Heuristics for the Cutting Stock with Setup Cost and Blood Collection Problems

A Dissertation

Presented to

the Faculty of the Department of Industrial Engineering

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

in Industrial Engineering

by

Azadeh Mobasher

May 2013

Heuristics for the Cutting Stock with Setup Cost and Blood Collection Problems

Azadeh Mobasher

Approved:

Chairman of the Committee
Ali Ekici, Assistant Professor,
Industrial Engineering

Committee Members:

Gino Lim,
Associate Professor,
Industrial Engineering

Eylem Tekin,
Instructional Associate Professor,
Industrial Engineering

Basheer Khumawala,
Professor,
Bauer College of Business

Funda Sahin,
Associate Professor,
Bauer College of Business

Suresh K. Khator, Associate Dean
Cullen College of Engineering

Gino Lim, Associate Professor and
Chairman, Industrial Engineering

Acknowledgements

I wish to take this opportunity to express my appreciation for numerous individuals who made my years in Houston a rewarding experience. I am very grateful to my dissertation supervisor, Prof. Ali Ekici, for giving me the opportunity and freedom to pursue my research under his supervision. Interacting with him as his student and advisee I benefited tremendously from his numerous qualities as a mentor and collaborator.

I also wish to thank the other members of my dissertation committee: Professors Gino Lim, Eylem Tekin, Basheer Khumawala, and Funda Sahin for having accepted to take the time out of their busy schedules to read my dissertation and to provide me with their comments and suggestions.

I have been fortunate to associate with many friends and family members during the time I spent in Houston. In fact, there are too many to list in the acknowledgements but I am nevertheless truly grateful. However, in particular, I would like to thank Maryam Aliakbari, Elham Mortazavi, Forozan Shaghaghi, Masoumeh Rajabi, Mahdieh Samea, and my dearest grandmother, Fatemeh Amizadeh for their unlimited kindness and prayers. Their love and memories kept me sane and helped me throughout my graduate life. I will always remember their support, friendship, and love.

Last, and most importantly, I am forever indebted to my parents, Aman Mobasher and Ashraf Khalilpour, my brother, Amin Mobasher, and my sisters, Azin and Arezou Mobasher. Their invaluable and relentless support, encouragement, and love are without doubt the most important reasons for my success. I could not have achieved this without their unlimited sacrifice.

My parents, the angels of my life, bore me, raised me, supported me, taught me,

and loved me. They are the reasons I could make it this far in my life. To them I dedicate this dissertation.

To my dear parents

Aman Mobasher & Ashraf Khalilpour

with all my love

Heuristics for the Cutting Stock with Setup Cost and Blood Collection Problems

An Abstract

of a

Dissertation

Presented to

the Faculty of the Department of Industrial Engineering

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

in Industrial Engineering

by

Azadeh Mobasher

May 2013

Abstract

The applications of Operations Research techniques enable decision makers to come up with better and more efficient decisions for a wide range of problems in different industries. In this dissertation, two different problems are studied: (i) the Cutting Stock Problem with Setup Cost, and (ii) Integrated Collection and Appointment Scheduling Problem. Both problems are analyzed and solution approaches are developed using Operations Research methodologies.

The first problem addresses the Cutting Stock Problem with Setup Cost (CSP-S), which is a more general case of the well-known Cutting Stock Problem (CSP). In CSP, one wants to minimize the stock items used while satisfying the demand for smaller-sized items. However, the number of patterns/setups to be performed on the cutting machine is ignored. In CSP-S, different cost factors for the material and the number of setups are considered, and the objective is to minimize the total production cost including both material and setup costs.

In Chapter 1, a mixed integer linear programming model, two local search algorithms, a pattern pool based approach and a column generation based heuristic algorithm are proposed for CSP-S. The effectiveness of the proposed algorithms is demonstrated on the instances from the literature.

In Chapters 2 and 3, the second problem is introduced, which is motivated by the processing requirements of the donated whole blood. This problem is called the Integrated Collection and Appointment Scheduling Problem (ICASP). The blood products have limited life time and are required to be collected from donation centers and delivered to a processing center for platelet extraction within a certain amount of time.

In Chapter 2, a mixed integer programming model is developed for ICASP. Since

the problem under consideration is NP-hard, an integer programming based algorithm and a construction based heuristic approach are proposed to find a good feasible solution. In Chapter 3, we assume that the number of scheduled donors that are showing up for donation is uncertain. Robust optimization and Chance Constrained Programming are utilized to account for the uncertainty of data. The effect of uncertainty is demonstrated on the instances from the Gulf Coast Regional Blood Center located in Houston, TX.

Table of contents

Acknowledgements **iv**

Abstract **viii**

Table of contents **x**

List of Figures **xiii**

List of Tables **xiv**

Chapter 1 Heuristics for the Cutting Stock Problem with Setup Cost 1

1.1 Introduction 1

1.2 Literature Review 2

1.3 Problem Definition 6

1.4 Mathematical Formulations 9

1.5 Complexity and A Special Case 12

1.6 Heuristic Algorithms 14

1.6.1 Local Search Algorithms 15

1.6.2 Column Generation Based Heuristic Algorithm 24

1.6.3 Pattern Pool Based Approach 27

1.7 Computational Results 28

1.8 Conclusion 32

Chapter 2 Integrated Collection and Appointment Scheduling Problem 34

2.1 Introduction 34

2.2 Literature Review 37

2.3 Problem Definition 39
2.4 A Mixed Integer Linear Programming Model 42
2.4.1 Tour-Related Constraints 43
2.4.2 Arrival Time Constraints 44
2.4.3 Collection Amount Constraints 45
2.4.4 Non-negativity and Integrality Constraints 47
2.5 Heuristic Approaches 47
2.5.1 Clustering Phase 47
2.5.2 Integer Programming Based Algorithm 48
2.5.3 Construction Based Heuristic Algorithm 49
2.6 Computational Results 54
2.7 Summary 57

Chapter 3 Robust Optimization and Chance Constrained Programming for the Collection Problem with Uncertainty 59

3.1 Introduction and Literature Review 59
3.2 Problem Definition and Mathematical Models 61
3.3 Robust Optimization 62
3.4 Chance Constrained Programming 66
3.5 Computational Results 67
3.6 Summary 74

Chapter 4 Conclusion and Future Work 76

4.1 Current Findings 76
4.1.1 Cutting Stock Problem with Setup Cost 76
4.1.2 Integrated Collection and Appointment Scheduling Problem 77

4.2 Future Research Directions 78
4.2.1 Cutting Stock Problem with Setup Cost 78
4.2.2 Integrated Collection and Appointment Scheduling Problem 79

References **81**

List of Figures

Figure 1.1 (a) CSP, (b) PMP, and (c) CSP-S solutions for the example with cost setting $C_s = \$100, C_p = \$1$. 7

Figure 1.2 Comparison of SKBA and SHPC solutions over 40 instances for different cost settings: (i) $C_s = \$100,000, C_p = \$1$, (ii) $C_s = \$10,000, C_p = \$1$, (iii) $C_s = \$1,000, C_p = \$1$, and (iv) $C_s = \$100, C_p = \$1$. 19

Figure 2.1 An example illustrating the importance of synchronizing the appointment and pickup schedules. 41

List of Tables

Table 1.1 Demand and length information for the demand items in the example. 6

Table 1.2 Solutions found by different problems for the example under different cost settings. 8

Table 1.3 Average CPU time and optimality gap of the solutions found by MPGA, SPGA, MKBA and SKBA. 30

Table 1.4 Average CPU time and optimality gap of the solutions found by LSE, LSH, CGA, PPBA, CSP, PMP and CGV. 31

Table 1.5 Average optimality gap of the solutions found by LSE, LSH, CGA, PPBA, CSP, PMP and CGV over 30 instances for which a feasible solution is found by CGV. 32

Table 2.1 Comparison gaps of solutions found by ICASP–MIP, IPBA and CBHA for 30 scenarios. 56

Table 3.1 Comparison gaps of solutions found by RICASP-C, RIPBA-C for the six deviation scenarios of 30 instances. 70

Table 3.2 Comparison gaps of solutions found by RICASP-B, RIPBA-B for the six deviation scenarios of 30 instances. 71

Table 3.3 Comparison gaps of solutions found by RICASP-E, RIPBA-E for the six deviation scenarios of 30 instances. 72

Table 3.4 Comparison d-gaps of solutions found by RIPBA-C, RICASP-C, RIPBA-B, RICASP-B, RIPBA-E, RICASP-E for the 30 instances. . . 73

Table 3.5 Comparison gaps of solutions found by ICASP-CCP, IPBA-CCP with different alpha values. 74

Chapter 1 Heuristics for the Cutting Stock Problem with Setup Cost

1.1 Introduction

The classical *Cutting Stock Problem* (CSP) addresses the problem of cutting stock materials of length W in order to satisfy the demand of smaller pieces with demand d_i and length w_i while minimizing the trim loss. CSP is first introduced by Kantorovich [36] and studied in one or more dimensions to deal with real-world applications in various industries such as the glass, fiber, paper and steel industries [21]. The main objective in CSP is to minimize total waste/material cost, and the number of setups (different cutting patterns) are usually ignored. However, in real world applications, the number of cutting patterns in a production plan has to be considered since each cutting pattern requires an additional setup on the cutting machine. This is especially important when doing the setup on the cutting machine is time-consuming and/or costly. For example, Chien and Deng [14] mention the importance of both material and setup cost in semiconductor industry, and several authors focus on minimizing the setup and waste cost simultaneously [8, 16, 19, 23, 34, 56].

In this chapter, we study a general version of the one-dimensional CSP, in which we try to minimize total production cost (setup + material cost) while cutting stock items of length W into demand items of smaller size (w_i) in order to satisfy the demand (d_i) for each item. Different from the classical CSP, in this problem the total production cost includes both material cost and setup cost. Every time a new cutting pattern is used, a setup cost is incurred to set up the cutting machine. We call this problem the *Cutting Stock Problem with Setup Cost* (CSP-S). CSP-S unifies a wide domain of packing/cutting stock problems. Note that CSP-S reduces to CSP when the setup cost is assumed to be zero. It reduces to the well-known *Bin Packing Problem* (BPP)

[38, 43, 52] when material cost is negligible and demand of each item is 1. Finally, CSP-S reduces to *Pattern Minimization Problem* (PMP) [49, 55] which focuses on minimizing the number of different cutting patterns with a given number of stock items to be cut to satisfy the demand when the material cost dominates the setup cost. According to the typology by Dyckhoff [22], CSP-S belongs to type 1/V/I/R, which means that it is a one-dimensional problem with an unlimited supply of stock materials of the same size and a set of demand items. According to the typology by Waescher et al. [58], CSP-S is an input (value) minimization assignment problem which has a weakly heterogeneous assortment.

The remainder of the chapter is organized as follows. We discuss the related work in the literature in Section 1.2. In Section 1.3, we provide a formal definition of the problem, and present an example to illustrate the problem. A mixed integer non-linear assignment formulation and a mixed integer linear model are presented in Section 1.4. In Section 1.5, we discuss the complexity of the problem and analyze a special case of the problem. We propose two local search algorithms, a pattern pool based approach and a column generation based heuristic algorithm for CSP-S in Section 1.6. We conduct computational experiments to compare the performance of the proposed algorithms and the other methods in the literature. The results are presented in Section 1.7. Conclusion is provided in Section 1.8.

1.2 Literature Review

The most related problems in the literature are *Cutting Stock Problem* (CSP) and *Pattern Minimization Problem* (PMP). Several heuristic and exact algorithms are developed to solve CSP and PMP. Gilmore and Gomory [27, 28] formulate CSP as an integer program where the decision variables are the number of times each cutting pattern is used. In CSP, when the number of demand items increases, the possible number of patterns and decision variables increase exponentially. Therefore, Gilmore

and Gomory propose a method where they first create the useful cutting patterns by solving an auxiliary problem, then they solve the linear programming (LP) relaxation and apply rounding procedure to a typically non-integer solution. This heuristic approach often results in an optimal integer solution. This algorithm is also known as *Column Generation* algorithm which is comprehensively explained in [25]. Valério de Carvalho [51] develops a combination of column generation algorithm and a branch-and-bound procedure to solve CSP. He formulates the problem as an arc flow model with some side constraints which also provides a better lower bound. Similar to the column generation algorithm developed by Gilmore and Gomory [28], the new model can be split into a master and a subproblem. At each node of the branch-and-bound tree, he first introduces branching constraints in the master problem and then generates columns. Instead of generating only one attractive arc (pattern), arc flow model introduces a set of arcs corresponding to valid cutting patterns which accelerates the column generation procedure.

Belov and Scheithauer [6] propose an exact cutting plane algorithm in combination with column generation to address CSP when stock items have different sizes. In general, a knapsack problem is solved to generate a new column [28]. In the proposed algorithm, a modified heuristic method is used to find new columns which is generally more difficult than solving a knapsack problem. Belov [7] discusses the known models of CSP and proposes several solution approaches like branch-and-price method and delayed pattern generation heuristic algorithm. Waescher and Gau [57] first discuss existing heuristic and exact methods to solve CSP by generating integer patterns and determining the production frequencies. Then, they propose several heuristic approaches to generate integer patterns for a relaxation of an integer model known as *Complete-Cut Model* [27, 28, 45]. The heuristics are tested on randomly generated instances in comparison with optimal solutions which are found using the proposed lower and upper bounds on the optimal solution.

In general, PMP can be defined as minimizing the number of different cutting patterns while satisfying the demand with a given number of stock items [55]. However, most of the authors use CSP solution to determine the number of stock items, say K, and look for a solution with minimum number of different patterns using K stock items. McDiarmid [39] proves that PMP is strongly NP-hard even for a special case where only two items fit into a stock item but none of the three do, and proposes a heuristic approach to find packings of balanced subsets in order to solve this special case of PMP. Alves and Valŕio de Carvalho [2] develop a branch-and-price-and-cut algorithm after formulating PMP using an arc flow idea. They use dual feasible functions and linear combinations of added constraints to derive new valid inequalities and strengthen the bounds of the column generation algorithm. Vanderbeck [55] develops a mixed integer formulation for PMP and solves it using a column generation approach with linear master problem and non-linear subproblem which is decomposed into bounded knapsack problems. In addition, a branch-and-bound procedure with several super-additive inequalities to tighten the feasible region is proposed. Belov [7] modifies Vanderbeck's compact mixed integer formulation to strenghten the LP relaxation bound and then proposes a simple non-linear model for PMP. His method is based on using an enumerative scheme for LP relaxation model. Umetani et al. [49] propose a heuristic algorithm, called *Iterated Local Search with Adaptive Pattern Generation* (ILS-APG), to minimize the number of different cutting patterns used in CSP. They first assume that the number of patterns is fixed, say K patterns, and then search for a solution with a minimum quadratic objective which shows the deviation of production from demand. Afterwards, they try to minimize K iteratively using ILS-APG.

Foerster and Waescher [24] propose a heuristic method to minimize the number of different patterns for minimum waste solution. This heuristic approach has two steps: (i) finding a minimum waste solution regardless of setup cost, and (ii) combining any

p patterns and substitute them with $p-1$ patterns for $p \leq 4$ using a method called KOMBI. Cui et al. [16] present a sequential heuristic algorithm, called *Sequential Heuristic Procedure - Cui* (SHPC), to solve PMP. This algorithm generates a cutting pattern using a subset of unassigned items, decides the production frequency and repeats until all items are assigned. Cui and Liu [17] modify SHPC by introducing two candidate sets for pattern selection. The first set of candidate items can be used for finding a new pattern, and the second set includes all the previously generated patterns. The final set of patterns are chosen considering the total material and setup cost. Yanasse and Limeira [60] propose a hybrid scheme to heuristically reduce the number of different patterns in CSP with any dimension. They first generate patterns and select good patterns using some predefined criteria and update the set of items whose demands are not satisfied. Then, they solve the reduced problem using the LP relaxation. Fractional values are rounded down to nearest integer, and the remaining demand is satisfied using *First Fit Decreasing* packing algorithm [35]. Finally, they apply the KOMBI heuristic proposed by Foerster and Waescher [24] to combine several patterns.

Finally, Farley and Richardson [23], Walker [56], Haessler [34], Belov and Scheithauer [8] also study problems related to CSP-S where both material and setup costs are considered. Farley and Richardson [23] introduce this problem as a subset of the general fixed-charge problem to solve the two-dimensional trim-loss problem in the glass industry. In Farley and Richardson [23], similar to Walker [56], an initial solution for a linear model of CSP with fixed charge per setup is produced. Then, a new pattern is allowed into basis (using simplex method), only if the total production cost is decreased. Afterwards, a heuristic algorithm is used to improve the solution by swapping cutting patterns. They use the algorithm by Gilmore and Gomory [29] to generate the patterns throughout the algorithm. Haessler [34] introduces a mixed integer pattern-based formulation to solve the one dimensional fixed-charge problem

with lower and upper bounds on the customer order requirements. He proposes a sequential heuristic approach and uses manually solved small instances to evaluate the performance of the proposed approach. Belov and Scheithauer [8] design a sequential heuristic approach to minimize number of input stock materials and present the effectiveness of this algorithm to minimize the number of different cutting patterns and the maximum number of open stacks during the cutting process.

1.3 Problem Definition

In this section, we first provide a formal definition of the problem, and then, give an example to illustrate the differences between CSP, PMP, and CSP-S. In CSP-S, we have stock items of length W and a set of demand items with given length (w_i) and demand (d_i) values. We use $I = \{1, \ldots, n\}$ to denote the set of demand items. We satisfy the demand for each item by cutting stock items according to some cutting patterns. For each cutting pattern, we incur a setup cost of C_s. Finally, C_p denotes the cost of a stock item per unit length. Our objective is to satisfy the demand with minimum total production cost (setup + material cost). For example, if we satisfy the demand by cutting K stock items using S different patterns, then the total cost will be $C_s S + C_p KW$.

To further explain the problem and illustrate the differences between CSP, PMP and CSP-S, we provide a simple example. In this example, we have 3 demand items and enough number of stock items with length 10. The demand and size information for the demand items is given in Table 1.1.

Table 1.1: Demand and length information for the demand items in the example.

Item	**1**	**2**	**3**
d_i	8	7	6
w_i	4	5	6

In Figure 1.1, we provide CSP, PMP and CSP-S solutions when material cost per unit length (C_p) is \$1 and unit setup cost (C_s) is \$100. In order to find optimal CSP solution, we use the mixed integer linear model presented by Kantorovich [36]. To find the PMP solution, we linearize Vanderbeck's compact formulation [55], details of which are explained in Section 1.4. Finally, CSP-S solution is found by solving the mixed integer linear formulation presented in Section 1.4 to optimality. In Figure 1.1, numbers on the patterns show the item indices, and the production frequency of a pattern is provided on the left side of each pattern.

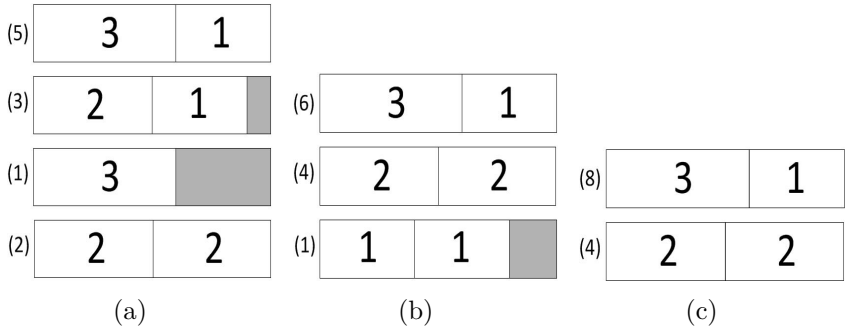

Figure 1.1: (a) CSP, (b) PMP, and (c) CSP-S solutions for the example with cost setting $C_s = \$100, C_p = \$1$.

If we solve CSP to minimize the total material cost by ignoring the setup cost, 4 different cutting patterns are used, 11 stock items are cut, and the total production cost is \$510 (see Figure 1.1(a)). In PMP, we intend to minimize total production cost while minimizing the number of setups has the second priority. While solving PMP, Vanderbeck [55] sets an upper bound (K) on the number of stock items that are used to satisfy the demand, and minimizes the number of different cutting patterns while satisfying the demand with K or less stock items. In general, K is set to CSP solution. However, it can be set to higher values to find the trade-off between material and setup cost as mentioned by Vanderbeck [55]. When we set K to 11 (CSP solution), PMP solution is the same as CSP solution with a total cost of \$510. However, if we set K to a larger number such as 21 ($= \sum_{i=1}^{3} d_i$), the final solution uses

14 stock items and 3 different cutting patterns with total cost of \$440. Note that increasing K beyond 21 does not change the solution since it is an upper bound on the number of stock items cut in any solution. Vanderbeck [55] is trying to satisfy the demand exactly, and thus, he is using equality for demand satisfaction constraints without allowing excess production. We can further improve solutions for PMP by allowing excess production. This approach helps us to find a better solution with 11 stock items and 3 different patterns and total production cost of \$410 (see Figure 1.1(b)). When we solve CSP-S to find a solution that minimizes both material and setup cost, 2 different cutting patterns are used, 12 stock items are cut, and the total production cost is \$320 (see Figure 1.1(c)), which is the minimum of all solutions.

We consider different cost settings as well, and provide a summary of the solutions for different problems in Table 1.2. Since CSP solutions do not depend on the cost setting, the solutions do not change under different cost settings. We see that CSP-S performs better when the setup cost is high and equally same as PMP when the setup cost is lower. This is reasonable since PMP is specifically designed for the low setup cost settings. On this example, we see that CSP-S is a better representation of the problem when there are separate setup and material costs. Moreover, CSP-S solution converges to PMP (or CSP) solution as the material cost increases.

Table 1.2: Solutions found by different problems for the example under different cost settings.

