KADİR HAS UNIVERSITY

SCHOOL OF GRADUATE STUDIES

DEPARTMENT OF INDUSTRIAL ENGINEERING

ANALYSIS OF THE STOCHASTIC SKIVING STOCK PROBLEM

TOLGA KUDRET KARACA

Ph.D. THESIS

ISTANBUL, APRIL, 2022

Tolga Kudret Karaca Ph.D. Thesis 2022

ANALYSIS OF THE STOCHASTIC SKIVING STOCK PROBLEM

TOLGA KUDRET KARACA

A thesis submitted to
the School of Graduate Studies of Kadir Has University
in partial fulfilment of the requirements for the degree of
Doctor of Philosophy in
Industrial Engineering

Istanbul, April, 2022

APPROVAL

This thesis titled ANALYSIS OF THE STOCHASTIC SKIVING STOCK PROBLEM submitted by TOLGA KUDRET KARACA, in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Industrial Engineering is approved by

Prof. Dr. Funda Samanlıoğlu (Advisor) (Kadir Has University)	....
Prof. Dr. Alev Taşkın Gümüş (Yıldız Technical University)	....
Prof. Dr. Ahmet Deniz Yücekaya (Kadir Has University)	....
Doç. Dr. Yeliz Ekinci (Istanbul Bilgi University)	....
Dr. Esra Ağca Aktunç (Kadir Has University)	....

I confirm that the signatures above belong to the aforementioned faculty members.

....
Prof. Dr. Mehmet Timur Aydemir
Director of School of Graduate Studies
Date of Approval: 25.04.2022

to my family

ACKNOWLEDGEMENT

I would like to thank my advisor Prof. Dr. Funda Samanlıoğlu for her consistent support and guidance during the running of this research. Prof. Dr. Funda Samanlıoğlu continuously provided encouragement and was always willing and enthusiastic to assist in any way she could throughout the research in this dissertation.

I would also like to thank committee members, Prof. Dr. Alev Taşkın Gümüş and Prof. Dr. Ahmet Deniz Yücekaya for their valuable inputs that make this research possible.

I also wish to thank Dr. Ayça Altay for providing advice and great support throughout this research process.

Finally, I would like to thank my family for supporting me during the compilation of this dissertation.

ANALYSIS OF THE STOCHASTIC SKIVING STOCK PROBLEM

ABSTRACT

This study addresses the stochastic version of the one-dimensional skiving stock problem (SSP), a rather recent combinatorial optimization challenge. The traditional SSP aims to determine the optimal structure that skives (combines) small items of various sizes side-by-side to form as many large items (products) as possible that satisfy a target width. This study considers a single-product and multi-product cases for the stochastic SSP. First, two-stage stochastic programming model is presented to minimize the total cost for the single product stochastic SSP which is under random demand. Integration of the Column Generation, Progressive Hedging Algorithm, and Branch and Bound is proposed where Progressive Hedging Algorithm is embedded in each node of the search tree to obtain the optimal integer solution. Next, the single product stochastic model is extended to the multi-product, multi-random variable model with the additional costs as a large size complex model. To examine this large-sized stochastic $\mathcal{NP}$-hard problem, a two-stage stochastic programming approach is implemented. Moreover, as a solution methodology, this problem is handled in two phases. In the first phase, the Dragonfly Algorithm constructs minimal patterns as an input for the next phase. The second phase executes a Sample Average Approximation method that provides solutions for the stochastic production problem with large size scenarios. Results indicate that the two-phase heuristic approach provides good feasible solutions under numerous scenarios without requiring excessive execution time. Finally, a multi-objective case for the deterministic SSP is analyzed where the objectives are minimization of the trim loss (waste), number of items in each product by considering the quality aspect, and number of pattern changes as the set-up. Lexicographic method is preferred for the multi-objective approach where preferences are ranked according to their importance. Column generation and Integer programming are further used to solve the multi-objective problem. In addition, a heuristic is proposed for the same multi-objective problem.

Keywords : Skiving Stock Problem, Stochastic Programming, Column Generation, Progressive Hedging Algorithm, Dragonfly Algorithm, Multi-objective, Sample Average Approximation.

STOKASTİK STOK BİRLEŞTİRME PROBLEMİ ANALİZİ

ÖZET

Bu çalışma, kombinatoriyal olması nedeni ile en iyileme zorluğu içeren tek boyutlu stok birleştirme probleminin (SBP) stokastik versiyonunu ele almaktadır. Geleneksel SBP, hedeflenen bir genişliği karşılayan, mümkün olduğunca çok sayıda büyük öğe (ürün) oluşturmak için çeşitli boyutlardaki küçük öğeleri yan yana birleştiren optimal bir yapıyı (planı) bulmayı amaçlar. Bu çalışma, stokastik SBP için tek ürünlü ve çok ürünlü durumları ele almaktadır. İlk olarak, rastgele talep altındaki tek ürünlü stokastik SBP için iki aşamalı stokastik programlama modeli sunulmuştur. Çözüm yöntemi olarak, kolon üretimi (Column Generation), aşamalı sınırlama (Progressive Hedging) ve dal sınır algoritmaları entegrasyonu önerilmiştir. Bu önerilen yöntemde kolon üretimi ile minimal bir birleştirme şablonu kümesi elde edilmiş olup, bu set kullanılarak, en iyi tamsayılı çözüm elde etmek için dal sınır algoritması arama ağacının her düğümünde, aşamalı sınırlama algoritması çalıştırılmıştır. Çalışmanın bir sonraki bölümünde, tek ürünlü stokastik model, çok ürünlü, çoklu rasgele değişken içeren ve ek maliyetler ihtiva eden büyük boyutlu karmaşık bir model haline getirilmiştir. Bu büyük boyutlu stokastik zor karmaşıklık sınıfındaki ($\mathcal{NP}$ hard) problem için iki aşamalı bir stokastik programlama yaklaşımı uygulanmıştır. Ayrıca bir çözüm yöntemi olarak bu sorun iki aşamada ele alınmaktadır. İlk aşamada, Yusufçuk Algoritması bir sonraki aşama için girdi olarak minimal birleştirme şablonları oluşturur. İkinci aşama, çok sayıda senaryo içeren stokastik probleme, aday çözümler sunan örneklem ortalaması yaklaşımı yöntemini yürütür. Sonuçlar, iki aşamalı sezgisel yaklaşımın, çok sayıda senaryo altında, aşırı uygulama süresi gerekmeksizin iyi çözümler sağladığını göstermektedir. Son olarak, deterministik stok birleştirme problemi için çoklu amaçlı bir örnek olay ele alınmıştır; bu amaçlar, fire en küçüklenmesi, birleştirme sebebi ile ürünlerde oluşan kaynak sayısının en küçüklenmesi ve üretim esnasında toplam kullanılan şablon sayısının en küçüklenmesi olarak belirlenmiştir. Bu çok amaçlı problem için tercihlerin önem derecesine göre sıralandığı, sıralama (Lexicographic) yöntemi tercih edilmiştir. Bu çok amaçlı problemin çözümünde, kolon üretme ve tamsayı programlama kombinasyonunun yanı sıra bir de sezgisel yöntem önerilmiş ve bu iki yöntemin sonuçları karşılaştırılmıştır.

Anahtar Sözcükler: Stok Birleştirme Problemi, Stokastik Programlama, Kolon Üretme, Aşamalı Sınırlama Algoritması, Yusufçuk Algoritması, Çok Amaçlı Yaklaşım, Örneklem Ortalaması Yaklaşımı

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

a_{ij}	the number of item type i in pattern j
x_j	the frequency (replication) of pattern j
L	the threshold (lower bound) width of product
l_i	the width of each item type i
i	index of (small) item types , $i \in \mathbb{I} := \{1, ..., m\}$
j	index of minimal patterns, $j \in \mathbb{J}^* := \{1, ..., J\}$
s	index of scenarios, $s \in \mathbb{S} := \{1, ..., S\}$
d_s	the demand in scenario s
C_j^{Pr}	the production cost which depends on the width of pattern j
C^H	the cost of overproduction
C^B	the cost of underproduction
P_s	the probability of scenario s
b_i	the available amount of item type i
q_s^+	the overproduction amount in scenario s
q_s^-	the underproduction amount in scenario s

LIST OF ACRONYMS AND ABBREVIATIONS

B&B	Branch and Bound Algorithm
BCP	Bin Covering Problem
BPP	Bin Packing Problem
CG	Column Generation Algorithm
CSP	Cutting Stock Problem
DA	Dragonfly Algorithm
DBPP	Dual Bin Packing Problem
EP	Evolutionary Programming
GA	Genetic Algorithm
IP	Integer Programming
KP	Knapsack Problem
LM	Lexicographic Method
LP	Linear Programming
MIP	Mixed Integer Programming
MOP	Multi-objective Optimization Problem
MP	Master Problem
PHA	Progressive Hedging Algorithm
PSO	Particle Swarm Optimization
RMIP	Relaxed Mixed Integer Programming
SA	Simulated Annealing
SAA	Sample Average Approximation
SIP	Stochastic Integer Programming
SMIP	Stochastic Mixed Integer Programming
SP	Stochastic Programming
SSP	Skiving Stock Problem
VSS	Value of Stochastic Solution
1D	One dimensional

1. INTRODUCTION

The Skiving Stock Problem (SSP) has been denoted as a derivative of the Cutting Stock Problem (CSP) by Zak [Zak, 2003] in terms of having similar inputs and solution approaches. The skiving process is a relatively recent technology that joins small items (auxiliary rolls) by combining, gluing, welding, or sewing to produce large items (products) that satisfy a minimum threshold width (Fig. 1.1). It aims, at large, to obtain as many large items as possible [Zak, 2003]. There are several real-life applications of the skiving process, such as the paper industry in which several narrow rolls are combined to form wider rolls [Zak, 2003]. In gear belt production, small rectangular leftover pieces forming after the cutting process are sewn to fabricate large rectangles [Arbib et al., 2002]. Other examples include pipe production, construction of firefighting systems [Ágoston, 2019], and spectrum aggregations in cognitive radio networks [Martinovic et al., 2016] in an SSP structure. The skiving process is especially applicable in industries with a high raw material waste rate [Arbib et al., 2002].

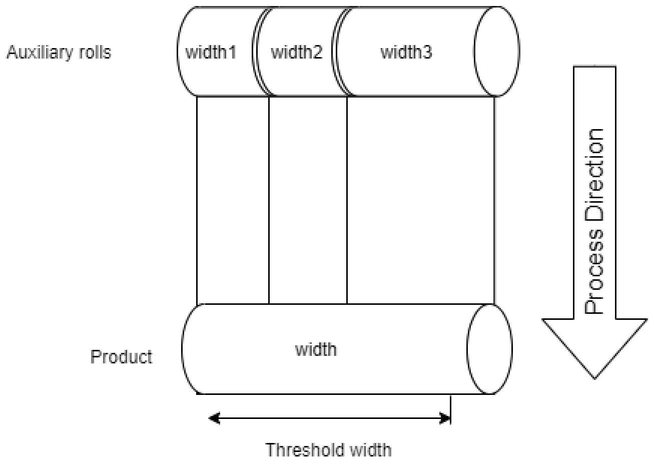


Figure 1.1: Illustration of Skiving Process

The primary purpose of the pure SSP is to construct the set of optimal pattern combinations that maximizes the amount of the desired output under availability constraints for small items. If a product allows being produced through skiving, it is possible to minimize the waste of leftovers and raw materials. Due to its $\mathcal{NP}$-hard structure, enumerating all possible minimal skiving patterns is difficult, especially for large-sized problems. The related literature offers a variety of solution approaches such as pattern-based models [Zak, 2003], arc flow models based on graph theory [Martinovic et al., 2020], or assignment models [Martinovic et al., 2016]. In terms of the methodology, the integration of Column Generation (CG) and Branch and Bound ($B\&B$) is the most common approach for the Integer Programming (IP) structure of pure SSP's mathematical model [Zak, 2003]. Due to the excessive execution time required to reach an optimal solution for large-sized problems and the curse of dimensionality, the majority of the proposed solution methods involve heuristics. These heuristic methods for the SSP aid in obtaining integer solutions after solving the linear relaxation [Wang et al., 2020] and constructing the minimal pattern sets as an input for the mathematical model [Chen et al., 2019]. While metaheuristic optimization methodologies are highly successful in similar problems such as the CSP, their performance on the SSP remains considerably unexplored in the literature.

The SSP literature overwhelmingly solves the deterministic SSP, assuming that the decision-maker has perfect information on all parameters. This assumption leads to unrealistic results as real-world problems involve a great deal of uncertainty in various parameters such as the demand for products, resource availability levels, yields, set-up and processing times, or costs. The deterministic planning by the sole consideration of expected values fails to provide an adaptive decision that can minimize the risk caused by the volatility of these parameters. While many studies have investigated the stochastic version of CSP where the demand [Beraldi et al., 2009; Alem et al., 2010] or the yield [Kazemi Zanjani et al., 2013] are random variables, the literature considerably lacks the stochastic solution approaches for the SSP.

First, in chapter 5, in order to explore the stochastic version of the SSP which has not been explored before in the literature, the pure SSP is extended by including the production cost driven by the patterns used in the skiving process and the random demand. A cost minimization model for the stochastic SSP is formulated, and a new solution methodology is implemented for this unexplored stochastic SSP. We employ a two-stage stochastic programming (SP) with a recourse model [Birge and Louveaux, 2011; Kall et al., 1994; Shapiro et al., 2014] to the problem. In this two-stage approach, skiving process (production) decisions are made before the demand occurs. This approach is called “here-and-now” in the literature as opposed to “wait-and-see” or ”scenario analyses” where decisions are made after the values of random values are revealed [Alem et al., 2010; Beraldi et al., 2009; Birge and Louveaux, 2011; Wets, 2002]. Replications of minimal skiving patterns are the scenario-independent first-stage decision variables, as these decisions are made before any scenario occurs. The underproduction and overproduction amounts constitute the scenario-dependent decision variables at the second stage. These variables represent the recourse action that ultimately must be taken when any possible scenario takes place.

The proposed solution methodology for the stochastic SSP includes the integration of Column Generation Algorithm (CG) [Gilmore and Gomory, 1961], Progressive Hedging Algorithm (PHA) [Rockafellar and Wets, 1991], and Branch and Bound(B&B) [Scheithauer, 2017]. In the solution methodology, the stochastic problem is decomposed into deterministic subproblems according to the number of scenarios. For each deterministic subproblem, the CG is used to obtain the minimal pattern set with minimum trim loss. In other words, the minimal pattern set having the minimum production cost is obtained for each deterministic subproblem. Given these patterns, the PHA obtains the solution for the LP relaxation of the stochastic SSP. Finally, this solution is discretized using B&B, where PHA solves the stochastic model at every node. Finally, the results of the proposed solution methodology for the stochastic SSP are presented.

Next, in chapter 6 the stochastic SSP presented in chapter 5 is extended in several dimensions by including *i*) the set-up cost for each pattern change, *ii*) the raw material costs of the required small items, *iii*) the required quantities of small items to skive. Furthermore, we implement a heuristic solution methodology for the multi-product SSP under a stochastic environment. We propose a new mathematical model for weakly heterogeneous large items under stochastic demand and stochastic waste rate, and employ a two-stage stochastic programming (SP) with a recourse model [Birge and Louveaux, 2011; Kall et al., 1994; Shapiro et al., 2014] for this stochastic version of the SSP problem. In the two-stage stochastic programming approach, skiving process (production) decisions are made before the demand occurs. Another independent random variable at this stage is the waste rate for overall production. Raw material quantities and replication of each skiving pattern are the scenario-independent first-stage decision variables. The underproduction and overproduction amounts constitute the scenario-dependent decision variables at the second stage. These variables represent the recourse action that must be taken after any possible scenario takes place. Moreover, we develop a two-phase solution methodology where the first phase generates the minimal skiving pattern set, and the second phase aims to minimize production costs. This two-phase procedure runs recursively until a certain calculated target production amount can be produced. The first phase implements the Dragonfly Algorithm (DA) [Mirjalili, 2016] and generates minimal skiving patterns. The output of this first phase serves as an input for the second phase, where we implement numerous scenarios for the random variable combinations and employ the Sample Average Approximation (SAA) method for the two-stage stochastic programming model to obtain a solution for the stochastic SSP [Shapiro et al., 2014; Shapiro and Homem-de Mello, 1998].

It should be emphasized that we ***do not*** mention the terms "two-phase" and "two-stage" interchangeably. The term "two-phase" belongs to the overall solution procedure. The first phase corresponds to the DA that generates the skiving patterns. The second phase indicates the two-stage stochastic programming with a recourse action model that minimizes production costs. The first stage of the stochastic programming model makes the production decision before any demand occurs, and the second stage of the stochastic programming penalizes the overproduction and underproduction amounts after the demand occurs.

While this study is a single-objective analysis that minimizes total production costs, it also aims to maintain an acceptable trim loss level as the objective of the DA in the first phase.

A multi-objective version of the SSP has not been investigated in the literature yet. However it is important to explore a multi-objective version of the SSP where there are several competing objectives for the paper production industry. Therefore, we extend pure SSP to a multi-objective problem where we minimize the total trim loss (waste), number of items used in a product, and total number of set-ups simultaneously, while satisfying product demand. Trim loss can be explained as the unused part (waste part) between the pattern width used in the production and threshold width of the product. It is important to use the minimum number of items in a product, because the number of welds in a pattern which is used to produce a product, effects the quality of the product. Briefly, if the number of welds (number of items) in a product increases, the quality of the product becomes poor. Finally, set-up can be explained as the pattern change in the skiving process. We used lexicographic method [Marler and Arora, 2004] to find efficient (Pareto optimal) solutions to this multi-objective problem. Since raw material is very expensive in the paper production or printing industry, the order of importance of the objectives can be denoted as the total trim loss >> number of items >> total set-up number according to decision makers. CG and IP are used together in a lexicographic method to solve the multi-objective problem. In addition, we propose a heuristic to obtain a feasible solution for the multi-objective problem where the Dragonfly Algorithm (DA) [Mirjalili, 2016] is integrated with a type of constructive heuristic. DA finds the minimal patterns with the minimum trim loss and the minimum number of items, and then constructive heuristic [Poldi and Arenales, 2009] replicates efficient patterns which satisfy the demand while minimizing the number of set-ups.

The organization of this dissertation is as follows: Chapter 2 elaborates the relevant literature, and chapter 3 overviews the standard definition and mathematical formulation for the pure SSP. In chapter 4, a general formulation of the two-stage stochastic programming with recourse action is presented. Integration of the Column generation, Progressive Hedging Algorithm, and Branch and Bound for the stochastic SSP is demonstrated in chapter 5. A two-phase pattern generation and production planning procedure is proposed for the extended version of the stochastic SSP in chapter 6, followed by lexicographic method for the multi-objective version of the SSP for which CG and IP are used to solve in chapter 7. Moreover, a heuristic approach and the application of lexicographic method are proposed for the multi-objective version of the SSP in chapter 7. Finally, conclusions are given in chapter 8.

2. LITERATURE REVIEW

The SSP was first introduced by Johnson et al. [Johnson et al., 1997] as an integral part of the CSP. They combined the CSP and the SSP into one problem as the cutting-and-skiving stock problem (CSSP), in which a two-stage process is used to model the cutting of large items and reusing of leftovers. They proposed a pattern-based mathematical formulation and a commercial software (MAJIQTRIM) in which heuristics and linear programming are utilized to solve the CSSP. The SSP was not recognized as a stand-alone problem until Zak [Zak, 2003] conducted a theoretical analysis comparing the SSP and the CSP models. Zak's study was the first to prove that the SSP was not the dual form of CSP. According to Zak [Zak, 2003], the SSP shares some input data similarities with the CSP in terms of item widths, the customer demand, and the scalar knapsack capacity. However, the SSP and the CSP have different pattern matrices because of having different structures. The CSP follows the structure of a set packing problem, whereas the SSP follows the structure of a set covering problem [Wäscher et al., 2007]. In the light of these findings, Zak [Zak, 2003] reported that the SSP is not the dual form of the CSP and introduced the SSP as an independent challenge in Combinatorial Optimization [Zak, 2003; Martinovic and Scheithauer, 2018]. Martinovic and Scheithauer [Martinovic and Scheithauer, 2019, 2016a,b] denoted that the SSP is structurally closer to the Dual Bin Packing Problem (DBPP), also known as a special type of the Bin Covering Problem (BCP). Nonetheless, the SSP has differences with DBPP both in formulation and solution approaches [Zak, 2003]. Martinovic et al. [Martinovic and Scheithauer, 2016a] pointed these differences as the heterogeneity level of small-item sizes and their availabilities based on the study by Wäscher et al. [Wäscher et al., 2007]. According to this study [Wäscher et al., 2007], the SSP can be associated with having weakly heterogeneous small items, while the DBBP includes strongly heterogeneous small items. Additionally, the DBPP formulation considers each type of small item's availability as equal to 1 [Assmann et al., 1984]. Zak [Zak, 2003] extended this problem by incorporating higher availability values for small items. The mathematical formulations and solution approaches of the DBPP are based on item-oriented models and heuristics.

Similar to the CSP, one challenge in the SSP is obtaining integer solutions for large problems. As a result, two-phase approaches are common where the first-phase solves a linear relaxation of the problem and the second-phase discretizes the solution. Arbib and Marinelli [Arbib et al., 2002] solved the CSSP with trim loss minimization in gear-belt production with a Column Generation (CG) [Gilmore and Gomory, 1963] and a Branch-and-Bound (*B&B*) algorithm. They extended the single-period CSSP model proposed in [Arbib et al., 2002] into a multi-period problem in which a *B&B* algorithm is used to solve a small-sized CSSP [Arbib and Marinelli, 2005]. Similar to this study, Zak [Zak, 2003] proposed CG [Gilmore and Gomory, 1961] for the linear relaxation of the problem and a *B&B* algorithm to obtain integer solutions for SSP. In the meantime, the pattern-based model and column generation (CG) proposed by Gilmore and Gomory [Gilmore and Gomory, 1961, 1963] for large-scale CSP are also suitable for the SSP [Zak, 2003]. Besides, the pure SSP can be categorized as an output maximization problem with the perspective of the study by Wäscher et al. [Wäscher et al., 2007].

