Hindawi
Applied Computational Intelligence and Soft Computing
Volume 2023, Article ID 9918022, 23 pages
https://doi.org/10.1155/2023/9918022

Research Article

A Two-Phase Pattern Generation and Production Planning Procedure for the Stochastic Skiving Process

Tolga Kudret Karaca,[1] **Funda Samanlioglu**,[2] **and Ayca Altay**[3]

[1]*Department of Computer Engineering, Istanbul Topkapı University, Zeytinburnu, Istanbul 34087, Türkiye*
[2]*Department of Industrial Engineering, Kadir Has University, Cibali, Istanbul 34083, Türkiye*
[3]*Department of Industrial and Systems Engineering, Rutgers University, 96 Frelinghuysen Rd., Piscataway, NJ 08540, USA*

Correspondence should be addressed to Tolga Kudret Karaca; tolgakudretkaraca@topkapi.edu.tr

Received 27 August 2023; Revised 9 October 2023; Accepted 25 October 2023; Published 6 November 2023

Academic Editor: Yiming Tang

The stochastic skiving stock problem (SSP), a relatively new combinatorial optimization problem, is considered in this paper. The conventional SSP seeks to determine the optimum structure that skives small pieces of different sizes side by side to form as many large items (products) as possible that meet a desired width. This study studies a multiproduct case for the SSP under uncertain demand and waste rate, including products of different widths. This stochastic version of the SSP considers a random demand for each product and a random waste rate during production. A two-stage stochastic programming approach with a recourse action is implemented to study this stochastic $\mathcal{NP}$-hard problem on a large scale. Furthermore, the problem is solved in two phases. In the first phase, the dragonfly algorithm constructs minimal patterns that serve as an input for the next phase. The second phase performs sample-average approximation, solving the stochastic production problem. Results indicate that the two-phase heuristic approach is highly efficient regarding computational run time and provides robust solutions with an optimality gap of 0.3% for the worst-case scenario. In addition, we also compare the performance of the dragonfly algorithm (DA) to the particle swarm optimization (PSO) for pattern generation. Benchmarks indicate that the DA produces more robust minimal pattern sets as the tightness of the problem increases.

1. Introduction

The skiving stock problem (SSP) has been described by Zak [1] as the companion piece to the cutting stock problem (CSP) since they have similar inputs and solution approaches. Skiving is a relatively new technology. It involves joining, binding, stitching, or sealing together small pieces (auxiliary rolls) to form large items (products) that meet a minimum specified width. It aims, on the whole, to obtain the maximum number of large items as possible [1]. Several narrow rolls are joined in the paper industry to construct wider rolls [1]. In manufacturing toothed belts, the small rectangular pieces remaining after cutting are stitched together to form large rectangles [2]. Other skiving applications include pipe manufacturing, firefighting system design [3], and cognitive radio network spectrum aggregation [4]. In particular, in industries with high raw material residues, the skiving process is widely applicable [2]. In this case, a set of optimal pattern combinations is generated, maximizing production with limited availability of smaller items. The skiving process is an inherently waste-minimization procedure, mainly modeled as a mixed-integer problem (MIP). Because of the $\mathcal{NP}$-hard structure of the MIP, enumerating all feasible patterns is difficult, specifically for large-scale problems. Pattern-based models [1], arc flow models [5], or assignment models [4] are some of the solution approaches presented in the relevant literature. Methodologically, for the MIP structure of its mathematical formulation [1], the synthesis of column generation (CG) and branch and bound (BB) is fundamental. Due to the arduous nature of obtaining an optimal solution for large problems and dimensional complexities, most of the proposed methodologies for the

SSP employ heuristics [6]. These heuristics can essentially aid in ensuring integer solutions after solving the linear relaxation [6], and the construction of the minimal pattern sets provides input to the mathematical model [7]. The performance of metaheuristic optimization methods on the SSP compared to the CSP remains under-researched in the literature, despite the fact that metaheuristic methods have been very promising on many similar problems, such as the CSP.

The vast majority of the SSP literature considers solving the deterministic SSP. These approaches stem from the assumption that the decision-maker has the perfect information regarding all parameters, which leads to unrealistic results. Real-world problems involve large uncertainties in various parameters such as product demand, resource availability, yields, set-up and processing times, and costs. In deterministic planning, where only expected values are considered, it is not possible to make an appropriate adaptive decision to minimize the risk caused by the variability of these factors. Many researchers have investigated the various structures for the stochastic CSP, where the demand [8, 9] or the yield [10] is a random variable. However, there is a noticeable lack of stochastic solution strategies for the SSP in the literature.

In this study, we expand upon the pure SSP in several dimensions by including (i) the set-up costs for each pattern change, (ii) the raw materials costs of the requisite small items, and (iii) the required quantities of small items to be used. Furthermore, a heuristic method for solving the multiproduct SSP in a given stochastic setting is implemented. For this stochastic version of the SSP problem, two-stage stochastic programming (SP) with recourse is employed [11–13] and a mathematical model for weakly inhomogeneous items under both stochastic demand and stochastic waste (scrap) rate is proposed. In the two-stage stochastic program, production decisions are made before the demand occurrence, as opposed to the "wait-and-see" approach where decisions are made after the reveal of the random variable values [8, 9, 11, 14]. We make decisions about production amounts, skiving patterns, and the number of each replicated skiving pattern before any scenario occurs, as they are scenario-independent decision variables. In the second stage, the scenario-dependent decision variables are the underproduction and overproduction quantities. These variables represent the measures taken under possible scenarios. Furthermore, a two-phase procedure is applied to solve the problem, where the first phase produces the minimum skiving pattern set and the second aims at minimizing production costs. This two-phase procedure continues recursively until the target production quantity is obtained. In the first phase, we implement the dragonfly algorithm (DA) [15] and generate the skiving patterns. This first phase's output serves as the second-stage's input, where we implement a range of scenarios combining random variables. The sample-average approximation (SAA) method for the two-stage stochastic programming model is used to obtain a solution to the SSP [13, 16].

The terms "two-phase" and "two-stage" are not interchangeable. The overall solution process is referred to as "two-phase." The first phase refers to the DA that generates the skiving patterns. The second phase refers to the two-stage stochastic programming with a recourse action model that minimizes the expected total production cost. This study is a single-objective analysis that minimizes the total production cost. However, it also aims to maintain an acceptable trim loss level as a DA goal in the first phase.

The paper is structured as follows: the next section reviews the relevant literature, and Section 3 outlines the stochastic SSP. The methodology is presented in Section 4. This section is followed by an illustrative example in Section 5. Numerical experiments and discussions are provided in Section 6, together with the computational complexity discussions. Finally, in Section 8, the conclusions and future work are presented.

2. Literature Review

The SSP was initially proposed by Johnson et al. [17] as an incorporated part of the CSP. They unified the CSP and the SSP into a single problem called the cutting and skiving stock problem (CSSP). The CSSP modeled the cutting and skiving of large products in a two-step framework. A pattern-based mathematical formulation and commercially available software (MAJIQTRIM) are proposed using heuristics and linear programming to solve the CSSP. It was only when Zak [1] carried out a theoretical analysis comparing the SSP and the CSP models that the SSP was recognized as a problem in its own right. Zak's study [1] proved that the SSP is not the dual form of the CSP. According to Zak [1], the SSP shares some input data similarities with the CSP in terms of item widths, consumer demand, and scalar knapsack capacity. However, the SSP and the CSP have different pattern matrices due to their different structures. The CSP is structured as a set-packing while the SSP is structured as a set-covering [18]. Following these findings, Zak [1] reported that the SSP is not the dual form of the CSP and launched SSP as a stand-alone challenge in combinatorial optimization [1, 19]. Martinovic and Scheithauer [20–22] stated that the SSP is structure-wise closer to the dual bin packing problem (DBPP), also referred to as a particular type of the bin covering problem (BCP). However, the SSP differs in being expressed and solving from the DBPP [1]. Martinovic and Scheithauer [20] pointed out these differences as the level of heterogeneity of item sizes and their available quantities based on the study of Wäscher et al. [18]. According to this study [18], the SSP is associated with items with low heterogeneity. In contrast, DBBP contains very heterogeneous items. Furthermore, the DBPP formulation regards the presence of each small item type as one [23]. Zak [1] has extended this problem by incorporating higher availability values for small items. Ultimately, item-oriented models and heuristics are used in the mathematical formulations and solution approaches of the DBPP.

The SSP is challenged by finding integer solutions for large problems, as with CSP. Consequently, the most common way is to obtain a linear relaxation and then discretize the solution. Arbib et al. [2] used a column generation (CG) [24] and a branch-and-bound (BB) algorithm to solve the CSSP with cutting loss minimization in the production of transmission belts. They extended the single-period CSSP model proposed in [2] to a multiperiod problem in which a BB algorithm solves a CSSP with a small size [25]. In a similar study, Zak [1] presented CG [26] for the linear relaxation of the problem and a (BB) algorithm to obtain integer solutions for the SSP. Meanwhile, Gilmore and Gomory's [24, 26] pattern-based model and column generation (CG) for the large-scale CSP are also convenient for the SSP [1]. Furthermore, the pure SSP can be categorized as an output maximization based on Wäscher et al. [18].

In addition, the 1D-CSP with a skiving option is by Ágoston [3] for a fire-fighting system, where single-sized pipes are cut into smaller pipes, and the residual parts can be joined using the skiving technique to produce extended pipes. The process allows only one welding operation on the pipe for safety reasons. Ágoston also transformed the CSP into a mixed-integer linear programming (MILP) model, which includes sequential cutting and welding patterns, and proposed a three-stage algorithm that minimizes inventory and the cost of different patterns. Karaca et al. [27] proposed two solution methods for the biobjective SSP, where the first objective is to minimize trim loss and the second is to minimize the number of welds in a product for quality and sustainability reasons [28]. They proposed CG and B&B algorithms as exact solvers and the dragonfly algorithm (DA) integrated with the constructive heuristic and a heuristic approaches. Finally, they presented comparative results of these two methods regarding solution quality and computational complexity.

Karaca et al. proposed the DA for the pattern generation procedure, stating that, unlike the CSP, metaheuristics are under-utilized in the SSP realm. Well-known metaheuristics in this area are Tabu search [29], simulated annealing (SA) [30–32], ant colony optimization (ACO) [33], and its variants or hybridizations [34], genetic algorithm [35–37], genetic symbiotic algorithm [38], evolutionary programming (EP) [39], and hybrid chemical reaction optimization (CRO) [40].

The studies mentioned above involve numerical implementations and real-world applications of the SSP. Nevertheless, the literature also offers theoretical analyses. Martinovic et al. are considered one of the most important pioneers of theoretical analysis of the SSP literature [4, 5, 19–22, 41]. In addition to the pattern-based SSP model, Martinovic and Scheithauer [20] presented three graph-theory-based models for the SSP. They further investigated the continuous relaxations of these models to prove their equivalences with the pattern-based model. In a latter study, Martinovic and Scheithauer [21] formalized the gap between the continuous relaxation value and the optimal objective function value. They also proposed a modified version of the best-fit algorithm to improve the upper bound of the optimality gap for the divisible case. Moreover, they investigated the proper relaxation concept using the proper pattern set, which gives tighter bounds than continuous relaxation [21]. They analyzed the integer round-down property (IRDP) and modified integer round-down property (MIRDP) and noninteger round-down property (non-IRDP) in discretizing the obtained solutions [21]. In [22], furthermore, Martinovic et al. [5] improved the standard arc flow model, which was previously presented in [20] by incorporating reversed loss arcs and minimizing the number of arcs, which dramatically reduced the execution time. They presented a new theoretical approach based on hypergraph matching to develop a relaxation by evaluating the proper gap for skiving stock instances. They benefited from the polyhedral theory to characterize IRDP instances for the SSP [19].

In parallel to the theoretical analysis, Martinovic et al. [4] also considered the problem of spectrum aggregation for cognitive radio as a real-world application of the SSP. This problem concerns the allocation of the radio network spectrum availability, in which the primary user allocates predetermined portions of a frequency band. Existing bandwidths, or spectrum holes, were too limited to support secondary users' bandwidth needs. They analyzed aggregating free spectrum holes to supply sufficient bandwidth to secondary users under hardware limitations [4]. They compared the standard SSP model based on the Zak, solved by the column generation method, to an arc flow model and an assignment model. Computations showed that the assignment model resulted in a significant reduction of the computational complexity.

The stochastic SSP literature is minimal, and it is possible to use solution methods previously employed in the CSP. Therefore, we also elaborate on the stochastic CSP literature in this section. The majority of the methods implement problem-based heuristics and various stochastic programming approaches. The CG is the most commonly applied method to retrieve an initial, noninteger solution [8, 42, 43]. Alem et al. [8] implemented a CG for a two-stage stochastic programming model where the demand was random in the CSP. Jin et al. [43] implemented a two-stage stochastic integer programming for the CSP that makes inventory replenishment decisions in the first-stage and cutting decisions in the second stage. Moreover, the CG is used for the LP relaxation for the cutting stock problem, and the residual heuristic is used to obtain integer solutions. Another solution methodology for the stochastic CSP by Demirci et al. [42] uses the CG and the L-shaped algorithm. Chauhan et al. [44]'s CG and BB approach was accompanied by both a fast pricing heuristic and a marginal cost heuristic in a stochastic problem where the demand is random. As an alternative method to CG, Beraldi et al. [9] suggested a two-stage stochastic programming model with Lagrangian decomposition and BB to decompose the problem into subproblems. These subproblems are fed into a proposed heuristic in the second stage. Moreover, Sculli [45] considered defects as random variables due to the winding process in the CSP. José Alem and Morubito [46] employed stochastic demand and set-up times for cutting patterns in furniture production. Zanjani et al. [10] presented a two-stage stochastic linear programming (LP) approach for the CSP where yields are random variables with discrete probability distributions. They used the sample-average approximation (SAA) scheme to approximate the problem to avoid high computational time caused by numerous scenarios in stochastic programming.

This study extends Zak's standard pattern-based SSP [1] by including production, set-up, and raw material costs. Having the quantity of each raw material as a decision variable also converts the model into an assortment problem. Moreover, the multiproduct version is adapted to the standard pattern-based SSP model [1, 4]. The main contribution of our study is to handle different sources of uncertainty: the product demand and the waste rate. First, the DA produces skiving patterns. The later stage deals with uncertainty in the SSP by a two-stage stochastic programming model with recourse formulation [11]. Furthermore, we implement an SAA approach [13] to cope with a large number of scenarios and previously applied in extensive problems such as supply chain network-based decision-making [47, 48]. Finally, a recursive solution procedure between the DA and the SAA is developed for the large-sized stochastic SSP under uncertain demand and waste rate.

