SET-COVERING BASED HEURISTIC APPROACHES FOR THE PROBLEMS FROM THE PRINTING INDUSTRY

A Dissertation

by

Emre Çankaya

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Doctor of Philosophy

in the
Department of Industrial Engineering

Özyeğin University
May 2022

SET-COVERING BASED HEURISTIC APPROACHES FOR THE PROBLEMS FROM THE PRINTING INDUSTRY

Approved by:

Associate Professor Ali Ekici,
Advisor
Department of Industrial Engineering
Özyeğin University

Professor Serhan Duran
Department of Industrial Engineering
Middle East Technical University

Associate Professor Okan Örsan Özener
Department of Industrial Engineering
Özyeğin University

Associate Professor Ertan Yakıcı
Department of Industrial Engineering
NDU Turkish Naval Academy

Date Approved: 23 May 2022

Assistant Professor Elvin Çoban
Department of Industrial Engineering
Özyeğin University

To my family...

ABSTRACT

In this thesis, we focus on two different planning production problems coming from the printing industry. The first problem is the label printing problem and the second one is a variant of the cover printing problem. In both studies, the problems take into account the best assignment of products on different templates in order to meet demand requirements. In the first part of the thesis, we focus on minimizing the waste, whereas the goal is to minimize the total production cost in the second problem. In these problems, each template can contain fixed number of products and suitable assignment of products to each template provides to decrease the waste of products and to improve the efficient of the printing production with minimum waste. We handle the first problem into two different cases. Each product can be assigned to a single template in the first case, whereas each product can be assigned to the all templates in the second case. In the second problem, we consider only second case due to the organizational constraints. We propose two-phase heuristic algorithms to solve these problems since the studied problems are hard. We conduct an extensive computational studies on real-world and randomly generated instances in order to assess the performances of the proposed algorithms and compare the performances of the proposed algorithms with respect to existing solution algorithms in the literature in terms of solution quality.

ÖZETÇE

Bu tezde, matbaacılık sektöründen gelen iki farklı üretim planlama problemine odaklanılmıştır. İlk problem etiket baskı problemi, ikincisi ise kapak baskı probleminin bir çeşididir. Her iki çalışmada da, problemler talep gereksinimlerini karşılamak için ürünlerin farklı şablonlarda en iyi şekilde atanmasını hesaba katmaktadır. Tezin ilk bölümünde israfı minimize etmeye odaklanılırken, ikinci problemde amaç toplam üretim maliyetini minimize etmektir. Bu problemlerde, her şablon sabit sayıda ürün içerebilir ve her şablona uygun ürün atanması, ürün israfını azaltmayı ve minimum atık ile baskı üretiminin verimliliğini artırmayı sağlar. İlk problem iki farklı durumda ele alınmıştır. İlk durumda her ürün tek bir şablona atanabilirken, ikinci durumda her ürün tüm şablonlara atanabilmektedir. İkinci problemde, organizasyonel kısıtlama nedeniyle sadece ikinci durum ele alınmıştır. Çalışılan problemler zor olduğu için bu problemleri çözmek için iki aşamalı sezgisel algoritmalar önerilmiştir. Önerilen algoritmaların performanslarını değerlendirmek ve önerilen algoritmaların performanslarını literatürdeki mevcut çözüm algoritmaları ile çözüm kalitesi açısından karşılaştırmak için gerçek dünya ve rastgele oluşturulmuş örnekler üzerinde kapsamlı bir hesaplama çalışması yapılmıştır.

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my advisor Ali Ekici and to Okan Örsan Özener for their patience, guidance and valuable support of my study and related research throughout the completion of this thesis. I also would like to thanks to all the faculty members of the Özyeğin University Industrial Engineering Department who have always helped me with my research, studies and academic development. It has always been a great honor and inspiration for me to be a part of this invaluable university and to work with valuable faculty members.

Secondly, I would like to thank my office office mates and colleagues for their support and encouragement.

I owe thanks to my all family for their endless support and encouragement at every stage of my education. Finally, I would like to thank my dear wife for giving incredible support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Efficient utilization of the production process has become important for the companies due to competitive market conditions. It is important for companies to make an efficient production plan to survive and to continue their existence in this competition. In this thesis, we address two different problems coming from the printing industry: the label printing problem and a variant of the cover printing problem. In both problems, making a good production plan increases the efficiency of the printing process and reduces the environmental impact.

Printing industry uses various types of printing processes, each resulting in different types of waste. In the first part of the thesis (Chapter 2), we address the *Label Printing Problem* coming from the this industry. In various sectors, different types of labels are used. These include clothing, food, beverage industries, labels on various customer products and so on. In this problem, templates are created before the production process begins. Here, each template has fixed number of slots each of which is used to print a single type of label. In the next step, the number of required templates and the assignment of labels to the slots of these templates are determined. Then, the printing frequency of each template is determined in order to satisfy customer demand. The main goal is to produce as less waste as possible.

In the second part (Chapter 3), we study a variant of the cover printing problem that has applications in real-world manufacturing industries, where offset-printing technology is used to print customer-specific designs and standard designs on napkin pouches. Customer-specific designs are printed for customer orders, whereas standard designs are printed for make-to-stock. It is a strategical task to find the best assignment of a set of customer-specific designs for napkin pouches to

offset plates for print. In this problem, the number of plates used is not fixed and the assignment of napkin pouches to offset plates aims to minimize the number of plates used and the number of plate rotations. Different than the setting in Chapter 2, there are technological and organizational constraints that have to be taken into account when determining the production plan.

In this thesis, we first present mixed-integer mathematical formulations for both problems. We propose set-covering based heuristic approaches in order to find good solutions as the analyzed problems are NP-hard. We test the proposed heuristics on the real-life and randomly generated instances and compare its performance against the benchmark algorithms in the literature. We obtain much better results compared to the best known results in the literature.

CHAPTER II

A TWO-PHASE HEURISTIC ALGORITHM FOR THE LABEL PRINTING PROBLEM

In this chapter, we study the *Label Printing Problem* (LPP) which has applications in the printing industry. In LPP, the demand for a set of labels is satisfied by printing the labels using templates with multiple slots. Given a fixed number of templates, the decisions in LPP are determining (i) the assignment of labels to the slots of the templates (which we call *template designs*), and (ii) the number of prints made using each template design. The objective is to satisfy the demand with minimum waste. We consider two variants of LPP where (i) each label can be assigned to the slot(s) of a single template, and (ii) each label can be assigned to the slot(s) of multiple templates. To address LPP, we propose a novel sampling-based construct-improve heuristic where we first generate "good" template designs and then choose the ones to be used and determine the number of prints made through a set covering-type mathematical model. Then, we improve the solution using some improvement ideas that utilize a strengthened linear integer model for the problem. Using the instances from the literature, we show that the proposed heuristic provides better results compared to the benchmark algorithms. We also find optimal solutions for some of the instances from the literature using the strengthened linear integer model. With the help of the optimal solutions found we identify some problems in the previously reported results in a related study. Finally, we observe that the proposed heuristic approach not only provides better solutions but also runs in less amount of time compared to the benchmark algorithm on the large instances.

2.1 Introduction

Printing industry is one of the largest manufacturing industries. In this industry, various types of items such as newspapers, books, magazines, labels, business cards, brochures, newsletters, postcards, etc. are printed. In the printing industry, paper waste is one of the largest sources of inefficiency. One of the major environmental problems we are facing in these times is deforestation, and 14% of global wood harvest is used to make paper [1]. Hence, reducing waste in the printing industry has the potential to alleviate the environmental problems. Moreover, reducing waste not only contributes to environmental protection but also helps companies reduce costs.

There are various types of printing processes including lithographic, gravure, flexographic, digital, letterpress, screen and label printing each resulting in different types of waste. In this chapter, we focus on label printing. Labels are used in a wide range of different sectors including clothing, food, beverage industries and on various consumer products. In a typical label printing process, templates each of which has several slots on it are used to print labels. First, the number of templates needed is determined. Then, the label to be printed on each slot of a template is determined. Such an assignment of labels to the slots of a template is called a *template design*. Finally, considering the demand for each label, the printing frequency of each template is decided in order to satisfy the label requirements. Poor template designs cause excessive printed labels, which means waste, and hence additional cost for the companies.

In this chapter, we study the problem called the *Label Printing Problem* (LPP) [2, 3, 4] motivated by the practices in the label printing industry. In LPP, given a fixed number of templates with a certain number of slots on them, the goal is to determine the assignment of a set of labels to the slots of the templates and decide on the number of prints made using each template to satisfy the demand for the labels with minimum waste. We analyze two variants of LPP where each label can be assigned to the slot(s) of one template only in the first one and multiple

templates in the second one. The first variant is called the *particular* case of LPP, and the second one is called the *general* case [3]. We first provide strengthened mixed-integer linear models for both variants. We develop a novel heuristic algorithm composed of two phases: (i) construction, and (ii) improvement. In the construction phase, we form an initial feasible solution which is based on a novel idea of generating several template designs using some type of a sampling method and choosing among them by solving a set covering-type mathematical model. Then, in the improvement phase utilizing a restricted version of the mixed-integer linear model we try to improve the initial solution. We obtain much better results compared to a benchmark algorithm in the literature [3]. Moreover, the average run time of the proposed heuristic approach is less than that of the benchmark algorithm on the large instances.

In the proposed heuristic approach, we develop a two-phase solution framework. We modify the *Multi-Run Heuristic* by [5] to generate a set of "good" template designs in the first phase. However, the novelty of our solution approach comes from our framework rather than the approach used as a subroutine to generate initial template designs. Any other algorithm to generate initial template designs can be used as well. In the proposed heuristic approach, we use the novel idea of choosing among a set of "good" template designs with the help of a set covering-type mathematical model. This helps us to find a good combination of template designs in the initial solution. Moreover, we utilize the strengthened mixed-integer linear formulations in a smart way within an improvement procedure to use the power of an exact method and exploit improvement opportunities more effectively. We introduce several heuristic techniques to reduce the search space when solving the mixed-integer linear formulations within the improvement step. Finally, as it can be seen from the computational results, the innovative use of strengthened formulations within the improvement phase is the key to the success of the proposed heuristic approach.

Our main contributions in this study are as follows:

- We solve all of the particular case instances with up to 25 labels and some of the general case instances optimally using the strengthened mixed-integer linear models adapted from the model developed for CPP by [5].

- Using the optimal solutions found, we identify some incorrectly reported results in an earlier study.

- We propose a novel two-phase solution approach which utilizes the strengthened mixed-integer linear models and outperforms the benchmark algorithm.

The rest of the chapter is organized as follows. In Section 2.2, we review the related works in the literature. In Section 2.3, we provide a formal definition of the *Label Printing Problem* (LPP) and present an illustrative example to further explain the problem. Moreover, we present strengthened mixed-integer linear models for both variants of the problem. In Section 2.4, we present our two-phase solution approach for the general case of the problem. We discuss how we modify the proposed heuristic for the particular case of the problem in Section 2.5. In order to evaluate the performance of the proposed solution approach, we conduct computational experiments both on the instances from the literature and on some new instances and compare the results against the benchmark algorithm in Section 2.6. Concluding remarks are provided in Section 2.7.

2.2 Literature Review

Label Printing Problem (LPP) is first introduced by [4]. They discuss that additional cost is incurred if a label is assigned to multiple templates. Hence, they focus on the particular case of the problem where each label is assigned to one template only. They assume a smaller weight (0.9 of the regular waste) for the waste produced due to empty slots on a template and look for a solution with minimum total weighted waste. The authors first propose a nonlinear integer programming formulation. Then, making some observations about the structure of the optimal solution they develop a two-stage approach where the outer level

determines the labels assigned to each template and the inner level determines the template design and the number of prints to make for each template.

Following this initial study, [2] propose an immune based evolutionary algorithm to solve the particular case of LPP. [3] study both the general case and particular case of LPP. They propose a greedy randomized adaptive search procedure (GRASP) to address the general case and modify it for the particular case of the problem. They obtain the best known results for the LPP instances in the literature.

Related to LPP, some authors study another problem called the *Cover Printing Problem* (CPP). Unlike LPP, in CPP the number of templates used is another decision the decision maker has to make assuming a fixed cost per template. In CPP, the goal is to satisfy the demand for the labels with the objective of minimizing the total template and paper cost. [6] is the first to study CPP and propose a method that combines simulated annealing with linear programming. [7] develop two heuristic algorithms combining genetic algorithm with linear programming. The first one is a single objective genetic algorithm which looks for a solution with minimum total cost. The second one is a multi-objective genetic algorithm which finds a set of Pareto optimal solutions considering the two objectives of minimizing the number of templates and minimizing the number of prints. [8] propose a GRASP heuristic and obtain solutions better than [7]. [5] study CPP and develop an efficient linear integer formulation. They strengthen the proposed formulation with valid inequalities. Finally, they propose two heuristic algorithms one of which is a 2-approximation algorithm. [9] propose an ad hoc heuristic algorithm, whereas [3] implement the GRASP procedure developed for LPP to CPP. Recently, [10] combine simulated annealing and linear programming techniques and hybridize tabu search with the heuristic proposed by [9] to solve CPP. They improve 68% of the best known solutions.

[11] analyze a variant of CPP where there is a minimum and maximum number of prints that can be made using each template. Furthermore, they treat the

problem as a multi-objective problem where minimizing the number of templates has a higher priority. They propose three heuristic algorithms followed by a local search procedure to improve the initial solution found. Finally, [12] study another variant of CPP that has applications in the manufacturing of napkin pouches. They have to satisfy certain color and white-border constraints when determining the template designs. They develop a multi-pass matching-based savings heuristic to address the problem.

Compared to the related studies in the literature, our main contribution in this study is a novel two-phase solution framework which utilizes exact methods to find better solutions. We implement a sampling-based approach to find a good initial solution by forming candidate template designs and choosing among them by a set covering-type mathematical model. Moreover, we propose a smart improvement phase which solves the restricted versions of the strengthened mixed-integer models to identify improvement opportunities.

2.3 Problem Definition and Mathematical Models

In this section, we provide a formal definition of the *Label Printing Problem* (LPP) along with a simple example to explain the problem structure and present strengthened mixed-integer linear models for the two variants of LPP.

2.3.1 Problem Definition

In LPP, there are N labels each of which has a certain demand. We use d_i to denote the demand of label i. Without loss of generality, we assume that labels are ordered in a nonincreasing order of their demands ($d_1 \geq d_2 \geq \ldots \geq d_N$). For the ease of notation, we denote the set of labels by $\mathcal{N}$ ($:= \{1, 2, \ldots, N\}$). These labels are printed on a printing machine which prints the labels using a fixed number of templates each of which has a fixed number of slots on it. T is used to denote the number of templates, and S is used to denote the number of slots on a template. At most one label is assigned to each slot of a template. Each printed large sheet using a template results in one copy of the labels assigned to the slots

of the template. In LPP, the decisions are to determine (i) the assignment of labels to the slots of each template, and (ii) the number of prints made using each template with the objective of satisfying the demand for the labels with minimum total waste. For each label, amount produced beyond the demand is considered as waste. Moreover, if a slot of a template is left empty, it also results in a waste each time a print is made using that template. In the literature, related studies [3, 4] assume a smaller weight for the waste produced due to empty slots. However, the performance of the proposed approaches in the literature are evaluated based on a waste criterion calculated assuming the same weight for both types of waste [3]. Hence, we assume there is only one type of waste in our study.

To better explain the problem, we provide a simple example with 4 labels and 2 templates with 4 slots on each. Demand data is as follows: $d_1 = 210, d_2 = 60, d_3 = 30, d_4 = 20$. A feasible assignment of labels to the slots of the templates, and the number of prints (L_j) for each template is provided in Figure 1. In this solution, label 1 is assigned to 3 slots of the first template, and label 2 is assigned to the remaining slot of this template. For the second template, label 3 is assigned to 2 slots, and labels 1 and 4 are assigned one slot each. The number of prints made using templates 1 and 2 are 65 and 20, respectively. The total number of copies of label 1 produced is 215 ($= 3 \times 65 + 20$), and hence, the waste is 5 ($= 215 - 210$). Similarly, waste for labels 2, 3 and 4 are 5, 10 and 0, respectively. Hence, the total waste is 20. The total waste can also be calculated by subtracting the total demand from the total number of labels printed. In this example, the total number of labels printed is equal to 340 ($= 4 \times (65 + 20)$), and the total demand is 320. Hence, the total waste is 20. Since the total demand is constant, minimizing total waste is equivalent to minimizing the total number of prints made. For the rest of the chapter, we assume an objective of minimizing the total number of prints.

We analyze two variants of LPP. In the first one, each label is assigned to the slot(s) of one template only [2, 3, 4]. This is called the *particular* case of LPP. In the second one, each label can be assigned to the slot(s) of multiple templates.

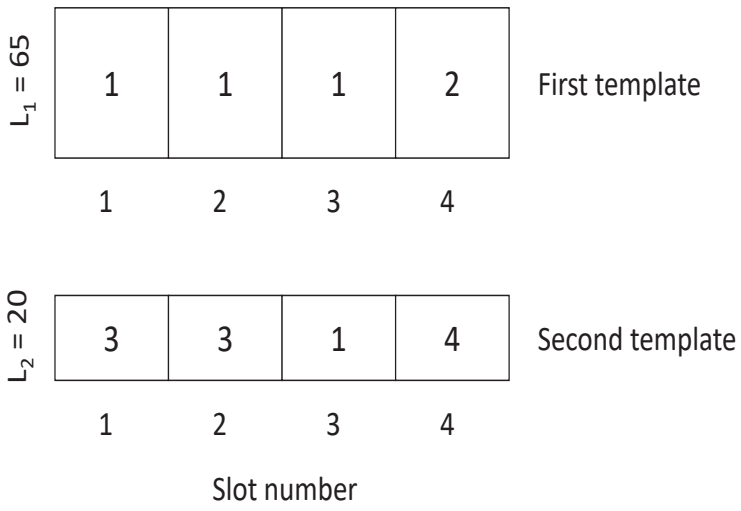

Figure 1: A feasible solution for the example.

This second variant is called the *general* case [3].

