This is the peer reviewd version of the followng article:

Mathematical models and decomposition algorithms for makespan minimization in plastic rolls production / Nesello, Vitor; Delorme, Maxence; Iori, Manuel; Subramanian, Anand. - In: JOURNAL OF THE OPERATIONAL RESEARCH SOCIETY. - ISSN 0160-5682. - 69:3(2018), pp. 326-339. [10.1057/s41274-017-0221-8]

Terms of use:

The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

28/04/2024 00:36

(Article begins on next page)

Mathematical Models and Decomposition Algorithms for Makespan Minimization in Plastic Rolls Production

Vitor Nesello, Maxence Delorme, Manuel Iori, Anand Subramanian

May 22, 2017

Abstract

We study an optimization problem that originates from the packaging industry, and in particular in the process of blown film extrusion, where a plastic film is used to produce rolls of different dimensions and colors. The film can be cut along its width, thus producing multiple rolls in parallel, and set-up times must be considered when changing from one color to another.

The optimization problem that we face is to produce a given set of rolls on a number of identical parallel machines by minimizing the makespan. The problem combines together cutting and scheduling decisions, and is of high complexity. For its solution we propose mathematical models and heuristic algorithms that involve a non-trivial decomposition method. By means of extensive computational experiments we show that proven optimality can be achieved only on small instances, whereas for larger instances good quality solutions can be obtained especially by the use of an iterated local search algorithm.

Keywords: Plastic Rolls Production, Blown Film Extrusion, Optimization, Mixed Integer Linear Programming, Iterated Local Search.

Introduction

In the packaging industry, plastic films are commonly obtained by blown film extrusion. A tube of polymer is inflated to form a thin film bubble. The bubble is collapsed to obtain a flat film, which is then used to produce the required items. A common process is to first use the film to produce plastic rolls, and then use the rolls later on the production process to obtain smaller items. The rolls have a certain width that depends on the extruder machine at use, and a (possibly very large) length that depends from the adopted operating time. To obtain the desired roll dimensions, the film can be cut both along its width and along its length. The cutting along the width is obtained by one or more devices called *slits* and allows

to produce multiple rolls in parallel on the same machine. We refer to, e.g., Cantor (2011) for a wide description of the blown film extrusion characteristics.

In this paper we focus on a problem that originates from an industry producing plastic bags for the South American market. Multiple identical extruded machines, each equipped with a limited number of slits, are available to produce in parallel a set of required plastic rolls. Each roll has a specific color, thickness, width, and length. The extruder machines make use of raw polymer, thus a colorant is required for the plastic bags to have the desired color. This implies that the rolls processed at the same time on the same machine have the same color and thickness, although they might have different width and length. The width of the film can be adjusted at any time so as to waste neither time nor plastic. A setup time is incurred when switching from one color to another, while changing the thickness is instantaneous. The aim is to produce the required set of rolls by minimizing the *makespan*, i.e., the completion time of the last item.

Note that the cutting process can be seen as a *three-stage two-dimensional guillotine cutting* (see, e.g., Silva et al. 2010): the first-stage of vertical cuts separates *blocks* of rolls having different colors; the second-stage of horizontal cuts separates *shelves* of rolls produced in parallel; the third-stage of vertical cuts separates a roll from the subsequent one thus obtaining the required products.

To better understand this complex problem, a simple example involving 12 items of 3 different colors and a unique thickness is depicted in Figure 1. The numbers on the arcs represent the setup times between pairs of colors. The widths of the items are proportional to the vertical dimensions of the depicted rectangles, whereas lengths are proportional to the horizontal dimensions (roll lengths have been scaled down to better fit the graphical representation). An optimal solution with two machines, both equipped with one slit and having maximum width 5, is provided in Figure 2. Machine 1 processes a first block containing only item 1 for 5 time units, then incurs in a setup of 4 units, and finally processes a second block containing items from 5 to 8 for 11 time units. The slit is used only in the second block to separate the shelf containing item 8 from the shelf containing the remaining items. Note that no plastic is wasted during the processing of the second block. The width of the film can indeed be reduced by lowering down by one unit the upper part when processing items 8 and 6, and by two units when processing 8 and 7. Similarly, the bottom can be lift up by one unit when processing only 7. A similar process is performed by machine 2, that processes two blocks, both made by two shelves. The

slit is used in the first block to separate item 4 from 2 and 3, and in the second block to separate items 9 and 10 from 11 and 12. The resulting makespan is of 20 time units.

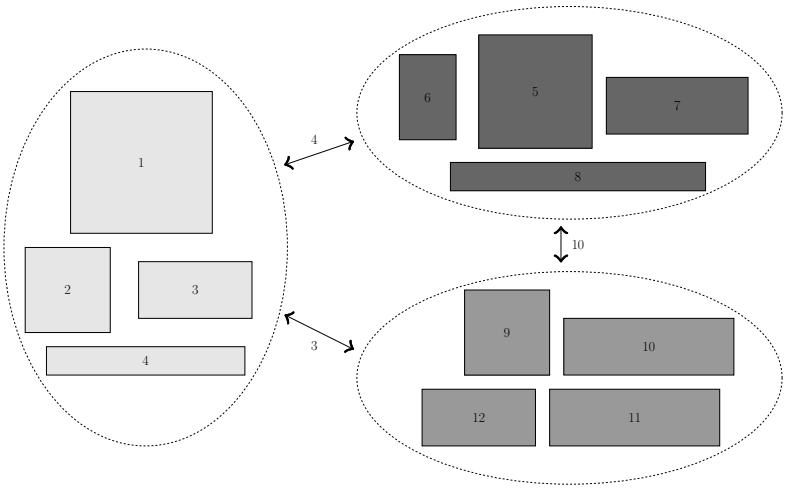

Figure 1: A simple PRPP instance.

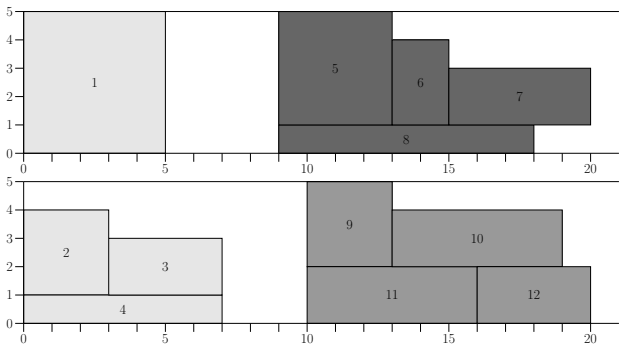

Figure 2: An optimal solution for the instance in Figure 1.

The resulting optimization problem, that we hereafter call *plastic rolls production problem* (PRPP), embeds a cutting component (the 3-stage two-dimensional guillotine cutting) into a classical scheduling problem of makespan minimization with setup times on parallel machines. Problems arising in the so-called *cutting and packing* research area are frequently encountered in practical industrial applications (see, e.g., Kallrath et al. 2014 and the special issue edited by Bennell et al. 2013). Cutting and packing problems are frequently combined with problems from other optimization areas so as to better model real-world situations. Iori et al. (2007) combined two-dimensional loading problem and vehicle-routing, Gramani et al. (2009)

and Silva et al. (2014) integrated lot-sizing with cutting stock problems, and Hu et al. (2015) considered two-dimensional spatial resource constraints in project scheduling, just to cite some examples.

To the best of our knowledge, the problem that is most similar to the PRPP is the three-stage two-dimensional guillotine cutting problem studied by Vanderbeck (2001). In this problem, rectangular bins of fixed width and length are cut into blocks, then blocks are cut into shelves, and shelves are cut into items. Fixed setup times occur between blocks, and the objective is mainly waste minimization. The problem was solved by a decomposition based on the recursive use of a column generation approach. The PRPP is however different from the problem in Vanderbeck (2001) because it has a limit on the number of shelves per block, it has sequence-dependent setup times, it involves rolls instead of bins of fixed length, and aims at minimizing makespan instead of waste material.

The aim of this paper is to formally introduce the PRPP, present exact and heuristic methods for its optimization, and perform an extensive computational evaluation of the proposed solution strategies. In details, we first model the PRPP using a compact *mixed integer linear programming* (MILP) formulation, i.e., a formulation that has a polynomial number of variables and constraints. We then decompose the problem by separating the cutting component from the scheduling one, and use this to derive valid lower and upper bounds on the optimal makespan. Notably, these bounds are obtained by solving three different MILP models, one involving a non-trivial pseudo-polynomial *arc-flow* formulation, another one requiring a tailored branch-and-cut implementation and the last one based on the generalized assignment problem. These techniques work well for small- and medium-size instances, thus, to efficiently address large-size instances, we developed a metaheuristic based on an *iterated local search* framework.

The remainder of the paper is organized as follows: (i) the PRPP is formally introduced and the relevant literature is discussed; (ii) the compact MILP formulation is presented; (iii) the decomposition approach and the lower bounds are explained; (iv) the upper bounding methods are described; (v) the proposed techniques are computationally evaluated; and (vi) conclusions are drawn in the last section.