3. The Stochastic Skiving Stock Problem under Uncertain Demand and Waste Rate

3.1. The SSP Definition and Mathematical Formulation. Basic definitions and the formulation for the SSP [4, 19–21] are presented as follows: $E := (m, l, L, b)$ is the SSP instance, with m is the number of small item types. l is an m-dimensional vector representing the width of each type and is composed of l_i, where $i \in \mathbb{I}$ and $|\mathbb{I}| = m$ and L is large item width [4, 19–21].

Large items with a minimum width of L should be produced during the skiving process. Moreover, b is an m-dimensional vector consisting of b_i which represents the availability of each small item type [4, 19–21]. For the sake of clarity and standardization of terminology, small and large items will be referred to as *items* and *products* throughout the rest of the study. All input data are positive integers ($\mathbb{Z}_+$) and satisfy $L > l_1 > \ldots > l_m$ as a sorted set. Every feasible set of items for constructing a product with a minimum width of L is called a *a feasible pattern* of the E. Any feasible pattern can be described by a nonnegative vector $a = (a_i, \ldots, a_m)^T \in \mathbb{Z}_+^m$, and $a_i \in \mathbb{Z}_+$ is the number (repetition) of ith item in a pattern. Finally, $P(E) := \{a \in \mathbb{Z}_+^m \mid l^T a \geqslant L\}$ represents a feasible set of patterns [4, 19–21].

The minimal pattern is one where the product width is less than the threshold value L if any element is dropped from the pattern. In plain expression, there is no feasible pattern $\tilde{a} \in P(E)$ such that $\tilde{a} \leq a$ holds component-wise ($\tilde{a}_i \leq a_i, \forall i \in \mathbb{I}$). A minimal pattern set (or set of minimal patterns) is denoted by $P^*(E)$ [4, 19–21]. Also, a pattern with index $ja^j \in P^*(E)$ is an exact pattern if $l^T a^j = L$, so every exact pattern is a minimal pattern. x_j is the decision variable indicating the repetition of the pattern j, $a^j = (a_{1j}, \ldots, a_{\text{mJ}})^T$ where $a^j \in \mathbb{Z}_+^m$, where j represents the index of the pattern. To conclude, the objective function of the SSP objective function is defined as follows [4, 19–21]:

$$z^*(E) = \max\left\{ \sum_{j\in\mathbb{J}^*} x_j \;\middle|\; \sum_{j\in\mathbb{J}^*} a_{\text{ij}} x_j \leq b_i, \quad i \in \mathbb{I}, x_j \in \mathbb{Z}_+, j \in \mathbb{J}^* \right\}. \tag{1}$$

Furthermore, a minimal pattern $a^j \in P^*(E)$ which satisfies $a_{\text{ij}} \leq b_i, \forall i, j$ is called a *minimal proper pattern.* In a sense, a minimal proper pattern also meets the item availability constraints for a given production quantity of each product. Else, it is a *nonproper minimal pattern* [4, 19–21]. We extend and define the propriety for the pattern set, as well. A proper minimal pattern set is denoted as $P_P^*(E \mid x_j) := \left\{ a\colon a^j \in P^*(E), \sum_{j\in\mathbb{J}^*} a_{\text{ij}} x_j \leq b_i, i \in \mathbb{I} \right\}$, that is, the patterns in $P_P^*(E)$ can produce a predetermined production amount with the available items.

Furthermore, without loss of generality, $E := (m, l, L, b)$ is extended by including multiple products of various widths, defined as the *multiproduct* case as $E := (m, K, l, L, b)$ where K represents the product type number. k is the index of product type $k \in \mathbb{K} := 1, \ldots, K$. L is no longer a constant but a vector of widths. x_{jk} refers to the amount of pattern j used to produce product k. Then, the formulation is extended as follows:

$$z^*(E) = \max\left\{ \sum_{j\in\mathbb{J}^*} \sum_{k\in\mathbb{K}} x_{\text{jk}} \;\middle|\; \sum_{j\in\mathbb{J}^*} a_{\text{ijk}} x_{\text{jk}} \leq b_i, \quad i \in \mathbb{I}, x_{\text{jK}} \in \mathbb{Z}_+, j \in \mathbb{J}^*, k \in \mathbb{K} \right\}. \tag{2}$$

3.2. Standard Formulation of a Two-Stage Stochastic Programming (SP) Model. There are several applications of the two-stage stochastic programming in literature, such as the air freight hub location and flight routes planning where the demand is random [49], supply chain planning in which the capacity of facilities and the customer demand are random [48], and the sawmill production planning problem in which the yield of production is random [10]. Below, we present the general framework.

Assume that ξ is a random vector that holds realizations of each scenario. The decision variables whose values must be decided before observing the actual values of ξ are called the first-stage decision variables. An instance of such a decision variable is denoted by the vector x. When any scenario occurs, it reveals the complete information on the random variables in ξ, and their values become known. Any decision after ξ is known as a second-stage decision variable or the recourse action, denoted by vector y. It is important to emphasize that the recourse action depends on both the first-stage decision variables and the outcomes of random variables. In other words, the two-stage stochastic programming recourse action model can respond to each possible outcome of random variables using the second-stage decisions (the recourse action).

The general formulation of A two-stage stochastic programming (SP) model with recourse general formulation is given as follows [11–13]:

$$\min_x C^T x + \mathbb{E}_\xi Q(x,\xi), \tag{3}$$

$$\begin{aligned} s.t. \quad Ax &= b, \\ x &\geq 0, \end{aligned} \tag{4}$$

where $Q(x,\xi) = \min\{q^T y | \mathrm{W}y = h - \mathrm{T}x, y \geq 0\}$. The random vector ξ is formed by the q^T, h^T, and T components, where $\mathbb{E}_\xi$ is mathematical expectation according to ξ where T is the technology matrix, h is the right-hand-side values, W is the recourse matrix, and q^T is the penalty cost vector of recourse decisions [11].

In the general formulation of the two-stage stochastic programming in equation (3), each value combination for the random variables in ξ corresponds to the scenario $s \in \mathbb{S} := 1, \ldots, S$ with the probability P_s. Therefore, equation (3) can be written in the form of the stochastic model as follows [11]:

$$\min_x C^T x + \sum_{s\in\mathbb{S}} P_s Q^s(x,\xi), \tag{5}$$

where $Q^s(\cdot)$ refers to the value of the $Q(x,\xi)$ under scenario s.

3.3. Mathematical Formulation of the Stochastic SSP under Demand and Waste-Rate Uncertainties. In the nomenclature, we solely list the notation used for the stochastic SSP model. It should be noted that the different phases of the solution methodology will also use their own notations as described in Section 4.

We transform the original SSP into a cost-minimization problem by including the production, raw material, set-up, overproduction, and underproduction costs. Furthermore, the demand D_k and the approved product rate Υ_k are the random variables for each product $k = 1, \ldots, K$. Υ_k, also known as the yield efficiency, is the rate of approved products after discarding the production waste. In this study, we will assume the same random yield efficiency for all products, which reduces Υ_k to Υ, resulting in a total of $K + 1$ random variables. Each combination of the values for D_k and Υ corresponds to a scenario indexed by $s \in \mathbb{S} := 1, \ldots, S$ with the probability P_s such that $P_s \geq 0$ and $\sum_{s\in\mathbb{S}} P_s = 1$. Finally, each scenario realization for every random variable is represented as $D(s) = (d_{s1}, \ldots, d_{sK})$ and $\Upsilon(s) = \upsilon_s$. Both random variables constitute the base for the mathematical model in the two-stage stochastic programming with recourse.

We partition the decision variables of the stochastic model into two parts: the first-stage and second-stage decision variables. The frequency of each pattern for each product denoted by the matrix $\mathbf{x}$ is a first-stage decision variable composed of x_{jk}'s. Another first-stage decision variable is the raw material needed, denoted by the vector r and composed of $r_i s$. Finally, the last first-stage decision variable is the vector y, denoting the set-up change decision $y_j, \forall j$. The values of $\mathbf{a}$ and Δ are obtained using the DA before solving the SP. The $\mathbf{a}$ matrix has the values of a_{ij}, and the Δ matrix holds the δ_{jk} values. These values fed into the first stage of the SP as parameters. The overproduction and the underproduction amounts $(q^+_{\mathrm{sk}}, q^-_{\mathrm{sk}})$ are the second-stage decision variables and depend on the scenario and the first-stage decisions. Then, we can construct the first-stage objective function as follows:

$$\min \quad \sum_{i\in\mathbb{I}} C^R_i r_i + \sum_{k\in\mathbb{K}} \sum_{j\in\mathbb{J}^*} C^O y_j + C^{\mathrm{Pr}} x_{\mathrm{jk}}. \tag{6}$$

Moreover, $s \in \mathbb{S}$, Q^s is the $Q(\cdot)$ function provided in equation (5) for scenario s and is defined as follows:

$$Q^s(x,\xi) = \min \quad \sum_{k\in\mathbb{K}} C^H_k q^+_{\mathrm{sk}} + C^B_k q^-_{\mathrm{sk}}. \tag{7}$$

Finally, by using equations (5) and (6), and the nomenclature, the deterministic equivalent of the stochastic SSP model with the instance $E := (m, K, l, L, b)$ is given through equations (8)–(17), and we obtain the following:

$$\min z_{\mathrm{SP}} = \sum_{i\in\mathbb{I}} C^R_i r_i + \sum_{k\in\mathbb{K}} \sum_{j\in\mathbb{J}^*} C^O y_j + C^{\mathrm{Pr}} x_{jk} + \sum_{s\in\mathbb{S}} P_s \sum_{k\in\mathbb{K}} C^H_k q^+_{\mathrm{sk}} + C^B_k q^-_{\mathrm{sk}}, \tag{8}$$

$$\text{s.t.} \quad \sum_{k\in\mathbb{K}} \sum_{j\in\mathbb{J}^*} a_{\mathrm{ij}} x_{\mathrm{jk}} = r_i \leq b_i, \quad \forall i \in \mathbb{I}, \tag{9}$$

$$q^+_{\mathrm{sk}} - q^-_{\mathrm{sk}} = \upsilon_s \sum_{j\in\mathbb{J}^*} x_{\mathrm{jk}} - d_{\mathrm{sk}}, \quad \forall s \in \mathbb{S}, k \in \mathbb{K}, \tag{10}$$

$$x_{\mathrm{jk}} \leq M_k y_j \delta_{\mathrm{jk}}, \quad \forall j \in \mathbb{J}^*, k \in \mathbb{K}, \tag{11}$$

$$x_{\mathrm{jk}} \in \mathbb{Z}_+, \quad \forall j \in \mathbb{J}^*, k \in \mathbb{K}, \tag{12}$$

$$a_{\mathrm{ij}} \in \mathbb{Z}_+, \quad \forall i \in \mathbb{I}, j \in \mathbb{J}^*, \tag{13}$$

$$r_i \in \mathbb{Z}_+, \quad \forall i \in \mathbb{I}, \tag{14}$$

$$q^+_{\mathrm{sk}}, q^-_{\mathrm{sk}} \geq 0, \quad \forall s \in \mathbb{S}, k \in \mathbb{K}, \tag{15}$$

$$y_j \in \{0, 1\}, \quad \forall j \in \mathbb{J}^*, \tag{16}$$

$$\delta_{\mathrm{jk}} \in \{0, 1\}, \quad \forall j \in \mathbb{J}^*, k \in \mathbb{K}. \tag{17}$$

The objective function is given in equation (8) (z_{SP}) minimizes both the first-stage and the second-stage costs. The first-stage costs are composed of the following components:

(i) Raw material cost: the cost of items used to form products involves the costs of generating leftovers or purchasing raw materials.

(ii) Set-up cost: if a pattern is used in the skiving process, the set-up cost for that specific pattern is incurred. However, since recent skiving machines are fully automated, the differences between set-up times of each pattern are assumed trivial. Therefore, we assume that the set-up cost is fixed and does not change with a specific pattern.

(iii) Production cost: an item roll naturally has two dimensions: width and length. We consider each item's variable width and fixed length, resulting in the 1D-SSP. Therefore, the production time and cost for the skiving process to form every product are assumed to be fixed.

The second-stage costs include overproduction and underproduction, lost sales or overtime production costs, and the costs of procuring products from other sellers to meet the demand. Constraint (9) multiplies the number of pattern repetitions by the number of items in each pattern to ensure enough items are available for the amount produced. Constraint (10) balances the overproduction or underproduction amount at the second stage due to the first-stage production decisions and the scenarios. The set-up constraint given in equation (11) states that if a pattern is used, the set-up cost is related to this pattern. Moreover, it can be used up to M_k times for product k, where M_k is an upper bound for the number of replications of pattern j in product k. However, the waste rate might require extra production. Therefore, a reasonably greater value than the maximum demand can validate the value for M_ks.

The set-up cost is incurred for every pattern change. As aforementioned, if multiple products can be produced using the same pattern, we can avoid excessive set-up costs by setting this pattern once and producing all products simultaneously. This way, we do not need to change patterns and pay the set-up cost only once. For this purpose, we construct a pattern pool that unites all patterns used in all products. An auxiliary and dependent binary variable δ controls the assignment of patterns to products.

Constraints (12)–(15) are integrality and positivity constraints, and constraints (16) and (17) impose that y_j and δ_{jk} are binary variables, respectively. We determine the values of the decision variables a_{ij} in constraint (9) and δ_{jk} in constraint (11) in the first phase. Since the values of both variables are determined in the first phase, we feed them as parameters for the second phase. Therefore, the final model becomes a stochastic mixed-integer programming (SMIP) model. We will analyze the performance of the DA using a larger numerical example in the next section.

3.4. Deterministic Counterpart of the First-Stage Model.

We must check the propriety of $P_P(E)$ throughout the procedure. In other words, whenever we make a production decision, we must check if we can produce this amount with the patterns on hand. If not, we can opt for underproduction or search for new patterns to avoid losing sales.

