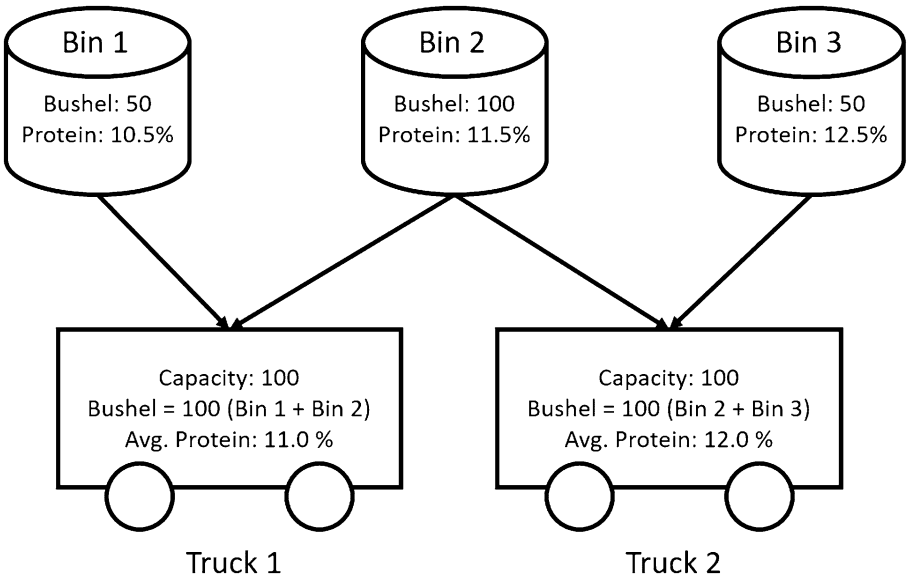


Fig. 1 A simple grain mixing example

investing in the technology and infrastructure needed to track the protein level and mix the wheat on site.

This paper extends our previous work [29] where we applied and adapted two different evolutionary approaches: genetic algorithms and differential evolution to solve the grain mixing problem. We adapted the order crossover (OX) operator for the GA implementation and proposed a new differential mutation operator for the DE implementation. The main extensions in this paper include the following. First, we provide a formal, mathematical programming specification of the problem. We then provide a theorem to show that the problem is NP-hard.[1] Next, we adapted both the partially mapped crossover (PMX) operator as an alternative crossover operator for the GA implementation and the relative position indexing (RPI) operator as an alternative mutation operator for the DE implementation. All these methods were compared against a naïve no-mix strategy to determine if mixing grain increases the profit in the first place and also compared against a deterministic greedy approach to evaluate if the added complexity of the evolutionary approaches is beneficial. We also included more experimental results with a detailed performance analysis of the methods used.

The remainder of this paper is organized as follows. The following section provides the background that includes the formal mathematical definition of the grain mixing problem, existing approaches, and concepts for the methodology used for this study. We also include a theorem to show that this problem is NP-hard, thus justifying the need for heuristic or approximate approaches. The data collected for this study is introduced in "Grain Mixing Dataset". The next section presents the deterministic approaches, and the consecutive sections present the evolutionary approaches for solving the grain mixing problem, and the experimental design, respectively. The experimental results and analysis of the results are presented in "Results and Analysis". "Conclusion and Future Works" concludes our paper and outlines the planned future work.

Background

The Grain Mixing Problem

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 base protein level is (11, 12)%, the price might be \$4 per bushel where it might increase to \$6/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. There are many cases where the protein level of a bin is short of reaching a higher price range or may be 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 (in terms of protein level) bin with a low-quality bin to improve the average protein level, where the low-quality bin reaches the next step in the price range and the high-quality bin remains in the same step in the price range, yielding a better overall profit.

[1] Due to its length, the full proof can be found in [30].

Table 1 Elevator price for grain mixing example

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

Figure 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 1 represents example elevator prices for different quality grains. Many small farmers, who do not track the protein level of their wheat would load each truck with grains from a single bin and sell the unmixed grain to the elevators. A truck has a maximum bushel capacity it can carry which is 100 bu in our example. The price they will get from all the bins without any mixing would be $(50 * \$3 + 100 * \$4 + 50 * \$6) = \$850$. 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 $(100 * \$4 + 100 * \$6) = \$1000$. 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 Fig. 1, there are also other ways to mix grains. 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 profit would be $(100 * \$4 + 100 * \$4) = \$800$ which yield 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 [43]) and store grains 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. The mixing cost might be \$0.1 per bushel in this case. Figure 2 depicts the mixing difficulty scenario based on the site location and Table 2 shows the associated mixing cost for different mixing difficulties, which were provided by one of the farmers with whom we are working.

The delivery cost for a truck transporting to the nearest elevators depends on the distance between the site location and the elevators. For example, a truck moving from site one to elevator one has a delivery cost of \$0.2 per bushel whereas moving to the same elevator from site three has a cost of \$0.18 per bushel. Moreover, the delivery cost varies based on the distance between a specific site and multiple elevators.

Problem Statement

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 mathematical model for the grain mixing problem studied in this paper. The input instances of the problem are described using the notation in Table 3 with the decision variables represented as shown in Table 4. The variables consist of both integer and continuous values. For example, the specific

Table 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

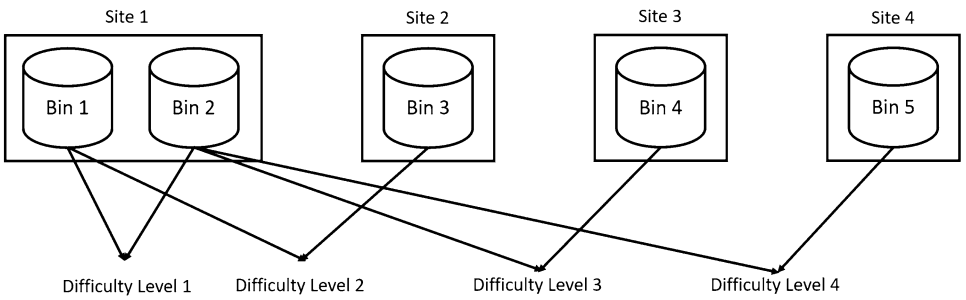


Fig. 2 Mixing difficulty based on farm site

Table 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 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.

variables with bins and elevators are restricted to be an integer whereas the variables with bushels, protein contents, mixing and delivery cost, and elevator prices are continuous.