Problem Description

The PRPP requires to produce n plastic rolls, called *items* for short in the following, on m identical parallel machines. Each item j has width w_j, length l_j, and belongs to a class γ_j, for $j = 1, \ldots, n$. Without loss of generality we suppose that the processing time of an item j is equal to its length l_j. Two items belong to the same class if they have the same color and thickness, thus allowing both of them to be processed at the same machine simultaneously. A setup cost s_{ij} occurs when a machine processes an item j after having processed an item i. A machine cannot process items during setup. Each machine has a maximum width W and is equipped with σ slits. A *block* is a subset of items of the same class, whereas a *shelf* is a sequence of items packed at the same width in a block. A machine can process items in parallel by using a three-stage two-dimensional guillotine cutting process: the first-stage vertical cuts separate blocks, the second-stage horizontal cuts separate shelves, and the third-stage vertical cuts separate items. The width of a shelf is given by the wider item in that shelf. The total width of the shelves produced in parallel on a machine cannot exceed W, whereas their total number cannot exceed $\sigma + 1$. The processing time of a shelf is given by the sum of the lengths of the items in that shelf, and the processing time of a block is the maximum processing time of the shelves in that block. The objective of the PRPP is to feasibly schedule the items into the machines by minimizing the makespan.

We consider that a fixed identical setup cost occurs on each machine at the beginning of the activities. As this cost is the same for all the machines it has no impact on the optimal solution, thus we simply disregard it from our study. Moreover, our models and algorithms could be easily adapted to include a starting setup cost that depends from an initial configuration on each machine.

The PRPP is composed by a two-dimensional cutting component (creating the blocks) and a scheduling component (assigning the blocks to the machines with sequence-dependent setup times). Because of its nature, it generalizes a number of quite different combinatorial optimization problems, as noticed in Table 1. If a single machine with 0 slits is available and all items have the same color, then the problem is trivial because there are no setup costs and items must be processed one after the other, so the makespan is simply the sum of all processing times. If a single machine with 0 slits is available and the items have different colors, then the PRPP reduces to the well-known *traveling salesman problem* (TSP), see,

e.g., Applegate et al. (2007), because items cannot be produced in parallel and thus minimizing the makespan is equivalent to finding the permutation of the items that leads to the lowest setup cost. If all items have the same color, and multiple machines are available but none of them has slits, then the PRPP is equivalent to the *identical parallel machine scheduling problem* ($P||C_{max}$), see, e.g., Dell'Amico et al. (2008). In this case, indeed, no setup cost is paid, items cannot be produced in parallel on a machine, and thus the problem is to distribute the items among the machines to obtain the best makespan. If all items have the same color and just one machine is available, then the PRPP becomes a *three-stage cardinality-constrained two-dimensional strip packing problem* (3SC2SPP), where the cardinality limit is imposed on the number of shelves. The strip packing problem (SPP) is to pack a set of rectangles into a strip of fixed width and minimum length, without overlapping and without rotation (see, e.g., Côté et al. 2014). Several SPP variants including multi-stage guillotine cuts have been studied in the literature (see, e.g., Silva et al. 2010 and Mrad 2015 among others), but, to the best of our knowledge, the 3SC2SPP is a new problem.

With the exception of the trivial case, all the discussed problem variants are NP-hard, so the same holds for the PRPP.

Table 1: Relevant PRPP variants under different input parameters.

colors	machines	slits	problem
1	1	0	trivial
2 or more	1	0	traveling salesman problem
1	2 or more	0	identical parallel machines scheduling problem
1	1	1 or more	three-stage two-dim. card.-constr. SPP
2 or more	2 or more	1 or more	plastic rolls production problem (this paper)

A Compact Mathematical Formulation

In this section we propose a compact model for the PRPP that generalizes the one presented by Lodi and Monaci (2003) for the two-stage two-dimensional knapsack problem by considering three-stage cutting and setup costs. In the following, let items be sorted by non-increasing width, breaking ties by non-increasing length.

Let us say that an item *initializes* a shelf if it is the lowest-index item in that shelf, and that it initializes a block if it is the lowest-index item in that block.

Now, let us first model the cutting part of the PRPP by using the decision variable

- $x_{jik} = \begin{cases} 1 & \text{if item } j \text{ is in shelf } i \text{ of block } k; \\ 0 & \text{otherwise;} \end{cases}$ for $1 \leq k \leq i \leq j \leq n$, $\gamma_k = \gamma_i = \gamma_j$.

Variable x has a threefold meaning: if $x_{kkk} = 1$, then item k initializes block k, meaning that item k has the greatest width of all items in block k; if $x_{iik} = 1$, $k < i$, then item i initializes shelf i in the block initialized by a previous item k, meaning that item i has the greatest width of all items in shelf i, but there is at least one item with greater width that initializes another shelf in block k; if $x_{jik} = 1$, $k < i < j$, then item j is packed into the shelf initialized by a previous item i in the block initialized by a previous item k, meaning that item j is placed in a shelf containing at least one item of greater width. This is necessary to enforce items with lower width to be packed to the right of the larger ones, therefore meeting the three-stage two-dimensional guillotine cutting configuration of the problem. For example, in Figure 2, the block containing items 2, 3, and 4 is the block number 2, because the item 2 is the one with greatest width. Hence, we say that item 2 initializes the block and shelf of the same number, and we have $x_{222} = 1$. Item 3 is packed on the shelf initialized by item 2, so $x_{322} = 1$, and item 4 initializes the shelf number 4, so $x_{442} = 1$. The other variables that take value 1 are x_{111}, x_{555}, x_{655}, x_{755}, x_{885}, x_{999}, $x_{10,9,9}$, $x_{11,11,9}$, and $x_{12,11,9}$.

Let us call for short "block k" the block initialized by item k and "shelf i" the shelf initialized by item i. To model the scheduling part of the PRPP let us introduce two dummy items, 0 and $n+1$, having 0 length and no setup costs from and to other items. Item 0 represents the beginning of the activities and item $n+1$ represents the end. Let us also denote $N = \{1, 2, \ldots, n\}$ and $N' = \{0, 1, \ldots, n, n+1\}$. We make use of the following decision variables:

- $\xi_{hk} = \begin{cases} 1 & \text{if block } h \text{ is followed by block } k; \\ 0 & \text{otherwise;} \end{cases}$ for $h, k \in N'$, $h \neq n+1$, $k \neq 0$, $h \neq k$;

- α_k : setup required for block k, for $k \in N$;

- β_k : processing time of block k, for $k \in N$;

- ϑ_{hk} : time flow between blocks h and k, for $h, k \in N'$, $h \neq n+1$, $k \neq 0$, $h \neq k$;
- z : makespan.

In practice, the ξ_{hk} variables keep track of the sequences of blocks in each machine, whereas the other variables are used to compute the completion times of the jobs and the makespan. For the example depicted in Figure 2, we have: $\beta_1 = 5$, $\beta_2 = 7$, $\beta_5 = 11$, $\beta_9 = 10$, $\xi_{01} = 1$, $\xi_{15} = 1$, $\xi_{5,13} = 1$, $\xi_{02} = 1$, $\xi_{29} = 1$, $\xi_{9,13} = 1$, $\alpha_5 = 4$, $\alpha_9 = 3$, $\vartheta_{15} = 5$, $\vartheta_{5,13} = 20$, $\vartheta_{29} = 7$, $\vartheta_{9,13} = 20$, and $z = 20$.

The PRPP can then be formulated as the following MILP model:

$$\min \quad z \tag{1}$$

subject to

$$\sum_{i=1}^{j}\sum_{k=1}^{i} x_{jik} = 1 \qquad j \in N \tag{2}$$

$$\sum_{i=k}^{n} w_i x_{iik} \leq W x_{kkk} \qquad k \in N \tag{3}$$

$$\sum_{i=k}^{n} x_{iik} \leq \sigma + 1 \qquad k \in N \tag{4}$$

$$x_{jik} \leq x_{iik} \qquad j, i, k \in N, k \leq i \leq j \tag{5}$$

$$\sum_{h \in N' \setminus \{k, n+1\}} \xi_{hk} = x_{kkk} \qquad k \in N \tag{6}$$

$$\alpha_k \geq \sum_{h \in N' \setminus \{k, n+1\}} s_{hk} \xi_{hk} \qquad k \in N \tag{7}$$

$$\beta_k \geq \sum_{j=i}^{n} l_j x_{jik} \qquad i, k \in N, k \leq i \tag{8}$$

$$\sum_{h \in N' \setminus \{0,k\}} \xi_{kh} - \sum_{h \in N' \setminus \{k, n+1\}} \xi_{hk} = \begin{cases} m & \text{if } k = 0; \\ -m & \text{if } k = n+1; \\ 0 & \text{otherwise} \end{cases} \qquad k \in N' \tag{9}$$

$$\sum_{h \in N' \setminus \{0,k\}} \vartheta_{kh} - \sum_{h \in N' \setminus \{k, n+1\}} \vartheta_{hk} = \begin{cases} 0 & \text{if } k = 0; \\ \alpha_k + \beta_k & \text{otherwise} \end{cases} \qquad k \in N' \setminus \{n+1\} \tag{10}$$

$$z \geq \vartheta_{h,n+1} \qquad h \in N' \setminus \{n+1\} \tag{11}$$

$$\alpha_k, \beta_k \geq 0 \qquad k \in N \tag{12}$$

$$0 \leq \vartheta_{hk} \leq U \xi_{hk} \qquad h, k \in N', h \neq n+1,$$

$$k \neq 0, h \neq k \tag{13}$$

$$\xi_{hk} \in \{0,1\} \qquad h,k \in N', h \neq n+1, \quad k \neq 0, h \neq k \tag{14}$$

$$x_{jik} \in \{0,1\} \qquad j,i,k \in N, k \leq i \leq j, \quad \gamma_k = \gamma_i = \gamma_j \tag{15}$$