For this checkpoint, we implement the deterministic counterpart of the SSP with a small modification. Without the loss of generality, the deterministic counterpart of the SMIP model given in equations (8)–(17) can be rewritten using the first-stage variables and costs. This model minimizes the first-stage cost at a given production amount as follows:

$$\min z_{\det} = \sum_{i\in\mathbb{I}} C_i^R r_i^+ \sum_{k\in\mathbb{K}} \sum_{j\in\mathbb{J}^*} C^O y_j + C^{\Pr} x_{\mathrm{jk}} + \Psi_k q_k^-, \tag{18}$$

$$s.t \quad \sum_{j\in\mathbb{J}^*} \sum_{k\in\mathbb{K}} a_{\mathrm{ij}} x_{\mathrm{jk}} = r_i \le b_i, \quad \forall i \in \mathbb{I}, \tag{19}$$

$$\sum_{j\in\mathbb{J}^*} x_{\mathrm{jk}} + q_k^- \ge \mathrm{upperBound}_k, \quad \forall k \in \mathbb{K}, \tag{20}$$

$$x_{\mathrm{jk}} \le M_k \delta_{\mathrm{jk}} y_j, \quad \forall j \in \mathbb{J}^*, k \in \mathbb{K}, \tag{21}$$

$$x_{\mathrm{jk}}, r_i, q_k^- \in \mathbb{Z}_+, y_j \in \{0,1\}, \delta_{\mathrm{jk}} \in \{0,1\}, \quad \forall j \in \mathbb{J}^*, k \in \mathbb{K}, \forall i \in \mathbb{I}. \tag{22}$$

In this deterministic model as shown in equations (18)–(22), the indices, the first-stage variables, and the first-stage costs are almost the same as the original stochastic problem given throughout in equations(8)–(17). The main difference with the stochastic counterpart is the introduction of the production upper bound for each product denoted by $\mathrm{upperBound}_k$ in equation (20). This upper bound represents a target production amount (usually dependent on the demand). The variable q_k^- tracks the lack of each product, and a penalty cost, Ψ_k, represents a large number such as $\Psi_k = C_k^B * 100000000, \forall k \in \mathbb{K}$ forces q_k^- to be zero. In other words, minimizing this objective function forces the total production of each product to be equal to the upper bound. This deterministic model is used iteratively in the algorithm to check whether the minimal pattern set generated by the DA can satisfy a given target production amount. If not, the algorithm recursively triggers the DA to generate additional minimal patterns until the target production amount (upper bound) is satisfied. Briefly, this process controls if additional patterns are needed, and in this way, it prepares an efficient pattern set for the next phase, which may improve stochastic solutions.

4. The Proposed Solution Methodology

In this section, we develop an iterative two-phase solution methodology for the stochastic SSP. An overview of the methodology is presented in Figure 1. In the first phase of the algorithm, we implement the dragonfly algorithm (DA) proposed in [15] to produce the minimal pattern set $P^*(E)$. This minimal pattern set $P^*(E)$ is fed into the second phase. In the second phase, the SMIP presented in Section 3.3 is solved by the SAA method [13]. This process provides candidate solutions and an upper bound for the production amount. We first present each module separately and later define the two-phase methodology that integrates these algorithms, including the DA results in the proposed methodology, producing a heuristic solution.

4.1. Dragonfly Algorithm Implementation Steps. The dragonfly algorithm (DA) is a method based on swarm intelligence. It employs nonlinear Lévy flights in the search space. Literature refers to the travelling salesman problem (TSP) [50], optimal dynamic scheduling of tasks [51], path planning optimization for mobile robots, moreover, optimization of 0-1 all knapsack problem versions [52], feature selection problems [53], graph coloring problems [54], and wind-solar-hydropower scheduling optimization by employing multiobjective version [55]. It performs better than PSO and GA when discrete problems are considered [15]. In this study, the DA, which was modified by Karaca et al. [27], is used for the generation of patterns for each product on a separate basis by following the steps.

4.1.1. Preprocessing. The width of item types is used in this process.

Step 1: items no longer available are excluded from the set $\mathbb{I}$.

Step 2: the amount of each item i necessary for the construction of a product k (n_{ik}) is calculated such that

$$n_{\mathrm{ik}} = \left\lceil \frac{L_k}{l_i} \right\rceil. \tag{23}$$

Step 3: the item set is extended by repetition of each item ($n_{\mathrm{ik}}-1$) times to get the expanded and sorted item set $\mathbb{I}^+$, so that there are n_{ik} of each item. The composition of the updated item set is shown in Figure 2.

Assume $I_+ = |\mathbb{I}^+|$. This is the number of items in the extended set. Furthermore, $\mathrm{cl}_h = \sum_{i'=1}^{h} l_{i'}$. In other expressions, cl_h is the accumulated length of the items from the first to the h^{th} position.

In the following, the extended set $\mathbb{I}^+$ is introduced into subsequent stages of the main dragonfly algorithm framework.

4.1.2. Initializing. The algorithm parameters are initiated. The change in position change (Δpos) is started as a zero matrix, i.e., $\mathbf{0}_{\mathrm{NoD},\widetilde{I}}$ where NoD is representative of the dragonfly number. The maximum iteration number (MaxIt) is returned.

Step 1: Each dragonfly has a dimension of I_+, and each dragonfly value is a random number uniformly distributed between 0 and 1. Generating this matrix is

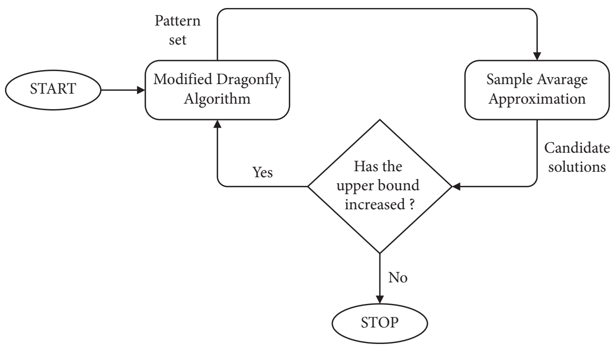
Pattern set
START
Modified Dragonfly Algorithm
Sample Avarage Approximation
Candidate solutions
Has the upper bound increased ?
Yes
No
STOP

Figure 1: Recursive two-phase algorithm for the stochastic SSP.

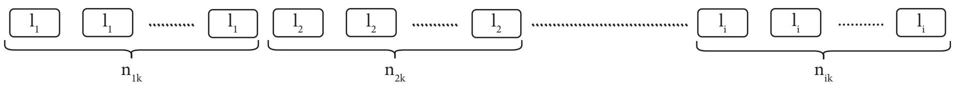

Figure 2: Extended and ordered item sets.

called as $\text{pos}_{\text{NoD},I_+}$. Each row of $\text{pos}_{\text{NoD},I_+}$ denotes a dragonfly, that is, a solution.

Step 2: For each dragonfly, the value of the objective function, the trim loss, is calculated. Let b be the b^{th} dragonfly's index. The objective function of the b^{th} dragonfly is calculated in the following way:

(1) Sort in decreasing order the b^{th} row of the $\text{pos}_{\text{NoD},\widetilde{I}}$ matrix.
(2) We now have the index vector for the values that have been sorted. This order is denoted as $\overrightarrow{[h]}$.
(3) The $\text{cl}_{[h]}$ values are summed up until L_k is obtained such that $\text{cl}_{[h-1]} < L_k$ and $\text{cl}_{[h]} \geq L_k$. That is, it will skip items until it reaches the desired length for product k.
(4) The trim loss is computed as $\sum_{h=1}^{h'} \bar{l}_{[h]} - L_k$.

Otherwise specified, the indexes of the values of the dragonfly are sorted in descending manner. Next, removing elements is started in this order until the product length is reached. As an illustration, assume the starting set is $\{6, 5, 3\}$, the extended and sorted sets $\mathbb{I}^+ = \{6, 6, 5, 5, 3, 3, 3\}$, and the product width $L_k = 9$. Suppose a dragonfly has the vector $[0.35, 0.45, 0.21, 0.76, 0.87, 0.98, 0.07]$ values. In this case, the sorted index vector is $\overrightarrow{[h]} = [6, 5, 4, 2, 1, 3, 7]$, meaning that the sixth dimension of the dragonfly is the largest and the fifth dimension of the dragonfly is the second largest. So $\text{cl}_{[1]} = l_6 = 3$, meaning that if the first order item is used, the length of the final product would be three units, which is less than L_k, so the skiving process goes on with the second-order item. If the next item is skived, $\text{cl}_{[2]} = l_{[1]} + l_{[2]} = l_6 + l_5 = 3 + 3 = 6$ is obtained. If the length obtained is still less than the desired threshold value for the product length, continue the skiving process with $\text{cl}_{[3]} = l_{[1]} + l_{[2]} + l_{[3]} = l_6 + l_5 + l_4 = 3 + 3 + 5 = 11$. The threshold product length of L_k is satisfied by these three skived items. So, the skiving process is stopped. For example, a dragonfly consisting of the vector $[0.35, 0.45, 0.21, 0.76, 0.87, 0.98, 0.07]$ denotes a product consisting of two elements of length 3 and one element of length 5, giving a product of length 11. $\text{cl}_{[3]} - L_k = 11 - 9 = 2$ is the trim loss of this product. An essential tip is to maintain the positions of the dragonflies between 0 and 1 throughout the algorithm. This adjustment prevents extreme position values.

The trim loss of each dragonfly is calculated, and the objective function value vector, $\overrightarrow{f}_{\text{NoD}}$, is obtained after decoding each dragonfly in the swarm. Each value of the vector is an indication of the objective function of a particular dragonfly.

Step 3: The food source and the enemy are updated as

$$\text{food} = \left\{\text{pos}_{b,*}\colon\ f_d = \min_{b'} \overrightarrow{f}, b' = 1, \ldots, \text{NoD}\right\},$$
$$\text{enemy} = \left\{\text{pos}_{b,*}\colon\ f_d = \max_{b'} \overrightarrow{f}, b' = 1, \ldots, \text{NoD}\right\}. \tag{24}$$

Step 4: Suppose the instantaneous iteration is t. The neighborhood border is updated as t/MaxIt. For each dragonfly, if there are no dragonflies in the neighborhood, a Lévy flight is used as described as follows [15]:

$$\Delta\text{pos}_{b,*}(t) = \text{pos}_{b,*} \cdot 0.01 \frac{r_1 A}{r_2^{1/B}}, \tag{25}$$

where r_1 and r_2 are random values between 0 and 1 in which B is a customized parameter by the user and A is calculated as

$$A = \left[\frac{\Gamma(a+B)\sin(\pi B/2)}{\Gamma[(1+B)/2]B2^{(B-1)/2}} \right]^{1/B}. \quad (26)$$

If there is at least one dragonfly in the neighborhood, then a weighted composite separation, alignment, cohesion, food source approach, and enemy escape vectors are computed, and the dragonfly's position change is as follows:

$$\Delta \mathrm{pos}_{b,*}(t) = \alpha \mathrm{Sep}_b + \beta \mathrm{Aln}_b + \gamma \mathrm{Coh}_b + \eta \left(\mathrm{food} - \mathrm{pos}_{b,*}\right) + \epsilon \left(\mathrm{enemy} + \mathrm{pos}_{b,*}\right) + \omega \Delta \mathrm{pos}_{b,*}(t-1), \quad (27)$$

where $\alpha, \beta, \gamma, \eta, \epsilon$ are given and tuned coefficients, ω is the inertia rate. food and enemy are the dragonfly positions with the best and worst objective function values. Suppose that Λ_b is the set of dragonflies in the neighborhood of the b^{th} dragonfly and λ_b is the number of neighboring dragonflies. The separation factor (Sep_b) prevents the dragonflies from colliding. The alignment (Aln_b) and cohesion (Coh_b) factors allow the dragonflies to use the search space with similar speeds and positions. The following formulae are used to calculate these components separately [15]:

$$\mathrm{Sep}_b = \sum_{b' \in \Lambda_b} \mathrm{pos}_{b,*} - \mathrm{pos}_{b',*},$$
$$\mathrm{Aln}_b = \frac{\sum_{b' \in \Lambda_b} \Delta \mathrm{pos}_{b',*}(t-1)}{\lambda_b}, \quad (28)$$
$$\mathrm{Coh}_b = \frac{\sum_{b' \in \Lambda_b} \mathrm{pos}_{b',*}}{\lambda_b} - \mathrm{pos}_{b',*}.$$

Step 5: The change in position is updated by means of the velocity and the inertia of the change in position such that

$$\mathrm{pos}_{b,*}(t) = \mathrm{pos}_{b,*}(t-1) + \Delta \mathrm{pos}_{b,*}. \quad (29)$$

Dragonfly positions are held between 0 and 1, so if the dragonfly position exceeds these limits in any dimension, the position will adjust to the closest border.

Step 6: Steps 2–5 are iterated until the maximum iterations are achieved.

The result of the algorithm is the minimum trim loss dragonfly, but it produces an extended and sorted version of the items. Each of the best dragonflies with the same objective function value constitutes a pattern.

4.1.3. Postprocessing. This process breaks down the pattern and calculates the amounts of items in each pattern. Assume that the dragonfly given in the illustrative example is the best dragonfly and the outcome of the algorithm. This pattern uses two items with a width of 3 and one with a width of 5. For the initial set of items with lengths $\{6, 5, 3\}$, the best dragonfly is decoded as $[0, 1, 2]^T$.

It should be kept in mind that different extended dragonflies can result in the same pattern. For example, let another dragonfly be $[0.15, 0.25, 0.11, 0.56, 0.77, 0.08, 0.89]$. This dragonfly also uses two items with length 3 and one item with length 5.

The DA creates a minimal pattern pool to be used as a parameter in the SP model as shown in equations (8)–(17). The produced minimal pattern pool is not necessarily proper. The propriety of the pattern pool is determined as a result of the second phase, the sample-average approximation (SAA) algorithm.

4.2. The Sample-Average Approximation (SAA) Algorithm. SAA is implemented for large-size stochastic problems that cannot be solved easily by using exact solution methods because of the large number of scenarios. SAA approximates the objective function by using generated samples of scenarios. The SAA generated N realizations of $(n = 1, 2, \ldots, N)$ of the random vector ξ. These realizations are denoted by $\xi_1, \ldots, \xi_N$. Then, the expectation $\mathbb{E}_{\xi} Q(x, \xi)$ is approximated by the sample-average function $N^{-1} \sum_{n=1}^{N} Q(x, \xi_n)$. Finally, the original problem (8)–(17) is approximated by the SAA problem [10, 16, 56], where $Q(x, \xi)$ is the objective function involving decision variables x and random variables ξ. According to Saphiro [57], SAA provides good convergence and robust statistical inferences, including analysis of error, stopping rules, and validation, and it is easy to implement with commercial software. The steps of the SAA are given as follows [56]:

Initialize: Generate G, $(g = 1, 2, \ldots, G)$, random samples from the distribution of random variable ξ, each of them is independent and identically distributed and has a sample size N where $|N_g| = N$. Also, generate a sufficiently large reference sample where $N' \gg N$.