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

$$\max \sum_{t_i \in T} \sum_{b_j,b_k \in B} \sum_{e \in E} \sum_{g_{e,l} \in G_e} ((g_{e,l}^{p} - (m_{b_j,b_k} + d_{b_k,e})) \times t_{i,j,k,e,l}) \tag{1}$$

Subject to:

$$t_{i,j,k} \leq C_T \qquad \forall t_i \in T \quad \forall b_j, b_k \in B \tag{2}$$

$$t_{i,j,k} \geq 0 \qquad \forall t_i \in T \quad \forall b_j, b_k \in B \tag{3}$$

$$\sum_{t_i \in T} t_{i,j} \leq b_j^{bu} \qquad \forall b_j \in B \tag{4}$$

$$\sum_{t_i \in T} t_{i,j,k} \leq \sum_{b_j \in B} b_j^{bu} \tag{5}$$

$$t_i^{pr} \times t_{i,j,k} = b_j^{pr} \times t_{i,j} + b_k^{pr} \times t_{i,k} \qquad \forall t_i \in T \tag{6}$$

$$t_i^{pr} \times t_{i,j,k,e,l} < g_{e,l}^{Max\_pr} \times t_{i,j,k} \qquad \forall t_i \in T \quad \forall g_{e,l} \in G_e \tag{7}$$

$$t_i^{pr} \times t_{i,j,k,e,l} \geq g_{l}^{Min\_pr} \times t_{i,j,k} \qquad \forall t_i \in T \quad \forall g_{e,l} \in G_e \tag{8}$$

Note that the objective function given in Eq. 1 reflects the nonlinearity of the premium-dockage curve with the inflection point given by the base price/bushel $g_{e,l}^{p}$. Eqs. 2 and 3 represents the linear truck capacity constraints. Bushels loaded from two bins in a truck cannot exceed the maximum capacity of that truck. Equations 4 and 5 represents 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 6 represents the weighted average protein level of a truck after mixing grain from two bins. Finally, Eqs. 7 and 8 are the grade protein requirements that assign a grade (price) and elevator to the truck based on elevator protein requirement and truck protein content. The last three constraints contain a product of two decision variables which introduces nonlinearity in these equations.

Problem Complexity

On the surface, it would appear that, by the nature of applying even an MILP-based approach, our problem could be very difficult. That said, we realize there also exist MILP instances with efficient exact solutions, so we endeavored to assess the underlying computational complexity of our problem formally. To that end, we prove a theorem that the grain mixing problem is NP-Hard following a reduction from the 3-Dimensional Matching problem. Due to its length, we refer the interested reader to [30]. There, the full proof is provided, as well as some additional discussion explaining how the specifics of the problem presented here fits within the scope of the theorem.

Existing Approaches

The grain mixing problem studied in this paper relates to the classical blending optimization problem [9]. 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.

Linear Programming LP is an optimization tool for solving a continuous-space optimization problem. It is possible to solve the classical blending problems by linear programming (LP) methods if both the objective function and the constraints are linear [19]. 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 [16]. Haas used the simplex algorithm to find the optimum blend that satisfies the customer's specific solvent retention capacities (SRC) [15]. 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.

Mixed-integer linear programming MILP is often used to solve many real-world blending problems [34]. 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 [20]. 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 [3]. MILP has also been used in the blending of oil [27], water [38], gasoline [18], and chemical fertilizer [2]. 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.

Meta-heuristic approaches For complex blending problems, meta-heuristic approaches such as evolutionary and swarm-based algorithms have also been used in the literature. Xiang *et al.* proposed a hybrid evolutionary method to solve the wheat blending problem in Australia [23]. 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. Evolutionary approaches have also been used for blending gasoline [7] and composite laminates [1]. Fomeni formulated blending tea optimization as a multi-objective optimization approach and used Monte Carlo simulation for solving the tea blending problem [11]

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.

Combinatorial Optimization Problems (COPs)

Combinatorial optimization is the process of maximizing or minimizing an objective function of a discrete, large configuration space domain [28]. The objective function is often subject to equality and/or inequality constraints, as well as integrity restrictions on many variables. Due to the robust general model, combinatorial optimization is used to represent a variety of problems in fields such as operations research, combinatorics, graph theory, and logic. Some example problems in operations research include efficient distribution of goods, machine sequencing, and production scheduling [8]. There exist many COPs in the literature that can be solved using polynomial-time algorithms. For example, some such problems can be modeled as LP or Integer LP (ILP) problems and can be solved exactly in polynomial time [25]. Some example problems that fall into this category are the shortest path, maximum flow, spanning tree, and matching problems [21].

However, for many real-world problems, the space of possible solutions is often too large to search exhaustively using brute force methods. In that case, approximation algorithms are often used to provide fast but near-optimal solutions. Example approximation algorithms that can provide approximation-guaranteed suboptimal solutions for some intractable problems include greedy, sequential, and local search algorithms [42, 44] and dynamic programming algorithms [39]. Metaheuristic approaches have also been used as approximation algorithms for solving NP-Complete COPs but generally fail to provide any optimality guarantees [4].