The objective function (1) minimizes the makespan. Constraints (2) ensure that every item is scheduled. Constraints (3) state that the maximum width of each block is not exceeded. Constraints (4) impose that the number of shelves in a block is not greater than the maximum number of slits plus one. Constraints (5) state that an item j can assigned to a shelf i only if the shelf has been initialized by i. Constraints (6) state that if a block k is used, then exactly one ξ_{hk} variable incoming into k should be used. Constraints (7) and (8) compute, respectively, the setup time and the processing time required for each block k, if any. Note that the processing time of the block is given by the maximum of the sums of the processing times on each shelf. Constraints (9) ensure that exactly m sequences are created and that these sequences are connected. Constraints (10) use the variables ϑ_{hk} as a commodity to compute the increasing time along the sequences. This has the effect of disregarding subtours, and also allows to compute, through constraint (11), the value of the makespan. Finally, constraints (12)-(15) impose the bounds on the variables, with U being a valid upper bound value on the makespan.

Lower Bounds based on a Decomposition Method

Model (1)-(15) has the merit of unambiguously describe the PRPP, but, as later shown, it has a poor computational performance. In this section and in the following one we study a natural decomposition of the PRPP into its two main components, namely, cutting and scheduling, and show how this can lead to the computation of lower and upper bounds on the optimal solution value. Figure 3 presents a flowchart of our decomposition. The *cutting component* (CC) solves a pure cutting problem, producing a set of possible cutting patterns and a set of selected blocks. The patterns are passed to a *scheduling component* (SC), that computes a color sequence for a pure scheduling problem and obtains a valid lower bound. Two algorithms, namely, an iterated local search and an approach based on a generalized assignment problem,

use the selected blocks (the latter also uses the color sequence) to produce a feasible solution of good quality.

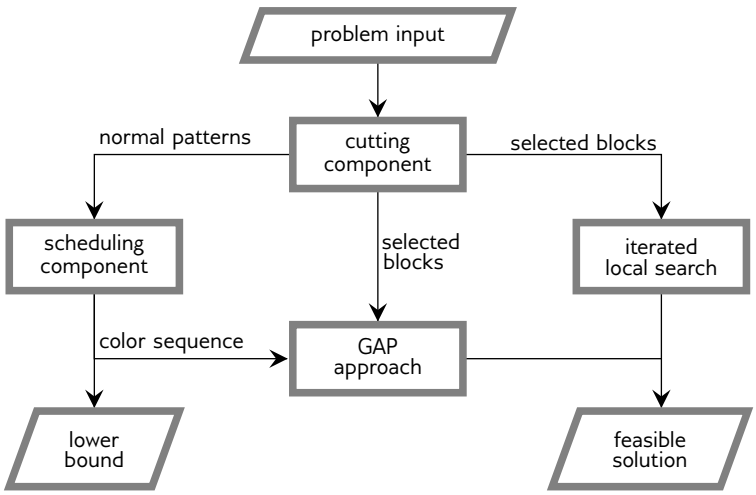

Figure 3: A flowchart of the complete algorithm.

Cutting component (CC)

Let S be the set of all item classes, and $N_s = \{j \in N : \gamma_j = s\}$ be the subset of items whose class is s, for $s \in S$. As described in the problem description Section, the problem of producing all items belonging to the same class from a film of minimum length is the 3SC2SPP. In this section we propose a method to solve the 3SC2SPP that is based on the arc-flow formulation introduced by Valério de Carvalho (1999) for the one-dimensional cutting stock problem, and later extended to the two-stage two-dimensional case by Macedo et al. (2010).

We need some additional notation to describe our formulation. First of all, let us define, for each $s \in S$, m_s^* as the number of different item widths and $\mathcal{W}_s^* = \{w_1^*, w_2^*, \ldots, w_{m_s^*}^*\}$ the corresponding width set. Additionally, we compute the set $\mathcal{L}_s$ of all possible combinations of item lengths as

$$\mathcal{L}_s = \left\{ x = \sum\nolimits_{j \in T} l_j : 0 \leq x \leq \Lambda_s, T \subseteq N_s \right\}, \tag{16}$$

with Λ_s being an upper bound on the maximum length of a block of items of class s. Using the definitions by Herz (1972) and Christofides and Whitlock (1977), $\mathcal{L}_s$ can be either called the set of canonical dissections or of normal patterns. Here we

use the term normal patterns. The computation of $\mathcal{L}_s$ may be obtained by invoking a standard dynamic programming procedure. We use $\mathcal{L}_s$ to determine the possible positions of the first-stage cuts.

To model the second-stage cuts, we make use of a graph $G'_s = (V'_s, A'_s)$. The vertex set is $V'_s = \{(a,b)\colon a = 0,1,\ldots,W; b = 0,1,\ldots,\sigma+1\}$. The arc set A'_s is composed by so-called *item arcs* and *loss arcs*: items arcs represent a second-stage cut corresponding to an item and their set is $\{((d,u),(e,u+1))\colon 0 \le u \le \sigma; 0 \le d < e \le W$ and $e - d \in \mathcal{W}^*_s\}$; loss arcs represent unused portions of the film and their set is $\{((a,b),(W,\sigma+1))\colon (a,b) \in V'_s\}$. In addition, let $A'_{j^*s} = \{a = ((d,u),(e,v)) \in A'_s : e - d = w^*_{j^*}\}$, for $w^*_{j^*} \in \mathcal{W}^*_s$. Let also $\delta^+(e,u)$, respectively $\delta^-(e,u)$, be the subset of arcs of A'_s that leaves, respectively enters, a node (e,u).

Third-stage cuts induced on a shelf by a width $w^*_{j^*} \in \mathcal{W}^*_s$, for $j^* = 1,\ldots,m^*_s$, are modeled by the use of a multi-graph $G''_{j^*s} = (V''_s, A''_{j^*s})$, where the vertex set is $V''_s = \{0,1,\ldots,\Lambda_s\}$ and the arc set is composed by a subset of item arcs and two subsets of loss arcs as $A''_{j^*s} = \{(k,d,e)\colon 0 \le d < e \le \Lambda_s$ and $\exists\, k \in N_s\colon e - d = l_k$ and $w_k \le w^*_{j^*}\} \cup \{(0,d,\Lambda_s) : d = 0,1,\ldots,\Lambda_s - 1\} \cup \{(0,d,d+1) : d = 0,1,\ldots,\Lambda_s - 1\}$. Moreover, for any $j^* = 1,\ldots,m^*_s$ we define $\delta^+_{j^*}(e)$, respectively $\delta^-_{j^*}(e)$, as the subset of arcs of A''_{j^*s} that leaves, respectively enters, a node e.

Let us introduce the following decision variables:

- φ_p = number of times a block of length $p \in \mathcal{L}_s$ is chosen (first-stage vertical cuts);
- φ'_{pa} = number of times arc $a \in A'_s$ is chosen as a second-stage horizontal cut on a block of length $p \in \mathcal{L}_s$;
- φ''_{j^*a} = number of times arc $a \in A''_{j^*s}$ is used as a third-stage vertical cut on a shelf of width $w^*_{j^*} \in \mathcal{W}^*_s$.