Step 1: Solve the problem (30) and optimal objective function value v^g and candidate solution x^g for each g.

$$\min_x \quad C^T x + \frac{1}{\left|N_g\right|} \sum_{n=1}^{N} Q(x, \xi_n). \quad (30)$$

Step 2: Compute $\overline{v}^G$ average (31) which is the unbiased estimator of the objective function of the original problem v^* and variance $\widehat{\sigma}^2_{\overline{v}^G}$ (32) of the objective function values obtained in the first step.

$$\overline{v}^G := \frac{1}{G}\sum_{g=1}^{G} v_g, \tag{31}$$

$$\widehat{\sigma}^2_{v^G} := \frac{1}{G(G-1)}\sum_{g=1}^{G}\left(v_g - \overline{v}^G\right)^2. \tag{32}$$

Step 3: Solve the problem g times with sample size N' (33) by using each candidate's solution $\overline{x}^g$ of each g in order to find $\widehat{v}^g$ for each and calculate $\widehat{\sigma}^2_{\widehat{v}^g}$ (34).

$$\widehat{v}^g := \min_x \quad C^T\overline{x}^g + \frac{1}{\left|N'\right|}\sum_{n=1}^{N'} Q\left(\overline{x}^g, \xi_n\right), \tag{33}$$

$$\widehat{\sigma}^2_{\widehat{v}^g} := \frac{1}{N'\left(N'-1\right)}\sum_{n=1}^{N'}\left(C^T\overline{x}^g + Q\left(\overline{x}^g, \xi_n\right) - \widehat{v}^g\right). \tag{34}$$

Step 4: Compute the estimation of the optimality gap for candidate solution $\mathrm{gap}_g(\overline{x}^g)$ (35) and the variance of the optimality gap $\widehat{\sigma}^2_{\mathrm{gap}_g}$ (36) to analyze the quality of the candidate solution.

$$\mathrm{gap}_g\left(\overline{x}^g\right) = \widehat{v}^g - \overline{v}^G, \tag{35}$$

$$\widehat{\sigma}^2_{\mathrm{gap}_g} = \widehat{\sigma}^2_{\widehat{v}^g} + \widehat{\sigma}^2_{v^G}. \tag{36}$$

Step 5: Select the x^g as an approximate solution of the SAA problem (x^{SAA}) that provides the best $\widehat{v}^g$, i.e., $x^{\mathrm{SAA}} = \mathrm{argmin} v^{SAA}$ where $v^{\mathrm{SAA}} = \min_{g=1,\ldots,G}\widehat{v}^g$.

According to Ahmed and Saphiro [58], the lower bound for v^* is provided by $\overline{v}^G$, and the upper bound for v^* is obtained by $\widehat{v}^g$. The optimal objective function value of the SAA problem converges to the optimal objective function value of the original problem in equation (3) with probability 1.00 as sample size N goes to infinity ($N \longrightarrow \infty$) [59]. A larger sample size ensures a stricter approximation. However, it also increases the computational complexity [57]. Therefore, using small independent and identically distributed (i.i.d) samples is more efficient than a large sample size. The SAA algorithm is based on this principle. The complexity increases exponentially as the sample size increases, especially for the integer problems [60]. This increase in complexity eventually causes a trade-off between the approximation quality and computational complexity.

4.3. The Two-Phase Solution Methodology. As shown in Figure 1, we implement an iterative approach in two phases: (i) the DA and (ii) the SAA algorithm. In a nutshell, the DA initially uses the information of items and products to return a set of patterns. This set of patterns is called a minimal pattern set. This pattern set is fed into the SAA to obtain the upper bounds of production amounts, as shown in Figure 3. This upper bound also serves as a reference for the highest production amount. If the patterns found cannot satisfy this upper bound due to the availability constraints, then the pattern set cannot be deemed proper. We emphasize that if predetermined production levels can be satisfied by using generated minimal patterns, this minimal pattern set is called a proper set. The DA is rerun to produce new patterns to obtain a proper minimal pattern set. These new patterns are, again, fed into the SAA. This recursion continues until convergence is obtained. We define convergence as no further change in two aspects: (i) the minimal pattern set and (ii) the upper bound of production.

The pseudocode of the methodology is given in Algorithm 1. Below, we define the parameters and variables used in the pseudocode:

z_{SP}: the objective function value of the stochastic model given through equations (8)–(17)

x^g_{jk}: the candidate's solution x_{jk} (the amount of pattern j in product k) of sample g, obtained by solving for z_{SP}, $g = 1, \ldots, G$

$\mathrm{prodLevel}_k(g)$: the total production amount for product $k \in \mathbb{K}$ obtained by $\sum_{j\in\mathbb{J}} x^g_{\mathrm{jk}}$ for sample g, $g = 1, \ldots, G$

$\mathrm{maxProdLevels}_k$: the maximum total production for product $k \in \mathbb{K}$ among all samples $\max_{g=1,\ldots,G} \mathrm{prodlevel}_k(g) = \max_{g=1,\ldots,G}\sum_{j\in\mathbb{J}} x^g_{\mathrm{jk}}$

$\mathrm{stepsize}_k$: the expansion amount in the search space for product $k \in \mathbb{K}$

$\mathrm{upperBound}_k$: the extended maximum production amount for product $k \in \mathbb{K}$ calculated by summing $\mathrm{stepsize}_k$ and $\mathrm{maxProdLevels}_k$

$z_{\det}$: the objective function of deterministic counterpart model equations (18)–(22)

output_k: the total production for product $k \in \mathbb{K}$ obtained through the optimal solution $\sum_{j\in\mathbb{J}} x^*_{\mathrm{jk}}$ of $z_{\det}$ equations (18)–(22)

We explain the detailed steps of the proposed approach as follows.

4.3.1. Generation of the Initial Minimal Pattern Set. In the proposed approach, the DA initially obtains a pattern set $P^*_k(E)$ for product k, $k \in \mathbb{K}$. It searches the patterns with the minimal trim loss, the extra width of the product that exceeds the width threshold, L_k, for each product. A pattern can be minimal for at least one product; some patterns can be common among multiple products. Using one pattern for multiple products can decrease the set-up cost for each pattern change during production. Therefore, we aggregate all patterns for all products into a minimal pattern set pool, $P^*(E) = \cup_{k\in\mathbb{K}} P^*_k(E)$. Additionally, we generate a binary pattern-product matrix, Δ, to tabulate the matches between patterns and products. $\delta_{\mathrm{jk}} = 1$ if pattern j is a minimal pattern for product k, and 0 otherwise. The outputs of DA are $P^*(E)$ and Δ. Both outputs of this phase serve as input parameters for the second phase. Due to the wordiness of the term and in the interest of simplification, we will refer to a "minimal pattern set" as simply a "pattern set" further on.

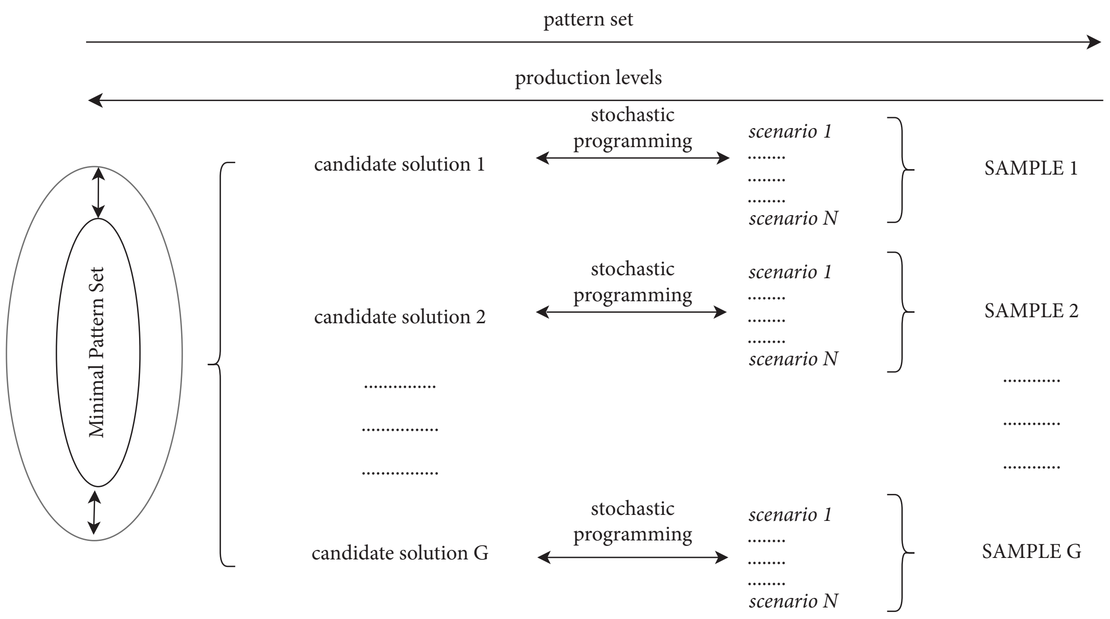

FIGURE 3: The recursion between the DA and the SAA.

```
 (1) **START**;
 (2) Determine parameter values and descriptors;
 (3) Set $b_i' = b_i, b_i = \infty, \forall i$, iteration = 1, $\text{maxProdLevels}_k(0) = 0$;
 (4) Run the DA to generate the pattern set $P^*(E)$ that has a minimum trim loss;
 (5) Run the SAA submodule;
 (6) **while** $\text{maxProdLevels}_k(\text{iteration}) > \text{maxProdLevels}_k(\text{iteration} - 1)$ **do**
 (7)   **while** $\forall k, \text{upperBound}_k > \text{output}_k$ **do**
 (8)     Run the DA to generate the additional pattern set $P'(E)$ that has a minimum trim loss;
 (9)     $P^*(E) \coloneqq P^*(E) \cup P'(E)$;
(10)     solve deterministic min SSP using $P^*(E)$; $x_{\text{jk}} \coloneqq \text{argmin}(z_{\text{det}})$ equations (18)–(22);
(11)     Compute $\forall k, \text{output}_{\text{k}} = \sum_{j \in \mathbb{J}} x_{jk}$
(12)   **end**
(13)   Set minimal proper pattern pool $P_p^*(E) \coloneqq P^*(E)$;
(14)   $b_i = b_i', \forall i$, iteration = iteration + 1;
(15)   Run the SAA submodule (Section 4.2);
(16) **end**
(17) **STOP**;
(18) **SAA Submodule**
 (1)   Run the SAA (Section 4.2), $x_{\text{jk}}^g \coloneqq \text{argmin}_{g=1,\ldots,G}(z_{SP})$, equation (30), using $P^*(E)$ or if available, $P_p^*(E)$;
 (2)   Obtain candidate production amounts for each sample; $\forall g$, $\text{prodLevel}_k(g) = \sum_{j \in \mathbb{J}} x_{\text{jk}}^g$;
 (3)   Select maximum production amount for each product; $\forall k\, \text{maxProdLevels}_k = \max(\text{prodLevel}_k(g))$;
 (4)   Compute upperBound using step size; $\forall k$, $\text{upperBound}_k = \text{maxProdLevels}_k + \text{stepsize}_k$;
 (5)   Solve deterministic min SSP using $P^*(E)$; $x_{\text{jk}} \coloneqq \text{argmin}(z_{\text{det}})$;
 (6)   Compute $\forall k, \text{output}_k = \sum_{j \in \mathbb{J}} x_{\text{jk}}$;
 (7)   **end**
```

ALGORITHM 1: The two-phase solution methodology.

4.3.2. Obtaining the Initial Production Amounts. As aforementioned, the mathematical model receives $P^*(E)$ and Δ as parameters. If we run the SAA algorithm with this pattern set, we reach the optimal solution for only this pattern set. A different pattern set could return a different optimal result for this particular set. Therefore, the initial pattern set does not provide direct answers to the following questions:

(1) Does the pattern set involve the pattern combinations that produce the products at a global optimal cost?

(2) How does the global optimal solution compare to the optimal solution of this pattern set?

(3) Are the item availabilities enough to produce either optimal amount?

Therefore, the quality and the sufficiency of this initial pattern set still need to be discovered. For this purpose, we first calculate a sufficiently high reference production amount for all products. As the iterations continue, we observe the changes in the production amounts and the growth in the pattern set. Based on these changes, we can clarify the answers to the questions above throughout iterations.

Recalling the definition, for a pattern set to be proper, the patterns in a set should have sufficient availability to produce given target production amounts. A nonproper pattern set indicates that the patterns from the DA are insufficient, and we should generate more patterns to produce the target production amount. Therefore, it is only possible to comment on $P^*(E)$ being proper with a target production amount.

To this end, the first recursion of the SAA (equations (8)–(17)) solves a relaxed model with no availability constraints (by ignoring the constraint given in equation (9)) to obtain initial reference production amounts for each sample as presented in Figure 3. In fact, the constraint relaxation method is applied for the problem to obtain the lower bound of expected global optima for the cost function, and the results reflect the optimal production amounts given the pattern set on hand without availability constraint [61]. We generate G number of samples, each with a sample size N, to apply the SAA approach that solves the SMIP presented in Section 3.3. Each sample may result in a different solution (i.e., varying optimal production amount). Consequently, we may have different production amounts (candidate solutions) for each sample denoted by $\text{prodLevel}_k(g)$. $P^*(E)$ being proper (i.e., identical to $P_p^*(E)$) means that it constitutes a proper pattern set for all scenarios and samples. If $P^*(E)$ can produce the highest production amount (maxProdLevels_k), then it can produce all candidate solutions. Therefore, as the initial reference level, we calculate $\text{maxProdLevels}_k = \max_{\{g:\, g=1,\ldots,G\}} \text{prodLevels}_k(g)$. Figure 4(a) shows such production levels for two products. Figure 4(a) displays the maximum production levels for two products.

4.3.3. Calculation of the Upper Bounds Using the Step Size. The previous step calculates an optimal production amount for each product based on the pattern set on hand. Three possibilities exist regarding the quality of this solution: (i) the items cannot produce the optimal amount for this pattern set due to their availabilities, new patterns may or may not improve the solution, (ii) the available items can produce the optimal amounts, but the solution could be improved if there were more patterns available, or (ii) the available items can produce the optimal amounts, and it is indeed the optimal solution, and therefore, any pattern changes would not affect the solution. In either possibility, new patterns must be generated to observe which case the solution fits. We construct a trigger for new pattern generation to understand if the skiving process requires more patterns than we have. This trigger entails increasing the maximum production amounts by a step size parameter. The step size is a preventative measure, especially when the availabilities are sufficient to produce the upper bounds. This iterative numerical method can be explained as the stepping approach to convergence to the expected global optima by finding better solutions in the search space. Briefly, producing the upper bounds indicates that more patterns could extend the search space and find a more affordable solution than the current one. Since the step size parameter is determined experimentally, it is better to run the algorithm many times with different step size parameters to avoid escaping better solution points (see Figure 4(b)) for a snapshot of the production amount and the upper bound and (see Figure 5) for changes between two consecutive iterations.