2.3.2 Mathematical Models

In the literature, [6] is the first to propose an integer linear model for CPP. Later, [5] developed a strengthened linear formulation with valid inequalities. Before discussing the proposed heuristic algorithm, we present strengthened mixed-integer linear models for both variants of LPP adapted from the model proposed for CPP [5]. We use these strengthened linear models to solve some of the instances to optimality. Moreover, we utilize it within the proposed heuristic algorithm.

We first provide a formulation for the general case and then modify it for the particular case of LPP. In the proposed model, we use $\mathcal{T}$ ($:= \{1, \ldots, T\}$) and $\mathcal{S}$ ($:= \{0, 1, \ldots, S\}$) to denote the set of templates and the set of possible number of slots each label is assigned to on a template. We define the following decision variables:

$$x_{ijk} = \begin{cases} 1, & \text{if label } i \text{ is assigned to } k \text{ slots of template } j \\ 0, & \text{otherwise.} \end{cases} \qquad i \in \mathcal{N}, j \in \mathcal{T}, k \in \mathcal{S}$$

q_{ijk} = amount of demand of item i satisfied by printing it on k slots of template j $\qquad i \in \mathcal{N}, j \in \mathcal{T}, k \in \mathcal{S}$

L_j = number of prints made using template j $\qquad j \in \mathcal{T}$

Using these decision variables, the waste minimization objective can be written as follows:

$$S \sum_{j \in \mathcal{T}} L_j - \sum_{i \in \mathcal{N}} d_i$$

Since the second term is constant, minimizing the total waste is equivalent to minimizing the total number of prints made using all the templates. The mixed-integer linear model (MIPG) proposed for the general case of LPP is as follows:

$$\text{MIPG: Min} \quad \sum_{j \in \mathcal{T}} L_j \tag{1}$$

s.t.

$$\sum_{j \in \mathcal{T}} \sum_{k \in \mathcal{S}} q_{ijk} \geq d_i \qquad i \in \mathcal{N} \tag{2}$$

$$\sum_{i \in \mathcal{N}} \sum_{k \in \mathcal{S}} k x_{ijk} \leq S \qquad j \in \mathcal{T} \tag{3}$$

$$q_{ijk} \leq k L_j \qquad i \in \mathcal{N}, j \in \mathcal{T}, k \in \mathcal{S} \tag{4}$$

$$q_{ijk} \leq M x_{ijk} \qquad i \in \mathcal{N}, j \in \mathcal{T}, k \in \mathcal{S} \tag{5}$$

$$\sum_{k \in \mathcal{S}} x_{ijk} = 1 \qquad i \in \mathcal{N}, j \in \mathcal{T} \tag{6}$$

$$L_j \geq L_{j+1} \qquad j \in \mathcal{T} \setminus \{T\} \tag{7}$$

$$\sum_{i \in \mathcal{N}} \sum_{k \in \mathcal{S} \setminus \{0\}} q_{ijk} \leq S L_j \qquad j \in \mathcal{T} \tag{8}$$

$$q_{ijk} \geq 0 \qquad i \in \mathcal{N}, j \in \mathcal{T}, k \in \mathcal{S} \tag{9}$$

$$L_j \in \mathbb{Z}_+ \qquad j \in \mathcal{T} \tag{10}$$

$$x_{ijk} \in \{0, 1\} \qquad i \in \mathcal{N}, j \in \mathcal{T}, k \in \mathcal{S} \tag{11}$$

In this model, the objective (1) is to minimize the total number of prints. Constraint (2) guarantees that the demand of each label is satisfied. Labels can be assigned to at most S slots on each template. This is enforced by constraint (3). Constraint (4) ensures that the total amount of label i produced using template j cannot be greater than the number of prints made using template j multiplied by the number of slots label i is assigned to. A template can be used to print label i only if label i is assigned to the slot(s) of that template. This is imposed by constraint (5). In this constraint, M is a large enough number. Constraint (6) helps determine the number of slots each label is assigned to on each template.

Constraint (7) sorts the templates according to the number of prints. Although this constraint is redundant, it is added to the model to eliminate the symmetry problem. Constraint (8) ensures that the total number of copies of labels produced using a template is limited by the number of prints made using that template multiplied by the number of slots. Similar to the previous constraint, this is actually a redundant constraint. However, it is added to the model to strengthen the formulation. Remaining constraints (9-11) are the sign and binary restrictions. Although q_{ijk}'s have to be integers in practice, we do not impose it in the model explicitly. Integrality of L_j's and d_i's would be enough to impose the integrality of q_{ijk}'s.

The above formulation can be modified to handle the particular case of LPP where each label can be assigned to the slot(s) of a single template only. We need an additional decision variable to limit the assignment of a label to one template only:

$$y_{ij} = \begin{cases} 1, & \text{if label } i \text{ is assigned to template } j \\ 0, & \text{otherwise.} \end{cases} \qquad i \in \mathcal{N}, j \in \mathcal{T}$$

The following constraints have to be added to the model MIPG to address the particular case:

$$\sum_{j \in \mathcal{T}} y_{ij} = 1 \qquad i \in \mathcal{N} \tag{12}$$

$$kx_{ijk} \leq Sy_{ij} \qquad i \in \mathcal{N}, j \in \mathcal{T}, k \in \mathcal{S} \tag{13}$$

$$y_{ij} \in \{0, 1\} \qquad i \in \mathcal{N}, j \in \mathcal{T} \tag{14}$$

Constraint (12) limits the assignment of a label to one template only. Constraint (13) determines the template each label is assigned to by relating x_{ijk} and y_{ij}. We denote the formulation for the particular case by MIPP.

2.3.3 Strengthening the Mathematical Models

We strengthen the proposed models MIPG and MIPP using the strengthening ideas proposed for CPP [5]. Since we use the mathematical models within the

proposed solution approach in the improvement phase, strengthening the mathematical models helps us improve the solution approach as well. We refer the reader to [5] for the details of these strengthening ideas. Simply, using a feasible solution for LPP, we (i) develop upper and lower bounds on the number of prints made using each template (L_j), and (ii) restrict the number of slots each label can be assigned to on each template. We use these strengthening ideas and the initial feasible solution found at the end of the construction phase to come up with the strengthened mathematical models.

Using a feasible solution with an objective function value of $\overline{Z}$, we obtain tighter lower (L_j^{LB}) and upper (L_j^{UB}) bounds on the number of prints made using template j and add the following valid inequalities to MIPG and MIPP.

$$L_j \geq L_j^{LB} \quad j \in \mathcal{T} \tag{15}$$

$$L_j \leq L_j^{UB} \quad j \in \mathcal{T} \tag{16}$$

$$\sum_{j \in \mathcal{T}} L_j \leq \overline{Z} \tag{17}$$

We first explain how upper and lower bounds on the number of prints made by each template are determined. Since the templates are ordered according to the number of prints made, we calculate the following upper bound for the number of prints made for each template in the optimal solution:

$$L_j^{UB} = \left\lfloor \frac{\overline{Z}}{j} \right\rfloor \qquad j \in \mathcal{T} \tag{18}$$

Next, we determine the first lower bound for the number of prints for each template. If a label is not assigned to any of the first $j-1$ templates, then it should be assigned to at least $\lceil \frac{d_i}{L_j} \rceil$ slots. This is due to the assumption that the templates are ordered in a nonincreasing order of their number of prints. We have at least $N - S(j-1)$ labels that are not assigned to any of the first $j-1$ templates. Moreover, since the demand values are ordered in a nonincreasing order, these labels should be assigned to at least $\sum_{i=S(j-1)+1}^{N} \lceil \frac{d_i}{L_j} \rceil$ slots. Excluding

the first $j-1$ templates, we have $S(T-j+1)$ slots left in total. Based on these observations, L_j should satisfy the following inequality:

$$\sum_{i=S(j-1)+1}^{N} \left\lceil \frac{d_i}{L_j} \right\rceil \leq S(T-j+1) \tag{19}$$

This gives us a lower bound (L_j^{UB}) on the L_j value, and this lower bound can be found by a simple search algorithm in the interval $[1, L_j^{UB}]$. Using this observation, we determine the lower bound for template j for $j \leq \lceil \frac{N}{S} \rceil$. For the remaining templates, the lower bounds are 0.

Finally, we update the upper bound L_j^{UB} values using the lower bounds determined above. We use L_{Extra} to denote the additional number of prints on top of the lower bound values determined. More specifically, the extra number of prints (L_{Extra}) is calculated as $L_{Extra} = \overline{Z} - \sum_{j \in \mathcal{T}} L_j^{LB}$. We use L_{Extra} to improve the upper bounds. Since the templates are ordered in a nonincreasing order of their number of prints, we we allocate L_{Extra} each template in Algorithm 1. With the help of this procedure, we aim to have tighter upper bounds.

Algorithm 1 Procedure for updating the upper bounds on the number of prints for each template

Input: Upper (L_j^{UB}) and lower (L_j^{LB}) bounds for the number of prints for each template and a feasible solution
Output: Updated upper bounds (L_j^{UB}) for the number of prints
$L_1^{UB} = \min\{L_1^{LB} + L_{Extra}, L_1^{UB}\}$
for $j = 2$ *to* T **do**
  $Total = L_{Extra}$
  for $t = j$ *to* 1 **do**
    $Total = Total + L_t^{LB}$
    if $\lfloor \frac{Total}{j-t+1} \rfloor \leq L_{t-1}^{LB}$ **then**
      $L_j^{UB} = \min\{\lfloor \frac{Total}{j-t+1} \rfloor, L_j^{UB}\}$
      Go to Step 12
    end if
  end for
end for

As a second strengthening step, we use L_j^{LB} and L_j^{UB} values to calculate the maximum number of slots label i can be assigned to on template j as follows:

$$S_{ij}^{UB} = \min\left\{ ST - \sum_{i' \in \mathcal{N} \setminus \{i\}} \left\lceil \frac{d_{i'}}{L_1^{UB}} \right\rceil, \left\lceil \frac{d_i}{\max\{L_j^{LB}, 1\}} \right\rceil, S \right\} \tag{20}$$

We use S_{ij}^{UB} values to limit the number of x_{ijk} variables defined which reduces the size of the problem. The lower bound on the number of prints made by each template provides an upper bound on the number of slots each label can be assigned to on that template. That is, label i can be assigned to at most $\left\lceil \frac{d_i}{L_j^{LB}} \right\rceil$ labels on template j. We consider the maximum of L_j^{LB} and 1 ($\max\{L_j^{LB},1\}$) to take into account the templates with a lower bound of 0 for the number of prints made. On the other hand, the upper bound on the number of prints made by the first template (the one with the highest number of prints) imposes a limit on the minimum number of slots each label is assigned to. Hence, $\sum_{i' \in \mathcal{N} \setminus \{i\}} \left\lceil \frac{d_{i'}}{L_1^{UB}} \right\rceil$ provides the minimum number of slots labels (other than label i) should be assigned to in an optimal solution. Subtracting this expression from the total number of slots (ST) gives us an upper bound on the number of slots label i can be assigned to. Combining these two upper bounds on the number of slots label i can be assigned to with the number of slots S on a template results in S_{ij}^{UB} value calculated using Equation (20).

To sum up, using a feasible solution with an objective function value $\overline{Z}$ we calculate upper and lower bounds for L_j variables and add constraints (15)-(17) and replace the set $\mathcal{S}$ by $\mathcal{S}_{ij}^{UB}$ ($:= \{0, 1, \ldots, S_{ij}^{UB}\}$) in MIPG and MIPP. We denote the strengthened versions of these two formulations by MIPG$^{\mathrm{S}}$ and MIPP$^{\mathrm{S}}$.

2.4 Proposed Solution Approach: General Case

We first explain the solution approach proposed for the general case of LPP and then discuss how it is modified to address the particular case. The proposed heuristic algorithm has two phases: (i) construction phase, and (ii) improvement phase. We call this heuristic algorithm the *Construct-Improve Heuristic - General* ($\mathrm{CIH_G}$). Once the labels assigned to the slots of a template are determined, we call such an assignment a *template design*. A template design specifies the number of slots each label is assigned to on that template. In the construction phase, we first generate several "good" template designs using some type of a sampling

method, and then we choose the best T template designs among the generated ones and determine the number of prints for each by solving a variant of the *Set Covering Problem* [13, 14]. In the improvement phase, starting from the solution constructed in the first phase we implement an improvement idea that utilizes a restricted version of MIPG$^{\text{S}}$ to find a better solution. Starting from the solution found in the construction phase, the main idea in the improvement phase is to decrease/increase the number of slots a label is assigned to on a template in order to obtain a better solution.

2.4.1 Construction Phase

In the construction phase, we generate several template designs without specifying the number of prints for each template. We modify the *Multi-Run Heuristic* (MRH) [5] to generate the candidate template designs and choose the best set of template designs among the generated ones by solving a set covering model. In order to generate the candidate template designs, we propose a novel sampling-based approach. We choose a subset $\mathcal{N}'$ of labels randomly and estimate the number of templates to be used to satisfy the demand of these labels as follows:

$$T' = \left\lceil \frac{\sum_{i \in \mathcal{N}'} d_i}{\sum_{i \in \mathcal{N}} d_i} T \right\rceil \tag{21}$$

Then, we run the *Modified Multi-Run Heuristic* (MMRH) to obtain a solution with minimum waste for the labels in $\mathcal{N}'$ using T' templates. The steps of MMRH are provided in Algorithm 2. The main idea here is to assign one slot to each label, and then increase the number of slots assigned to the label with the highest demand-to-assigned slots ratio by one until all the slots are assigned to a label (Steps 7 - 12). Then, we determine the number of prints for the template with the highest number of prints (Step 13). In MMRH, after determining the current template design using the initial assignment of labels to the slots (Steps 17 - 20), we check if the number of prints to be made using this template can be increased without producing waste (Steps 21 - 26). If it can be increased, we update the number of prints and the label-slot assignment for the current template. The main

Algorithm 2 *Modified Multi-Run Heuristic* (MMRH)

1: **Input:** A subset of $\mathcal{N}'$ of labels
2: **Output:** A set of template designs
3: Initialize the integer value: $\mathrm{l} = 1$
4: Initialize the demand values: $d'_i = d_i$, $\forall i \in \mathcal{N}'$
5: Estimate the number of templates T' using Equation (21)
6: **while** $\max_{i\in\mathcal{N}'}\{d'_i\} > 0$ **do**
7: Initialize the number of slots label i is assigned to template l: a_{il}=0, $\forall i \in \mathcal{N}'$
8: Initialize the number of slots each label is assigned to: s_i=1, $\forall i \in \mathcal{N}'$
9: **while** $\sum_{i\in\mathcal{N}'} s_i < ST'$ **do**
10: $j = \text{argmax}_{i\in\mathcal{N}'}\{\lceil \frac{d'_i}{s_i} \rceil\}$
11: $s_j = s_j + 1$
12: **end while**
13: Determine the maximum number of prints for a template: $L_{max} = \max_{i\in\mathcal{N}'}\{\lceil \frac{d'_i}{s_i} \rceil\}$
14: $ImprovementFlag = 1$
15: **while** $ImprovementFlag = 1$ **do**
16: Initialize the temporary demand and temporary assigned slots: $d''_i = d'_i$, $s'_i = 0$, $a_{il} = 0$, $\forall i \in \mathcal{N}'$
17: **while** $\sum_{i\in\mathcal{N}'} s'_i < S$ **do**
18: $j = \text{argmax}_{i\in\mathcal{N}'}\{d''_i\}$
19: Assign label j to one slot and update its demand: $s'_j = s'_j + 1$, $d''_j = d''_j - \min\{d''_j, L_{max}\}$
20: **end while**
21: **if** $L_{max} < \min_{i\in\mathcal{N}', s'_i\neq 0}\{\lceil \frac{d'_i}{s'_i} \rceil\}$ **then**
22: Update L_{max} considering the current assignment: $L_{max} = \min_{i\in\mathcal{N}', s'_i\neq 0}\{\lceil \frac{d'_i}{s'_i} \rceil\}$
23: $ImprovementFlag = 1$
24: **else**
25: $ImprovementFlag = 0$
26: **end if**
27: **end while**
28: Fix the template design for this template using s'_i values
29: Update the demand, a_{il} and the number of remaining templates: $a_{il} = s'_i, T' = T' - 1$, $d'_i = \max\{d'_i - s'_i L_{max}, 0\}$, $\forall i \in \mathcal{N}'$
30: **if** $\max_{i\in\mathcal{N}'}\{d'_i\} > 0$ **then**
31: Update l and the set of labels: $l = l + 1$, $\mathcal{N}' = \{i \in \mathcal{N}' | d'_i \neq 0\}$
32: **end if**
33: **end while**

difference between the original version of MRH and MMRH are Steps 21 - 26. This modification is implemented to increase the utilization of a template design. If the current template design is not updated, after fixing the current template design (Step 28), we update the remaining demand (Step 29) and continue until all T' template designs are determined.

After running MMRH, we only store the template designs found for each random subset $\mathcal{N}'$. We run MMRH multiple times (each time with a randomly generated subset $\mathcal{N}'$) to generate several candidate template designs. After generating the template designs, we solve a set covering-type mathematical model to

select the best T template designs among the generated ones and determine the number of prints for each selected template design. In this model, our objective is to minimize the number of prints while satisfying the demand for each label. We use Γ to represent the set of candidate template designs generated using MMRH, and a_{il} to represent the number of slots label i is assigned to in template design l ($\in \Gamma$). We have two types of decisions in the proposed model: (i) which template designs to use, and (ii) how many prints to make using each selected template design. The following decision variables are used in the mathematical model:

$$u_l = \begin{cases} 1, & \text{if template design } l \text{ is chosen} \\ 0, & \text{otherwise.} \end{cases} \qquad l \in \Gamma$$

$$K_l = \text{number of prints made using template design } l \qquad l \in \Gamma$$

The proposed set covering-type mathematical model is as follows:

$$\text{SCP-G: Min} \quad \sum_{l \in \Gamma} K_l \tag{22}$$

s.t.

$$\sum_{l \in \Gamma} K_l a_{il} \geq d_i \qquad i \in \mathcal{N} \tag{23}$$

$$\sum_{l \in \Gamma} u_l \leq T \tag{24}$$

$$K_l \leq M u_l \qquad l \in \Gamma \tag{25}$$

$$K_l \geq 0 - integer \qquad l \in \Gamma \tag{26}$$

$$u_l \in \{0, 1\} \qquad l \in \Gamma \tag{27}$$

In this model, the objective (22) is to minimize the total number of prints made. Constraint (23) ensures that the demand is satisfied for each label. The number of used/selected template designs is limited by T in constraint (24). A template design can only be used to make prints if it is selected. This is enforced by constraint (25). Remaining constraints are the binary and integrality restrictions for the decision variables.