The film of minimum length required to produce all items of class $s \in S$ can be obtained by solving the following MILP model:

$$\min \quad z^{CC}_s = \sum_{p \in \mathcal{L}_s} p\varphi_p \tag{17}$$

subject to

$$\sum_{a\in\delta^+(e,u)} \varphi'_{pa} - \sum_{a\in\delta^-(e,u)} \varphi'_{pa} = \begin{cases} \varphi_p & \text{if } (e,u)=(0,0); \\ -\varphi_p & \text{if } (e,u)=(W,\sigma+1); \\ 0 & \text{otherwise,} \end{cases} \qquad (e,u)\in V'_s, p\in\mathcal{L}_s \tag{18}$$

$$\sum_{a\in\delta^+_{j^*}(e)} \varphi''_{j^*a} - \sum_{a\in\delta^-_{j^*}(e)} \varphi''_{j^*a} = \begin{cases} \sum_{p\in\mathcal{L}} \sum_{a\in A'_{j^*s}} \varphi'_{pa} & \text{if } e=0; \\ -\sum_{p\in\mathcal{L}} \sum_{a\in A'_{j^*s}} \varphi'_{pa} & \text{if } e=\Lambda_s; \\ 0 & \text{otherwise,} \end{cases} \qquad j^*=1,\dots,m^*_s \tag{19}$$

$$\sum_{j^*=1,\dots,m^*_s, w^*_{j^*}\geq w_k} \sum_{a\in A''_{j^*s}} \varphi''_{j^*a} = 1 \qquad k\in N_s \tag{20}$$

$$\varphi''_{j^*(0p\Lambda_s)} = \sum_{a\in A'_{j^*s}} \varphi'_{pa} \qquad j^*=1,\dots,m^*_s, \quad p\in\mathcal{L}_s \tag{21}$$

$$\varphi_p \geq 0, \text{integer} \qquad p\in\mathcal{L}_s \tag{22}$$

$$\varphi'_{pa} \geq 0, \text{integer} \qquad p\in\mathcal{L}_s, a\in A'_s \tag{23}$$

$$\varphi''_{j^*a} \geq 0, \text{integer} \qquad j^*=1,\dots,m^*_s, \quad a\in A''_{j^*s} \tag{24}$$

The objective function (17) minimizes the total used length. Constraints (18) impose flow conservation among the φ' variables, and also link together φ' with φ by stating that shelves can be created only in those blocks p that have a positive φ_p value. Similarly, constraints (19) ensure flow conservation among the third-stage cuts and allow to produce items only in shelves that have been created by second-stage cuts. Constraints (20) ensure that all items are produced, while constraints (21) force an empty space from p to Λ_s for shelves produced in block p.

For example, to model the solution depicted in the left-most block of the bottom machine in Figure 2, supposing $\Lambda_s = 9$ and $\sigma = 3$, the following variables would take value 1: φ_7, $\varphi'_{7((0,0),(1,1))}$, $\varphi'_{7((1,1),(4,2))}$, $\varphi'_{7((4,2),(5,4))}$, $\varphi''_{1(4,0,7)}$, $\varphi''_{1(0,7,9)}$, $\varphi''_{4(2,0,3)}$, $\varphi''_{4(3,3,7)}$, and $\varphi''_{4(0,7,9)}$. For the right-most block in the same machine, the following variables would instead take value 1: φ_{10}, $\varphi'_{10((0,0),(2,1))}$, $\varphi'_{10((2,1),(5,2))}$, $\varphi'_{10((5,2),(5,4))}$, $\varphi''_{2(11,0,6)}$, $\varphi''_{2(12,6,10)}$, $\varphi''_{2(0,10,19)}$, $\varphi''_{3(9,0,3)}$, $\varphi''_{3(10,3,9)}$, $\varphi''_{3(0,9,10)}$, and $\varphi''_{3(0,10,19)}$.

Model (17)–(24) may have a slow convergence to an optimal solution because it may contain a large number of variables: $O(\Lambda_s)$ for the first set of cuts, $O(\Lambda_s m^*_s \sigma)$

for the second, and $O(\Lambda_s m_s^* n)$ for the third. We use a heuristic to limit the value of Λ_s and some preprocessing techniques to improve its computational performance.

In terms of preprocessing, we adopted the two following techniques:

- for each item $j \in N_s$ we compute, through dynamic programming, the maximum width $w'_j \leq W - w_j$ that can be taken by a subset of items packed side by side with j. If $w_j + w'_j < W$, then the width of item j is increased to $w_j = W - w'_j$.
- let p be the item with smallest width and q the item with second smallest width. If there is an item j ($j \neq p \neq q$), such that, $w_j + w_p \leq W$, $w_j + w_q > W$, and $l_j \geq l_p$, then we pack items j and p alone in a single block of length l_j.

A consequence of the first preprocessing is that if $w'_j = 0$ (that is, no item can be packed side by side with j) then w_j is set to W and j is packed alone in a block of length l_j.

Limiting the value taken by Λ_s may decrease consistently the number of variables. We pursue this by means of a two-step algorithm. By remarking that any upper bound for z_s^{CC} is also a valid upper bound for Λ_s, we first solve model (17)–(24) heuristically, by limiting Λ_s to a small value (in our implementation we chose $\Lambda_s = 1.5 \times \max_{j \in N_s}\{l_j\}$). If the solution obtained, say, $\bar{z}_s^{CC}$, satisfies $\bar{z}_s^{CC} \leq \Lambda_s$, then we terminate with a proof of optimality. If instead $\bar{z}_s^{CC} > \Lambda_s$, we set $\Lambda_s = \bar{z}_s^{CC}$ and solve the model once more. A solution obtained for a given Λ_s can be easily mapped into a solution for another $\Lambda'_s > \Lambda_s$, thus, the solution obtained at the end of first step is given as a "warm start" to the solver at the beginning of the second step.

Scheduling component (SC)

The second component of our decomposition approach is a scheduling problem that takes as input the information provided by solution of the cutting component. Recall that S is the set of item classes (defined by thickness and color). Let us now define by C the set of colors, and by $S_c \subseteq S$ the subset of classes whose color is c, for $c \in C$. Model (17)-(24) is invoked for each class $s \in S$ to determine the minimum length film necessary to produce all items in that class. Let $\bar{z}_s^{CC}$ be the optimal solution value of the model and let

$$b_c = \sum\nolimits_{s \in S_c} \bar{z}_s^{CC} \tag{25}$$

be the minimum film length required to produce all items of color c. Let us also determine all possible positions for a first-stage cut on items of color c by computing

$$\mathcal{L}_c = \left\{ x = \sum\nolimits_{j \in T} l_j : 0 \leq x \leq b_c, T \subseteq \bigcup\nolimits_{s \in S_c} \{N_s\} \right\}. \tag{26}$$

Production typically happens on more than one machine, and in such a case the length b_c will be split among the machines. The values used to split b_c can be limited to those in (26). This consideration is at the basis of a MILP model that we developed to solve the SC.

To this purpose, we build a graph $\widetilde{G} = (\widetilde{C}, \widetilde{A})$. The vertex set is $\widetilde{C} = C \cup \{0\}$, where 0 is a dummy vertex that represents the beginning and the end of the activities. The arc set $\widetilde{A}$ connects each pair of vertices in $\widetilde{C}$. Let s_{cd} be the value of the setup length when changing production from color c to color d (recall that no setup occurs when keeping the same color and just changing the thickness). We aim at creating a working sequence for each machine $p \in M$, specifying the order in which colors are processed on that machine, and their corresponding quantity.

Let us first introduce a variable z^{SC} indicating the value of the optimal makespan. Let us also introduce two sets of three-index binary variables:

- $y_{cdp} = \begin{cases} 1 & \text{if vertex } c \text{ is followed by vertex } d \text{ on machine } p; \\ 0 & \text{otherwise;} \end{cases}$ for $c, d \in \widetilde{C}$, $p \in M$;
- $\omega_{c\ell p} = \begin{cases} 1 & \text{if } \ell \text{ units of film of color } c \text{ are used on machine } p; \\ 0 & \text{otherwise;} \end{cases}$ for $c \in C$, $\ell \in \mathcal{L}_c$, $p \in M$.

The scheduling component can be modeled as the following MILP model:

$$\min \quad z^{SC} \tag{27}$$

subject to

$$z^{SC} \geq \sum_{c \in \widetilde{C}} \sum_{d \in \widetilde{C}} s_{cd} y_{cdp} + \sum_{c \in C} \sum_{\ell \in \mathcal{L}_c} \ell \omega_{c\ell p} \qquad p \in M \tag{28}$$

$$\sum_{c \in \widetilde{C}} y_{0cp} = 1 \qquad p \in M \tag{29}$$

$$\sum_{c \in \widetilde{C}} y_{c0p} = 1 \qquad p \in M \tag{30}$$

$$\sum_{p \in M} \sum_{\ell \in \mathcal{L}_c} \ell \omega_{c\ell p} = b_c \qquad c \in C \tag{31}$$

$$\sum_{d \in \widetilde{C}} y_{dcp} = \sum_{d \in \widetilde{C}} y_{cdp} \qquad c \in \widetilde{C}, p \in M \tag{32}$$

$$\sum_{\ell \in \mathcal{L}_c} \omega_{c\ell p} \leq \sum_{d \in \widetilde{C}} y_{dcp} \qquad c \in C, p \in M \tag{33}$$

$$\sum_{c \in T} \sum_{d \in T} y_{cdp} \leq |T| - 1 \qquad T \subset C, |T| \geq 1, p \in M \tag{34}$$

$$y_{cdp} \in \{0, 1\} \qquad c, d \in \widetilde{C}, p \in M \tag{35}$$

$$\omega_{c\ell p} \in \{0, 1\} \qquad c \in C, \ell \in \mathcal{L}_c, p \in M \tag{36}$$

The objective function (27) minimizes the makespan of the schedule. This is forced to be not lower than the total workload (including setup and production times) on each machine p by constraints (28). Constraints (29) and (30) ensure, respectively, that a single path starts and ends at vertex 0 for each machine. Note that y_{00p}=1 would correspond to an empty path for machine p. Constraints (31) guarantee that b_c units are processed in total for each color c. Constraints (32) impose flow conservation for each path on each vertex. Constraints (33) impose that if color c is processed on machine p then the path adopted for p should enter vertex c. Constraints (34) are the classical subtour elimination constraints and are used to impose the connectivity of the solution.