4.3.4. Checking for the Propriety of the Pattern Set (The Inner While Loop in Algorithm 1). In the next step, we include the item availability constraint in equation (9) to check whether the pattern set on hand can produce this upper bound. In other words, the propriety of $P^*(E)$ is checked for each upper bound by using the deterministic model equations (18)–(22) as presented in Section 3.4. The deterministic model itself is not used to solve the original stochastic problem, but it aids in deciding whether producing the upper bound is feasible and triggers the DA to generate new patterns.

Two possible outcomes exist: the items can or cannot produce the upper bound. The DA is rerun to produce more patterns if the available items cannot produce the upper bound. In Figure 5, the item availabilities cannot produce the upper bounds, and the availabilities lead to another solution.

While generating additional patterns, denoted by the set $P'(E)$, the new patterns should not involve any unavailable items. Similarly, the DA should not reproduce any already-existing patterns. For this reason, we eliminate any information regarding unavailable items from the inputs and only present information related to items that are still available. This adjustment contributes to the generation of different pattern configurations. The DA produces the pattern set $P'(E)$ for every product, and the new minimal pattern set is joined to the pool as $P^*(E) \coloneqq P^*(E) \cup P'(E)$. Similarly, the pattern-product matrix Δ is updated by adding the new patterns. The pattern generation recursion continues until either one of the following conditions is satisfied:

(i) The DA has produced numerous patterns, but the production of the upper bound is infeasible. The total width of the remaining items is less than the product width, so the pattern set is already exhaustive.

(ii) The most recent $P^*(E)$ can produce the upper bounds, and the minimal pattern set becomes proper. In this case, we move on to the next step.

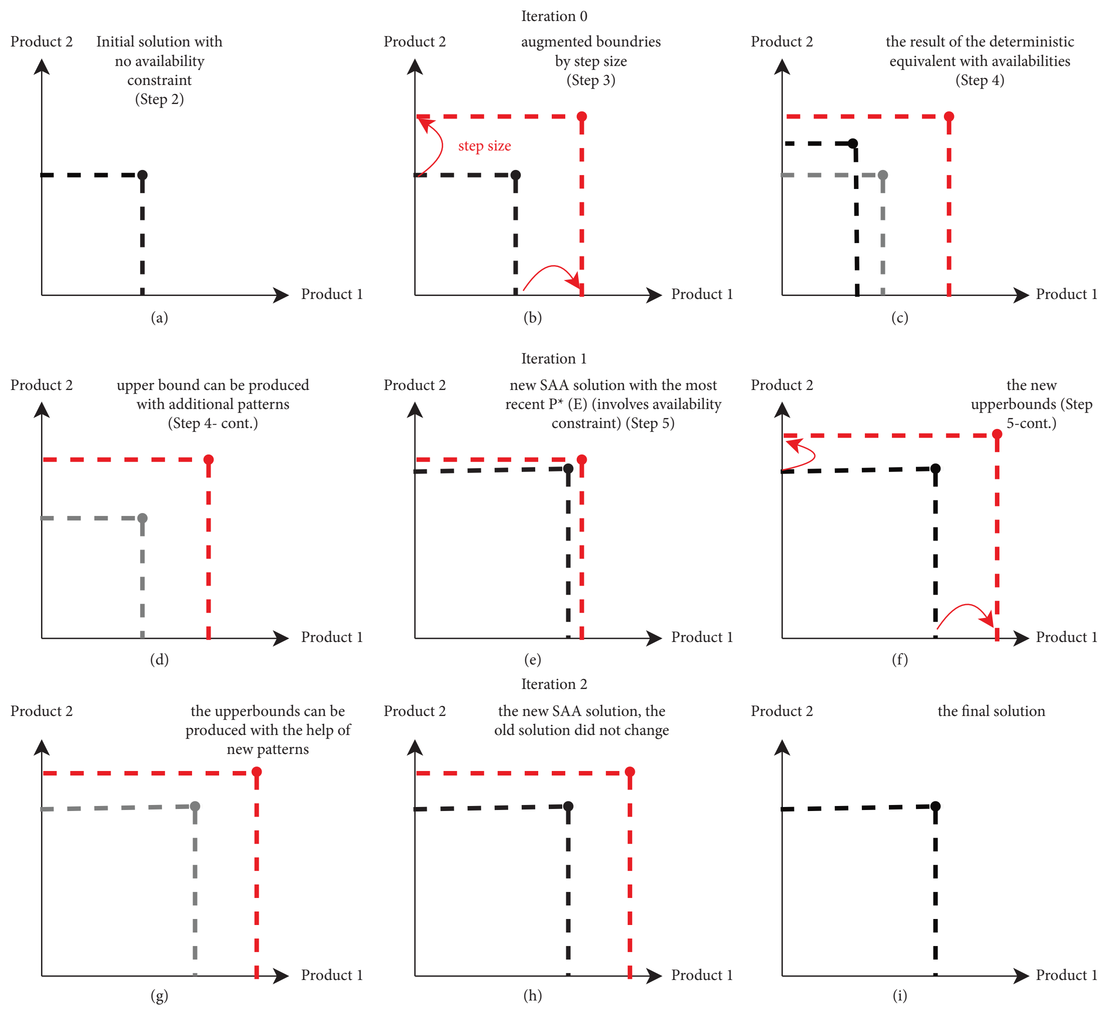
Iteration 0
Product 2
Initial solution with no availability constraint (Step 2)
Product 1
(a)
augmented boundries by step size (Step 3)
step size
(b)
the result of the deterministic equivalent with availabilities (Step 4)
(c)
Iteration 1
upper bound can be produced with additional patterns (Step 4- cont.)
(d)
new SAA solution with the most recent P* (E) (involves availability constraint) (Step 5)
(e)
the new upperbounds (Step 5-cont.)
(f)
Iteration 2
the upperbounds can be produced with the help of new patterns
(g)
the new SAA solution, the old solution did not change
(h)
the final solution
(i)

Figure 4: (a) The initial solution is free of item availability restrictions and serves as an initial reference point. (b) The step size augments the search space. (c) The available items cannot produce the upper bound or the optimal point (greyed-out point). The item availability constraints are binding and lead to another solution (shown in black). (d) The DA finds new patterns and augments the pattern set to produce the upper bound. The available items can produce the upper bounds. This augmented pattern set is proper. (e) The SAA finds a new optimal solution with item availabilities. (f) The new upper bound is calculated and a different upper bound than the previous iteration. The solution may improve when new patterns are added. We check if this new upper bound triggers new pattern generation. (g) The existing pattern set cannot produce the most recent upper bound. Hence, the DA is rerun until it can. (h) The SAA is rerun with all patterns found. The solution is the same as the previous iteration. The addition of new patterns did not affect the solution. Hence, the production amounts have converged. (i) Since convergence is caught, we accept this solution as the optimal solution.

In either case, we pass on to the next stage: solving the original problem with SAA. However, if the upper bound is infeasible, we run the SAA once to obtain the final results, knowing we cannot generate any more patterns. We continue the recursions if $P^*(E)$ is proper.

In Figure 4(c), the items cannot produce the upper bounds or the optimal amount, per se. Therefore, the initial pattern set needs to be revised. We rerun the DA until the upper bounds can be produced. Figure 4(d) shows such a case.

4.3.5. Solving the Two-Stage Stochastic Programming with Recourse (The Outer While Loop in Algorithm 1). The SAA is run with the most recent pattern set on the original problem with availability constraints given through equations (8)–(17) as mentioned in Section 4.3. This recursion analyzes whether the newly added patterns change the previous optimal solution and the upper bound. If the upper bounds differ from the previous iterations, the new minimal pattern set enables different pattern-product configurations than previous iterations. Figure 4(e) shows the new solution with the most recent pattern set and the item availabilities. Based on this solution, Figure 4(f) finds the new upper bounds. The upper bounds differ from those found in Figure 4(b); hence, the recursion starts.

4.3.6. Recursion until Convergence. The recursion between the pattern generation (Step 4) and the SAA (Step 5) continues until (i) no further patterns can be obtained, or (ii) the production amounts and the upper bounds remain the same

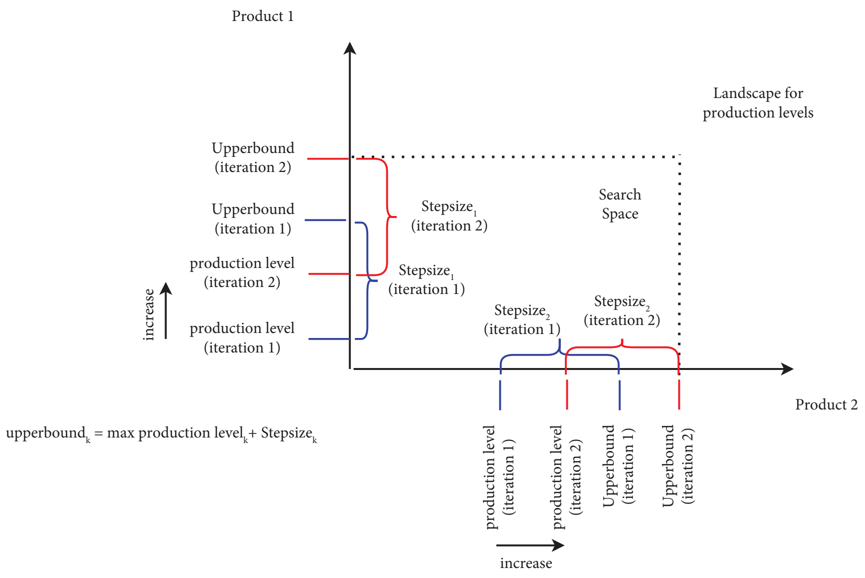
Product 1
Landscape for production levels
Upperbound (iteration 2)
Upperbound (iteration 1)
production level (iteration 2)
production level (iteration 1)
increase
Stepsize$_1$ (iteration 2)
Stepsize$_1$ (iteration 1)
Search Space
Stepsize$_2$ (iteration 1)
Stepsize$_2$ (iteration 2)
Product 2
upperbound$_k$ = max production level$_k$ + Stepsize$_k$
production level (iteration 1)
production level (iteration 2)
Upperbound (iteration 1)
Upperbound (iteration 2)
increase

FIGURE 5: The expansion of the search space at each iteration.

between two consecutive iterations in which the original problem (problem with availability constraint) is solved in each iteration. If there is no further change in these outputs, the pattern set is sufficient to produce the optimal amounts, and the optimal solution has converged.

In Figure 4(g), we check if the new upper bounds can be produced with the pattern set from the previous iteration. If not, the DA produces new patterns, as mentioned in Step 4. However, Figure 4(g) shows the augmented set that can solve the upper bounds. When $P^* (E)$ is proper ($P_p^* (E)$), the SAA is rerun to find the optimal solution with the addition of new patterns as shown in Figure 4(h). The solution found in Figure 4(h) is the same as in the previous iteration (as shown in Figure 4(e)). The addition of new patterns has not changed the optimal solution. The production amounts and the upper bounds have converged. Therefore, we stop the algorithm and accept the final solution as shown in Figure 4(i).

With these recursions, the search space boundaries are dynamically updated through iterations by using step size (see Figure 4). $P_p^* (E)$ is obtained for each production amount of each sample by checking if $P^* (E)$ can produce the upper bounds. Finally, we obtain a solution of the original stochastic problem [16] as the SAA output. The recursive structure is also visualized in Figure 3.

5. An Illustrative Example

In this section, we present a small instance of $E \coloneqq (m, K, l, L, b)$ as a stochastic and multiproduct version of the SSP as an illustrative example. In this example, $E \coloneqq (3, 2, (500, 400, 300), (1000, 1500, (45, 65, 85)))$. The demand random variables are $D_1 \sim \text{Pois}(10)$ and $D_2 \sim \text{Pois}(20)$, respectively. In the illustrative example, the step size parameter of each product is expressed in terms of the standard deviation of the demand distributions. It helps us to track changes in results by changing the standard deviation multipliers. Therefore, the standard deviations for the demand are $\sigma_1 \approx 3.16$ and $\sigma_2 \approx 4.47$. Let d_{sk} denote the value of the demand for product $k \in \mathbb{K}$ under scenario $s \in \mathbb{S}$. Furthermore, assume that the random approved product rate is $\Upsilon \sim \mathcal{N}(0.95, 0.0001)$. $C^{\text{Pr}} = 100$ euros per product, $C^O = 60$ euros per pattern change, and $C^R = [75 \quad 60 \quad 45]$ euros for each item, $i \in \mathbb{I}$. Moreover, let $C^H = [320 \quad 480]$ and $C^B = [800 \quad 1200]$ euros. Initially, we will assume the step size for each product arbitrarily as $2\sigma_1$ and $3\sigma_2$ and then explain the impact of the step size on the solution quality. The following illustrates the methodology based on this example:

Iteration 0

(1) The DA is run to generate the minimal pattern set $P^* (E)$. Matrices $\mathbf{a}$ and Δ are presented below. Each column of $\mathbf{a}$ represents a pattern, and each row represents an item. Consequently, each cell a_{ij} indicates the amount of item i used in pattern j.

$$\mathbf{a} = \begin{bmatrix} 0 & 0 & 1 & 2 & 3 \\ 1 & 3 & 1 & 0 & 0 \\ 2 & 1 & 2 & 0 & 0 \end{bmatrix},$$
$$\Delta = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}^T. \qquad (37)$$

(2) The SAA is applied to the relaxed problem (without the item availability constraint given in equation (9)) with $G = 10$. Initial reference values for $\text{ProdLevels}_k(g)$ and maxProdLevels_k for all k are given as follows:

$$\text{prodLevels} = \begin{bmatrix} 10 & 21 \\ 10 & 20 \\ 10 & 21 \\ 10 & 22 \\ 11 & 21 \\ 10 & 21 \\ 10 & 20 \\ 10 & 20 \\ 10 & 21 \\ 10 & 20 \end{bmatrix} \Rightarrow \text{maxProdLevels} = \begin{bmatrix} 11 & 22 \end{bmatrix}. \tag{38}$$

(3) Upper bounds for each product (upperBound_k) are computed such that $\text{upperBound} = \begin{bmatrix} 11 + 2\sigma_1 & 22 + 3\sigma_2, \end{bmatrix}$ making $\text{upperBound}_1 = 17$ and $\text{upperBound}_2 = 33$.

Iteration 1:

(4) The deterministic SSP model in equations (18)–(22) is used to check whether the minimal pattern set on hand satisfies the upper bound equation (20) If not, the DA would have to generate an additional minimal pattern set until the upper bound can be produced. In that case, the proper minimal pattern set is produced. In our example, the output (solution) of the deterministic SSP indicates that $P_p^*(E)$ was obtained.

$$\mathbf{a} = \begin{bmatrix} 0 & 0 & 1 & 2 & 3 \\ 1 & 3 & 1 & 0 & 0 \\ 2 & 1 & 2 & 0 & 0 \end{bmatrix}, \quad \Delta = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}^T, \tag{39}$$

and by using $P^*(E)$, total production amount for each product is obtained.