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 (or minimized). In the grain mixing problem, the solutions can be represented as a set of bin-pair 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 makes it a permutation-based problem. Other well-known permutation-based COPs include the Traveling Salesperson Problem (TSP), the Flow Shop Scheduling Problem (FSSP), and the Quadratic Assignment Problem (QAP) [6]. 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 been shown to be efficient approaches for solving permutation-based COPs in the literature [37].

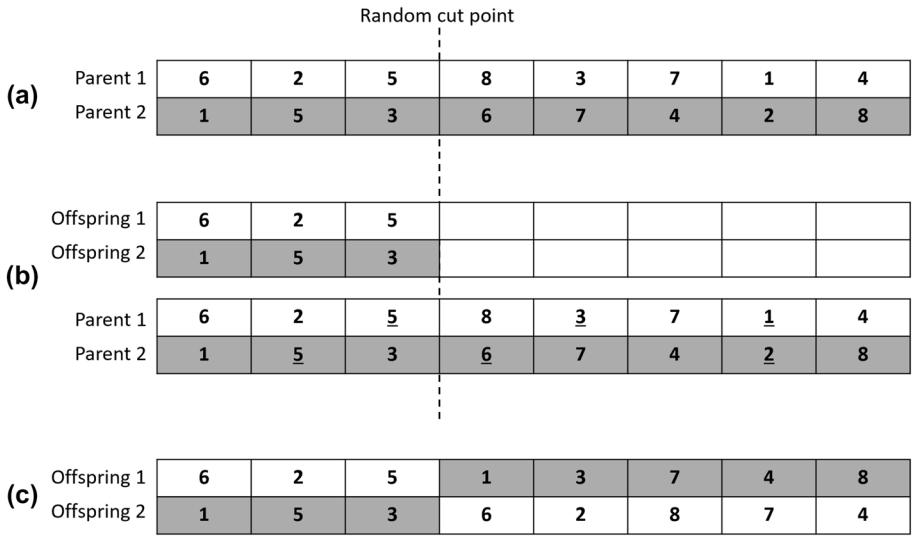


Fig. 3 Order crossover example on two TSP parents

Genetic Algorithm (GA)

A Genetic Algorithm (GA) is a stochastic search algorithm that mimics the evolutionary process of natural selection and "survival of the fittest." The GA was first introduced by John Holland [12] 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.

A GA tries to simulate 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 to be optimized, also sometimes called the fitness function. Then a selection process takes place where more fit individuals have a better chance to participate in the mating pool for producing offspring for the next generation. A crossover procedure takes place to create new offspring (i.e., children) that share some common gene characteristics from both parents. A mutation operator is often applied that alters some genes in the individuals that help the GA to prevent converging to local optima. The process of fitness evaluation, selection, crossover, and mutation continue until finding a satisfactory solution or meeting some termination criterion.

The crossover operator plays an important role in GA's search. 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 [17, 35]. For this study, 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. Next, we discuss the two crossover operators in context with solving traditional permutation-based problems, prior to adaptation.

Ordered Crossover (OX)

The OX operator was first introduced by Davis [10] 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 [32]. 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 series 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 3 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 gene in parent 2 that is already present in the sequence and copies the rest of the genes from parent 2 in the order they are present (excluding the marked one). Therefore, the OX operator respects the relative orders of the genes from parents when generating offspring. It also creates the offspring 2 analogously.

Partially Mapped Crossover (PMX)

The PMX operator was first introduced by Goldberg and Lingel [13], and for some problems, it provides better

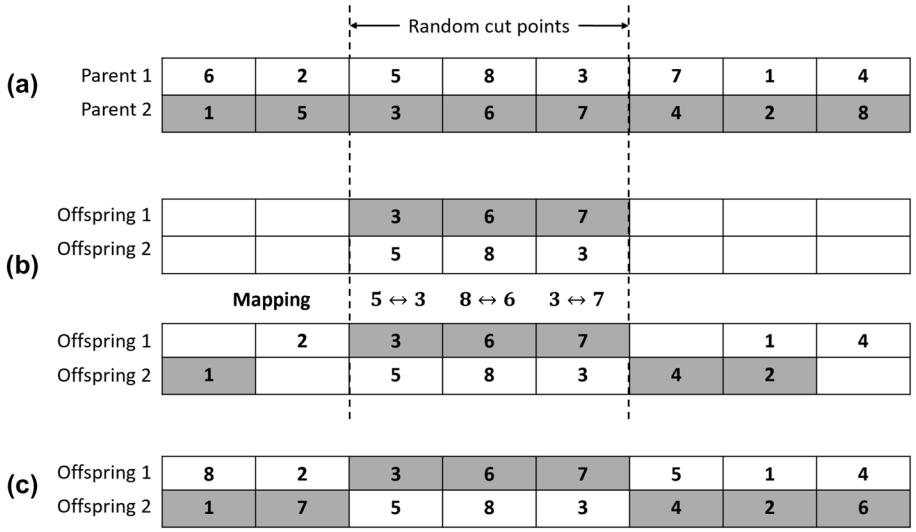


Fig. 4 Partially mapped crossover example on two TSP parents

performance than other crossover operators. Figure 4 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 Fig. 4b, 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* $5 \leftrightarrow 3$). It then creates the offspring 2 analogously.

Differential Evolution (DE)

The computational steps of DE, which was first introduced by Storn and Price [40], are similar to the GA with a difference in the parameter vectors for exploring the search space. DE is computationally simple, reliable, and robust which makes it a competitive approach for solving continuous optimization problems; however, DE needs to be adapted for combinatorial problems such as ours. Here, we will briefly describe the search mechanism of standard 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. 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.

The differential mutant 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).

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 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.

Finally, a selection procedure selects a 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.

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. Some common mutation operator includes the Permutation Matrix, Adjacency Matrix, Relative Position Indexing, and Forward-Backward Transformation approaches [31]. 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. Next, we will briefly discuss the Relative Position Indexing operator when applied to TSP instances and our proposed perturbation method.