As the number of template designs generated increases, solving SCP-G can be challenging. Hence, we propose processing the generated template designs

in batches. That is, we generate a set of template designs running MMRH for several subsets of $\mathcal{N}$. Then, we solve SCP-G considering these template designs only and store the selected templates in a list (say, *SelectedList*). Then, we generate another set of template designs using MMRH and implement the same steps again. At the end of the construction phase, we solve SCP-G one more time considering the template designs in *SelectedList* and determine a feasible solution for LPP. The steps of the construction phase are summarized in Algorithm 3.

Algorithm 3 Construction Phase

Input: An LPP instance
Output: A feasible solution
$SelectedList = \emptyset$
for $IterationCount1 = 1$ *to* $IterationLimit1$ **do**
    $GeneratedList = \emptyset$
    for $IterationCount2 = 1$ *to* $IterationLimit2$ **do**
        Generate an integer number (say R) randomly between 1 and $|\mathcal{N}|$
        Select R elements from the label set $\mathcal{N}$ randomly and place them into set $\mathcal{N}'$
        Run MMRH and add the template designs in the solution to *GeneratedList*
    end for
    Solve SCP-G with the template designs in *GeneratedList* and add the template designs selected to *SelectedList*
end for
Solve SCP-G with the template designs in *SelectedList*

2.4.2 Improvement Phase

In the improvement phase, given the initial solution found at the end of the construction phase we aim to improve the solution by changing the number of slots a label is assigned to. When checking if there is an improvement or not, we solve a restricted version of the mathematical model MIPG$^{\mathrm{S}}$. By doing this, we aim to use the power of an exact method when searching for an improvement.

We use $\overline{\Gamma}$ to denote the set of template designs selected at the end of the construction phase. Note that $|\overline{\Gamma}| = T$. Then, for certain template-label pairs we allow decreasing and increasing the number of slots each label is assigned to (by

up to some λ) and determine the best solution by solving a mathematical model. We first sort the template designs in $\overline{\Gamma}$ in a nonincreasing order according to their number of prints found after solving SCP-G since the template designs are sorted in a nonincreasing order of their number of prints in the strengthened model (see Equation (7)). Here, a'_{ij} denotes the number of slots label i is assigned to in the template with the jth highest number of prints made. We denote the maximum and minimum allowable number of slots label i can be assigned to in template j by max_{ij} and min_{ij}, respectively. Using max_{ij} and min_{ij}, we define the set of allowable values for the number of slots label i can be assigned to in template j as $\mathcal{S}_{ij} = \{min_{ij}, \ldots, max_{ij}\}$. In model MIPG$^{\text{S}}$, we replace the set $\mathcal{S}_{ij}^{UB}$ from which k is chosen by $\mathcal{S}_{ij}$ and solve this restricted version of the proposed model at each iteration of the improvement phase. We denote this restricted model by MIPG$^{\text{S}}$-R. We continue iterations until there is no more improvement in the solution found.

We implement this improvement idea in two stages. In the first stage, we limit the number of templates whose designs can be modified by two and look for a better solution until there is no improvement. That is, we allow decreasing/increasing the number of slots each label is assigned to for two templates only at a time. Then, after verifying that there is no more potential improvement with two templates, we allow changing the number of slots some selected labels are assigned to in all the templates in the second stage. If an improvement is achieved at the end of the second stage, we go back to the first stage. Otherwise, the procedure stops.

The outline of the first stage of the improvement phase is presented in Algorithm 4. In the first stage, we use the initial solution found at the end of the construction phase to strengthen model MIPG. Each pair of templates are considered for a potential improvement, and improvement efforts continue until no improvement is observed. For a given a pair of templates in the current solution, we allow increasing and decreasing the number of slots each label is assigned to by some λ (Steps 6 - 12) and determine the best assignment of labels to the slots by solving MIPG$^{\text{S}}$-R (Step 13). We store the best solution found if it is better than

the current solution (Steps 14 - 16). At the end, we update the current solution if there is an improvement (Steps 18 - 23).

Algorithm 4 First Stage of the Improvement Phase

1: **Input:** A feasible solution for LPP with total number of prints $L_{current}$ (a'_{ij} denotes the number of slots label i is assigned to on template j)
2: **Output:** A (potentially better) feasible solution
3: $L_{new} = L_{current}, ImprovementFlag = 1$
4: **while** $ImprovementFlag = 1$ **do**
5: **for** $(j_1, j_2) \in \mathcal{T}^2$ **do**
6: **for** $i \in \mathcal{N}$ **do**
7: $min_{ij} = max_{ij} = a'_{ij}, \forall j \in \mathcal{T}$
8: **if** $a'_{ij_1} + a'_{ij_2} > 0$ **then**
9: $min_{ij} = \max\{0, a'_{ij} - \lambda\}, max_{ij} = \min\{S, a'_{ij} + \lambda\}, \forall j \in \{j_1, j_2\}$
10: **end if**
11: **end for**
12: $\mathcal{S}_{ij} = \{min_{ij}, \ldots, max_{ij}\}, \forall i \in \mathcal{N}, j \in \mathcal{T}$
13: Solve MIPG$^{\mathrm{S}}$-R (with optimal solution: $x^*_{ijk}, L^*_j, q^*_{ijk}$)
14: **if** $\sum_{j \in \mathcal{T}} L^*_j < L_{new}$ **then**
15: Store the new solution: $L_{new} = \sum_{j \in \mathcal{T}} L^*_j$; $a^{new}_{ij} = \sum_{k \in \mathcal{S}_{ij}} k x^*_{ijk}, \forall i \in \mathcal{N}, j \in \mathcal{T}$
16: **end if**
17: **end for**
18: **if** $L_{new} < L_{current}$ **then**
19: Update the best solution: $L_{current} = L_{new}; a'_{ij} = a^{new}_{ij}, \forall i \in \mathcal{N}, j \in \mathcal{T}$
20: $ImprovementFlag = 1$
21: **else**
22: $ImprovementFlag = 0$
23: **end if**
24: **end while**

In the second stage of the improvement phase, we choose a subset of labels and allow changing the number of slots these labels are assigned to for all the templates. The second stage of the improvement phase is provided in Algorithm 5. In the second stage, we use the best solution found at the end of the first stage to strengthen model MIPG. We first randomly choose a subset of labels (Step

6), and then allow increasing and decreasing the number of slots these labels are assigned to in all the templates (Steps 7 - 14). We again determine the best solution by solving MIPG$^{\mathrm{S}}$-R (Step 15). We store the best solution found during the improvement step (Steps 16 - 18) and then update the current solution if there is an improvement when the iteration limit ($IterationLimit3$) is reached (Steps 20 - 25). We continue iterations until there is no more improvement.

Algorithm 5 Second Stage of the Improvement Phase

1: **Input:** A feasible solution for LPP with total number of prints $L_{current}$ (a'_{ij} denotes the number of slots label i is assigned to in template j)
2: **Output:** A (potentially better) feasible solution
3: $L_{new} = L_{current}, ImprovementFlag = 1$
4: **while** $ImprovementFlag = 1$ **do**
5: **for** $IterationCount3 = 1$ *to* $IterationLimit3$ **do**
6: Choose a subset ($\mathcal{N}'$) of the label set $\mathcal{N}$ randomly
7: **for** $i \in \mathcal{N}$ **do**
8: **if** $i \in \mathcal{N}'$ **then**
9: $min_{ij} = \max\{0, a'_{ij} - \lambda\}, max_{ij} = \min\{S, a'_{ij} + \lambda\}, \forall j \in \mathcal{T}$
10: **else**
11: $min_{ij} = max_{ij} = a'_{ij}, \forall j \in \mathcal{T}$
12: **end if**
13: **end for**
14: $\mathcal{S}_{ij} = \{min_{ij}, \ldots, max_{ij}\}, \forall i \in \mathcal{N}, j \in \mathcal{T}$
15: Solve MIPG$^{\mathrm{S}}$-R (with optimal solution: $x^*_{ijk}, L^*_j, q^*_{ijk}$)
16: **if** $\sum_{j \in \mathcal{T}} L^*_j < L_{new}$ **then**
17: Store the new solution: $L_{new} = \sum_{j \in \mathcal{T}} L^*_j$; $a^{new}_{ij} = \sum_{k \in \mathcal{S}_{ij}} k x^*_{ijk}, \forall i \in \mathcal{N}, j \in \mathcal{T}$
18: **end if**
19: **end for**
20: **if** $L_{new} < L_{current}$ **then**
21: Update the best solution: $L_{current} = L_{new}; a'_{ij} = a^{new}_{ij}, \forall i \in \mathcal{N}, j \in \mathcal{T}$
22: $ImprovementFlag = 1$
23: **else**
24: $ImprovementFlag = 0$
25: **end if**
26: **end while**

Finally, the overall framework of the proposed heuristic approach *Construct-Improve Heuristic - General* (CIH_G) is presented in Algorithm 6. In summary, we first generate template designs (using Algorithm 2) and then solve a series of set covering problems to determine the initial solution (Step 3). Then, we improve the solution by changing the label-slot assignments for two templates only at a time (Step 5) and then changing the label-slot assignment of a subset of labels (Step 6).

Algorithm 6 *Construct-Improve Heuristic - General* (CIH_G)

1: **Input:** An LPP instance
2: **Output:** A feasible solution
3: Run Algorithm 3
4: **repeat**
5: Run Algorithm 4
6: Run Algorithm 5
7: **until** No improvement is observed

2.5 Modifying the Solution Approach for the Particular Case

In this section, we explain how the solution approach proposed for the general case of LLP is modified to address the particular case. We call this version of the algorithm the *Construct-Improve Heuristic - Particular* (CIH_P). In the construction phase, we still use MMRH to generate candidate template designs. However, we store both the template design and the number of prints made for each template design. Each label can be assigned to the slot(s) of a single template only in the particular case. Hence, once a template design is determined, the number of prints required if this template design is used in the production is also determined. More specifically, the number of prints required is equal to the highest demand-to-assigned slots ratio among all the labels assigned to this template. For each generated template design l in Γ, we use $\overline{K}_l$ to denote the number of prints required.

Next, we modify the set covering-type mathematical model to guarantee that

each label is assigned to the slot(s) of one template only. In the particular case, we only decide if a template design is selected or not. The number of prints for each template is a parameter as discussed above. We use the same decision variable u_l for each $l \in \Gamma$. Γ_i denotes the set of generated template designs that has some slots assigned to label i. The modified version of the set covering model is as follows:

$$\text{SCP-P: Min} \quad \sum_{l \in \Gamma} \overline{K}_l u_l \tag{28}$$

s.t.

$$\sum_{l \in \Gamma} \overline{K}_l a_{il} u_l \geq d_i \qquad i \in \mathcal{N} \tag{29}$$

$$\sum_{l \in \Gamma} u_l \leq T \tag{30}$$

$$\sum_{l \in \Gamma_i} u_l = 1 \qquad i \in \mathcal{N} \tag{31}$$

$$u_l \in \{0, 1\} \qquad l \in \Gamma \tag{32}$$

In addition to K_l being a parameter for each template design, another difference between SCP-G and SCP-P is constraint (31) which limits the number of templates each label is assigned to by one.

The outline of the construction phase is quite similar. In Steps 11 and 13 of Algorithm 3, we solve SCP-P instead of SCP-G. Finally, in both stages of the improvement phase, instead of solving MIPG$^{\text{S}}$-R, we solve a restricted version of MIPP$^{\text{S}}$ (denoted by MIPP$^{\text{S}}$-R) where the set $\mathcal{S}_{ij}^{UB}$ which includes the number of slots a label can be assigned to in each template is replaced by $\mathcal{S}_{ij}$ as explained above. Furthermore, the lower bound (L_j^{LB}) is set to 1 for template j for $j > \lceil \frac{N}{S} \rceil$, and then Algorithm 1 is implemented to determine the updated upper bounds to be used in the strengthened mathematical model.

2.6 Computational Study

We perform computational experiments to evaluate the performance of the proposed heuristic approach. All the computational experiments are carried out on a 64-bit Windows Server with two 2.4 Ghz Intel Xeon CPU's and 24 GB RAM.

Algorithms are implemented using JAVA and CPLEX with 12.6.0 Concert Technology.

The proposed solution approach has several parameters such as $IterationLimit1$, $IterationLimit2$, $IterationLimit3$ and λ. A well-designed parameter calibration can further improve the performance of the proposed solution approach. However, we did not calibrate the values of the parameters used in the proposed solution approach, but followed a simple approach based on a set of preliminary results on the real-life and random instances provided by [3]. We consider the trade-off between the solution time and quality while determining the values of these parameters.

Increasing the number of template designs generated in each batch (i.e., increasing $IterationLimit2$) and the number of batches solved ($IterationLimit1$) has the potential to improve the performance of the proposed solution approach. However, it increases the number and size of the set covering-type mathematical models to be solved as well. We observe that if $IterationLimit2$ is greater than 30, the solution time increases significantly. Thus, we set $IterationLimit2$ to 30. For smaller values of $IterationLimit2$, we cannot obtain a promising combination of template designs in the optimal solution of SCP-G. We set $IterationLimit1$ to 10. Preliminary tests with $IterationLimit1$ values of 20 and 30 doubled and tripled the solution time of the construction phase without much improvement in the initial solution. The iteration limit in the second stage of the improvement phase ($IterationLimit3$) is set to 100. An improvement check is made at every 100 iterations. Then, the algorithm goes back to the first stage of the improvement phase if an improvement is observed after 100 iterations. Otherwise, the improvement phase is terminated. Tests with an $IterationLimit3$ of 50 resulted in an early termination of the algorithm with a worse solution. Increasing $IterationLimit3$ to 200 did not bring much improvement over the solutions obtained with an $IterationLimit3$ value of 100.

After deciding on the values of $IterationLimit1$, $IterationLimit2$, and $Iteration\ Limit3$, we performed a more detailed study to determine the value of λ which

determines the maximum amount of decrease/increase in the number of slots assigned to a label in the improvement phase. The higher the value of λ, the higher the chances of finding a better solution. On the other hand, choosing a larger λ value makes the integer models MIPG$^{\mathrm{S}}$-R and MIPP$^{\mathrm{S}}$-R more challenging. When solving MIPG$^{\mathrm{S}}$-R or MIPP$^{\mathrm{S}}$-R in the first stage of the improvement phase, we set a time limit of 100 seconds. In the second stage, we decrease the time limit to 20 seconds and report the best solution found within the prespecified time limit. For the general case, we conducted experiments with λ values of 1, 2 and 3, and obtained average waste values of 2.199%, 2.058% and 2.157%, respectively on the second set of instances tested. Hence, when implementing CIH$_{\mathrm{G}}$, we set the values of the parameters as follows: $IterationLimit1 = 10$, $IterationLimit2 = 30$, $IterationLimit3 = 100$, and $\lambda = 2$.

For the particular case, the number of prints made is not a decision variable any more in the set covering model which makes SCP-P easier to solve compared to SCP-G. Hence, in CIH$_{\mathrm{P}}$ we increase $IterationLimit2$ to generate a larger set of template designs and find a better solution. Moreover, MIPP$^{\mathrm{S}}$-R is easier to solve than MIPG$^{\mathrm{S}}$-R, and hence, we increase λ to 5 in the improvement phase of CIH$_{\mathrm{P}}$.

We test the performance of the proposed solution approach on three sets of instances. Firstly, we use the instances used in an earlier study by [3]. There are two real-life instances with (i) 18 labels and templates with 14 slots, and (ii) 22 labels and templates with 15 slots. Each of these instances are solved with 3, 4 and 5 templates. Moreover, there is a quite large instance with 100 labels and 8 templates each of which with 25 slots proposed by [3]. The second set of instances are the randomly generated instances which are first used by [3]. For the number of labels in the randomly generated instances, four different settings are considered: 10, 15, 20, and 25. Five instances are generated for each setting. The demand of each label is generated randomly between 1,000 and 10,000. Each instance is solved assuming 3, 4 and 5 templates

with 10 and 20 slots. Finally, the third set of instances includes 10 large instances with 100 labels and 25-slot templates. These instances are originally proposed for CPP [9] and are available at http://www.matcuer.unam.mx/ davidr/-coverprinting/Datasets.html. We solve each of these large instances with 6, 8 and 10 templates. Hence, we have 30 instances in total. We provide these instances and the best solutions obtained by the proposed solution approach at https://faculty.ozyegin.edu.tr/aliekici/researchdata/.

In order to evaluate the solution quality, we use the waste percentage calculated as the percentage of the total production using the following formula:

$$\frac{S \sum_{j \in \mathcal{T}} L_j - \sum_{i \in \mathcal{N}} d_i}{S \sum_{j \in \mathcal{T}} L_j} \tag{33}$$

For the first two sets of instances, we compare our results with the ones presented in [3]. They develop a GRASP algorithm to solve both general and particular cases. They denote the version for the general case by GRASP_m and for the particular case by GRASP_s. In the article, the authors say that waste is calculated as a percentage of the total demand (where the denominator in Equation (33) is replaced by $\sum_{i \in \mathcal{N}} d_i$) when presenting the results. However, based on our personal communication with the authors they said that the waste is calculated as a percentage of the total production (similar to Equation (33)) in Table 3 of [3]. Hence, in order to make a consistent comparison with the results presented in [3] we calculate the waste as a percentage of the total production both for our solutions and their solutions. We do not have the results of the algorithms by [3] on the third set of instances with 100 labels. In order to make a comparison on these instances, we implement their algorithm to the best of our understanding. We denote our implementation of their GRASP algorithm by $\text{GRASP}_\text{m}^{\text{our}}$ and $\text{GRASP}_\text{s}^{\text{our}}$ for the general and particular cases, respectively. We test $\text{GRASP}_\text{m}^{\text{our}}$ and $\text{GRASP}_\text{s}^{\text{our}}$ on the first two sets of instances as well to show the consistency between our results and the ones by [3]. Following the implementation by [3], we take 10 independent runs of $\text{GRASP}_\text{s}^{\text{our}}$ and $\text{GRASP}_\text{m}^{\text{our}}$ on the first two sets of

instances and report the best solution found. [3] increased the number of independent runs to 20 for the instance with 100 labels. We also increase the number of runs to 20 for the large instances with 100 labels.