The optimal solution value for z_{SC} represents a lower bound on the optimal PRPP solution value. The values taken by the y and ω variables are used to derive also a valid upper bound, as explained in the following.

Upper Bounding Procedures

The solution of the CC consists of a series of selected blocks. These can be used to produce all the items, but , in order to produce a feasible PRPP solution, they must be allocated to the machines by taking into account set-up times and makespan minimization. In this section we propose two upper bounding procedures that make use of this idea, that is, they take in input the blocks generated by the CC and then focus on the best way to schedule them on the machines.

A Heuristic Based on a Generalized Assignment Problem

The solution of the CC consists of the blocks defined by the selected $\bar{\varphi}_p$ variables (selected first-stage vertical cuts of length p), whose total length is equal to $\bar{z}_s^{CC}$ as stated in (17). The set of selected blocks for a color c is thus given by the union of the selected blocks for all $s \in S_c$, and their total length is b_c as stated in (25).

The solution of the SC does not directly consider the item lengths, but partitions b_c into a set of film segments whose length is determined by the selected $\bar{\omega}_{c\ell p}$ variables, each corresponding to a segment of length ℓ (refer also to (31)) .

If we manage to allocate the selected blocks from the CC into the film segments produced by the SC, then we would produce a proven optimal solution (being feasible for both cutting and scheduling and having cost equal to the lower bound z^{SC}). If this is not possible, we can at least use the allocation of minimum excess, which produces a heuristic solution. This idea is at the basis of our first upper bounding procedure.

To simplify notation, let B be the set of blocks selected by the cutting component, l'_j the length of each block $j \in B$, and c'_j the color of each block $j \in B$. Let F be the set of film segments produced by the scheduling component, l''_i the length of each segment $i \in F$ and c''_i the color of each segment $i \in F$. Let also m_i be the index of the machine processing block $i \in F$ anf ζ_p the total time (working and setup) of machine $p \in M$ in the solution of the SC.

By introducing the following decision variables

- $x_{ij} = 1$ if block $j \in B$ is assigned to segment $i \in F$, 0 otherwise;
- $s_i =$ value of the slack of segment $i \in F$;
- $z =$ makespan,

the problem of allocating blocks to segments can be modeled as the following MILP:

$$\min \quad z \tag{37}$$

subject to

$$\sum_{i \in F} x_{ij} = 1 \qquad j \in B \tag{38}$$

$$\sum_{j \in B, c'_j = c''_i} l'_j x_{ij} \leq s_i + l''_i \qquad i \in F \tag{39}$$

$$z \geq \sum_{i \in F, m_i = p} s_i + \zeta_p \qquad p \in M \tag{40}$$

$$x_{ij} \in \{0, 1\} \qquad i \in F, j \in B \tag{41}$$

The objective function (37) minimizes the makespan. Constraints (38) state that every block must be assigned exactly once. Constraints (39) force the sum of the slack variable s_i and the length l''_i of the segment $i \in F$ to be not smaller than the sum of the lengths l'_j of the blocks assigned to that segment. Constraints (40) compute the makespan by forcing z to be greater than or equal to the original working time of a machine p plus the total slack assigned to that machine.

Model (37)–(41) is reminiscent of the well-known *generalized assignment problem* (GAP), so our first upper bounding procedure is called *GAP based approach* (GAPBA) in the following.

An Iterated Local Search Algorithm

Our second upper bounding procedure relies on scheduling the set B of blocks generated by the CC on m identical machines (see the previous Section for a formal definition of B). The objective is to minimize the makespan, i.e., the maximum completion time of a block, but, as the blocks may be of different colors, sequence-dependent setup times must be taken into account. According to the three-field notation proposed by Graham et al. (1979), this $\mathcal{NP}$-hard problem can be denoted as $P|s_{hk}|C_{\max}$.

The algorithm that we use to solve the $P|s_{hk}|C_{\max}$ is an adapted version of the ILS-RVND heuristic by Subramanian (2012), originally designed to solve vehicle routing problems (VRPs). The most common objective function in VRPs is to minimize the total tour length, which is equivalent, on scheduling problems, to minimize the total time spent by the machines to process all jobs. However, the objective function of the $P|s_{hk}|C_{\max}$ minimizes the makespan, which is equivalent to minimizing the longest route length on VRPs. The adaptations that we implemented to take care of this difference essentially consist in modifying the way the objective function is computed throughout the algorithm. The input data for the ILS-RVND is basically a matrix that stores the time spent by a machine to process a block k immediately after a block h, which is computed by summing up the processing time of block k plus the setup time between h and k. The values for the processing

times are derived from the length of the blocks generated by the CC and the setup times come directly from the PRPP input data, more precisely, from the setup times between different colors.

In short, ILS-RVND is a multi-start heuristic that combines *iterated local search* (ILS), see, e.g., Lourenço et al. (2010), with *randomized variable neighborhood descent* (RVND), see, e.g., Mladenović and Hansen (1997) and Subramanian et al. (2010). The algorithm alternates between local search and perturbation procedures, where the latter modifies a local optimal solution by randomly moving or swapping items between different machines. Initial solutions are generated using a greedy randomized algorithm. A detailed and comprehensive description of ILS-RVND can be found in Subramanian (2012).

The local search is performed in two different levels: (i) inter-machine, that is, moves involving different machines; (ii) intra-machine, i.e., moves involving a single machine. In what follows, we describe each of the neighborhood structures used in ILS-RVND. They are exhaustively examined in a random order using the best improvement strategy.

Inter-machine neighborhoods:

- `Insertion inter(1,0)`: a block is removed and inserted in another machine;
- `Insertion inter(2,0)`: two adjacent blocks are removed and inserted in another machine;
- `Swap inter(1,1)`: permutation of two blocks assigned to different machines;
- `Swap inter(2,1)`: permutation of a block in a machine with two adjacent blocks in another machine;
- `Swap inter(2,2)`: permutation of two pairs of adjacent blocks in two different machines;
- `Cross`: the sequences of two distinct machines are split into two, creating two initial and two final subsequences. The initial subsequences are interchanged to build two new sequences (each containing an initial and a final subsequence provided by two distinct machines).

Intra-machine neighborhoods:

- `Swap`: permutation of two blocks in the same machine;
- `1-block insertion`: a block is removed from its current position and inserted in another position in the same machine;

- `2-block insertion`: two adjacent blocks are removed and inserted in another position in the same machine;
- `3-block insertion`: three adjacent blocks are removed and inserted in another position in the same machine.

Computational Experiments

The algorithms were coded in C++ and the experiments were conducted on a single core of an Intel Core i7 processor with 3.4 Ghz and 16 GB of RAM, running Ubuntu 12.04. All formulations were solved using CPLEX 12.6. A time limit of 3600 seconds and a memory limit of 10 GB were imposed for the compact formulation. For the CC, we set a time limit of 1200 seconds because only one instance could not be solved to optimality within such limit (and the remaining instances could not be solved even allowing a much larger time). The ILS-RVND algorithm was executed 10 times for each instance by adopting the same parameter values as in the original work Subramanian (2012). As for the SC, we first imposed a time limit of 3600 seconds but we later verified that this value was overestimated. Figure 4 depicts the value of the average gap considering all instances (described in details in the next section) between the lower bound after a particular runtime and the lower bound found after 3600 seconds. We can observe that the average initial gap is already small (0.17%) and after 300 seconds it reduces to 0.12%. The improvement obtained from that point on is not significant. Therefore, we decided to adopt 300 seconds as time limit for the SC, as it seems that this setting offers good compromise between time spent and lower bound quality.

Instances

Two sets of instances have been created on the basis of observations of processes in the industry producing plastic bags that was at the origin of our research. The values of the parameters were generated using uniform distribution considering the minimum and maximum values observed in practice. The first set contains 50 small instances with number of items ranging from 10 to 50, while the second one is composed of larger instances with number of items ranging from 100 to 250. For each value of n (independently of the set), two different numbers of machines were selected and two groups of instances were created, each group containing 5 randomly gener-

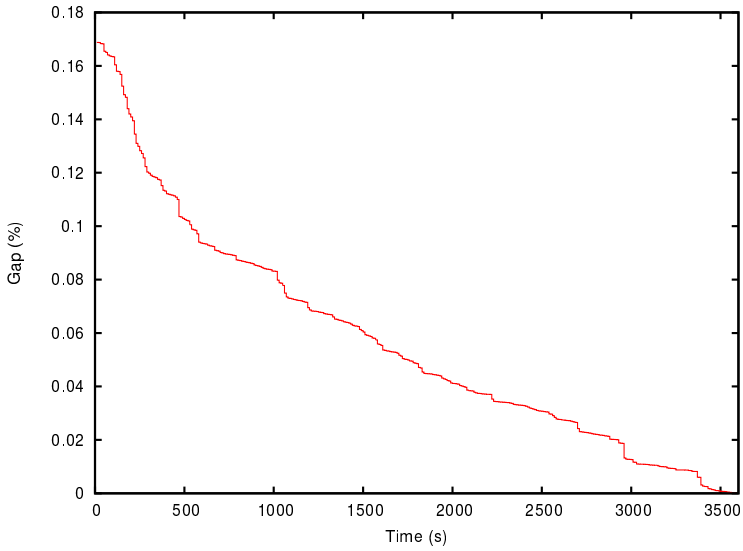

Figure 4: Evolution of the lower bound produced by the SC.

ated instances. Table 2 shows the parameters adopted for each group of instances. The number of colors (#colors), the number of classes (#γ), and the number of slits (σ) were generated by using a random integer uniform distribution within the indicated intervals. For all instances, the value of W was set to 180, whereas the values of w, l, and s were uniformly randomly generated as integer values in the intervals [30,180], [10,200], and [10,30], respectively, with s being restricted to take values that are multiples of 5.