Relative Position Indexing (RPI)

The RPI approach was adapted from the original differential mutation and is only applicable for permutation-based problems [24]. It has been applied successfully in COPs like permutation flow-shop scheduling [33, 36]. 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.

$$x^g_{r1} = [1\ 5\ 3\ 6\ 7\ 4\ 2\ 8]$$
$$x^g_{r2} = [6\ 2\ 5\ 1\ 3\ 7\ 4\ 8]$$
$$x^g_{r3} = [8\ 2\ 3\ 6\ 7\ 5\ 1\ 4]$$

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:

$$x(f)^g_{r1} = [0.125\ 0.625\ 0.375\ 0.750\ 0.875\ 0.500\ 0.250\ 1.000]$$
$$x(f)^g_{r2} = [0.750\ 0.250\ 0.625\ 0.125\ 0.375\ 0.875\ 0.500\ 1.000]$$
$$x(f)^g_{r3} = [1.000\ 0.250\ 0.375\ 0.750\ 0.875\ 0.625\ 0.125\ 0.500]$$

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:

$$v(f)^g_i = x(f)^g_{r1} + F \times \left(x(f)^g_{r2} - x(f)^g_{r3}\right)$$
$$v(f)^g_i = [0.025\ 0.625\ 0.475\ 0.500\ 0.675\ 0.600\ 0.400\ 1.200]$$

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:

$$v^g_i = [1\ 6\ 3\ 4\ 7\ 5\ 2\ 8]$$

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.

Generation Best Perturbation (GBP)

The second differential mutation operator that we used involved perturbing the chromosome of the generation best individual. A similar approach has been proposed in the literature for solving application-specific problems using discrete DE [14, 41]. 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 proposed GBP operator can be represented as follows:

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

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_{r3}. If the gene is already present in the mutant vector, it moves to the next index until it finds a valid gene entry.

Grain Mixing 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

Table 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

Table 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

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 5, and the details on each bin for the year 2017 are shown in Table 6.

The data provided also includes 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 7.

Figure 5 illustrates 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. 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.

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

Baseline Deterministic Approaches

In this section, we introduced two deterministic approaches for solving the grain mixing problem. We use these two approaches for performing a baseline comparison to our proposed evolutionary methods. The first method is referred to as the "no mixing" 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 deterministic approach is referred to as the "greedy" mixing approach where the grain mix is determined based on a lookup table with combinations of bin

Table 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

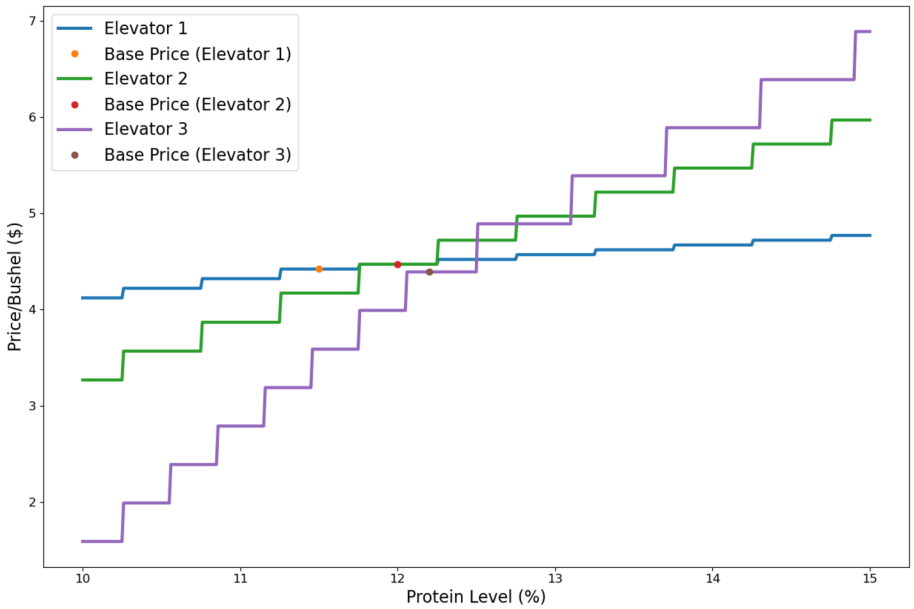


Fig. 5 Premium-dockage curve of three elevators in Northwest Montana in 2017

pairs and pre-computed mixing ratios to provide the highest immediate profit, under the assumption 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.

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. Finally, the total profit is calculated by summing the individual profits from all loaded trucks.

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 and $P(16, 2) = 16!/(16-2)! = 240$ ways to select two bins. Here, we used the permutation to determine the delivery cost when a truck transporting to an elevator only depends on the site number of the second bin. Therefore, the order from which a truck loads grain changes the profit. Moreover, to make the LPT finite, we discretized the mixing ratio $\alpha \in \{0.1, 0.2, 0.3, ..., 0.9\}$. Therefore, in the LPT, there are a total of $240 \times 9 = 2160$ unique entries. Table 8 shows a subset of the LPT for a particular set of elevators/markets and their premium-dockage curves. 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.

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 5000 bushels left in bin 1 but only 3000 bushels left in bin 2, the algorithm will take 4000 bushels from bin 1 (half the capacity of the truck) and 3000 bushels from bin 2. Therefore, the total number of bushels in that truck will be 7000, 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 1 shows the pseudocode for the greedy mixing approach.

Table 8 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

Fig. 6 Sample encoding for genetic algorithm

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

Algorithm 1: GreedyMix

```
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
```

Evolutionary Approaches

For this study, we utilized two evolutionary approaches namely a GA and DE, to solve the grain mixing problem. We introduced a novel permutation-based representation for these approaches tailored specifically to the grain mixing problem. Specifically, we used a permutation to specify an order for how to load trucks so as ensure the constraints remain satisfied. Next, we discuss our specific implementation of the GA and DE for solving the grain mixing problem.