Cost Setting	Problem	# of Stock Items	# of Patterns	Total Cost (\$)
$C_s = \$100, C_p = \$1$	CSP	11	4	510
	PMP	11	3	410
	CSP-S	12	2	**320**
$C_s = \$100, C_p = \$5$	CSP	11	4	950
	PMP	11	3	850
	CSP-S	12	2	**800**
$C_s = \$100, C_p = \$100$	CSP	11	4	11,400
	PMP	11	3	**11,300**
	CSP-S	11	3	**11,300**

1.4 Mathematical Formulations

In the literature, different mathematical formulations are proposed for CSP, which can be divided into four categories: (i) pattern-based formulations, (ii) assignment models, (iii) one-cut formulations, and (iv) arc flow models [1]. In this section, we provide an assignment-type mixed integer non-linear formulation for the CSP-S, and then develop a mixed integer linear model by linearizing the non-linear formulation. We use this formulation to find optimal solutions or lower bounds which are used to evaluate the performance of the heuristic algorithms.

We start with an observation about the number of patterns/setups in the optimal solution. The number of setups in an optimal solution for CSP-S is not greater than the number of setups in an optimal solution for CSP. This is because the setup cost is not considered in CSP, and the only objective is to minimize the total material cost. Therefore, any solution with more patterns compared to the CSP solution will not decrease the number of stock items cut to satisfy the demand. Hence, while developing a mathematical model for the problem, we assume that the solution has at most $\mathcal{N}$ number of different patterns which is found by solving the CSP. We use $J = \{1, 2, \ldots, \mathcal{N}\}$ to denote the set of patterns. We can formulate CSP-S as a mixed integer non-linear program (MINLP) by using the following decision variables:

$$X_j = \text{Number of times pattern } j \text{ is used} \qquad \forall j \in J$$

$$P_{ij} = \text{Number of times item } i \text{ is cut in pattern } j \qquad \forall i \in I,\ j \in J$$

$$Y_j = \begin{cases} 1, & \text{if pattern } j \text{ is used} \\ 0, & \text{otherwise} \end{cases} \qquad \forall j \in J$$

The proposed formulation is as follows:

$$\text{MINLP: Minimize} \quad C_s \sum_{j \in J} Y_j + C_p W \sum_{j \in J} X_j, \tag{1.1}$$

s.t.

$$\sum_{j \in J} P_{ij} X_j \geq d_i, \qquad \forall i \in I, \tag{1.2}$$

$$\sum_{i \in I} P_{ij} w_i \leq W Y_j, \qquad \forall j \in J, \ (1.3)$$

$$X_j \in \mathbb{N}, \qquad \forall j \in J, \ (1.4)$$

$$P_{ij} \in \mathbb{N}, \qquad \forall i \in I, \ j \in J, \ (1.5)$$

$$Y_j \in \{0, 1\}, \qquad \forall j \in J. \ (1.6)$$

Equation (1.1) is the objective function which is the summation of total setup cost and total material cost. For each item i, constraints (1.2) make sure that demand is satisfied. Constraints (1.3) ensure that the total length of items cut in a pattern does not exceed the length of the stock item. These constraints also determine whether a pattern is used or not. Finally, constraints (1.4-1.6) represent the integrality and non-negativity of decision variables. Here, we use $\mathbb{N}$ to denote the set of natural numbers including 0.

This mathematical model is not easy to solve due to non-linearities. However, we can linearize this formulation by introducing some new variables and utilizing the method proposed by Glover [30]. First, we provide a result about how to linearize a multiplication of two positive integer variables if there exists an upper bound on one of the variables. We assume that we have a constraint of the form $xy \geq C$ where x and y are positive integer decision variables and C is a constant. If there exists an upper bound for x, i.e., $x \leq B$ for some constant B, we can rewrite x as a linear function of binary variables. Using new binary variables, say z_t, x can be written as $x = \sum_{t=0}^{\lfloor \log_2 B \rfloor} 2^t z_t$. Now the original constraint is equivalent to $\sum_{t=0}^{\lfloor \log_2 B \rfloor} 2^t z_t y \geq C$. When we have a multiplication of a positive integer and binary variable ($z_t y$), we can replace it with a new positive variable, say n_t, and introduce the following three constraints (i) $n_t \leq y$, (ii) $n_t \leq M z_t$, and (iii) $n_t \geq y + M(z_t - 1)$ where M is a large constant. As a result, $xy \geq C$ can be replaced by the following four linear constraints: (i) $\sum_{t=0}^{\lfloor \log_2 B \rfloor} 2^t n_t \geq C$, (ii) $n_t \leq y$, (iii) $n_t \leq M z_t$, and (iv) $n_t \geq y + M(z_t - 1)$ where n_t's are positive variables and z_t's are binary variables.

Using the procedure explained above, we linearize constraints (1.2). We know that P_{ij} is limited by $\lfloor \frac{W}{w_i} \rfloor$. Therefore, we can reformulate these non-linear constraints by first introducing new binary variables E_{ijk} for each $i \in I, j \in J, k \in K_i = \{0, 1, \ldots, \lfloor \log_2 \lfloor \frac{W}{w_i} \rfloor \rfloor\}$. Then, P_{ij} is equivalent to $\sum_{k \in K_i} 2^k E_{ijk}$ and constraints (1.2) become $\sum_{j \in J} \sum_{k \in K_i} 2^k E_{ijk} X_j \geq d_i$. After introducing $Z_{ijk} = E_{ijk} X_j$ as new positive variables, we come up with the following mixed integer linear program (MILP):

$$\text{MILP: Minimize } C_s \sum_{j \in J} Y_j + C_p W \sum_{j \in J} X_j, \tag{1.7}$$

s.t.

$$\sum_{j \in J} \sum_{k \in K_i} 2^k Z_{ijk} \geq d_i, \qquad \forall i \in I, \tag{1.8}$$

$$\sum_{i \in I} \sum_{k \in K_i} 2^k E_{ijk} w_i \leq W Y_j, \qquad \forall j \in J, \tag{1.9}$$

$$Z_{ijk} \leq X_j, \qquad \forall i \in I,\ j \in J,\ k \in K_i, \tag{1.10}$$

$$Z_{ijk} \leq M E_{ijk}, \qquad \forall i \in I,\ j \in J,\ k \in K_i, \tag{1.11}$$

$$Z_{ijk} \geq X_j + M(E_{ijk} - 1), \qquad \forall i \in I,\ j \in J,\ k \in K_i, \tag{1.12}$$

$$X_j \in \mathbb{N}, \qquad \forall j \in J, \tag{1.13}$$

$$Z_{ijk} \geq 0, \qquad \forall i \in I,\ j \in J,\ k \in K_i, \tag{1.14}$$

$$Y_j \in \{0, 1\}, \qquad \forall j \in J, \tag{1.15}$$

$$E_{ijk} \in \{0, 1\}, \qquad \forall i \in I,\ j \in J,\ k \in K_i. \tag{1.16}$$

In MILP, constraints (1.8), (1.10), (1.11), and (1.12) are the linearized versions of constraints (1.2). Constraints (1.9) are the new versions of constraints (1.3) after replacing P_{ij} with $\sum_{k \in K_i} 2^k E_{ijk}$. Finally, constraints (1.14) and (1.16) are the integrality and nonnegativity restrictions on the new variables.

MINLP is partly developed based on the assignment models by Kantorovich [36] and Vanderbeck [55]. Kantorovich [36] is the first to propose an integer linear assignment formulation for CSP using an upper bound, say K, on the maximum number of stock items. Valério de Carvalho [53] and Vance [54] report that the branch-and-bound algorithms based on the Kantorovich's model fail to solve some instances optimally because of weak LP relaxation bound. Recently, Vanderbeck [55] presents

the following compact formulation for PMP which is similar to MINLP. We will use the linearized version of this formulation (using the procedure explained above) for solving PMP.

$$\text{Minimize} \quad \sum_{k=1}^{K} Y_k, \tag{1.17}$$

s.t.

$$\sum_{k=1}^{K} X_k P_{ik} = d_i, \qquad \forall i \in I, \tag{1.18}$$

$$\sum_{k=1}^{K} X_k \leq K, \tag{1.19}$$

$$X_k \leq K Y_k, \qquad \forall k \in \{1, 2, \ldots, K\}, \tag{1.20}$$

$$\sum_{k=1}^{K} w_i P_{ik} \leq W Y_k, \qquad \forall k \in \{1, 2, \ldots, K\}, \tag{1.21}$$

$$P_{ik} \in \mathbb{N}, \qquad \forall i \in I,\ k \in \{1, 2, \ldots, K\}, \tag{1.22}$$

$$Y_k \in \{0, 1\}, \qquad \forall k \in \{1, 2, \ldots, K\}, \tag{1.23}$$

$$X_k \in \mathbb{N}, \qquad \forall k \in \{1, 2, \ldots, K\}. \tag{1.24}$$

In this formulation, K is set to the minimum number of stock items required to satisfy the demand which can be found by solving CSP. However, as mentioned by Vanderbeck, it can be set to higher values to examine the trade-off between material and setup cost minimization as well. The main drawbacks of this compact model are the weak LP relaxation bound [55] and constraints (1.18). Since the goal in Vanderbeck's model is to satisfy the demand exactly without allowing excess production, constraints (1.18) may increase the material usage and the number of different patterns used.

1.5 Complexity and A Special Case

In this section, we discuss the complexity of CSP-S and analyze a special case. We use the algorithm developed for this special case to develop heuristic algorithms

for the general problem.

Proposition 1.5.1. *CSP-S is strongly NP-hard.*

McDiarmid [39] proves that PMP is strongly NP-hard. Proposition (1.5.1) follows from this result since PMP is a special case of CSP-S where material cost dominates setup cost.

Next, we analyze a special case of CSP-S where setup cost is dominating and summation of sizes of all the items is smaller than or equal to the length of the stock item, i.e., $\sum_{i \in I} w_i \leq W$. We call this special case, *Single Setup Problem*. In this case, there is only one setup/pattern in the optimal solution since we look for a solution with minimum number of setups. The only remaining thing to be determined is the cutting pattern of this setup. We propose Algorithm 1, called *Single Setup Algorithm* (SSA), to find the optimal solution. The proposed algorithm runs in pseudo-polynomial time ($O(\frac{W}{\min_{i \in I} w_i})$). We know that each item appears in the cutting pattern at least once. Then, the main idea is to increase the number of appearances for the item that has the largest ratio of demand divided by the number of appearances. When the algorithm terminates, P_i determines the number of times item i appears in the cutting pattern and L determines the number of stock items to be cut.

Algorithm 1 *Single Setup Algorithm* (SSA)

1: $P_i = 1$ for all $i \in I$
2: $RemainingLength = W - \sum_{i \in I} w_i$
3: $ChosenItem = \text{argmax}_{i \in I} \lceil \frac{d_i}{P_i} \rceil$
4: **if** $w_{ChosenItem} \leq RemainingLength$ **then**
5: $\quad P_{ChosenItem} = P_{ChosenItem} + 1$
6: $\quad RemainingLength = RemainingLength - w_{ChosenItem}$
7: $\quad$ Go to Step 3
8: **else**
9: $\quad$ Set $L = \max_{i \in I} \lceil \frac{d_i}{P_i} \rceil$
10: **end if**

Theorem 1.5.2. *Algorithm 1 finds an optimal solution for the Single Setup Problem.*

Proof. We do the proof by contradiction. Assume that the solution found by the algorithm is not optimal, and let L' be the number of stock items cut in the optimal solution. We have $L' < L$ due to our assumption. Let O_i be the number of appearances for item i in the optimal solution. We know that $\sum_{i \in I} O_i w_i \leq W$ and $\sum_{i \in I} P_i w_i \leq W$. Moreover, $L' = \max_{i \in I} \lceil \frac{d_i}{O_i} \rceil$ and $L = \max_{i \in I} \lceil \frac{d_i}{P_i} \rceil$. Let $k = \text{argmax}_{i \in I} \lceil \frac{d_i}{P_i} \rceil$. In this case, we have $O_k \geq P_k + 1$ since $\lceil \frac{d_k}{O_k} \rceil \leq L' < L = \lceil \frac{d_k}{P_k} \rceil$. Our claim is that there exists $l \in I$ such that $P_l > O_l$. If we assume that no such l exists, then we obtain $w_k + \sum_{i \in I} P_i w_i \leq \sum_{i \in I} O_i w_i \leq W$ which contradicts the algorithm. The algorithm should not have terminated because we can assign one more place to item k which has the largest ratio of demand divided by the number of appearances in the pattern.

At some point during the algorithm, the number of appearances for item l increases from O_l to $O_l + 1$. At this point, let $P'_i (\leq P_i)$ be the number of appearances for item i. Then, we have $\lceil \frac{d_l}{O_l} \rceil \leq L' < \lceil \frac{d_k}{P_k} \rceil \leq \lceil \frac{d_k}{P'_k} \rceil$. This shows that according to the algorithm, O_l should not be increased to $O_l + 1$ since item l does not have the largest ratio. □

1.6 Heuristic Algorithms

The integer program proposed in Section 1.4 has a weak LP relaxation, and the formulation is highly symmetric similar to the formulations by Kantorovich [36] and Vanderbeck [55] due to interchangeability of the indices j which denote different cutting patterns. Therefore, obtaining an optimal solution using MILP formulation in Section 1.4 is computationally challenging and developing fast and robust heuristic methods is essential. In this section, we discuss the algorithms proposed for CSP-S. We discuss three types of algorithms: (i) local search algorithms, (ii) a column generation based heuristic algorithm, and (iii) a pattern pool based approach.

1.6.1 Local Search Algorithms

In general, a local search algorithm starts with an initial solution and looks for a better solution in the "neighborhood" of the current solution until there is no improvement. First, we discuss how to find an initial feasible solution. For that purpose, we develop constructive heuristic algorithms which help in finding a "good" initial solution.

1.6.1.1 Finding an Initial Solution

To find a "good" initial solution, we develop four constructive heuristic algorithms motivated by Algorithm 1 which is proposed for the *Single Setup Problem*. Namely, the four algorithms are (i) *Multiple Pattern Generation Algorithm* (MPGA), (ii) *Sequential Pattern Generation Algorithm* (SPGA), (iii) *Multiple Knapsack-Based Algorithm* (MKBA), and (iv) *Sequential Knapsack-Based Algorithm* (SKBA).

The main idea behind all four algorithms is to solve a *Single Setup Problem* with a stock item of length rW for some r value between $\lceil \sum_{i \in I} w_i/W \rceil$ and $\mathcal{N}$. Note that $\lceil \sum_{i \in I} w_i/W \rceil$ is a lower bound on the number of different patterns and $\mathcal{N}$ is an upper bound (implied by CSP solution) on the number of different patterns. Then, we form the patterns one-by-one by following a rule. The pseudocodes are provided in Algorithms 2, 3, 4, and 5. In MPGA (Algorithm 2), we assign the items to the patterns based on the number of times that each item is cut. After we determine the number of appearances (P_i) for each item using SSA, we replace item i with P_i sub-items with demand $\lceil \frac{d_i}{P_i} \rceil$. Then, we sort the sub-items in a descending order with respect to demand ($\lceil \frac{d_i}{P_i} \rceil$) or length ($w_i$) and assign them to individual patterns one-by-one considering the width of the stock item. When the current pattern is full, i.e., there is no more place to put an additional item from the sorted list of sub-items, we close this pattern and start a new pattern. This continues until all the sub-items are assigned to a pattern. The number of times each pattern is produced

is determined by the maximum demand of the sub-items assigned to this pattern. The expected final number of patterns is r, but we may end up with more patterns due to the waste in each cutting pattern. We run this procedure for a set of r values ($r \in \{\lceil \sum_{i \in I} w_i/W \rceil, \ldots, \mathcal{N}\}$) and choose the one with the smallest total cost.

Algorithm 2 *Multiple Pattern Generation Algorithm* (MPGA)

1: $MinCost = +\infty$
2: **for** $r = \lceil \frac{\sum_{i \in I} w_i}{W} \rceil$ to $\mathcal{N}$ **do**
3: Solve the *Single Setup Problem* assuming we have stock items of length rW.
4: Let P_i be the number of appearances for item i found after solving the *Single Setup Problem*.
5: Replace item i with P_i number of sub-items with demand $\lceil \frac{d_i}{P_i} \rceil$.
6: Sort the new $\sum_{i \in I} P_i$ sub-items according to their demand or length in a non-increasing order.
7: Start with the first pattern: $t = 1$.
8: Starting from the first sub-item in the list, assign each sub-item to pattern t if it fits into the stock item of length W. If it does not fit, finalize pattern t, and start a new pattern: $t = t + 1$. Continue this until all the sub-items in the list are assigned to a pattern.
9: After determining the patterns, each pattern will be produced equivalent to maximum demand in that pattern.
10: Calculate the total cost for the solution found. Let $TotalCost$ be the total cost.
11: **if** $TotalCost < MinCost$ **then**
12: $MinCost = TotalCost$
13: **end if**
14: **end for**

In MPGA, after solving *Single Setup Problem*, we can sort the sub-items with respect to demand values or lengths. Moreover, while assigning the items into the patterns (Step 8), we can still use the previous patterns if the current sub-item in the list fits to that pattern. This results in a different version of the algorithm. We consider all these combinations and report the best result.

In SPGA (Algorithm 3), similar to MPGA we first solve a *Single Setup Problem* and follow the same steps to determine the first pattern. Different from MPGA, after assigning each pattern, we update the remaining set of items and demand values and run MPGA for the updated set of items assuming that we have stock items of length $(r-1)W$. We repeat the procedure until all the items are assigned to patterns.

Algorithm 3 *Sequential Pattern Generation Algorithm* (SPGA)

1: $MinCost = +\infty$
2: **for** $r = \lceil \frac{\sum_{i \in I} w_i}{W} \rceil$ to $\mathcal{N}$ **do**
3: Set $k = r, H = I$.
4: Start with the first pattern: $t = 1$.
5: Solve the *Single Setup Problem* assuming we have stock items of length kW.
6: Let P_i be the number of appearances for item i found after solving the *Single Setup Problem*.
7: Replace item i with P_i number of sub-items with demand $\lceil \frac{d_i}{P_i} \rceil$.
8: Sort the new $\sum_{i \in H} P_i$ sub-items according to their demand or length in a nonincreasing order.
9: Starting from the first item in the list, assign each item to pattern t if it fits into the stock item of length W. If it does not fit, finalize pattern t. Update the set of unassigned items H.
10: **if** $H = \emptyset$ **then**
11: Go to Step 19.
12: **else**
13: **if** $\sum_{i \in H} w_i \leq (k-1)W$ **then**
14: Set $k = k - 1$, open a new pattern: $t = t + 1$ and go to Step 5.
15: **else**
16: Start a new pattern: $t = t + 1$ and go to Step 5.
17: **end if**
18: **end if**
19: After determining the patterns, each pattern will be produced equivalent to maximum demand in that pattern.
20: Calculate the total cost for the solution found. Let $TotalCost$ be the total cost.
21: **if** $TotalCost < MinCost$ **then**
22: $MinCost = TotalCost$
23: **end if**
24: **end for**

In MKBA (Algorithm 4), similar to MPGA first we solve a *Single Setup Problem* to determine P_i and $\lceil \frac{d_i}{P_i} \rceil$ for each item. While forming the patterns, instead of assigning sub-items one-by-one, we solve a knapsack problem. We assume any item cut on stock item of length rW is an independent item, and thus, we have $n' (= \sum_{i \in I} P_i)$ sub-items each with associated demand of $\lceil \frac{d_i}{P_i} \rceil$. Let U be the set of sub-items. We use d'_i and w'_i to denote the demand and length of sub-item i for $i \in U$. Defining x_i as a binary variable indicating whether sub-item i is placed into the current pattern or not, we solve the following knapsack problem to determine the sub-items to be produced in the current pattern. After fixing this pattern, we update the set of remaining

sub-items (U) and solve another knapsack problem for the remaining sub-items.

$$\text{KP: Maximize} \sum_{i \in U} d'_i x_i, \tag{1.25}$$

s.t.

$$\sum_{i \in U} w'_i x_i \leq W, \tag{1.26}$$

$$x_i \in \{0, 1\}, \qquad \forall i \in U. \tag{1.27}$$

In another variant of the MKBA, we replace the objective function coefficients of x_i's, which are d'_i currently, with $d'_i w'_i$. We report the best solution found among these two variants.

Algorithm 4 *Multiple Knapsack-Based Algorithm* (MKBA)

1: $MinCost = +\infty$
2: **for** $r = \lceil \frac{\sum_{i \in I} w_i}{W} \rceil$ to $\mathcal{N}$ **do**
3: Solve the *Single Setup Problem* assuming we have stock items of length rW.
4: Let P_i be the number of appearances for item i found after solving the *Single Setup Problem*, and $n' = \sum_{i \in I} P_i$.
5: Replace item i with P_i number of sub-items with demand $\lceil \frac{d_i}{P_i} \rceil$.
6: Initialize the set of sub-items: $U = \{1, 2, \ldots, n'\}$, and start with the first pattern: $t = 1$.
7: **if** $U = \emptyset$ **then**
8: Go to Step 13.
9: **else**
10: Solve KP (Equations (1.25)-(1.27)) to determine the current pattern.
11: Update U, start a new pattern: $t = t + 1$ and go to Step 7.
12: **end if**
13: After determining the patterns, each pattern will be produced equivalent to maximum demand in that pattern.
14: Calculate the total cost for the solution found. Let $TotalCost$ be the total cost.
15: **if** $TotalCost < MinCost$ **then**
16: $MinCost = TotalCost$
17: **end if**
18: **end for**

Finally, SKBA (Algorithm 5) is a variant of MKBA. The only difference is that when the first pattern is found by solving the knapsack problem, the remaining items

and their demand values are updated, and then, we solve another *Single Setup Problem* with the remaining items to form the next pattern. This is repeated until all the demand is satisfied. Cui et al. [16] propose a similar algorithm, called SHPC, to solve PMP. In SHPC, the authors start fitting items into stock items by solving a bounded knapsack problem where the weight of each item is defined as a function of the item's length based on some predefined parameters. While cost setting is important in determining the best solution in SKBA, SHPC is not sensitive to cost parameters. In Figure 1.2, we provide a comparison of the solutions found by SKBA and SHPC over 40 instances from the literature [49] for different cost settings: (i) $C_s = \$100,000, C_p = \$1$, (ii) $C_s = \$10,000, C_p = \$1$, (iii) $C_s = \$1,000, C_p = \$1$, and (iv) $C_s = \$100, C_p = \$1$. The solutions are normalized assuming that the solutions found by SHPC correspond to 100%. We see that the solutions generated by SKBA are around 35%, 36%, 36%, and 51% better than the solutions found by SHPC for cost settings (i), (ii), (iii), and (iv), respectively.