Recently, Ágoston [Ágoston, 2019] investigated 1D-CSP with a skiving option in which single-sized pipes are cut into smaller-sized pipes, and residual parts can be welded together using the skiving process to obtain extended pipes to aid in the fire fighting system. For safety reasons, there should be at most one welding procedure on the pipe. Ágoston further modeled the CSP as a Mixed Integer Programming (MIP) model, including sequential cutting and welding patterns, and proposed a three-stage algorithm that is used to minimize the stock level and different pattern costs.

While metaheuristics are not an explored field in the SSP realm, they have been widely implemented for similar combinatorial problems such as the CSP. Well-known metaheuristics in this area are Tabu Search [Álvarez-Valdés et al., 2002], Simulated Annealing (SA) [Chen et al., 1996; Lai and Chan, 1997; Jahromi et al., 2012], Ant Colony Optimization (ACO) [Ducatelle and Levine, 2001], and its variants or hybridizations [Levine and Ducatelle, 2004], Genetic Algorithm [Hinterding and Khan, 1993; Leung et al., 2001; Lu et al., 2013], Genetic Symbiotic Algorithm [Golfeto et al., 2009], Evolutionary Programming (EP) [Liang et al., 2002], and Hybrid Chemical Reaction Optimization (CRO) [Yang et al., 2017].

The studies mentioned above involve numerical implementations and real-world applications of the SSP. Nevertheless, the literature also includes theoretical analysis. Martinovic is one of the most important pioneers of theoretical analysis of the SSP literature [Martinovic and Scheithauer, 2016a,b; Martinovic et al., 2016; Martinovic and Scheithauer, 2017, 2018, 2019; Martinovic et al., 2020]. In addition to the pattern-based SSP model, Martinovic and Scheithauer [Martinovic and Scheithauer, 2016a] presented a graph theory-based arc flow model for the SSP. They further investigated continuous relaxations of this model to prove equivalences with the pattern-based, assignment, and one-stick models. In a latter study, Martinovic and Scheithauer [Martinovic and Scheithauer, 2016b] 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 upperbound 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 [Martinovic and Scheithauer, 2016b]. They analyzed integer round-down property (IRDP), modified integer round-down property (MIRDP), and non-integer round-down property (non-IRDP) in discretizing the obtained solutions [Martinovic and Scheithauer, 2016b]. Furthermore, Martinovic et al. [Martinovic et al., 2020] improved the standard arc flow model, which was previously presented in [Martinovic and Scheithauer, 2016a] 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 [Martinovic and Scheithauer, 2018].

In addition to theoretical analysis, Martinovic et al. [Martinovic et al., 2016] focused on the spectrum aggregation problem in cognitive radio networks, as a real-world application of the SSP. This problem involves allocating spectrum resources of radio networks where primary licensed users occupy predetermined parts of a frequency band. The available bandwidths, spectrum holes, were too small to meet the bandwidth requirements of secondary users. They investigated the aggregation of empty spectrum holes to obtain a large-enough bandwidth for secondary users under hardware limitations [Martinovic et al., 2016]. They compared the results and computational complexities of Zak's standard pattern-based SSP model, an arc flow model, and an assignment model. According to numerical computations, the assignment model remarkably reduced the computational complexity.

The stochastic SSP literature is not thoroughly investigated, and it might be possible to use solution methods that have previously been employed on the CSP. Therefore, we also elaborate on the stochastic CSP literature in this section. CG is the most commonly applied method to retrieve an initial, non-integer solution [Alem et al., 2010; Demirci et al., 2008; Jin et al., 2012] for the CSP. Alem et al. [Alem et al., 2010] implemented a CG for a two-stage stochastic programming model where the demand was random in CSP. Jin et al. [Jin et al., 2012] developed a two-stage stochastic integer programming model for the CSP that makes inventory replenishment decisions in the first stage and the cutting decisions in the second stage. Moreover, CG is used for the LP relaxation of the cutting stock problem, and residual heuristic is used to obtain integer solutions. Another solution methodology for the stochastic CSP by Demirci et al. [Demirci et al., 2008] proposed the combination of the CG and the L-shaped algorithm. In the study of Chauhan et al. [Chauhan et al., 2008] CG and $B\&B$ algorithms was accompanied by both fast pricing heuristic and marginal cost heuristic in a stochastic problem where the demand is random.

As an alternative method to CG, Beraldi et al. [Beraldi et al., 2009] suggested a two-stage stochastic programming model with Lagrangian decomposition and B&B to decompose the problem into subproblems. These subproblems are fed into a proposed heuristic in the second stage. Moreover, Sculli [Sculli, 1981] considered defects as random variables due to the winding process in the CSP. Alem and Morubito [José Alem and Morabito, 2012] employed stochastic demand and set-up times for cutting patterns in furniture production. Kazemi-Zanjani et al. [Kazemi Zanjani et al., 2013] presented a two-stage stochastic Linear Programming (LP) approach for the CSP a random variable with discrete probability distribution. They used the Sample Average Approximation (SAA) scheme to approximate the problem to avoid high computational time caused by a large number of scenarios in stochastic programming.

In the first part of this study, Zak's standard pattern-based SSP output maximization model [Zak, 2003] is extended to the single product stochastic cost minimization SSP model by using two-stage stochastic programming paradigm in which demand is random. Furthermore, as a solution methodology, CG, PHA, and *B&B* algorithms are integrated to solve the stochastic problem. Moreover, in the second part of this study, we extend the pure SSP model by including production, set-up, and raw material costs. Including the quantity of each raw material as a decision variable converts the model into an assortment problem, as well. Moreover, as a contribution, the multi-product version is adapted to the standard pattern-based SSP model [Zak, 2003; Martinovic et al., 2016]. Another important contribution of our study is to handle different sources of uncertainty, that are, the product demand and the waste rate. As a main contribution, two-phase algorithm is proposed as a solution methodology. At the first phase, the DA produces skiving patterns. At the second phase, stochastic multi-product SSP is solved by a two-stage Stochastic Program in which we implement an SAA [Shapiro et al., 2014] approach to cope with a large number of scenarios. Finally, a recursive solution procedure between the DA and the SAA is developed for the extended large-sized Stochastic SSP. At the end, a multi-objective case which is not considered for the SSP in the literature is analyzed. Three competing objective functions which are the minimization of the total trim loss, the number of items in each product, and the total set-up number are considered in the multi-objective approach. Lexicographic method is applied in which CG and IP are used to solve the multi-objective version of the SSP.

3. STANDARD DEFINITION AND MATHEMATICAL FORMULATION FOR THE PURE SSP

In this section, we present fundamental definitions and the formulation for the pure SSP [Martinovic et al., 2016; Martinovic and Scheithauer, 2016a,b, 2018]. Let $E := (m, l, L, b)$ denote an instance of the SSP, where m is the number of small item types. l is an m-element vector denoting the width of each small item type, and it is comprised of l_i's where $i \in \mathbb{I}$ and $|\mathbb{I}| = m$. L is the width for the large item [Martinovic et al., 2016; Martinovic and Scheithauer, 2016a,b, 2018]. The skiving process should produce large items with the minimum width of L. Finally, b is an m-element vector that denotes the availability of each small item type i, and it is comprised of b_i's [Martinovic et al., 2016; Martinovic and Scheithauer, 2016a,b, 2018]. For the sake of comprehensibility and standardization of terms, we will refer to small items as *items*, and to large items as *products*. All input data is assumed to be positive integers ($\mathbb{Z}_+$) and satisfy $L > l_1 > ... > l_m$ as an ordered set. Any feasible arrangement of items to form a product with the minimum width L is called *a feasible pattern* of instance E. Any feasible pattern can be represented by a non-negative vector $a = (a_1, ..., a_i, ..., a_m)^T \in \mathbb{Z}_+^m$, and $a_i \in \mathbb{Z}_+$ represents the number (recurrence) of ith item in a pattern. Finally, $P(E) := \{a \in \mathbb{Z}_+^m \mid l^T a \geqslant L\}$ represents a feasible pattern set [Martinovic et al., 2016; Martinovic and Scheithauer, 2016a,b, 2018].

A *minimal pattern* is the feasible pattern in which the width of the product becomes smaller than the threshold L, when any item were to be removed from the pattern. As a clear representation here, any pattern $\tilde{a} \in P(E)$ such that $\tilde{a} \leq a$ holds component-wise ($\tilde{a}_i \leq a_i$). A minimal pattern set is represented with $P^*(E)$. Moreover, a pattern $a \in P(E)$ is an *exact pattern*, if $l^T a = L$. Thus, every exact pattern is a minimal pattern. x_j is the decision variable representing the frequency of minimal patterns $a^j = (a_{1j}, \dots, a_{mJ})^T$ where $a \in \mathbb{Z}_+^m$ and $j \in \mathbb{J}^* = 1, \dots, J$ is the index of the minimal pattern set $P^*(E)$. Eventually, the objective function of the pure SSP is formulated as [Martinovic et al., 2016; Martinovic and Scheithauer, 2016a,b, 2018]:

$$z^*(E) = max\left\{ \sum_{j \in \mathbb{J}^*} x_j | \sum_{j \in \mathbb{J}^*} a_{ij} x_j \leq b_i, i \in \mathbb{I}, x_j \in \mathbb{Z}_+, j \in \mathbb{J}^* \right\}. \tag{3.1}$$

In addition, a minimal pattern $a^j \in P^*(E)$ that satisfies $a_{ij} \leq b_i, \forall i, j$ is called a *minimal proper pattern*. In other words, a minimal proper pattern also satisfies the item availability constraints. Otherwise, it is called a *non-proper minimal pattern* [Martinovic et al., 2016; Martinovic and Scheithauer, 2016a,b, 2018]. We extend and define the propriety for the pattern set, as well. A proper minimal pattern set is defined as $P_P^*(E|x_j) := \{a : a^j \in P^*(E), \sum_{j \in \mathbb{J}^*} a_{ij} x_j \leq b_i, i \in \mathbb{I}\}$, that is, the patterns in $P_P^*(E)$ can produce a predetermined production amount with available items.

Furthermore, without loss of generality, $E := (m, l, L, b)$ is extended by including multiple products of various widths, defined as the *multi-product* case as $E := (m, K, l, L, b)$ where K denotes the number of product types. k is the index of product type $k \in \mathbb{K} := 1, \dots, K$. 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 formulation is extended as:

$$z^*(E) = max\left\{ \sum_{j \in \mathbb{J}^*} \sum_{k \in \mathbb{K}} x_{jk} | \sum_{j \in \mathbb{J}^*} \sum_{k \in \mathbb{K}} a_{ijk} x_{jk} \leq b_i, i \in \mathbb{I}, x_{jk} \in \mathbb{Z}_+, j \in \mathbb{J}^*, k \in \mathbb{K} \right\}. \tag{3.2}$$

4. GENERAL FOLMULATION OF THE TWO-STAGE STOCHASTIC PROGRAMMING WITH RECOURSE ACTION (SP)

Assume that ξ is a vector of random variables. Decisions taken before the realization of the ξ are called first-stage decision variables and are represented by the vector x. After receiving the values of ξ, information concerning random variables is obtained. Decisions taken after the realization of the ξ form the second-stage decision variables. They are also called the recourse action and denoted by vector y. Recourse action depends on both the first-stage decision variables and the outcomes of random variables. A two-stage stochastic programming model with recourse action formulation is presented by [Birge and Louveaux, 2011] as follows:

$$\min_{x} \quad C^T x + \mathbb{E}_\xi Q(x, \xi) \tag{4.1}$$

$$s.t. \quad Ax = b, \tag{4.2}$$

$$x \geq 0, \tag{4.3}$$

where $Q(x, \xi) = min\{c^T q | Wq = h - Tx, q \geq 0\}$. ξ is the random vector formed by the c^T, h^T, T components and $\mathbb{E}_\xi$ is the expectation according to ξ where T is the technology matrix, h is the right hand side, W is the recourse matrix, and c^T is the penalty cost vector of recourse decisions [Birge and Louveaux, 2011]. From the general formulation of two-stage stochastic programming Eq. 4.1, the value for random variable ξ corresponds to the scenario $s \in \mathbb{S} := 1, \ldots, S$ with the probability P_s, and Eq. 4.1 can be also written as follows [Birge and Louveaux, 2011]:

$$\min_{x} \quad C^T x + \sum_{s \in \mathbb{S}} P_s Q^s(x, \xi) \tag{4.4}$$

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

5. INTEGRATION OF THE COLUMN GENERATION, PROGRESSIVE HEDGING ALGORITHM AND BRANCH AND BOUND ALGORITHM FOR THE STOCHASTIC SKIVING STOCK PROBLEM

5.1 Two-Stage SP formulation of Stochastic SSP

Nomenclature for the Stochastic SSP Model

Indices

i	index of (small) item types , $i \in \mathbb{I} := \{1, ..., m\}$;
j	index of minimal patterns, $j \in \mathbb{J}^* := \{1, ..., J\}$;
s	index of scenarios, $s \in \mathbb{S} := \{1, ..., S\}$.

Parameters

a_{ij}	the number of item type i in pattern j, (being generated by CG);
b_i	the available amount of item type i;
c	cost for unit width (i.e. 1 Euro/mm);
C^B	the cost of underproduction;
C^H	the cost of overproduction;
C_j^{Pr}	the production cost which depends on the width of pattern j, $C_j^{Pr} = c * \sum_{i \in \mathbb{I}} l_i a_{ij}, \forall j \in \mathbb{J}$;
d_s	the demand in scenario s;
L	the threshold (lower bound) width of product;
l_i	the width of each item type i;
P_s	the probability of scenario s.

First-stage decision variables

x_j	the frequency (replication) of pattern j;

Second-stage decision variables (recourse actions)

q_s^+	the overproduction amount in scenario s;
q_s^-	the underproduction amount in scenario s.

We present the nomenclature of our formulation for two-stage stochastic SSP model; it should be kept in mind that different phases in the solution methodology will also have their own notations. The pure SSP is extended to the cost minimization problem, including production, overproduction, and underproduction costs. Furthermore, the demand (D) is assumed to be a random variable. Each possible value for the demand is a scenario $s \in \mathbb{S} := 1, \dots, S$ with probability P_s such that $P_s \geq 0$ and $\sum_{s \in \mathbb{S}} P_s = 1$. $D(s) = (d_1, \dots, d_s)$ represents scenario realizations. As a result, the model is constructed as a two-stage stochastic recourse model.

We split the decision variables of the two-stage stochastic programming SSP model into two groups, the first stage decision variables and the second stage decision variables. The frequency of each pattern for the product x_j is the first stage decision variable. The values of a_{ij} are obtained before solving SP as coefficients of the "column" being generated by CG, and used in the first stage of the SP as parameters. The overproduction amount q_s^+ and the underproduction amount q_s^- are the second stage decision variables. Their values depend on the scenarios and the first stage decisions. We can then construct the first stage of the objective function as:

$$\min \quad \sum_{j \in \mathbb{J}^*} C_j^{Pr} x_j \tag{5.1}$$

and $s \in \mathbb{S}$, Q^s function in Eq.4.4 is defined as:

$$Q^s(x, \xi) \; = \min \quad (C^H q_s^+ + C^B q_s^-) \tag{5.2}$$

Finally, given Eq.s (5.1, 5.2) and the nomenclature, the deterministic equivalent of the stochastic SSP model with the instance of $E := (m, l, L, b)$ is given through Eq.s 5.3-5.8:

$$\min \; z_{SP} = \quad \sum_{j \in \mathbb{J}^*} C_j^{Pr} x_j + \sum_{s \in \mathbb{S}} P_s (C^H q_s^+ + C^B q_s^-) \tag{5.3}$$

$$\text{s.t.} \quad \sum_{j \in \mathbb{J}^*} a_{ij} x_j \leq b_i, \quad \forall i \in \mathbb{I} \tag{5.4}$$

$$q_s^+ - q_s^- = \sum_{j \in \mathbb{J}^*} x_j - d_s, \quad \forall s \in \mathbb{S}, \tag{5.5}$$

$$x_j \in \mathbb{Z}_+, \quad \forall j \in \mathbb{J}^* \tag{5.6}$$

$$a_{ij} \in \mathbb{Z}_+, \quad \forall i \in \mathbb{I}, j \in \mathbb{J}^* \tag{5.7}$$

$$q_s^+, q_s^- \geq 0, \quad \forall s \in \mathbb{S}, \tag{5.8}$$

The objective function (z_{SP}) minimizes both the first stage and the second stage costs. The first stage cost is the production cost determined according to each pattern type used for the skiving process to form the product. The second stage costs include the overproduction and the underproduction costs. Some examples of these costs are lost sales, overtime production costs, or product supplies from other sellers to meet the demand. Constraint (5.4) balances the quantity of each small item type by multiplying the total replications of patterns and the number of items in each pattern. This constraint imposes the availability of small items in the warehouse or the supplier. In other words, it controls the supply limitation of small items. Constraint (5.5) balances the overproduction amount or underproduction amount at the second stage according to each scenario and the production quantity at the first stage.

5.2 The Proposed Solution Methodology for the Stochastic SSP

The column generation algorithm (CG), the Progressive hedging algorithm (PHA), and Branch and Bound ($B\&B$) are integrated to solve the stochastic integer problem (SIP). CG provides the minimal pattern set with the minimum trim loss. PHA is run in the B&B procedure iteratively to obtain the integer optimal solution of each node. The following subsections explain the solution methodology in detail.

5.2.1 Column Generation Algorithm for the SSP

Column Generation is a simplex-based algorithm for large-sized linear programming models [Gilmore and Gomory, 1961]. It aims to derive a subset of valuable variables, avoid nonbasic ones. The CG splits the problem into two subproblems, the master LP problem (MP) (Eq. 5.9) and minimum knapsack subproblem (KP) (Eq. 5.10). For the pure SSP whose objective function is output maximization, these two subproblems are solved iteratively by providing interaction between two problems.

$$(MP) : z_k = max\{\sum_{j\in\mathbb{J}_k} x_j | \sum_{j\in\mathbb{J}_k} a^j x_j \leq b, x_j \geq 0, j \in \mathbb{J}_k\} \tag{5.9}$$

where x_j is the decision for frequency of pattern j , a^j represents the vector of pattern j, and b represents the availability vector for small items.

$$(KP) : \bar{c} = min\{u^T a^\tau | l^T a^\tau \geq L, \tau \in \mathbb{J}\} \tag{5.10}$$

Zak [Zak, 2003] proved that CG is also a valuable method for SSP. This output

```
1  START ;
2  Determine parameters $E := (m, l, L, b)$;
3  Set $k := 0$, Run CG;
4  Choose a subset $(a^j : j \in \mathbb{J}_0)$ of the columns such that;
     $z^k = max\{\sum_{j\in\mathbb{J}_k} x_j | \sum_{j\in\mathbb{J}_k} a^j x_j \leq b, x_j \geq 0, j \in \mathbb{J}_k\}$ master problem (MP) is
     solvable;
5  while $\bar{c} < 1$ do
6      Solve the MP $(z_k)$; $x^k := \arg\max(z^k)$ is solution and $u^k$ is the vector of simplex
         multipliers.
7      Solve the subproblem; $\bar{c} = min\{u^T a^\tau | l^T a^\tau \geq L$,for some $\tau \in \mathbb{J}\}$;
8      Set $k : k + 1$, $\mathbb{J}_k := \mathbb{J}_{k-1} \bigcup \{\tau\}$
9  end
10 STOP;
```

Algorithm 1: Column Generation for SSP

maximization problem is solved as the master LP and and, the dual solution is fed to the minimum knapsack subproblem as a coefficient for the objective function of the knapsack problem as given in Eq. 5.10. Next, the minimum knapsack problem is solved to obtain the primal solution, and this primal solution is converted to a new column for the master LP. The CG algorithm continues until the optimal solution is obtained. For the SSP, we generate minimal patterns by using every single item as an initial minimal pattern set. Moreover, the minimum knapsack problem is used to generate candidate columns for the MP. When the optimal solution value is greater than or equal to 1.0 for the KP, the algorithm stops with $z^*_{ssp} = z^k$, $x^*_{ssp} = x^k$ and

$\mathbb{J}^* := \mathbb{J}_k$ [Zak, 2003]. The pseudocode is presented in Alg.1, where u is a vector of a dual variables obtained from MP and then used as a coefficient vector for KP. a^τ is a vector of decision variables of KP; in other words, it is a new column that has the potential to improve the objective function value of MP. L is threshold width, and l represents the vector of small item widths [Zak, 2003].

5.2.2 Progressive Hedging Algorithm

Progressive Hedging Algorithm (PHA) is a decomposition-based algorithm proposed by Rockafellar and Wets [Rockafellar and Wets, 1991]. Even if other stochastic programming solvers fail to attain a solution due to a large number of scenarios and variables, the PHA still attains the solution. PHA does not guarantee convergence to the optimal solution if the model involves discrete variables. Therefore, for the stochastic IP and MIP models, the PHA is implemented as a heuristic method [Aydin, 2012]. PHA splits the problem into subproblems according to each scenario to relax the non-anticipativity constraints, which means the first stage decision variables should be equal in all scenarios. Then, the solutions of the subproblems are forced to converge to be equal while solving each subproblem with a quadratic model re-defined by including penalty terms [Aydin, 2012].