Genetic Algorithm

Encoding

For an individual chromosome in the target population, we utilized a permutationbased representation where each gene in the chromosome is a unique tuple $\langle ID, bin\_pair, \alpha \rangle$. Here *bin_pair* specifies the pair of bins used to load a truck, and α represents the mixing ratio (the number of bushels drawn from each bin for mixing). There are a total of 2160 unique mixes defined in the LPT, and *ID* 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 6.

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*), and the size of chromosome *m* is also a tunable parameter. The *m* mixing

combinations 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.

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 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 profit. Otherwise, it discards the grain in that 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 2160 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 $l < m$) without violating any constraints and returns the total profit of all loaded trucks. The pseudocode for *IndividualMix* is shown in algorithm 2.

Selection

The GA uses tournament selection [26] 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. The selection pressure that controls the GA's convergence rate highly depends on the tournament size.

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 Sects. "Ordered Crossover (OX)" and "Partially Mapped Crossover (PMX)", 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 2160 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.

As the fitness function sorts the mixing combinations in the chromosome based on the maximum profit, 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

Algorithm 2: IndividualMix

```
Input: Individual chromosome of m mixing combinations (MC)
Sort MC with respect to the maximum profit
for each combination in the sorted MC:
    if both bins have sufficient grain remaining:
        Load truck fully/partially based on bin pair and mixing ratio
        if truck makes profit: record profit
        else: discard grain
for each combination in the unused sorted MC:
    if single bin has sufficient grain remaining:
        Load truck fully/partially from the single bin
        if truck makes profit: record profit
        else: discard grain
return sum of truck profits
```

new mixing combinations in the chromosome list affects the fitness value. Although a gene is represented by the tuple $< ID, bin\_pair, \alpha >$ 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 crossover operators.

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).

Differential Evolution

Our approach to applying DE to the grain mixing problem is similar to the GA approach; however, the major differences lie in the computational steps and the search mechanism. The encoding, initial population, and fitness function used in DE are the same as in GA. The main differences lie in the *mutation*, *crossover*, and *selection* operators. Next, we discuss our specific implementation of these operators.

Mutation

For this study, we used two differential mutation operators. The first one is an adaptation of Relative Position Indexing (RPI) as described in Sect. "Relative Position Indexing (RPI)". To illustrate the adapted RPI, consider three donor vectors x_{r1}, x_{r2}, and x_{r3}. For example purposes only, assume there are a total of 10 mixing combinations, and the vector length of each donor vector is 5 (i.e., draws 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):

$$x_{r1} = [9\ 4\ 6\ 1\ 3]$$
$$x_{r2} = [7\ 10\ 2\ 4\ 8]$$
$$x_{r3} = [3\ 10\ 9\ 1\ 4]$$

From these, we create a union vector that includes all of the different combination *ID*s from the three donor vectors as follows:

$$U = x_{r1} \cup x_{r2} \cup x_{r3} = [1\ 2\ 3\ 4\ 6\ 7\ 8\ 9\ 10]$$

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:

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

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:

$$v(f) = x(f)^g_{r1} + F \times \left(x(f)^g_{r2} - x(f)^g_{r3}\right) = [1.2\ 0.44\ 0.32\ 0.26\ 0.53]$$

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:

$$U = [1\ 2\ 3\ 4\ 6\ 7\ 8\ 9\ 10]$$
$$U(f) = [0.1\ 0.2\ 0.3\ 0.4\ 0.6\ 0.7\ 0.8\ 0.9\ 1.0]$$
$$v(f) = [1.2\ 0.44\ 0.32\ 0.26\ 0.53]$$
$$v = [10\ 4\ 3\ 2\ 6]$$

The second differential mutation operator is the Generation Best Perturbation (GBP) as described in Sect. "Generation Best Perturbation (GBP)". For the GBP illustration, let us consider the same donor vectors that we used in the RPI example. With a mutation factor of 0.5 and $q \in random(0, 1)$, the mutation vector v can be obtained as follows:

$$q = [0.57\ 0.19\ 0.32\ 0.84\ 0.07]$$
$$x_{best} = [9\ 4\ 6\ 1\ 3]$$
$$x_{r2} = [7\ 10\ 2\ 4\ 8]$$
$$x_{r3} = [3\ 10\ 9\ 1\ 4]$$
$$v = [7\ 9\ 4\ 3\ 6]$$

Table 9 Parameter settings for GA and DE algorithms

Algo	#MaxIter	Population	#MC	Tournament	Mutation
GA	500	200	50	5	0.2
Algo	**#MaxIter**	**Population**	**#MC**	**CR**	**F**
DE	500	100	100	0.9	0.5

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.

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.

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. To further evaluate the performance of the 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 year 2017 was used with the elevator prices from 2016 and vice-versa. Then we created 5 artificial datasets for each year. For these datasets, the elevators, mixing cost, and delivery cost remained the same, however, the bin contents were altered. We used the same 16 bins, and the number of bushels and the protein content in each bin were assigned randomly falling within a pre-specified lower and upper bound.

For the initial population, GA and DE implementations created 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 and DE algorithms is to find the best sequence from the profit table for which the 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 and DE algorithms, were tuned manually. Table 9 shows the tuned parameter values used in the GA and DE algorithms for all of test cases. Column *#MaxIter* represents the maximum generation number, and column *#MC* represents the number of mixing combinations in the chromosome list (chromosome size). Column *F* represents the mutation factor for the DE algorithm. These sets of parameters provided the highest average profit for the real datasets, so they were used for all of the experiments.

To further assess the effectiveness of the evolutionary approaches, we also introduced a *Random* algorithm as a stochastic baseline. This method creates 100 individuals by randomly taking 100 mixing combinations from the total 2,160 combinations and returns the individual with the highest profit. The *Random* algorithm helps us assess if the evolutionary algorithms are exploring the search space efficiently to provide a better solution than random search.