Using the proposed mathematical model MIPP$^{\mathrm{S}}$, we were able to solve all of the instances for the particular case (except the large instances with 100 labels) to optimality. However, for the general case we were able to solve only some of the instances to optimality, and the optimal solutions for these general case instances highlighted a problem with the results reported in [3]. For the instances with 25 labels and 3 templates with 10 slots, the reported waste percentages are lower than that of the optimal solutions. After our communication with the authors, they also agreed that there is a problem with the results reported and they sent us the new results after rerunning the GRASP$_{\mathrm{m}}$ for the general case. Therefore, we use the new results when comparing against our results.

We present the results for the first set of instances in Tables 1 and 2. The results on the randomly generated instances [3] are provided in Tables 3 and 4. Since five instances are generated for each (N, S, T) tuple in Tables 3 and 4, the numbers in each row of these two tables are averages of five instances. Finally, the results on the large instances with 100 labels are provided in Tables 5 and 6.

In all the tables, the first three columns represent the number of labels (N), number of slots on each template (S) and number of templates (T). In the following four columns under "CIH$_{\mathrm{P}}$" in Tables 1, 3 and 5, we present the results in each phase of the proposed algorithm for the particular case. In Tables 2, 4 and 6, under "CIH$_{\mathrm{G}}$", we present the same results for the general case. Under columns "CIH$_{\mathrm{P}}$" and "CIH$_{\mathrm{G}}$", we provide the detailed waste percentage and run time (in seconds) for each phase of the proposed heuristic approach in order to show how different phases of the algorithm affect the final solution. We present the results by [3] under "GRASP$_{\mathrm{s}}$" and "GRASP$_{\mathrm{m}}$" columns. The results under "GRASP$_{\mathrm{m}}$" column are the new results provided the authors, whereas the results for the particular case (under "GRASP$_{\mathrm{s}}$" column) are taken from [3]. We

present the results obtained by our implementation of GRASP under columns "$\text{GRASP}_\text{s}^{\text{our}}$" and "$\text{GRASP}_\text{m}^{\text{our}}$" columns for the particular and general cases, respectively. The waste percentage in the optimal solution and the run time (in seconds) of the mathematical model used to find the optimal solution are provided under column "Optimal". These optimal solutions are obtained by solving MIPP^S and MIPG^S. We used the best solution found with the heuristic algorithm ($\overline{Z}$) to strengthen the mathematical models. Although we solve all of the particular case instances to optimality (except the large instances with 100 labels), we can solve only few of the general case instances.

In Table 1, for the particular case both CIH_P and GRASP_s find optimal solutions for the real-life instances (with 18 and 22 labels). This is quite expected since these instances are relatively small, and the setting of the particular case makes the problem even easier. However, for the large instance with 100 labels, CIH_P outperforms GRASP_s by more than a 5% waste margin. Because of the size of the instance, we cannot find the optimal solution. For the instances with 18 and 22 labels, it takes around 3 minutes in total for CIH_P to find the final solution on average. However, we observe that the two stages of the improvement phase do not provide any improvement. The initial solution identified at the end of the construction phase in around 2 minutes is already optimal. GRASP_s finds a solution in less than 5 seconds [3]. For the large instance with 100 labels, it took CIH_P around an hour to find the final solution. The improvement phase reduces the waste percentage by around 1%, and the initial solution found (in around 25 minutes) at the end of the construction phase produces around 4% less waste compared to the solution found by GRASP_s. Finally, our implementation of GRASP_s ($\text{GRASP}_\text{s}^{\text{our}}$) also finds the optimal solution for the instances with 18 and 22 labels. However, for the large instance with 100 labels, it provides a better solution with around 1.5% lower waste compared to GRASP_s. We implemented MIPP^S using CPLEX Concert Technology to find the optimal solution for the instances with 18 and 22 labels. However, for the large instance with 100 labels,

optimal solution cannot be found.

Table 1: Comparison of the particular case results for the real-life instances and a large instance

Instance			CIH_P				$GRASP_s$	$GRASP_s^{our}$	Optimal	
			Construction		Improvement					
N	S	T	Waste	Time	Waste	Time	Waste	Waste	Waste	Time
18	14	3	5.854%	29	5.854%	21	5.854%	5.854%	5.854%	2
18	14	4	3.243%	82	3.243%	29	3.243%	3.243%	3.243%	31
18	14	5	1.373%	201	1.373%	87	1.373%	1.373%	1.373%	206
22	15	3	7.538%	44	7.538%	37	7.538%	7.538%	7.538%	2
22	15	4	3.458%	142	3.458%	83	3.458%	3.458%	3.458%	319
22	15	5	1.941%	221	1.941%	119	1.941%	1.941%	1.941%	20955
100	25	8	4.270%	1436	3.486%	1832	8.630%	7.122%	-	-

When we look at the general case results in Table 2, CIH_G improves the best known solutions for 5 out of 6 real-life instances by 0.4% on average. For the remaining one instance, both CIH_G and $GRASP_m$ find solutions with the same amount of waste. For the two instances, which are solved to optimality, CIH_G also finds an optimal solution. Without the improvement phase, $GRASP_m$ outperforms CIH_G by around 0.3% margin. The average run time of CIH_G for these instances is around 4 minutes whereas $GRASP_m$ can find a solution in less than 10 seconds. For the large instance with 100 labels, CIH_G improves the best solution by around 0.44%. However, it takes the improvement phase around 2.5 hours to improve the initial solution, and the initial solution is 0.26% worse than the solution found by $GRASP_m$. As reported by the authors, it takes 310 seconds (per run) for $GRASP_m$ to find a solution for this instance with 100 labels. In total, it makes around 6200 seconds for 20 independent runs. $GRASP_m^{our}$ finds the same solution as $GRASP_m$ for the instances with 18 and 22 labels. For the large instance with 100 labels, $GRASP_m^{our}$ provides a solution with 0.056% higher waste compared to $GRASP_m$. The total run time of $GRASP_m^{our}$ is around 8000 seconds, which is also consistent with the results reported by [3].

We find an optimal solution for two of these general case instances. The first one is obtained by solving MIPG^{S} to optimality, and it takes around 80 minutes to find the optimal solution. For the second instance (marked with "*"), MIPG^{S} cannot find an optimal solution after 24 hours. However, since the best lower bound provided by CPLEX at the beginning is equal to the solution found by CIH$_{G}$, we conclude that it is optimal.

Table 2: Comparison of the general case results for the real-life instances and a large instance

Instance			**CIH$_{G}$**				**GRASP$_{m}$**	**GRASP$_{m}^{our}$**	**Optimal**	
			Construction		**Improvement**					
N	***S***	***T***	**Waste**	**Time**	**Waste**	**Time**	**Waste**	**Waste**	**Waste**	**Time**
18	14	3	3.777%	32	2.658%	34	3.108%	3.108%	2.658%	4822
18	14	4	0.765%	89	0.340%	48	0.340%	0.340%	-	-
18	14	5	0.340%	187	0.007%	229	0.102%	0.102%	0.007%*	-
22	15	3	5.424%	47	4.547%	93	5.142%	5.424%	-	-
22	15	4	1.763%	131	1.126%	171	1.763%	1.763%	-	-
22	15	5	0.479%	242	0.271%	225	0.479%	0.479%	-	-
100	25	8	2.162%	1548	1.460%	9172	1.902%	1.958%	-	-

When we look at the results on the randomly generated instances in Table 3, we observe that CIH$_{P}$ and GRASP$_{s}$ find similar results for the particular case. The average waste of the solutions obtained by CIH$_{P}$ is slightly better on average. We find an optimal solution for each of these particular case instances, and CIH$_{P}$ also finds an optimal solution for all the instances but only one with 25 labels, and 5 templates with 20 slots. The average run time of CIH$_{P}$ on these instances is around 4 minutes. [3] report the average run time of GRASP$_{s}$ as less than 10 seconds. However, it is not clear whether it is the run time per run or for 10 runs. Given that they report the average run time per run for the instance with 100 labels, we assume that [3] report the solution time per run. Hence, GRASP$_{s}$ finds a solution in less than 100 seconds. The average run time of GRASP$_{s}^{our}$ on these particular case instances (for a total of 10 independent runs) is 33 seconds which

also supports our assumption on the run time of GRASP_s. The performances of GRASP_s and $\text{GRASP}_\text{s}^{\text{our}}$ are very close to each other. Although we find the optimal solution for all the particular case instances, solving MIPP^S becomes challenging for the instances with 20-slot templates as the number of labels and the number of templates increase beyond 15 and 4, respectively.

In Table 4, we present the results for the randomly generated instances for the general case of the problem. In the general case, the problem becomes more complicated and we find an optimal solution for only few of the instances. CIH_G performs equally or better than GRASP_m on each set of instances of the general case. On average, CIH_G improves the best known solutions by around 0.43%. The average run time of CIH_G on these instances is around 8 minutes, whereas GRASP_m finds a solution in less than 100 seconds (in total for 10 independent runs). Our implementation of GRASP_m ($\text{GRASP}_\text{m}^{\text{our}}$) provides very similar results with an average run time of 47 seconds. The average waste of the solutions obtained by $\text{GRASP}_\text{m}^{\text{our}}$ is only 0.062% higher than that of the solutions obtained by GRASP_m. We solve quite few of the general case instances to optimality using MIPG^S. Although the flexibility in assigning the labels to the slots in the general case provides an opportunity to obtain a better solution compared to the particular case, solving MIPG^S to optimality becomes prohibitive.

Finally, we present the results on the third set of instances with 100 labels. There are 25 slots on a template in these instances. We run each instance with 6, 8 and 10 templates. We present the particular case results in Table 5. Under "Time" column, we present the total run time of each algorithm. For $\text{GRASP}_\text{s}^{\text{our}}$, we report the total run time of 20 independent runs. As expected, as the the number of templates increase the amount of waste decreases both for CIH_P and $\text{GRASP}_\text{s}^{\text{our}}$. CIH_P outperforms $\text{GRASP}_\text{s}^{\text{our}}$ on each instance. The final solutions obtained by CIH_P has around 2.8% less waste compared to the benchmark algorithm. Even the initial solutions found at the end of the construction phase has 1.4% less waste on average than the solutions obtained by $\text{GRASP}_\text{s}^{\text{our}}$. The average run time of

Table 3: Comparison of the particular case results for the randomly generated instances

Instance			CIH_P				$GRASP_s$	$GRASP_s^{our}$	Optimal	
			Construction		Improvement					
N	***S***	***T***	**Waste**	**Time**	**Waste**	**Time**	**Waste**	**Waste**	**Waste**	**Time**
10	10	3	4.046%	8.2	4.046%	52.6	4.046%	4.046%	4.046%	2
10	10	4	1.787%	21.4	1.787%	66.4	1.787%	1.787%	1.787%	4
10	10	5	1.039%	94.8	1.039%	41.2	1.039%	1.039%	1.039%	10
10	20	3	2.110%	21.6	2.110%	42.4	2.110%	2.110%	2.110%	7
10	20	4	1.033%	49.6	1.033%	84.4	1.033%	1.033%	1.033%	27
10	20	5	0.679%	57.8	0.679%	56	0.679%	0.679%	0.679%	50
15	10	3	7.729%	5	7.729%	84.8	7.729%	7.729%	7.729%	2
15	10	4	3.328%	35.4	3.328%	74.8	3.328%	3.328%	3.328%	10
15	10	5	2.097%	255.4	2.097%	92.2	2.097%	2.097%	2.097%	79
15	20	3	3.361%	12.2	3.361%	52.8	3.361%	3.361%	3.361%	12
15	20	4	1.745%	90.4	1.745%	122.4	1.745%	1.745%	1.745%	443
15	20	5	0.882%	298.2	0.882%	201.2	0.882%	0.882%	0.882%	8780
20	10	3	14.428%	20	13.733%	48.6	13.733%	13.733%	13.733%	2
20	10	4	5.543%	91.4	5.543%	97.6	5.543%	5.543%	5.543%	8
20	10	5	3.165%	264.6	3.165%	140.2	3.165%	3.165%	3.165%	89
20	20	3	5.373%	52.4	4.789%	188.4	4.789%	4.789%	4.789%	16
20	20	4	2.652%	79	2.378%	203.4	2.581%	2.578%	2.378%	1210
20	20	5	1.348%	650.6	1.348%	208	1.364%	1.362%	1.348%	101798
25	10	3	18.078%	20.2	16.655%	102.4	16.655%	16.655%	16.655%	2
25	10	4	10.841%	42.4	9.413%	91.4	9.413%	9.413%	9.413%	7
25	10	5	5.920%	234.6	4.758%	112.4	4.758%	4.758%	4.758%	141
25	20	3	11.153%	45.2	7.301%	305.6	7.301%	7.301%	7.301%	12
25	20	4	4.801%	122.2	3.268%	368.2	3.311%	3.323%	3.268%	3751
25	20	5	2.633%	348.8	1.969%	394.4	2.070%	2.209%	1.919%	157942
Average			**4.824%**	**121.7**	**4.340%**	**134.7**	**4.355%**	**4.361%**	**4.338%**	**11434**

CIH_P (including both the construction and improvement phases) is around 2800 seconds which is less than 50% of the average run time of $GRASP_s^{our}$.

The results for the general case are presented in Table 6. Out of 30 instances,

Table 4: Comparison of the general case results for the randomly generated instances

Instance			CIH_G				$GRASP_m$	$GRASP_m^{our}$	Optimal	
			Construction		Improvement					
N	***S***	***T***	**Waste**	**Time**	**Waste**	**Time**	**Waste**	**Waste**	**Waste**	**Time**
10	10	3	1.851%	7.6	0.957%	99.8	1.733%	1.342%	-	-
10	10	4	0.484%	19.4	0.147%	178.4	0.393%	0.394%	-	-
10	10	5	0.085%	97.2	0.035%	313.6	0.060%	0.176%	-	-
10	20	3	0.476%	20.8	0.261%	146.8	0.572%	0.547%	-	-
10	20	4	0.090%	54.6	0.052%	256.2	0.082%	0.278%	-	-
10	20	5	0.032%	66.8	0.023%	374.4	0.024%	0.133%	-	-
15	10	3	5.570%	7.8	3.675%	73.6	4.917%	4.103%	-	-
15	10	4	1.763%	40.4	1.195%	214.8	1.546%	1.560%	-	-
15	10	5	0.380%	273.2	0.305%	351.6	0.334%	0.622%	-	-
15	20	3	1.319%	11	0.903%	308.4	1.536%	1.743%	-	-
15	20	4	0.289%	95.2	0.154%	530.6	0.330%	0.555%	-	-
15	20	5	0.065%	313.6	0.038%	778.8	0.043%	0.452%	-	-
20	10	3	10.356%	22	8.116%	54	10.168%	8.185%	7.890%	687.2
20	10	4	3.622%	86.4	2.552%	176.4	3.664%	3.140%	-	-
20	10	5	1.509%	273.6	0.841%	328.8	1.330%	1.991%	-	-
20	20	3	2.340%	53.2	2.007%	288.2	2.616%	2.959%	-	-
20	20	4	0.654%	83.2	0.475%	579.4	0.748%	1.781%	-	-
20	20	5	0.173%	666.8	0.158%	982.2	0.185%	0.768%	-	-
25	10	3	18.375%	22.4	15.321%	19.2	15.321%	15.321%	15.105%	21.8
25	10	4	6.411%	37.2	5.487%	152.4	6.323%	5.990%	-	-
25	10	5	2.728%	254.8	1.973%	253	2.399%	2.946%	-	-
25	20	3	3.821%	40.4	3.316%	536.4	3.820%	3.838%	-	-
25	20	4	1.190%	115.8	1.025%	674.6	1.179%	1.419%	-	-
25	20	5	0.453%	368.2	0.372%	843.8	0.360%	0.944%	-	-
Average			**2.668%**	**126.3**	**2.058%**	**354.8**	**2.487%**	**2.549%**	**-**	**-**

CIH_G provides a better solution for 29 instances compared to $GRASP_m^{our}$. For the remaining one instance, $GRASP_m^{our}$ obtains a better solution with only 0.003%

Table 5: Comparison of the particular case results for the large instances with 100 labels

Instance			CIH_P				$GRASP_s^{our}$	
			Construction		Improvement			
N	***S***	***T***	**Waste**	**Time**	**Waste**	**Time**	**Waste**	**Time**
100	25	6	9.450%	1095	7.835%	1277	9.940%	4667
100	25	6	7.470%	1050	6.652%	1336	9.197%	4288
100	25	6	8.198%	1121	6.430%	1361	7.708%	4859
100	25	6	9.135%	1172	8.231%	1167	10.374%	4885
100	25	6	7.636%	1134	6.098%	1381	11.583%	5097
100	25	6	9.271%	1088	7.845%	1254	11.622%	4285
100	25	6	8.902%	1072	7.098%	1346	9.151%	4174
100	25	6	8.101%	1174	6.990%	1158	11.180%	4966
100	25	6	8.336%	1213	7.614%	1353	10.343%	4742
100	25	6	9.516%	1294	8.110%	1125	10.710%	4473
Average			**8.601%**	**1141**	**7.290%**	**1276**	**10.181%**	**4644**
100	25	8	4.499%	1203	3.779%	1489	7.384%	6050
100	25	8	5.313%	1295	4.026%	1588	7.018%	5873
100	25	8	5.872%	1126	3.854%	1734	6.497%	5221
100	25	8	4.681%	1168	3.805%	1346	6.549%	6018
100	25	8	4.782%	1464	3.980%	1594	6.806%	6359
100	25	8	5.118%	1167	4.035%	1342	8.050%	6102
100	25	8	5.120%	1104	4.064%	1579	6.129%	5465
100	25	8	5.730%	1246	4.000%	1323	7.018%	6081
100	25	8	5.748%	1491	4.004%	1778	8.022%	5278
100	25	8	5.357%	1288	4.068%	1351	8.032%	5197
Average			**5.222%**	**1255**	**3.961%**	**1512**	**7.151%**	**5764**
100	25	10	5.585%	1233	3.294%	1861	6.592%	7055
100	25	10	4.308%	1336	3.422%	2008	5.325%	6848
100	25	10	5.439%	1308	3.771%	1872	5.752%	7650
100	25	10	4.580%	1329	3.506%	2315	5.134%	7893
100	25	10	5.555%	1353	3.635%	1879	6.336%	6878
100	25	10	4.123%	1302	3.636%	2047	5.712%	7725
100	25	10	5.848%	1625	3.701%	1714	5.727%	7011
100	25	10	4.656%	1610	3.325%	1823	5.720%	6906
100	25	10	5.442%	1501	3.388%	2134	6.140%	6762
100	25	10	4.429%	1356	3.727%	1826	5.607%	7069
Average			**4.997%**	**1395**	**3.541%**	**1948**	**5.805%**	**7180**

less waste. On the instances with 6 templates, CIH_G provides an average improvement of 0.79% in terms of waste compared to CIH_G. As the number of templates increases to 8 and 10, the average improvement by CIH_G reduces to 0.393% and 0.111%, respectively. On average, the solutions obtained by CIH_G has 0.43% less waste compared to $\text{GRASP}_\text{m}^{\text{our}}$. The average run times of both algorithms are comparable. CIH_G finds a solution in around 7150 seconds, whereas the average run time of $\text{GRASP}_\text{m}^{\text{our}}$ is close to 7400 seconds. These run times are in line with the run time reported by [3] as well. According to [3], the average run time for GRASP_m is 310 seconds per run for the instance with 100 labels and 8 templates. The average run time (per run) for the instances with 8 templates in Table 6 is around 360 seconds.