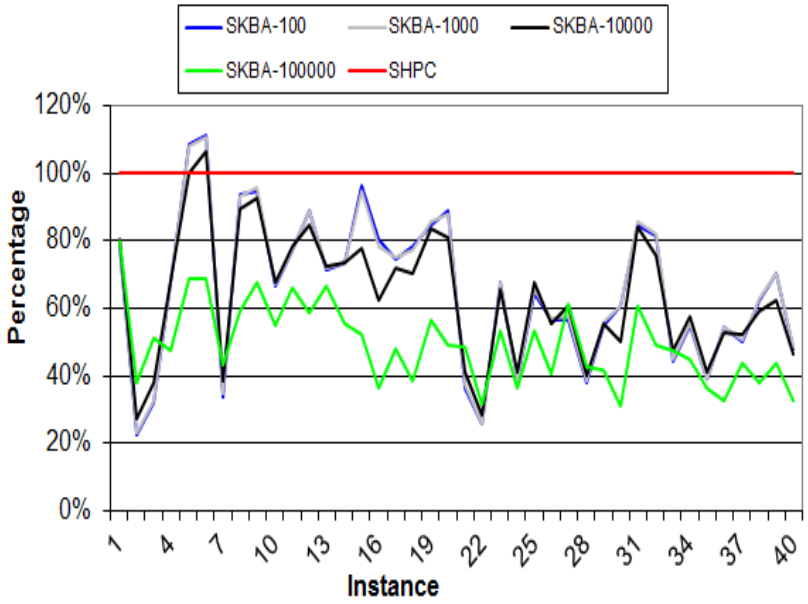

Figure 1.2: Comparison of SKBA and SHPC solutions over 40 instances for different cost settings: (i) $C_s = \$100,000, C_p = \$1$, (ii) $C_s = \$10,000, C_p = \$1$, (iii) $C_s = \$1,000, C_p = \$1$, and (iv) $C_s = \$100, C_p = \$1$.

Algorithm 5 *Sequential Knapsack-Based Algorithm* (SKBA)

1: $MinCost = +\infty$
2: **for** $r = \lceil \frac{\sum_{i \in I} w_i}{W} \rceil$ to $\mathcal{N}$ **do**
3: Set $k = r$, $H = I$.
4: Start the first pattern: $t = 1$.
5: **if** $H = \emptyset$ **then**
6: Go to Step 18.
7: **else**
8: Solve the *Single Setup Problem* assuming we have stock items of length kW.
9: Let P_i be the number of appearances for item i found after solving the *Single Setup Problem*.
10: Replace item i with P_i number of sub-items with demand $\lceil \frac{d_i}{P_i} \rceil$, and let U be the set of sub-items.
11: Solve KP (Equations (1.25)-(1.27)) to determine the current pattern.
12: Update H, and start a new pattern: $t = t + 1$.
13: **if** $\sum_{i \in H} w_i \leq (k-1)W$ **then**
14: Set $k = k - 1$.
15: **end if**
16: Go to Step 5.
17: **end if**
18: After determining the patterns, each pattern will be produced equivalent to maximum demand in that pattern.
19: Calculate the total cost for the solution found. Let $TotalCost$ be the total cost.
20: **if** $TotalCost < MinCost$ **then**
21: $MinCost = TotalCost$
22: **end if**
23: **end for**

After finding an initial solution using the above mentioned constructive heuristic algorithms, we try to improve the solution iteratively by checking the other solutions in the neighborhood of the current solution. Different local search methods are developed in the literature to solve CSP and PMP. Foerster and Waescher [24] present a method of combining the cutting patterns of a CSP solution to produce a new solution with fewer setups and generally more number of stock items than CSP solution. The method is called KOMBI, in which they combine 2 patterns to find 1 pattern, or combine 3 or 4 patterns to come up with 2 or 3 patterns instead, respectively. Using a similar idea of combining patterns, we propose two local search algorithms: (i) *Local Search with Exact Combination* (LSE), and (ii) *Local Search with Heuristic*

Combination (LSH). Different from KOMBI heuristic, in which only p to $p-1$ combination is allowed and p is at most 4, we consider combining p patterns into $p-i$ patterns where $i \in \{1, 2, \ldots, p-1\}$ and p can be larger than 4. Starting from $p = 2$, we consider all possible p values until there is not a feasible combination.

1.6.1.2 Local Search with Exact Combination

In *Local Search with Exact Combination* (LSE), we try to combine the patterns by solving a mixed integer program. The main idea is to combine p patterns into less than p patterns using a mixed integer program with the objective of minimizing the number of stock items used. We combine the patterns if the solution improves. We consider combining the patterns with a certain probability even if the new solution is worse. We use I' to denote the set of items produced in the patterns considered for combining. For each item in I', we use d_i^r to denote the demand that is satisfied by the patterns under consideration and should be satisfied by the newly formed patterns.

Assuming we consider p patterns to combine, the following decision variables are used in the proposed mixed integer program:

$$x_j = \text{Number of times pattern } j \text{ is produced} \qquad \forall j \in \{1, 2, \ldots, p-1\}$$
$$y_{ij} = \text{Number of times item } i \text{ is cut in pattern } j \qquad \forall i \in I', j \in \{1, 2, \ldots, p-1\}$$

We can formulate the problem as a non-linear model as follows:

$$\text{NLSE: Minimize} \sum_{j \in \{1,\ldots,p-1\}} x_j, \tag{1.28}$$

s.t.

$$\sum_{i \in I'} y_{ij} w_i \leq W, \qquad \forall j \in \{1, 2, \ldots, p-1\}, \tag{1.29}$$

$$\sum_{j \in \{1,\ldots,p-1\}} x_j y_{ij} \geq d_i^r, \qquad \forall i \in I', \tag{1.30}$$

$$x_j \in \mathbb{N}, \qquad \forall j \in \{1, 2, \ldots, p-1\}, \tag{1.31}$$

$$y_{ij} \in \mathbb{N}, \qquad \forall i \in I', j \in \{1, 2, \ldots, p-1\}. \tag{1.32}$$

In NLSE, objective function is minimizing the total material cost. Constraints (1.29) make sure that every new pattern fits into the stock item. Constraints (1.30) assure that the partial demand of items in I' is satisfied. The remaining constraints are the integrality restrictions. Using the upper bound on variable y_{ij}, we introduce the following two new decision variables to linearize constraints (1.30).

$$t_{ijk} = \begin{cases} 1, & \text{if item } i \text{ is cut } k \text{ times in pattern } j \\ 0, & \text{otherwise} \end{cases} \qquad \forall i \in I', j \in \{1, 2, \ldots, p-1\}, k \in K_i$$

$$e_{ijk} = x_j t_{ijk} \qquad \forall i \in I', j \in \{1, 2, \ldots, p-1\}, k \in K_i$$

The linearized version of NLSE is as follows:

$$\text{LLSE: Minimize} \sum_{j \in \{1,2,\ldots,p-1\}} x_j, \tag{1.33}$$

s.t.

$$\sum_{i \in I'} \sum_{k \in K_i} 2^k t_{ijk} w_i \leq W, \qquad \forall j \in \{1, 2, \ldots, p-1\}, \tag{1.34}$$

$$\sum_{j \in \{1,2,\ldots,p-1\}} \sum_{k \in K_i} 2^k e_{ijk} \geq d_i^r, \qquad \forall i \in I', \tag{1.35}$$

$$e_{ijk} \leq x_j, \qquad \forall i \in I', j \in \{1, 2, \ldots, p-1\}, k \in K_i, \tag{1.36}$$

$$e_{ijk} \geq x_j + M(t_{ijk} - 1), \qquad \forall i \in I', j \in \{1, 2, \ldots, p-1\}, k \in K_i, \tag{1.37}$$

$$e_{ijk} \leq M t_{ijk}, \qquad \forall i \in I', j \in \{1, 2, \ldots, p-1\}, k \in K_i, \tag{1.38}$$

$$x_j \in \mathbb{N}, \qquad \forall j \in \{1, 2, \ldots, p-1\}, \tag{1.39}$$

$$e_{ijk} \geq 0, \qquad \forall i \in I, j \in \{1, 2, \ldots, p-1\}, k \in K_i, \tag{1.40}$$

$$t_{ijk} \in \{0, 1\}, \qquad \forall i \in I, j \in \{1, 2, \ldots, p-1\}, k \in K_i. \tag{1.41}$$

The pseudocode for LSE is provided in Algorithm 6. Note that in Step 4 of the algorithm, we sort the patterns in the solution in a nondecreasing order with respect to the number of usage or in a nondecreasing order with respect to the total waste. We consider both of these alternatives and report the best solution found. Here, the total waste of a pattern is calculated by multiplying the number of times the pattern is used by the waste in the pattern.

Algorithm 6 *Local Search Algorithm with Exact Combination* (LSE)

Let *TotalCost* be the total production cost of the best initial solution found by MPGA, SPGA, MKBA and SKBA.
Initialize the best solution found so far: $BestSolution = TotalCost$.
Initialize the number of patterns to be combined: $p = 2$.
Order patterns in a nondecreasing order according to the number of usage (or in a nondecreasing order according to total waste).
Choose first p patterns.
Solve LLSE.
if LLSE is infeasible **then**
    Increase the number of patterns considered: $p = p + 1$, and go to Step 5.
else
    Calculate the total cost of new solution: *TempCost*.
    if *TempCost* < *TotalCost* **then**
        Update the current solution: *TotalCost* = *TempCost*.
        if *TotalCost* < *BestSolution* **then**
            Update the best solution found so far: *BestSolution* = *TotalCost*.
        end if
        Go to Step 3.
    else
        Generate a threshold value p_1, and a moving probability p_2 uniformly from [0,1].
        if $p_2 \geq p_1$ **then**
            Move to this new solution: *TotalCost* = *TempCost*, and go to Step 3.
        else
            Terminate the algorithm, and report the best solution: *BestSolution*.
        end if
    end if
end if

1.6.1.3 Local Search with Heuristic Combination

In *Local Search with Heuristic Combination* (LSH), we combine the patterns in a heuristic way using SSA or one of the constructive heuristic algorithms. Similar to LSE, we try to combine the chosen patterns in such a way that the portion of the demand satisfied by the patterns under consideration will be satisfied by the newly formed patterns. If we consider combining two patterns into one, we use SSA since it finds the optimal solution for one setup case. Otherwise, we use the constructive heuristic algorithms discussed in Section 1.6.1.1. The algorithm is outlined in Algorithm 7.

Algorithm 7 *Local Search with Heuristic Combination* (LSH)

1: Let *TotalCost* be the total production cost of the best initial solution found by MPGA, SPGA, MKBA and SKBA.
2: Initialize the best solution found so far: $BestSolution = TotalCost$.
3: Initialize the number of patterns to be combined: $p = 2$.
4: Order patterns in a nondecreasing order according to the number of usage (or in a nondecreasing order according to total waste).
5: Choose first p patterns.
6: **if** $p = 2$ **then**
7: Run SSA (Algorithm 1) to combine two patterns into one pattern, and store the solution: *TempCost*.
8: **else**
9: Run all the four constructive heuristic algorithms (MPGA, SPGA, MKBA and SKBA) to combine p patterns into less than p patterns, and store the best solution found: $TempCost$.
10: **end if**
11: **if** *TempCost* < *TotalCost* **then**
12: Update the current solution: *TotalCost* = *TempCost*.
13: **if** *TotalCost* < *BestSolution* **then**
14: Update the best solution found so far: *BestSolution* = *TotalCost*.
15: **end if**
16: Go to Step 3.
17: **else**
18: Generate a threshold value p_1, and a moving probability p_2 uniformly from [0,1].
19: **if** $p_2 \geq p_1$ **then**
20: Move to this new solution: *TotalCost* = *TempCost*, and go to Step 3.
21: **end if**
22: **end if**
23: **if** New solution with total cost *TempCost* has p number of patterns **then**
24: Increase the number of patterns considered: $p = p + 1$, and go to Step 4.
25: **end if**
26: Terminate the algorithm, and report the best solution: *BestSolution*.

1.6.2 Column Generation Based Heuristic Algorithm

In this section, we develop a *Column Generation Based Heuristic Algorithm* (CGA) to solve CSP-S by modifying the column generation method developed by Vanderbeck [55] to solve PMP. CGA is different from the column generation algorithm of Vanderbeck [55] in terms of production requirements, objective function, and method to solve the subproblem. In CGA, we allow excess production in favor of reducing the number of patterns, thus the demand constraint we are dealing with is different.

In addition, the objective function of the CGA's master problem is minimizing total production cost, while Vanderbeck's model only minimizes the number of setups given the number of stock items to be used. In Vanderbeck [55], the subproblem is originally non-linear and solved with a branch-and-bound approach. Although the CGA subproblem is also non-linear, we use the method discussed in Section 1.4 to linearize the subproblem and obtain the optimal solution at each iteration of the CGA.

Similar to Vanderbeck [55], we define Q as the set of feasible patterns as follows:

$$Q = \{q = (q_0, q_1, \ldots, q_n) \in \mathbb{N}^{n+1} : \sum_{i=1}^{n} w_i q_i \leq W, q_0 q_i \leq d_i \ \ \forall i\}. \tag{1.42}$$

Here, the first entry of a vector in Q represents the number of times this cutting pattern is used, and the remaining entries specify the cutting pattern. Note that we have $q_0 q_i \leq d_i$ constraint in order to be able to utilize the same procedure although it is not necessarily true for CSP-S. Otherwise, the related subproblem is not a meaningful problem without defining a value for q_0. For any vector q in Q, we define a decision variable λ_q, which is 1 if pattern q $(q_1, \ldots, q_n)$ is produced q_0 times and 0 otherwise. Using these vectors, the master problem of the column generation procedure can be formulated as follows:

$$\text{MP-CG: Minimize} \quad C_s \sum_{q \in Q} \lambda_q + C_p W \sum_{q \in Q} q_0 \lambda_q, \tag{1.43}$$

s.t.

$$\sum_{q \in Q} q_0 q_i \lambda_q \geq d_i, \qquad \forall i \in I, \tag{1.44}$$

$$\lambda_q \in \{0, 1\}, \qquad \forall q \in Q. \tag{1.45}$$

We generate the profitable columns (vectors in Q) using the dual prices of the LP relaxation of MP-CG. Assuming that π_i $(\forall i \in I)$ are the dual variables associated with constraints (1.44), the columns are generated by solving the following subproblem:

$$\text{NSP-CG: Minimize} \quad C_p W q_0 - \sum_{i \in I} q_0 q_i \pi_i, \tag{1.46}$$

s.t.

$$q_0 q_i \leq d_i, \qquad \forall i \in I, \quad (1.47)$$

$$\sum_{i \in I} q_i w_i \leq W, \quad (1.48)$$

$$q_0 \in \mathbb{N}, \quad (1.49)$$

$$q_i \in \mathbb{N}, \qquad \forall i \in I. \quad (1.50)$$

Note that this is a non-linear formulation. However, we can linearize it using the idea discussed in Section 1.4. For item i, we know that $\lfloor \frac{W}{w_i} \rfloor$ is an upper bound on q_i. Therefore, we can linearize NSP-CG as follows:

$$\text{LSP-CG: Minimize} \quad C_p W q_0 - \sum_{i \in I} \pi_i \sum_{k \in K_i} 2^k A_{ik}, \quad (1.51)$$

s.t.

$$\sum_{k \in K_i} 2^k A_{ik} \leq d_i, \qquad \forall i \in I, \quad (1.52)$$

$$\sum_{i \in I} \sum_{k \in K_i} 2^k E_{ik} w_i \leq W, \quad (1.53)$$

$$A_{ik} \leq q_0, \qquad \forall i \in I, k \in K_i, \quad (1.54)$$

$$A_{ik} \leq M E_{ik}, \qquad \forall i \in I, k \in K_i, \quad (1.55)$$

$$A_{ik} \geq q_0 + M(E_{ik} - 1), \qquad \forall i \in I, k \in K_i, \quad (1.56)$$

$$q_0 \in \mathbb{N}, \quad (1.57)$$

$$A_{ik} \geq 0, \qquad \forall i \in I, k \in K_i, \quad (1.58)$$

$$E_{ik} \in \{0, 1\}, \qquad \forall i \in I, k \in K_i. \quad (1.59)$$

In LSP-CG, E_{ik} is a new binary variable such that $q_i = \sum_{k \in K_i} 2^k E_{ik}$, and A_{ik} is a new positive decision variable which is equal to $q_0 E_{ik}$. In this model, E_{ik} and A_{ik} are used to linearize constraints (1.47).

In the classical column generation, when a new pattern is found, one of the previous patterns is removed from the consideration set after solving the LP relaxation of the master problem. When there is no candidate pattern found to enter the set of patterns, the algorithm is terminated and the original integer master problem is solved using the final set of patterns. However, in our implementation, we keep track

of all the columns generated, and stop the procedure when enough patterns, say $100n$, are found. At the end, we solve the master problem (MP-CG). Although this method increases the solution time of master problem, it helps include a higher number of different patterns and improve the quality of the final solution.

1.6.3 Pattern Pool Based Approach

In this section, we present a *Pattern Pool Based Approach* (PPBA) to solve CSP-S. ILOG IBM CPLEX introduced a solution pool feature to find multiple solutions for a problem within a percentage of the optimal solution. This feature has two steps. In the first step, it solves the problem's mathematical model until certain stopping criteria are met and records the set of solutions found. In the second step, the mathematical model will run based on the solutions that were saved. In Section 1.6.1.1, we proposed four constructive heuristic algorithms that can find good initial solutions (sets of patterns) for the CSP-S. The solution pool feature and available set of patterns found by the constructive heuristic methodologies motivated us to propose a similar algorithm for our problem called as *Pattern Pool Based Approach*.

In PPBA, we first run all of the four heuristic algorithms proposed in Section 1.6.1.1 and finds $\mathcal{J}$, a set of feasible patterns. Then we feed the generated patterns as a pool of solutions to a mathematical model (PPBA) with the following decision variables.

$$X_j = \text{Number of times pattern } j \text{ is used} \qquad \forall j \in \mathcal{J}$$

$$Y_j = \begin{cases} 1, & \text{if pattern } j \text{ is used} \\ 0, & \text{otherwise} \end{cases} \qquad \forall j \in \mathcal{J}$$

The proposed formulation is as follows:

$$\text{PPBA: Minimize} \quad C_s \sum_{j \in \mathcal{J}} Y_j + C_p W \sum_{j \in \mathcal{J}} X_j, \tag{1.60}$$

s.t.

$$\sum_{j \in \mathcal{J}} a_{ij} X_j \geq d_i, \qquad \forall i \in I, \tag{1.61}$$

$$X_j \leq M Y_j, \qquad \forall j \in \mathcal{J}, \tag{1.62}$$

$$X_j \in \mathbb{J}, \qquad \forall j \in \mathcal{J}, \tag{1.63}$$

$$Y_j \in \{0, 1\}, \qquad \forall j \in \mathcal{J}. \tag{1.64}$$

where M is a very large constant. a_{ij} is the number of times that item i is cut on pattern j. The values of a_{ij} are calculated based on the constructive heuristic algorithm solutions. This model tries to choose a set of patterns from the provided pool of patterns to minimize the total production cost. It also decides about the number of times that each pattern should be used in order to satisfy the demand. The patterns, that are available in the pool, will satisfy $\sum_{i \in I} a_{ij} w_i \leq W$, because the heuristic algorithms make sure patterns are feasible. Therefore, there is no need to add constraints (1.3) of proposed model in Section 1.4.

1.7 Computational Results

To test the performance of proposed algorithms, we conduct computational experiments on the instances available in the literature for different cost settings. We use 40 CSP instances from a chemical fiber company including up to 29 items which are provided by Umetani et al. [49] at `http://www-sys.ist.osaka-u.ac.jp/~umetani/instance-e.html`. All the computational experiments are carried out on a system with two 2.4 GHz Xeon processors and 4 GB RAM. The algorithms are implemented in C++, and CPLEX 12.2, with default settings, is used as the optimization engine. We consider four different cost settings in the experiments to see the performance of the algorithms under different cost structures: (i) $C_p = 1, C_s = 100,000$, (ii) $C_p = 1, C_s = 10,000$, (iii) $C_p = 1, C_s = 1,000$, and (iv) $C_p = 1, C_s = 100$. As the setup cost increases, total setup cost comprises a larger portion of the total cost, and therefore, the number of setups becomes more important.

In addition to the two local search algorithms (LSE and LSH), the column generation based heuristic algorithm (CGA) and pattern pool based approach (PPBA), we include the solutions of three other problems/algorithms in the comparisons as well: (i) classical *Cutting Stock Problem* (CSP), (ii) *Pattern Minimization Problem* (PMP) and (iii) *Vanderbeck's Column Generation Algorithm* [55] (CGV). To solve CSP, we use the assignment formulation proposed by Kantorovich [36]. Total production cost for CSP is then calculated using the considered C_p and C_s values. To find PMP solutions, the compact formulation of Vanderbeck [55] is solved after linearization using the same method in Section 1.4. While implementing PMP and CGV of Vanderbeck [55], in order to make a fair comparison we allow excess production, and we consider different K values as an upper bound on the number of stock items to be cut. As mentioned by Vanderbeck, the trade-off between material and setup cost minimization can be examined by setting K to different values.