$\mathbf{x} = \begin{bmatrix} 10 & 0 & 0 & 7 & 0 \\ 0 & 2 & 31 & 0 & 0 \end{bmatrix}^T \Rightarrow \text{output} = \begin{bmatrix} 17 & 33 \end{bmatrix}$, then $\text{since output} = \begin{bmatrix} 17 & 33 \end{bmatrix} \geq \text{upperBound} = \begin{bmatrix} 17 & 33 \end{bmatrix}$,

$P_p^*(E) = P^*(E)$ is obtained.

Since the minimal pattern set is proper, we move on to generate the SAA result and observe any change in the upper bounds.

(5) The SAA is applied to the original problem (with the availability constraint (9)). ProdLevels, maxProdLevels, and upperBound are computed.

$$\text{prodLevels} = \begin{bmatrix} 10 & 21 \\ 10 & 20 \\ 10 & 21 \\ 10 & 22 \\ 11 & 21 \\ 10 & 21 \\ 10 & 20 \\ 10 & 20 \\ 10 & 21 \\ 10 & 20 \end{bmatrix} \Rightarrow \text{maxProdLevels} = \begin{bmatrix} 11 & 22 \end{bmatrix} \Rightarrow \text{upperBound} = \begin{bmatrix} 17 & 33 \end{bmatrix}. \tag{40}$$

(6) Since the results of the SAA have not changed, and $\text{maxProdLevels}_k(1) = \text{maxProdLevels}_k(2), \forall k$ (maxProdLevels did not increase for either product), we stop the algorithm.

$$\text{maxProdLevels}(1) = \begin{bmatrix} 11 & 22 \end{bmatrix} \geq \text{maxProdLevels}(2) = \begin{bmatrix} 11 & 22 \end{bmatrix}. \tag{41}$$

Recall from Section 4.2, the SAA approach searches for the solution of N_g instances having N scenarios each given in the model presented in equations (8)–(17). The candidate solutions are evaluated by solving the SAA algorithm with each solution obtained from N_g instances in the reference sample N'. In this illustrative example, the number of samples (G), the sample size for each sample (N), the reference sample size (N'), the candidate solutions, and the summary statistics for the objective function values are presented in Table 1. Monte Carlo sampling is used [16] in which the sampling takes place before the solution procedure.

The DA is coded in MATLAB R2017b, and the SAA is implemented in GAMS 34.2.0. CPLEX is used as a solver in GAMS. The algorithm is executed on a computer with Intel core i5-3230M, 2.60 GHz CPU, and 4 GB RAM. The candidate solutions and statistical computations of objective function values are presented in Table 1. According to this

Table 1: The SAA illustrative example with Monte Carlo sampling $N = 100$, $N' = 400$, $G = 10$.

g	X_{11}	X_{32}	v^g	$\widehat{v}^g$	$\widehat{\sigma}^2_{\widehat{v}^g}$	$gap_g(\overline{x}^g)$	$\widehat{\sigma}^2_{gap_g}$
1	10	21	13662	14047	28934	318	36376
2	10	20	14075	14159	35391	430	42833
3	10	21	13681	14047	28934	318	36376
4	10	22	13657	14085	23186	356	30628
5	11	21	13982	14063	26975	334	34417
6	10	21	13546	14047	28934	318	36376
7	10	20	13279	14159	35391	430	42833
8	10	20	13691	14159	35391	430	42833
9	10	21	14176	14047	28934	318	36376
10	10	20	13537	14159	35391	430	42833
$\overline{v}^G = 13729$							
$\widehat{\sigma}^2_{\overline{v}^G} = 7442$							

table, the first, third, sixth, and ninth samples have the smallest expected total costs according to Table 1. Moreover, the convergence is more robust in these samples since they have smaller optimality gaps and smaller variances for the optimality gap than the other samples. Therefore, the results of the first, third, sixth, and ninth samples can be denoted as the best candidate solutions. According to these results, the production amounts are $x_1 = 10$ and $x_2 = 20$ or 21. G, N, N' can be increased to obtain a stricter convergence. Nonetheless, the increase in computational complexity should not be overlooked in response to increasing sample sizes.

An important point is that the SAA method solves the SMIP problem optimally under the pattern set generated by the DA. Hence, different pattern sets may lead to different optimal points. Because the DA performance directly affects the solution quality, parameter tuning becomes essential to the DA.

6. Numerical Example and Results

6.1. Numerical Example and Discussions. This section presents a large-sized multiproduct stochastic SSP and its results. The instance SSP $E := (m, K, l, L, b)$ is $E := (50, 2, l, (1200, 1300), b)$ where $D_1 \sim \text{Pois}(150), D_2 \sim \text{Pois}(200)$ and $\sigma_1 \approx 12.24\sigma_2 \approx 14.44$ for each product. The random approved product rate is $\Upsilon \sim \mathcal{N}(0.95, 0.0001)$. $C^{\text{Pr}} = 100$ euros per product and $C^O = 60$ euros per pattern change, and the cost of each small item is computed as $C_i^R = ((l_i/1000) * 3000\,\text{m}) * (0.05\,\text{euros/m}^2)$, respectively for $i \in \mathbb{I}$. The width of small items and products is denoted in millimeters. Moreover, $C^H = [320\ 480]$ euros per product and $C^B = [800\ 1200]$ euros per product. l and b vectors are presented as $l^T = [197\ 195\ 194\ 191\ 185\ 173\ 171\ 168\ 164\ 160\ 156\ 155\ 153\ 152\ 138\ 133\ 131\ 130\ 128\ 120\ 116\ 115\ 114\ 105\ 101\ 99\ 98\ 96\ 91\ 89\ 86\ 76\ 75\ 66\ 64\ 60\ 56\ 49\ 39\ 36\ 31\ 30\ 23\ 19\ 14\ 13\ 12\ 11\ 9\ 6]$; $b^T = [200\ 300\ 300\ 300\ 300\ 300\ 300\ 300\ 300\ 300\ 300\ 300\ 300\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100\ 100]$.

The parameters of the SAA are $G = 30$, $N = 100$, and $N' = 400$. Similarly, the parameters of the DA are $B = 0.05$, $\alpha = 0.5, \beta = 0.03, \gamma = 0.07, \eta = 0.05, \epsilon = 0.05$, $\omega = 0.05$. Finally, the step sizes are $0.6\sigma_1$ and $0.75\sigma_2$.

We run the algorithm three times with different $\Lambda_b' = (20, 50, 70)$ and MaxIt = (10, 20, 30) values to capture the behavior of the SAA for different $P_p^*(E)$. The statistics for the computational complexity of the SAA and their deterministic equivalents and trial results are presented in Table 2.

The objective function values, the optimality gaps, and the variances of the gaps are highly close to each other. Because the second trial has the smallest expected total cost, we can choose $P_p^*(E)$ of the second trial. The candidate solution of the seventh sample with minimum $\widehat{v}^g$ can be determined as the best solution in the second trial, i.e., $x^{\text{SAA}} = \text{argmin} v^{\text{SAA}}$ where $v^{\text{SAA}} = \min_{g=1,\ldots,G} \widehat{v}^g$. The obtained solution is presented in Table 3.

The used amount of items is presented as a vector $r^{\text{SAA}} = [113\ 185\ 22\ 78\ 84\ 245\ 0\ 210\ 298\ 299\ 248\ 107\ 129\ 0\ 0\ 77\ 51\ 0\ 0\ 22\ 99\ 100\ 20\ 51\ 52\ 51\ 0\ 100\ 26\ 71\ 6\ 99\ 100\ 77\ 0\ 26\ 54\ 92\ 0\ 74\ 48\ 26\ 74\ 56\ 0\ 78\ 51\ 6\ 0\ 0]$.

One of the contributions of our algorithm is to obtain the preferred $P_p^*(E)$ by using the minimization model equations (18)–(22), which controls the propriety of the $P^*(E)$. In our 50-item example, the computational time given in Table 2 displays the time required by the initial proper minimal pattern set. Even with the enlarged pattern sets, the computational time is around two to three minutes.

We conduct a further sensitivity analysis to analyze the impact of the step size through different multipliers for σ. Given all other parameters and random variable values remaining the same, the comparisons regarding various statistics are presented in Table 4. The primary effect of the step size is on the proper minimal pattern set and the computational time. A larger step size yields a higher production upper bound, which, as a result, requires a more extensive pattern set. The increase in the pattern set also requires a higher computational time. However, the most important performance measures are (i) the expected total cost, (ii) the variance of the total cost, (iii) the optimality gap, and (iv) the variance of the optimality gap. Table 4 shows the average mean and variance of the objective function values, the optimality gaps, and the average CPU time (in terms of seconds) of 30 runs of the algorithm. For a fair comparison, all scenarios are kept fixed for every step size value. Table 4 visualizes the effect of the step size on the average objective function value and the CPU time. As can be seen from the table, even though the step size is increased more than ten times, the computational time increased logarithmically. However, the decrease in the objective function value as the step size increases is much less significant compared to the increase in the computational time. The improvement in the objective function value remains less than 0.1%.

Table 2: The computational complexity and results for $N = 100$, $N' = 400$, $G = 30$.

Trial	Variables	Constraints	g	$\hat{v}^{\text{best}}$	$\hat{\sigma}^2_{\hat{v}^{\text{best}}}$	$\text{gap}_{\text{best}}(\bar{x}^g)$	$\hat{\sigma}^2_{\text{gap}_{\text{best}}}$	CPU time (sec.)
1	1510	2000	8	121562	241086	203	296115	53.6916
2	2888	3430	7-8	121536	229547	213	284571	61,0528
3	3895	4475	7	121835	241086	228	296011	74.6621

Table 3: Frequency of patterns in x^{SAA}.

Product (k)/Pattern (j)	x_2	x_{12}	x_{18}	x_{19}	x_{27}	x_{40}	x_{43}	x_{44}
1	26	—	—	—	22	—	—	107
2	—	51	20	56	—	78	6	—

Table 4: Analysis of different step sizes.

Stepsize$_1$ =	$13\sigma_1$	$10\sigma_1$	$6.5\sigma_1$	$3.25\sigma_1$	$1.25\sigma_1$	$0.6\sigma_1$
Stepsize$_2$ =	$16\sigma_2$	$12\sigma_2$	$7.5\sigma_2$	$3,75\sigma_2$	$1.5\sigma_2$	$0.75\sigma_2$
v^{SAA}	121523	121559	121547	121597	121618	121593
$\hat{\sigma}^2_{\hat{v}^{\text{SAA}}}$	229468	229547	231179	230194	230470	229401
gap_{SAA}	175	191	168	176	175	191
$\hat{\sigma}^2_{\text{gap}_{\text{SAA}}}$	280227	285398	286006	284908	282488	283713
CPU time (sec)	258	102	76	70	74	55

Even though this study is not a multiobjective analysis of the stochastic SSP, it also supervises the trim loss. While the main objective of the SAA is to minimize the total cost incurred throughout the production process, the DA also contributes to minimizing the trim loss. The variable δ_{jk} only allows minimal patterns to be used so that the SAA model does not allow any nonminimal patterns to be included in skiving even if the end product satisfies the given thresholds. A counterargument for this structure is the case of a very high set-up cost. If the set-up cost is considerably high, the model may sacrifice the trim loss, and the longest pattern can be used to manufacture all products. This way, the model avoids paying expensive set-up costs when patterns change. In such cases, $\delta_{\text{jk}} = 1$ for shorter products even though the pattern is not minimal. For example, if the set-up cost was 6000 instead of 60, then the patterns for the second product (with a width of 1300) could be used for the first product (with a width of 1200).

6.2. Performance Analysis for the DA. While the first phase of the algorithm uses a metaheuristic, the SAA in the second phase implements an MIP algorithm under various scenarios. Hence, the second phase solves the problem of optimality under given parameters and scenarios. If the number of samples and scenarios is sufficient, the result converges to the optimal solution by the law of large numbers. Nonetheless, the parameters fed into the second phase are the results of the DA, which does not guarantee the optimal pattern set. Hence, in this section, we analyze the effect of the DA parameters on the objective function value. The literature recommends that $\alpha + \beta + \gamma = 1$ [15]; hence, the first nine rows of Table 5 analyze this case, and the last nine rows use $\alpha + \beta + \gamma < 1$ for slower convergence and avoiding jumping over the optimal solution. As can be seen from the table, the mean absolute difference rate between the highest and lowest values of the average objective function is approximately 0.3%. For this problem, the DA is robust against parameter changes and produces an abundance of patterns.

Further analysis is carried out about the run time of the algorithm by increasing the number of item types in Figure 6 and the number of product types in Figure 7. According to both graphics, when the number of item types and the number of product types increase, the run time increases, as shown in these figures. Next, we increase the means of demand random variables of both product types by using several growth rates to investigate the relation between run time and demand quantity (Figure 8).

For further analysis, experiments for several demand levels were carried out to analyze and compare the performance of the particle swarm optimization algorithm (PSO) and DA for the model as shown in equations (18)–(22) (Table 6). For each demand level, PSO and DA are run five times, and we recorded the minimum results among five runs for the PSO and DA. According to the results, for the minimization of the total cost, DA is superior to the PSO for every demand level in the experiments. Moreover, the CPU times of the DA are more stable than the CPU times of the PSO. Especially, for the high demand levels, there is a big difference in terms of run time. For high demand levels, the PSO generates a large number of unnecessary patterns because the generated patterns seem to be similar. On the other hand, DA seems to generate patterns with different configurations because DA has better exploration features.

7. Discussions

Zak [1] denotes that the CSP is a set-covering problem and the SSP is a set-packing problem. Their names, however, are the names of the two fundamental technological processes: cutting and skiving. A cutting process is the same as a packing process. It is the same for the skiving and covering. Zak also noted that the cutting (packing) process is well-modeled by a knapsack, while the skiving (covering) process is well-modeled by an unbounded knapsack problem (UKP). Moreover, according to Zak, the CSP can be converted to a set-covering problem, while the SSP can be converted to a set-packing problem. Because they have similar input matrices, they have a strong relationship with each other in terms of modeling and solution approaches [1].

Such as the CSP, the SSP is also an NP-hard problem [4, 20], reducible to the unbounded knapsack problem or

Table 5: Performance analysis for the parameters of the DA.