Overall, on the small instances with up to 25 labels the proposed heuristic provides similar (with a minor improvement) results for the particular case of the problem compared to the benchmark algorithm. However, for the large instances with 100 labels the proposed heuristic not only finds better solutions with around 3% less waste but also runs in around 50% less time compared to the benchmark algorithm. On the general case instances, the proposed heuristic approach obtains better solutions with an average improvement of of 0.4% in waste. The run time of the proposed approach is comparable with the benchmark algorithm on the large instances for the general case.

2.7 Conclusion

In this chapter, we analyze the *Label Printing Problem* (LPP). We present a strengthened linear integer model for the problem. We propose a novel two-phase heuristic algorithm (called the *Construct-Improve Heuristic* (CIH)) that uses the power of exact solution methods both in the first phase and second phase to find high-quality solutions. In the proposed approach, we first generate a set of good template designs using a sampling approach, and then choose the best set of template designs by solving a set covering-type problem. Then, in the second phase

Table 6: Comparison of the general case results for the large instances with 100 labels

Instance			CIH_G				$GRASP_m^{our}$	
			Construction		Improvement			
N	***S***	***T***	**Waste**	**Time**	**Waste**	**Time**	**Waste**	**Time**
100	25	6	10.666%	1177	6.097%	3355	7.067%	5400
100	25	6	9.613%	1136	5.592%	2977	6.244%	5743
100	25	6	10.050%	1203	5.446%	3966	6.189%	5525
100	25	6	9.791%	1259	5.790%	2257	6.562%	5888
100	25	6	10.403%	1214	5.377%	2482	6.283%	5901
100	25	6	8.335%	1182	6.055%	2294	6.766%	5412
100	25	6	10.463%	1163	4.974%	3054	5.680%	6126
100	25	6	8.952%	1248	5.668%	3857	6.786%	5337
100	25	6	10.189%	1287	5.777%	3367	6.290%	5953
100	25	6	10.049%	1386	6.048%	5519	6.853%	5748
Average			**9.851%**	**1226**	**5.682%**	**3313**	**6.472%**	**5703**
100	25	8	2.641%	1276	1.861%	5435	2.186%	6883
100	25	8	3.159%	1388	1.538%	10069	1.651%	7228
100	25	8	3.213%	1202	1.657%	6727	2.229%	6985
100	25	8	2.491%	1246	1.926%	4125	2.491%	7831
100	25	8	3.179%	1594	1.711%	9997	2.476%	7087
100	25	8	3.217%	1240	1.882%	4259	2.226%	7502
100	25	8	3.283%	1179	1.462%	5984	1.691%	7733
100	25	8	3.224%	1223	1.700%	8078	1.885%	7762
100	25	8	3.138%	1570	1.904%	8854	2.352%	6929
100	25	8	3.219%	1344	1.945%	7692	2.328%	6857
Average			**3.076%**	**1326**	**1.759%**	**7122**	**2.151%**	**7280**
100	25	10	0.907%	1316	0.719%	4680	0.739%	8571
100	25	10	1.028%	1388	0.617%	5546	0.730%	9930
100	25	10	0.901%	1373	0.557%	5085	0.647%	9496
100	25	10	0.866%	1472	0.743%	7219	0.740%	9904
100	25	10	1.086%	1462	0.639%	5722	0.695%	8656
100	25	10	1.226%	1324	0.807%	8976	0.901%	9208
100	25	10	1.125%	1629	0.615%	9484	0.953%	9186
100	25	10	1.018%	1661	0.610%	10081	0.816%	8932
100	25	10	1.027%	1497	0.562%	8103	0.748%	8703
100	25	10	0.892%	1393	0.695%	5718	0.706%	9151
Average			**1.008%**	**1452**	**0.656%**	**7061**	**0.767%**	**9174**

we improve the solution found by using some improvement ideas. We utilize the strengthened linear integer model in the improvement phase.

We implement the proposed solution approach for both variants (particular and general) of LPP and first test it on the instances from the literature. We find an optimal solution using the proposed mathematical model for all the particular case instances of LPP except the large instance with 100 labels. On these instances, our proposed heuristic algorithm obtains slightly better results compared to the benchmark algorithm. On the general case instances, we obtain much better results, and the proposed heuristic approach improves the best known solutions by around 0.4%. We also test the proposed heuristic CIH on some new large instances with 100 labels and compare its performance against that of the benchmark algorithm. We observe that CIH outperforms the benchmark algorithm by a margin of 2.89% and 0.43% for the particular and general cases, respectively on these large instances. Finally, the average run times of CIH and the benchmark algorithm are comparable on these large instances.

CHAPTER III

A SET-COVERING BASED HEURISTIC APPROACH FOR A VARIANT OF THE COVER PRINTING PROBLEM

In this chapter, we study a planning problem with an application in the offset printing process for manufacturing napkin pouches. After the customer orders are received, the customer-specific designs are assigned to one or more slots of a multi-slot plate which is used to print the napkin pouches. Then, the number of plate rotations is determined for each plate layout to satisfy the demand. The number of napkin pouches produced in each a slot is equal to the number of plate rotations. Each time a new plate layout (with assignment of napkin pouches to slots) is used in the printing process, a setup cost is incurred. Each customer order includes a customer-specific napkin pouch design with a certain demand amount, color code and with/without a white border feature. Due to technological and organizational constraints, (i) a plate cannot contain designs with more than a maximum number of different colors, (ii) either design(s) with a white border feature have to assigned to the slots at both ends of the plate or a standard design (considered as overproduction) has to be assigned to a slot on one end of the plate, and (iii) a design cannot be assigned to more than one plate. The goal in this planning problem is to determine (i) the number of plates used, (ii) the plate layout for each plate, and (iii) the number of plate rotations to satisfy the demand with minimum total setup and overproduction cost. We develop a set-covering based solution approach followed by an improvement phase to address this problem. We first generate several feasible plate layouts and then solve a set-covering type problem to determine the number of plates to be used, their layouts and the number of rotations for each. Then, we improve the solution by

performing a local search. We compare the performance of the proposed solution approach against the benchmark algorithms and obtain much better results.

3.1 Introduction

In this chapter, we consider a production planning problem in offset printing with an application in napkin pouch manufacturing. The setting studied in this chapter is first introduced by [12]. We briefly explain the production environment analyzed here.

In the setting under study, a manufacturer receives customer orders with customer-specific designs for the napkin pouches in addition to its standard designs. The designs are imprinted on the pouches using multi-slot plates. Each slot is used to imprint one design only, however, multiple slots of a plate can be used to imprint a specific design. Once the assignment of designs to a plate is decided, then a plate is rotated to the surface of printing paper and one unit of each design assigned to the slots of the plate are produced. A plate can be rotated as many times as desired to imprint more copies of the designs assigned to its slots. Each time the assignment of pouch designs to the slots is changed to imprint different designs, a setup is required on the plate.

After receiving customer orders, the manufacturer decides about (i) how many plates to use, (ii) how to allocate customer-specific designs to each plate, and (iii) the number of rotations for each plate while minimizing the total setup cost and overproduction cost. Each customer order includes a customer-specific design (with a demand amount) with a certain color and with or without a white border. Due to technological constraints, at most a certain number of different colored designs can be assigned to the slots of a plate. Moreover, due to a small gap between the plate borders leading to a stripe on the paper, designs with a white border have to be assigned to at least two slots of the plate. Another solution to this technological restriction is to assign a standard design (which can be of any color) to one slot of a plate. To satisfy the demand for all the customer-specific

designs, some designs may be overproduced (due to the incurred setup cost if the plate layout is changed). The total amount of standard design napkin pouches produced is also considered as overproduction and has a lower overproduction cost compared to a customer-specific design [12]. In general, a customer-specific design can be assigned to the slot(s) of multiple plates to obtain a better solution. However, in that case packaging of a certain customer-specific design has to be postponed until the production on all the plates used to imprint it is completed. This requires more space for packaging and storage and is not desired. Hence, each design is assigned to slot(s) of a single plate only. This is called the split constraint [12].

The problem investigated in this chapter is a variant of the *Cover Printing Problem* (CPP) [3, 7, 9]. It has additional features such as the color constraint and the white border constraint. Hence, we call this problem the *Cover Printing Problem with Design Restrictions* (CPP-DR). These features add additional complexity to the planning of the production in practice and makes the problem more challenging. In this chapter, we analyze CPP-DR and propose a set-covering based heuristic approach for the problem. In the proposed solution approach, we first generate a set of feasible assignment of designs to the slots of the plates. Then, within each plate, we determine the number of slots assigned to each design. Finally, we solve a set-covering model to determine which plates to use in production. As a last step, we implement a local search phase to improve the final solution by changing the the assignment of designs to the slots. We test both the proposed heuristic and the different implementations of the benchmark algorithm [12] on the instances generated based on the setting in [12]. We observe much better results when we compare the proposed two-phase heuristic with the benchmark algorithms in the literature.

The rest of the chapter is organized as follows. In Section 3.2, we review the related works in the literature. In Section 3.3, we introduce a formal definition of the *Cover Printing Problem with Design Restrictions* (CPP-DR) and present an

illustrative example to further explain the problem. In Section 3.4, we present our set-covering based heuristic algorithm two-phase. In order to evaluate the performance of the proposed approach under different problem settings, we demonstrate a computational study and compare the solutions found against algorithms developed in the literature. We present the results of the computational experiments in Section 3.5. Concluding remarks are provided in Section 3.6.

3.2 Literature Review

In this section, we review the related works in the literature. Two problems that are most related to our study are *Label Printing Problem* (LPP) and *Cover Printing Problem* (CPP).

CPP has applications in offset printing including printing book covers, labels, business cards, advertisement items, etc. For simplicity of the discussion, we will use book cover printing as an application of CPP. CPP deals with production planning in offset printing using multi-slot templates. Each slot of a template can be used to print one type of book cover. A book cover is assigned to the slot(s) of one or more templates. Each time a different template (with a new assignment of book covers to the slots) is used in the production a setup cost is incurred. Once a production is initiated using a given template, the book cover assigned to each slot is printed until the production process is stopped. A variable production cost is incurred depending on the number of times each template is used to print book covers. Production beyond the demand is considered as waste. Decisions involved in CPP are (i) how many templates to use, (ii) the assignment of book covers to slots of each template, and (iii) how many times to print using each template. The goal is to satisfy the demand for a set of book covers with minimum total setup and production cost.

CPP is first introduced by [6]. The authors develop a simulated annealing algorithm that utilizes linear program to determine the number of times each template is used to print for a given set of templates. Motivated by the practices in

the fashion industry, [15] propose a mixed integer model for CPP and then linearize it by fixing the number of setups. [7] develop an adaptive genetic algorithm and a Pareto fitness genetic algorithm to handle two conflicting objectives of the problem. [8] present a greedy random adaptive search procedure (GRASP) which is later redesigned by the same authors both for CPP and its variant LPP [3]. [5] propose a strengthened mixed-integer linear model, and two ad hoc heuristics to solve the problem one of which has an approximation guarantee of 2. [9] analyze some special cases of the problem with a fixed number of templates and present an ad hoc heuristic for CPP that uses the special case with two templates in a local search framework. Recently, [10] propose two heuristics: (i) a simulated annealing algorithm where the number of imprints made for a given a number templates (with specified slot-item assignments) is determined by solving a linear program, and (ii) a hybrid approach using tabu search and the ad hoc procedure developed by [9].

A variant of CPP is LPP where the number of setups/templates is given, and the goal is to find a production plan with minimum waste. Two variants of LPP are studied in the literature. The first one allows assigning each book cover to the slot(s) of one template only (called the particular case), and the second variant (call the general case) allows assigning each book cover to the slot(s) of more than one plate. [16] propose two alternative mixed integer models for the general case of LPP. [17] develop two genetic algorithms using these two alternative formulations for the problem. [18] combine an ant colony optimization with an integer programming model in order to find good combinations of templates by minimizing excess production. [4] present a two-level solution approach for the particular case where the assignment of book covers to the templates is determined in the outer level, and the assignment of each book cover to the slots for each template and the number of imprints for each template template is determined in the inner level. This two-level solution approach is implemented within a simulated annealing framework. [2] present an effective immune based evolutionary algorithm for

the particular case. [3] analyze both general and particular cases of the problem and propose a GRASP algorithm.

In addition to LPP and CPP, [19] study a variant where the goal is find the minimum setup production plan among all the minimum waste ones. They propose an exact two-stage enumerative approach to find all optimal solutions. [20] aim to find the best combination of number of layers while satisfying the demand. [21] study the similar problem of study [20] and they compare their proposed approaches with the algorithms in study [20]. [11] formulate a variant of CPP as a multi-objective nonlinear integer model. They consider upper and lower bounds for the number of imprints for each template and propose three ad hoc heuristics to address the problem.

The setting studied in this chapter is first presented by [12]. The authors develop a mixed-integer model for the problem with some symmetry-breaking constraints. They also propose savings-based heuristics where each item is assigned to a separate template, and then the templates are merged based on savings using three different selection rules. In the improvement phase, the swapping of items between two templates and moving an item to another template are implemented iteratively. For the same problem, we present a novel set-covering based heuristic followed by an improvement step and obtain much better results.

3.3 Problem Definition

In this section, we introduce a formal definition of the extension of the *Cover Printing Problem* (CPP-DR) which has applications in the manufacturing of napkin pouches and propose strengthened mixed-integer linear model for this problem. This problem is related to the publishing industry and focus on finding the best assignment of a set of customer-specific designs for napkin pouches to offset plates for print. A fixed number of N different customer-specific designs each of which has a certain demand and color-code is considered. For each type of design, d_i denotes the demand of design i. Designs are ordered according to

their demands ($d_1 \geq d_2 \geq \ldots \geq d_N$) and we use $\mathcal{N}$ to denote the set of designs ($\mathcal{N} := \{1, 2, \ldots, N\}$). In CPP-DR, plates are used to produce the designs and the number of plates used is a decision. Each plate includes a fixed number of slots and each slot is assigned to at most one design at the same time. We use P to denote the number of plates and use S to denote the number of slots on each plate. C is used to denote the number of color-codes of the designs. In this problem, a maximum of P plates can be used to produce designs. Each slot of each plate can be used to produce at most one type of design. However, one design can be assigned to multiple slots. The number of produced design in each slot is same for all slots on a plate and this value is equal to the number of rotations of that plate. In an extension of CPP (CPP-DR), different decisions to determine in this problem are i) how many plates to use, ii) the plate layout for each plate, and iii) the number of rotations to satisfy demand for each design. The aim in this problem is to minimize the total production cost. These cost includes i) over-production costs, which is for the excessive demand, and ii) setup costs, which occurs for initiating a new plate that is used. Unlike the traditional CPP, we consider two additional constraints due to technological reasons while minimizing the total production cost. The first one is white-border (WR) constraint which necessitates to assign at least two slots by a design that has a white-border on each plate. Alternatively, a standard design can be assigned to one slot to avoid this constraint. The second one is color-code constraint that forces the maximum number of color-codes of the designs can be assigned to a plate. Furthermore, we focus on customer-specific designs to satisfy their demands and we assume that standard designs do not have demand.

We present a simple numerical example with 5 designs and 7 slots on each plate to illustrate the problem. At most two different colors of the designs must be assigned to each plate. The setup cost per plate is 540 and the excessive cost per unit of a standard design is 0.001. The remaining data set is presented in Table 7. We need at least three plates to satisfy the demand requirements because of

the color constraint. We provide an assignment of designs on the plates and the number of rotations (L_j) for each plate in Figure 2. In this solution, design 3 is assigned to 2 slots and design 5 is assigned to 4 slots of first plate. Design 1 is assigned to 2 slots, and the remaining slots are assigned to design 4 for the second plate. For the third plate, design 2 is assigned to 6 slots of this plate. One slot standard design is assigned to plates 1 and 3 to satisfy the white-border constraint. The number of rotations using plates 1, 2 and 3 are 12500, 6000 and 3334, respectively. The total number of produced units of design 1 is 12000 ($= 2 \times 6000$), and waste is 2000 ($= 12000 - 10000$). Similarly, excessive production for designs 2, 3, 4 and 5 are 4, 0, 0 and 0, respectively. Hence, total waste is 2004 and excessive production cost is 7.014 ($= 2004 \times 0.0035$). Total excessive production of standard designs for plates 1 and 3 are 12500 and 3334, respectively. Hence, total waste is 15834 and waste cost for these designs is 15.834 ($= 15834 \times 0.001$). Total setup cost is 1620 ($= 3 \times 540$) and total production cost is 1642.848 ($= 1620 + 15.834 + 7.014$).