To calculate the average optimality gap of the solutions found by any of these algorithms, we need to find the optimal solution or a lower bound for CSP-S. We set a 4-hour time limit on MILP formulation and report the optimal solution or lower bound (best LP relaxation objective) found at the end of 4 hours. Using this optimal solution or lower bound, denoted by LB, the optimality gap of a solution with total cost TC is calculated as follows:

$$\text{Optimality Gap} = \frac{TC - LB}{LB} \times 100\%. \tag{1.65}$$

We summarize the computational results in Tables 1.3, 1.4 and 1.5. Table 1.3 presents the average optimality gaps and CPU times of the constructive heuristic algorithms for different cost settings, which are later used as subroutines for LSE and LSH, and also provide the pool of patterns used in PPBA. Although the average optimality gaps of MKBA or SKBA are lower than MPGA or SPGA, we consider

all these algorithms while implementing LSE, LSH and PPBA, because each of these four algorithms can find the best solution in almost 10% of the instances.

Table 1.3: Average CPU time and optimality gap of the solutions found by MPGA, SPGA, MKBA and SKBA.

	MPGA	SPGA	MKBA	SKBA
	$C_p = 1$, $C_s = 100,000$			
Opt. Gap (%)	29.60	43.37	**21.30**	30.10
CPU Time (sec)	0.01	0.74	40.60	5.13
	$C_p = 1$, $C_s = 10,000$			
Opt. Gap (%)	14.62	23.39	11.08	**9.16**
CPU Time (sec)	0.01	0.74	35.25	5.11
	$C_p = 1$, $C_s = 1,000$			
Opt. Gap (%)	11.49	9.38	7.56	**3.96**
CPU Time (sec)	0.01	0.74	46.35	5.05
	$C_p = 1$, $C_s = 100$			
Opt. Gap (%)	10.93	9.04	7.54	**3.89**
CPU Time (sec)	0.01	0.74	45.20	5.13

In Table 1.4, we provide the average optimality gap of the solutions and average CPU time over 40 instances for different cost settings. The column headings represent the algorithm used. For CSP, PMP and CGV, since the setup costs are not explicitly considered, the solution times for different cost settings are same. In its original form (with equality constraint), CGV fails to find a solution for almost 25% of the instances (generally larger instances) in all cost settings due to infeasibility of the final integer master problem while using the patterns found by the branch-and-bound algorithm. Therefore, the optimality gaps for CGV are over 30 instances. Cerqueira and Yanasse [12] also report the same deficiency in solving large instances using Vanderbeck's solution approach. The optimality gaps under LSH, LSE and PPBA columns demonstrate the improvement on the initial solution found by the constructive heuristics after using local search methods or running PPBA model. We see that LSE performs 1% better than LSH on average. However, the solution time of LSH is around 94% lower than that of LSE. PPBA performs better than

both LSH and LSE for lower setup cost settings. When the setup costs are high, although LSE performs better than PPBA, PPBA still has a superior performance over LSH. Among the proposed algorithms, CGA performs slightly better than PPBA for low setup cost settings. However, when the setup costs are high, all LSE, LSH and PPBA outperform CGA. When we compare the proposed algorithms with PMP, CSP and CGV, we see that the proposed algorithms provide significantly better results when the setup cost is high. Although CGV and CSP are specifically designed for small/negligible setup cost settings, the proposed algorithms still provide similar or slightly worse solutions. Note that for some cost settings optimality gaps of PMP solutions are higher than the gaps for CGV solution because of the 4-hour time limit on PMP formulation. Thus, for lower cost settings CGV results should be considered as PMP results as well.

Table 1.4: Average CPU time and optimality gap of the solutions found by LSE, LSH, CGA, PPBA, CSP, PMP and CGV.

	LSH	LSE	CGA	PPBA	PMP	CSP	CGV
			$C_p = 1$, $C_s = 100,000$				
Opt. Gap (%)	10.08	**8.84**	13.88	9.97	44.72	164.22	88.95
CPU Time (sec)	73.98	1450.02	22.32	1603.98	9949.68	8675.43	94.08
			$C_p = 1$, $C_s = 10,000$				
Opt. Gap (%)	2.12	**1.54**	2.08	1.76	27.38	27.38	8.56
CPU Time (sec)	77.20	1470.31	269.00	1561.23	9949.68	8675.43	94.08
			$C_p = 1$, $C_s = 1,000$				
Opt. Gap (%)	1.91	0.85	**0.45**	0.53	3.29	3.09	0.58
CPU Time (sec)	91.11	1607.70	722.33	1700.67	9949.68	8675.43	94.08
			$C_p = 1$, $C_s = 100$				
Opt. Gap (%)	2.04	0.90	**0.43**	0.50	2.57	**0.43**	0.55
CPU Time (sec)	99.87	1709.16	2925.25	1636.71	9949.68	8675.43	94.08

Finally, to provide a fair comparison of the algorithms, we compare the solutions found over 30 instances that CGV could find a solution. We provide the results in Table 1.5. We observe that over these 30 instances CGA performs best except the highest setup cost setting. LSE, LSH and PPBA still provide comparable results for

low cost settings and better results for the highest setup cost setting.

Table 1.5: Average optimality gap of the solutions found by LSE, LSH, CGA, PPBA, CSP, PMP and CGV over 30 instances for which a feasible solution is found by CGV.

	LSH	LSE	CGA	PPBA	PMP	CSP	CGV
	$C_p = 1$, $C_s = 100,000$						
Opt. Gap (%)	10.81	**9.16**	11.29	9.99	46.99	150.72	88.95
	$C_p = 1$, $C_s = 10,000$						
Opt. Gap (%)	1.90	1.50	**0.92**	0.98	32.25	23.28	8.56
	$C_p = 1$, $C_s = 1,000$						
Opt. Gap (%)	1.42	0.80	**0.28**	0.34	2.69	1.72	0.58
	$C_p = 1$, $C_s = 100$						
Opt. Gap (%)	1.01	0.71	**0.24**	0.30	2.50	0.35	0.55

1.8 Conclusion

In this chapter, we introduce a general version of the one-dimensional cutting stock problem, called *Cutting Stock Problem with Setup Cost*, in which our goal is to minimize the total production cost including material and setup costs. *Cutting Stock Problem with Setup Cost* provides a unified framework for a wide domain of packing/cutting stock problems including the well-known *Cutting Stock Problem* and *Pattern Minimization Problem*.

We first develop a mixed integer linear model for this NP-hard problem. Then, we analyze a special case of the problem and propose an algorithm to find the optimal solution. We develop (i) two local search algorithms each of which utilize constructive heuristic algorithms motivated by a special case of the problem, (ii) a pattern pool based approach and (iii) a column generation based heuristic algorithm. We test the performance of these algorithms on the instances from the literature. Our results show that the proposed column generation algorithm provides solutions with smaller total cost for low setup cost settings, while local search algorithms and pattern pool based approach provide better solutions for high setup cost settings. Compared to other

algorithms for similar problems in the literature, the proposed algorithms provide significantly better results for high setup cost settings, and similar results for low setup cost settings.

Chapter 2 Integrated Collection and Appointment Scheduling Problem

2.1 Introduction

Blood is one of the vital products needed for medical treatments including cancer treatment, orthopedic and cardiovascular surgeries, organ and marrow transplants and blood disorder treatments. In the U.S., every two seconds a patient needs blood or blood products, and more than 38,000 blood donations are needed daily to satisfy the demand [3]. In many countries, people still die because of inadequate supply of blood products [59].

According to the eligibility rules established by the U.S. Food and Drug Administration (FDA), around 38% of the population is eligible for blood donation in the U.S, but only 3% of the population donates blood in a year. Hence, managing this limited blood supply efficiently is as crucial as promoting blood donation.

When a donor donates blood at a donation site, this donated blood (also called *whole blood*) is separated into its components by *centrifugation* [3]. The three main blood products used in transfusion are red blood cells, platelets and plasma. Plasma is used for burn and trauma patients. Red blood cells are needed for any patient requiring transfusion. They are mainly used for anemia treatment, surgery, treatment of blood disorders and for premature babies. Finally, platelets are used to treat cancer patients, accident and malaria victims, asthma patients and others with blood clotting problems. Although all of these products have limited shelf-lives, platelets are the most critical one due to its short life-span (5 to 9 days).

In the U.S., FDA and the American Association of Blood Banks (AABB) regulate collection, processing and storage of blood and its components. According to these regulations, whole blood must be processed within 8 hours of donation to extract

platelets [61]. Similar regulations are imposed in other countries such as Austria and Turkey as well [20, 48]. The platelet extraction is generally done at a central processing center. For example, all the blood collected by American Red Cross in Buffalo, NY is sent to Rochester, NY for processing, and then distributed to hospitals in Buffalo [4]. Similarly, in Connecticut, the blood units picked up from donation sites are delivered to headquarters in Farmington [61]. Hence, blood collection organizations have to schedule continuous pickups from the donation sites and deliver the collected blood units to the processing center for platelet production. Since processing takes around 2 hours, any donated blood unit that stays at a donation site or in a collection vehicle more than 6 hours prior to processing cannot be used for platelet production. However, those units can still be used for extracting other blood products. We call this 6-hour requirement *processing time limit*.

Most of the blood collection organizations operate on an appointment based schedule in order to improve staff and equipment utilization [33]. This also decreases donor waiting time which is important for future donations from the repeat donors. Furthermore, since the time of donations affects platelet production as well due to 6-hour processing time limit, synchronizing the appointment and pickup schedules can improve the platelet supply. For example, assume that there is only one scheduled pickup from a donation site at 3pm, and the collection vehicle returns back to the processing center by 6pm. In this case, instead of randomly scheduling the appointments, it is better to schedule as many donations as possible between 12pm and 3pm in order to increase platelet production while considering the capacity (bed/staff/equipment) of the donation site.

In this chapter, we analyze the pickup and appointment schedules at the donation sites while considering the processing time limit on platelet production. More specifically, we study the problem of coordinating the collection and appointment schedules in order to maximize platelet production. We call this problem *Integrated Collection*

and Appointment Scheduling Problem (ICASP). In ICASP, we have multiple blood donation sites and a central processing center with prespecified opening and closing times. Donation sites have capacities which limit the number of donations that can be performed at the same time. We have a fleet of vehicles to collect the donated blood units from the donation sites. Using the estimated number of donors scheduling an appointment at each donation site daily, we try to determine the pickup and appointment schedules simultaneously for an improved platelet supply.

In ICASP, we assume that the donation sites are partitioned into clusters where each vehicle is assigned to a single cluster and the donation sites in a cluster are visited by the same vehicle. This clustering assumption is more practical since it eliminates the need for coordinating the visits by different vehicles to the same donation site. We first develop a mixed integer nonlinear programming formulation for ICASP, and then linearize it. Later, we propose two heuristic algorithms to find a "good" solution: (i) *Integer Programming Based Algorithm* (IPBA), and (ii) *Construction Based Heuristic Algorithm* (CBHA). Both IPBA and CBHA first cluster the donation sites using a variant of the well-known k-means clustering algorithm [37], and then determine the pickup and appointment schedules for each cluster.

The remainder of the chapter is organized as follows. In Section 2.2, we provide a review of the related work in the literature. We present the formal problem definition in Section 2.3. We also provide a simple example to illustrate the benefit of coordinated pickup and appointment scheduling. We present the mixed integer programming formulations for ICASP in Section 2.4. In Section 2.5, we discuss the heuristic algorithms proposed for solving ICASP. To compare the performances of the proposed mathematical model and the heuristic algorithms, we conduct a computational study using the instances generated from Gulf Coast Regional Blood Center's data [33]. The results of the computational experiments are presented in Section 2.6. We conclude the chapter in Section 2.7.

2.2 Literature Review

ICASP is related to the well-known *Vehicle Routing Problem* (VRP). VRP is extensively studied by several authors since the early work by Dantzig and Ramser [18]. We refer the reader to Toth and Vigo [47] for a survey of exact algorithms and to Cordeau et al. [15] for a survey of heuristic algorithms proposed for VRP. Recently, Golden et al. [31] discuss the latest advances and new challenges in VRP. The main differences between VRP and ICASP are the processing time limit for platelet production and the accumulating behavior of the blood donations. Because of these differences, some donation sites can be visited more than once per day to increase platelet production. Finally, in ICASP the objective is to maximize platelet production whereas the objective in a typical VRP is minimizing the total transportation cost.

Among the variants of VRP, the most related problem to ICASP is the milk collection problem [42]. In Sankaran and Ubgade [42], the authors study the transportation of milk from milk collection centers to the dairy with the objective of minimizing the transportation cost. Similar to ICASP, there is a limit on the time that milk can spend in a collection vehicle. However, in milk collection problem milk is assumed to be available for pickup at the collection centers early in the day. Hence, the way that the time limit affects the vehicle routes, the availability of products at the beginning of the day and the objective are the main differences between ICASP and the milk collection problem.

Collection of donated blood units from donation sites is first discussed by Prastacos [41], but he does not consider the processing time limit. The most related studies in the literature are the ones by Doerner et al. [20], Ghandforoush and Sen [26], and Yi and Scheller-Wolf [61]. Although the processing time limit is considered in all these studies and multiple visits to donation sites are allowed in Doerner et al. [20] and Ghandforoush and Sen [26], they all ignore the appointment scheduling aspect and

assume that the donation times are predetermined.

Ghandforoush and Sen [26] develop a decision support system to manage platelet production for the units that are donated at mobile blood drives. They assume that shuttles (collection vehicles) make round trips between the blood drives and the processing center. The amount that can be picked up from a blood drive per visit is bounded from above and below. The objective in their model is to minimize the total daily cost which includes production, transportation and yield loss costs (due to testing, delays in transportation, contamination, etc.) while satisfying the demand. The authors develop a non-convex integer programming model to determine the shuttle schedules. Then, they linearize it using linearization techniques [30]. They only consider round trips and do not allow visiting more than one blood drive per route. Moreover, they assume that a constant amount is picked up from a blood drive in each visit.

Doerner et al. [20] also study a related problem where the donations are assumed to be uniformly distributed over the operating hours of a donation site. Their objective is to collect all the donated units for platelet production with minimum transportation cost. They propose a mixed integer programming formulation, an exact method and several constructive heuristic approaches to solve the problem. In both exact and heuristic approaches, the number of pickups from each donation site is fixed at the beginning. The main shortcoming of the study by Doerner et al. [20] is that even a single donated unit has to be collected. However, in practice not all of the donated units are used for platelet production. Furthermore, they do not consider the availability of the collection vehicles.

Assuming that the donation sites are visited only once, Yi and Scheller-Wolf [61] study the blood collection operations from donation centers. They look for a minimum cost solution while collecting a pre-determined amount of blood for platelet collection. Similar to Doerner et al. [20], they assume that the donations occur

uniformly throughout the day. They develop an exact algorithm in which all feasible tours are generated beforehand which is computationally challenging in general. In this study, each donation site is visited only once which limits the platelet production significantly in practice.

2.3 Problem Definition

In this section, we first provide a formal definition of the problem, and then illustrate the benefit of coordinating collection and appointment schedules on a simple example.

In ICASP, we have N donation sites operated by a blood collection organization. The donated units have to be delivered to a central processing center (denoted by 0) for platelet production. We use I ($= \{1, 2, \ldots, N\}$) to denote the set of donation sites, and I_0 ($= \{0, 1, 2, \ldots, N\}$) to denote the set of all locations including the processing center. We assume that the total daily number of donations to be scheduled at donation site i is constant and denoted by D_i for all $i \in I$. Furthermore, we assume that it takes q hours to complete a single donation. The capacity of donation site i, defined as the number of donations that can be handled at the same time, is denoted by C_i. In practice, the number of beds, equipment and staff determine C_i. $[a_i, b_i]$ is the operating hours of donation site i. Appointments for blood donation can be scheduled any time between a_i and b_i, and all of the donations have to be completed before b_i. Moreover, we assume that the processing center operates during the time interval $[a_0, b_0]$. We use t_{ij} to denote the travel time (in hours) from location i to location j for all $i, j \in I_0$, and assume that t_{ij}'s satisfy triangle inequality.

We have L uncapacitated collection vehicles to collect donated units from the donation sites and deliver them to the processing center. Due to the small size of the blood bags, vehicle capacities are ignored. All the vehicles are initially (at time a_0) located at the processing center and have to return back to the depot at the end

of the day (no later than b_0). When a vehicle visits location i (it can be a donation site or the processing center), it spends f_i hours for loading/unloading. Although donation site i operates during $[a_i, b_i]$, we assume that a collection vehicle can still visit donation site i after b_i to collect the remaining units for platelet production. Finally, processing time limit on the donated units for platelet production is denoted by S (hours). That is, in order to extract platelets from a donated blood unit, we have to deliver it to the processing center within S hours of its donation time.

In ICASP, while assuming that we can schedule the blood donation appointments at any time during the operating hours, we want to determine the appointment and collection schedules for all the donation sites with the objective of maximizing the platelet production (or, equivalently, maximizing the total number of donated units delivered to the processing center within S hours of donation).

We provide a simple example (see Figure 2.1) to illustrate the importance of synchronizing the appointment and pickup schedules to maximize platelet production. In Figure 2.1, we have a single processing center, two donation sites, and a single vehicle to collect donated blood from these donation sites. The corresponding travel times (in hours) are given on each edge. We assume that each donation site is open for 4 hours during 8am-12pm, and 8 donations are performed/scheduled (daily) at each donation site. Furthermore, we assume that it takes one hour to perform a single donation, and each donation can be scheduled at one of the following 1-hour intervals: 8am-9am, 9am-10am, 10am-11am, 11am-12pm. Each donation site can handle 4 donations at the same time, i.e., $C_i = 4$. Finally, we assume that donated blood has to be processed within 6-hours of donation time for platelet extraction. This means that blood units donated during time interval 8am-9am have to be delivered to the processing center no later than 3pm. For simplicity, we assume that $f_i = 0$ for all locations and the processing center is open during the entire day.

If we spread the donations over time in order to have a better utilization and

balanced workload, we schedule 2 donations at each 1-hour interval for each donation site. One can easily see that one vehicle cannot collect and deliver all the donated blood in this case. The maximum amount that can be collected/processed is 12 units. To collect 12 units of blood, the vehicle has to leave the processing center at 7am, and visit donation sites 1 and 2 at 10am and 12pm, respectively. However, if the collection and appointment schedules are considered together, we can collect all the donated units by scheduling 4 donations at each of 8am-9am and 9am-10am intervals at donation site 1 and keeping the schedule for the second donation site same.

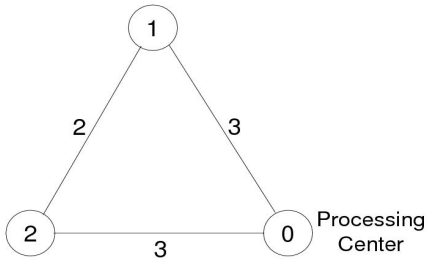

Figure 2.1: An example illustrating the importance of synchronizing the appointment and pickup schedules.

Consider the following instance of ICASP: (i) the donation site capacities are very large, (ii) the loading/unloading times are zero, (iii) all the donation sites open at the same time, say a, (iv) processing center closes at $a + S$, and opens at a time much earlier than a, (v) there is a single collection vehicle, and (vi) donation takes negligible amount of time. In this case, the vehicle can make only one tour and all the donations at a visited donation site can be scheduled so that they will be completed right before the pickup. The problem becomes finding a path of length not more than S while maximizing the total number of donations at the visited donation sites. This is the well-known *Orienteering Problem* which is known to be NP-hard [32]. Hence, ICASP is NP-hard as well.

2.4 A Mixed Integer Linear Programming Model

In this section, we present a mixed integer linear formulation for ICASP. We use Q to denote the maximum number of tours a vehicle can make on a day. We use the travel time between the processing center and the donation sites to find a value for Q.

$$Q = \frac{b_0 - a_0}{\min_{i \in I}\{2t_{0i}\}}.$$

In addition to the notation defined so far, we define V $(= \{1, \ldots, L\})$ as the set of vehicles and $\mathcal{M}$ $(= \{1, \ldots, Q\})$ as the set of possible tours. Finally, we use A as a large constant. The decision variables used in the formulation are as follows:

O_{imk} = Amount of blood units picked up from donation site i by vehicle k in tour m and used for platelet production — $i \in I, m \in \mathcal{M}, k \in V$

x_{imk} = Arrival time of vehicle k at donation site i in tour m — $i \in I, m \in \mathcal{M}, k \in V$

$$z_{ijmk} = \begin{cases} 1, & \text{if location } j \text{ is visited right after location } i \text{ by vehicle } k \text{ in tour } m \\ 0, & \text{otherwise} \end{cases} \qquad i, j \in I_0, m \in \mathcal{M}, k \in V$$

w_{mk} = Returning time of vehicle k to depot at the end of tour m — $m \in \mathcal{M}, k \in V$

v_{imk} = Latest time by which the donations that are collected by vehicle k from site i in tour m should be completed — $i \in I, m \in \mathcal{M}, k \in V$

$$e_{mk} = \begin{cases} 1, & \text{if vehicle } k \text{ leaves depot for tour } m \\ 0, & \text{otherwise} \end{cases} \qquad m \in \mathcal{M}, k \in V.$$

Here, v_{imk} is equal to min $\{x_{imk}, b_i\}$. In the formulation, the value of x_{imk} is set to the value of $x_{i(m-1)k}$ if donation site i is not visited by vehicle k in tour m. In ICASP, the objective is to maximize the amount of donated blood units that are processed

to extract platelet. The objective function is as follows:

$$\max \sum_{i \in I} \sum_{k \in V} \sum_{m \in \mathcal{M}} O_{imk}. \tag{2.1}$$

Next, we explain the constraints in the formulation.