Table 2: Instance parameters

Small instances					Large instances				
n	m	#colors	#γ	σ	n	m	#colors	#γ	σ
10	2	[2,3]	[#colors,3]	[1,2]	100	4	[4,6]	[#colors,10]	[1,4]
	3	[2,3]	[#colors,3]	[1,2]		6	[4,6]	[#colors,10]	[1,4]
20	2	[2,3]	[#colors,4]	[1,2]	150	6	[4,8]	[#colors,14]	[1,4]
	3	[2,3]	[#colors,4]	[1,2]		8	[4,8]	[#colors,14]	[1,4]
30	3	[2,4]	[#colors,6]	[1,3]	200	6	[4,8]	[#colors,14]	[1,5]
	4	[2,4]	[#colors,6]	[1,3]		8	[4,8]	[#colors,14]	[1,5]
40	3	[2,4]	[#colors,8]	[1,3]	250	8	[4,10]	[#colors,18]	[1,5]
	4	[2,4]	[#colors,8]	[1,3]		10	[4,10]	[#colors,18]	[1,5]
50	4	[2,5]	[#colors,10]	[1,4]					
	5	[2,5]	[#colors,10]	[1,4]					

Algorithm performance

With respect to the small instances, we report the results for all algorithms proposed in this paper, whereas for the larger instances, we only present the results found by the lower bounding procedure (cutting plus scheduling component) and by the ILS-RVND heuristic. In the latter case it is prohibitively expensive to use the compact formulation, and the GAPBA approach produces poor upper bounds because it relies on the output of the SC, which is executed for a short time period, thus generating low quality sequences.

In the tables presented hereafter, #inst represents the number of instances of a group, LB and UB correspond to the lower and upper bound, respectively, #opt denotes the number of optimal solutions found, time (s) indicates the CPU time in seconds, BKLB represents the best known lower bound and gap (%) is the percentage gap between the UB found by a given method and BKLB, that is: $100(UB - BKLB)/UB$. Detailed results for each instance are provided in the appendix (see Tables 6 and 7).

Table 3 presents the aggregate results obtained by the compact formulation on the small instances. The formulation finds proven optimal solutions for 13 instances with up to 20 items, but cannot prove optimality for any of the larger instances. The average CPU time for the 10-item instances is acceptable, but it increases rapidly with the number of items. Nonetheless, the average gaps between the UBs an the BKLBs are of high quality, even for the 50-item instances. Furthermore, we notice that the average gaps tend to increase with the number of machines.

Table 4 presents, for the small instances, the average results obtained in terms of lower bound by the CC followed by the SC (CS-LB), and in terms of upper bounds by GAPBA and ILS-RVND. For CS-LB we report the average CPU time spent by the CC (time_{CC}) and by the SC (time_{SC}), and the average lower bound produced for each group of instances. For GAPBA we provide the average values of CPU time, gap, and upper bound, as well as the number of proven optimal solutions found (#opt). As for ILS-RVND, we compute for each instance the average time, the best and average gaps, and the average UB (by considering the 10 runs). Then in the table we report the means of these values considering the 5 instances per line, as well as the number of proven optimal solutions.

The combination of the two components clearly runs faster than the compact model and the former finds much better LBs than the latter, except for some 10-

Table 3: Aggregate results for the compact formulation

n	m	#inst	avg. BKLB	Compact formulation avg. UB	avg. LB	gap (%)	#opt	time (s)
10	2	5	571.2	571.2	571.2	0.0	5	18.3
	3	5	287.8	287.8	287.8	0.0	5	32.5
20	2	5	1001.0	1001.0	993.5	0.0	1	2895.4
	3	5	595.6	596.4	585.6	0.1	2	2961.7
30	3	5	826.1	827.6	819.5	0.2	0	3488.8
	4	5	640.2	643.8	602.4	0.6	0	3153.6
40	3	5	1027.6	1035.2	834.2	0.8	0	3595.5
	4	5	972.2	982.6	806.6	1.1	0	3595.8
50	4	5	1088.8	1092.4	682.9	0.4	0	3596.4
	5	5	806.6	823.4	441.2	2.1	0	3595.9
avg/total		50	781.7	786.1	662.5	0.5	13	2693.4

Table 4: Results obtained by CS-LB, GAPBA, and ILS-RVND for the small instances

n	m	#inst	CS-LB time$_{CC}$ (s)	time$_{SC}$ (s)	avg. LB	GAPBA time (s)	gap (%)	avg. UB	#opt	ILS-RVND avg. time (s)	best gap (%)	avg. gap (%)	avg. UB	#opt
10	2	5	< 0.1	< 0.1	571.2	< 0.1	0.0	571.2	5	0.1	0.1	0.0	571.2	5
	3	5	< 0.1	0.2	287.4	< 0.1	1.4	292.0	3	0.1	0.8	0.9	290.4	3
20	2	5	< 0.1	22.5	1001.0	< 0.1	0.0	1001.0	5	0.1	0.1	0.0	1001.0	5
	3	5	0.1	3.9	595.6	< 0.1	0.9	601.0	3	0.1	0.9	1.0	601.0	3
30	3	5	0.7	58.8	826.1	< 0.1	0.2	827.4	2	0.1	0.2	0.2	827.4	2
	4	5	1.2	146.4	640.2	< 0.1	0.5	643.2	2	0.1	0.5	0.5	643.2	2
40	3	5	1.6	155.8	1027.6	< 0.1	0.5	1032.8	2	0.1	0.4	0.4	1032.0	2
	4	5	0.2	299.3	972.2	< 0.1	0.9	981.2	0	0.2	0.4	0.4	976.9	0
50	4	5	1.9	178.6	1088.8	< 0.1	0.2	1091.4	3	0.2	0.2	0.2	1090.8	3
	5	5	10.2	299.6	806.6	0.1	0.9	814.2	0	0.2	0.7	0.7	812.6	0
avg/total		50	1.5	116.5	781.7	< 0.1	0.6	785.5	25	0.1	0.4	0.4	784.6	25

item instances and very few 20-item instances (see Table 6 for details). We can also observe that scheduling is much more time consuming than cutting. GAPBA and ILS-RVND have an equivalent performance and both methods were capable of finding 25 proven optimal solutions, meaning that CS-LB was equal to the best UB found by GAPBA and ILS-RVND in half of the total number of instances.

GAPBA and ILS-RVND are very competitive in terms of CPU time and solution quality, the former is slightly faster, while the later provides solutions with better gaps. Both find the same number of optimal solutions and the average gap difference is of only 0.2%. Regarding the scalability of the methods with respect to the instance size, they both appear to be much better suited for practical applications, as shown by the experiments.

Figure 5 illustrates the average gap obtained by the compact formulation, GAPBA, and ILS-RVND for the small instances. The formulation found, on average, the best gaps for the instances with $n \leq 20$, but it is outperformed by the other two approaches as the size of the instances increase. Moreover, the solutions found by ILS-RVND appear to be systematically better than or equal to those found by GAPBA, with just a very limited increase in the required computational effort. Overall, the proposed decomposition manages to find good quality solutions and small average gaps within a much smaller computational effort that the one required by the compact formulation.

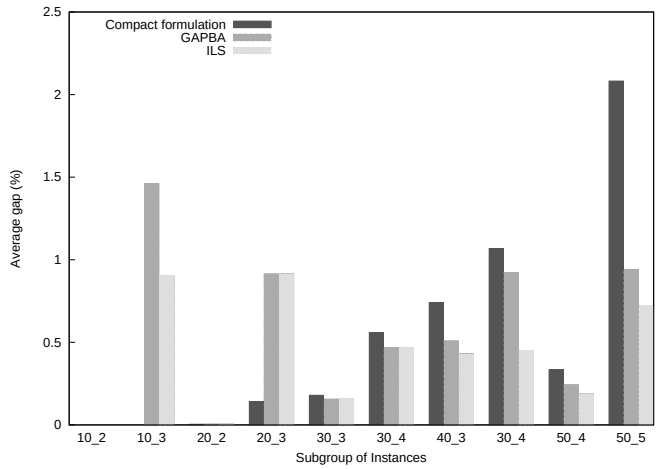

Figure 5: Comparison of the upper bounds on some small instances

Table 5 shows the aggregate results obtained by CS-LB and ILS-RVND for the large instances involving 100 to 250 items. We do not prove the optimality for any

instance, but we report significantly small gaps for instances with 100 items. The gap is considerably small for instances with up to 150 items, but for the larger instances (200 and 250 items) it tends to increase considerably, especially when the number of machines is large. Nevertheless, this does not necessarily imply that the UBs found by ILS-RVND are of poor quality, since one cannot ensure that the CS-LBs are of high quality. In addition, the difficulty in producing high quality blocks by the CC may affect the performance of ILS-RVND in finding high quality solutions.