The deterministic approaches always yield the same solution; however, that is not the case with the evolutionary approaches and the random search algorithm. They are stochastic, so to evaluate their performance, we ran 10 experiments on each dataset and recorded the average overall profit.

Results and Analysis

Profitability Analysis

Table 10 shows the performance of the different algorithms for each of the different test cases studied. 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 both GA and DE, there are two versions, one for each of the different adapted operators. The overall profit from the

Table 10 Performance of the different algorithms (*profit* in thousands of dollars)

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

stochastic algorithms (i.e., GA, DE, and Random) 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 solution (i.e., NoMix). If the increased profit is more than $5000 from the base solution, we call it sufficient as the average price for protein monitors costs around $5000.

Based on different algorithms' performance, we note that the evolutionary approaches lead to a higher overall profit compared to the *NoMix* solution in every case. For all datasets, GA's OX and PMX operators, and DE's GBP operator consistently provided a sufficient profit increase. Surprisingly, 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. Comparing the two crossover operators of the GA, the PMX operator for all test cases provided a higher profit than the OX operator. 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 overall profit from the *Random* and *Greedy* algorithms are consistently lower than the evolutionary approaches 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 evolutionary approaches are 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

Table 11 Best solution from GA(PMX) for *R*_2017 dataset

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

Table 12 Best solution from DE(GBP) for *R*_2017 dataset

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

provided, which would be necessary to provide a more complete estimate of 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. But, 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. 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 test cases) than the base solution. The evolutionary approaches do suggest that taking a sub-optimal bin-pair combination can yield a higher overall profit than the greedy optimal choice due to the varying bin sizes.

Next, we examine the best complete solutions obtained from the GA's PMX operator in Table 11 and DE's GBP operator in Table 12 for the real *R*_2017 dataset. In these tables, 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 8000 bu) and "Elv" identifies the elevator where the truck delivers the grain.

The mixing ratio of the last truck in the PMX solution (Table 11) is 0, which indicates that after the first sweep in the mixing combination (chromosome list), sufficient grain remained in bin 16. Therefore, it loaded the truck separately with the remaining grain without incurring any mixing cost. We observed the same scenario in the GBP solution (Table 12) where the last three trucks show a mixing ratio of 0, and sufficient grain remained in bin 16 for all three trucks. Both best solutions loaded grain from bin 16 separately, which implies that mixing bin 16 with other bins might reduce the overall profit.

All of the grain-mixing algorithms provided a similar solution format but with different bin-pair mixings. A complete solution shows a farmer how to load each truck with bushels from respective bins to generate the solution's overall profit.

Population Diversity

One of the questions to be addressed is whether or not the evolutionary methods, those superior to the deterministic and random methods, might be converging prematurely. 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 both GA and DE.

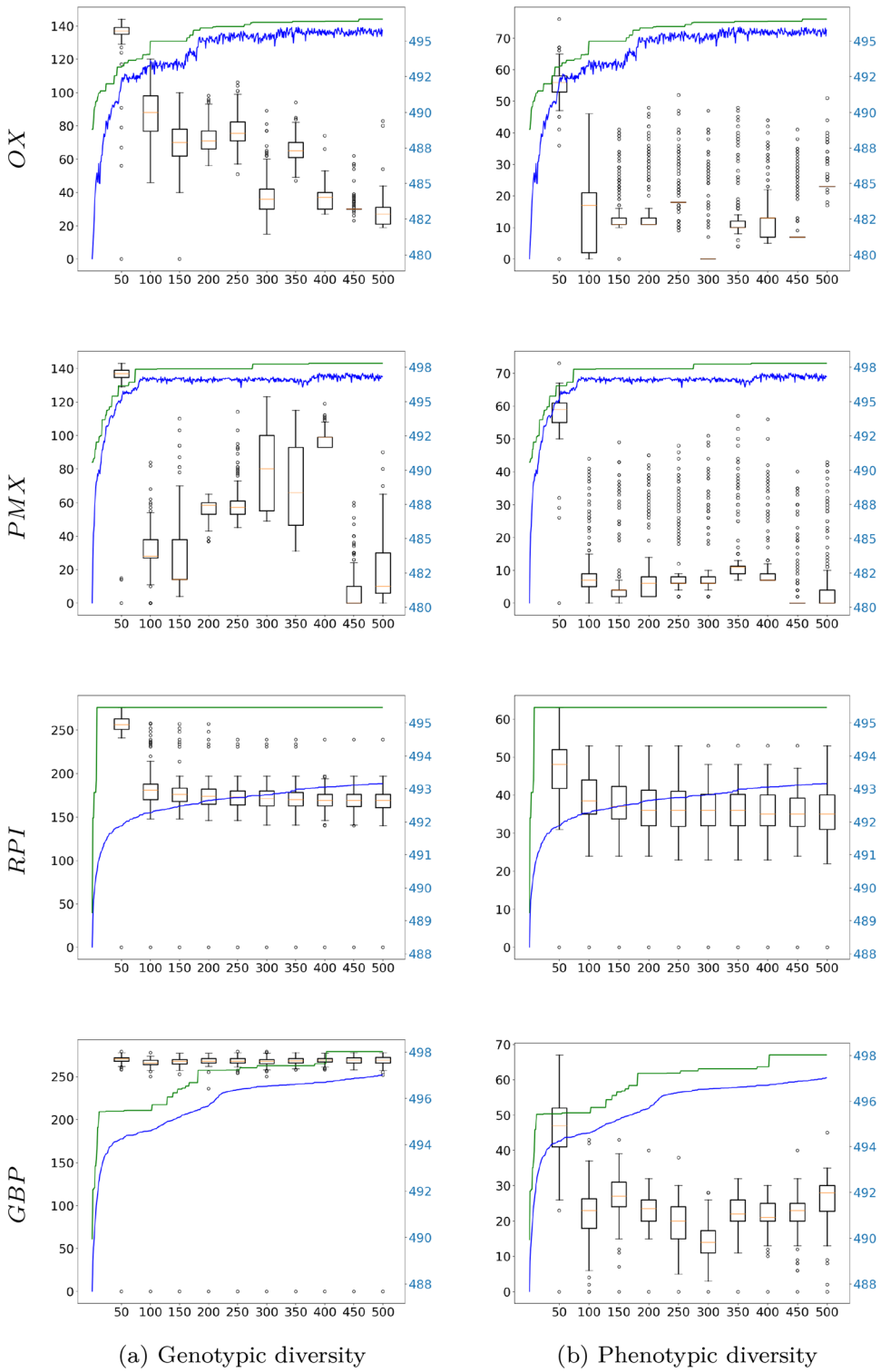
To assess the population diversity, we tracked the fitness (profit) of the best individual and the average fitness of the population in each generation. Since our permutation-based representation is composed of mixing combinations, for different mixing combinations the fitness might be similar. Therefore, in our case, using only fitness values to monitor diversity may not be appropriate. 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, and *phenotypic diversity*, which compares the actual mixing combinations used in the final solution (generated by *IndividualMix*) between two individuals.