2.4.1 Tour-Related Constraints

To ensure the feasibility of tours, we introduce a set of constraints to the model. Constraints (2.2) and (2.3) make sure that each donation site is visited at most once in a tour. Finally, constraints (2.4) help to satisfy the flow balance for all the locations,

$$\sum_{i \in I_0} z_{ijmk} \leq 1, \qquad \forall j \in I, m \in \mathcal{M}, k \in V, \tag{2.2}$$

$$\sum_{j \in I_0} z_{ijmk} \leq 1, \qquad \forall i \in I, m \in \mathcal{M}, k \in V, \tag{2.3}$$

$$\sum_{i \in I_0} z_{ijmk} - \sum_{i \in I_0} z_{jimk} = 0, \qquad \forall j \in I_0, m \in \mathcal{M}, k \in V. \tag{2.4}$$

Constraints (2.5) make sure that a tour with a certain index cannot be executed unless all of the tours with smaller indices are executed. Constraints (2.6) guarantee that if vehicle k leaves the depot to execute a tour, then it will return back to the depot. Finally, constraints (2.7) ensure that if vehicle k does not leave the depot to execute tour m, none of the donation sites will be visited on that tour,

$$e_{mk} \leq e_{(m-1)k}, \qquad \forall m \in \mathcal{M} \setminus \{1\}, k \in V, \tag{2.5}$$

$$\sum_{j \in I} z_{0jmk} + \sum_{j \in I} z_{j0mk} = 2e_{mk}, \qquad \forall m \in \mathcal{M}, k \in V, \tag{2.6}$$

$$\sum_{i \in I_0} \sum_{j \in I_0} z_{ijmk} \leq A e_{mk}, \qquad \forall m \in \mathcal{M}, k \in V. \tag{2.7}$$

In ICASP, we utilize a clustered structure, where all the donation sites in the same

cluster are visited by the same vehicle. Also, a vehicle visits donation sites in one cluster only. We add constraints (2.8) to cluster the donation sites,

$$\sum_{j\in I_0}(z_{jimk}+\sum_{n\in\mathcal{M}}\sum_{\substack{l\in V\\ l\neq k}} z_{jinl})\leq 1, \qquad \forall i\in I, m\in\mathcal{M}, k\in V. \quad (2.8)$$

2.4.2 Arrival Time Constraints

Constraints (2.9)–(2.11) determine the arrival time of a vehicle to a donation site in a tour,

$$x_{imk}+f_i+t_{ij}\leq x_{jmk}+A(1-z_{ijmk}), \qquad \forall i,j\in I, m\in\mathcal{M}, k\in V, \quad (2.9)$$

$$w_{(m-1)k}+f_0+t_{0j}\leq x_{jmk}+A(1-z_{0jmk}), \qquad \forall j\in I, m\in\mathcal{M}\setminus\{1\}, k\in V, \quad (2.10)$$

$$a_0+t_{0j}\leq x_{j1k}+A(1-z_{0j1k}), \qquad \forall j\in I, k\in V. \quad (2.11)$$

A donation site cannot be visited before its opening time. This is guaranteed by constraints (2.12). Constraints (2.13) make sure that the visiting time of a donation site in the first tour of a vehicle is set to a_i if it is not visited,

$$a_i\leq x_{i1k}, \qquad \forall i\in I, k\in V, \quad (2.12)$$

$$a_i+A\sum_{j\in I_0} z_{ji1k}\geq x_{i1k}, \qquad \forall i\in I, k\in V. \quad (2.13)$$

The arrival time of a vehicle to the depot, after executing a tour, is determined by constraints (2.14)–(2.15),

$$x_{imk}+f_i+t_{i0}\leq w_{mk}+A(1-z_{i0mk}), \qquad \forall i\in I, m\in\mathcal{M}, k\in V, \quad (2.14)$$

$$w_{mk}+f_0\leq b_0, \qquad \forall m\in\mathcal{M}, k\in V. \quad (2.15)$$

If donation site i is not visited on tour m by vehicle k, then we set x_{imk} equal to

the visiting time of this donation site in the previous tour by the same vehicle, i.e., $x_{i(m-1)k}$. This is guaranteed by constraints (2.16)–(2.17),

$$x_{i(m-1)k} + A \sum_{j \in I_0} z_{jimk} \geq x_{imk}, \qquad \forall i \in I, m \in \mathcal{M} \setminus \{1\}, k \in V, \quad (2.16)$$

$$x_{i(m-1)k} \leq x_{imk}, \qquad \forall i \in I, m \in \mathcal{M} \setminus \{1\}, k \in V. \quad (2.17)$$

Finally, we add constraints (2.18) to make sure that when a donation site is visited, a positive amount of blood units is picked up for platelet extraction from that donation site,

$$w_{mk} + f_0 \leq x_{imk} + S + A(1 - \sum_{j \in I_0} z_{jimk}), \qquad \forall i \in I, m \in \mathcal{M}, k \in V. \quad (2.18)$$

2.4.3 Collection Amount Constraints

In order to calculate the number of appointments that can be scheduled and collected when a vehicle visits a donation site, we need to find the latest time (v_{imk}) by which all the donations, to be picked up on this tour, are completed. Constraints (2.19)–(2.20) make sure that the donations to be picked up on a tour are completed by no later than the visiting time of the vehicle and the closing time of the donation site,

$$v_{imk} \leq x_{imk}, \qquad \forall i \in I, m \in \mathcal{M}, k \in V, \quad (2.19)$$

$$v_{imk} \leq b_i, \qquad \forall i \in I, m \in \mathcal{M}, k \in V. \quad (2.20)$$

Similar to the discussion above, about the value of x_{imk}, when donation site i is not visited by vehicle k in tour m, we add constraints (2.21) to make sure that v_{imk}

is equal to $v_{i(m-1)k}$ if donation site i is not visited by vehicle k in tour m,

$$v_{i(m-1)k} \leq v_{imk}, \qquad \forall i \in I, m \in \mathcal{M} \setminus \{1\}, k \in V, \quad (2.21)$$

$$v_{imk} \leq v_{i(m-1)k} + A \sum_{j \in I_0} z_{jimk}, \qquad \forall i \in I, m \in \mathcal{M} \setminus \{1\}, k \in V. \quad (2.22)$$

Using v_{imk}, the amount of donated units that can be picked up from donation site i in tour m by vehicle k is determined by constraints (2.23)-(2.25),

$$O_{imk} \leq D_i - \sum_{n<m} O_{ink}, \qquad \forall i \in I, m \in \mathcal{M}, k \in V, \quad (2.23)$$

$$O_{imk} \leq C_i \min \left\{ \left\lfloor \frac{v_{imk} - v_{i(m-1)k}}{q} \right\rfloor, \left\lfloor \frac{v_{imk} - a_i}{q} \right\rfloor \right\}, \qquad \forall i \in I, m \in \mathcal{M}, k \in V, \quad (2.24)$$

$$O_{imk} \leq \left\lceil \frac{v_{imk} - (w_{mk} + f_0 - S)}{q} \right\rceil C_i + A(1 - \sum_{j \in I_0} z_{jimk}), \qquad \forall i \in I, m \in \mathcal{M}, k \in V. \quad (2.25)$$

Note that constraints (2.24)–(2.25) are not linear, but we can linearize them using auxiliary integer variables r_{imk} and p_{imk} as follows:

$$O_{imk} \leq r_{imk} C_i, \qquad \forall i \in I, m \in \mathcal{M}, k \in V, \quad (2.26)$$

$$r_{imk} \leq \frac{v_{imk} - v_{i(m-1)k}}{q}, \qquad \forall i \in I, m \in \mathcal{M}, k \in V, \quad (2.27)$$

$$r_{imk} \leq \frac{v_{imk} - a_i}{q}, \qquad \forall i \in I, m \in \mathcal{M}, k \in V, \quad (2.28)$$

$$O_{imk} \leq p_{imk} C_i + A(1 - \sum_{j \in I_0} z_{jimk}), \qquad \forall i \in I, m \in \mathcal{M}, k \in V, \quad (2.29)$$

$$p_{imk} - 1 \leq \frac{v_{imk} - (w_{mk} + f_0 - S)}{q} \leq p_{imk}, \qquad \forall i \in I, m \in \mathcal{M}, k \in V. \quad (2.30)$$

2.4.4 Non-negativity and Integrality Constraints

Finally, constraints (2.31)–(2.36) represent the non-negativity and integrality restrictions of the decision variables,

$$x_{imk}, v_{imk} \geq 0, \qquad \forall i \in I, m \in \mathcal{M}, k \in V, \quad (2.31)$$

$$O_{imk} \geq 0, \qquad \forall i \in I, m \in \mathcal{M}, k \in V, \quad (2.32)$$

$$z_{ijmk} \in \{0, 1\}, \qquad \forall i, j \in I_0, m \in \mathcal{M}, k \in V, \quad (2.33)$$

$$w_{mk} \geq 0, \qquad \forall m \in \mathcal{M}, k \in V, \quad (2.34)$$

$$e_{mk} \in \{0, 1\}, \qquad \forall m \in \mathcal{M}, k \in V, \quad (2.35)$$

$$r_{imk}, p_{imk} \geq 0 \text{ and integer }, \qquad \forall i \in I, m \in \mathcal{M}, k \in V. \quad (2.36)$$

The mixed integer linear formulation (MILP) is composed of (2.1)–(2.23) and (2.26)–(2.36).

2.5 Heuristic Approaches

In this section, we discuss two types of heuristic algorithms. In both algorithms, we first cluster the donation sites using a variant of the well-known k-means clustering algorithm. Then, in *Integer Programming Based Algorithm*, we solve a single vehicle version of ICASP for each cluster. In *Construction Based Heuristic Algorithm*, we construct a solution using insertion techniques and then improve the solution by exchanging the donations (neighborhood search) between the clusters.

2.5.1 Clustering Phase

In this section, we use a variant of the k-means clustering algorithm to cluster the donation sites in to L clusters, where L represents the number of vehicles. Each vehicle is going to collect donated units from the donation sites in one cluster only.

Since our objective is to schedule appointments and collections in order to maximize the total platelet production while considering the processing time limitation, we cluster the donation sites based on geographical proximity.

In the clustering phase, we start with randomly assigning one donation site to each cluster. This first assigned donation site is set as the center of that cluster. Then, the remaining donation sites are assigned to the closest clusters where the distance between a cluster and a donation site is calculated as the distance from the cluster center to the donation site. When all the donation sites are assigned to a cluster, we call this cluster structure a *clustering solution*. The quality of a clustering solution is determined based on a *cluster distance* criteria which is calculated as the summation of the distances between the processing center and the cluster centers and the distance of each donation site to its cluster center. Assuming I^k is the set of donation sites in cluster k and j_k is the center of cluster k, the cluster distance of this clustering solution is equal to $\sum_{k \in V} (t_{0j_k} + \sum_{i \in I^k} t_{ij_k})$. We run the clustering algorithm multiple times (100 in our case) and calculate the cluster distance for each clustering solution. Then, we choose the clustering solution with the smallest cluster distance. We expect to increase the quality of the clustering solution by implementing the clustering algorithm multiple times.

2.5.2 Integer Programming Based Algorithm

In the first algorithm, called *Integer Programming Based Algorithm* (IPBA), we solve a single vehicle version of ICASP for each cluster using MILP proposed in Section 2.4. After clustering phase, the size of the problem reduces significantly which is important in terms of run time of MILP. Note that every time a new clustering solution is generated at the end of clustering phase, we find a different solution if we use IPBA. Therefore, we implement clustering phase 100 times and solve IPASP for each cluster in all these 100 clustering solutions. Then, we report the best solution

found.

2.5.3 Construction Based Heuristic Algorithm

The second algorithm we propose is called the *Construction Based Heuristic Algorithm* (CBHA). In CBHA, we first solve the single vehicle problem for each cluster using an insertion technique. Then, after finding the appointment and collection schedules for each cluster, we implement an improvement idea to further improve the solution by switching the donation sites between clusters.

First, we propose an heuristic algorithm, called *Single Tour Heuristic*, to find a single tour that visits the donation sites to pickup as many as donated units as possible. The pseudocode for this algorithm is provided in Algorithm 8. We use $\tilde{I}$ to denote the donation sites in the cluster under consideration.

Algorithm 8 *Single Tour Heuristic* (STH)

1: The tour is initially empty: $\Gamma = \emptyset$
2: Set $U = \tilde{I}$, $F = \emptyset$
3: Calculate M_{ij} the total amount that can be collected on a tour when donation site i is inserted into the jth position in tour Γ where $j \in \{1, \ldots, |\Gamma| + 1\}$ ($|\Gamma|$ = the number of donation sites in the tour)
4: $MaximumCollected = \max_{i \in U, j \in \{1,\ldots,|\Gamma|+1\}} M_{ij}$
5: $(ChosenSite, ChosenPosition) = \text{argmax}_{i \in U, j \in \{1,\ldots,|\Gamma|+1\}} M_{ij}$
6: **if** $MaximumCollected > 0$ **then**
7: Update the set of remaining donation sites: $U = U \setminus \{ChosenSite\}$, insert donation site $ChosenSite$ into position $ChosenPosition$ in tour Γ, and go to Step 3
8: **else**
9: Terminate
10: **end if**

In Algorithm 8, we start with finding the best tour among different tours of size 1. Each tour consists of the depot and one of the centers. For example, assume a tour that starts from the depot to center i and then ends in the depot. Then we start calculating M_{ij} for every tour, which is M_{i1} when tour size is 1. One important aspect is to determine the best time to visit a center. A donation center can be visited as late as b_i or as early as the beginning of the working time plus a donation unit time $(a_i + q)$, to ensure that at least one set of donation package is available to be picked

up.

To find the maximum number of donations that can be picked up from a single center (M_{i1}), we calculate $\lfloor \frac{b_i-(a_i+q)}{q} \rfloor$ values, where q represents unit donation time and this fraction shows total number of unit donation time available within working hours of a center. For example, if we assume that we visit the center i at b_i, then we can collect at most $\min\{D_i, \lfloor \frac{b_i-\max\{a_i,b_i-(S-(t_{i0}+f_i+f_0))\}}{q} \rfloor\} C_i$, where $\lfloor \frac{b_i-\max\{a_i,b_i-(S-(t_{i0}+f_i+f_0))\}}{q} \rfloor C_i$ shows the maximum number of donation packages that we can collect considering the capacity of the center and the processing time limit (S).

Following the same approach, we calculate the remaining $\lfloor \frac{b_i-(a_i+q)}{q} \rfloor - 1$ values, where the only difference is on the time of visit (here b_i), which will be decreased by constant q values each time until the first time that it becomes smaller than or equal to $a_i + q$. Among these values the maximum number will determine M_{i1}. The same method is used to find the M_{i1} for all $i \in I$.

Next step, is to choose which site could form a tour with the largest M_{i1}. The corresponding donation site will be fixed as a member of final tour. Then, we try to find a tour of larger size. For any other site that is not assigned to the final tour, we try all the possible sequences of adding a new site to the tour. For example, if we have a tour of size 1 consist of donation center i, then there are two possibilities for any center k to be added to this tour. We can either visit the center k first and then visit center i or the reverse order. In this specific case, we need to determine (1) the best position to add a new center, and (2) the best time to visit the centers.

For example, assume we try to calculate the M_{k1} when we add center k to a tour of size 1 that includes site i. There are $\lfloor \frac{b_i-(a_i+q)}{q} \rfloor$ values that we need to calculate, however to calculate M_{k2} there are $\lfloor \frac{b_k-(a_k+q)}{q} \rfloor$ values that are required to be calculated. This means that in our algorithm, the last center on the tour defines the number of calculations required.

In other words, the earliest time a tour of size n can be visited is the start of working time of the last site on the tour plus the donation unit time. This method ensures that we do not visit the last center without collecting any donations. For a tour of size 2 (e.g. depot–k–i–depot), if we visit the second site at b_i, then the total amount of donated blood that can be picked up is $\min\{D_i, \lfloor \frac{b_i - \max\{a_i, b_i - (S-(t_{i0}+f_i+f_0))\}}{q} \rfloor$ $C_i\} + \min\{D_k, \lfloor \frac{(b_i-(t_{ki}+f_k)) - \max\{a_k, b_i-(S-(t_{i0}+f_i+f_0))\}}{q} \rfloor\}$, where the first element shows the total amount that can be picked up from the center i, when it is visited second on tour and the second element represents the total amount of donated blood that can be collected from the center k when it is visited first on tour.

In order to find the remaining $\lfloor \frac{b_i-(a_i+q)}{q} \rfloor - 1$ values, we decrease the b_i in the above expressions by q every time until it becomes smaller than or equal to $a_i + q$. Then we set M_{k1} to the largest number among all these $\lfloor \frac{b_i-(a_i+q)}{q} \rfloor - 1$ values. This process continues and later the tour size increases one by one, we terminate the algorithm when adding one more center starts does not improve the amount that can be collected. For each tour size, we keep the tour with largest total amount collected. The tour with largest amount collected is reported as the solution of this algorithm.

Next, using STH as a subroutine, we propose the *Multi-Tour Heuristic* (MTH), which finds multiple tours for a single vehicle in each cluster with the objective of collecting as many donated units as possible. The pseudocode for MTH is provided in Algorithm 9.

The main idea behind MTH is to construct one tour at a time using STH. Note that the STH solution is a single tour that shows the order of visit, the time to visit the last center and the amount can be collected from the center. Using the tour information, we can easily determine the time each center is visited and how many donations are collected and which time intervals are available before the vehicle visit time to schedule appointments. For example, assume there is a tour

Algorithm 9 *Multi-Tour Heuristic* (MTH)

Set the number of tours to zero: $t = 0$, and initialize the available vehicle time: $[a_0, b_0]$
Run STH to find a single tour.
if A tour is found **then**
  if The tour time is not in conflict with the VehicleAvilableTime **then**
    Update the number of tours constructed so far: $t = t + 1$
    Update the number of remaining donations not scheduled/picked up (D_i^r) for donation site i and the time interval from which donated blood of the site is collected.
    Replace the donation sites that are visited in the tour with two donation sites and allocate the remaining number of donations to the new sites.
    Update VehicleAvailableTime (remove the traveling time of previous tour). Set U to all the sites in I and newly added donation sites, and fix F to empty set.
    Go to Step 2.
  else
    Terminate the algorithm.
  end if
else
  Terminate the algorithm.
end if

of size 2 (e.g. depot–k–i–depot) and we can pick up a greater number of donations if we visit the second site at b_i, then following the explanation on STH algorithm, we know $\min\{D_i, \lfloor \frac{b_i - \max\{a_i, b_i - (S - (t_{i0} + f_i + f_0))\}}{q} \rfloor C_i\}$ is collected from center i and $\min\{D_k, \lfloor \frac{(b_i - (t_{ki} + f_k)) - \max\{a_k, b_i - (S - (t_{i0} + f_i + f_0))\}}{q} \rfloor\}$ is picked up from center k. If D_i or D_k is not chosen in these equations, it means that there are some remaining number of donations at each center, which is the difference of D_i or D_k with what is collected. Also, the first time to collect any donation is set to $\max\{a_i, b_i - (S - (t_{i0} + f_i + f_0))\}$ for center i and is set to $\max\{a_k, b_i - (S - (t_{i0} + f_i + f_0))\}$ for center k.

At this stage, we check whether the tour (considering the start and end time of the tour) is not in conflict with the previously constructed tours. If it does not conflict, we add it to the tour list and update vehicle available time. Otherwise, we terminate the algorithm. For example, for the same tour (depot–k–i–depot), the tour traveling time is $[b_i - (t_{ki} + f_k) - (t_{0k}), b_i + (f_i + t_{i0} + f_0)]$.

After a tour is constructed, for the donation sites visited in this tour, we remove the time interval for which the appointments are scheduled and then consider the two separate time intervals as two separate donation sites. We allocate the remaining donations of this donation site to these two "new" donation sites according to their

available time intervals. For example, consider a donation site with capacity 2 which operates from 8am to 12:30pm. Assume that the total number of donations is 16 and a single donation takes 0.5 hours. If a vehicle visits this donation site at 11:30am collecting only 6 donated units. This means that blood donated from 10am to 11:30am is collected, and the 10 donations left. At this point, we replace the donation site with two new donation sites: first one operating from 8am to 10am and the second one operating from from 11:30am to 12:30pm. The capacities of these two donation sites are same as the original one. The remaining number of donations is divided between these two donation sites according to their operating hours. In this example, the operating hours of first new center is 2 times the second new site. Therefore, $\frac{2}{3}$ of the remaining donations will be allocated to the first site. When we have fractional numbers, we always round up the fractional number for the center with the larger time interval. Note that we always respect the capacity restrictions for both time intervals. In this example, the first site has 7 donors and the second one has 3 donors.

Finally, after forming the appointment and pickup schedules for all the clusters using MTH, we try to improve the solution further by switching the donation sites between clusters. In this *Neighborhood Search* (NS) step, we randomly choose two clusters and switch two donation sites from these clusters.

This process is repeated at each iteration while we calculate the number of donation units that can be collected with new clustering solution. At the end of each iteration, if we have reached to a maximum number of iterations (e.g. 500), we will generate two random numbers from $[0, 1]$ called p_1 and p_2, if $p_1 \geq p_2$ we stop the improvement process. Otherwise, we continue switching to the neighborhood solution even if the solution does not improve. An outline of NS is provided in Algorithm 10.

Similar to IPBA, we try to improve the performance of CBHA by switching donation centers at each iteration of NS and examining a new clustering solution.