There are several applications of PHA for stochastic IP and MIP models. For example, Lokketangen and Woodruff [Lokketangen and Woodruff, 1996] proposed a Tabu Search and PHA combination for the stochastic multistage MIP (0-1) problem. The Tabu Search algorithm is used to obtain solutions of subproblems, and the PHA is applied to coordinate and blend the solutions of subproblems for the convergence. Moreover, the PHA is used in the metaheuristic framework where subproblems are solved heuristically for the multistage stochastic lot-sizing problem [Haugen et al., 2001]. A B&B procedure is used with PHA for a stochastic MIP model where the subproblems are modified in each node [Atakan and Sen, 2018]. Pseudocode of the PHA [Rockafellar and Wets, 1991] is as follows (Alg. 2):
Here, ϵ is the stopping criteria, ρ is the penalty constant, k represents the index

```
START ;
Determine $\epsilon$, $\rho$;
Set $k := 0$ as iteration number ;
$\forall s \in \mathbb{S}$, $x_s^k := \arg\min_{x,y_s}(cx + fy_s) : (x, y_s) \in Q(x,\xi)$: $x_s^k \geq 0$ and $y_s \geq 0$, $\forall s \in \mathbb{S}$ in
  each iteration ;
$\bar{x}^k := \sum_{s \in S} P_s x_s^k$;
$\forall s \in \mathbb{S}$, $\omega_s^k := \rho(x_s^k - \bar{x}^k)$;
while $\pi^k \geq \epsilon$ do
    $k =: k + 1$;
    $\forall s \in \mathbb{S}$, $x_s^k := \arg\min_{x,y_s}(cx + \omega^{k-1}x + \frac{\rho}{2}\|x - \bar{x}^{k-1}\|^2 + fy_s) : (x, y_s) \in Q(x,\xi)$;
    $\bar{x}^k := \sum_{s \in S} P_s x_s^k$;
    $\forall s \in \mathbb{S}$, $\omega_s^k := \omega_s^{k-1} + \rho(x_s^k - \bar{x}^k)$;
    $\pi^k := \sum_{s \in \mathbb{S}} P_s \|x_s^k - \bar{x}^k\|$;
end
STOP;
```

Algorithm 2: Progressive Hedging Algorithm

of iteration number, and π^k represents the Euclidean distance in iteration k. The vector x_s^k represents decision variables for each scenario in iteration k, and $\bar{x}^k$ is the average of decision variables in iteration k. The f represents the penalty cost vector of recourse decisions, and y_s is the vector of second stage decision variables in scenario $s \in \mathbb{S}$. ω_s^k is the dual multiplier for each scenario in iteration k. The algorithm stops with the solution $x^{PHA} = \bar{x}^k$. If decision variables are continuous, the solution converges to $\bar{x}$ in linear time [Rockafellar and Wets, 1991].

5.2.3 Branch and Bound Procedure

Scheithauer [Scheithauer, 2017] emphasizes that the classical B&B procedure is a common method for the one-dimensional cutting stock problem (1D-CSP) to obtain integer optimal solutions. We integrate the B&B algorithm to our solution methodology to obtain an integer solution from the LP relaxations of the stochastic integer programming (SIP) model. Since we use the PHA algorithm to solve the stochastic program, we run PHA in each node of the B&B search tree until an integer solution is obtained. Let the stochastic version of SSP has an instance $E := (m, l, L, b)$ with the random demand D and a^j, $j \in \mathbb{J}^*$ represents the minimal pattern. All active subproblems in the solution process form the set of Π, including π_k. Π contains the initial problem, π_0: $z_0{:=}z_{sp}$.

```
1  START ;
2  Determine parameters;
3  Initialize $\Pi := \pi_0$ and $z^* := \infty$
4  while $\Pi \neq \emptyset$ do
5      Select a sub problem $\pi_k \in \Pi$ and set $\Pi := \Pi \setminus \{\pi^k\}$
6      Compute $x^k$ of the LP relaxation for $\pi_k$ with $z_k$
7      If $z_k \geq z^*$, then go to (5)
8      If $x^k \in \mathbb{Z}^+$ then $x^* := x^k$, $z^* := z_k$ and go to (4)
9      Determine $j_k$ Eq. 5.11
10     Determine successor problems $\pi_{k,1}$, $\pi_{k,2}$, Eq.s. 5.12-5.13
11     $\Pi := \Pi \cup \{\pi_{k,1}, \pi_{k,2}\}$, go to (4)
12 end
13 STOP;
```

Algorithm 3: Branch and Bound Algorithm

In the branching process, the active subproblem is divided into two successor problems by determining two index sets $J^{\leq}$, $J^{\geq}$. These two index sets demonstrate the patterns which require a replication number in the LP solution in which "at least" or "at most" as bounds of the replication numbers are determined. x_k is the solution vector of the LP relaxation of the k^{th} subproblem $\pi_k \in \Pi$, and k is the number of subproblem. The pseudocode of B&B which is inspired from Scheithauer's study[Scheithauer, 2017] is adapted to the SSP problem (Alg. 3).

$$j_k := \arg\max \{ \, x_j^k : x_j^k \notin \mathbb{Z}^+, j \in \mathbb{J}^* \} \tag{5.11}$$

$u_{j_k} := \lceil x_{j_k} \rceil$ then two successor problems are defined as;

$$(\pi_{k,1}) := (\pi_k) \wedge (x_{j_k} \geq u_{j_k}), (j^{\geq} := j^{\geq} \cup \{j_k\}) \tag{5.12}$$

$$(\pi_{k,2}) := (\pi_k) \wedge (x_{jk} \leq u_{jk} - 1), (j^{\leq} := j^{\leq} \cup \{j_k\}) \tag{5.13}$$

Then, by using the branching strategy, each subproblem becomes as follows:

$$(\pi_k) : z_k := min\{z_{sp} : x_{j_k} \geq u_{j_k}, j \in j^{\geq}, x_{jk} \leq u_{jk} - 1, j \in j^{\leq}, x_j \in \mathbb{Z}^+, j \in \mathbb{J}^*\} \tag{5.14}$$

5.2.4 Integration of CG, PHA, and Branch and Bound Algorithm

We propose the integration of CG, PHA, and B&B for the solution of the stochastic SSP. Initially, the stochastic problem (Fig 5.1.a) is decomposed into S subproblems. In this step, each of the subproblems is denoted as the deterministic cost minimization problem which must satisfy the scenario demand (Fig 5.1.b). Next,

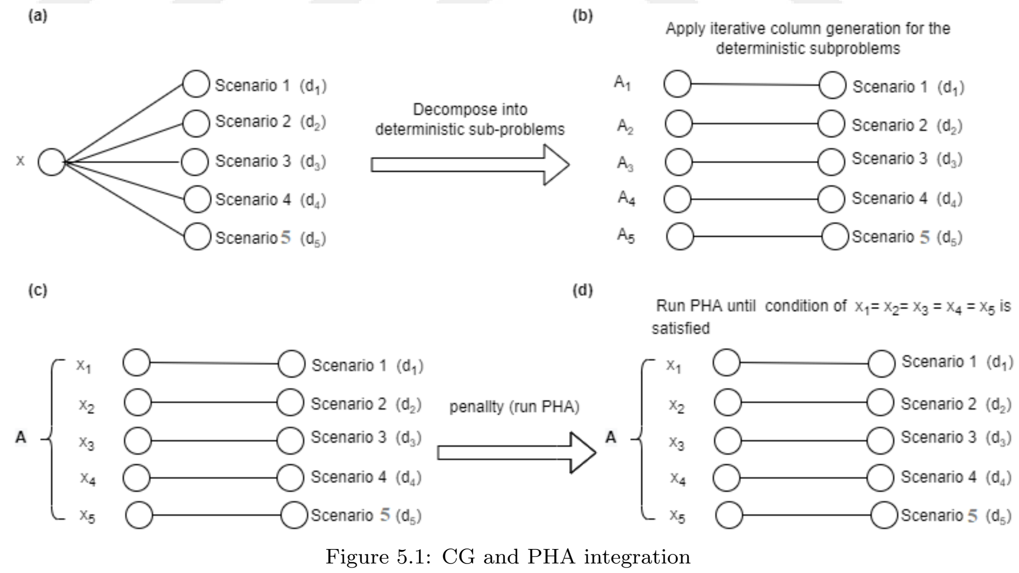

(a)
Scenario 1 (d1)
Scenario 2 (d2)
Scenario 3 (d3)
Scenario 4 (d4)
Scenario 5 (d5)
x
Decompose into deterministic sub-problems
(b)
Apply iterative column generation for the deterministic subproblems
A1
A2
A3
A4
A5
(c)
A
x1
x2
x3
x4
x5
penallty (run PHA)
(d)
Run PHA until condition of x1= x2= x3 = x4 = x5 is satisfied

Figure 5.1: CG and PHA integration

CG is applied to each subproblem to obtain the most efficient pattern set of each subproblem; in other words, the obtained matrix A_s can minimize the total cost of subproblem s. On the other hand, CG for the pure SSP presented in section 5.2.1 is used to obtain the maximum output, and demand is not taken into consideration. Since subproblems are cost minimization problems including demand constraints, it is important to find the most efficient pattern set for each subproblem which can minimize the total cost. Therefore, an auxiliary constraint $l^T a^\tau \leq W, \tau \in \mathbb{J}$ where W represents the target width is added to the minimum knapsack problem (Eq. 5.10) to control the width of the generated pattern. Since the production cost for the product strictly depends on the width of used patterns, the cost of the pattern can be controlled and restricted by controlling W. Then, CG is applied to the subproblem iteratively which is inspired from the dynamic programming approach. At the beginning of iterations, W is set as the threshold width of product ($W = L$). In each iteration, W is increased by 1 unit. It means that the width of patterns generated by the minimum knapsack problem of CG is increased by 1 unit until the demand of the subproblem is satisfied (Fig 5.2). By using CG, the cost of the patterns obtained

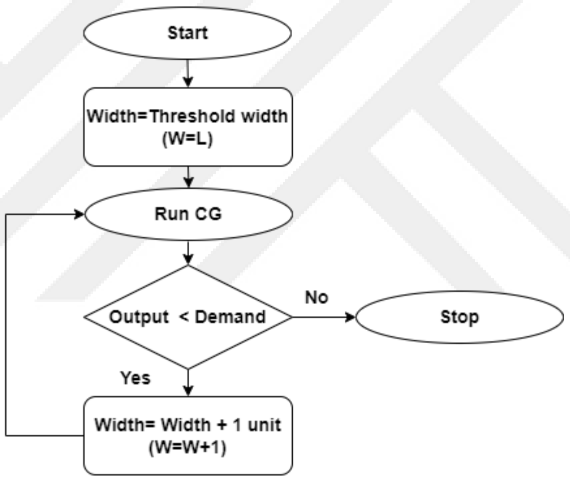


Figure 5.2: Use of Column Generation in the Algorithm

to satisfy the demand and demand fulfilment are controlled at the same time. In this way, the obtained pattern set or matrix A_s which is the combination of the patterns A_s^k for each iteration of CG, such that $A_s = A_s^1 \cup A_s^2 \cup \ldots$, where k is the iteration number $k = \{1, 2, \ldots\}$, can provide the minimum objective function value (minimum cost) for each subproblem when the LP is applied to each subproblem. It must be emphasised that the upperbound for the width W can be denoted as the sum of the threshold width and the width of the largest item type. As the nature of the two-stage stochastic programming, all matrices of subproblems are combined to form a common matrix $\mathbf{A}$ as a parameter where the common pattern matrix can be denoted as $\mathbf{A} = \bigcup_{s \in \mathbb{S}} A_s$. LP is applied to each subproblem to obtain the optimal solution of each subproblem in the PHA by using the given matrix **A** (Fig 5.1.c). Until the decision variable vectors of each subproblem are equal to each other ($X_1 = X_2 = X_3 = X_4 = X_5$), differences of decision variables are penalized by PHA (Fig 5.1.d). PHA algorithm provides relaxation solution and the objective function value of the stochastic SSP model (Eq. 5.3-5.8). Furthermore, we run the B&B to obtain the integer solution of the stochastic SSP (Eq. 5.3-5.8) in which the PHA is run at each node of the B&B tree to solve the B&B subproblems (it must not be confused with PHA subproblems) as in (Alg. 4).

```
START ;
Determine parameters, scenarios $E := (m, l, l, b)$ and;
while $s \neq S$ do
    set $W = L$;
    while $z_{ssp} \leq d_s$ do
        $W = W + 1$;
        Run CG to solve $z_{ssp}$ Eq.3.1 ;
        $A_s^k := a_{ij}^k$, $k$ is the iteration number $k := \{1, 2, \dots\}$ ;
    end
    $A_s = A_s^1 \cup A_s^2 \cup A_s^3 \dots$;
end
$\mathbf{A} = \bigcup_{s \in \mathbb{S}} A_s$;
while $x^{PHA} \notin \mathbb{Z}^+$ do
    Apply BB algorithm to run PHA in each node
end
STOP
```

Algorithm 4: CG, PHA, *B&B* Combination

5.3 Numerical Illustration

In this section, we present a small instance of $E := (m, l, L, b)$ of a stochastic version of the SSP as an illustrative example. In this example, E:=(6, (25, 35, 45, 60, 75, 85), 100, (250, 200, 300, 100, 100, 100)) where $D \sim Pois(150)$ is the random variable representing the demand of the product. $C_j^{Pr} = \sum_{i \in \mathbb{I}} l_i a_{ij}, \forall j \in \mathbb{J}^*$ is the variable production cost for each product according to used pattern type. Moreover, let $C^H = 320$ and $C^B = 800$. Five scenarios are determined for the illustrative example. For the PHA parameters, the penalty constant and stopping criterion are defined as $\rho = 2$ and $\epsilon = 10^{-5}$, respectively. Finally, we present the computations in each step of the illustrative example to explain the solution methodology in detail.

1. Stochastic problem is decomposed into S deterministic subproblems. To simplify the illustration, the number of scenarios is determined as $S = 5$. Therefore, 5 deterministic subproblems are considered.
2. The CG is run for the deterministic subproblem s to obtain the most efficient minimal pattern set, matrix $A_s, \forall s \in \mathbb{S}$ that can satisfy the demand of each deterministic subproblem. The matrix **A** is presented below. Each column of matrix **A** represents a pattern, and each row represents an item type. Each value of a_{ij} indicates the amount of item i used in pattern j. The obtained pattern sets (matrices, A_s) for subproblems are presented below.

$$A_1 = \begin{bmatrix} 4 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, A_2 = \begin{bmatrix} 4 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}, A_3 = \begin{bmatrix} 4 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 3 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, A_4 = \begin{bmatrix} 4 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 3 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$A_5 = \begin{bmatrix} 4 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 3 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

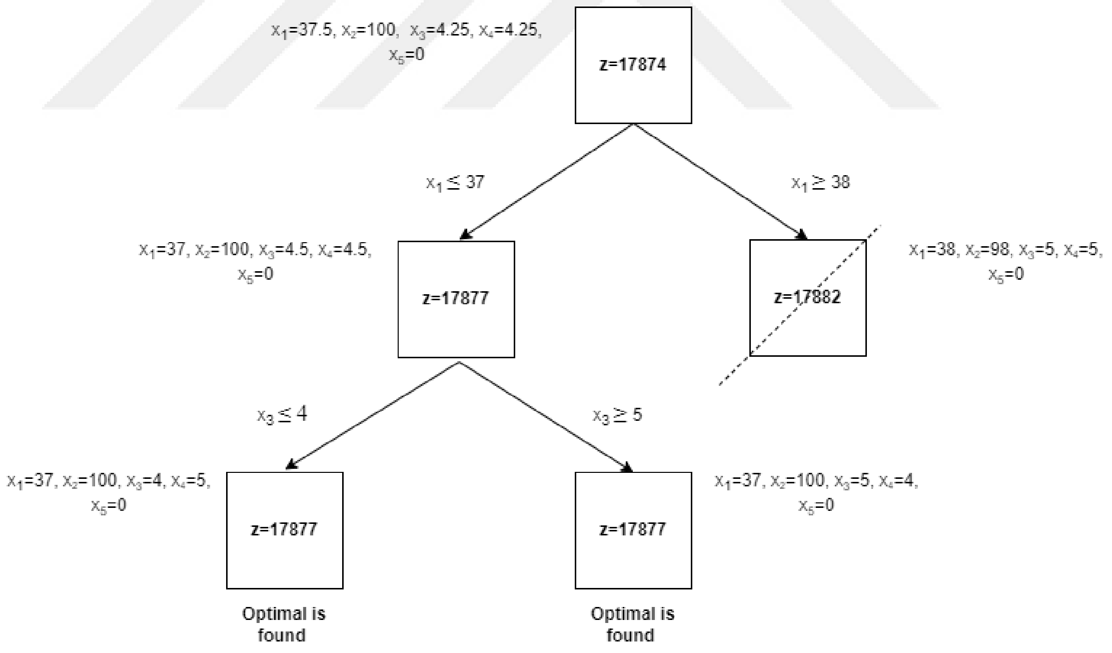
Then, all matrices are combined in the master matrix $\mathbf{A} = \bigcup_{s \in \mathbb{S}} A_s$;

$$\mathbf{A} = \begin{bmatrix} 4 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 3 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

Moreover, variable production cost vector according to each pattern is presented as:

$C^{Pr} = \begin{bmatrix} 100 & 100 & 105 & 105 & 105 \end{bmatrix}$

3. PHA is run in the first node of B&B search tree in order to obtain a solution for LP relaxation of the stochastic SSP (Eq.s 5.3-5.8) by using **A** obtained by CG. Initial objective function value and solution are obtained as: $z^{SP} = 17874$ and $x^{SP} = \begin{bmatrix} 37.5 & 100 & 4.25 & 4.25 & 0 \end{bmatrix}$;



x_1=37.5, x_2=100, x_3=4.25, x_4=4.25, x_5=0
z=17874
$x_1 \leq 37$
$x_1 \geq 38$
x_1=37, x_2=100, x_3=4.5, x_4=4.5, x_5=0
z=17877
z=17882
x_1=38, x_2=98, x_3=5, x_4=5, x_5=0
$x_3 \leq 4$
$x_3 \geq 5$
x_1=37, x_2=100, x_3=4, x_4=5, x_5=0
z=17877
z=17877
x_1=37, x_2=100, x_3=5, x_4=4, x_5=0
Optimal is found
Optimal is found

Figure 5.3: Branch and Bound search

4. Since the solution set includes continuous variables, the B&B procedure continues, and the PHA is applied in each node to solve the new subproblem (Eq. 5.3-5.8). The algorithm continues until an integer solution is obtained. The results for each node and search tree are demonstrated in Fig 5.3.

5.4 Numerical Results

The CG, PHA, and B&B are coded and integrated with GAMS 34.2.0, and CPLEX is used as a solver for the RMIP, MIP, and LP. Antigone is used as a solver for the quadratic model to obtain the optimal solution. Moreover, GAMS stochastic library is used for the scenario generation. The overall algorithm is executed on a computer with Intel Core i5-3230M, 2.60 GHz CPU, 4 GB RAM. Experiments have been carried out for the problem presented in Section 5.3 for a number of scenarios as shown in Table 5.1. In each experiment, scenarios are generated from the discrete probability distribution $D \sim Pois(150)$ by using the Monte Carlo sampling method. To illustrate the computational complexity for the deterministic equivalent of the model for each experiment, we can calculate the number of variables and constraint as $[(m*J)+J+(2*S)]$ and $[(m*J)+(3*S)+(2*J)+i]$, respectively. To analyze the performance of the solution methodology, we present the number of scenarios, the run-time (sec), the number of decision variables, the number of constraints, and the expected total cost for each experiment; moreover, to analyze the quality of the solution methodology, we apply SP to the problem by using CPLEX solver and present the results in Table 5.1. Also, the expected value solution for the problem is

Table 5.1: Experiments and results for different number of scenarios

Number of Scenarios	Variables	Constraints	CPU time (sec)	Expected total cost (PHA)	CPLEX Solver (SP)
5	31	45	2152	17877	17877
10	41	60	3601	19987	19987
20	61	90	13119	19434	19434
50	121	180	38659	21188	21188
100	221	330	46085	19996	19996
200	421	630	57659	20250	20250

found as 15065, and the difference between stochastic objective function value and expected value is 17877-15065=2812 (VSS). Moreover, we can see that while the objective function value converges to the optimal solution by increasing the scenario number, CPU time increases linearly. The quality of the solution seems good for this example.

6. A TWO-PHASE PATTERN GENERATION AND PRODUCTION PLANNING PROCEDURE FOR THE STOCHASTIC SKIVING STOCK PROBLEM

6.1 Two-Stage SP formulation of the Multi-product SSP

We primarily provide the nomenclature for the mathematical model in Table 6.1 on the next page. This nomenclature includes the notation for the stochastic SSP model only, and it should be kept in mind that different phases in the solution methodology will also have their own notations in Section 6.2.