Table 7: Input data of illustrative example

Designs	Demand	White Border	Color Code	Excessive Unit Cost
D1	10000	Yes	1	0.0035
D2	20000	No	2	0.0035
D3	25000	No	3	0.0035
D4	30000	No	4	0.0035
D5	50000	No	5	0.0035

We develop a strengthened mixed-integer linear model for extension of CPP (CPP-DR) inspired by model presented in [5]. For the ease of notation, we define $\mathcal{P}(:= \{1, \ldots, P\})$, $\mathcal{S}(:= \{0, 1, \ldots, S\})$ and $\mathcal{C}(:= \{0, 1, \ldots, C\})$ as the set of indices for the plates, slots and color-codes, respectively. We use $\mathcal{N}^o$ and $\mathcal{N}^p$ for the customer-specific designs and standard designs. The set of indices $\mathcal{N}_c$ and $\mathcal{N}_w$ are determined for the designs with color-codes and white-border, respectively. We first define the parameters and decision variables used in the formulation:

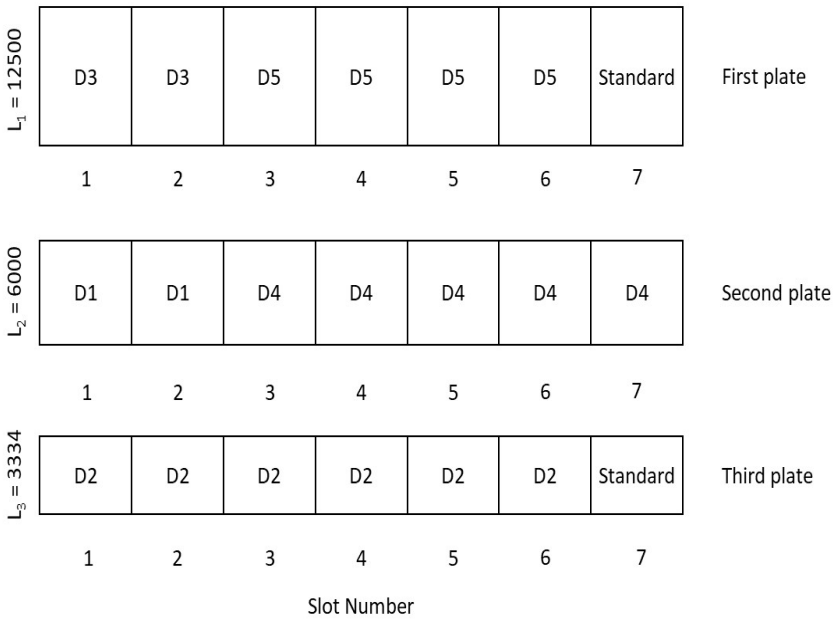

Figure 2: A feasible solution for the example.

Parameters

$\bar{c}$: Maximum number of different color-codes per plate

c^o : Overproduction cost per unit of customer-specific design i

c^j : Setup cost per plate

c^p : Overproduction cost per unit of standard design

d_i : Demand of design i

M : Sufficiently large number

Decision Variables

$$\mathrm{w}_j = \begin{cases} 1, & \text{if plate } j \text{ is used} \\ 0, & \text{otherwise.} \end{cases} \qquad j \in \mathcal{P}$$

$$\mathrm{y}_{ij} = \begin{cases} 1, & \text{if customer-specific design } i \text{ is assigned to plate } j \\ 0, & \text{otherwise.} \end{cases} \qquad i \in \mathcal{N}^o, j \in \mathcal{P}$$

$$\mathrm{x}_{ijk} = \begin{cases} 1, & \text{if design } i \text{ is assigned to } k \text{ slots of plate } j \\ 0, & \text{otherwise.} \end{cases} \qquad i \in \mathcal{N}, j \in \mathcal{P}, k \in \mathcal{S}$$

$$\mathrm{z}_{cj} = \begin{cases} 1, & \text{if any design with color-code } c \text{ is assigned to plate } j \\ 0, & \text{otherwise.} \end{cases} \qquad c \in \mathcal{C}, j \in \mathcal{P}$$

q_{ijk} = copies of design i printed using plate j if design i is assigned to k slots of plate j $\quad \mathrm{i} \in \mathcal{N}, j \in \mathcal{P}, k \in \mathcal{S}$

L_j = number of rotations made using plate j $\quad j \in \mathcal{P}$

v_j = overproduction copies of standard designs on plate j $\quad j \in \mathcal{P}$

Using these parameters and decision variables, we propose the mathematical formulation of CPP-DR as follows:

$$\text{MIP: Min} \quad \sum_{j\in\mathcal{P}}(c^o(L_jS - \sum_{i\in\mathcal{N}^o} d_i) - c^o v_j) + \sum_{j\in\mathcal{P}} c^p v_j + \sum_{j\in\mathcal{P}} c^j w_j \tag{34}$$

s.t.

$$\sum_{j\in\mathcal{P}}\sum_{k\in\mathcal{S}} q_{ijk} \geq d_i \qquad i\in\mathcal{N} \tag{35}$$

$$\sum_{i\in\mathcal{N}}\sum_{k\in\mathcal{S}} kx_{ijk} \leq S \qquad j\in\mathcal{P} \tag{36}$$

$$q_{ijk} \leq kL_j \qquad i\in\mathcal{N}, j\in\mathcal{P}, k\in\mathcal{S} \tag{37}$$

$$q_{ijk} \leq Mx_{ijk} \qquad i\in\mathcal{N}, j\in\mathcal{P}, k\in\mathcal{S} \tag{38}$$

$$q_{ijk} \geq kL_j - M(1-x_{ijk}) \qquad i\in\mathcal{N}^P, j\in\mathcal{P}, k\in\mathcal{S} \tag{39}$$

$$\sum_{k\in\mathcal{S}} x_{ijk} = 1 \qquad i\in\mathcal{N}, j\in\mathcal{P} \tag{40}$$

$$\sum_{j\in\mathcal{P}} y_{ij} = 1 \qquad i\in\mathcal{N}^o \tag{41}$$

$$kx_{ijk} \leq Sy_{ij} \qquad i\in\mathcal{N}^o, j\in\mathcal{P}, k\in\mathcal{S} \tag{42}$$

$$L_j \leq Mw_j \qquad j\in\mathcal{P} \tag{43}$$

$$kx_{ijk} \leq Sw_j \qquad i\in\mathcal{N}, j\in\mathcal{P}, k\in\mathcal{S} \tag{44}$$

$$\sum_{c\in\mathcal{C}} z_{cj} \leq \bar{c} \qquad j\in\mathcal{P} \tag{45}$$

$$\sum_{i\in\mathcal{N}_c}\sum_{k\in\mathcal{S}} x_{ijk} \leq Sz_{cj} \qquad c\in\mathcal{C}, j\in\mathcal{P} \tag{46}$$

$$v_j = \sum_{i\in\mathcal{N}^P}(\sum_{k\in\mathcal{S}} q_{ijk} - d_i) \qquad j\in\mathcal{P} \tag{47}$$

$$w_j \leq \sum_{i\in\mathcal{N}_w}\sum_{k\in\mathcal{S}} \tfrac{k}{2}x_{ijk} + \sum_{i\in\mathcal{N}^p}\sum_{k\in\mathcal{S}} kx_{ijk} \qquad j\in\mathcal{P} \tag{48}$$

$$\sum_{i\in\mathcal{N}^p}\sum_{k\in\mathcal{S}} kx_{ijk} \leq 1 \qquad j\in\mathcal{P} \tag{49}$$

$$L_j \geq L_{j+1} \qquad j\in\mathcal{P}\setminus\{P\} \tag{50}$$

$$\sum_{i\in\mathcal{N}}\sum_{k\in\mathcal{S}\setminus\{0\}} q_{ijk} \leq SL_j \qquad j\in\mathcal{P} \tag{51}$$

$$q_{ijk} \geq 0 \qquad i\in\mathcal{N}, j\in\mathcal{P}, k\in\mathcal{S} \tag{52}$$

$$L_j \geq 0 - integer \qquad j\in\mathcal{P} \tag{53}$$

$$v_i \geq 0 - integer \qquad i\in\mathcal{N} \tag{54}$$

$$x_{ijk} \in \{0,1\} \qquad i\in\mathcal{N}, j\in\mathcal{P}, k\in\mathcal{S} \tag{55}$$

$$y_{ij} \in \{0,1\} \qquad i\in\mathcal{N}^o, j\in\mathcal{P} \tag{56}$$

$$z_{cj} \in \{0,1\} \qquad c\in\mathcal{C}, j\in\mathcal{P}, k\in\mathcal{S} \tag{57}$$

$$w_j \in \{0,1\} \qquad j\in\mathcal{P} \tag{58}$$

In this model, the objective (34) is to minimize the total production cost.

Constraint (35) guarantees that the demand of each type of design is satisfied. Constraint (36) ensures that at most S slots can be assigned to the designs in each plate. Constraint (37) provides that total amount of produced design i on plate j cannot be greater than the multiply of the corresponding values for k and L_j. Constraint (38) guarantees that a plate can be used to produce design i only if any slot(s) are assigned to design i for that plate. Here, M is large enough number. The number of units standard designs produced of a plate must be equal with the number of the rotations of that plate. This is enforced by constraint (39). Constraint (40) determines the number of slots assigned to a design in a plate. Constraint (41) provides that a customer-specific design must be allocated to one plate only. Constraint (42) helps to determine the plate each design is assigned to by relating x_{ijk} and y_{ij}. The number of rotations for a plate is determined only if that plate is used. This is enforced by constraint (43). Constraint (44) provides that a design can be assigned to some slot(s) on a plate only if that plate is used. Constraint (45) ensures that at most $\bar{c}$ different color-codes of designs can be assigned to a plate. Constraint (46) determines the color-code of the design is assigned to a plate. In constraint (47), the number of excessive units of a standard design are calculated for each plate. For each plate, constraint (48) provides that either designs that have white-border can be assigned to at least two slots or a standard design can be assigned to one slot. Moreover, at most one slot standard design can be assigned to each plate. This is provided by constraint (49). In constraint (50), plates are sorted according to the number of rotations. Constraint (51) ensures that the total amount of designs produced using a plate is limited by the number of rotations using that plate multiplied by the number of slots. Constraints (50) and (51) are added to strengthen the formulation even though they are redundant constraints. Remaining constraints (52-58) are the sign, binary and integrality restrictions.

We strengthen the proposed model MIP using a strengthening idea to restrict the solution area. Here, we propose a lower bound on the number of plates to be

used in this problem. In CPP-DR, each plate must include at most $\bar{c}$ color-codes. Hence, this lower bound is found as follow:

$$P_{LB} = \max\{\lceil \frac{N}{S} \rceil, \lceil \frac{C}{\bar{c}} \rceil\} \tag{59}$$

We use P_{LB} to strengthen the mathematical model and add the following valid inequality to model MIP. We use MIP$^{\mathrm{S}}$ to present the strengthened version of the proposed model by MIP. The strengthened model provides to tighten the feasible solution area. Therefore, we use MIP$^{\mathrm{S}}$ to solve some instances optimally and to find lower bound in a limited time for the large instances.

$$\sum_{j \in \mathcal{P}} w_j \geq P_{LB} \tag{60}$$

3.4 Proposed Solution Approach

In this section, we analyze the extension of the CPP which is called as CPP-DR. We propose a heuristic algorithm to find a good solution since CPP-DR is an NP-hard problem. We first describe the proposed heuristic algorithm which is consisted of the two-phases: i) construction phase and ii) improvement phase. We call this heuristic algorithm the Iterative-Improvement Algorithm (IIA). Here, we assign the designs to the slots of a plate. This layout is called as a plate design. A plate design denotes the number of slots are assigned for each design in that plate. Firstly, we obtain an initial feasible solution with using a construction subroutine by determining the number of rotations for plates and the allocation of the designs over the plates. To do this, we first generate several plate designs and then the best plate designs among the all generated sets are chosen by solving a variant of the *Set Covering Problem* [22]. Furthermore, we determine the number of plates with using the set-covering type problem. The main idea in the proposed algorithm is to construct an initial feasible solution, and then improve this solution using an improving idea iteratively. In the wake of the constructing an initial feasible solution, we try to improve this solution by using a restricted version of MIP$^{\mathrm{S}}$.

We decrease/increase the number of slots each design is assigned to on each plate while improving the initial solution.

3.4.1 Construction Phase

In the construction phase, we first implement a construction subroutine for generating random subsets of designs and to determine the plate layouts of these random subsets with their total cost. We construct a feasible solution by choosing the best set of plate designs among all generated several plate designs. We first generate candidate several plate designs. The plate layout and the number of rotations for each plate is determined with implementing a heuristic procedure called *PlateLayout* (PL). Then, we solve a set covering model [22] in order to select the best set of plate designs of all generated sets. To do that, we use a construction subroutine for generating random subsets of designs and construct an initial plate designs with using these generated random subsets. We generate only one plate design for each random generated subset of designs.

We focus on generating several plate designs with determining the number of rotations and total production cost for each generated plate. To do this, we first generate plate designs and then, implement the PL procedure to determine the number of rotations and the total production cost of each generated plate. So, we determine the plate design and its total production cost with using PL procedure. In this procedure, we first assign one slot to each design in $\mathcal{N}'$ and calculate the number of slots with white-border (Steps 5-10). After calculating the number of slots with white-border, assign one slot to standard design if a standard design is necessary (Steps 11-13). Then, we increase the number of slots assigned to the design considering the highest demand-to-assigned slots ratio by one. This step continues until all the slots are assigned to a design. At each iteration, we update the number of slots with white-border and if a standard design is necessary, we assign one slot to this design in that plate to satisfy the white-border constraint. This implementation is provided in steps (14-21). Finally, we determine the number of rotations for a plate with using number of slots assigned

Algorithm 7 PlateLayout (PL)

1: **Input:** A subset of $\mathcal{N}'$ of designs
2: **Output:** A set of plate layouts
3: nw = 0 (Initialize the number of white-border designs)
4: st = 0 (Assign zero slot to standard design)
5: **for** $i \in \mathcal{N}'$ **do**
6: $r_i = 1$ (Assign one slot to each design in subset $\mathcal{N}'$)
7: **if** design i has white-border **then**
8: nw = nw + r_i
9: **end if**
10: **end for**
11: **if** $nw \leq 1$ **then**
12: st = 1
13: **end if**
14: **while** $\sum_{i \in \mathcal{N}'} r_i < S - st$ **do**
15: $k = \text{argmax}_{i \in \mathcal{N}'}\{\lceil \frac{d_i}{r_i} \rceil\}$
16: $r_k = r_k + 1$
17: Update nw
18: **if** $nw \geq 2$ **then**
19: st = 0
20: **end if**
21: **end while**
22: $L_{max} = \max_{i \in \mathcal{N}'}\{\lceil \frac{d_i}{r_i} \rceil\}$
23: Calculate Cost
24: **if** $st = 0$ & $nw \geq 2$ **then**
25: st = 1
26: r'_i=1, $\forall i \in \mathcal{N}'$
27: **while** $\sum_{i \in \mathcal{N}'} r'_i < S - st$ **do**
28: $k = \text{argmax}_{i \in \mathcal{N}'}\{\lceil \frac{d_i}{r'_i} \rceil\}$
29: $r'_k = r'_k + 1$
30: **end while**
31: $L'_{max} = \max_{i \in \mathcal{N}'}\{\lceil \frac{d_i}{r_i} \rceil\}$
32: Calculate $Cost'$
33: **if** $Cost' < Cost$ **then**
34: Cost = $Cost'$
35: $r_i = r'_i$, $\forall i \in \mathcal{N}'$
36: $L_{max} = L'_{max}$
37: **end if**
38: **end if**
39: **if** Solution is infeasible **then**
40: Cost = ∞
41: **end if**

(Step 22) and calculate the total production cost (Step 23). When the number of slots with white-border is greater than or equal to two and standard design is not assigned on that plate, we check whether assign one slot to standard design or not considering the total production cost. Lastly, we finalize the plate design and its total production cost (Steps 24-38). However, we consider that a plate may contain designs at most $\bar{c}$ number of different color-codes. Thus, if a plate includes more than $\bar{c}$ number of different color-codes, this plate becomes infeasible. If a plate design does not have a feasible solution, then we update the total production cost as ∞ (Steps 39-41). We present the steps of PL in Algorithm 7.

In the wake of implementing PL algorithm, we store both the plate designs,

the number of rotations and the total production cost for each random subset $\mathcal{N}'$ which has feasible solution. Namely, we determine the number of rotations and the total production cost of each plate whose design is determined. In order to generate several candidate plate designs, a subset $\mathcal{N}'$ is generated randomly and then, we implement PL algorithm multiple times. This implementation is made each time with a randomly generated subset $\mathcal{N}'$. After generating the plate designs with the number of rotations and the total production cost, we solve a set covering model in order to choose a best set of plate designs among all the generated sets. Also, we determine the number of plates used with this model. The main objective is to minimize the total overproduction and setup costs while satisfying the demand for each design.