Algorithm 10 *Neighborhood Search* (NS)

1: Set *NumberOfCluster* to L, generate a random clustering solution. Set *MaxCollected* and BestCanBeCollected to 0. Set number of iterations to 0 (*iter*).
2: Run MMH, set *AmountCanBeCollected* to the solution of MMH.
3: **if** *AmountCanBeCollected* $>$ *MaxCollected* **then**
4: set *MaxCollected* = *AmountCanBeCollected*
5: **end if**
6: **if** *BestCanBeCollected* $<$ *MaxCollected* **then**
7: set *BestCanBeCollected* = *MaxCollected*
8: **end if**
9: **if** *iter* > 500 **then**
10: Generate two random numbers from $[0, 1]$ (p_1 and p_2).
11: **if** $p_1 \geq p_2$ **then**
12: report *BestCanBeCollected* and terminate.
13: **else**
14: randomly pick two regions and switch two arbitrarily selected donation sites between two clusters. Set *MaxCollected* to 0 and $iter = iter + 1$ and go to step 2.
15: **end if**
16: **else**
17: Randomly pick two regions and switch two arbitrarily selected donation sites between two clusters. Set *MaxCollected* to 0, $iter = iter + 1$ and go to step 2.
18: **end if**

2.6 Computational Results

To test the performance of our proposed mathematical model and solution algorithms, we conduct computational experiments on 30 instances (to represent a complete month of data) generated according to the real data provided online [33]. Gulf Coast Regional Blood Center was serving 17 donation centers during the continuous months that we were observing the data. Although, they have closed one of the centers recently, we are using 17 donation centers because of our observation time interval. According to the 4 years of previous data and one month current data, using ARENA Input Analyzer version 12.0, the daily average number of donations is following a Normal distribution with mean of 878 whole blood units (including the number of donations from the processing center) and standard deviation of 153. In our calculations we use 3 vehicles, which work from 9am to 7pm and we assume there is not any limitations on the availability of drivers and vehicles. For these experiments, we suppose that donation unit time is fixed to an hour and service time of all the centers is 0.2 hour. The average donation unit time is reported to be around

50 minutes [40], however we add 10 more minutes for the purpose of considering transferring time from one donor to the next donor and cleaning of the resources for consecutive donations. The service time at each center can be set to any number as it is representing a deterministic parameter. In these scenarios the average truck speed is set to 45 miles per hour [50], to account for the average driving speed at the neighborhood of all the donation centers in Houston, TX.

The algorithms are implemented in Microsoft Visual C# using SQL Server as the database platform. CPLEX 12.3, with default settings, is used as the optimization engine. All the experiments are carried on a system with two 2.4 GHz Xeon processors and 4 GB RAM.

To calculate the comparison gap of a solution found by the mathematical model or proposed heuristic approaches, we need to find the optimal solution or an upper bound for the ICASP. We set an 8-hour time limit on MILP formulation in Section 2.4 and report the solution found at the end of 8 hours. The model was not able to solve any of the scenarios to optimality within the pre-determined solution time limit. Therefore, to perform a fair comparison, we use the minimum of the summation of number of donations in all the 17 donation centers on a day and the upper bound (best LP relaxation solution) found by MILP as an upper bound (denoted by UB) to calculate the comparison gap. The comparison gap of a solution found by the mathematical model or any of the heuristic algorithms (denoted by Sol) is calculated as follows,

$$\text{Comparison Gap} = \frac{UB - Sol}{UB} \times 100\%. \tag{2.37}$$

We summarize the computational results in Table 2.1. The table represents the comparison gaps of all the 30 scenarios. In this table, values under SC column are the scenario numbers and ND are the total estimated number of donors showing up for donation in all the 17 donation centers on a day.

Table 2.1 shows that the ICASP is unable to find a good solution within 8 hours.

Table 2.1: Comparison gaps of solutions found by ICASP–MIP, IPBA and CBHA for 30 scenarios.

SC	ND	ICASP–MIP	IPBA		CBHA	
		Gap(%)	Gap(%)	Time(min)	Gap(%)	Time(min)
1	533	5.25	2.25	16.12	4.32	0.34
2	593	10.62	4.38	6.43	8.60	0.43
3	600	15.33	2.17	6.83	11.33	0.32
4	633	13.27	2.21	11.10	5.06	0.35
5	644	12.42	3.73	8.00	6.99	0.49
6	669	12.56	2.99	7.43	8.97	0.31
7	695	12.09	4.03	15.83	10.50	0.37
8	699	11.59	2.29	7.16	8.44	0.33
9	729	11.25	2.19	7.56	9.60	0.35
10	755	16.82	1.06	6.86	5.03	0.39
11	769	11.57	2.21	5.84	8.32	0.35
12	780	15.64	2.18	6.07	8.72	0.31
13	784	19.52	2.93	7.46	6.38	0.33
14	794	21.28	2.27	7.18	4.16	0.31
15	801	5.37	2.25	19.59	4.00	0.41
16	810	16.17	2.22	15.08	6.67	0.32
17	816	10.42	2.94	11.57	6.62	0.30
18	831	9.99	0.24	11.81	2.29	0.29
19	838	15.75	4.65	13.33	11.58	0.35
20	844	18.01	1.78	14.50	5.21	0.45
21	852	24.06	2.35	11.25	5.28	0.39
22	882	19.73	0.45	17.61	6.69	0.40
23	891	17.85	2.92	16.79	7.86	0.40
24	905	19.56	2.76	6.41	11.38	0.34
25	920	22.50	0.65	6.02	12.61	0.37
26	937	21.77	2.99	20.00	6.51	0.42
27	941	23.27	2.98	16.47	8.18	0.47
28	999	18.92	6.51	21.14	12.71	0.32
29	1068	23.22	5.15	22.99	9.46	0.48
30	1088	30.61	4.60	14.75	8.18	0.48
Average		16.21	2.74	11.97	7.72	0.37

The average comparison gaps are 19.76%, 17.17%, 16.21% and 16.21% after running the ICASP model with 1-hour, 2-hour, 4-hour and 8-hour time limits respectively. According to our observations, a scenario can be solved with ICASP to optimality within 3 to 4 days and that is only possible if the system does not get to an out-of-memory status meanwhile. However, IPBA is able to find a better solution with respect to the solution quality and time in comparison with ICASP–MIP. The average

solution time of IPBA is very dependent on the number of iterations we run the algorithm. The average comparison gaps of IPBA are 3.25% and 2.74% when we set the number of iterations to 50 and 100 respectively, while the average solution times are 3.41 and 11.97 minutes respectively. Note that if we increase the number of iterations to 150 or 200, the comparison gaps remain the same for all the scenarios, while the solution time increases rapidly (e.g., 4 to 5 hours). Therefore we only report the gaps for 100 iterations in the Table 2.1. CBHA is a very fast algorithm and can find a good feasible solution within a half minute even when running the algorithm for 500 iterations, however the quality of solutions is not as good as IPBA. Note that CBHA is run for 600 and 700 iterations as well and no improvement is observed. On average, CBHA improves the average solution time at least 11 minutes, while the average comparison gap increases by 5%. Also, note that the solution time of the IPBA is very dependent of the clustering solution, which may result in very long solution times when the mathematical model tries to reach to the default optimality gap. To decide, which solution algorithm can be utilized to schedule the vehicle time visits for the ICASP is tied upon the preferences of the users.

2.7 Summary

In this chapter, we introduce *Integrated Collection and Appointment Scheduling Problem* (ICASP), in which our goal is to maximize the number of donated blood packages that can be collected and delivered to a processing center within a limited time for platelets extraction, and we schedule the donation appointments accordingly. First, a mixed integer programming formulation is developed. Then, we propose two heuristic algorithms: (i) *Integer Programming Based Algorithm* (IPBA), and (ii) *Construction Based Heuristic Algorithm* (CBHA). We perform a computational study to test the performance of the proposed algorithms in terms of solution quality and computational efficiency on the instances from Gulf Coast Regional Blood Center located

in Houston, TX. The results show that the proposed algorithms provide good feasible solutions for ICASP in significantly less time than the mathematical formulation (MILP). The MILP can not find optimal solution for none of the instances within the specific time limits, while IPBA is able to find the best solution among the proposed approaches within a reasonable amount of time and CBHA can generate relatively good feasible solutions in less than a minute.

Chapter 3 Robust Optimization and Chance Constrained Programming for the Collection Problem with Uncertainty

3.1 Introduction and Literature Review

In the blood collection operations, we are dealing with a problem of determining the routing schedule of a set of vehicles from a processing center to several donation centers in geographically different locations and we try to collect donated blood units within a limited time and to deliver the packages for platelet extraction. This problem is explained comprehensively in Chapter 2. In Section 2.3, we assume that the number of donors showing up daily for donation is deterministic and estimated based on the historical data. However, donors may not show up for donation on the specific day that their donations are scheduled. Also, it is possible that a larger number of donors than what is estimated try to schedule appointments for a given day. These variability and congestion in supply affect the amount that can be collected for platelet production and hence reduce the reliability of the system and increase the cost of shortages/out-dates.

Therefore, it is necessary to develop routing schedules that can account for the uncertainty in the number of donors showing up daily. Generally, current methodologies apply stochastic optimization approaches to account for uncertainty of data. In stochastic optimization, the uncertain data are assumed to be random and follow a known probability distribution. For example, Chance Constrained Programming is a stochastic approach that is first introduced by Charnes et al. [13]. In more advanced settings, there might not be enough information to estimate the probability distribution. In such cases, we can use a methodology called Robust Optimization that is first introduced by Soyster [44]. Unlike any other stochastic optimization method, we are

not required to know an exact probability distribution of an uncertain data. In such cases, a partial knowledge about this distribution function is enough. For example, the data uncertainty can be represented with an uncertainty set that the actual data is within a pre-specified range. Also, solutions of a stochastic approach are only feasible when the estimated distribution function is valid. However, the solutions of a Robust Optimization method is robust feasible for all the possible scenarios of data within the uncertainty set. In other words, robust optimization generally follows a "worst-case-oriented" philosophy. This philosophy results in more conservative solutions compared to a stochastic methodology. Therefore, if we prefer to narrow the uncertainty set and utilize an exact distribution function, it is suggested in the literature to apply a stochastic optimization approach like Chance Constrained Programming. Otherwise, we can take advantage of Robust Optimization techniques to find a robust feasible solution for all the possible scenarios of a specific problem. In this chapter, to account for the uncertain number of donors showing up for donation, we apply both Robust Optimization and Chance Constrained Programming techniques for the Integrated Collection and Appointment Scheduling Problem (ICASP).

The most related problem to ICASP is the general Vehicle Routing Problem (VRP). In the literature, there are many studies that are concerned with the uncertainty of customers and demand in VRP [46]. Sungur et al. [46] utilize robust optimization approach and chance constrained programming for a capacitated vehicle routing problem with uncertain demand. They use the approaches proposed in [9] and [10] and consider different types of uncertainty sets. They propose robust mathematical models and a clustered algorithm to find solutions with a lower level of unmet demand and additional cost over deterministic solutions. They compare their results with stochastic VRP models. Also, several studies focus on uncertainty of blood supply chain [5], which are mostly about the inventory planning in the blood supply chain.

The remainder of this chapter is as follows. In Section 3.2, we briefly define the problem. In Sections 3.3 and 3.4, we present the derivations of the Robust and Chance Constrained Programming formulations for ICASP with uncertainty. The computational results are provided in Section 3.5. We complete the chapter with concluding remarks in Section 3.6.

3.2 Problem Definition and Mathematical Models

In this chapter, we focus on a problem instance of *Integrated Collection and Appointment Scheduling Problem* (ICASP) that contains N donation sites and a single processing center ($I_0 = \{1, 2, \ldots, N\}$). We assume that D is a column vector that represents the number of donors showing on a specific day. D is uncertain and belongs to a bounded set $\mathcal{U}$. The expected number of donors for each center is shown by d_i^0. We assume that it takes exactly q units of time to complete donation. The capacity of donation site i is C_i, where it is defined as the number of donations that site can handle per q units of time. Limited number of staff or equipments at each donation site determines C_i. $[a_i, b_i]$ is the working hours of each donation center. t_{ij} is traveling time from center i to center j where $i, j \in I \cup \{0\}$. It is assumed that t_{ij} is nonnegative and satisfy triangular inequality.

The 6 hour collection time limit is called *processing time limit* (for platelets extraction) and we show it with S. We are required to deliver as much donated blood as we can to the depot for extraction. We have L vehicles that start their tours from depot, travel to a subset of donation sites and collect available donated blood which can be used for platelets extraction and then return to the depot. More details are provided in Chapter 2.

Previously we assumed that we can schedule the appointments without having any conflicts with the donors' preferences. It means that the donors show up with 100% probability at the requested appointment time. However, in real world scenarios,

some donors may not be able to show up for donation at the time of their scheduled appointments or some people may decide to donate blood without informing the donation center and walk-in for donation. In these two cases, the number of donors is uncertain. Without loss of generality, we assume that we have some information about the estimated number of donors that are showing up. We assume a lower bound and an upper bound for the number of donors. The lower and upper bounds determine an uncertainty set, called $\mathcal{U}$, for all d_i values.

Given the capacities of donation sites and the number of donors showing up at each center, the first goal of this problem is to determine when vehicles should visit donation sites in order to deliver maximum amount of donated blood to the depot. The second goal is to determine the best schedule of donations at each donation site with respect to the visiting times of vehicles. Similar to Chapter 2, when we find a solution that maximizes the amount of blood collected considering the capacities of donation centers, number of donors and center's working hours, we can determine the schedule of appointments accordingly with the knowledge of donation unit time, working hours and capacities. In this chapter, we apply robust optimization and chance constrained programming techniques utilizing both the mathematical models proposed for ICASP (see Section 2.4) and IPBA (see Section 2.5.2).

3.3 Robust Optimization

Current methods of representing the uncertainty in routing problems make strong assumptions about the distribution of the uncertain information. For example, they assume that the distribution function of the uncertain data is either known or estimated. In such cases, stochastic programming can be used to handle the uncertainty. However, for a system that is strongly dependent on the human behaviors, it is not easy to determine such probability functions in practice. Instead, it might be possible to estimate the bounds of the uncertain parameters using the historical data.

Considering the bound of parameters, we can utilize robust optimization approaches to generate good solutions for the ICASP. Unlike stochastic programming models that address a single case of the uncertainty (specific distribution function), robust optimization can obtain a robust solution that is good for all the possible scenarios of uncertain data. Note that we apply robust optimization method introduced by Ben-Tal and Nemirovski [9] and utilized for a capacitated VRP by Sungur et al. [46].

In this chapter, we assume that the expected number of donors for each center is shown by d_i^0 and the possible deviation from these expected values are fixed at each scenario (s) and represented by d_i^s. The deviation can have negative, zero or positive values. We suppose $\mathcal{P}$ is the set of all possible scenarios and the general uncertainty set $\mathcal{U}$ is $\mathcal{U}_{d_i} = \{d_i^0 + \sum_{s=1}^{|\mathcal{P}|} y_s d_i^s, y_s \in \mathbb{R}\}$, where y_s is the weight of scenario s. Note that Y shows the column vector of y_s values ($Y \in \mathbb{R}^{|\mathcal{P}|}$).

The bounded set of y_s can be a Convex Hull, where $\{Y \in \mathbb{R}^{|\mathcal{P}|} | Y \geq 0, \sum_{s=1}^{|\mathcal{P}|} y_s \leq 1\}$ or it can a Box Set as $\{Y \in \mathbb{R}^{|\mathcal{P}|} | \|y_s\|_\infty \leq 1\}$ or an Ellipsoidal Set as $\{Y \in \mathbb{R}^{|\mathcal{P}|} | y^T A y \leq 1\}$, where A is a positive definite matrix.

The *Robust Integrated Collection and Appointment Scheduling Problem* (RICASP) can be formulated as a mixed integer linear model similar to the model proposed for ICASP (Chapter 2). The only difference is on constraints (2.23). When the number of donors is uncertain, the constraints (2.23) can be rewritten as follows:

$$O_{imk} + \sum_{n \in \mathcal{M}; n<m} O_{ink} - d_i^0 \leq \sum_{s=1}^{|\mathcal{P}|} y_s d_i^s, \qquad \forall i \in I, m \in \mathcal{M}, k \in V. \tag{3.1}$$

For a set of given decision variables O, the left hand side of the constrains (3.1) is referred to $\phi_{imk} = O_{imk} + \sum_{n \in \mathcal{M}; n<m} O_{ink} - d_i^0$ for all $i \in I, m \in \mathcal{M}, k \in V$. Similar to Sungur et al. [46] and in order to follow the "worst-case-oriented" philosophy of the classic robust optimization approach, it is enough to force the feasibility of constrains (3.1) ($\phi_{imk} \leq \sum_{s=1}^{|\mathcal{P}|} y_s d_i^s$) for the $\inf_{Y \in \mathcal{U}} \sum_{s=1}^{|\mathcal{P}|} y_s d_i^s$.

Proposition 3.3.1. *If U is a convex hull, then constraints* (3.1) *can be replaced by constraints* (3.2),

$$O_{imk} + \sum_{n \in \mathcal{M}; n<m} O_{ink} \leq d_i^0 + min_s \{d_i^s\}, \qquad \forall i \in I, m \in \mathcal{M}, k \in V. \quad (3.2)$$

Proof. When U is a convex hull, $\inf_{Y \in \mathcal{U}} \sum_{s=1}^{|\mathcal{P}|} y_s d_i^s$ of each donation center $i \in I$ can be represented as a standard minimization linear primal problem $\{\text{Min.} \sum_{s=1}^{|\mathcal{P}|} y_s d_i^s \mid (-\sum_{s=1}^{|\mathcal{P}|} y_s) \geq -1 \ \& \ y_s \geq 0; \forall s \in \{1, \ldots, |\mathcal{P}|\}\}$ and a corresponding dual problem $\{\text{Max.}(-\theta) \mid -\theta \leq d_i^s; \forall s \in \{1, \ldots, |\mathcal{P}|\} \ \& \ \theta \geq 0\}$. Here θ is dual decision variable. According to the strong duality, if the primal and dual problems are feasible and bounded, then the optimal objective values of primal and dual problems are equal for the optimal solutions y^* and θ^*. Thus, $\sum_{s=1}^{|\mathcal{P}|} y_s^* d_i^s = -\theta^*$ and due to feasibility of dual problem, $\sum_{s=1}^{|\mathcal{P}|} y_s^* d_i^s \leq d_i^s$ for all $s \in \{1, \ldots, |\mathcal{P}|\}$ and therefore the constraints (3.2) is correct. Note that the infeasibility of primal or dual problem results in infeasible robust optimization model, which is a possible case when we are utilizing this robust optimization methodology [46]. □

The corresponding mathematical model with constraints (3.2) is called RICASP-C. Following Proposition 3.3.1, we can use constraints (3.3) in IPBA and develop a Robust Mathematical Model (RIPBA-C) for the integer programming based solution approach. Note that I' represents a subset of donation sites in a cluster,

$$O_{im} + \sum_{n \in \mathcal{M}; n<m} O_{in} \leq d_i^0 + \min_s \{d_i^s\}, \qquad \forall i \in I', m \in \mathcal{M}. \quad (3.3)$$

Proposition 3.3.2. *If U is a box set, then constraints* (3.1) *can be rewritten as constraints* (3.4),

$$O_{imk} + \sum_{n \in \mathcal{M}; n<m} O_{ink} \leq d_i^0 - \sum_{s=1}^{|\mathcal{P}|} |d_i^s|, \qquad \forall i \in I, m \in \mathcal{M}, k \in V. \quad (3.4)$$

Proof. Similar to the proof of Proposition 3.3.1, we first write primal and dual problem of $\inf_{Y\in U}\sum_{s=1}^{|\mathcal{P}|} y_s d_i^s$, where θ_s and γ_s represent dual variables. $\{\text{Min.} \sum_{s=1}^{|\mathcal{P}|} y_s d_i^s \mid y_s \leq 1; \forall s \in \{1,\ldots,|\mathcal{P}|\}\ \&\ y_s \geq -1; \forall s \in \{1,\ldots,|\mathcal{P}|\}\}$ represents the primal problem and $\{\text{Max.} -(\sum_{s=1}^{|\mathcal{P}|}\theta_s + \gamma_s) \mid -\theta_s + \gamma_s = d_i^s; \forall s \in \{1,\ldots,|\mathcal{P}|\}\ \&\ \theta_s, \gamma_s \geq 0; \forall s \in \{1,\ldots,|\mathcal{P}|\}\}$ is corresponding dual formulation. The two sets of constraints in primal problem are equivalent to $\|y\|_\infty \leq 1$, ($\|y\|_\infty = \max_s\{|y_s|\}$). Any feasible solution of dual problem satisfies $-\theta_s + \gamma_s = d_i^s; \forall s \in \{1,\ldots,|\mathcal{P}|\}$ and according to the strong duality we can rewrite objective value of primal problem as $\sum_{s=1}^{|\mathcal{P}|} y_s(-\theta_s + \gamma_s)$. The feasibility of primal problem suggests that $y_s \geq -1$ for all s. Therefore, we have $y_s(\gamma_s - \theta_s) \leq -(\gamma_s - \theta_s)$, for $\gamma_s - \theta_s \leq 0$. In addition, $y_s \geq 1$ for all s and thus $y_s(\gamma_s - \theta_s) \leq (\gamma_s - \theta_s)$, for $\gamma_s - \theta_s \leq 0$. If we substitute $\gamma_s - \theta_s$ with d_i^s, we can conclude the proof. □

The corresponding mathematical model with constraints (3.4) is called RICASP-B. Following the Proposition 3.3.2, we can use constraints (3.5) in IPBA and develop a Robust Mathematical Model (RIPBA-B) for the integer programming based solution approach,

$$O_{im} + \sum_{n\in\mathcal{M};n<m} O_{in} \leq d_i^0 - \sum_{s=1}^{|\mathcal{P}|} |d_i^s|, \qquad \forall i \in I', m \in \mathcal{M}. \tag{3.5}$$

Proposition 3.3.3. *If U is an ellipsoidal set, the constraints* (3.1) *will be replaced by constraints* (3.6)*, where d is a column vector of* d_s*,*

$$O_{imk} + \sum_{n\in\mathcal{M};n<m} O_{ink} \leq d_i^0 - (\frac{1}{\sqrt[2]{d^T A^{-1} d}} A^{-1} d)^T d, \qquad \forall i \in I, m \in \mathcal{M}, k \in V. \tag{3.6}$$

Proof. When U is an ellipsoidal set, $\inf_{Y\in U}\sum_{s=1}^{|\mathcal{P}|} y_s d_i^s$ can be represented as $\{\text{Min. } Y^T d \mid Y^T A Y \leq 1\}$. If we change decision variables with $x = A^{\frac{1}{2}} Y$ and $\tilde{d} = A^{-\frac{1}{2}} d$, the model will be a linear function over a unit ball ($\{\text{Min. } \tilde{d}^T x \mid x^T x \leq 1\}$), where the optimal

solution is $x^* = -\frac{\tilde{d}}{\|\tilde{d}\|}$ [11]. Thus $y^* = -\frac{1}{\sqrt[2]{d^T A^{-1} d}} A^{-1} d$, which completes the proof. $\square$

The corresponding mathematical model with constraints (3.6) is called RICASP-E. Following the Proposition 3.3.3, we can use constraints (3.7) in IPBA and develop a Robust Mathematical Model (RIPBA-E) for the integer programming based solution approach,

$$O_{im} + \sum_{n \in \mathcal{M}; n<m} O_{in} \leq d_i^0 - (\frac{1}{\sqrt[2]{d^T A^{-1} d}} A^{-1} d)^T d, \qquad \forall i \in I', m \in \mathcal{M}. \quad (3.7)$$

In this dissertation, when Y is an ellipsoidal set, we will assume that A is an identity matrix to simplify the problem and focus on studying the effects of scenario deviations.