α	β	γ	Average objective function value
Number of dragonflies: 20			
$\alpha+\beta+\gamma=1$			
0.1	0.3	0.6	121498
0.1	0.5	0.4	121586
0.1	0.7	0.2	121639
0.3	0.1	0.6	121627
0.3	0.3	0.4	121542
0.3	0.5	0.2	121482
0.5	0.3	0.2	121493
0.7	0.2	0.1	121466
$\alpha+\beta+\gamma<1$			
0.3	0.03	0.07	121590
0.3	0.05	0.05	121667
0.3	0.07	0.03	121676
0.5	0.03	0.07	121542
0.5	0.05	0.05	121662
0.5	0.07	0.03	121542
0.7	0.03	0.07	121542
0.7	0.05	0.05	121667
0.7	0.07	0.03	121499
Number of dragonflies: 50			
$\alpha+\beta+\gamma=1$			
0.1	0.1	0.8	121442
0.1	0.3	0.6	121625
0.1	0.5	0.4	121502
0.1	0.7	0.2	121472
0.3	0.1	0.6	121541
0.3	0.3	0.4	121688
0.3	0.5	0.2	121516
0.5	0.3	0.2	121620
0.7	0.2	0.1	121694
$\alpha+\beta+\gamma<1$			
0.3	0.03	0.07	121405
0.3	0.05	0.05	121536
0.3	0.07	0.03	121533
0.5	0.03	0.07	121782
0.5	0.05	0.05	121623
0.5	0.07	0.03	121669
0.7	0.03	0.07	121542
0.7	0.05	0.05	121666
0.7	0.07	0.03	121604
Number of dragonflies: 100			
$\alpha+\beta+\gamma=1$			
0.1	0.1	0.8	121602
0.1	0.3	0.6	121615
0.1	0.5	0.4	121590
0.1	0.7	0.2	121597
0.3	0.1	0.6	121588
0.3	0.3	0.4	121609
0.3	0.5	0.2	121523
0.5	0.3	0.2	121548
0.7	0.2	0.1	121662
$\alpha+\beta+\gamma<1$			
0.3	0.03	0.07	121602
0.3	0.05	0.05	121594
0.3	0.07	0.03	121480
0.5	0.03	0.07	121718
0.5	0.05	0.05	121503

Table 5: Continued.

α	β	γ	Average objective function value
0.5	0.07	0.03	121494
0.7	0.03	0.07	121534
0.7	0.05	0.05	121491
0.7	0.07	0.03	121528

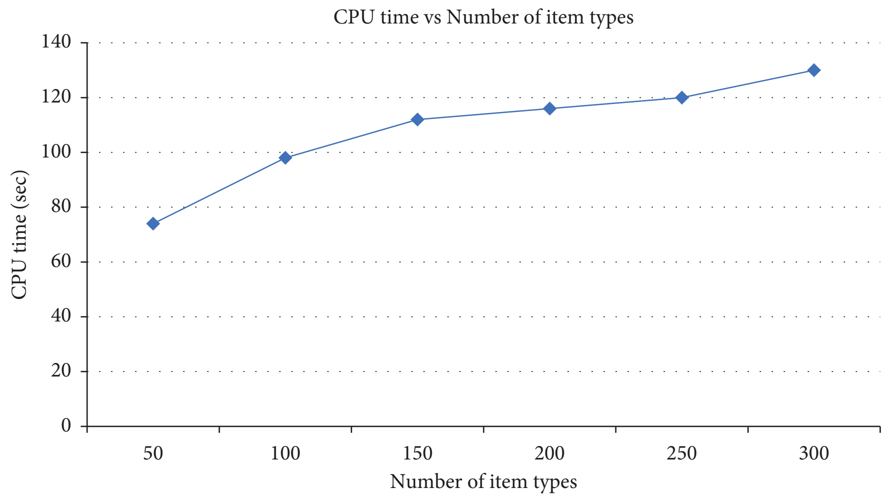

Figure 6: CPU time (sec) vs. number of item types.

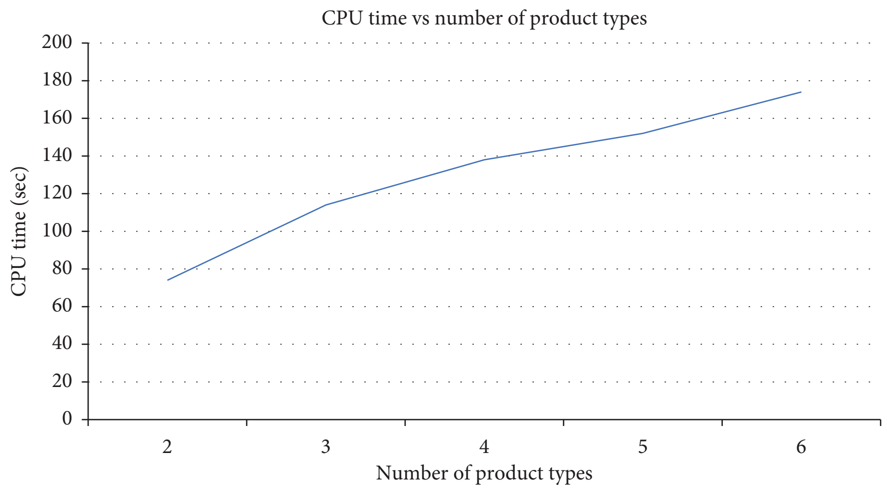

Figure 7: CPU time (sec) vs. number of product types.

dual bin packing problem which are known as the NP-hard in the strong sense [23, 62]. The overall problem can be formulated as an integer linear programming problem [1, 20].

The dual bin packing can be described as a special case of the unbounded multiple knapsack problem (UMKP), a particular generalized assignment problem [63]. UMKP describes each item by its weight and integer value. There are m knapsacks, each with its carrying capacity. The aim is to pack a subset of the items so that all the items fit into the knapsacks without violating the weight constraints, and the total value of the packed items is as large as possible. In the uniform UMKP, the capacities of the bins are the same. The problem is studied for the first time and proved to be NP-hard by Assmann et al. [23]. Furthermore, the DBBP, and hence MKP, is strongly NP-hard even for $m = 2$ by [63].

Our model is the well-known multiconstrained integer linear programming problem (the deterministic equivalent of the stochastic integer program), which is the function of the nonuniform DBPP or the UMKP. The DBPP or the

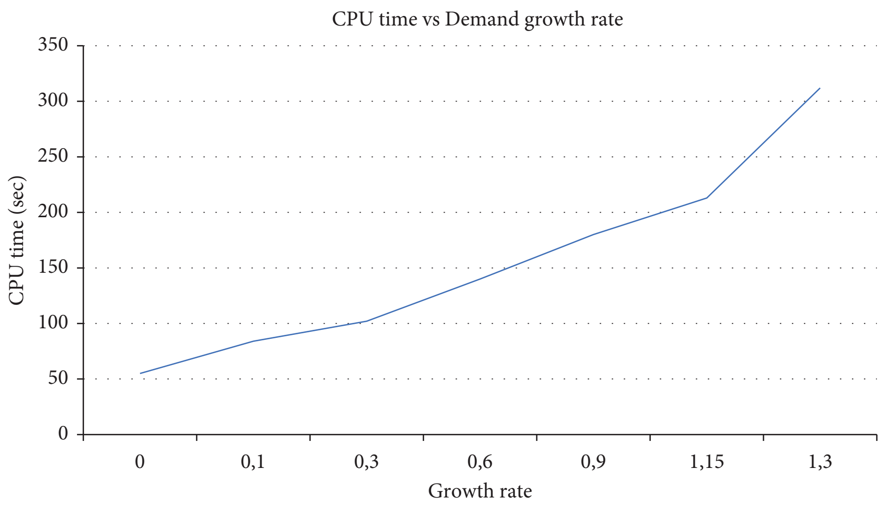

Figure 8: CPU time (sec) vs. growth rates for the demand means of two product types.

Table 6: Comparison of the PSO and DA for the total cost minimization.

Demand		PSO				DA			
Product 1	Product 2	Total cost	CPU (sec)	Product 1 shortage	Product 2 shortage	Total cost	CPU (sec)	Product 1 shortage	Product 1 shortage
25	50	22050	8.34	0	0	21870	20.03	0	0
50	25	21555	7.55	0	0	21499	20.4	0	0
50	50	28870	9.51	0	0	28870	19.03	0	0
100	50	43350	12.55	0	0	42930	19.83	0	0
50	100	44100	25.06	0	0	43680	19.38	0	0
100	100	58220	25.95	0	0	57620	19.94	0	0
150	100	72340	55.39	0	0	72340	37.29	0	0
100	150	73510	109.83	0	0	72550	37.72	0	0
150	150	88170	60.96	0	0	87510	53.55	0	0
200	150	102770	533.53	0	0	101810	51.86	0	0
150	200	—	—	—	—	102920	68.13	0	0
200	200	—	—	—	—	116295	229.58	1	3

UMKP is the $\mathcal{NP}$ hard where optimal solution of the IP in our study is greater and equal to output maximization of the pure SSP (maximum output for the DBPP) (another aspect $x_{\text{sp}}^{*} \longrightarrow x_{\text{ssp}}^{\text{relaxation}}$). The IP is also considered as the NP-hard [62]. As a result, as the number of bins increases, in other words, the number of patterns, which are the decision variables of our IP model, increases, the complexity of our problem increases rapidly.

On the other hand, for the complexity of stochastic programming problems such as our problem, Shapiro [57] states that the approximate solutions, with sufficiently high accuracy, of linear two-stage stochastic programs with fixed recourse are NP-hard even if independent uniform distributions govern the random problem data.

Another discussion about combinatorial optimization problems is the solution methodology. While many methods exist in the literature for decision-making under uncertainty, the most comparable approach to the SP is robust optimization (RO). This study implements the SP due to its flexibility to employ multistage models and its ability to handle scenarios with recourse actions that are not traditionally a part of RO [64]. RO approaches are valuable in risk-sensitive and risk-critical systems, such as power-grid maintenance optimization to avoid blackouts [65] or the adoption of new surgical treatments [66]. In addition, the RO becomes necessary in the absence of information on the probability distributions. However, its conservative planning nature toward the worst-case scenarios can lead to extreme risk-averse decision-making in more risk-neutral systems [67]. Nonetheless, numerous authors propose a trade-off between the SP and the RO to balance the costs related to risks [68–70]. This study implements the SP model due to the availability of the probability distributions and the recourse actions. Moreover, a risk-averse decision in the SSP leads to higher inventory levels for unlikely extreme demand scenarios. Regardless of these assumptions, the stochastic SSP is also a suitable candidate for the analysis of different risk-based approaches depending on the criticality of the decisions.

8. Conclusions and Future Work

Skiving is essential in various industries, such as steel and textiles. Despite being a long process in the industry, the SSP is still an emerging field in research. While several studies have dealt with the deterministic version of the SSP, the stochastic nature of the problem is still under investigation.

This study has developed an iterative two-phase algorithm to solve the multiproduct stochastic SSP problem. We have adapted the DA to the problem to obtain an efficient set of proper patterns in the first phase. This set provides a feasible landscape for the second phase in which the SAA solves the stochastic problem. This phase offers the pattern set with minimized trim loss, whereas the later phase minimizes the total costs. While cost minimization is the primary objective in this problem, trim loss is secondarily minimized by the DA. Finally, the SAA finds the solution to the SMIP model through the pattern set obtained from the first phase and the best candidate solution, which gives the minimum or improved expected total cost for the stochastic problem. The case of pattern-dependent set-up costs also poses a consideration for future work. Additional research directions for the stochastic SSP involve developing a random search procedure for the upper bounds of the production amounts, combining two phases of the algorithm instead of expanding the search space by using the stepping method.

Nomenclature

Indices

i:	Item type index, $i \in \mathbb{I} \coloneqq \{1, \ldots, m\}$
j:	Minimal pattern index, $j \in \mathbb{J}^* \coloneqq \{1, \ldots, J\}$
k:	Product type index, $k \in \mathbb{K} \coloneqq \{1, \ldots, K\}$
s:	Scenario index, $s \in \mathbb{S} \coloneqq \{1, \ldots, S\}$

Parameters

v_s:	The approved product rate (1-waste rate) in scenario s
b_i:	The available amount of item type i
C^O:	The set-up cost for the change of each pattern (set-up costs are assumed to be the same)
C^{Pr}:	The fixed production cost for rolls with fixed length (i.e., 3000 meters)
C_i^R:	The cost of the item type i
C_k^B:	The cost of underproduction for product type k
C_k^H:	The cost of overproduction for product type k
d_{sk}:	The demand for product type k in scenario s
l_i:	The width of each item type i
L_k:	The threshold (lower bound) width of product type s
M_k:	The large number for each product such as $M_k \gg \max\{d_{sk}, \forall s \in \mathbb{S}, k \in \mathbb{K}\}$
P_s:	The probability of scenario s

First-Stage Decision Variables

r_i:	The amount of raw material i
x_{jk}:	The repeat frequency of pattern j for product type k
y_j:	A binary variable indicating whether a pattern j is used

Second-Stage Decision Variables (Recourse Actions)

q_{sk}^+:	The overproduction amount for product type k in scenario s
q_{sk}^-:	The underproduction amount for product type k in scenario s

Dragonfly Algorithm Decision Variables (Parameters for the Second Phase)

δ_{jk}:	A binary variable indicating whether pattern j can be used for product k or not
a_{ij}:	The number of item type i in pattern j.

Data Availability

The data used to support the findings of this study are included within the article.

Disclosure

"Incorporation of Thesis Work: A substantial portion of the research and findings presented in this manuscript originates and have evolved from the first and corresponding author's original doctoral dissertation "ANALYSIS OF THE STOCHASTIC SKIVING STOCK PROBLEM. The foundation established during the thesis work has significantly contributed to shaping the content, methodology, and insights. As such, this manuscript stands as an extension and refinement of the work undertaken during the first author's doctoral dissertation research."

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] E. Zak, "The skiving stock problem as a counterpart of the cutting stock problem," *International Transactions in Operational Research*, vol. 10, no. 6, pp. 637–650, 2003.

[2] C. Arbib, F. Marinelli, F. Rossi, and F. Di Iorio, "Cutting and reuse: an application from automobile component manufacturing," *Operations Research*, vol. 50, no. 6, pp. 923–934, 2002.

[3] K. C. Agoston, "The effect of welding on the one-dimensional cutting-stock problem: the case of fixed firefighting systems in the construction industry," *Advances in Operations Research*, vol. 2019, Article ID 6507054, 12 pages, 2019.

[4] J. Martinovic, E. Jorswieck, and G. Scheithauer, "The skiving stock problem and its application to cognitive radio Networks**This work is partly supported by the German research foundation (DFG) in the collaborative research center 912 highly adaptive energy-efficient computing," *IFAC-PapersOnLine*, vol. 49, no. 12, pp. 99–104, 2016.

[5] J. Martinovic, M. Delorme, M. Iori, G. Scheithauer, and N. Strasdat, "Improved flow-based formulations for the skiving stock problem," *Computers Operations Research*, vol. 113, Article ID 104770, 2020.