In this model, the set of candidate plate designs is denoted as Γ. For each generated plate design l in Γ, we use z_{il} to denote the number of assigned slots to design i in plate l and K_l to represent the number of rotations for that plate. Furthermore, we use B_l to denote the total production cost for plate l. We have only one type of decision which indicates a plate is selected or not. The number of rotations and the total production cost for each plate are parameters in the proposed model. Decision variable is used as the following:

$$u_l = \begin{cases} 1, & \text{if plate design } l \text{ is chosen} \\ 0, & \text{otherwise.} \end{cases} \qquad l \in \Gamma$$

The proposed set covering-type mathematical model is as follows:

$$\text{SCP: Min} \quad \sum_{l \in \Gamma} B_l u_l \tag{61}$$

s.t.

$$\sum_{l \in \Gamma} K_l z_{il} u_l \geq d_i \quad i \in \mathcal{N} \tag{62}$$

$$\sum_{l \in \Gamma} u_l \leq P \tag{63}$$

$$\sum_{l \in \Gamma_i} u_l = 1 \qquad i \in \mathcal{N} \tag{64}$$

$$u_l \in \{0, 1\} \qquad l \in \Gamma \tag{65}$$

In this model, the objective function (61) is to minimize the total production cost. In Constraint (62), we provide to satisfy demand for each design. In constraint (63), we limit the number of used/selected plate designs by P. Constraint (64) ensures that each design is assigned to only one plate. Remaining constraint is the binary restriction for the decision variable.

If the number of plate designs generated increases, it can be difficult to solve SCP. Therefore, we limit the size of the generated plate designs to simplify to solve the problem. Namely, we first generate a set of plate designs. We obtain these plate designs using PL algorithm for each subset of $\mathcal{N}$. We store the plate designs with the number of rotations and their total production cost for each plate in a list (say, $CandidateList$). Then, we solve SCP and select the best set of plate designs among the all sets in $CandidateList$. Finally, we determine an initial feasible solution. The steps of the construction subroutine are summarized in Algorithm 8.

Algorithm 8 Construction Subroutine (CS)

Input: An CPP-DR instance
Output: A feasible solution
$Cost = \emptyset$
$CandidateList = \emptyset$
for $i = 1$ *to* $Max\_Iteration1$ **do**
  $\mathcal{N} = \{1, 2, \ldots, N\}$
  γ=0
  while $\gamma < N$ **do**
    α = Random(1,min$\{S, N-\gamma\}$) (Choose a subset length)
    $\mathcal{N}' = (j_1, j_2, .., j\alpha)$ (Choose a random subset of $\mathcal{N}'$ from design set of $\mathcal{N}$)
    Solve PL and add the plate designs found to $CandidateList$ if the solution is feasible
    Remove selected designs from design set of $\mathcal{N}$
    $\gamma = \gamma + \alpha$
  end while
end for
Solve SCP for $CandidateList$
return Cost

3.4.2 Improvement Phase

After construction an initial solution, we implement an iterative improvement idea in order to obtain a better solution in the improvement phase. We change the allocation of designs to slots and solve a restricted version of the mathematical model MIP$^{\mathrm{S}}$ to determine the plate layouts by minimizing the total production cost.

At the end of the construction phase, the set of selected plate designs is represented as $\overline{\Gamma}$. In the improvement phase, we first determine a subset of $\mathcal{N}$ and we allow that the number of slots assigned to each design in each plate to increase or decrease by up to λ for the designs in this subset. Then, we solve a mathematical model in order to improve the solution. We now use z'_{ij} as a parameter to present number of slots assigned to design i in plate j. In the improvement phase, we determine two values (max_{ij} and min_{ij}) to present the maximum and minimum allowable number of slots for design i in plate j. Thus, we limit the number of slots design i might be assigned to in plate j. We replace subset S with the set $\mathcal{S}_{ij}$ values ($\mathcal{S}_{ij} = \{min_{ij}, \ldots, max_{ij}\}$) by using the values max_{ij} and min_{ij}.

The implementation at the improvement phase continues until there is no improvement. At each iteration of the this phase, we update the set $\mathcal{S}_{ij}$ as the value k is selected by $\mathcal{S}_{ij}$. Then, we solve the restricted version of the proposed model with using updated set $\mathcal{S}_{ij}$.

This restricted model is denoted as MIP$^{\mathrm{S}}$-R. After solving MIP$^{\mathrm{S}}$-R, we update z'_{ij} values if a better solution is found and continue to solve MIP$^{\mathrm{S}}$-R until there is no improvement.

In the improvement part of the IIA algorithm, we only focus on changing the number of slots for designs in selected subset. For this designs, we allow changing the number of slots in all the plates. We first choose a subset of designs randomly and allow increasing/decreasing number of assigned slots to selected designs for all the plates. The improvement phase is provided in Algorithm 9. In this phase, we try to obtain a better solution with changing the number of assigned slots to

selected designs and this process is implemented for a certain number of iterations ($Max\_Iteration2$). We determine a best solution at the end of the number of a certain number of iterations. If the solution is improved, we update the solution and the improvement implementation continues until there is no improvement.

Algorithm 9 Improvement Phase

Input: A feasible solution found at the construction phase
Output: A best solution obtained at the improvement phase
$Cost(BestSolution) = Cost$
$Cost(NewSolution) = Cost(BestSolution)$
$\theta = 0$
while $\theta = 0$ **do**
    for i=1 to Max_Iteration2 **do**
        Generate a subset ($\mathcal{N}'$) randomly from design set ($\mathcal{N}$)
        for $i \in \mathcal{N}$ **do**
            if $i \in \mathcal{N}'$ **then**
                $min_{ij} = z'_{ij} - \lambda, \forall j \in \mathcal{P}$
                if $min_{ij} < 0$ **then**
                    $min_{ij} = 0$
                end if
                $max_{ij} = z'_{ij} + \lambda, \forall j \in \mathcal{P}$
                if $max_{ij} > S$ **then**
                    $max_{ij} = S$
                end if
            else
                $min_{ij} = z'_{ij}, \forall j \in \mathcal{P}$
                $max_{ij} = z'_{ij}, \forall j \in \mathcal{P}$
            end if
        end for
        $\mathcal{S}_{ij} = \{min_{ij}, \ldots, max_{ij}\}, \forall i \in \mathcal{N}, j \in \mathcal{P}$
        Solve MIP$^{\mathrm{S}}$-R to determine cost and plate designs
        if $Cost(CurrentSolution) < Cost(NewSolution)$ **then**
            $Cost(NewSolution) = Cost(CurrentSolution)$
            $z'_{ij}(NewSolution) = z'_{ij}(CurrentSolution)$
        end if
    end for
    if $CostNewSolution < Cost(BestSolution)$ **then**
        $Cost(BestSolution) = Cost(NewSolution)$
        $z'_{ij}(BestSolution) = z'_{ij}(NewSolution)$
        $\theta = 0$
    else
        $\theta = 1$
    end if
end while

3.5 Computational Study

We conduct an computational study to assess the performance of the proposed approach in terms of solution quality. We assess the performance of our proposed algorithm in comparison with the benchmark algorithms in the literature. We implement the all algorithms using JAVA and CPLEX 12.5.1 wih Concert Technology on a 64-bit Windows Server with two 2.4 Ghz Intel Xeon CPUs and 24 GB RAM. In the construction phase, we obtain a feasible solution iteratively and we fixed $Max\_Iteration1 = 50000$. In the improvement phase, we fixed $Max\_Iteration2 = 30$. We check the improvement at every 30 iterations. This improvement continues until there is no improvement after 30 iterations. In this phase, we also use another parameter which indicates the maximum amount of increase or decrease in the number of assigned slots to a design. We consider λ as 2 for the improvement phase. As the value of λ increases, the algorithm has high chance to obtain a better solution. On the other hand, it makes the MIP$^{\text{S}}$-R more complicated and increases the solution time. We obtain the best results when $Max\_Iteration1 = 50000$, $Max\_Iteration2 = 30$ and $\lambda = 2$. In the improvement phase, we set a time limit of 5 seconds when running MIP$^{\text{S}}$-R. Furthermore, we set a time limit of 180 seconds for running SCP while implementing the construction subroutine.

We generate randomly instances inspired in study [12]. For the number of designs, these instances are generated randomly by considering nine different setings: 5, 10, 15, 20, 25, 30, 50, 70, 90. For each case (the different number of designs), we generate eight different instances. We also generate instances with a low white-border ratio and high white-border ratio. The values of low white-border and high white-border ratio is 0.33 and 0.66, respectively. We give a random number between 0 and 1 to each design. If the selected number is equal to or smaller than the white-border ratio, then this design is called as a white-border design. Furthermore, we use color-code ratio (CR) as a parameter in order to estimate the maximum number of different napkin colors relative to the total number of

designs. We also generate instances with two different settings of color-code ratios: i) low color-code ratio and ii) high color-code ratio. The value of the low color-code ratio is 0.15 and the value of the high color-code ratio is 0.30. First, we calculate the maximum number of different napkin colors. To do that, we multiply the corresponding values for CR and N. Then, we round up this value. After that, we choose an integer random number for each design. This number is chosen between 1 and the computed maximum number of different napkin colors and it becomes the color-code of the specified design. Similar to the color-code ratio, we have demand ratio (DR) as a parameter. We generate instances with low demand-ratio and high demand-radio. We consider that the low demand-ratio is 0.2 and the high demand-ratio is 0.4. In order to evaluate the maximum number of different demand values, we multiply the corresponding values for DR and N. Then, we round up this value. Next, we divide the interval evenly by the number of intervals of the maximum different demand values. We choose one of these intervals randomly and round the center of this interval to the nearest multiple of 500 in order to calculate demand value. The printing plates include seven slots for each instance. We classify the instances according to their sizes and divide into two categories: small-and medium and large instances. In total, we performed our experiments on randomly generated 40 small- and medium-sized instances and 32 large-sized instances. In all instances, the overproduction cost per unit of customer-specific designs is 0.0035 and it is 0.001 per unit of standard design. The setup cost per plate is 540.

We report numerical results of the proposed heuristic algorithm, benchmark algorithms from the literature [12] and MIP^{S} in Tables 8, 9 and 10. To make a fairly comparison between the obtained results, we compare the results of the algorithms with respect to the lower bound solution in a limited time. While evaluating the performances of the proposed algorithms, we use benchmark algorithm from the literature [12] developed a heuristic algorithm consisting of two stages. In the first stage, they generate an initial feasible solution by implementing three different

strategies. Finally, they try to improve the solution with two different ideas in the second stage. We compare the proposed approaches against the results of the heuristic under the three alternative selection strategies presented in [12].

We test the performance of our two-phase approach in all instances and compare the relative performance to heuristic algorithms with three different strategies presented in [12]. The basis of comparison is the relative gap value with respect to the lower bound solution found in a limited time. We present the results for all 72 instances randomly generated for different values of parameter $\bar{c}$ and report the results in Tables 8, 9 and 10. We generate four instances for each (N, WR) tuple. We generate one instance for each combination ($CR = 0.15$, $CR = 0.30$, $DR = 0.2$, $DR = 0.4$). In Tables 8, 9 and 10, the first column is the number of customer specific designs, and, the second column is white-border ratio. The following four columns under "IIA" present the results obtained by the proposed two-phase approach. Similarly, the following six columns under the three different heuristics proposed by [12] indicate the results found by these algorithms. Finally, the following two columns under "MIP^{S}" present the best results found in a limited time. Each row of tables include four instances for each (N, WR) tuple. The numbers in each row of these tables present averages of these four instances. We test the performance of our two-phase approach in all instances and compare the relative performance to proposed algorithms in [12]. We first observe that all of the algorithms manage to find a feasible solution for each instance. According to the results, we observe that the some small- and medium-sized instances are solved to optimality in a reasonable time using the commercial solver. The average relative gap of the solutions provided by IIA is only 17.54% for the values of parameter $\bar{c}$ equals 2. For this parameter, the average relative gap of the solutions provided by three different strategies are 23.74%, 21.75% and 20.54%, respectively. When $\bar{c}$ equals 3, the average relative gap of the solution provided by IIA is only 11.95% and the average relative gap of the solutions provided by three different strategies are 20.57%, 17.35% and 17.19%, respectively. For $\bar{c}$ equals 4,

the average relative gap of the solutions for IIA is 10.03% and the average relative gap of the solutions for three different strategies are 20.13%, 15.06% and 16.42%, respectively. According to the numerical results, matching-based strategy outperforms the roulette-based and best-based strategies when $\bar{c}$ equals 2 and 3. However, roulette-based strategy outperforms the matching-based and best-based strategies for $\bar{c}$ equals 4. On the instances, we obtain much better results, and the proposed heuristic approach improves the existing algorithms proposed by [12] in terms of solution quality. For $\bar{c}$ equals 2 and 3, the proposed heuristic approach outperforms the matching-based strategy by around 3.00% and 5.24%, respectively. Also, IIA improves the roulette-based strategy solutions by around 5.03% when $\bar{c}$ equals 4.

Table 8: Comparison of the results for the randomly generated instances when $\bar{c} = 2$

Instance		**IIA**				**Baumann**						**MIP^{S}**	
		CS		**Imp**		**Saving**		**Roulette**		**Matching**			
N	***WR***	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**
5	0.33	0.00%	14	0.00%	18	0.00%	1	0.00%	1	0.00%	1	0.00%	0
5	0.66	0.00%	13	0.00%	18	0.00%	1	0.00%	1	0.00%	1	0.00%	0
10	0.33	0.00%	13	0.00%	25	0.00%	1	0.00%	1	0.00%	5	0.00%	1
10	0.66	0.00%	13	0.00%	27	0.28%	1	0.00%	1	0.00%	5	0.00%	1
15	0.33	1.74%	27	0.00%	35	5.76%	1	0.02%	1	0.00%	6	0.00%	28
15	0.66	0.00%	28	0.00%	34	9.21%	1	0.00%	1	0.00%	7	0.00%	17
20	0.33	3.28%	37	2.46%	51	13.04%	1	8.29%	1	9.44%	8	2.22%	998
20	0.66	1.03%	42	0.00%	43	10.65%	1	4.45%	1	3.88%	8	0.00%	369
25	0.33	10.93%	58	8.16%	58	18.51%	1	14.86%	1	14.29%	35	8.18%	2126
25	0.66	7.15%	59	3.88%	55	16.85%	1	7.13%	1	11.75%	34	3.88%	2677
30	0.33	14.22%	71	9.02%	93	19.85%	1	17.33%	1	13.95%	95	8.53%	2794
30	0.66	21.20%	79	16.39%	86	24.21%	1	24.86%	1	21.03%	98	16.43%	3049
50	0.33	48.39%	152	40.94%	369	41.87%	5	47.34%	6	44.49%	308	41.59%	3600
50	0.66	43.62%	162	35.07%	387	41.24%	6	41.30%	6	37.83%	304	36.38%	3600
70	0.33	57.67%	176	48.60%	639	54.23%	8	55.06%	16	53.59%	310	57.20%	3600
70	0.66	58.58%	163	49.41%	769	54.77%	8	56.87%	20	52.70%	322	58.58%	3600
90	0.33	57.57%	270	52.15%	608	57.56%	13	58.48%	43	55.18%	362	66.68%	3600
90	0.66	55.23%	253	49.61%	645	53.21%	12	55.58%	45	51.61%	344	65.39%	3600
Average		**21.15%**	**90.6**	**17.54%**	**220**	**23.74%**	**3.6**	**21.75%**	**8.2**	**20.54%**	**125.2**	**20.28%**	**1870**

Table 9: Comparison of the results for the randomly generated instances when $\bar{c} = 3$

Instance		IIA				Baumann						MIP^{S}	
		CS		Imp		Saving		Roulette		Matching			
N	***WR***	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**
5	0.33	0.00%	14	0.00%	16	0.00%	1	0.00%	1	0.00%	1	0.00%	0
5	0.66	0.00%	14	0.00%	19	0.00%	1	0.00%	1	0.00%	1	0.00%	0
10	0.33	0.00%	15	0.00%	24	2.51%	1	0.00%	1	0.00%	8	0.00%	1
10	0.66	0.00%	14	0.00%	25	0.28%	1	0.00%	1	0.00%	8	0.00%	1
15	0.33	0.00%	24	0.00%	32	13.87%	1	0.00%	1	0.00%	9	0.00%	12
15	0.66	0.00%	41	0.00%	30	11.01%	1	0.00%	1	0.00%	10	0.00%	20
20	0.33	1.27%	35	0.00%	32	11.79%	1	1.43%	1	3.46%	12	0.00%	246
20	0.66	4.16%	51	0.00%	36	8.09%	1	5.00%	1	4.46%	14	0.00%	50
25	0.33	4.95%	56	0.00%	57	5.29%	1	5.03%	1	10.50%	39	0.00%	476
25	0.66	4.57%	63	0.11%	48	14.42%	1	7.74%	1	8.35%	36	0.00%	350
30	0.33	8.12%	93	0.40%	62	16.40%	1	11.23%	1	10.94%	103	0.40%	2110
30	0.66	7.06%	78	1.79%	66	12.79%	1	9.80%	1	13.71%	107	1.79%	1627
50	0.33	35.52%	162	25.38%	351	37.14%	5	37.36%	8	32.15%	318	24.58%	3600
50	0.66	34.78%	176	24.35%	357	35.53%	4	34.81%	8	31.53%	316	22.76%	3600
70	0.33	49.62%	167	38.42%	442	47.77%	8	46.42%	22	46.05%	331	39.84%	3600
70	0.66	51.54%	162	41.15%	451	50.73%	7	50.03%	25	48.84%	314	45.30%	3600
90	0.33	53.24%	268	42.42%	651	53.56%	12	52.83%	51	50.71%	346	49.13%	3600
90	0.66	49.65%	258	41.12%	622	49.04%	10	50.62%	46	48.70%	339	48.80%	3600
Average		**16.92%**	**93.9**	**11.95%**	**184.5**	**20.57%**	**3.2**	**17.35%**	**9.6**	**17.19%**	**128.4**	**12.92%**	**1471.8**

3.6 Conclusion

In this chapter, we analyze a problem called *Cover Printing Problem with Design Restrictions* (CPP-DR) that has applications in the manufacturing area. We focus on obtaining assignment plan of a variant of CPP in order to find a good feasible solution with plate designs. We formulate a linear integer model with a valid inequality for the problem. The motivation of the problem is to minimize the total production cost by satisfying the each customer design. This problem is NP-hard problem, hence we propose a novel two-phase heuristic algorithm called Iterative Improvement Algorithm (IIA) to solve the problem. We first generate a set of several plate designs, and then we solve a set covering problem in order to select the best set of plate designs among all generated sets. In the second phase, we try to improve the solution using an improvement idea with using a restriction version of linear integer model. Finally, we test the performance of the our proposed algorithm on all randomly generated instances. The computational