3.4 Chance Constrained Programming

In addition to robust optimization approach, one other way of finding a solution for the ICASP with uncertainty is to use *Chance Constrained Programming*. Chance Constrained Programming is known as one of the major methodologies to deal with uncertain random parameters in Operations Research problems. The main difficulty of this approach is that the distribution functions of random parameters should be known prior to observing the random parameters.

When the number of donors that are showing for donation is uncertain and we know that at each donation center it follows a cumulative distribution function of $F_i(x)$ (for each $i \in I$, where x represents the random number of donors), then the only difference of Chance Constrained Model from the mathematical model proposed for ICASP will be constraints (2.23). These constraints can be rewritten as follows,

$$\Pr(O_{imk} + \sum_{n \in \mathcal{M}; n<m} O_{ink} \leq D_i) \geq (1 - \alpha_i), \qquad \forall i \in I, m \in \mathcal{M}, k \in V. \quad (3.8)$$

where α_i represents the probability of violating original constraint ($O_{imk} + \sum_{n \in \mathcal{M}; n<m}$

$O_{ink} \leq D_i$) at donation center i. Considering a general distribution function for the number of donors showing at site i, we can substitute constraints (2.23) with constraints (3.9),

$$O_{imk} + \sum_{n \in \mathcal{M}; n<m} O_{ink} \leq F_i^{-1}(\alpha_i), \qquad \forall i \in I, m \in \mathcal{M}, k \in V. \tag{3.9}$$

According to the scenarios generated based on Gulf Cost Regional Blood Center data, the probability distribution functions of donors showing up at all the centers are following normal distributions with mean of μ_i and standard deviation of σ_i. Therefore, if we calculate the standard score of α_i ($= \psi$), the right hand side of constraints (3.9) will be $\mu_i + \psi \times \sigma_i$. Corresponding mathematical model with constraints (3.9) is called ICASP-CCP. Following the same approach, we can use constraints (3.10) in IPBA and develop an integer programming based solution approach (IPBA-CCP) for the chance constrained formulation,

$$O_{im} + \sum_{n \in \mathcal{M}; n<m} O_{in} \leq F_i^{-1}(\alpha_i), \qquad \forall i \in I', m \in \mathcal{M}. \tag{3.10}$$

3.5 Computational Results

To test the performance of our proposed robust mathematical models, chance constrained program and clustering solution algorithms, we conduct computational experiments on 30 instances (to represent a complete month of data) generated using data from Gulf Coast Regional Blood Center [33]. This Blood Center was serving 17 donation centers during the months that we observed the data. We use same set of scenarios as in Section 2.6. Our set of experiments for Robust Optimization Models consists of 6 deviations from the expected number of donors showing up for donation at each scenario: +2, -2, +5, -5, +10 and -10 percent deviations. We run RICASP-B, RICASP-C, RICASP-E to find the optimal solutions for the possible types of

uncertainty sets and we also run RIPBA-B, RIPBA-C and RIPBA-E to find good feasible solutions for the robust problems within a reasonable amount of time. To investigate the results of Chance Constrained Programming, we find the distribution function of number of donors showing for donation at each center according to the 30 scenarios and then we set α, the probability of violating the constraints (3.9) and (3.10), to 0.01, 0.02, 0.05, 0.08 and 0.1.

In our calculations we use 3 vehicles, which work from 9am to 7pm and we assume there are not any limitations on the availability of drivers and vehicles. For these experiments, we suppose that donation unit time is fixed to an hour and service time of all the centers is 0.2 hour. The average donation unit time is reported to be around 50 minutes [40], however we add 10 more minutes for the purpose of considering transferring time from one donor to the next donor and cleaning of the resources for consecutive donations. The service time at each center can be set to any number as it is representing a deterministic parameter. In these scenarios, the average truck speed is set to 45 miles per hour [50].

The models are implemented in Microsoft Visual C# using SQL Server as the database platform. CPLEX 12.3, with default settings, is used as the optimization engine. All the experiments are carried on a system with two 2.4 GHz Xeon processors and 4 GB RAM.

To calculate the comparison gap of a solution found by the mathematical model or proposed heuristic approaches, we need to find the optimal solution or an upper bound for the RICASP. We set an 8-hour time limit on RICASP (-B,-C and -E) formulations in Section 3.3 and report the solution found at the end of 8 hours. The model was not able to solve any of the scenarios to the optimality within the pre-determined solution time limit. Therefore, to perform a fair comparison, we use the minimum of the summation of number of donations in all the 17 donation centers on a day and the upper bound (best LP relaxation solution) found by RICASP (-B,-C

and -E) as an upper bound (denoted by UB) to calculate the comparison gap. The comparison gap of a solution found by the mathematical model or any of the heuristic algorithms (denoted by *Sol*) is calculated as follows:

$$\text{Comparison Gap} = \frac{UB - Sol}{UB} \times 100\%. \tag{3.11}$$

To compare the robust and the deterministic solutions, we use the best solution found by the deterministic integer programming based algorithm (IPBA) of Chapter 2 (denoted by *DSol*) and report a gap (denoted by *d-gap*), which is calculated as follows:

$$\text{d-gap} = \frac{DSol - Sol}{DSol} \times 100\%. \tag{3.12}$$

We summarize the computational results of Robust Optimization models in Tables 3.1, 3.2 and 3.3. Each table represents the comparison gaps of all the 30 scenarios and their deviations. In all the tables, values under SC column are the scenario numbers, positive or negative signs represent a positive or negative deviation from the expected number of donors under column ND and the value adjacent to the sign shows the deviation percentage.

Tables 3.1, 3.2 and 3.3 show that among the three types of uncertainty sets, box set is generating the most conservative results while convex hull set is providing the least conservative solutions and ellipsoidal set, that represent a normal population, finds solutions with comparison gaps smaller than the results of the box uncertainty and larger than solutions of convex hull uncertainty. All the tables represent that the clustered mathematical models with any type of uncertainty - developed in Section 3.3 - can generate solutions with smaller gaps compared to robust mixed integer programming formulation. This fact is concluded in deterministic situation as well (Section 2.6) and it suggests that both the deterministic and robust approaches can benefit from pre-clustering.

Table 3.1: Comparison gaps of solutions found by RICASP-C, RIPBA-C for the six deviation scenarios of 30 instances.

SC	ND	RICASP-C							RIPBA-C						
		+10	+5	+2	±0	-2	-5	-10	+10	+5	+2	±0	-2	-5	-10
1	533	24.1	20.7	18.8	16.5	14.7	11.9	7.3	21.2	17.6	15.7	13.3	11.5	8.5	3.7
2	593	26.1	22.6	20.3	18.5	16.7	14.1	10.1	18.6	14.7	12.2	10.3	8.3	5.3	0.9
3	600	24.5	20.9	18.6	16.8	15.0	12.3	8.3	20.3	16.5	14.0	12.2	10.2	7.4	3.1
4	633	25.5	21.6	19.3	17.7	16.0	13.3	8.1	21.3	17.3	14.9	13.1	11.3	8.5	3.0
5	644	23.9	20.1	17.8	16.1	14.4	11.8	6.6	19.1	15.1	12.6	10.9	9.0	6.2	0.7
6	669	29.5	26.0	24.0	22.3	20.7	18.4	13.8	22.5	18.6	16.4	14.5	12.8	10.2	5.1
7	695	27.3	23.9	21.7	20.0	18.2	15.6	11.2	18.3	14.5	12.0	10.1	8.1	5.2	0.2
8	699	32.0	28.8	26.7	25.2	23.5	21.1	17.0	18.2	14.4	11.9	10.0	8.0	5.1	0.2
9	729	27.2	23.9	21.8	20.2	18.5	16.0	11.7	17.5	13.9	11.4	9.6	7.7	4.9	0.0
10	755	31.1	27.7	25.7	24.2	22.7	20.4	15.9	19.9	15.9	13.6	11.9	10.1	7.5	2.2
11	769	28.0	24.1	22.1	20.5	19.0	16.6	12.0	21.7	17.4	15.2	13.5	11.8	9.3	4.2
12	780	26.7	22.8	20.7	19.2	17.6	15.3	10.0	20.6	16.3	14.1	12.4	10.7	8.2	2.4
13	784	29.8	25.8	23.9	22.4	20.9	18.7	13.6	23.4	19.1	17.0	15.4	13.8	11.4	5.8
14	794	27.3	23.5	21.3	19.8	18.2	16.0	10.8	18.7	14.5	12.0	10.3	8.6	6.1	0.3
15	801	31.8	28.2	26.1	24.7	23.3	20.9	16.4	21.5	17.4	14.9	13.4	11.7	8.9	3.7
16	810	30.5	27.0	24.7	23.3	21.9	19.5	14.7	22.1	18.2	15.6	14.1	12.4	9.7	4.4
17	816	28.8	25.3	23.0	21.6	20.1	17.6	12.7	22.8	19.0	16.5	14.9	13.4	10.7	5.3
18	831	31.0	27.6	25.4	24.0	22.7	20.1	15.6	23.7	20.1	17.6	16.1	14.6	11.8	6.8
19	838	30.9	27.4	25.3	23.9	22.5	19.9	15.5	21.4	17.5	15.1	13.5	11.9	9.0	4.0
20	844	29.0	26.0	23.4	21.9	20.5	17.9	13.4	20.1	16.7	13.8	12.2	10.6	7.7	2.6
21	852	32.6	29.5	27.3	25.7	24.4	21.9	17.7	19.0	15.3	12.6	10.7	9.1	6.2	1.0
22	882	33.2	30.1	28.0	26.4	24.8	22.3	18.4	18.1	14.4	11.8	9.9	7.9	4.8	0.0
23	891	32.0	29.0	26.8	25.2	23.6	21.1	17.0	18.2	14.5	11.9	10.0	8.0	5.0	0.0
24	905	29.5	26.4	24.1	22.5	20.9	18.3	14.1	17.9	14.3	11.7	9.8	7.9	4.9	0.0
25	920	32.6	29.7	27.6	26.1	24.5	22.1	18.2	17.8	14.3	11.7	9.9	8.0	5.0	0.2
26	937	33.0	29.8	27.8	26.2	24.6	22.4	18.4	20.5	16.7	14.3	12.5	10.6	7.9	3.2
27	941	32.7	29.2	27.3	25.7	24.1	21.8	17.9	19.6	15.6	13.2	11.4	9.4	6.7	2.0
28	999	35.2	32.7	30.8	29.4	28.0	25.8	21.6	20.3	17.3	14.9	13.2	11.4	8.7	3.6
29	1068	35.3	35.3	35.3	34.1	32.7	30.6	26.9	16.6	16.6	16.6	15.1	13.3	10.6	5.8
30	1088	32.8	32.8	32.8	32.8	31.4	29.4	25.4	13.4	13.4	13.4	13.4	11.6	9.0	3.9
Average		29.8	26.6	24.6	23.1	21.5	19.1	14.7	19.8	16.2	13.9	12.3	10.5	7.7	2.6

In Table 3.1, for all the instances, the number of donations at each center – for both the robust mixed integer mathematical model and clustered mathematical model – is set to the minimum deviation from deterministic donation numbers. This means that the comparison gaps are the least possible for the -10% deviation and they increase gradually when the deviation increases. In Table 3.2, for all the instances, the number of donation is set to the deterministic number of donation at each center minus the summation of all the deviations. The box uncertainty set results in conservative robust solutions that are feasible for all of the scenarios similar to convex hull and

Table 3.2: Comparison gaps of solutions found by RICASP-B, RIPBA-B for the six deviation scenarios of 30 instances.

SC	ND	RICASP-B							RIPBA-B						
		+10	+5	+2	±0	-2	-5	-10	+10	+5	+2	±0	-2	-5	-10
1	533	43.5	41.0	39.6	37.9	36.6	34.5	31.0	41.8	39.2	37.8	36.0	34.7	32.5	29.0
2	593	45.1	42.5	40.8	39.5	38.1	36.1	33.1	41.7	38.9	37.1	35.7	34.3	32.2	29.0
3	600	44.2	41.5	39.8	38.5	37.1	35.1	32.2	41.1	38.3	36.5	35.2	33.7	31.6	28.5
4	633	44.5	41.6	40.0	38.7	37.4	35.4	31.6	41.8	38.8	37.0	35.7	34.3	32.3	28.2
5	644	43.8	41.0	39.3	38.0	36.8	34.8	31.0	42.0	39.0	37.3	36.0	34.7	32.7	28.7
6	669	43.4	40.5	39.0	37.5	36.3	34.4	30.7	42.7	39.9	38.2	36.8	35.5	33.6	29.9
7	695	43.9	41.3	39.6	38.3	36.9	34.9	31.7	42.7	40.1	38.3	37.0	35.6	33.5	30.0
8	699	45.0	42.5	40.8	39.5	38.2	36.2	32.9	44.5	41.9	40.2	38.9	37.6	35.6	32.2
9	729	39.1	36.3	34.5	33.2	31.8	29.7	26.1	40.2	37.5	35.7	34.4	33.0	31.0	24.5
10	755	43.4	40.6	39.0	37.7	36.5	34.6	30.9	39.4	36.4	34.7	33.4	32.0	30.0	26.0
11	769	42.4	39.2	37.6	36.4	35.1	33.3	29.5	41.1	37.9	36.2	35.0	33.7	31.8	27.9
12	780	43.5	40.4	38.9	37.7	36.5	34.7	30.6	39.8	36.5	34.8	33.6	32.3	30.4	26.0
13	784	42.4	39.1	37.5	36.3	35.1	33.3	29.1	40.5	37.2	35.5	34.3	33.0	31.1	26.8
14	794	44.1	41.2	39.4	38.3	37.1	35.4	31.4	41.1	38.1	36.2	35.0	33.8	31.9	27.7
15	801	44.3	41.4	39.8	38.6	37.4	35.4	31.8	41.2	38.1	36.3	35.1	33.8	31.8	27.9
16	810	43.4	40.7	38.8	37.6	36.5	34.5	30.6	41.7	38.8	36.8	35.7	34.5	32.4	28.4
17	816	43.5	40.7	38.9	37.7	36.6	34.6	30.7	41.5	38.6	36.7	35.5	34.3	32.3	28.2
18	831	44.1	41.4	39.6	38.5	37.4	35.3	31.7	41.6	38.8	36.9	35.7	34.6	32.4	28.6
19	838	43.4	40.6	38.9	37.7	36.6	34.5	30.9	41.1	38.2	36.4	35.2	34.0	31.9	28.1
20	844	45.3	42.9	40.9	39.8	38.7	36.7	33.2	39.0	36.4	34.2	32.9	31.7	29.3	25.6
21	852	44.6	42.1	40.3	39.0	37.9	35.8	32.4	43.9	41.4	39.6	38.3	37.2	35.1	31.6
22	882	44.3	41.8	40.0	38.7	37.3	35.2	31.9	42.7	40.1	38.3	37.0	35.6	33.4	30.1
23	891	42.1	39.5	37.7	36.4	35.0	32.8	29.3	42.3	39.8	37.9	36.6	35.2	33.1	29.5
24	905	43.5	41.0	39.2	37.9	36.6	34.5	31.1	41.8	39.3	37.4	36.1	34.8	32.6	29.2
25	920	43.9	41.5	39.7	38.5	37.2	35.2	31.9	41.2	38.7	36.8	35.5	34.2	32.1	28.6
26	937	43.2	40.4	38.8	37.5	36.1	34.2	30.8	41.5	38.6	36.9	35.5	34.1	32.1	28.7
27	941	43.9	41.1	39.4	38.1	36.8	34.9	31.6	43.3	40.5	38.8	37.5	36.2	34.2	30.9
28	999	48.2	46.2	44.6	43.5	42.4	40.6	37.3	39.2	36.9	35.1	33.8	32.5	30.4	26.5
29	1068	37.7	37.7	37.7	36.5	35.2	33.2	29.6	37.3	37.3	37.3	36.1	34.8	32.8	29.2
30	1088	42.6	42.6	42.6	42.6	41.5	39.7	36.3	37.7	37.7	37.7	37.7	36.4	34.5	30.8
Average		43.5	41.0	39.4	38.2	36.9	35.0	31.4	41.2	38.6	37.0	35.7	34.4	32.3	28.5

ellipsoidal sets. In Table 3.3, for all the donation centers, the number of donation is set to a constant multiplied by the deterministic value. This constant is found according to the method presented in Section 3.3 and it is always smaller than the number of donation in an instance with convex hull uncertainty set and it is always greater than the number of donation in the same instance with the box uncertainty set.

Table 3.4 represents the deterministic gaps (equation 3.12) for all the instances. Column names show the model used to find the robust solution. Although, these

Table 3.3: Comparison gaps of solutions found by RICASP-E, RIPBA-E for the six deviation scenarios of 30 instances.

SC	ND	RICASP-E							RIPBA-E						
		+10	+5	+2	±0	-2	-5	-10	+10	+5	+2	±0	-2	-5	-10
1	533	30.2	27.0	25.4	23.3	21.6	19.0	14.8	30.2	27.0	25.4	23.3	21.6	19.0	14.8
2	593	31.5	28.2	26.1	24.4	22.8	20.3	16.6	29.0	25.6	23.4	21.7	20.0	17.4	13.6
3	600	31.2	27.9	25.8	24.2	22.5	20.0	16.4	28.3	24.9	22.7	21.0	19.2	16.7	12.9
4	633	32.2	28.7	26.6	25.1	23.5	21.1	16.4	27.5	23.8	21.5	19.9	18.2	15.6	10.6
5	644	33.8	30.5	28.5	27.0	25.5	23.2	18.7	25.3	21.6	19.3	17.7	16.0	13.4	8.3
6	669	32.9	29.6	27.6	26.0	24.5	22.3	17.9	25.7	22.0	19.9	18.1	16.5	14.0	9.1
7	695	33.1	30.0	27.9	26.3	24.7	22.3	18.2	29.1	25.8	23.7	22.0	20.3	17.7	13.4
8	699	32.0	28.8	26.7	19.3	23.5	21.1	17.0	27.6	24.2	22.0	20.3	18.6	16.0	11.6
9	729	34.2	31.2	29.3	27.8	26.3	24.1	20.2	28.4	25.2	23.1	21.5	19.9	17.5	13.2
10	755	30.2	26.8	24.8	23.3	21.8	19.5	14.8	26.4	22.8	20.6	19.1	17.4	15.0	10.1
11	769	36.6	33.2	31.4	30.0	28.6	26.6	22.5	24.5	20.4	18.2	16.6	15.0	12.5	7.6
12	780	34.5	31.0	29.2	27.8	26.4	24.3	19.6	26.3	22.3	20.2	18.7	17.1	14.8	9.4
13	784	29.9	26.0	24.0	22.6	21.1	18.8	13.8	25.1	20.8	18.8	17.2	15.6	13.2	7.8
14	794	36.6	33.4	31.4	30.0	28.7	26.8	22.3	25.1	21.2	18.9	17.4	15.8	13.5	8.1
15	801	30.9	27.3	25.1	23.7	22.3	19.8	15.3	25.0	21.1	18.7	17.2	15.6	13.0	8.0
16	810	30.7	27.3	25.0	23.6	22.1	19.7	15.0	25.0	21.3	18.8	17.3	15.7	13.1	8.0
17	816	30.6	27.2	24.9	23.5	22.1	19.7	14.9	24.9	21.2	18.8	17.3	15.7	13.1	7.9
18	831	29.4	26.0	23.8	22.4	21.0	18.3	13.8	25.4	21.8	19.4	17.9	16.4	13.7	8.8
19	838	34.3	31.1	29.0	27.7	26.4	24.0	19.7	27.3	23.8	21.5	20.0	18.6	15.9	11.3
20	844	33.3	30.4	28.0	26.7	25.3	22.9	18.7	26.5	23.4	20.7	19.2	17.7	15.1	10.4
21	852	34.2	31.2	29.0	27.5	26.2	23.8	19.6	27.4	24.0	21.7	19.9	18.5	15.9	11.3
22	882	32.1	29.1	26.9	25.3	23.6	21.1	17.1	24.7	21.3	18.9	17.1	15.3	12.5	8.0
23	891	34.8	31.9	29.8	28.3	26.7	24.3	20.3	24.7	21.3	18.9	17.2	15.4	12.6	7.98
24	905	36.8	34.0	32.0	30.6	29.1	26.8	23.0	27.1	23.8	21.5	19.9	18.2	15.5	11.1
25	920	36.1	33.3	31.3	29.9	28.4	26.1	22.4	25.8	22.5	20.2	18.6	16.9	14.2	9.9
26	937	32.2	28.9	26.8	25.3	23.7	21.3	17.4	24.6	20.9	18.7	17.0	15.2	12.6	8.1
27	941	35.7	32.5	30.6	29.1	27.6	25.4	21.6	24.9	21.0	18.8	17.1	15.3	12.7	8.3
28	999	42.4	40.2	38.5	37.2	36.0	34.0	30.3	25.0	22.1	19.9	18.3	16.6	14.1	5.2
29	1068	32.7	32.7	32.7	31.5	30.0	27.9	24.0	20.7	20.7	20.7	19.2	17.5	15.0	10.4
30	1088	34.2	34.2	34.2	34.2	32.8	30.8	26.9	20.4	20.4	20.4	20.4	18.8	16.3	11.6
Average		33.3	30.2	28.2	26.5	25.2	22.9	18.7	26.1	22.7	20.5	18.9	17.2	14.7	10.0

results are not comparable with the solutions in Tables 3.1, 3.2 and 3.3, they show that when uncertainty increases, deterministic models outperform the robust models. Also, the clustered mathematical model is performing better in collecting more number of donated blood than robust mixed integer program for all the types of uncertainty. Note that the solution times of the models suggest that the integer programming based approach is finding the solutions within several minutes and can outperform the quality of the solutions found by the RICASP (-B, -C, -E).