Table 5: Results obtained by DM and ILS-RVND for the large instances

			CS-LB			ILS-RVND			
n	m	#inst	time_{CC} (s)	time_{SC} (s)	avg. LB	avg. time (s)	best gap (%)	avg. gap (%)	avg. UB
100	4	5	18.4	300.0	2056.8	1.4	0.4	0.4	2065.8
	6	5	4.2	300.0	1440.7	1.8	0.7	0.8	1452.7
150	6	5	188.4	300.0	1909.6	4.0	4.5	4.6	1990.4
	8	5	123.0	300.0	1615.9	5.4	0.8	0.9	1631.2
200	6	5	225.5	300.0	2774.2	13.9	3.5	3.5	2857.4
	8	5	496.0	300.0	1449.1	3.6	17.5	17.6	1771.7
250	8	5	360.2	300.0	1590.1	6.5	9.4	9.5	2260.4
	10	5	634.3	300.0	2040.3	13.2	20.8	20.9	1965.8
avg/total		40	289.6	300.0	1859.6	6.3	7.2	7.3	1999.4

Concluding Remarks

In this paper we introduced the Plastic Rolls Production Problem (PRPP), which integrates cutting and scheduling decisions, thus generalizing several well-known optimization problems. We proposed different approaches for obtaining lower and upper bounds for the PRPP. The first one is a compact mathematical formulation that works quite well for instances involving up to 20 items. The second approach is based on a two-phase decomposition method designed to generate improved lower bounds, especially for instances with more than 20 items, and a heuristic information on the way to cut items into blocks. These blocks are later used in a generalized assignment problem based approach (GAPBA), as well as in a iterated local search (ILS-RVND) heuristic, to produce feasible solutions. The compact formulation found high quality lower and upper bounds for very small instances, but failed in producing good solutions for larger instances. The decomposition method found high quality lower bounds for instances with up to 100 items, although its

performance seems to degradate for larger instances. Both GAPBA and ILS-RVND generated high quality solutions for instances with up to 50 items, but only the latter managed to produce good solutions for larger instances with up to 250 items.

As for future work, one can improve the quality of the performance of the two-phase decomposition by implementing a combinatorial branch-and-bound approach for solving the cutting component, as well as a column generation based algorithm for solving the scheduling component. Additionally, improved upper bounds could be obtained by proposing alternative ways of generating blocks for the ILS-RVND heuristic. The study of different problem variants, with alternative objective functions or machine characteristics, is also of interest.

References

D.L. Applegate, R.E. Bixby, V. Chvátal, and W.J. Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, Princeton, NJ, USA, 2007.

J.A. Bennell, J.F. Oliveira, and G. Wäscher. Cutting and packing. *International Journal of Production Economics*, 145(2):449 – 450, 2013.

K. Cantor. *Blown Film Extrusion: An Introduction*. Hanser Publishers, Munich, second edition, 2011.

N. Christofides and C. Whitlock. An algorithm for two-dimensional cutting problems. *Operations Research*, 25(1):30 – 44, 1977.

J.-F. Côté, M. Dell'Amico, and M. Iori. Combinatorial benders' cuts for the strip packing problem. *Operations Research*, 62(3):643 – 661, 2014.

M. Dell'Amico, M. Iori, S. Martello, and M. Monaci. Heuristic and exact algorithms for the identical parallel machine scheduling problem. *INFORMS Journal on Computing*, 20(3):333 – 344, 2008.

R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In E.L. Johnson P.L. Hammer and B.H. Korte, editors, *Discrete Optimization II Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications of the Systems Science Panel of NATO and of the Discrete Optimization Symposium co-sponsored by IBM Canada and SIAM Banff, Aha. and Vancouver*, volume 5 of *Annals of Discrete Mathematics*, pages 287 – 326. Elsevier, 1979.

M.C.N. Gramani, P.M. França, and M.N. Arenales. A lagrangian relaxation approach to a coupled lot-sizing and cutting stock problem. *International Journal of Production Economics*, 119(2):219 – 227, 2009.

J.C. Herz. Recursive computational procedure for two-dimensional stock cutting. *IBM Journal of Research and Development*, 16(5):462 – 469, 1972.

S. Hu, S. Wang, Y. Kao, T. Ito, and X. Sun. A branch and bound algorithm for project scheduling problem with spatial resource constraints. *Mathematical Problems in Engineering*, 2015:9, 2015.

M. Iori, J. J. Salazar-González, and D. Vigo. An exact approach for the vehicle routing problem with two-dimensional loading constraints. *Transportation Science*, 41 (2):253 – 264, 2007.

J. Kallrath, S. Rebennack, J. Kallrath, and R. Kusche. Solving real-world cutting stock-problems in the paper industry: Mathematical approaches, experience and challenges. *European Journal of Operational Research*, 238(1):374 – 389, 2014.

A. Lodi and M. Monaci. Integer linear programming models for 2-staged two-dimensional Knapsack problems. *Mathematical Programming, Series B*, 94(2): 257 – 278, 2003.

H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search: Framework and applications. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 363 – 397. Springer US, 2010.

R. Macedo, C. Alves, and J.M. Valério de Carvalho. Arc-flow model for the two-dimensional guillotine cutting stock problem. *Computers & Operations Research*, 37(6):991 – 1001, 2010.

N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097 – 1100, 1997.

M. Mrad. An arc flow-based optimization approach for the two-stage guillotine strip cutting problem. *Journal of the Operational Research Society*, 66(11):1850–1859, 2015.

E. Silva, F. Filipe Alvelos, and J.M. Valério de Carvalho. An integer programming model for two- and three-stage two-dimensional cutting stock problems. *European Journal of Operational Research*, 205(3):699 – 708, 2010.

E. Silva, F. Alvelos, and J. M. Valério de Carvalho. Integrating two-dimensional cutting stock and lot-sizing problems. *Journal of the Operational Research Society*, 65(1):108–123, 2014. ISSN 1476-9360.

A. Subramanian. *Heuristic Exact and Hybrid Approaches for Vehicle Routing Problems*. PhD thesis, Universidade Federal Fluminense, 2012.

A. Subramanian, L.M.A. Drummond, C. Bentes, L.S. Ochi, and R. Farias. A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, 37(11):1899 – 1911, 2010. Metaheuristics for Logistics and Vehicle Routing.

J.M. Valério de Carvalho. Exact solution of bin-packing problems using column generation and branch-and-bound. *Annals of Operations Research*, 86(0):629 – 659, 1999.

F. Vanderbeck. A nested decomposition approach to a three-stage, two-dimensional cutting-stock problem. *Management Science*, 47(6):864 – 879, 2001.

Appendix

In Tables 6 and 7 we present detailed results for each instance. The name of the instance is made by the number of items, followed by the number of machines and by a progressive number from 1 to 5. The names of the columns have the same meanings as those used in the computational experiments.