To compute diversity, we used Levenshtein distance (i.e., edit distance) [22] 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 $< bin\_pair1, bin\_pair2, \alpha >$. 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 $< 2, 15, 0.6 >$ 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 7 shows the fitness, genotypic and phenotypic diversity of the evolutionary algorithms for the *R*_2017 dataset.

In Fig. 7, column (a) shows the genotypic diversity, and column (b) shows the phenotypic diversity for the different recombination operators used in GA and DE. 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 green line shows the best individual fitness for the current generation, and the lower blue line represents the average individual 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 different (since each combination is encoded by a three-character value) for the genotypic conversion. However, DE uses 100 combinations, and the maximum distance, in this case, is 300. 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.

Looking at the diversity plots for GA's OX and PMX operators, the average fitness curve seems to converge on the generation best; however, the edit distance box plots show



(a) Genotypic diversity (b) Phenotypic diversity

Fig. 7 Monitoring population diversity of the best solutions in *R*_2017 dataset. The *x* axis shows the generation number, the left *y* axis shows the edit distance, and the right *y* axis shows the fitness. The top green line tracks the best individual fitness, and the lower blue line tracks the average individual fitness

that there is diversity both in the chromosome representation (genotypic) and actual (phenotypic) solutions. Furthermore, PMX seems to control the population diversity slightly better than OX, which might explain why PMX provided a better solution for all test cases compared to OX. On the other hand, the diversity plots for the DE's mutation operators are different from the GA operators, portraying the different search mechanisms of the two approaches. The average fitness curve for DE is increasing gradually as it always replaces a current individual with an individual with a better fitness value. The plots from the RPI operator show that convergence is slow, compared to the GBP operator. This

may be the reason why RPI performed the worst compared to all other operators. Although GBP's genotypic diversity is quite high for all generations, the phenotypic diversity shows that, as generations progress, more individuals with slightly different solutions from the generation's best solution appear in the population.

Conclusion and Future Works

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. In some states, the price of a bushel of wheat depends on its protein level as measured at the elevator. Our intent was to determine if mixing different protein levels can yield a mix such that the farmer will receive more profit than taking them separately, thus justifying the expense of purchasing and employing expensive protein monitoring equipment on their harvesters. We adapted and applied two different evolutionary approaches—a genetic algorithm and differential evolution—to address the grain mixing (wheat blending) problem. The experimental results from both real and simulated datasets showed that the evolutionary approaches consistently provided an increased profit compared to no mixing, greedy mixing, and random mixing approaches. Furthermore, all of the test cases showed there was benefit in investing in the protein monitors.

For this study, we made certain assumptions, such as limiting the mixing to only two bins at a time for the problem representation. This was based on limits given to us by the farmers. In future work, we would like to consider if mixing more than two bins would provide a better profit, thereby justifying to the farmers the more complicated process of mixing from multiple bins. It is also our intent to explore the suitability of comparing to a MINLP or a relaxed MILP model; however, the inherent nonlinearity constraints of the problem would seem to suggest optimization quality would be limited. Furthermore, a more realistic cost model that includes the production cost associated with harvesting, the cost of the protein tracking device and supporting infrastructure, and alternative representations of the model will be considered as future work.

Funding There is no funding associated with this work.

Declarations

Conflicts of interest The authors declare that they have no conflict of interest.