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 $k = 1, \dots, K$. 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 $s \in \mathbb{S} := 1, \dots, S$ 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 $D(s) = (d_{s1}, \dots, d_{sK})$ 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 as 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 amount of raw material needed,

Table 6.1: Nomenclature for Multi-product Stochastic SSP

Symbol	Abbreviation
	Indices
i	index of (small) item types , $i \in \mathbb{I} := \{1, ..., m\}$;
j	index of minimal patterns, $j \in \mathbb{J}^* := \{1, ..., J\}$;
k	index of product (large item) types, $k \in \mathbb{K} := \{1, ..., K\}$;
s	index of scenarios, $s \in \mathbb{S} := \{1, ..., S\}$;
	Parameters
a_{ij}	the number of item type i in pattern j;
δ_{jk}	a binary parameter that indicates if pattern j can be used for product k or not;
L_k	threshold (lower bound) width of product type k;
l_i	width of item type i;
d_{sk}	the demand of product type k in scenario s;
υ_s	the approved product rate (1-waste rate) in scenario s;
C^{Pr}	the fixed production cost for product rolls with fixed length (i.e 3000 meters);
C_i^R	the cost of the item type i;
C^O	the set up cost for pattern change (set up costs are assumed to be the same);
C_k^H	the overproduction cost for product type k;
C_k^B	the underproduction cost for product type k;
P_s	the probability of scenario s;
M_k	a large number for product type k such that $M_k >> max\{d_{sk}, \forall s \in \mathbb{S}\}$;
b_i	the available amount of item type i;
	Decision variables
y_j	a binary variable that indicates whether a pattern j is used or not;
x_{jk}	the frequency of pattern j for product type k;
r_i	amount of raw material i used;
q_{sk}^+	the overproduction amount for product type k in scenario s;
q_{sk}^-	the underproduction amount for product type k in scenario s;

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 $y_j, \forall j$. Matrix **a** having values of a_{ij} and matrix Δ having δ_{jk} values are obtained by using the Dragonfly Algorithm (DA) before solving the SP. These values are fed into the first stage of the SP as parameters. The overproduction and the underproduction amounts (q_{sk}^+

, q^{-}_{sk}) are the second-stage decision variables, and they depend on the scenario and the first-stage decisions. Then, we can construct the first-stage objective function as:

$$\min \quad \sum_{i\in\mathbb{I}} C^R_i r_i + \sum_{k\in\mathbb{K}} \sum_{j\in\mathbb{J}^*} (C^O y_j + C^{Pr} x_{jk}), \tag{6.1}$$

and, $s \in \mathbb{S}$, Q^s the $Q(\cdot)$ function provided in Eq. 4.4 for scenario s and is defined as:

$$Q^s(\,x,\xi) \; = \min \quad \sum_{k\in\mathbb{K}} (C^H_k q^{+}_{sk} + C^B_k q^{-}_{sk}) \tag{6.2}$$

Finally, by using Eq.s (6.1) and (6.2), and the nomenclature, the deterministic equivalent of the stochastic SSP model with the instance of $E := (m, K, l, L, b)$ is given through Eq.s(6.3)-(6.12).

$$\min \; z_{SP} = \quad \sum_{i\in\mathbb{I}} C^R_i r_i + \sum_{k\in\mathbb{K}} \sum_{j\in\mathbb{J}^*} (C^O y_j + C^{Pr} x_{jk}) + \sum_{s\in\mathbb{S}} P_s \sum_{k\in\mathbb{K}} (C^H_k q^{+}_{sk} + C^B_k q^{-}_{sk}) \tag{6.3}$$

$$\text{s.t.} \quad \sum_{k\in\mathbb{K}} \sum_{j\in\mathbb{J}^*} a_{ij} x_{jk} = r_i \leq b_i, \quad \forall i \in \mathbb{I} \tag{6.4}$$

$$q^{+}_{sk} - q^{-}_{sk} = \upsilon_s \sum_{j\in\mathbb{J}^*} x_{jk} - d_{sk}, \quad \forall s \in \mathbb{S}, k \in \mathbb{K} \tag{6.5}$$

$$x_{jk} \leq M_k y_j \delta_{jk}, \quad \forall j \in \mathbb{J}^*, k \in \mathbb{K} \tag{6.6}$$

$$x_{jk} \in \mathbb{Z}_+, \quad \forall j \in \mathbb{J}^*, k \in \mathbb{K} \tag{6.7}$$

$$a_{ij} \in \mathbb{Z}_+, \quad \forall i \in \mathbb{I}, j \in \mathbb{J}^* \tag{6.8}$$

$$r_i \in \mathbb{Z}_+, \quad \forall i \in \mathbb{I} \tag{6.9}$$

$$q^{+}_{sk}, q^{-}_{sk} \geq 0, \quad \forall s \in \mathbb{S}, k \in \mathbb{K} \tag{6.10}$$

$$y_j \in \{0,1\}, \quad \forall j \in \mathbb{J}^* \tag{6.11}$$

$$\delta_{jk} \in \{0,1\}, \quad \forall j \in \mathbb{J}^*, k \in \mathbb{K}, \tag{6.12}$$

The objective function given in Eq. 6.3 (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 that must be used to form products involves the cost 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 a variable width and a fixed length for each item, resulting in the 1D-SSP. Therefore, the production time and the production cost for the skiving process to form every product are assumed to be fixed.

The second-stage costs include the costs for the overproduction and the underproduction, that is, lost sales or overtime production costs, as well as the costs for procuring products from other sellers to meet the demand. Constraint (6.4) ensures that the available items are sufficient for the produced amount by multiplying the number of pattern replications and the number of items required in each pattern. Constraint (6.5) balances the overproduction or underproduction amount at the second stage as a result of the first-stage production decisions and the scenarios. The set-up constraint given in Eq. (6.6) states if a pattern is used, then the set-up cost incurs related to this pattern. M_k is an upperbound 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 must be used for M_k.

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 at once. In this way, we do not need to change patterns, and we pay the set-up cost for once. For this purpose, we construct a pattern pool that unites all patterns used in all products. An auxiliary and dependent binary variable δ which is generated by using DA and used as a parameter in the model, controls the assignment of patterns to products. Constraints (6.7), (6.8), (6.9), (6.10) are integrality and non-negativity constraints, and constraint (6.11) imposes that y_j is the binary variable. We determine the values of the decision variables a_{ij} in constraint (6.4) and δ_{jk} in constraint (6.6) 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.

6.1.1 Deterministic Counterpart of the First-Stage Model

Throughout the procedure, we need to check the propriety of $P_P(E)$. In other words, whenever we make a production decision, we need to check if we can produce this amount with the patterns on hand. If not, we can either opt for underproduction or search new patterns in order to not lose sales.

For this checkpoint, we implement the deterministic counterpart of the SSP with a small modification. Without loss of generality, the deterministic counterpart of the SMIP model given in Eq.s 6.3-6.12 can be rewritten using the first-stage variables and costs. This model minimizes the first-stage cost at a given production amount as follows:

$$min\ z_{det} = \sum_{i \in \mathbb{I}} C_i^R r_i^+ \sum_{k \in \mathbb{K}} \sum_{j \in \mathbb{J}^*} (C^O y_j + C^{Pr} x_{jk} + \Psi_k q_k^-) \tag{6.13}$$

$$s.t \qquad \sum_{j \in \mathbb{J}^*} \sum_{k \in \mathbb{K}} a_{ij} x_{jk} = r_i \leq b_i, \forall i \in \mathbb{I} \tag{6.14}$$

$$\sum_{j \in \mathbb{J}^*} x_{jk} + q_k^- \geq upperBound_k, \forall k \in \mathbb{K} \tag{6.15}$$

$$x_{jk} \leq M_k \delta_{jk} y_j, \forall j \in \mathbb{J}^*, k \in \mathbb{K} \tag{6.16}$$

$$r_i, \forall i \in \mathbb{I} \tag{6.17}$$

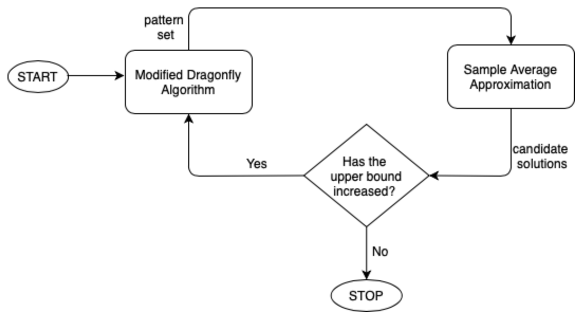
$$y_j \in \{0, 1\}, \quad \forall j \in \mathbb{J}^* \tag{6.18}$$

$$x_{jk}, q_k^- \in \mathbb{Z}_+, \quad \forall j \in \mathbb{J}^*, k \in \mathbb{K} \tag{6.19}$$

In this deterministic model (Eq.s 6.13-6.19), the indices, the first-stage variables, and the first-stage costs are almost the same with the original stochastic problem given throughout Eq.s 6.3-6.12. The main difference with the stochastic counterpart is the introduction of the production upperbound for each product denoted by $upperBound_k$ in Eq. (6.15). This upperbound 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 that represents a large number such as $\Psi_k = C_k^B * M$, $\forall k \in \mathbb{K}$, 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 upperbound. 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 (upperbound) is satisfied.

6.2 The Proposed Solution Methodology for the Stochastic Multi-product SSP

In this section, we develop an iterative two-phase solution methodology for the stochastic SSP. An overview of the solution methodology is presented in Fig. 6.1. In the first phase of the algorithm, we implement the Dragonfly Algorithm (DA)


pattern set
START
Modified Dragonfly Algorithm
Sample Average Approximation
candidate solutions
Has the upper bound increased?
Yes
No
STOP
Figure 6.1: Recursive two-phase algorithm for the stochastic SSP

proposed in [Mirjalili, 2016] to produce the minimal pattern set $P^*(E)$. This minimal pattern set $P^*(E)$ is then fed into the second phase. In the second phase, the SMIP presented in Section 6.1 is solved by the SAA method [Shapiro et al., 2014]. This process provides candidate solutions and an upperbound for the production amount.

We first present each module separately, and later define the two-phase methodology that integrates these algorithms. The inclusion of the DA results in the proposed methodology to produce a heuristic solution.

6.2.1 Dragonfly Algorithm

Dragonfly Algorithm (DA) is a Swarm Intelligence-based method that implements non-linear Lévy flights in the search space. It is motivated by improving the mechanism of the Particle Swarm Optimization (PSO) and Cuckoo Search (CS) algorithms. The PSO has a linear search direction that does not provide a thorough search structure and does not conduct enough exploration and CS has an exponential search pattern that can jump over optimal points. DA attempts to improve these algorithms by introducing various concepts. While the dragonflies are moving, they obey three rules. First rule is *alignment*, which states that they fly together and their positions are close. The second rule, *separation*, states that while they are close, they do not collide. The last rule, *cohesion* necessitates that their velocities should be also close to be able to fly together. Similarly, the dragonflies want to be close to a *food* source (the best solution found) and away from an *enemy* or a predator, that is, the worst solution found. These components are adjusted linearly. If there is no dragonflies nearby, they approach each other with a nonlinear Lévy flight, which promotes exploration.

The applications of the DA in the literature involve the Traveling Salesman Problem (TSP) [Hammouri et al., 2018], optimal dynamic scheduling of tasks [Shirani and Safi-Esfahani, 2020], optimization of path planning for mobile robots, moreover, solving 0-1 knapsack problems [Abdel-Basset et al., 2017], feature selection problems [Mafarja et al., 2017], graph coloring problems [Baiche et al., 2019], as well as optimization of wind-solar–hydro power scheduling by using multi-objective version [Li et al., 2019]. Its performance on discrete problems is found to be superior to PSO and GA [Mirjalili, 2016]. This study implements the DA for pattern generation for each product separately using the following steps.

Pre-processing of patterns: This process uses width of small items.

Step 1: Unavailable or out-of-stock items are removed from the item set $\mathbb{I}$.

Step 2: The amount of each item i required to construct a product k, (n_{ik}) is calculated such that

$$n_{ik} = \left\lceil \frac{L_k}{l_i} \right\rceil. \tag{6.20}$$

Step 3: The item set is augmented by repeating each item $(n_{ik} - 1)$ times to obtain the extended and ordered item set $\mathbb{I}^+$, so that there are n_{ik} of each small item. Figure 6.2 shows the structure of the extended item set.

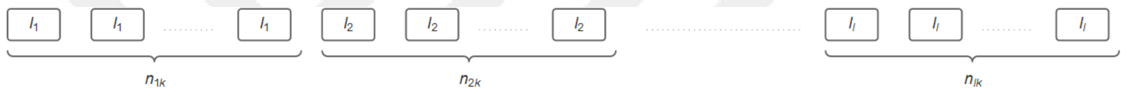


Figure 6.2: Extended and ordered item set

Let $I_+ = |\mathbb{I}^+|$, that is, the cardinality of the extended item set. Furthermore, $cl_h = \sum_{i=1}^{h} l_i$. In other words, cl_h indicates the cumulative length of the items from the first to the h^{th} location.

After these steps, the extended set of items, $\mathbb{I}^+$, is fed into the following steps of the main Dragonfly Algorithm:

Initialize: Algorithm parameters are initiated. The position change (Δpos) matrix is initialized as zero matrix with dimensions NoD and I_+, i.e. $\mathbf{0}_{NoD,I_+}$ where NoD represents the number of dragonflies. The number of maximum iterations $(MaxIt)$ is determined.

Step 1: Each dragonfly has a dimension of I_+, and each value of the dragonfly is a randomly generated number uniformly distributed between 0 and 1. We will specify this matrix as $\mathbf{pos}_{NoD,I_+}$. Each row of $\mathbf{pos}_{NoD,I_+}$ indicates a solution or a dragonfly.

Step 2: The objective function value, the trim loss, of each dragonfly is calculated.

Let b indicate the index of the b^{th} dragonfly. The objective function of the b^{th} dragonfly is calculated as follows:

1. The b^{th} row of the $\mathbf{pos}_{NoD,I_+}$ matrix is sorted in a decreasing order.
2. The index vector for the sorted values is obtained. We will denote that order by $[\vec{h}]$.
3. The $cl_{[h]}$ values are added until L_k is reached such that $cl_{[h-1]} < L_k$ and $cl_{[h]} \geq L_k$. In other words, the items are skived until the desired length for product k is reached.
4. The trim loss is calculated such that $\sum_{h=1}^{h'} cl_{[h]} - L_k$. The excess width is calculated as the trim loss.

Otherwise stated, we sort the indices of the values of the dragonfly in a descending order. Next, we start skiving items in that order until the product length is reached. As an illustrative example, let the initial item set be $\{6, 5, 3\}$, the extended and ordered item set $\mathbb{I}^+ = \{6, 6, 5, 5, 3, 3, 3\}$, and width of the product $L_k = 9$. Assume that a dragonfly has the value vector $[0.35, 0.45, 0.21, 0.76, 0.87, 0.98, 0.07]$. Then the ordered index vector is $[\vec{h}] = [6, 5, 4, 2, 1, 3, 7]$, meaning that the sixth dimension of the dragonfly is the largest, the fifth dimension of the dragonfly is the second largest, and so on. Hence, $cl_{[1]} = l_6 = 3$, i.e. meaning that if the item in the first order is used, the length of the final product would be 3 units, which is smaller than L_k; therefore, the skiving process continues with the item in the second order. When the next item is skived, we obtain $cl_{[2]} = l_{[1]} + l_{[2]} = l_6 + l_5 = 3 + 3 = 6$. The obtained length is still less than the desired product length, and the skiving is continued with $cl_{[3]} = l_{[1]} + l_{[2]} + l_{[3]} = l_6 + l_5 + l_4 = 3 + 3 + 5 = 11$. These three skived items satisfy the desired product length of L_k. Therefore, we stop the skiving process. As a result, a dragonfly, comprised of the vector $[0.35, 0.45, 0.21, 0.76, 0.87, 0.98, 0.07]$, indicates a product composed of two of the item with length 3, and an item of length 5, resulting in a product of length 11. The trim loss of this product is $cl_{[3]} - L_k = 11 - 9 = 2$. An important reminder is to keep the positions of the dragonflies between 0 and 1 throughout the algorithm so that extreme position values are prevented.

After decoding each dragonfly in the swarm, the trim loss of each dragonfly is calculated, and the objective function value vector, $\vec{f}_{NoD}$ is obtained. Each value of the vector indicates the objective function of a dragonfly.

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

$$food = \{\mathbf{pos}_{b,*} : f_d = \min_{b'} \vec{f}, b' = 1, \ldots, NoD\}, \tag{6.21}$$

and

$$enemy = \{\mathbf{pos}_{b,*} : f_d = \max_{b'} \vec{f}, b' = 1, \ldots, NoD\}. \tag{6.22}$$

Step 4: Assume that the current iteration is t. The neighborhood range is updated as $t/MaxIt$. For each dragonfly, if the neighborhood does not have any dragonflies, a Lévy flight is employed as given below [Mirjalili, 2016]:

$$\Delta \text{pos}_{b,*}(t) = \mathbf{pos}_{b,*} \cdot 0.01 \frac{r_1 A}{r_2^{1/B}}, \tag{6.23}$$

where r_1 and r_2 are random values between 0 and 1, and where B is a user-defined parameter, and A is calculated as:

$$A = \left[\frac{\Gamma(a+B)\sin(\pi B/2)}{\Gamma[(1+B)/2]B2^{(B-1)/2}}\right]^{1/B}. \tag{6.24}$$

If there is at least one dragonfly in the neighborhood, then weighted compound vector of separation, alignment, cohesion, food source approach, and enemy escape is calculated, and the position change of the dragonfly is as follows:

$$\Delta \text{pos}_{b,*}(t) = \alpha Sep_b + \beta Aln_b + \gamma Coh_b + \eta(food - \mathbf{pos}_{b,*}) + \epsilon(enemy + \mathbf{pos}_{b,*}) + \omega \Delta \text{pos}_{b,*}(t-1) \tag{6.25}$$

where $\alpha, \beta, \gamma, \eta, \epsilon$ are predetermined and tuned coefficients, ω is the inertia rate. $food$ and $enemy$ are the positions of the dragonfly with the best and the worst objective function values, respectively. Assume 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 component (Sep_b) avoids collision of the dragonflies, the alignment (Aln_b) and the cohesion (Coh_b) components enable dragonflies to exploit the search space with similar velocities and positions (Figure 6.3). The calculations of these components follow the formulae below [Mirjalili, 2016]:

$$Sep_b = \sum_{b' \in \Lambda_b} \mathbf{pos}_{b,*} - \mathbf{pos}_{b',*}, \tag{6.26}$$

$$Aln_b = \frac{\sum_{b' \in \Lambda_b} \Delta \mathrm{pos}_{b',*}(t-1)}{\lambda_b}, \tag{6.27}$$

$$Coh_b = \frac{\sum_{b' \in \Lambda_b} \mathbf{pos}_{b',*}}{\lambda_b} - \mathbf{pos}_{b,*}. \tag{6.28}$$

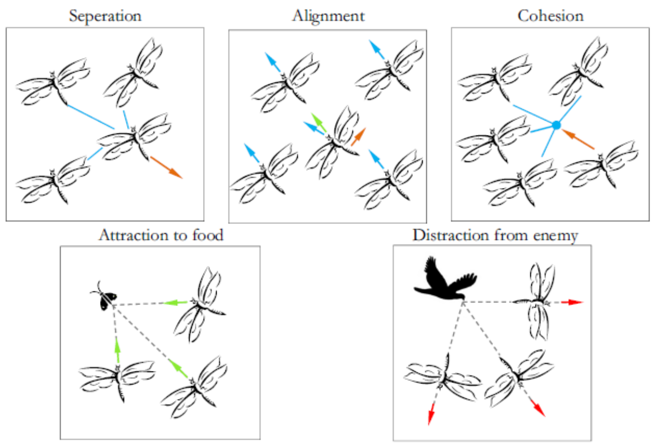


Figure 6.3: Behavior of Swarms [Mirjalili, 2016]

Step 5: The position change is updated using the velocity and the inertia on the position change such that

$$\mathbf{pos}_{b,*}(t) = \mathbf{pos}_{b,*}(t-1) + \Delta\mathbf{pos}_{b,*} \tag{6.29}$$

We keep the dragonfly positions between 0 and 1; therefore, if a dragonfly position value in any dimension exceeds these values, the position is re-adjusted in a way to be equal to the nearest boundary.

Step 6: Steps 2-5 are repeated until the maximum number of iterations is reached.

The outcome of the algorithm is the dragonfly with the minimum trim loss, but it will yield an extended and ordered version of the items. Any dragonflies with the same objective function value represent a pattern.

Post-processing: 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 width 3 and one item with width 5. Given the initial item set 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 Eq.s 6.3-6.12. The produced minimal pattern pool is not necessarily proper. The propriety of the pattern pool is determined as a result of the second phase, that is, the Sample Average Approximation (SAA) algorithm.

6.2.2 Sample Average Approximation (SAA)

The SAA is implemented for large-size stochastic problems which 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 $(n = 1, 2, \dots, N)$ of the random vector ξ. These realizations are denoted by $\xi_1, \dots, \xi_N$. 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 (6.3) - (6.12) is approximated by the SAA problem [Kazemi Zanjani et al., 2013], [Shapiro and Homem-de Mello, 1998], [Aydin, 2012], where $Q(x, \xi)$ is the objective function involving decision variables x and random variables ξ. According to Shapiro [Shapiro, 2005], the SAA provides good convergence and robust statistical inferences including an analysis of error, stopping rules validation, and it is easy to implement with a commercial software. The steps of the SAA are given as follows [Aydin, 2012]:

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 sufficiently large reference sample where $N' >> N$.