Table 6: Detailed results for the small instances

instance	compact formulation				CS-LB			GAPBA			ILS-RVND			
	UB	LB	time (s)	gap (%)	LB	$time_{CC}$ (s)	$time_{SC}$ (s)	time (s)	UB	gap (%)	best UB	avg. UB	time (s)	best gap (%)
10_2_1	678	678.0	34.0	0.0	678.0	0.0	0.0	0.0	678	0.0	678	678.0	0.1	0.0
10_2_2	499	499.0	29.4	0.0	499.0	0.0	0.1	0.0	499	0.0	499	499.0	0.1	0.0
10_2_3	549	549.0	25.3	0.0	549.0	0.0	0.0	0.0	549	0.0	549	549.0	0.1	0.0
10_2_4	493	493.0	1.7	0.0	493.0	0.0	0.0	0.0	493	0.0	493	493.0	0.1	0.0
10_2_5	637	637.0	0.9	0.0	637.0	0.0	0.0	0.0	637	0.0	637	637.0	0.1	0.0
10_3_1	286	286.0	63.9	0.0	286.0	0.0	0.2	0.0	286	0.0	286	286.0	0.2	0.0
10_3_2	329	329.0	3.6	0.0	329.0	0.0	0.3	0.0	342	3.8	342	342.0	0.1	3.8
10_3_3	236	236.0	1.3	0.0	236.0	0.0	0.1	0.0	236	0.0	236	236.0	0.1	0.0
10_3_4	230	230.0	40.3	0.0	228.0	0.0	0.2	0.0	238	3.4	230	230.0	0.1	0.0
10_3_5	358	358.0	53.2	0.0	358.0	0.0	0.1	0.0	358	0.0	358	358.0	0.2	0.0
20_2_1	973	973.0	96.4	0.0	973.0	0.1	0.2	0.0	973	0.0	973	973.0	0.2	0.0
20_2_2	1112	1103.0	3595.1	0.0	1112.0	0.0	0.4	0.0	1112	0.0	1112	1112.0	0.4	0.0
20_2_3	804	799.5	3595.6	0.0	804.0	0.0	0.2	0.0	804	0.0	804	804.0	0.1	0.0
20_2_4	1228	1215.7	3594.7	0.0	1227.9	0.0	102.4	0.0	1228	0.0	1228	1228.0	0.3	0.0
20_2_5	888	876.5	3595.3	0.0	887.9	0.1	9.0	0.0	888	0.0	888	888.0	0.2	0.0
20_3_1	691	680.7	3594.3	0.0	690.9	0.0	13.4	0.0	691	0.0	691	691.0	0.6	0.0
20_3_2	433	426.0	3595.9	0.2	432.0	0.0	2.0	0.0	432	0.0	432	432.0	0.2	0.0
20_3_3	660	660.0	916.1	0.0	659.9	0.5	0.8	0.0	662	0.3	662	662.0	0.3	0.3
20_3_4	504	504.0	3106.7	0.0	501.0	0.1	0.4	0.0	526	4.2	526	526.0	0.1	4.2
20_3_5	694	657.2	3595.4	0.0	693.9	0.0	3.1	0.0	694	0.0	694	694.0	0.6	0.0
30_3_1	720	712.1	3596.6	0.3	717.9	0.0	28.9	0.0	720	0.3	720	720.0	0.5	0.3
30_3_2	640	633.4	3596.3	0.2	638.9	3.2	7.1	0.0	640	0.2	640	640.0	0.3	0.2
30_3_3	876	868.7	3596.2	0.4	872.9	0.2	11.8	0.0	876	0.3	876	876.0	0.4	0.3
30_3_4	949	934.4	3060.5	0.1	947.9	0.0	238.5	0.0	948	0.0	948	948.0	0.8	0.0
30_3_5	953	949.0	3594.7	0.0	952.9	0.1	7.6	0.0	953	0.0	953	953.0	0.6	0.0
30_4_1	769	747.7	3595.9	0.1	768.5	0.0	299.5	0.0	769	0.0	769	769.0	1.3	0.0
30_4_2	519	467.4	3595.3	1.4	511.8	0.4	299.2	0.1	517	1.0	517	517.0	0.4	1.0
30_4_3	524	497.7	3181.7	0.8	519.9	0.0	107.6	0.0	524	0.8	524	524.0	0.5	0.8
30_4_4	511	469.9	3596.4	0.6	508.0	5.7	16.0	0.0	513	1.0	513	513.0	0.4	1.0
30_4_5	896	829.2	1798.7	0.3	892.9	0.0	9.8	0.0	893	0.0	893	893.0	1.1	0.0
40_3_1	1074	1062.2	3595.3	0.0	1073.9	1.2	28.3	0.0	1074	0.0	1074	1074.0	0.9	0.0
40_3_2	919	700.0	3595.6	2.5	896.5	2.8	299.2	0.0	909	1.3	909	909.0	0.5	1.3
40_3_3	1092	893.2	3596.2	0.6	1085.9	0.3	64.1	0.0	1086	0.0	1086	1086.0	0.9	0.0
40_3_4	957	450.1	3595.5	0.1	955.9	3.8	87.9	0.0	957	0.1	957	957.0	0.5	0.1
40_3_5	1134	1065.7	3594.9	0.7	1125.8	0.1	299.4	0.0	1138	1.1	1134	1134.0	1.1	0.7
40_4_1	827	672.9	3596.0	0.7	821.3	0.6	299.3	0.0	824	0.2	824	824.0	1.0	0.2
40_4_2	1069	893.0	3595.6	0.8	1060.8	0.0	299.4	0.0	1064	0.3	1064	1065.3	2.7	0.3
40_4_3	1024	720.1	3595.5	1.5	1009.1	0.0	299.2	0.0	1037	2.6	1017	1017.0	1.2	0.7
40_4_4	1207	1093.2	3595.7	0.6	1199.6	0.0	299.3	0.0	1201	0.1	1201	1201.0	2.9	0.1
40_4_5	786	653.9	3596.1	2.0	770.4	0.6	299.2	0.0	780	1.2	777	777.2	0.8	0.8
50_4_1	885	432.4	3596.4	0.0	884.9	8.4	133.1	0.0	885	0.0	885	885.0	1.5	0.0
50_4_2	1356	870.7	3596.3	0.0	1355.9	0.0	142.2	0.0	1356	0.0	1356	1356.0	3.9	0.0
50_4_3	1057	912.6	3596.4	1.1	1045.1	0.1	299.3	0.0	1057	1.0	1054	1054.0	1.7	0.8
50_4_4	1353	681.1	3596.4	0.1	1352.0	0.0	299.1	0.0	1353	0.1	1353	1353.0	3.2	0.1
50_4_5	811	517.6	3596.7	0.6	806.0	1.1	19.2	0.0	806	0.0	806	806.0	1.3	0.0
50_5_1	848	428.1	3595.4	3.0	822.8	0.0	299.2	0.2	833	1.2	833	833.4	2.1	1.2
50_5_2	891	535.0	3596.6	0.5	886.6	0.1	299.3	0.1	893	0.7	891	891.0	2.2	0.4
50_5_3	797	412.0	3595.8	3.2	771.6	0.2	299.2	0.0	788	2.0	781	781.1	2.0	1.2
50_5_4	756	401.2	3596.1	2.2	739.0	50.6	299.3	0.0	743	0.5	743	743.0	1.1	0.5
50_5_5	825	429.8	3595.6	1.5	813.0	0.1	300.8	0.1	814	0.1	814	814.4	1.9	0.1

Table 7: Detailed results for the large instances

instance	CS-LB			ILS-RVND			
	LB	$time_{CC}$ (s)	$time_{SC}$ (s)	best UB	avg. UB	avg. time (s)	best gap (%)
100_4_1	1746	9.6	300	1755	1755.9	0.8	0.5
100_4_2	2133	7.5	300	2147	2147.2	1.1	0.6
100_4_3	1731	68.4	300	1736	1736.5	0.5	0.2
100_4_4	2516	0.0	300	2519	2519.0	3.1	0.1
100_4_5	2159	6.4	300	2169	2170.3	1.4	0.4
100_6_1	1366	9.3	300	1379	1380.7	0.7	0.9
100_6_2	1310	5.3	300	1319	1320.9	1.0	0.7
100_6_3	1665	0.0	300	1672	1674.6	3.3	0.4
100_6_4	1190	6.1	300	1199	1200.4	0.6	0.8
100_6_5	1675	0.0	300	1686	1687.0	3.4	0.7
150_6_1	1772	118.6	300	1787	1788.7	1.7	0.8
150_6_2	2102	12.3	300	2112	2112.9	3.4	0.5
150_6_3	1366	624.0	300	1721	1723.8	1.8	20.6
150_6_4	1807	187.1	300	1814	1814.0	2.0	0.4
150_6_5	2503	0.1	300	2512	2512.6	11.2	0.4
150_8_1	1941	0.0	300	1958	1960.5	11.3	0.8
150_8_2	1398	50.6	300	1409	1409.8	2.0	0.8
150_8_3	2004	0.0	300	2014	2016.2	11.1	0.5
150_8_4	1326	424.9	300	1339	1341.7	1.4	1.0
150_8_5	1412	139.4	300	1426	1427.8	1.3	1.0
200_6_1	2516	220.2	300	2524	2525.0	4.2	0.3
200_6_2	1931	709.7	300	2295	2296.1	3.3	15.9
200_6_3	2646	197.5	300	2656	2657.8	4.8	0.4
200_6_4	3232	0.0	300	3244	3246.2	29.2	0.4
200_6_5	3547	0.0	300	3559	3562.0	28.0	0.3
200_8_1	1656	364.0	300	1666	1668.0	3.8	0.6
200_8_2	1583	629.9	300	1707	1709.1	2.7	7.3
200_8_3	1751	89.0	300	1757	1759.3	3.4	0.3
200_8_4	1713	200.8	300	1883	1884.5	4.0	9.0
200_8_5	544	1196.3	300	1836	1837.5	4.3	70.4
250_8_1	2328	115.5	300	2345	2345.7	7.8	0.7
250_8_2	2163	244.1	300	2168	2170.3	5.7	0.2
250_8_3	2152	418.6	300	2173	2174.4	8.0	1.0
250_8_4	1540	596.4	300	2311	2313.4	5.7	33.4
250_8_5	2022	426.4	300	2296	2298.1	5.4	11.9
250_10_1	1856	388.6	300	1869	1872.0	5.5	0.7
250_10_2	762	1091.4	300	1869	1870.7	4.6	59.2
250_10_3	2724	0.1	300	2730	2732.0	47.4	0.2
250_10_4	1065	1094.1	300	1697	1698.5	3.8	37.2
250_10_5	1546	597.2	300	1653	1655.9	4.8	6.5