References

1. Adams DB, Watson LT, Gürdal Z, Anderson-Cook CM. Genetic algorithm optimization and blending of composite laminates by locally reducing laminate thickness. Adv Eng Softw. 2004;35(1):35–43.
2. Ashayeri J, van Eijs A, Nederstigt P. Blending modelling in a process manufacturing: a case study. Euro J Oper Res. 1994;72(3):460–8.
3. Bilgen B, Ozkarahan I. A mixed-integer linear programming model for bulk grain blending and shipping. Int J Prod Econ. 2007;107(2):555–71.
4. Blum C, Roli A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Comput Surv. 2003;35(3):268–308.
5. Bond JJ. Wheat sector at a glance. U.S. Department of Agriculture. https://www.ers.usda.gov/topics/crops/wheat/wheat-sector-at-a-glance. Last accessed 07/14/21.
6. Ceberio J, Irurozki E, Mendiburu A, Lozano J. A review on estimation of distribution algorithms in permutation-based combinatorial optimization problems. Progress Artificial Intell. 2012;1:103–17.
7. Chen X, Wang N. Optimization of short-time gasoline blending scheduling problem with a dna based hybrid genetic algorithm. Chem Eng Process. 2010;49(10):1076–83.
8. Crama Y. Combinatorial optimization models for production scheduling in automated manufacturing systems. Euro J Oper Res. 1997;99(1):136–53.
9. Dantzig G. Formulation a linear programming model. In: Linear programming and extensions, pp. 42–50. Princeton Univ. Press, Princeton, NJ (1963).
10. Davis L. Applying adaptive algorithms to epistatic domains. In: Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 1, p. 162–164. Morgan Kaufmann Publishers Inc. (1985).
11. Djeumou Fomeni F. A multi-objective optimization approach for the blending problem in the tea industry. Int J Prod Econ. 2018;205:179–92.
12. Goldberg DE, Holland JH. Genetic algorithms and machine learning. Mach Learn. 1988;3(2–3):95–9.
13. Goldberg DE, Lingle R. Alleles, loci and the traveling salesman problem. In: Proceedings of the 1st International Conference on Genetic Algorithms, p. 154–159. L. Erlbaum Associates Inc., USA (1985).
14. Guo Q, Tang L. Modelling and discrete differential evolution algorithm for order rescheduling problem in steel industry. Comput Ind Eng. 2019;130:586–96.
15. Haas N. Optimizing Wheat Blends for Customer Value Creation: A Special Case of Solvent Retention Capacity. MS Thesis, Kansas State University, USA (2011).
16. Hayta M, Cakmalki U. Optimization of wheat blending to produce breadmaking flour. J Food Process Eng. 2001;24:179–92.
17. Hughes JA, Houghten S, Ashlock D. Permutation Problems, Genetic Algorithms, and Dynamic Representations, pp. 123–149. Springer International Publishing (2017).
18. Jia Z, Ierapetritou M. Mixed-integer linear programming model for gasoline blending and distribution scheduling. Ind Eng Chem Res. 2003;42(4):825–35.
19. Karloff H. Linear Programming. USA: Birkhauser Boston Inc.; 1991.
20. Krentel MW. The complexity of optimization problems. J Comput Syst Sci. 1988;36(3):490–509.
21. Lawler E. Combinatorial Optimization: Networks and Matroids. Rinehart and Winston: Holt; 1976.

22. Levenshtein VI. Binary codes capable of correcting deletions, insertions, and reversals. Cybernet Control Theory. 1966;10(8):707–10.
23. Li, X., Bonyadi, M.r., Michalewicz, Z., Barone, L.: A hybrid evolutionary algorithm for wheat blending problem. TheScientificWorldJournal **2014**, 967254 (2014)
24. Lichtblau D. Relative position indexing approach. In: Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization, pp. 81–120. Springer Berlin Heidelberg (2009).
25. Matouek J, Gärtner B. Understanding and using linear programming. Berlin, Heidelberg: Springer-Verlag; 2006.
26. Miller B, Goldberg D. Genetic algorithms, tournament selection, and the effects of noise. Complex Syst. 1995;9:193–212.
27. Moro LFL, Pinto JM. Mixed-integer programming approach for short-term crude oil scheduling. Ind Eng Chem Res. 2004;43(1):85–94.
28. Nemhauser GL, Wolsey LA. Integer and Combinatorial Optimization. USA: Wiley-Interscience; 1988.
29. Noor MA, Sheppard JW. Evolutionary grain-mixing to improve profitability in farming winter wheat. In: Applications of Evolutionary Computation, pp. 113–129. Springer International Publishing (2021).
30. Noor MA, Yaw S, Zhu B, Sheppard JW. Optimal grain mixing is NP-Complete. arXiv:2112.08501 (2021). https://arxiv.org/abs/2112.08501v1.
31. Onwubolu G, Davendra D. Differential evolution for permutation—based combinatorial problems. In: Differential Evolution: a handbook for global permutation-based combinatorial optimization, pp. 13–34. Springer Berlin Heidelberg (2009).
32. Otman A, Jaafar A. A comparative study of adaptive crossover operators for genetic algorithms to resolve the traveling salesman problem. Int J Comput Appl. 2011;31(11):49–57.
33. Pan QK, Wang L, Qian B. A novel differential evolution algorithm for bi-criteria no-wait flow shop scheduling problems. Comput Oper Res. 2009;36(8):2498–511.
34. Pochet Y, Wolsey LA. Production planning by mixed integer programming. 1st ed. Incorporated: Springer Publishing Company; 2010.
35. Potvin J. Genetic algorithms for the traveling salesman problem. Ann Oper Res. 1996;63:337–70.
36. Qian B, Wang L, Hu R, liang Wang W, Huang D, Wang X. A hybrid differential evolution method for permutation flow-shop scheduling. Int J Adv Manuf Technol **38**, 757–777 (2008).
37. Radhakrishnan A, Jeyakumar G. Evolutionary algorithm for solving combinatorial optimization—a review. In: Innovations in Computer Science and Engineering, pp. 539–545. Springer Singapore (2021).
38. Randall D, Cleland L, Kuehne CS, Link GWB, Sheer DP. Water supply planning simulation model using mixed-integer linear programming "engine." J Water Resources Plann Manag. 1997;123(2):116–24.
39. Sniedovich M. Dynamic programming: foundations and principles. Taylor & Francis (2010).
40. Storn R, Price K. Differential evolution—a simple and efficient adaptive scheme for global optimization over continuous spaces. Tech. rep., TR-95-012, International Computer Science Institute, Berkeley (1995).
41. Tasgetiren MF, Pan QK, Liang YC. A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times. Comput Oper Res. 2009;36(6):1900–15.
42. Vazirani VV. Approximation algorithms. Springer (2001).
43. Whelan B. Site-Specific Crop Management, pp. 597–622. Springer International Publishing (2018).
44. Williamson DP, Shmoys DB. The Design of Approximation Algorithms. Cambridge University Press (2011).