Step 1: Solve the problem (6.30) and optimal objective function value v^g and candidate solution x^g for each g.

$$\min_x \quad C^T x + \frac{1}{|N_g|} \sum_{n=1}^{N} Q(\, x, \xi_n) \tag{6.30}$$

Step 2: Compute $\bar{v}^G$ average (6.31) which is the unbiased estimator of the objective function of the original problem v^* and variance $\hat{\sigma}^2_{\bar{v}^G}$ (6.32) of the objective function values obtained in the first step.

$$\bar{v}^G := \frac{1}{G} \sum_{g=1}^{G} v_g \tag{6.31}$$

$$\hat{\sigma}^2_{\bar{v}^G} := \frac{1}{G(G-1)} \sum_{g=1}^{G} (v_g - \bar{v}^G)^2 \tag{6.32}$$

Step 3: Solve the problem g times with sample size N' (6.33) by using each candidate solution $\bar{x}^g$ of each g in order to find $\hat{v}^g$ for each g. And calculate $\hat{\sigma}^2_{\hat{v}^g}$ (6.34).

$$\hat{v}^g := \min_x \quad C^T \bar{x}^g + \frac{1}{|N'|} \sum_{n=1}^{N'} Q(\, \bar{x}^g, \xi_n) \tag{6.33}$$

$$\hat{\sigma}^2_{\hat{v}^g} := \frac{1}{N'(N'-1)} \sum_{n=1}^{N'} (\, C^T \bar{x}^g + Q(\, \bar{x}^g, \xi_n) \, - \hat{v}^g) \tag{6.34}$$

Step 4: Compute the estimation of optimality gap for the candidate solution $gap_g(\bar{x}^g)$ (6.35) and the variance of the optimality gap $\hat{\sigma}^2_{gap_g}$ (6.36) in order to analyze the quality of the candidate solution.

$$gap_g(\bar{x}^g) = \hat{v}^g - \bar{v}^G \tag{6.35}$$

$$\hat{\sigma}^2_{gap_g} = \hat{\sigma}^2_{\hat{v}^g} + \hat{\sigma}^2_{v^G} \tag{6.36}$$

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

According to Ahmed and Shapiro [Ahmed and Shapiro, 2002], the lower bound for v^* is provided by $\bar{v}^G$, and the upperbound for v^* is obtained by $\hat{v}^g$. The optimal objective function value of the SAA problem converges to the optimal objective function value of the original problem given in Eq. (4.1) with probability 1.00 as sample size N goes to infinity ($N \rightarrow \infty$) [Shapiro, 2003]. A larger sample size ensures a stricter approximation. However, it also increases the computational complexity [Shapiro, 2005]. Therefore, instead of using a large sample size, using small independent and identically distributed (i.i.d) samples is more efficient. The SAA algorithm is based on this principle. The complexity increases exponentially as the sample size increases, especially for the integer problems [Kleywegt et al., 2002]. This increase in complexity eventually causes a trade-off between quality of the approximation and computational complexity.

6.2.3 The Two-Phase Solution Methodology

As shown in Fig. 6.1, 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 upperbounds of production amounts, as shown in Fig. 6.4. This upperbound also serves as a reference for the highest production amount. If the patterns found cannot satisfy this upperbound due to the availability constraints, then, the pattern set cannot be deemed proper. To obtain a proper minimal pattern set, the DA is rerun to produce new patterns. 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 upperbound of production.

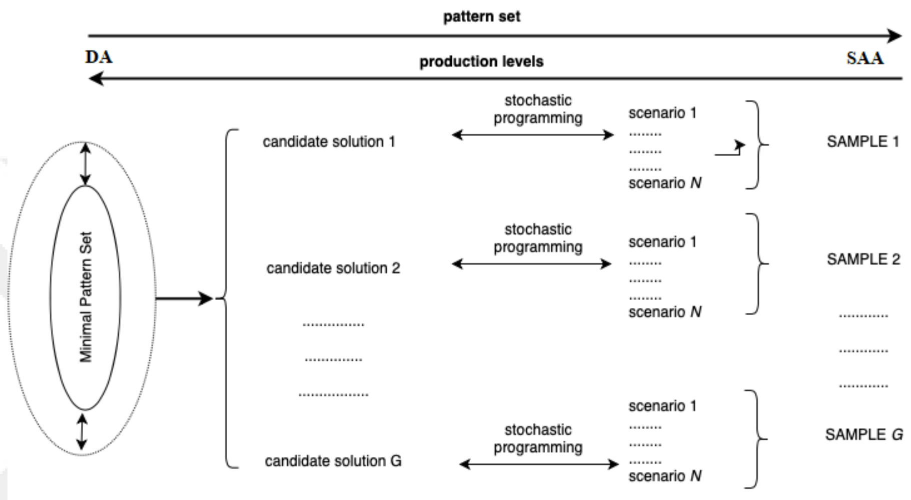


Figure 6.4: The recursion between the DA and the SAA

The pseudocode of the methodology is given in Algorithm 5. Below, we define the parameters and variables used in the pseudocode:

z_{SP}: The objective function value of the stochastic model given through Eq.s 6.3-6.12.;

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

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

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

$stepsize_k$: the expansion amount in the search space for product $k \in \mathbb{K}$;

$upperBound_k$: the extended maximum production amount for product $k \in \mathbb{K}$ calculated by summing $stepsize_k$ and $maxProdLevels_k$;

z_{det}: the objective function of deterministic counterpart model Eq.s 6.13-6.19;
$output_k$: the total production for product $k \in \mathbb{K}$ obtained through the optimal solution $\sum_{j\in\mathbb{J}} x^*_{jk}$ of z_{det} Eq.s 6.13-6.19. And finally, **t** is the iteration number.

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

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, $P^*(E) = \bigcup_{k\in\mathbb{K}} P^*_k(E)$. Additionally, we generate a binary pattern-product matrix, Δ, to tabulate the matches between patterns and products. $\delta_{jk} = 1$ 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.
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 result that is optimal for this particular set. Therefore, the initial pattern set does not provide direct answers to the following questions:
   (a) Does the pattern set involve the pattern combinations that produce the products at a global optimal cost?
   (b) How does the global optimal solution compare to the optimal solution of this pattern set?

```
1  START ;
2  Determine parameters;
3  Set $b'_i = b_i, b_i = \infty, \forall i$, $iteration(t) = 0$ ;
4  Run the DA to generate the pattern set $P^*(E)$ that has a minimum trim loss ;
5  Run the SAA submodule;
6  while $maxProdLevels_k(t) > maxProdLevels_k(t-1)$ do
7      while $\forall k, upperBound_k > output_k$ do
8          Run the DA to generate the additional pattern set $P'(E)$ that has a
             minimum trim loss;
9          $P^*(E) := P^*(E) \bigcup P'(E)$;
10         solve deterministic min SSP using $P^*(E)$; $x_{jk} := \arg\min(z_{det})$ Eq.s 6.13-6.19 ;
11         Compute $\forall k, output_k = \sum_{j \in \mathbb{J}} x_{jk}$
12     end
13     Set minimal proper pattern pool $P^*_p(E) := P^*(E)$ ;
14     $b_i = b_i', \forall i$ , $t = t + 1$;
15     Run the SAA submodule (Section 6.2.2);
16 end
17 STOP;
```

```
18   SAA Submodule
1      Run the SAA (Section 6.2.2), $x^g_{jk} := \underset{g=1,\dots,G}{\arg\min}(z_{SP})$, Eq. 6.30, using $P^*(E)$ or if
         available, $P^*_p(E)$ ;
2      Obtain candidate production amounts for each sample;$\forall g$,
         $prodLevel^g_k(t) = \sum_{j \in \mathbb{J}} x^g_{jk}$;
3      Select maximum production amount for each product; $\forall k$
         $maxProdLevels_k(t) = \max(prodLevel^g_k(t))$;
4      Compute $upperBound(t)$ vector using step size; $\forall k$,
         $upperBound_k(t) = maxProdLevels_k(t) + stepsize_k$;
5      Solve deterministic min SSP using $P^*(E)$; $x_{jk} := \arg\min(z_{det})$ ;
6      Compute $\forall k, output_k = \sum_{j \in \mathbb{J}} x_{jk}$ ;
7    end
```

Algorithm 5: The two-phase solution methodology

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

Therefore, the quality and the sufficiency of this initial pattern set are unknown. For this purpose, we first calculate a sufficiently high reference production amount for all products. As the iterations go on, 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 non-proper 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 not possible to comment on $P^*(E)$ being proper without any target production amount.

To this end, the first recursion of the SAA (Eq.s 6.3-6.12) solves a relaxed model with no availability constraints (by ignoring the constraint given in Eq. 6.4) to obtain initial reference production amounts for each sample as presented in Fig. 6.4. The results reflect the optimal production amounts, given the pattern set on hand. The main purpose of ignoring the item availability constraint given in Eq. (6.4) is two-folds: *i*) to observe the optimal production amounts if there were an unlimited supply of items, and *ii*) later to check if these amounts can be produced with the available items. We generate G number of samples, each with a sample size N to apply the SAA approach that solves the SMIP presented in Section 6.1. 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 $prodLevel_k^g$. $P^*(E)$ 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 $maxProdLevels_k = max_{\{g:g=1...,G\}} prodLevels_k^g$. Fig. 6.6a shows such production levels for two products. Fig. 6.6a displays the maximum production levels for two products.

3. *Calculation of the upperbounds 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 on hand.

   This trigger entails increasing the maximum production amounts by a step size parameter. The step size works as a preventative measure, especially when the availabilities are sufficient to produce the upperbounds. Producing the upperbounds indicates that more patterns could extend the search space and find a better solution with less cost than the current one. (Fig. 6.6b) for

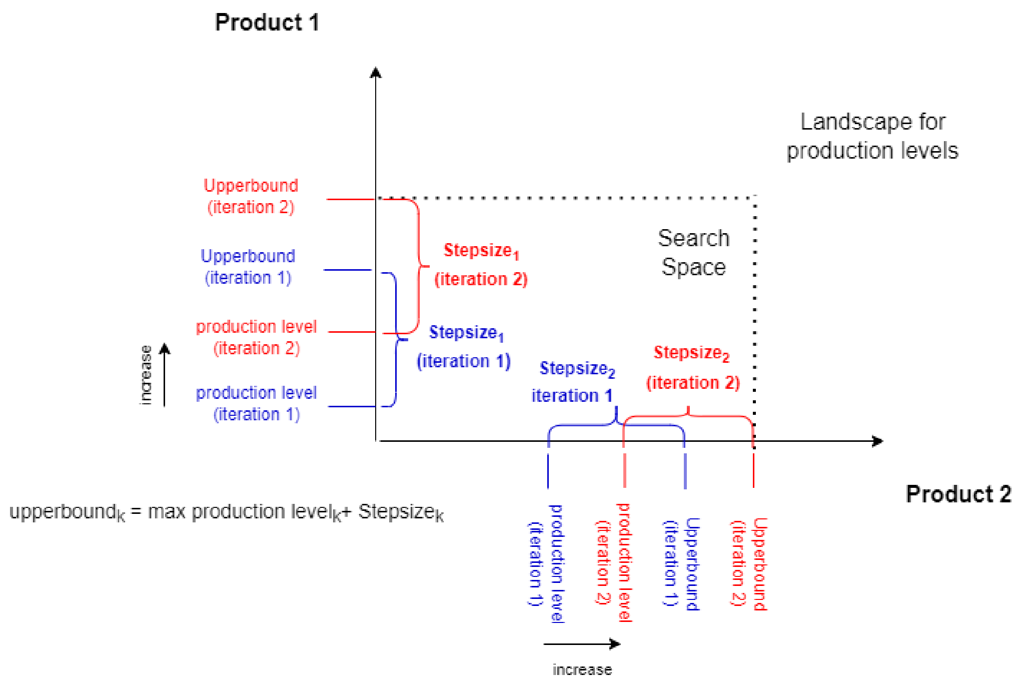
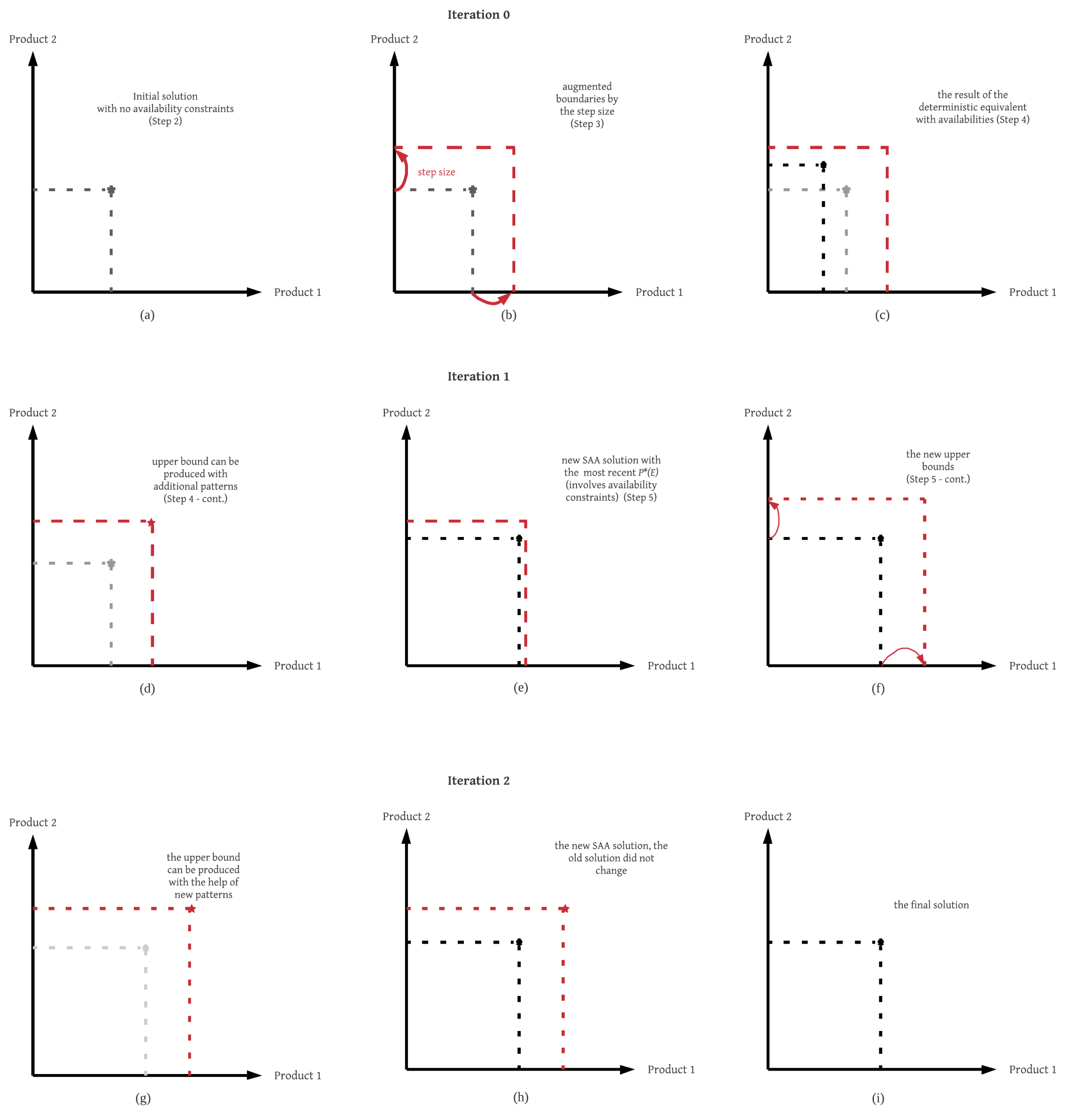

Product 1
Landscape for production levels
Upperbound (iteration 2)
Upperbound (iteration 1)
production level (iteration 2)
production level (iteration 1)
increase
Stepsize$_1$ (iteration 2)
Stepsize$_1$ (iteration 1)
Search Space
Stepsize$_2$ iteration 1
Stepsize$_2$ (iteration 2)
Product 2
upperbound$_k$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

Figure 6.5: The expansion of the search space at each iteration

a snapshot of the production amount and the upperbound, and Fig. 6.5 for changes between two consecutive iterations.)



Iteration 0
Product 2
Product 1
Initial solution with no availability constraints (Step 2)
(a)
augmented boundaries by the 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 constraints) (Step 5)
(e)
the new upper bounds (Step 5 - cont.)
(f)
Iteration 2
the upper bound 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 6.6: Step-by-step search space extension to trigger the generation of new patterns.

4. *Checking for propriety of the pattern set (The inner while loop in Alg. 5)*
   In the next step, we include the item availability constraint in Eq. (6.4) to check whether the pattern set on hand can produce this upperbound. In other words, the propriety of $P^*(E)$ is checked for each upperbound by using the deterministic model Eq.s 6.13-6.19 presented in section 6.1.1. The deterministic model, itself, is not used to solve the original stochastic problem, but it aids in deciding whether producing the upperbound is feasible.

There are two possible outcomes; the items can or cannot produce the upperbound. If the available items cannot produce the upperbound, the DA is rerun to produce more patterns. In Fig. 6.5c, the item availabilities cannot produce the upperbounds, 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 $P^*(E) := P^*(E) \cup P'(E)$. 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:

- The DA has produced numerous patterns, but the production of upperbound is infeasible. The total width of the remaining items is less than the product width, so the pattern set is already exhaustive.
- The most recent $P^*(E)$ can produce the upperbounds, and the minimal pattern set becomes proper. In this case, we move on to the next step.

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

In Fig. 6.6c, the items cannot produce the upperbounds or the optimal amount, per se. Therefore, the initial pattern set is not proper. We rerun the DA until the upperbounds can be produced. Fig. 6.6d shows such a case.

5. *Solving the two-stage stochastic programming with recourse (The outer while loop in Alg. 5)*

   The SAA is run with the most recent pattern set on the original problem with availability constraints given through Eq.s 6.3-6.12 as mentioned in Section 6.2.3. This recursion analyzes whether the newly-added patterns change the previous optimal solution and the upperbound. If the upperbounds are different from the previous iterations, the new minimal pattern set enables different pattern-product configurations than previous iterations. Fig. 6.6e shows the new solution with the most recent pattern set and the item availabilities. Based on this solution, Fig. 6.6f finds the new upperbounds. The upperbounds are different from the ones found in Fig. 6.6b; hence, the recursion starts.

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 upperbounds remain the same 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 is obtained.

   In Fig. 6.6g, we check if the new upperbounds can be produced with the pattern set from the previous iteration. If not, the DA produces new patterns as mentioned in Step 4. However, Fig. 6.6g shows the augmented set that can solve the upperbounds. When $P^*(E)$ is proper ($P^*_P(E)$), the SAA is rerun to find the optimal solution with the addition of new patterns as in Fig. 6.6h. The solution found in Fig. 6.6h is the same as in the previous iteration (given in Fig. 6.6e. The addition of new patterns has not changed the optimal solution. The production amounts and the upperbounds have converged. Therefore, we stop the algorithm and accept the final solution as in Fig. 6.6i.

With these recursions, the boundaries of the search space are dynamically updated through iterations by using step size (see Fig. 6.6). $P^*_p(E)$ is obtained for each production amount of each sample by checking if $P^*(E)$ can produce the upperbounds. Finally, we obtain a solution of the original stochastic problem [Shapiro and Homem-de Mello, 1998] as the SAA output. The recursive structure is also visualized in Fig. 6.4.

6.3 An Illustrative Example

In this section, we present a small instance of $E := (m, K, l, L, b)$ as a stochastic and multi-product version of the SSP. In this example, E:=(3, 2, (500, 400, 300), (1000, 1500), (45, 65, 85)). The demand random variables are $D_1 \sim Pois(10)$ and $D_2 \sim Pois(20)$, respectively. The standard deviations for the demand is, therefore, $\sigma_1 \approx 3.16$ 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 $\Upsilon \sim \mathcal{N}(0.95, 0.0001)$. $C^{Pr} = 100$ Euros per product, $C^O = 60$ Euros per pattern change, $C^R = [75 \quad 60 \quad 45]$ 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 illustrate the methodology based on this example:

Iteration t= 0:

1. The DA is run to generate the minimal pattern set $P^*(E)$. Matrices **a** and Δ are presented below. Each column of **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.

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

2. The SAA is applied to the relaxed problem (without the item availability constraint given in Eq. 6.4) with $G = 10$. Initial reference values for $prodLevels_k^g(t)$ and $maxProdLevels_k^g(t)$ for all k in all g are as follows:

$$prodLevels(t) = \begin{bmatrix} 10 & 21 \\ 10 & 20 \\ 10 & 21 \\ 10 & 22 \\ 11 & 21 \\ 10 & 21 \\ 10 & 20 \\ 10 & 20 \\ 10 & 21 \\ 10 & 20 \end{bmatrix} \implies maxProdLevels(t) = \begin{bmatrix} 11 & 22 \end{bmatrix}.$$

3. Upperbounds for each product ($upperBound_k$) are computed such that $upperBound(t) = \begin{bmatrix} 11 + 2\sigma_1 & 22 + 3\sigma_2, \end{bmatrix}$ making $upperBound_1(t) = 17$ and $upperBound_2(t) = 33$.
4. The deterministic SSP model in Eq.s 6.13-6.19 is used to check whether the minimal pattern set on hand satisfies the upperbound Eq. 6.15. If not, the DA would have to generate an additional minimal pattern set until the upperbound 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 as follows.

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

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

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

$output = \begin{bmatrix} 17 & 33 \end{bmatrix} \geq upperBound = \begin{bmatrix} 17 & 33 \end{bmatrix}$, $P_p^*(E)$=$P^*(E)$ is obtained.

Iteration t=1:

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

5. The SAA is applied to the original problem (with the availability constraint Eq. 6.4). $ProdLevels(t)$, $maxProdLevels(t)$, and $upperBound(t)$ are computed.
$$prodLevels(t) = \begin{bmatrix} 10 & 21 \\ 10 & 20 \\ 10 & 21 \\ 10 & 22 \\ 11 & 21 \\ 10 & 21 \\ 10 & 20 \\ 10 & 20 \\ 10 & 21 \\ 10 & 20 \end{bmatrix} \Longrightarrow maxProdLevels(t) = \begin{bmatrix} 11 & 22 \end{bmatrix} \Longrightarrow$$
$upperBound(t) = \begin{bmatrix} 17 & 33 \end{bmatrix}$.
6. We continued the computations, and in **iteration t=2**, the results of the SAA has not changed, and $maxProdLevels_k(1) = maxProdLevels_k(2), \forall k$, then we stop the algorithm.
$maxProdLevels(1) = \begin{bmatrix} 11 & 22 \end{bmatrix} \geq maxProdLevels(2) = \begin{bmatrix} 11 & 22 \end{bmatrix}$.