Table 3.4: Comparison d-gaps of solutions found by RIPBA-C, RICASP-C, RIPBA-B, RICASP-B, RIPBA-E, RICASP-E for the 30 instances.

SC	RIPBA-C	RICASP-C	RIPBA-B	RICASP-B	RIPBA-E	RICASP-E
1	11.3	14.6	34.5	36.5	21.5	21.5
2	6.2	14.8	32.8	36.7	18.2	21.0
3	10.2	15.0	33.7	37.1	19.2	22.5
4	11.1	15.8	34.2	37.3	18.1	23.4
5	7.4	12.9	33.5	35.6	14.5	24.2
6	11.9	19.9	34.8	35.6	15.6	23.7
7	6.3	16.6	34.3	35.7	18.7	23.2
8	7.9	23.4	37.5	38.1	18.4	17.4
9	7.6	18.4	33.0	31.7	19.8	26.2
10	11.0	23.4	32.7	37.1	18.2	22.5
11	11.6	18.7	33.5	35.0	14.8	28.5
12	10.5	17.4	32.1	36.3	16.9	26.2
13	12.9	20.1	32.3	34.4	14.7	20.2
14	8.2	17.9	33.5	36.9	15.5	28.5
15	11.4	23.0	33.6	37.2	15.3	22.0
16	12.1	21.6	34.2	36.2	15.4	21.8
17	12.4	19.2	33.6	35.9	14.8	21.2
18	15.9	23.9	35.6	38.4	17.7	22.2
19	9.3	20.1	32.0	34.7	16.1	24.2
20	10.6	20.5	31.7	38.7	17.7	25.3
21	8.5	23.9	36.8	37.5	18.0	25.7
22	9.4	26.1	36.7	38.4	16.7	24.9
23	7.3	23.0	34.7	34.4	14.7	26.1
24	7.3	20.3	34.3	36.1	17.6	28.6
25	9.3	25.6	35.1	38.7	18.0	29.4
26	9.8	24.0	33.5	35.5	14.4	23.0
27	8.6	23.4	35.6	36.2	14.6	26.9
28	7.2	24.5	29.2	39.6	12.6	32.9
29	10.5	30.5	32.7	33.1	14.8	27.7
30	9.2	29.6	34.7	39.9	16.6	31.0
Average (%)	9.8	21.0	33.9	36.5	16.6	27.7
Time (hr)	0.29	8	0.25	8	0.21	8

Table 3.5 presents the comparison and deterministic gaps for the Chance Constrained Model presented in Section 3.4. To compare the chance constrained programming results with the deterministic situation, we are using two types of gaps. The first type, called gap, represents the percentage of number of donations that could not be collected, when we set the number of donors to $\mu_i + \psi \times \sigma_i$ at each center and we calculate the gap using equation (3.11), where UB denotes the minimum of

Table 3.5: Comparison gaps of solutions found by ICASP-CCP, IPBA-CCP with different alpha values.

α	ICASP-CCP		IPBA-CCP	
	Gap	D-Gap	Gap	D-Gap
0.01	2.9	41.1	6.2	43.1
0.02	0.8	35.6	6.8	39.5
0.05	3.5	30.2	9.4	34.5
0.08	3.9	26.3	4.4	26.7
0.1	0.9	22.0	6.5	26.4
Average	2.4	31.0	6.7	34.0
Time (hr)	8		0.28	

$\sum_{i \in I}(\mu_i + \psi \times \sigma_i)$ and best LP relaxation solution found by the ICASP-CCP within 8 hours. Note that α defines the ψ (standard score) value. Although, the values under Gap column do not follow a specific trend, the d-gap column shows that when α – the probability of violating chance constrained constraint – is higher, the deterministic gap is smaller, because the standard score is smaller and thus the model tries to collect greater number of donated blood. d-gap in Table 3.5 is calculated according to the equation (3.12), where *DSol* represents the summation of number of donors of the instance with $d_i = \mu_i$ and *Sol* is total number of donation packages collected using the chance constraint model. DSol remains the same for all the scenarios with different α values. Note that the chance constrained programming is not as conservative as robust optimization, because of the knowledge about the distribution function of donation numbers. Thus the average gap for all the α values, is smaller than the average gap of solutions in robust approaches.

3.6 Summary

In this chapter, we introduce applications of *Robust Optimization* and *Chance Constrained Programming* for the *Integrated Collection and Appointment Scheduling Problem* (ICASP), in which our goal is to maximize the number of donated blood

packages that can be collected and delivered to a processing center within a limited time for platelets extraction, while it is known that the numbers of donors showing up at donation centers are uncertain. First, three types of uncertainty sets are introduced for the robust optimization problem and the corresponding mixed integer programming formulations are developed. Then the *Integer Programming Based Approach* of Section 2.5.1 is modified to find robust feasible solutions within reasonable time. The performance of the models are tested on six deviation scenarios of 30 instances that are generated based on real data from Gulf Coast Regional Blood Center located in Houston, TX. The results show that the box set generates the most conservative solutions, while ellipsoidal set finds less conservative solutions and convex hull results in the least comparison gaps. Also, the gaps represent that the integer programming based approach outperforms the mixed integer formulation for all the types of uncertainty. Also, a chance constrained programming formulation is developed. According to the 30 instances, it is concluded that the number of donations at all the centers follow a normal distribution. Different values are used for the probability of violating the mean number of donation and the results are reported accordingly. When the probability of violating the mean of number of donation increases, more number of donors show up and thus the chance constrained model tries to collect more units of donated blood.

Chapter 4 Conclusion and Future Work

In this dissertation, applications of Operations Research techniques are studied to address two different problems in the industry and health care settings. In this chapter, we summarize the research that has been completed in this dissertation and we discuss future research directions.

4.1 Current Findings

4.1.1 Cutting Stock Problem with Setup Cost

In Chapter 1, the *Cutting Stock Problem with Setup Cost* (CSP-S) is studied. The goal is to minimize both material and setup costs while cutting raw materials of certain length to satisfy the demand for smaller sized items.. This problem covers a wide domain of packing and cutting problems including the classical *Cutting Stock Problem*, *Bin Packing Problem* and *Pattern Minimization Problem*. CSP-S is applicable when setting cutting machines are time consuming and/or costly and material cost is considerable as well. First, a mixed integer linear model is developed for CSP-S. Then, four constructive heuristic algorithms are proposed to find an initial feasible solution for the local search methods proposed and a pool of feasible patterns. Moreover, a column generation based heuristic algorithm is proposed. The performance of local search methods, pattern pool based algorithm and column generation approach are tested on the instances from the literature and the results are reported. Results show that, for over 30 instances, the column generation approach (CGA) performs best except for the high setup cost setting. In addition, local search methods and pattern pool based algorithm provide comparable results for low cost settings and better results for the highest cost setting.

4.1.2 Integrated Collection and Appointment Scheduling Problem

In Chapters 2 and 3, the *Deterministic and Stochastic Integrated Collection and Appointment Scheduling Problem* (ICASP and RICASP) are introduced. In ICASP, it is assumed that donors are willing to donate blood at any time that we schedule their appointments, thus the only requirement is to find "good" vehicle arrival times at each donation center such that the number of donated blood units collected – within processing time limit – are maximized. Then, the schedule of appointments can be found determined based on the pickup times (see Section 2.3).

First, a mixed integer linear model is developed for ICASP, then an Integer Programming Based Approach and a Construction Based Heuristic Algorithm are proposed to find solutions within a reasonable amount of time. Results show that, for all the 30 scenarios generated according to the data from Gulf Coast Regional Blood Center located in Houston, TX, the Integer Programming Based Approach produces better results. However, the construction based heuristic algorithm can find comparable results very fast.

In RICASP, we incorporate the uncertainty in the number of donors showing up to the model. A robust optimization approach is proposed for different types of uncertainty sets. In the robust approach, the final solution is robust feasible for all the seven scenarios presented. Also, a chance constrained programming model is proposed, while it is assumed that the number of donations follows normal distribution in all the donation centers.

The results show that the integer programming based heuristic model outperforms the proposed mixed integer mathematical model both for the robust optimization and chance constrained models. Also, our knowledge about the distribution function of the number of donations can increase the solution quality of ICASP and therefore

the chance constrained programming can find better solutions than robust optimization. However, it is generally difficult to find an exact distribution function and it is suggested to follow a conservative approach such as robust optimization to ensure a robust feasible solution for all the possible scenarios.

4.2 Future Research Directions

This section includes the overview of future research directions in the cutting stock with setup cost and blood collection problems.

4.2.1 Cutting Stock Problem with Setup Cost

In Chapter 1, we propose a mixed integer programming for the cutting stock problem with setup cost. We also develop several heuristic algorithms to solve this problem quickly and precisely. Numerical results indicate that the heuristic algorithms can find comparable results with the mathematical model within reasonable time, while it is computationally challenging to find the global optimal solution of CSP-S. However, one future direction can be to improve the quality of heuristic solutions and decrease the optimality gap with the optimal solution.

For example, in the two local search methods that are presented in Sections 1.6.1.2 and 1.6.1.3, we can try to enhance the quality of solutions generated by constructive heuristic algorithms in Section 1.6.1.1. In Algorithm 6, to find the production frequency of newly formed patterns, an MIP formulation is used and it is assumed that other patterns are being produced a fixed number of times. However the quality of solutions may be improved, if production frequencies of old patterns are considered as decision variables and determined using a modified MIP formulation. Also, in Algorithms 6 and 7, to combine every p patterns, the first p cutting patterns that are generated with constructive heuristic algorithm are chosen for combination; However,

a randomized or systematic approach to define best p patterns to be combined may increase the quality of solutions further. It is also a fruitful research area to consider two or more dimensions of the cutting stock problem with setup cost and use the one-dimensional problem introduced in Chapter 1 as a research base.

4.2.2 Integrated Collection and Appointment Scheduling Problem

In Chapter 2, we propose a mixed integer formulation for the deterministic blood collection problem. Also, we develop two heuristic algorithms in order to find solutions for the ICASP within reasonable time. Numerical experiments show that it is computationally challenging to solve the mathematical model. Also, the heuristic algorithms can outperform it within a limited time. However, it is not guaranteed that the heuristic algorithms can find the optimal solution of a ICASP instance. Therefore, one future suggestion is to decrease the solution time of the mathematical model. ICASP is a more general version of the vehicle routing problem with time windows, where every center can be visited more than once. Hence, it might be possible to utilize the exact solution approaches that are developed for the vehicle routing problem while we assume that any center can be visited at most a constant time (e.g. c) and then we can duplicate any center into c similar centers, where each duplicated center can be visited only once per day. Also, it is possible to apply simulated annealing integrated by the proposed neighborhood search method and improve the quality of solutions.

In Chapter 3, we propose several robust mathematical models considering different types of uncertainty sets and we also develop a chance constrained program for the integrated blood collection and appointment scheduling problem for the time that the number of daily donations is uncertain. However, one may prefer to include stochasticity of traveling times, service time or donation unit time. Also, it might be

interesting to consider that the availability of drivers and vehicles may change during the day. The mathematical model can be further improved with including cost of transportation or service with a lower priority than the main objective introduced in this dissertation.

References

[1] Alves, C., 2005, Cutting and packing: Problems, models and exact algorithms, Ph.D. thesis, Universidade do Minho.

[2] Alves, C., and Valério de Carvalho, J.M., 2008, "A branch-and-price-and-cut algorithm for the pattern minimization problem," *RAIRO - Operations Research* **42** 435–453.

[3] American Red Cross, 2012, http://www.redcrossblood.org.

[4] American Red Cross Blood Services, 2012, http://volunteer.united-e-way.org/uwbec/org/567730.html.

[5] Beliën, J., Forcé, H., 2012, "Supply chain management of blood products: A literature review," *European Journal of Operational Research*, **217**(1), 1–16.

[6] Belov, G., Scheithauer, G., 2002, "A cutting plane algorithm for the one-dimensional cutting stock problem with multiple stock lengths," *European Journal of Operational Research* **141**(2) 274–294.

[7] Belov, G., 2004, Problems, models and algorithms in one-and two-dimensional cutting, Ph.D. thesis, Dresden.

[8] Belov, G., Scheithauer, G., 2007, "Setup and open-stacks minimization in one-dimensional stock cutting," *INFORMS Journal on Computing* **19**(1) 27–35.

[9] Ben-Tal, A., and Nemirovski, A., 1998, "Robust convex optimization," *Mathematics of Operations Research*, **23**(4), 769–805.

[10] Bertsimas, D., and Sim, M., 2003, "Robust discrete optimization and network flows," *Mathematical Programming*, **98**(1), 49–71.

[11] Boyd, S., and Vanderberghe, L., 2004, Convex optimization, *Cambridge University Press.*

[12] Cerqueira, G., Yanasse, H., 2009, "A pattern reduction procedure in a one-dimensional cutting stock problem by grouping items according to their demands," *Journal of Computational Interdisciplinary Sciences* **1**(2) 159-164.

[13] Charnes, A., Cooper, W.W., and Symonds, G.H., 1958, "Cost horizons and certainty equivalents: An approach to stochastic programming of heating oil," *Management Science* **4**, 235–263.

[14] Chien, C., Deng. J., 2001, "Optimization of wafer exposure patterns using a two-dimensional cutting algorithm," *International Transactions in Operational Research* **8**(5) 535–545.

[15] Cordeau, J.F., Gendreau, M., Laporte, G., Potvin. J.Y., Semet, F., 2002, "A guide to vehicle routing heuristics," *Journal of the Operational Research Society* **53**(5) 512–522.

[16] Cui, Y., Zhao, X., Yang, Y., Yu, P., 2008, "A heuristic for the one-dimensional cutting stock problem with pattern reduction," *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* **222**(6) 677–685.

[17] Cui, Y., Liu, Z., 2011, "C-sets-based sequential heuristic procedure for the one-dimensional cutting stock problem with pattern reduction," *Optimization Methods and Software* **26**(1) 155-167.

[18] Dantzig, G.B., Ramser, J.H., 1959, "The truck dispatching problem," *Management Science* **6**(1) 80–91.

[19] Diegel, A., Montoccohio, E., Walters, E.,Schalkwyk, S., Naidoo, S., 1996, "Setup minimising conditions in the trim loss problem," *European Journal of Operational Research* **95** 631-640.

[20] Doerner, K.F., Gronalt, M., Hartl, R.F, Kiechle, G., Reimann, M., 2008, "Exact and heuristic algorithms for the vehicle routing problem with multiple interdependent time windows," *Computers and Operations Research* **35**(9) 3034–3048.

[21] Dyckhoff, H., Kruse, D., 1985, "Trim loss and related problems," *Omega* **13**(1) 59–72.

[22] Dyckhoff, H., 1990, "A typology of cutting and packing problems," *European Journal of Operational Research* **44** 145–159.

[23] Farley, A., Richardson, K., 1984, "Fixed charge problems with identical fixed charges," *European Journal of Operational Research* **18**(2) 245–249.

[24] Foerster, H., Waescher, G., 2000, "Pattern reduction in one-dimensional cutting stock problems," *International Journal of Production Research* **38**(7) 1657–1676.

[25] Ford, L.R., Fulkerson, D.R., 1962, Flows in Networks, *Princeton University Press*, Princeton, New Jersey.

[26] Ghandforoush, P., Sen, T.K., 2010, "A DSS to manage platelet production supply chain for regional blood centers," *Decision Support Systems* **50**(1) 32–42.

[27] Gilmore, P., Gomory, R., 1961, "A linear programming approach to the cutting-stock problem," *Operations Research* **9**(6) 849–859.

[28] Gilmore, P., Gomory, R., 1963, "A linear programming approach to the cutting stock problem-part II," *Operations Research* **11**(6) 863–888.

[29] Gilmore, P., Gomory. R., 1965, "Multistage cutting stock problems of two and more dimensions," *Operations Research* **13**(1) 94–120.

[30] Glover, F., 1975, "Improved linear integer programming formulations of nonlinear integer problems," *Management Science* **22**(4) 455–460.

[31] Golden, B., Raghavan, S., Wasil, E., 2008, The vehicle routing problem: Latest advances and new challenges. *Operations Research/Computer Science Interfaces Series*, Springer.

[32] Golden, B., Levy, L., Vohra, R., 1987, "The orienteering problem," *Naval Research Logistics* **34**(3) 307–318.

[33] Gulf Coast Regional Blood Center, 2012, http://www.giveblood.org.

[34] Haessler, R.W., 1975, "Controlling cutting pattern changes in one-dimensional trim problems," *Operations Research* **23**(3) 483–493.

[35] Johnson, D.S., 1973, Near optimal bin packing algorithms, Ph.D. thesis, Massachusetts Institute of Technology.

[36] Kantorovich, L., 1960, "Mathematical methods of organizing and planning production," *Management Science* **6**(4) 366–422.

[37] MacQueen, J., 1967, "Some methods for classification and analysis of multivariate observations," *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability* **1** 281–297.

[38] Martello, S., Toth, P., 1990, Knapsack problems: Algorithms and computer implementations. *Wiley*, New York.

[39] McDiarmid, C., 1999, "Pattern minimisation in cutting stock problems," *Discrete Applied Mathematics* **98**(1-2) 121–130.

[40] Michaels, J.D., Brennan, J.E., Golden, B.L, Fu. M.C., 1993, "A simulation study of donor scheduling systems for the American Red Cross," *Computers and Operations Research* **20**(2) 199–213.

[41] Prastacos, G.P., 1984, "Blood inventory management: an overview of theory and practice," *Management Science* **30**(7) 777–780.

[42] Sankaran, J.K., Ubgade, R.R., 1994, "Routing tankers for dairy milk pickup," *Interfaces* **24**(5) 59–66.

[43] Simchi-Levi, D., 1994, "New worst-case results for the bin-packing problem," *Naval Research Logistics* **41** 579-585.

[44] Soyster, A.L., 1973, "Convex programming with set-inclusive constraints and applications to inexact linear programming," *Operations Research*, 1154–1157.

[45] Stadtler, H., 1988, "A comparison of two optimization procedures for 1- and 1 1/2-dimensional cutting stock problems," *OR Spectrum* **10**(2) 97–111.

[46] Sungur, I., Ordóñez, F., Dessouky, M., 2008, "A robust optimization approach for the capacitated vehicle routing problem with demand uncertainty," *IIE Transactions*, **40**(5), 509–523.

[47] Toth, P., Vigo, D., 2002, "Models, relaxations and exact approaches for the capacitated vehicle routing problem," *Discrete Applied Mathematics* **123**(1) 487–512.

[48] Turkish Ministry of Health, 2011, http://www.saglik.gov.tr.

[49] Umetani, S., Yagiura, M., Ibaraki, T., 2003, "One-dimensional cutting stock problem to minimize the number of different patterns," *European Journal of Operational Research* **146**(2) 388–402.

[50] U.S. Department of Energy, 2011, http://www1.eere.energy.gov/vehiclesandfuels.

[51] Valério de Carvalho, J.M., 1998, "Exact solution of cutting stock problems using column generation and branch-and-bound," *International Transactions in Operational Research* **5**(1) 35–44.

[52] Valério de Carvalho, J.M., 1999, "Exact solution of bin-packing problems using column generation and branch-and-bound," *Annals of Operation Research* **86** 629–659.

[53] Valério de Carvalho, J.M., 2002, "LP models for bin packing and cutting stock problems," *European Journal of Operational Research* **141** 253–273.

[54] Vance, P., 1999, "Branch-and-price algorithms for the one dimensional cutting stock problem," *Computational Optimization and Applications* **9** 211–228.

[55] Vanderbeck, F., 2000, "Exact algorithm for minimising the number of setups in the one-dimensional cutting stock problem," *Operations Research* **48**(6) 915–926.

[56] Walker, W., 1976, "A heuristic adjacent extreme point algorithm for the fixed charge problem," *Management Science* **22**(5) 587–596.

[57] Waescher, G., Gau, T., 1996, "Heuristics for the integer one-dimensional cutting stock problem: A computational study," *OR Spectrum* **18**(3) 131–144.

[58] Waescher, G., Haubner, H., Schumann, H., 2007, "An improved typology of cutting and packing problems," *European Journal of Operational Research* **183** 1109-1130.

[59] World Health Organization, 2012, http://www.who.int.

[60] Yanasse, H., Limeira, M., 2006, "A hybrid heuristic to reduce the number of different patterns in cutting stock problems," *Computers and Operations Research* **33**(9) 2744–2756.

[61] Yi, J., Scheller-Wolf, A., 2003, "Vehicle routing with time windows and time-dependent rewards: a problem from the American Red Cross," *Manufacturing and Service Operations Management* **5**(1) 74–77.