Table 10: Comparison of the results for the randomly generated instances when $\bar{c} = 4$

Instance		**IIA**				**Baumann**						**MIP^{S}**	
		CS		**Imp**		**Saving**		**Roulette**		**Matching**			
N	***WR***	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**
5	0.33	0.00%	14	0.00%	15	0.00%	1	0.00%	1	0.00%	1	0.00%	0
5	0.66	0.00%	14	0.00%	16	0.00%	1	0.00%	1	0.00%	1	0.00%	0
10	0.33	0.00%	16	0.00%	23	2.51%	1	0.00%	1	0.00%	10	0.00%	1
10	0.66	0.00%	16	0.00%	25	0.28%	1	0.00%	1	0.00%	9	0.00%	1
15	0.33	0.00%	22	0.00%	30	14.53%	1	0.00%	1	0.00%	12	0.00%	7
15	0.66	0.00%	37	0.00%	39	9.38%	1	0.00%	1	0.00%	11	0.00%	11
20	0.33	5.50%	40	0.00%	35	16.25%	1	5.03%	1	4.04%	13	0.00%	106
20	0.66	6.57%	40	0.00%	51	10.15%	1	5.70%	1	7.37%	12	0.00%	39
25	0.33	5.08%	63	0.00%	56	7.90%	1	4.88%	1	10.35%	36	0.00%	217
25	0.66	5.57%	61	0.00%	63	16.75%	1	5.10%	1	12.16%	38	0.00%	269
30	0.33	8.58%	96	0.00%	94	16.36%	1	8.62%	1	10.86%	109	0.00%	389
30	0.66	7.50%	85	2.57%	79	12.74%	1	6.24%	1	9.97%	103	2.43%	2109
50	0.33	32.59%	165	21.75%	324	33.53%	4	29.11%	10	31.34%	321	18.94%	3600
50	0.66	32.37%	167	18.29%	342	33.88%	5	26.83%	11	28.89%	315	18.00%	3600
70	0.33	45.19%	171	34.88%	351	46.87%	9	44.38%	23	45.28%	341	36.70%	3600
70	0.66	45.13%	163	34.09%	357	46.71%	8	45.08%	25	44.44%	328	37.54%	3600
90	0.33	48.86%	255	35.15%	443	49.79%	11	46.64%	59	46.29%	424	39.00%	3600
90	0.66	43.83%	246	33.86%	451	44.71%	10	43.54%	57	44.63%	411	40.91%	3600
Average		**15.93%**	**92.8**	**10.03%**	**155.2**	**20.13%**	**3.3**	**15.06%**	**10.9**	**16.42%**	**138.6**	**10.75%**	**1374.9**

results show that our two-phase approach outperforms than the existing algorithms developed by study [12].

CHAPTER IV

CONCLUSION

In this thesis, we handle two production planning problems related to the publishing industry: the Label Printing Problem and Cover Printing Problem with Design Restrictions. In both problems, we concentrate on finding the best assignment plan by satisfying the demand requirements. We formulate linear integer models strengthened with valid inequalities for these problems. The motivation of the first problem is to obtain minimum waste, whereas the main goal is to minimize the total production cost by satisfying the customer demands for the second problem. Both problems are NP-hard problems, and hence we develop two different heuristic approaches which is consisted of two-phase in order to solve each problem type. We test the proposed heuristics on the real-world instances and randomly generated instances in terms of solution quality. The experimental results show that the proposed heuristic approaches give better results than the benchmark algorithms developed in the literature.

APPENDIX A

SUPPLEMENTARY

Table 11: Best solution obtained for the instances with 18 labels

T	Algorithm	Waste	j	L_j	Template Design: Label (Assigned Slots)
3	CIH_G	2.658%	1	1400	1(1) 9(1) 10(1) 13(2) 14(1) 15(1) 16(1) 17(2) 18(4)
			2	550	1(1) 3(1) 6(1) 7(1) 8(1) 9(2) 10(2) 12(1) 13(2) 15(2)
			3	200	1(2) 2(1) 4(1) 5(2) 11(2) 12(3) 15(1) 17(2)
	CIH_P	5.854%	1	1400	9(2) 12(1) 13(3) 14(1) 15(2) 16(1) 18(4)
			2	550	1(4) 3(1) 6(1) 7(1) 8(1) 17(6)
			3	273	2(1) 4(1) 5(1) 10(9) 11(2)
4	CIH_G	0.340%	1	1350	1(1) 9(1) 10(1) 13(2) 14(1) 15(1) 16(1) 17(2) 18(4)
			2	550	1(1) 6(1) 7(1) 8(1) 9(2) 10(2) 12(2) 13(2) 15(2)
			3	150	1(2) 2(1) 3(3) 4(1) 5(1) 11(2) 15(1) 17(2) 18(1)
			4	50	2(1) 3(1) 5(2) 9(1) 11(1) 12(1) 13(1) 14(1) 15(2) 16(1) 17(1) 18(1)
	CIH_P	3.243%	1	700	15(4) 16(2) 18(8)
			2	625	9(4) 10(4) 11(1) 17(5)
			3	550	1(4) 6(1) 7(1) 8(1) 13(7)
			4	288	2(1) 3(2) 4(1) 5(1) 12(4) 14(5)
5	CIH_G	0.007%	1	1350	1(1) 9(1) 10(1) 13(2) 14(1) 15(2) 17(2) 18(4)
			2	550	1(1) 6(1) 7(1) 8(1) 9(2) 10(2) 12(2) 13(2) 16(2)
			3	100	1(3) 2(2) 3(1) 5(1) 11(3) 16(3) 18(1)
			4	50	3(2) 4(2) 5(3) 9(1) 11(1) 12(1) 13(1) 14(1) 17(1) 18(1)
			5	43	3(7) 17(7)
	CIH_P	1.373%	1	700	15(4) 16(2) 18(8)
			2	550	1(4) 6(1) 7(1) 8(1) 13(7)
			3	500	3(1) 9(5) 10(5) 14(3)
			4	255	2(1) 5(1) 17(12)
			5	117	4(1) 11(3) 12(10)

Table 12: Best solution obtained for the instances with 22 labels

T	Algorithm	Waste	j	L_j	**Template Design:** Label (Assigned Slots)
3	CIH_G	4.547%	1	2050	7(2) 11(2) 12(1) 15(1) 16(1) 17(2) 18(2) 20(1) 21(1) 22(2)
			2	1138	4(1) 8(1) 10(1) 13(1) 14(1) 15(3) 16(1) 17(2) 18(2) 19(2)
			3	800	1(1) 2(1) 3(3) 5(1) 6(1) 9(1) 12(2) 16(1) 20(2) 21(1) 22(1)
	CIH_P	7.538%	1	2100	7(2) 11(2) 15(3) 16(2) 17(3) 18(3)
			2	1217	4(1) 8(1) 10(1) 12(3) 13(1) 14(1) 20(3) 22(4)
			3	800	1(1) 2(1) 3(3) 5(1) 6(1) 9(1) 19(3) 21(4)
4	CIH_G	1.126%	1	2450	3(1) 7(1) 11(1) 12(1) 15(2) 16(1) 17(2) 18(2) 20(1) 21(1) 22(2)
			2	850	4(1) 6(1) 7(2) 8(1) 9(1) 10(1) 11(2) 12(1) 16(1) 17(1) 18(1) 19(1) 20(1)
			3	350	2(2) 5(2) 13(1) 14(3) 16(1) 17(1) 18(1) 19(3) 20(1)
			4	200	1(3) 10(1) 13(3) 15(2) 16(1) 17(1) 18(1) 19(2) 21(1)
	CIH_P	3.458%	1	1350	11(3) 15(4) 16(3) 20(3) 21(2)
			2	1050	10(1) 13(1) 14(1) 17(6) 18(6)
			3	850	2(1) 3(3) 4(1) 6(1) 8(1) 9(1) 12(4) 19(3)
			4	693	1(1) 5(1) 7(6) 22(7)
5	CIH_G	0.271%	1	2392	3(1) 7(1) 11(1) 12(1) 15(2) 16(1) 17(2) 18(2) 20(1) 21(1) 22(2)
			2	858	4(1) 7(2) 8(1) 10(1) 11(1) 12(1) 13(1) 14(1) 16(1) 17(1) 18(1) 19(2) 20(1)
			3	400	1(1) 2(1) 5(1) 6(2) 9(2) 11(2) 15(1) 16(1) 17(1) 18(1) 19(1) 20(1) 20(1)
			4	100	1(2) 2(3) 5(1) 10(1) 13(1) 14(2) 16(1) 17(2) 18(1) 19(1)
			5	67	5(2) 10(1) 12(1) 15(2) 17(1) 18(2) 19(1) 21(4) 22(1)
	CIH_P	1.941%	1	1255	16(3) 18(5) 20(3) 22(4)
			2	884	4(1) 8(1) 12(4) 15(6) 21(3)
			3	820	3(3) 6(1) 7(5) 9(1) 11(5)
			4	573	17(11) 19(4)
			5	350	1(2) 2(2) 5(2) 10(3) 13(3) 14(3)

Table 13: Best solution obtained for the instance with 100 labels

T	Algorithm	Waste	j	L_j	**Template Design:** Label (Assigned Slots)
8	CIH_G	1.460%	1	79428	2(1) 4(1) 7(1) 12(1) 16(1) 17(1) 20(1) 21(1) 28(1) 30(1) 39(1) 43(1) 46(1) 47(1) 52(1) 55(1) 57(1) 61(1) 62(1) 74(1) 78(1) 83(1) 88(1) 92(1) 100(1)
			2	63945	15(1) 22(1) 23(1) 24(1) 29(1) 34(1) 35(1) 40(1) 41(1) 50(1) 51(1) 53(1) 63(1) 64(1) 66(1) 69(1) 71(1) 75(1) 80(1) 84(1) 87(1) 89(1) 94(1) 98(1) 99(1)
			3	40027	3(1) 5(1) 6(1) 8(1) 11(1) 14(1) 25(1) 26(1) 27(1) 31(1) 33(1) 37(1) 38(1) 49(1) 56(1) 59(1) 67(1) 68(1) 70(1) 72(1) 82(1) 86(1) 90(1) 95(1) 97(1)
			4	18339	1(2) 12(1) 13(1) 14(1) 16(1) 24(1) 27(1) 36(2) 43(1) 44(1) 50(1) 58(1) 59(1) 60(2) 61(1) 73(1) 77(2) 79(1) 81(1) 91(1) 97(1)
			5	16842	9(1) 10(2) 11(1) 13(1) 15(1) 18(1) 19(1) 21(1) 24(1) 45(1) 48(2) 49(1) 54(1) 72(1) 75(1) 80(1) 85(1) 92(1) 93(2) 95(1) 96(2)
			6	12607	18(1) 22(1) 26(1) 28(1) 32(1) 42(1) 44(1) 46(1) 50(1) 51(1) 53(1) 54(1) 58(1) 65(1) 66(1) 70(1) 71(1) 75(1) 76(1) 83(1) 84(1) 87(1) 88(1) 90(1) 99(1)
			7	6668	5(1) 7(1) 8(1) 23(1) 25(1) 34(1) 35(1) 39(2) 45(1) 51(2) 54(1) 56(1) 62(1) 63(1) 69(1) 74(1) 76(1) 82(1) 85(1) 86(1) 87(1) 89(1) 90(1)
			8	1760	2(1) 5(1) 6(1) 15(1) 26(1) 29(2) 32(1) 33(1) 35(1) 37(1) 38(2) 43(1) 62(1) 72(1) 74(1) 80(2) 82(2) 91(2) 92(1) 98(1)
	CIH_P	3.486%	1	40002	3(1) 4(2) 17(2) 20(2) 30(2) 31(1) 47(2) 52(2) 53(2) 55(2) 57(2) 67(1) 78(2) 100(2)
			2	38128	1(1) 13(1) 22(2) 23(2) 35(2) 36(1) 48(1) 54(1) 60(1) 63(2) 66(2) 68(1) 71(2) 77(1) 84(2) 93(1) 99(2)
			3	33116	10(1) 12(3) 16(3) 21(3) 24(3) 43(3) 61(3) 92(3) 96(1) 98(2)
			4	31190	39(3) 41(2) 46(3) 50(3) 58(1) 64(2) 75(3) 83(3) 88(3) 90(2)
			5	29987	14(2) 27(2) 28(3) 44(1) 49(2) 51(3) 59(2) 62(3) 72(2) 74(3) 97(2)
			6	27828	2(3) 5(2) 7(3) 11(2) 15(3) 18(1) 26(2) 70(2) 82(2) 87(3) 95(2)
			7	23369	8(2) 25(2) 29(3) 34(3) 38(2) 45(1) 56(2) 69(3) 85(1) 86(2) 89(3) 91(1)
			8	21026	6(2) 9(1) 19(1) 32(1) 33(2) 37(2) 40(3) 42(1) 65(1) 73(1) 76(1) 79(1) 80(4) 81(1) 94(3)

REFERENCES

[1] The World Counts, "Paper waste facts." theworldcounts.com. http://www.theworldcounts.com/stories/Paper-Waste-Facts. (accessed September 09, 2019).

[2] Y. Hsieh and P. You, "An immune evolutionary approach for the label printing problem," *International Journal of Computational Intelligence Systems*, vol. 7, no. 3, pp. 515–523, 2014.

[3] D. Tuyttens and A. Vandaele, "Towards an efficient resolution of printing problems," *Discrete Optimization*, vol. 14, no. 1, pp. 126–146, 2014.

[4] K. Yiu, K. Mak, and H. Lau, "A heuristic for the label printing problem," *Computers & Operations Research*, vol. 34, no. 9, pp. 2576–2588, 2007.

[5] A. Ekici, Ö. Ergun, P. Keskinocak, and M. Lagoudakis, "Optimal job splitting on a multi-slot machine with applications in the printing industry," *Naval Research Logistics*, vol. 57, no. 3, pp. 237–251, 2010.

[6] J. Teghem, M. Pirlot, and C. Antoniadis, "Embedding of linear programming in a simulated annealing algorithm for solving a mixed integer production planning problem," *Journal of Computational and Applied Mathematics*, vol. 64, no. 1-2, pp. 91–102, 1995.

[7] S. Elaoud, J. Teghem, and B. Bouaziz, "Genetic algorithms to solve the cover printing problem," *Computers & Operations Research*, vol. 34, no. 11, pp. 3346–3361, 2007.

[8] D. Tuyttens and A. Vandaele, "Using a greedy random adaptative search procedure to solve the cover printing problem," *Computers & Operations Research*, vol. 37, no. 4, pp. 640–648, 2010.

[9] D. Romero and F. Alonso-Pecina, "Ad hoc heuristic for the cover printing problem," *Discrete Optimization*, vol. 9, no. 1, pp. 17–28, 2012.

[10] F. Alonso-Pecina and D. Romero, "A hybrid simulated annealing/linear programming approach for the cover printing problem." *Mathematical Problems in Engineering, Article ID 6193649, 11 pages*, 2018.

[11] S. Mohan, S. Neogy, A. Seth, N. Garg, and S. Mittal, "An optimization model to determine master designs and runs for advertisement printing," *Journal of Mathematical Modelling and Algorithms*, vol. 6, no. 2, pp. 259–271, 2007.

[12] P. Baumann, S. Forrer, and N. Trautmann, "Planning of a make-to-order production process in the printing industry," *Flexible Services and Manufacturing Journal*, vol. 27, no. 4, pp. 534–560, 2015.

[13] D. Groß, H. Hamacher, S. Horn, and A. Schobel, "Stop location design in public transportation networks: covering and accessibility objectives," *TOP*, vol. 17, pp. 335–346, 2009.

[14] P. Pusztai, "An application of the greedy heuristic of set cover to traffic checks," *Central European Journal of Operations Research*, vol. 16, no. 4, pp. 407–414, 2008.

[15] Z. Degraeve and M. Vandebroek, "A mixed integer programming model for solving a layout problem in the fashion industry," *Management Science*, vol. 44, no. 3, pp. 301–310, 1998.

[16] Z. Degraeve, W. Gochet, and R. Jans, "Alternative formulations for a layout problem in the fashion industry," *European Journal of Operational Research*, vol. 143, no. 1, pp. 80–93, 2002.

[17] J. Martens, "Two genetic algorithms to solve a layout problem in the fashion industry," *European Journal of Operational Research*, vol. 154, no. 1, pp. 304–322, 2004.

[18] C. L. Yang, R. H. Huang, and H. Huang, "Elucidating a layout problem in the fashion industry by using an ant optimisation approach," *Production Planning & Control*, vol. 32, no. 3, pp. 248–256, 2011.

[19] D. M. Rose and D. R. Shier, "Cut scheduling in the apparel industry," *Computers & Operations Research*, vol. 34, no. 11, pp. 3209–3228, 2007.

[20] R. M'Hallah and A. Bouziri, "Heuristics for the combined cut order planning two-dimensional layout problem in the apparel industry," *International Transactions in Operational Research*, vol. 23, no. 1-2, pp. 321–353, 2016.

[21] R. M'Hallah and A. Bouziri, "Hybrid heuristics for the cut ordering planning problem in apparel industry," *Computers & Industrial Engineering*, vol. 144, no. 1-2, p. 106478, 2020.

[22] S. Al-Shihabi, M. Arafeh, and M. Barghash, "An improved hybrid algorithm for the set covering problem," *Computers & Industrial Engineering*, vol. 85, pp. 328–334, 2015.

VITA

Emre Çankaya received the B.Sc degree and M.Sc degree in industrial engineering from Özyeğin University. In February, 2017, he started Ph.D. Program in Graduate School of Engineering at Özyeğin University, in Istanbul. He worked as a teaching and research assistant throughout his M.Sc. and PH.D. degrees. His research areas include production planning, mathematical models and logistic systems.