Recall from Section 6.2.2, the SAA approach searches for the solution of N_g instances having N scenarios each given in the model presented in Eq.s (6.3)-(6.12). 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), and the reference sample size (N'), the candidate solutions, and the summary statistics for the objective function values are presented in Table 6.2. Monte Carlo sampling is used [Shapiro and Homem-de Mello, 1998] 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.

Table 6.2: SAA illustrative example with Monte Carlo Sampling N=100, N'=400, G=10

g	X_{11}	X_{32}	v^g	$\hat{v}^g$	$\hat{\sigma}^2_{\hat{v}^g}$	$gap_g(\bar{x}^g)$	$\hat{\sigma}^2_{gap_g}$
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
				$\bar{v}^G$=13729			
				$\hat{\sigma}^2_{v^G}$=7442			

CPLEX is used as the solver in GAMS. The algorithm is executed on a computer with Intel core i5-3230M, 2.60 GHz CPU, 4 GB RAM. The candidate solutions and statistical computations of objective function values are presented in Table 6.2. According to this table, the first, third, sixth, and ninth samples have the smallest expected total costs as shown in Table 6.2. Moreover, the convergence seems more robust in these samples than the others 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 closer convergence. Nonetheless, in response to increasing sample sizes, the increase in the computational complexity should not be overlooked.

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

6.4 Numerical Example and Results

6.4.1 Numerical Example and Discussions

In this section, a large-sized multi-product stochastic SSP and its results are presented. The instance SSP $E := (m, K, l, L, b)$ is $E := (50, 2, l, (1200, 1300), b)$ where $D_1 \sim Pois(150), D_2 \sim Pois(200)$ and $\sigma_1 \approx 12.24\sigma_2 \approx 14.44$ for each product. The random approved product rate is $\Upsilon \sim \mathcal{N}(0.95, 0.0001)$. $C^{Pr} = 100$ Euros per product, $C^O = 60$ Euros per pattern change, the cost of each small item is computed as C_i^R= ((l_i/1000)*3000 m)*(0.05 Euros / m^2), respectively for $i \in \mathbb{I}$. The width of small items and products are 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 $B = 0.05, \alpha = 0.5, \beta = 0.03, \gamma = 0.07, \eta = 0.05, \epsilon = 0.05$, ω=0.05. Finally, the step sizes are $0.6\sigma_1$ and $0.75\sigma_2$.

We run the algorithm three times with different Λ_b'=(20,50,70) and $MaxIt$=(10,20,30) values in order 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 6.3.

Table 6.3: The computational complexity and results for N=100, N'=400, G=30

$Trial$	$Variables$	$Constraints$	g	$\hat{v}^{best}$	$\hat{\sigma}^2_{\hat{v}^{best}}$	$gap_{best}(\bar{x}^g)$	$\hat{\sigma}^2_{gap_{best}}$	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

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 seventh sample with minimum $\hat{v}^g$ can be determined as the best solution in the second trial, i.e $x^{SAA} = \arg\min v^{SAA}$ where $v^{SAA} = \min_{g=1,\ldots,G} \hat{v}^g$. The obtained solution is presented at Table 6.4:

Table 6.4: Frequency of patterns in x^{SAA}

$Product(k)/Pattern(j)$	2	12	18	19	27	40	43	44
1	26	-	-	-	22	-	-	107
2	-	51	20	56	-	78	6	-

The used amount of items are presented as a vector r^{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 preferred $P^*_p(E)$ by using the minimization model Eq.s 6.13-6.19 which controls the properity of the $P^*(E)$. In our 50-item example, the computational time given in Table 6.3 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 also conduct a sensitivity analysis to analyze the impact of the step size through different multipliers for σ. Given that all other parameters and random variable values remain the same, the comparisons regarding various statistics are presented

Table 6.5: 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
$\hat{\sigma}^2_{\hat{v}^{SAA}}$	229468	229547	231179	230194	230470	229401
gap_{SAA}	175	191	168	176	175	191
$\hat{\sigma}^2_{gap_{SAA}}$	280227	285398	286006	284908	282488	283713
CPU time (sec)	258	102	76	70	74	55

in Table 6.5. 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 upperbound, 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 6.5 shows the average mean and variance of the objective function values and the optimality gaps, and the average CPU time (in terms of seconds) of 30 runs of the algorithm. For the sake of fair comparison, all scenarios are kept the same for every step size value. Table 6.5 visualizes the effect of the step size on the average objective function value and the CPU time. As can be seen from the table, when the step size is increased more than 10 times, the computational time increased logarithmically. However, the decrease in the objective function value as the step size increases might not be as significant compared to the increase in the computational time. The improvement in the objective function value remains less than 0.1%. Another performance measure is the service level, i.e., the average rate of met demands. Table 6.6 shows the average service levels for different step size multipliers. One way of determining the step size is through experimentation. We also recommend choosing several step size multipliers with respect to covering a threshold service level (e.g., 95%) while observing any significant change in the objective function value. In our example, assuming a 95% service level for each product leads to a selection of 1-1.25 (recommended) for the σ_k multipliers of each product given that the cost decrease might not be significant for higher multiplier values.

Step size multiplier for product 1 (σ_1)	**upperbound for product 1 (units)**	**Service level for product 1** (%)	**Step size multiplier for product 2** (σ_2)	**upperbound for product 2** (units)	**Service level for product 2** (%)
13	314	100	16	430	100
10	277	100	12	378	100
6.5	235	100	7.5	315	100
3.25	196	99.99	3, 75	262	99.99
1.25	172	96.46	1.5	231	98.55
0.6	165	89.58	0.75	221	93.39

Table 6.6: Service levels with respect to the step size

Even though this study is not a multi-objective analysis of the stochastic SSP, it also controls the trim loss in the meantime. 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 non-minimal patterns to be included in skiving even if the end product satisfies the given thresholds. A counter argument for this structure is the case of a very high set-up cost. If the set-up cost is considerably high, the model may choose to sacrifice the trim loss, and the longest pattern can be used to manufacture all products. In this way, the model avoids paying expensive set-up costs when patterns change. In such cases, $\delta_{jk} = 1$ for shorter products even though the pattern is not minimal. For example, if the set-up cost were 6000 instead of 60, then the patterns for the second product (with width 1300) could be used for the first product (with width 1200).

6.4.2 Performance Analysis for the DA

While the first phase of the algorithm uses a metaheuristic, the SAA in the second phase implements MIP under various scenarios. Hence, the second phase solves the problem to optimality under given parameters and scenarios. If the number of samples and scenarios is sufficient, by the law of large numbers, the result converges to the optimal solution. Nonetheless, the parameters fed into the second phase are the results of the DA which does not guarantee the optimal pattern set. Hence, in

Number of dragonflies: 20			
$\alpha+\beta+\gamma=1$			
α	β	γ	**Average objective function value**
0.1	0.1	0.8	121579
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			
$\alpha+\beta+\gamma=1$			
α	β	γ	**Average objective function value**
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			
$\alpha+\beta+\gamma=1$			
α	β	γ	**Average objective function value**
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
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

Table 6.7: Performance analysis for the parameters of the DA

this subsection, we analyze the effect of the DA parameters on the objective function value. The literature recommends that $\alpha+\beta+\gamma = 1$ [Mirjalili, 2016], hence the first 9 rows of Table 6.7 analyze this case, and the last 9 rows use $\alpha+\beta+\gamma < 1$ for slower convergence and to avoid 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 were carried about the run time of the algorithm by increasing the number of item types in Figure 6.7 and the number of product types in Figure 6.8. 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 6.9).

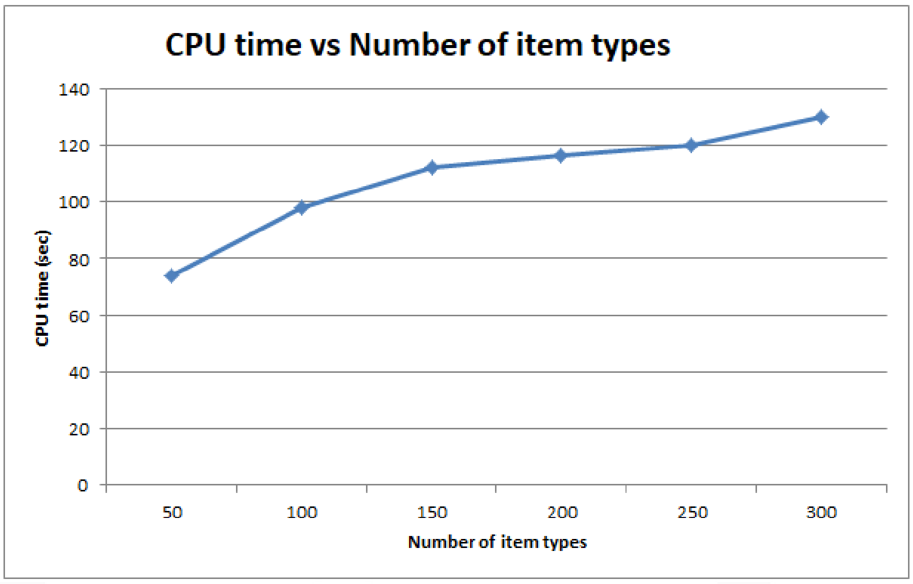


Figure 6.7: CPU time (sec) vs number of item types

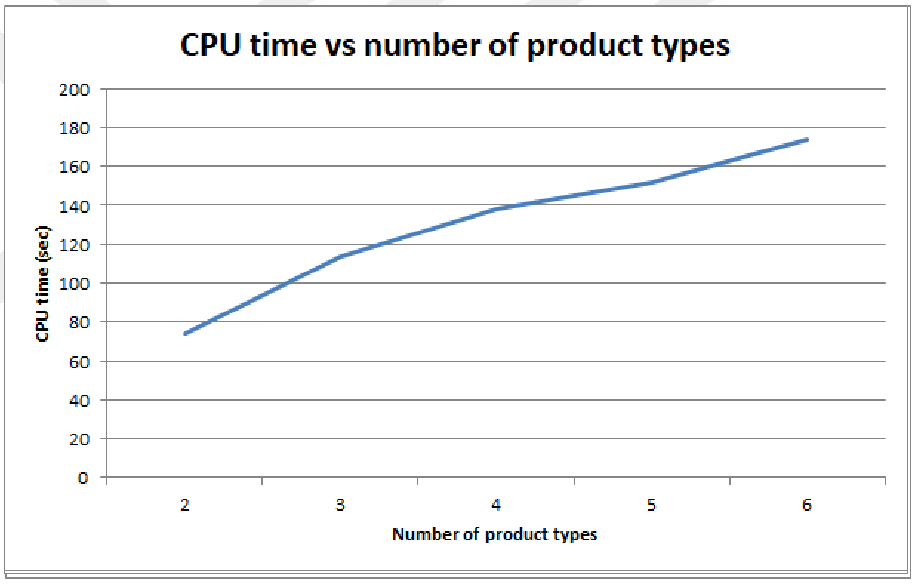


Figure 6.8: CPU time (sec) vs number of product types

For further analysis, experiments for several demand levels were carried out to analyze and compare the performance of Particle Swarm Optimization algorithm (PSO) and DA for the model Eq.s 6.13-6.19 (Table 6.8). 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, 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 the terms of run time. For the high

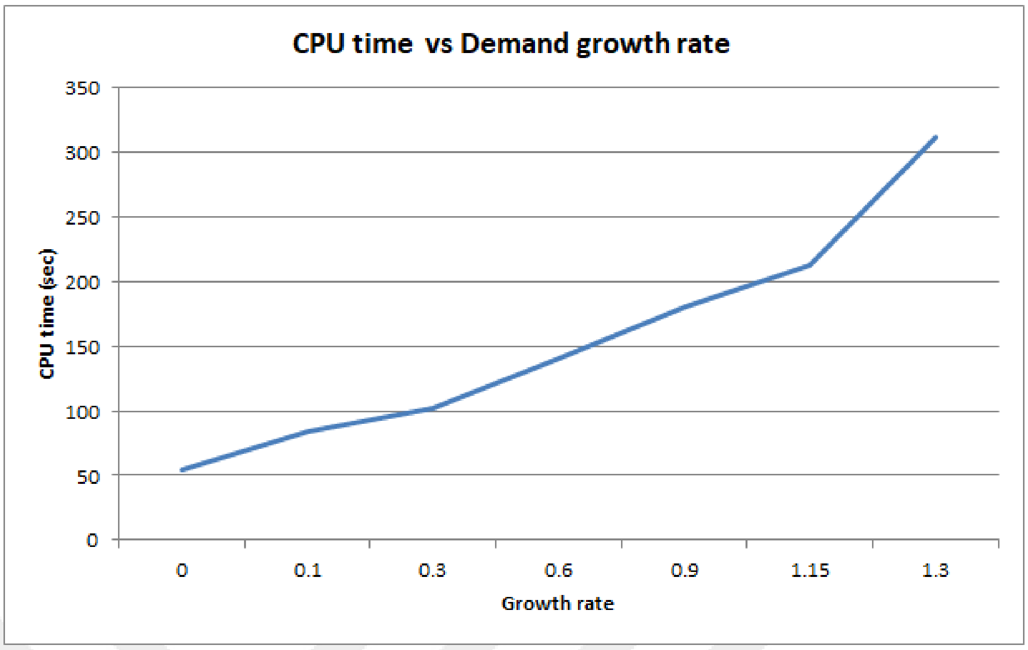


Figure 6.9: CPU time (sec) vs growth rates for the demand means of two product types

Table 6.8: Comparison of the PSO and DA for the Total Cost Minimization

Demand		PSO				DA			
Product 1	Product 2	Total cost	CPU (sec)	Lack of Product 1	Lack of Product 2	Total cost	CPU (sec)	Lack of Product 1	Lack of Product 2
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

demand levels, the PSO generates a large number of unnecessary patterns because generated patterns seem to be similar. It causes the high run time or licence error in gams because of the high number of variables. On the other hand, DA seems to generate patterns with different configuration because DA has better exploration feature.

7. LEXICOGRAPHIC METHOD FOR THE MULTI-OBJECTIVE CASE FOR THE DETERMINISTIC SSP AND HEURISTIC APPROACH

7.1 Formulation of the Multi-objective version of SSP

We primarily provide the nomenclature for the multi-objective SSP mathematical model in Table 7.1.

Table 7.1: Nomenclature for Multi-objective SSP

Symbol	**Abbreviation**
	Indices
i	index of (small) item types , $i \in \mathbb{I} := \{1, ..., m\}$;
j	index of minimal patterns, $j \in \mathbb{J}^* := \{1, ..., J\}$;
p	the index of the objective functions, $p \in \mathbb{P} := \{1, ..., k\}$;
	Parameters
a_{ij}	the number of item type i in pattern j (being generated by Column Generation);
L	threshold (lower bound) width of product;
c_j	trim loss (waste) for pattern j;
l_i	width of item type i;
d	the demand of product;
M	the large number for each product such as $M \geq d$;
s_j	Number of items in pattern j;
b_i	the available amount of item type i;
w_j	Width of pattern j;
	Decision variables
y_j	a binary variable that indicates whether a pattern j is used or not (number of set-ups);
x_j	the frequency of pattern j;

We transform the original SSP into a minimization version of SSP instead of an output maximization version by including trim loss, number of items in a product, and minimization of total number of set-ups. One important objective is to minimize the total trim loss in the skiving process in which the trim loss of each pattern can be determined as the waste part between the width of a pattern w_j and product threshold width L and calculated as $c_j = w_j - L$. Moreover, another objective is minimizing the total number of set-ups in a skiving process. Set-up can be determined as the pattern change and represented as the binary variable y_j that tracks if pattern j is used or not. Finally, use of the patterns with a minimum number of items (number of welds) is another objective. The number of items in a pattern is represented as s_j. Usage of pattern with many welds for the product effects the quality of the product negatively. If number of welds in a product increases, the quality of the products decreases. There may be several quality problems such as printing difficulties on the product, breaks in product rolls from the welds. These are potentially competing objectives in a production process. Therefore, we consider the problem as a multi-objective problem. The objective functions are given below;

First objective function is minimizing the total trim loss. Trim loss for each product is represented by c_j.

$$\min \ z_1 = \sum_{j \in \mathbb{J}^*} c_j x_j, \tag{7.1}$$

Second objective function is minimizing the number of items (welds) in the product. The number of items in the pattern is represented by s_j. Then, it helps us to use patterns with minimum items or welds.

$$\min \ z_2 = \sum_{j \in \mathbb{J}^*} s_j x_j, \tag{7.2}$$

Third objective function is minimizing the total number of set-ups in the production process.

$$\min \ z_3 = \sum_{j \in \mathbb{J}^*} y_j \tag{7.3}$$

Finally, by using Eq.s (7.1), (7.2) and (7.3) the multi-objective model of the new version SSP with the instance of $E := (m, l, L, b)$ is given through Eq.s(7.4)-(7.12).

$$\min \ z_1 = \sum_{j \in \mathbb{J}^*} c_j x_j \tag{7.4}$$

$$\min \ z_2 = \sum_{j \in \mathbb{J}^*} s_j x_j \tag{7.5}$$

$$\min \ z_3 = \sum_{j \in \mathbb{J}^*} y_j \tag{7.6}$$

$$\text{s.t.} \quad \sum_{j \in \mathbb{J}^*} a_{ij} x_j \leq b_i, \quad \forall i \in \mathbb{I} \tag{7.7}$$

$$\sum_{j \in \mathbb{J}^*} x_j \geq d, \tag{7.8}$$

$$x_j \leq M y_j \quad \forall j \in \mathbb{J}^*, k \in \mathbb{K} \tag{7.9}$$

$$x_j \in \mathbb{Z}_+, \quad \forall j \in \mathbb{J}^*, \tag{7.10}$$

$$a_{ij} \in \mathbb{Z}_+, \quad \forall i \in \mathbb{I}, j \in \mathbb{J}^* \tag{7.11}$$

$$y_j \in \{0, 1\}, \quad \forall j \in \mathbb{J}^* \tag{7.12}$$

Constraint (7.7) ensures that available items are sufficient for the produced amount by multiplying the number of pattern replications and the number of items required in each pattern. Constraint (7.8) balances the production according to demand. The set-up constraint given in Eq. (7.9) states if a pattern is used, then a set-up is required due to a pattern change. Moreover, it can be used up to M times, where M is an upperbound for the number of replications of pattern j. Constraints (7.10), (7.11) are integrality and non-negativity constraints, and constraint (7.12), imposes that y_j is binary variable.

7.2 The Proposed Solution Methodology for the Multi-objective SSP

In this research, lexicographic method is implemented where objective functions are ranked in the order of importance, and the solution of the previous objective becomes a constraint for the latter objective [Marler and Arora, 2004]. First, Column generation (CG) [Zak, 2003] is used to obtain the minimal pattern set with the minimum trim loss. Next, a minimal pattern set which is found by CG is used as the parameters (a_{ij}) for the IP in order to solve each objective function. Logic of the Lexicographic Method for multi-objective problem is presented as follows:

7.2.1 Lexicographic Method

A multi-objective program (MOP) as: [Samanlioglu et al., 2008]

$$\min \quad F(x) = F_1(x), F_2(x), \ldots, F_k(x) \tag{7.13}$$

$$s.t. \quad x \in X, \tag{7.14}$$

is assumed to have k ($k \geq 2$) competing objective functions ($x \in R^n, F_p : R^n \to R$) minimized simultaneously.

Definition: A decision vector $x^* \in X$ is an efficient (Pareto Optimal) for MOP, if there does not exist $x \in X$, $x^* \neq X$ such that $F_p(x) \leq F_p(x^*)$ for $p = 1, \ldots, k$ with strict inequality holding for at least one index p ($x^* \in X$ is efficient, $F_p(x^*)$, is non-dominated) [Samanlioglu et al., 2008].

We denote the importance ranking of these objective functions as $F_{[1]}(x)$, $F_{[2]}(x)$, $\ldots$, $F_{[k]}(x)$, where $[i]$, $i = 1, \ldots, k$ denotes the order indicating the objective function with the i^{th} priority. Therefore, $[i] = p$ indicates that the p^{th} objective function is in the i^{th} rank in terms of importance. In this study, $F_1(x)$ is minimizing the trim loss as given in Eq. 7.1, $F_2(x)$ is minimizing the number of skives as given in Eq. 7.2, and $F_3(x)$ is minimizing the number of set-ups as given in Eq. 7.3. Initially, we assign $F_{[1]}(x) = F_1(x)$, $F_{[2]}(x) = F_2(x)$, and $F_{[3]}(x) = F_3(x)$ or $[1] = 1, [2] = 2$, and $[3] = 3$.