[6] D. Wang, F. Xiao, L. Zhou, and Z. Liang, "Two-dimensional skiving and cutting stock problem with setup cost based on

column-and-row generation," *European Journal of Operational Research*, vol. 286, no. 2, pp. 547–563, 2020.

[7] Y. Chen, X. Song, D. Ouelhadj, and Y. Cui, "A heuristic for the skiving and cutting stock problem in paper and plastic film industries," *International Transactions in Operational Research*, vol. 26, no. 1, pp. 157–179, 2019.

[8] D. J. Alem, P. A. Munari, M. N. Arenales, and P. A. V. Ferreira, "On the cutting stock problem under stochastic demand," *Annals of Operations Research*, vol. 179, no. 1, pp. 169–186, 2010.

[9] P. Beraldi, M. E. Bruni, and D. Conforti, "The stochastic trim-loss problem," *European Journal of Operational Research*, vol. 197, no. 1, pp. 42–49, 2009.

[10] M. K. Zanjani, D. A. Kadi, and M. Nourelfath, "A stochastic programming approach for sawmill production planning," *International Journal of Mathematics in Operational Research*, vol. 5, no. 1, pp. 1–18, 2013.

[11] J. R. Birge and F. Louveaux, *Introduction to Stochastic Programming*, Springer Science Business Media, Berlin, Germany, 2011.

[12] P. Kall, S. W. Wallace, and P. Kall, *Stochastic Programming*, Springer, Berlin, Germany, 1994.

[13] A. Shapiro, D. Dentcheva, and A. Ruszczyński, *Lectures on Stochastic Programming: Modeling and Theory*, SIAM, Philadelphia, PL, USA, 2014.

[14] R. J. Wets, "Stochastic programming models: wait-and-see versus here-and-now," in *Decision Making Under Uncertainty*, pp. 1–15, Springer, Berlin, Germany, 2002.

[15] S. Mirjalili, "Dragonfly algorithm a new meta-heuristic optimization technique for solving single-objective discrete and multi-objective problems," *Neural Computing & Applications*, vol. 27, no. 4, pp. 1053–1073, 2016.

[16] A. Shapiro and T. Homem-de Mello, "A simulation-based approach to two-stage stochastic programming with recourse," *Mathematical Programming*, vol. 81, no. 3, pp. 301–325, 1998.

[17] M. Johnson, C. Rennick, and E. Zak, "Case studies from industry: skiving addition to the cutting stock problem in the paper industry," *SIAM Review*, vol. 39, no. 3, pp. 472–483, 1997.

[18] G. Wäscher, H. Haußner, and H. Schumann, "An improved typology of cutting and packing problems," *European Journal of Operational Research*, vol. 183, no. 3, pp. 1109–1130, 2007.

[19] J. Martinovic and G. Scheithauer, "Characterizing IRDP-instances of the skiving stock problem by means of polyhedral theory," *Optimization*, vol. 67, no. 10, pp. 1797–1817, 2018.

[20] J. Martinovic and G. Scheithauer, "Integer linear programming models for the skiving stock problem," *European Journal of Operational Research*, vol. 251, no. 2, pp. 356–368, 2016.

[21] J. Martinovic and G. Scheithauer, "The proper relaxation and the proper gap of the skiving stock problem," *Mathematical Methods of Operations Research*, vol. 84, no. 3, pp. 527–548, 2016.

[22] J. Martinovic and G. Scheithauer, "New theoretical investigations on the gap of the skiving stock problem," *Pesquisa Operacional*, vol. 39, no. 1, pp. 1–35, 2019.

[23] S. F. Assmann, D. S. Johnson, D. J. Kleitman, and J.-T. Leung, "On a dual version of the one-dimensional bin packing problem," *Journal of Algorithms*, vol. 5, no. 4, pp. 502–525, 1984.

[24] P. C. Gilmore and R. E. Gomory, "A linear programming approach to the cutting stock problem—Part II," *Operations Research*, vol. 11, no. 6, pp. 863–888, 1963.

[25] C. Arbib and F. Marinelli, "Integrating process optimization and inventory planning in cutting-stock with skiving option: an optimization model and its application," *European Journal of Operational Research*, vol. 163, no. 3, pp. 617–630, 2005.

[26] P. C. Gilmore and R. E. Gomory, "A linear programming approach to the cutting-stock problem," *Operations Research*, vol. 9, no. 6, pp. 849–859, 1961.

[27] T. K. Karaca, A. Altay, and F. Samanlioglu, "Solution approaches for the bi-objective skiving stock problem," *Computers & Industrial Engineering*, vol. 179, Article ID 109164, 2023.

[28] F. Golbabaei and M. Khadem, "Air pollution in welding processes—assessment and control methods," in *Current Air Quality Issues*, F. Nejadkoorki, Ed., IntechOpen, Rijeka, 2015.

[29] R. Álvarez-Valdés, A. Parajón, and J. M. Tamarit, "A tabu search algorithm for large-scale guillotine (un) constrained two-dimensional cutting problems," *Computers & Operations Research*, vol. 29, no. 7, pp. 925–947, 2002.

[30] C.-L. S. Chen, S. M. Hart, and W. M. Tham, "A simulated annealing heuristic for the one-dimensional cutting stock problem," *European Journal of Operational Research*, vol. 93, no. 3, pp. 522–535, 1996.

[31] M. H. Jahromi, R. Tavakkoli-Moghaddam, A. Makui, and A. Shamsi, "Solving an one-dimensional cutting stock problem by simulated annealing and tabu search," *Journal of Industrial Engineering International*, vol. 8, no. 1, pp. 24–28, 2012.

[32] K. Lai and J. W. Chan, "Developing a simulated annealing algorithm for the cutting stock problem," *Computers & Industrial Engineering*, vol. 32, no. 1, pp. 115–127, 1997.

[33] F. Ducatelle and J. Levine, "Ant colony optimisation for bin packing and cutting stock problems," in *UK Workshop on Computational Intelligence*, pp. 1–16, UKCI-01), Edinburgh, Scotland, 2001.

[34] J. Levine and F. Ducatelle, "Ant colony optimization and local search for bin packing and cutting stock problems," *Journal of the Operational Research Society*, vol. 55, no. 7, pp. 705–716, 2004.

[35] R. Hinterding and L. Khan, "Genetic algorithms for cutting stock problems: with and without contiguity," in *Progress in Evolutionary Computation*, pp. 166–186, Springer, Berlin, Germany, 1993.

[36] T. Leung, C. Yung, and M. D. Troutt, "Applications of genetic search and simulated annealing to the two-dimensional non-guillotine cutting stock problem," *Computers & Industrial Engineering*, vol. 40, no. 3, pp. 201–214, 2001.

[37] H.-C. Lu, Y.-H. Huang, and K.-A. Tseng, "An integrated algorithm for cutting stock problems in the thin-film transistor liquid crystal display industry," *Computers & Industrial Engineering*, vol. 64, no. 4, pp. 1084–1092, 2013.

[38] R. R. Golfeto, A. C. Moretti, and L. L. D. Salles Neto, "A genetic symbiotic algorithm applied to the one-dimensional cutting stock problem," *Pesquisa Operacional*, vol. 29, no. 2, pp. 365–382, 2009.

[39] K.-H. Liang, X. Yao, C. Newton, and D. Hoffman, "A new evolutionary approach to cutting stock problems with and without contiguity," *Computers & Operations Research*, vol. 29, no. 12, pp. 1641–1659, 2002.

[40] G. Yang, W. Tang, and R. Zhao, "An uncertain furniture production planning problem with cumulative service levels," *Soft Computing*, vol. 21, no. 4, pp. 1041–1055, 2017.

[41] J. Martinovic and G. Scheithauer, "An upper bound of Δe<3/2for skiving stock instances of the divisible case," *Discrete Applied Mathematics*, vol. 229, pp. 161–167, 2017.

[42] M. C. Demirci, A. J. Schaefer, and J. M. Rosenberger, "Column generation within the L-shaped method for stochastic linear programs," Technical report, University of Pittsburgh, Department of Industrial Engineering, Pittsburgh, PA, USA, 2008, https://d-scholarship.pitt.edu/8295/1/Demirci_Dissertation_071108.pdf.

[43] H. Jin, Z. Wang, and C.-F. Chien, "A cut-to-order strategy for one-dimensional cable cutting and a case study," *Journal of the Chinese Institute of Industrial Engineers*, vol. 29, no. 8, pp. 572–586, 2012.

[44] S. S. Chauhan, A. Martel, and S. DAmour, "Roll assortment optimization in a paper mill: an integer programming approach," *Computers & Operations Research*, vol. 35, no. 2, pp. 614–627, 2008.

[45] D. Sculli, "A stochastic cutting stock procedure: cutting rolls of insulating tape," *Management Science*, vol. 27, no. 8, pp. 946–952, 1981.

[46] D. José Alem and R. Morabito, "Production planning in furniture settings via robust optimization," *Computers & Operations Research*, vol. 39, no. 2, pp. 139–150, 2012.

[47] H. Mohammadi Bidhandi and J. Patrick, "Accelerated sample average approximation method for two-stage stochastic programming with binary first-stage variables," *Applied Mathematical Modelling*, vol. 41, pp. 582–595, 2017.

[48] H. Mohammadi Bidhandi and R. Mohd Yusuff, "Integrated supply chain planning under uncertainty using an improved stochastic approach," *Applied Mathematical Modelling*, vol. 35, no. 6, pp. 2618–2630, 2011.

[49] T.-H. Yang, "Stochastic air freight hub location and flight routes planning," *Applied Mathematical Modelling*, vol. 33, no. 12, pp. 4424–4430, 2009.

[50] A. I. Hammouri, E. T. A. Samra, M. A. Al-Betar, R. M. Khalil, Z. Alasmer, and M. Kanan, "A dragonfly algorithm for solving traveling salesman problem," in *Proceedings of the 2018 8th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, pp. 136–141, IEEE, Penang, Malaysia, November 2018.

[51] M. R. Shirani and E. F. Safi, "Dynamic scheduling of tasks in cloud computing applying dragonfly algorithm biogeography based optimization algorithm and mexican hat wavelet," *The Journal of Supercomputing*, vol. 77, no. 2, pp. 1214–1272, 2020.

[52] M. Abdel-Basset, Q. Luo, F. Miao, and Y. Zhou, "Solving 0–1 knapsack problems by binary dragonfly algorithm," in *Proceedings of the International Conference on Intelligent Computing*, pp. 491–502, Springer, Liverpool, UK, August 2017.

[53] M. M. Mafarja, D. Eleyan, I. Jaber, A. Hammouri, and S. Mirjalili, "Binary dragonfly algorithm for feature selection," in *Proceedings of the 2017 International Conference on New Trends in Computing Sciences (ICTCS)*, pp. 12–17, IEEE, Amman, Jordan, October 2017.

[54] K. Baiche, Y. Meraihi, M. D. Hina, A. Ramdane-Cherif, and M. Mahseur, "Solving graph coloring problem using an enhanced binary dragonfly algorithm," *International Journal of Swarm Intelligence Research*, vol. 10, no. 3, pp. 23–45, 2019.

[55] J. Li, J. Lu, L. Yao, L. Cheng, and H. Qin, "Wind-solar-hydro power optimal scheduling model based on multi-objective dragonfly algorithm," *Energy Procedia*, vol. 158, pp. 6217–6224, 2019.

[56] N. Aydin, *Sampling based progressive hedging algorithms for stochastic programming problems*, PhD Thesis, Wayne State University, Detroit, Michigan, 2012.

[57] A. Shapiro, *Complexity of Two and Multi-Stage Stochastic Programming Problems*, Tutorial Notes for School of Industrial and Systems Engineering, Atlanta, Georgia, 2005.

[58] S. Ahmed and A. Shapiro, "The sample average approximation method for stochastic programs with integer recourse," *SIAM Journal on Optimization*, vol. 12, pp. 479–502, 2002.

[59] A. Shapiro, "Inference of statistical bounds for multistage stochastic programming problems," *Mathematical Methods of Operations Research*, vol. 58, no. 1, pp. 57–68, 2003.

[60] A. J. Kleywegt, A. Shapiro, and T. Homem-de Mello, "The sample average approximation method for stochastic discrete optimization," *SIAM Journal on Optimization*, vol. 12, no. 2, pp. 479–502, 2002.

[61] R. L. Rardin and R. L. Rardin, *Optimization In Operations Research*, Prentice Hall Upper Saddle River, NJ, Old Bridge, NJ, USA, 1998.

[62] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, USA, 1979.

[63] C. Chekuri and S. Khanna, "A polynomial time approximation scheme for the multiple knapsack problem," *SIAM Journal on Computing*, vol. 35, no. 3, pp. 713–728, 2005.

[64] S. Torres-Rincón, M. Sánchez-Silva, and E. Bastidas-Arteaga, "A multistage stochastic program for the design and management of flexible infrastructure networks," *Reliability Engineering & System Safety*, vol. 210, Article ID 107549, 2021.

[65] M. Nazari-Heris and B. Mohammadi-Ivatloo, "Chapter 2-application of robust optimization method to power system problems," in *Classical and Recent Aspects of Power System Optimization*, A. F. Zobaa, S. H. Abdel Aleem, and A. Y. Abdelaziz, Eds., Academic Press, Cambridge, MA, USA, 2018, https://www.sciencedirect.com/science/article/pii/B9780128124413000021.

[66] S. Neyshabouri and B. P. Berg, "Two-stage robust optimization approach to elective surgery and downstream capacity planning," *European Journal of Operational Research*, vol. 260, no. 1, pp. 21–40, 2017.

[67] M. Wei and J. Zhong, "Optimal bidding strategy for demand response aggregator in day-ahead markets via stochastic programming and robust optimization," in *Proceedings of the 2015 12th International Conference on the European Energy Market (EEM)*, pp. 1–5, IEEE, Lisbon, Portugal, May 2015.

[68] F. Shen, L. Zhao, W. Du, W. Zhong, X. Peng, and F. Qian, "Data-driven stochastic robust optimization for industrial energy system considering renewable energy penetration," *ACS Sustainable Chemistry & Engineering*, vol. 10, no. 11, pp. 3690–3703, 2022.

[69] R. Wang, K. S. Shehadeh, X. Xie, and L. Li, "Data-driven integrated home service staffing and capacity planning: stochastic optimization approaches," *Computers & Operations Research*, vol. 159, Article ID 106348, 2023.

[70] D. Yu, J. Wu, W. Wang, and B. Gu, "Optimal performance of hybrid energy system in the presence of electrical and heat storage systems under uncertainties using stochastic p-robust optimization technique," *Sustainable Cities and Society*, vol. 83, Article ID 103935, 2022.