In the LM, we solve the problem as:

$$\min \quad F_{[i]}(\mathbf{x}) \tag{7.15}$$

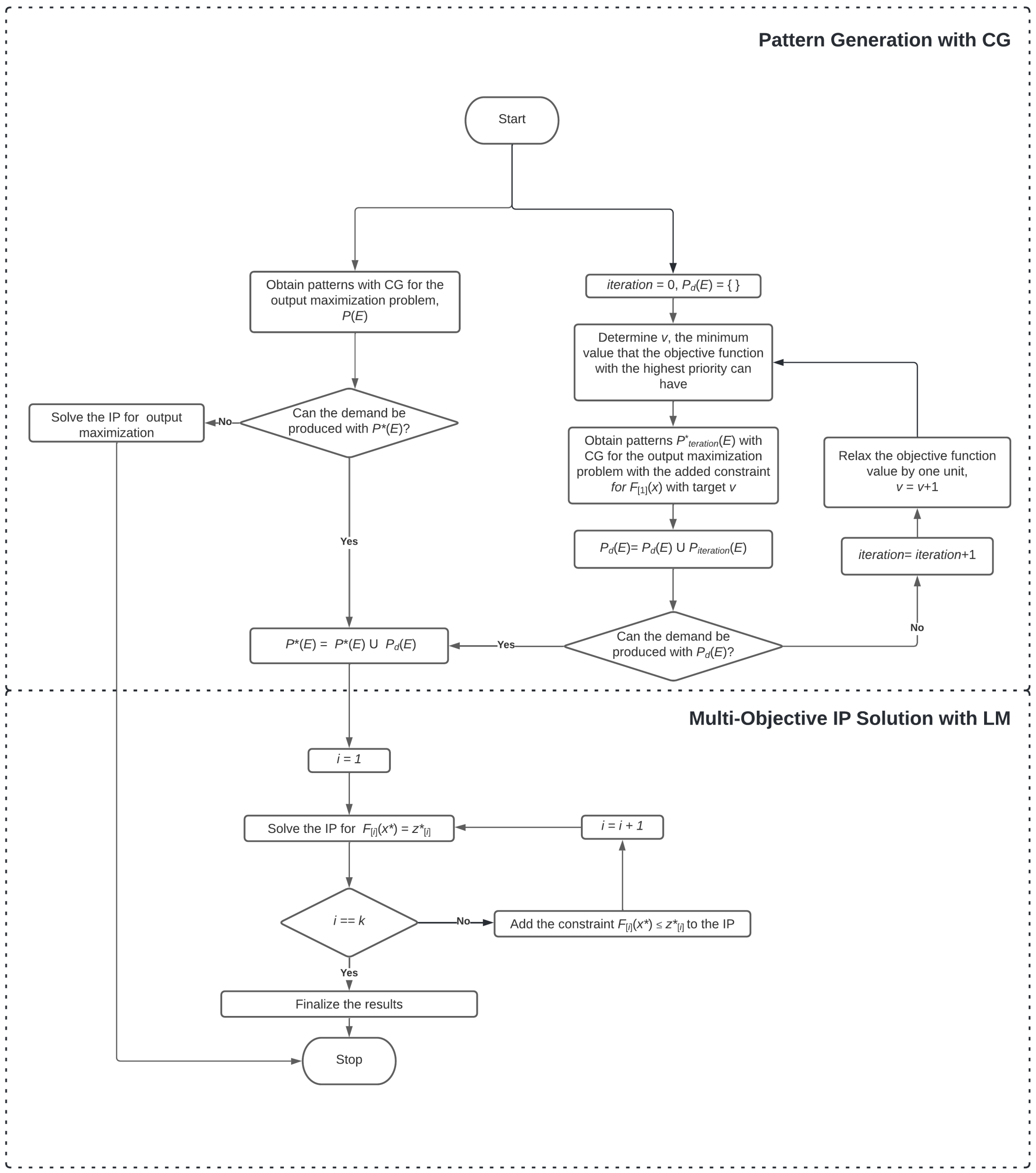
$$s.t \quad \mathbf{x} \in \mathcal{X} \tag{7.16}$$

$$F_{[n]}(\mathbf{x}) \leq F_{[n]}(\mathbf{x}^*_{[n]}), n = 1, 2, \dots, i-1, \tag{7.17}$$

where i represents the importance rank position of the objective functions and $\mathcal{X}$ represents the search space constrained by the SSP constraints given in Eq.s 7.7-7.12. $F_{[n]}(\mathbf{x}^*_{[n]})$ is the optimum of the n^{th} objective function. LM starts with $n = 1$, and solves the model for the most important objective function and the constraints given in Eq.s 7.7-7.12. With this first iteration, $F_{[1]}$'s optimal value is $F_{[1]}(\mathbf{x}^*_{[1]})$. However, this optimal solution at $\mathbf{x}^*_{[1]}$ is not necessarily optimal for the less important objective functions. Nonetheless, LM optimizes the other objective functions while retaining the previous optimal values. For $n > 1$, LM introduces the constraints given in Eq. 7.17 at every iteration. In a nutshell, LM filters down to a solution for the multi-objective problem with k objective functions throughout k iterations, adding a new constraint at every iteration [Marler and Arora, 2004][Stadler, 1988].

7.2.2 Integration of CG and LM

The flowchart of the solution methodology is given in Fig. 7.1. Initially, we run the CG for the output maximization problem. The output of this process is $P^*(E)$, that is, the minimal pattern set for producing the largest production amount possible. The reason for running the CG for the output maximization SSP without incorporating any of the objective functions or the demand constraint is two-fold: *i*) The LP relaxation gives an upper bound for the number of products produced. The decision-maker can then compare this amount to the demand and check whether the available patterns produce the demand. If not, we use the $P^*(E)$ to meet as much demand as possible by producing the maximum number of products given availabilities. *ii*) The patterns obtained from this problem can extend the search space for $F_{[n]}, n > 1$. After optimizing $F_{[1]}$, the constraint $F_{[1]}(x) \leq F_{[1]}(\mathbf{x}^*_{[1]})$ is added to the problem and $F_{[2]}$ is optimized. The presence of $P(E)$ has the potential of providing an alternative solution for $F_{[1]}$, and a better solution for $F_{[2]}$ and other less important objective functions.



Pattern Generation with CG
Start
Obtain patterns with CG for the output maximization problem, P(E)
iteration = 0, P_d(E) = { }
Determine v, the minimum value that the objective function with the highest priority can have
Solve the IP for output maximization
No
Can the demand be produced with P*(E)?
Obtain patterns P*_iteration(E) with CG for the output maximization problem with the added constraint for F_[1](x) with target v
Relax the objective function value by one unit, v = v+1
Yes
P_d(E)= P_d(E) ∪ P_iteration(E)
iteration= iteration+1
No
P*(E) = P*(E) ∪ P_d(E)
Yes
Can the demand be produced with P_d(E)?
Multi-Objective IP Solution with LM
i = 1
Solve the IP for F_[i](x*) = z*_[i]
i = i + 1
i == k
No
Add the constraint F_[i](x*) ≤ z*_[i] to the IP
Yes
Finalize the results
Stop

Figure 7.1: Flowchart of the Solution Methodology

Given that the availabilities meet the demand, we start searching for the patterns necessary for the objective function with the highest priority using CG. Besides we also incorporate the demand constraint given in Eq. 7.8. While the objectives have priorities, the produced patterns should also suffice the other objectives. We aim for the pattern pool to be sufficient to provide options for the lesser critical objective functions. For this reason, during the CG process, we start by assigning the value v, that is, the minimum value the objective function can have for each pattern. For example, for the trim loss minimization, $v = 0$, the possible minimum trim loss for each used pattern is 0. If the most important objective function were to be the number of items skived, we would assign $v = 2$, because there should be at least two items in a pattern.

Assuming that the most important objective function is the trim loss, we start solving the output maximization problem with a trim loss constraint where $v = c_j y_j = 0, \forall j$. This constraint enables us to search for patterns that have zero trim loss. If we can produce all demand with zero trim loss, we obtain the necessary pattern set to minimize the first objective function. If not, we relax the objective function target value by one, and search for patterns where $c_j y_j \leq 1, \forall j$. We check whether the patterns with zero or one unit of trim loss can produce the demand. We pursue relaxing the target objective function value until all demand is met. In other words, we go through iterations until $P_d(E)$ can satisfy the demand. In this notation, $P_d(E)$ is the minimal pattern set for the objective function with the highest priority given the demand d.

Once two pattern sets are obtained, for the next phase, we join two pattern sets on $P^*(E)$, that is $P^*(E) = P^*(E) \cup P_d(E)$. In this phase, we solve the IP model of each objective function with CPLEX iteratively in the order of priority. At each stage, the solution of the prior model becomes a constraint to preserve the solutions of the previous iterations. LM filter downs to the solution where each objective function is optimized with the order of importance.

7.2.3 An Illustrative Example

In this illustrative example, we provide the instance E:=(3, (25, 35, 45), 50, (2, 2, 2)) where the demand is 1 unit. With this example, we show the mechanisms of the algorithm. Firstly, $P^*(E)$ is produced for the output maximization problem. The maximum output is found as 3 units. The patterns produced are as follows:

$$P^*(E) = \mathbf{A}_{\mathbf{maxoutput}} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}, \mathbf{x}^*_{\mathbf{maxoutput}} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^T$$

Once $P^*(E)$ is available, we compare the demand to the maximum output. Since $d = 1 < 3$, the demand is producible. Next, we solve the output maximization problem by adding the new constraint of $v = \sum_j c_j y_j = 0$ and find the patterns that produce the maximum output with zero trim loss. For $v = 0$, we find that $P_1(E) = \begin{bmatrix} 2 & 0 & 0 \end{bmatrix}^T$ and the maximum production amount is 1, that satisfies the demand. An important note about the increments of v in this illustrative example is that for $v = 1$, there are no available patterns CG can find. Since the item widths are $l = \begin{bmatrix} 25 & 35 & 45 \end{bmatrix}$, their greatest common divisor is 5. Thus, any trim loss value between $v = 0$ and $v = 5$ is infeasible. In the light of this example, we suggest using the greatest common divisors of item widths as the increment once a feasible minimum trim loss is found.

$$P_1(E) = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}, \mathbf{x}^* = 1$$

The total production amount is 1 and it is equal to the demand. Hence, we stop to generate pattern and find the IP solution to the trim loss problem, which is $x^* = 1$ with a total trim loss of 0 units. The pattern generation process terminates with this solution. This example also proves by contradiction that $P^*(E) \subseteq P_d(E)$ or $P_d(E) \subseteq P^*(E)$ do not necessarily hold. A counter example for either statement can be found as $P_d(E) \setminus P^*(E) \neq \emptyset$ and $P^*(E) \setminus P_d(E) \neq \emptyset$ as given above.

In the next stage, $P^*(E)$ and $P_d(E)$ are merged. Because $P^*(E) \subseteq (P^*(E) \cup P_d(E))$, the merged pattern set both retains the optimal solution for the most important objective function and may also be capable of attaining a better solution then using $P^*(E)$ alone. We join the two sets over $P^*(E)$, such that $P^*(E) = P^*(E) \cup P_1(E)$ and run the LM to obtain the final solution.

$$P^*(E) = \mathbf{A}^* = \begin{bmatrix} 1 & 0 & 1 & 2 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}, \mathbf{x}^* = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^T$$

Table 7.2 displays the change in the final result when $P^*(E)$ or $P_1(E)$ are used marginally, as opposed to the combined minimal pattern set. While this instance is robust in terms of the secondary and tertiary objective functions, the trim loss changes greatly with respect to the pattern set used. In this instance, the minimum trim loss is 0 provided by $P_1(E)$. This solution does not have an alternative in $P^*(E)$, hence, it does not change when two minimal pattern sets are joined. In this example, the optimal results is obtained by $P_1(E)$.

Table 7.2: Effect of each pattern set on the result for the illustrative example

Pattern set	**Trim loss**	**Total number of items used**	**Total number of set-ups**
$P^*(E)$	10	2	1
$P_1(E)$	0	2	1
$P^*(E) \cup P_1(E)$	0	2	1

7.3 Numerical Example and Results

In this section, we solve a sample of the multi-objective SSP with the proposed solution methodology, and present its results in a larger experiment. GAMS is used to code the CG and the IP. Moreover, the solution methodology is executed on a computer with Intel Core i5-3230M, 2.60 GHz CPU, 4 GB RAM. The SSP instant is presented as $E := (m, l, L, b)$ is $E := (50, l, 1200, b)$, where l and b are given exclusively 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=[200 300 300 300 300 300 300 300 300 300 300 300 300 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100 100].

In this example, we set seven different demand levels as $d_1 = 100, d_2 = 200, d_3 = 300, d_4 = 400, d_5 = 500, d_6 = 600, d_7 = 700$ to demonstrate the changes on the objective function values as the demand increases. Initially, the priority of the objective functions are $F_{[1]}(x) = F_1(x)$, $F_{[2]}(x) = F_2(x)$, and $F_{[3]}(x) = F_3(x)$. The CG generates 116 minimal patterns for output maximization, that is, $\mid P^*(E) \mid$=116. The LP relaxation of output maximization model states that at most 776.6 products can be produced. Since the objective with the highest priority is the trim loss, we also run CG incorporating trim loss for output maximization. The sizes of each minimal pattern set for different demand levels are $\mid P_{100}(E) \mid$=9, $\mid P_{200}(E) \mid$=12, $\mid P_{300}(E) \mid$=33, $\mid P_{400}(E) \mid$=37, $\mid P_{500}(E) \mid$=43, $\mid P_{600}(E) \mid$=52, $\mid P_{700}(E) \mid$=68 and if we join all the sets and the output maximization set, total number of the new master set $P^*(E)$ includes 261 patterns. The next phase solves the IP model using LM. The results and comparisons are presented in Table 7.3. Moreover, we

Table 7.3: Objective Function Values for Different Demand Levels

Demand	**trim loss (z_1)**	**Total Number of items used (z_2)**	**Number of Set-ups (z_3)**
100	0	700	2
200	0	1488	4
300	0	2288	7
400	0	3164	9
500	0	4064	11
600	0	5060	17
700	0	6218	28

can see the number of patterns for each demand level in the set-up column of the Table 7.3. The third objective function, the number of set-ups, also equals to the number of patterns used, because each pattern requires a set-up. Because we are also minimizing the number of patterns used, even for the highest demand, a little more than 10% of the 261 patterns are sufficient to obtain the results. The abundance of these patterns extends the search space and provides more alternative solutions when we fix the value of important objective function values one-by-one.

Table 7.4: Non-dominated solutions for the demand=700

Objectives	trim loss (mm)	Set-up number	Total Number of items used
$z_1 >> z_2 >> z_3$	0	28	6218
$z_1 >> z_3 >> z_2$	0	25	6283
$z_3 >> z_1 >> z_2$	1541	23	6027
$z_3 >> z_2 >> z_1$	2334	23	5920
$z_2 >> z_1 >> z_3$	13070	35	5763
$z_2 >> z_3 >> z_1$	13070	35	5763

Table 7.5: Pattern types and Frequencies

d=100		**d=200**		**d=300**		**d=400**		**d=500**		**d=600**		**d=700**	
Items	**Frequency**	**Items**	**Frequency**	**Items**	**Frequency**	**Items**	**Frequency**	**Items**	**Frequency**	**Items**	**Frequency**	**Items**	**Frequency**
7	34	7	50	7	34	7	36	7	36	8	10	8	10
7	66	7	50	7	50	7	50	7	50	7	36	7	38
		8	88	8	30	8	48	8	48	7	50	7	50
		7	12	7	50	7	36	7	36	9	7	9	12
				8	75	8	84	8	84	8	50	8	50
				7	12	9	50	9	48	8	83	9	8
				8	49	7	23	9	42	9	50	10	8
						8	50	7	23	10	1	8	72
						9	23	8	50	9	42	9	50
								9	50	11	13	11	7
								9	33	10	25	12	4
										7	50	12	10
										8	50	11	12
										9	50	10	10
										9	33	9	37
										9	33	11	14
										10	17	10	35
												7	50
												11	7
												8	50
												10	8
												10	14
												9	30
												9	17
												9	33
												10	17
												13	5

For the second objective which is minimizing number of items in each product, it is important to obtain a product with minimum number of welds because of the quality factor. For further analysis, the number of products produced and their item numbers are presented in Table 7.5. Furthermore, for $d = 700$ different non-dominated solutions that are obtained by changing the order of importance of objective functions Table 7.4. Even though, we present these non-dominated solutions by considering the all combinations of order of importance of objective functions, the preferred and realistic order of importance in paper production and printing industry is $z_1 >> z_2 >> z_3$.

7.4 Heuristic Approach

We also developed a hybrid solution approach that integrates modified version of the Dragonfly Algorithm (DA) and Constructive Heuristic (CH) [Poldi and Arenales, 2009] to solve the multi-objective problem when the order of importance $z_1 >> z_2 >> z_3$. As mentioned before, this order of importance is more realistic in paper production and printing industry. CH is known as the exhaustive repetition algorithm in which the most efficient pattern is found and replicated as many times as possible [Hinxman, 1980]. For our main order of importance which is $z_1 >> z_2 >> z_3$, first, DA produces the set of minimal skiving patterns with the minimum trim loss. Next, the overall algorithm sorts the minimal patterns in a decreasing order by using two sorting criteria; the first criterion is the trim loss, and the second criterion is the total item numbers in a pattern. First, minimal patterns are sorted to obtain patterns with the minimum trim loss to minimize the total trim loss; then obtained minimal patterns are again sorted in an increasing order according to the second criterion to obtain minimal patterns with the minimum number of items from these sorted minimal pattern set. CH replicates the minimal patterns with the minimum trim loss and with minimum number of items as much as it can by using items efficiently under availability limitation to satisfy the product demand. In this way, CH selects the most repetitive patterns from the sorted minimal pattern set in order to minimize the number of set-ups. Integration of modified DA and the CH as a solution methodology are presented below; In the algorithm in Alg. 6, the objective functions are arranged in the order of importance as follows; $minz_1 >> minz_2 >> minz_3$. DA is used to obtain the minimal patterns with the minimum trim loss. We sort the minimial patterns in an increasing order and select the pattern with the minimum trim loss, and if we have more than one, we select

```
1  START ;
2  Determine parameter values;
3  Set $P^*(E) := \emptyset$ ;
4  while $Demand \geq 0$ or $\sum_{i \in \mathbb{I}} b_i^{'} \geq L$ do
5      Run the DA to generate the minimal patterns for the $P^{*^{'}}(E)$ ;
6      Set $P^*(E) := P^*(E) \bigcup P^{*'}(E)$;
7      Select minimal patterns with minimum trim loss $j_t := \arg\min\{c_j : j \in \mathbb{J}^*\}$;
8      Set $\mathbb{J}_t^* := j_t$;
9      Select minimal patterns with minimum welds $j_n := \arg\min\{s_j : j \in \mathbb{J}_t^*\}$;
10     Set $\mathbb{J}_n^* := j_n$;
11     $b_i^{'} = b_i$;
12     while $|\mathbb{J}_n^*| > 0$ do
13         Compute fraction $f_{ij} = \frac{b_i}{a_{ij}}, \forall j \in \mathbb{J}^*, i \in \mathbb{I}$ ;
14         Find the minimum fraction for each pattern $f_j^{min} = min\{f_{ij}\}, \forall j \in \mathbb{J}^*$ ;
15         Select the pattern with maximum fraction $j_{max} := \arg\max\{f_j^{min}\}$ and
            $f_{j_{max}}^* = max\{f_j^{min}\}$;
16         Replicate the selected pattern $j_{max}$ as much as possible without
            exceeding the availability of associated items by using $a_{ij}$,
            $x_{j_{max}} = \left\lfloor f_{j_{max}}^* \right\rfloor$;
17         Set $\mathbb{J}_{use}^* := \mathbb{J}_{use}^* \bigcup j_{max}$ ;
18         Set $\mathbb{J}_n^* := \mathbb{J}_n^* \setminus j_{max}$ ;
19         Update item availability $b_i^{'} = b_i^{'} - (x_{j_{max}} * a_{ij}), \forall i \in \mathbb{I}$ ;
20         Update demand, $demand = demand - x_{j_{max}}$;
21     end
22 end
23 Output= $\sum_{j \in \mathbb{J}_{use}^*} x_j$
24 ; STOP
```

Algorithm 6: Solution methodology

all of them. Next, we sort the selected patterns in an increasing order according to the number of items used in each pattern. We select the pattern which contains the minimum number of items. In this way, we filter the patterns first according to the minimum trim loss and then filter the filtered patterns according to the minimum number of items. Next, we apply CH to these filtered patterns in order to find the patterns that can be replicated with maximum number of times among all filtered patterns (inner while loop of algorithm 6). In this way, we can decrease the number of set-ups in the skiving process. In the constructive heuristic, we first compute the fraction of each item f_{ij} in each pattern by using $\frac{b_i}{a_{ij}}$. Then, we find the minimum fraction f_j^{min} for each pattern. Then, we compare all fractions and obtain the maximum one. We select the pattern with the maximum fraction $f^*_{j_{max}}$. We replicate the selected pattern as much as the fraction of that pattern to determine the frequency of pattern in the skiving process. Next, we update demand and availability of items. If CH could not find the pattern set that satisfies the demand, the procedure returns to the beginning of the overall algorithm. As a significant contribution, DA uses an updated item availability which is obtained by constructive heuristic to generate new patterns with a different configuration (Fig. 7.2).

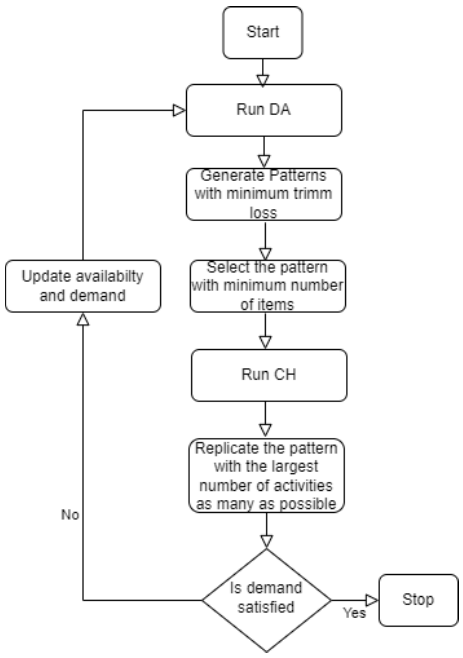

Start
Run DA
Generate Patterns with minimum trimm loss
Select the pattern with minimum number of items
Run CH
Replicate the pattern with the largest number of activities as many as possible
Is demand satisfied
Yes
Stop
No
Update availabilty and demand

Figure 7.2: Flow chart of the proposed Heuristic

7.5 Numerical Example and Results

In this section, a multi-objective version of the SSP is solved by using the proposed solution methodology, and its results are presented. The proposed algorithm is coded in Matlab R2017b, and the algorithm is executed on a computer with Intel core i5-3230M, 2.60 GHz CPU, 4 GB RAM. The instance SSP $E := (m, l, L, b)$ is $E := (50, l, (1200), b)$ where the demand is $d = 500$. The width of small items and product are denoted in millimeters. l where 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 DA are $B = 0.05, \alpha = 0.5, \beta = 0.03, \gamma = 0.07, \eta = 0.05, \epsilon = 0.05$, ω=0.05.
The primary objective function is determined as the minimization of the total trim loss (z_1)(Eq.7.1); the secondary objective is the minimization of the number of items in a product (z_2)(Eq.7.2), and the set-up reduction is defined as the third objective function(z_3)(Eq.7.3). By using the proposed methodology, 40 minimal patterns have been found, and 16 of them have been selected. Frequency of each patterns are represented as the **X** vector as follows:

X=[50 100 50 0 50 100 50 0 16 28 20 0 0 2 16 18];
With the proposed solution methodology for the multi-objective SSP, the first objective function, the total trim loss is found as 0 mm. Moreover, the total number of items used is 4260 units with varying width. Finally, 12 different patterns are used to produce the demand. It means that the number of set-up change is found as 12. Then, integer programming is used to solve the same problem with lexicographic method to obtain the solution. Column generation is used iteratively to obtain the

most efficient pattern set, and IP is used to solve each objective function based on the order of importance in lexicographic method. Results and comparison are presented in Table 7.6.

Table 7.6: Comparison of Results

Objectives (Min)	Heuristic obj.Values	IP obj.Values	Absolute Gap
Min Total Trim Loss	0 mm	0 mm	0
Min Number of Items	4260 unit	4064 unit	196
Min Number of Set-up	12 times	11 times	1
CPU	40,0865 sec	180.562 sec	

Moreover, for the second objective which is minimizing the number of items in each product, it is important to obtain a product with the minimum number of welds because of the quality factor. For CG-IP and proposed heuristic, the distribution of the number of products produced and their item numbers are presented in Fig. 7.3. When the size or the tightness of the problem increases ($d \geq 770$ for this example),

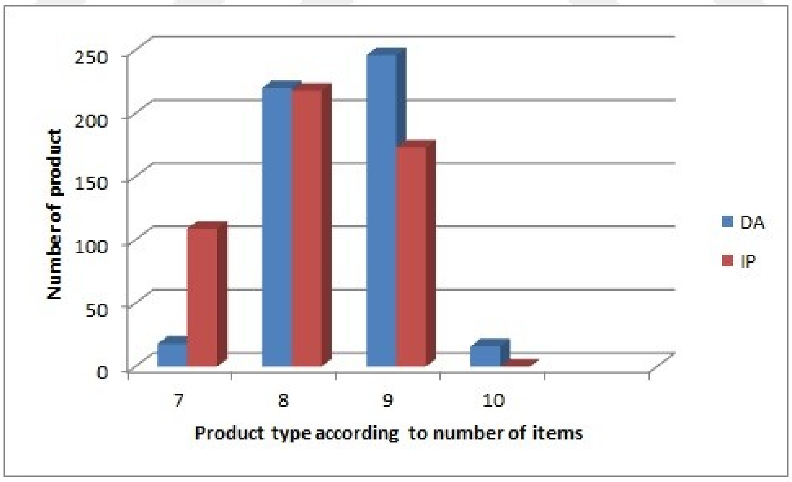


Figure 7.3: Comparison of DA and CG-IP in terms of produced product types

the number of patterns also increases in the IP solution. In this case, it becomes a difficult task to solve large problems by using IP in a reasonable time.

8. CONCLUSIONS

The skiving process is essential in various industries such as steel, textiles, paper. Although the skiving process has been a part of the industry for a long time, it 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 an unexplored area.

We did not encounter the stochastic version of the SSP in the literature. Therefore, as a contribution to the literature, the pure SSP is reformulated as the stochastic version and the cost minimization problem under uncertain demand. Two-stage stochastic programming paradigm is proposed for this stochastic version of the SSP. The model includes the production cost, which depends on the pattern widths as the first-stage cost, and the overproduction and underproduction costs as the second-stage costs. As a solution methodology, the combination of CG, PHA, and B&B is used to obtain integer solutions of the stochastic SSP problem. In the proposed methodology, the stochastic problem is decomposed into deterministic subproblems. CG is applied to each deterministic subproblem to obtain the minimal patterns with minimum cost while satisfying the demand for each subproblem. In this step, a new constraint is added to the CG which provides to search for patterns with the minimum cost. In this way, CG is used iteratively to generate a minimal pattern set with the minimum cost while satisfying the demand. Selected minimal patterns are fed into the stochastic SSP model as parameters. As an important point, a scenario decomposition-based PHA is run to solve the stochastic program in each node of the search tree of the B&B algorithm to obtain integer solutions. Finally, several experiments with a number of scenarios are carried out, and results of these experiments are presented. We also solve these problems by using CPLEX solver to compare and to validate the solution quality of our methodology. According to the results, solutions of the PHA do not differ from the CPLEX solver for all experiments. Moreover, CPU time of the proposed methodology is analyzed for each experiment. According to the results, CPU time seems to increase linearly when we increase the scenarios.

Afterwards, our single-product stochastic SSP is extended to the multi-product stochastic model in which additional costs such as set-up cost, raw material cost exist. In this extended multi-product stochastic SSP model, several random variables exist such as random demand for each product and random yield of skiving process. Two-stage stochastic programming is used for this stochastic model, and SAA method is preferred to overcome the difficulties of working with a large number of scenarios. As a proposed solution methodology, an iterative two-phase heuristic algorithm is developed to solve the multi-product stochastic SSP problem. DA is adapted 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 provides the pattern set with a minimized trim loss, whereas the later phase minimizes the total costs. While cost minimization is the primary objective in this problem, the 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 is determined. Several experiments are carried out with different parameter configurations as a parameter tuning. According to the results, the mean absolute difference rate between the highest and the lowest values of the average objective function is approximately 0.3%. For our example, the DA is robust against parameter changes and produces an abundance of patterns. Furthermore, we increase the dimensions of the problem by increasing the number of item types up to 300, by increasing the number of product types up to 6, which means increasing the tightness of the problem. According to the results of these experiments, the proposed heuristic can solve the problem in a reasonable time. Finally, we compare the results of pattern generation performances DA and well-known metaheuristic PSO. Given the results, DA is superior to the PSO in both obtained objective function value and CPU time.

Moreover, a multi-objective version of the SSP is considered in which minimization of trim loss, the number of items in a product, and the number of pattern changes (set-ups) are the objectives. Lexicographic method is preferred for this multi-objective problem where objectives are ordered according to their importance, and optimal value of the priority objective becomes a constraint for the later objectives. Minimization of the trim loss (waste) is the primary objective of our problem. Minimization of the number of items is presented as the secondary objective. Minimization of the number of set-ups is considered as the third objective. As a solution approach, CG and IP are used to obtain the solution where a master minimal pattern set is generated by CG and used as the parameter for the IP. Then, IP is used to solve the objective functions by using generated master minimal pattern set. Furthermore DA and CH are hybridized as a heuristic method in order to solve the same problem and compare the results. In this heuristic, DA obtains the minimal patterns with the minimum trim loss and with minimum number of items; then CH replicates the efficient patterns to satisfy the demand while minimizing the number of set-ups. Comparison of CG-IP and heuristic method is presented. According to the results, heuristic method is as good as the CG-IP in finding the same result of primary objective function for our example in a reasonable period of time. On the other hand, for the secondary and tertiary objective functions, absolute gaps between results of the CG-IP and heuristic methods exist. When the problem gets larger, especially gets close to the upperbound (LP relaxation), efficiency of the IP decreases to obtain the integer solutions because of the run time increment while proposed heuristic obtains the feasible solutions in a reasonable period of time.

In this section, we present limitations of our study and extensions for future work. We used the horizontal decomposition-based algorithm (PHA) in our solution methodology for the single-product stochastic SSP. Since there is no solution approach for stochastic version of the SSP, we could not evaluate the efficiency of the proposed solution methodology in terms of CPU time. Therefore, as future research, we can adapt a vertical decomposition-based solution approach such as L-shaped algorithm to evaluate the efficiencies of vertical and horizontal decomposition-based algorithms for the single-product stochastic SSP. In this research, we utilized B&B algorithm to run the PHA in each node in order to solve nodal relaxation for the single-product stochastic SSP. Even though worst case complexity of the B&B algorithm is known as exponential, it is evaluated with its average case complexity which is lower than exponential in the literature because of its efficient search and branching strategies. Therefore, as future work, we can also adapt the Branch and Cut algorithm to our solution methodology for the single-product stochastic SSP in order to analyze the efficiency and try to obtain significant improvement in terms of CPU time. Since the SSP is a classical $\mathcal{NP}$-hard problem, there is no exact solution approach in the literature for the multi-product stochastic SSP. Therefore, also as future research, we can adapt a well-known heuristics in the literature to the multi-product stochastic SSP and compare with the modified DA in order to investigate thoroughly the quality of feasible solutions obtained with DA.

BIBLIOGRAPHY

Abdel-Basset, M., Luo, Q., Miao, F., and Zhou, Y. (2017). Solving 0–1 knapsack problems by binary dragonfly algorithm. In *International Conference on Intelligent Computing*, pages 491–502. Springer.

Ágoston, K. C. (2019). 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*, 2019.

Ahmed, S. and Shapiro, A. (2002). The sample average approximation method for stochastic programs with integer recourse. *SIAM Journal of Optimization*, 12:479–502.

Alem, D. J., Munari, P. A., Arenales, M. N., and Ferreira, P. A. V. (2010). On the cutting stock problem under stochastic demand. *Annals of Operations Research*, 179(1):169–186.

Álvarez-Valdés, R., Parajón, A., et al. (2002). A tabu search algorithm for large-scale guillotine (un) constrained two-dimensional cutting problems. *Computers & Operations Research*, 29(7):925–947.

Arbib, C. and Marinelli, F. (2005). Integrating process optimization and inventory planning in cutting-stock with skiving option: An optimization model and its application. *European Journal of Operational Research*, 163(3):617–630.

Arbib, C., Marinelli, F., Rossi, F., and Di Iorio, F. (2002). Cutting and reuse: An application from automobile component manufacturing. *Operations Research*, 50(6):923–934.

Assmann, S. F., Johnson, D. S., Kleitman, D. J., and Leung, J.-T. (1984). On a dual version of the one-dimensional bin packing problem. *Journal of Algorithms*, 5(4):502–525.

Atakan, S. and Sen, S. (2018). A progressive hedging based branch-and-bound algorithm for mixed-integer stochastic programs. *Computational Management Science*, 15(3):501–540.

Aydin, N. (2012). *Sampling based progressive hedging algorithms for stochastic programming problems*. PhD thesis, Wayne State University.

Baiche, K., Meraihi, Y., Hina, M. D., Ramdane-Cherif, A., and Mahseur, M. (2019). Solving graph coloring problem using an enhanced binary dragonfly algorithm. *International Journal of Swarm Intelligence Research (IJSIR)*, 10(3):23–45.

Beraldi, P., Bruni, M. E., and Conforti, D. (2009). The stochastic trim-loss problem. *European Journal of Operational Research*, 197(1):42–49.

Birge, J. R. and Louveaux, F. (2011). *Introduction to stochastic programming*. Springer Science & Business Media.

Chauhan, S. S., Martel, A., and D'Amour, S. (2008). Roll assortment optimization in a paper mill: An integer programming approach. *Computers & Operations Research*, 35(2):614–627.

Chen, C.-L. S., Hart, S. M., and Tham, W. M. (1996). A simulated annealing heuristic for the one-dimensional cutting stock problem. *European Journal of Operational Research*, 93(3):522–535.

Chen, Y., Song, X., Ouelhadj, D., and Cui, Y. (2019). A heuristic for the skiving and cutting stock problem in paper and plastic film industries. *International Transactions in Operational Research*, 26(1):157–179.

Demirci, M. C., Schaefer, A. J., and Rosenberger, J. M. (2008). Column generation within the L-shaped method for stochastic linear programs. *Computational Optimization and Applications*.

Ducatelle, F. and Levine, J. (2001). Ant colony optimisation for bin packing and cutting stock problems. In *UK Workshop on Computational Intelligence (UKCI-01), Edinburgh*, pages 1–16.

Gilmore, P. C. and Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859.

Gilmore, P. C. and Gomory, R. E. (1963). A linear programming approach to the cutting stock problem — Part II. *Operations Research*, 11(6):863–888.

Golfeto, R. R., Moretti, A. C., and Salles Neto, L. L. d. (2009). A genetic symbiotic algorithm applied to the one-dimensional cutting stock problem. *Pesquisa Operacional*, 29(2):365–382.

Hammouri, A. I., Samra, E. T. A., Al-Betar, M. A., Khalil, R. M., Alasmer, Z., and Kanan, M. (2018). A dragonfly algorithm for solving traveling salesman problem. In *2018 8th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, pages 136–141. IEEE.

Haugen, K. K., Løkketangen, A., and Woodruff, D. L. (2001). Progressive hedging as a meta-heuristic applied to stochastic lot-sizing. *European Journal of Operational Research*, 132(1):116–122.

Hinterding, R. and Khan, L. (1993). Genetic algorithms for cutting stock problems: With and without contiguity. In *Progress in Evolutionary Computation*, pages 166–186. Springer.

Hinxman, A. (1980). The trim-loss and assortment problems: A survey. *European Journal of Operational Research*, 5(1):8–18.

Jahromi, M. H., Tavakkoli-Moghaddam, R., Makui, A., and Shamsi, A. (2012). Solving an one-dimensional cutting stock problem by simulated annealing and tabu search. *Journal of Industrial Engineering International*, 8(1):1–8.

Jin, H., Wang, Z., and Chien, C.-F. (2012). A cut-to-order strategy for one-dimensional cable cutting and a case study. *Journal of the Chinese Institute of Industrial Engineers*, 29(8):572–586.

Johnson, M., Rennick, C., and Zak, E. (1997). Case studies from industry: Skiving addition to the cutting stock problem in the paper industry. *Siam Review*, 39(3):472–483.

José Alem, D. and Morabito, R. (2012). Production planning in furniture settings via robust optimization. *Computers & Operations Research*, 39(2):139–150.

Kall, P., Wallace, S. W., and Kall, P. (1994). *Stochastic programming*. Springer.

Kazemi Zanjani, M., Ait-Kadi, D., and Nourelfath, M. (2013). A stochastic programming approach for sawmill production planning. *International Journal of Mathematics in Operational Research*, 5(1):1–18.

Kleywegt, A. J., Shapiro, A., and Homem-de Mello, T. (2002). The sample average approximation method for stochastic discrete optimization. *SIAM Journal on Optimization*, 12(2):479–502.

Lai, K. and Chan, J. W. (1997). Developing a simulated annealing algorithm for the cutting stock problem. *Computers & Industrial Engineering*, 32(1):115–127.

Leung, T., Yung, C., and Troutt, M. D. (2001). Applications of genetic search and simulated annealing to the two-dimensional non-guillotine cutting stock problem. *Computers & Industrial Engineering*, 40(3):201–214.

Levine, J. and Ducatelle, F. (2004). Ant colony optimization and local search for bin packing and cutting stock problems. *Journal of the Operational Research Society*, 55(7):705–716.

Li, J., Lu, J., Yao, L., Cheng, L., and Qin, H. (2019). Wind-solar-hydro power optimal scheduling model based on multi-objective dragonfly algorithm. *Energy Procedia*, 158:6217–6224.

Liang, K.-H., Yao, X., Newton, C., and Hoffman, D. (2002). A new evolutionary approach to cutting stock problems with and without contiguity. *Computers & Operations Research*, 29(12):1641–1659.

Lokketangen, A. and Woodruff, D. L. (1996). Progressive hedging and tabu search applied to mixed integer (0, 1) multistage stochastic programming. *Journal of Heuristics*, 2(2):111–128.

Lu, H.-C., Huang, Y.-H., and Tseng, K.-A. (2013). An integrated algorithm for cutting stock problems in the thin-film transistor liquid crystal display industry. *Computers & Industrial Engineering*, 64(4):1084–1092.

Mafarja, M. M., Eleyan, D., Jaber, I., Hammouri, A., and Mirjalili, S. (2017). Binary dragonfly algorithm for feature selection. In *2017 International Conference on New Trends in Computing Sciences (ICTCS)*, pages 12–17. IEEE.

Marler, R. T. and Arora, J. S. (2004). Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395.

Martinovic, J., Delorme, M., Iori, M., Scheithauer, G., and Strasdat, N. (2020). Improved flow-based formulations for the skiving stock problem. *Computers & Operations Research*, 113:104770.

Martinovic, J., Jorswieck, E., and Scheithauer, G. (2016). The skiving stock problem and its application to cognitive radio networks. *IFAC-PapersOnLine*, 49(12):99–104.

Martinovic, J. and Scheithauer, G. (2016a). Integer linear programming models for the skiving stock problem. *European Journal of Operational Research*, 251(2):356–368.

Martinovic, J. and Scheithauer, G. (2016b). The proper relaxation and the proper gap of the skiving stock problem. *Mathematical Methods of Operations Research*, 84(3):527–548.

Martinovic, J. and Scheithauer, G. (2017). An upper bound of $\Delta(e) < 3/2$ for skiving stock instances of the divisible case. *Discrete Applied Mathematics*, 229:161–167.

Martinovic, J. and Scheithauer, G. (2018). Characterizing IRDP-instances of the skiving stock problem by means of polyhedral theory. *Optimization*, 67(10):1797–1817.

Martinovic, J. and Scheithauer, G. (2019). New theoretical investigations on the gap of the skiving stock problem. *Pesquisa Operacional*, 39(1):1–35.

Mirjalili, S. (2016). Dragonfly algorithm: A new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. *Neural Computing and Applications*, 27(4):1053–1073.

Poldi, K. C. and Arenales, M. N. (2009). Heuristics for the one-dimensional cutting stock problem with limited multiple stock lengths. *Computers & operations research*, 36(6):2074–2081.

Rockafellar, R. T. and Wets, R. J.-B. (1991). Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of operations research*, 16(1):119–147.

Samanlioglu, F., Ferrell Jr, W. G., and Kurz, M. E. (2008). A memetic random-key genetic algorithm for a symmetric multi-objective traveling salesman problem. *Computers & Industrial Engineering*, 55(2):439–449.

Scheithauer, G. (2017). *Introduction to cutting and packing optimization: Problems, modeling approaches, solution methods*, volume 263. Springer.

Sculli, D. (1981). A stochastic cutting stock procedure: Cutting rolls of insulating tape. *Management Science*, 27(8):946–952.

Shapiro, A. (2003). Inference of statistical bounds for multistage stochastic programming problems. *Mathematical Methods of Operations Research*, 58(1):57–68.

Shapiro, A. (2005). Complexity of two and multi-stage stochastic programming problems. *Tutorial Notes for School of Industrial and Systems Engineering. Atlanta, Georgia*, pages 30332–0205.

Shapiro, A., Dentcheva, D., and Ruszczyński, A. (2014). *Lectures on Stochastic Programming: Modeling and Theory*. SIAM.

Shapiro, A. and Homem-de Mello, T. (1998). A simulation-based approach to two-stage stochastic programming with recourse. *Mathematical Programming*, 81(3):301–325.

Shirani, M. R. and Safi-Esfahani, F. (2020). Dynamic scheduling of tasks in cloud computing applying dragonfly algorithm, biogeography-based optimization algorithm and mexican hat wavelet. *The Journal of Supercomputing*, pages 1–59.

Stadler, W. (1988). *Multicriteria Optimization in Engineering and in the Sciences*, volume 37. Springer Science & Business Media.

Wang, D., Xiao, F., Zhou, L., and Liang, Z. (2020). Two-dimensional skiving and cutting stock problem with setup cost based on column-and-row generation. *European Journal of Operational Research*, 286(2):547–563.

Wäscher, G., Haußner, H., and Schumann, H. (2007). An improved typology of cutting and packing problems. *European Journal of Operational Research*, 183(3):1109–1130.

Wets, R. J. (2002). Stochastic programming models: wait-and-see versus here-and-now. In *Decision Making Under Uncertainty*, pages 1–15. Springer.

Yang, G., Tang, W., and Zhao, R. (2017). An uncertain furniture production planning problem with cumulative service levels. *Soft Computing*, 21(4):1041–1055.

Zak, E. (2003). The skiving stock problem as a counterpart of the cutting stock problem. *International Transactions in Operational Research*, 10(6):637–650.

Curriculum Vitae

Personal Information

Name, Surname	: Tolga Kudret Karaca

Education

Bachelor’s Degree	: Economy, (Anadolu University Faculty of Economics)
Master’s Degree	: Engineering Managment, (Arel University, I.S.)
Bildiği Yabancı Diller	: English

Job Experiences

Corporation Name and Date:	Plastikkart Akıllı Kart Sanayi ve Ticaret A.Ş. (2002-2010)
	Etisan Etiket Ltd. Şti. (2011-2020)