International Journal of Industrial Engineering & Production Research September 2026, Vol. 37, No. 3: 121-133
DOI: 10.22068/ijiepr.37.3.2601

RESEARCH PAPER

A Mixed-Integer Programming and Genetic Algorithm Approach for a Two-Stage Hybrid Flow Shop with Shared Machines in Cardboard Cutting Operations

Oumaima Ben Rebah[1,*], Abdelkarim Elloumi[2], Racem Mellouli[3]

Received 30 October 2025; Revised 09 June 2026; Accepted 01 July 2026;
© Iran University of Science and Technology 2026

ABSTRACT
This paper addresses pattern scheduling within cutting stock problems, treating it as a specialized instance of a two-stage hybrid flow shop (HFS) problem with unrelated parallel machines at each stage. Originating from a real industrial case in the corrugated cardboard-cutting industry (UNIPACK), the problem incorporates machine eligibility restrictions, shared bi-functional machines across stages, and a novel positional constraint that prevents job splitting on shared machines. To address these complexities, we propose a mixed-integer programming (MIP) model that minimizes production costs comprising weighted job flow-time costs and makespan-related labor costs. We complement this with a genetic algorithm (GA) metaheuristic for larger instances. The MIP achieves optimality for instances with up to 20 jobs; for instances with 22–32 jobs, it returns the best feasible solution found within a 225-second time limit. For all instances where an MIP reference exists (up to 32 jobs), the GA solutions deviate from the MIP reference by an average of 5.2%. For larger instances (up to 200 jobs), the GA produces near-optimal solutions in under 120 seconds, demonstrating strong scalability. Computational experiments on 16 benchmark instances confirm the effectiveness of both approaches and highlight their complementary strengths.

KEYWORDS: *Cutting stock; Scheduling; Hybrid flow shop; Eligible machine; Shared machine; Positional constraint; Setup time; Mixed-integer programming; Genetic algorithm.*

1. Introduction

In recent years, companies have recognized the need to enhance the flexibility of their production processes for improved productivity and efficiency. Increasing the number of machines capable of performing identical processes is identified as an effective approach to achieve this objective. However, establishing an optimal schedule is challenging due to inherent combinatorial complexities. Despite the time-consuming nature of scheduling, it has emerged as a critical factor in reducing labor costs, enhancing customer satisfaction, and facilitating streamlined operations.

Certain practical challenges often intertwine cutting stock and operations scheduling, motivating the development of efficient optimization approaches. These challenges involve determining cutting plans using bin or strip packing, or two-dimensional knapsack problems, to minimize waste generated from cutting stock rolls. The literature describes various scheduling settings based on the obtained cutting patterns within the cutting stock context.

Scheduling, defined as the process of allocating operations to resources over time to optimize a criterion [13], encompasses critical issues, with hybrid flow-shop scheduling (HFS) being one of the most significant. Since the introduction of HFS problems by [1], numerous researchers have delved into this area. Various real-world sectors, including glass, steel, textile, electronics, and paper, operate within the hybrid flow-shop environment. A hybrid flow shop, an extension of the classical flow shop, involves parallel machines performing the same operation. When a stage has more than one parallel machine, a new variation is introduced, with each machine at the stage carrying out job tasks. However, hybrid flow-shop scheduling problems pose greater challenges than flow-shop scheduling problems, as demonstrated by [6] and [7], who established their NP-hard nature.

This paper addresses a hybrid flow-shop problem within the corrugated cardboard cutting stock industry, specifically focusing on a constrained two-stage problem with unrelated parallel machines at each stage. The primary objective is to minimize

* *Corresponding author: Oumaima Ben Rebah*
oumaima.benrebah@fss.usf.tn

1. *Department of Management Graduate School of Commerce of Sfax, University of Sfax. Sfax, Tunisia*
2. *Department of Quantitative Methods Faculty of Economics and Management, University of Sfax. Sfax, Tunisia*
3. *Department of Quantitative Methods Faculty of Economics and Management, University of Sfax. Sfax, Tunisia*

production costs, encompassing job flow time costs and makespan-related labor time costs.

The present paper makes the following distinct contributions to the literature:

1. We formulate the first MIP model for a two-stage HFS with unrelated parallel machines that simultaneously incorporates:
   - Machine eligibility restrictions,
   - Shared bi-functional machines across stages,
   - A novel positional constraint that enforces sequential, non-preemptive processing on shared machines.
2. We propose a GA metaheuristic adapted to this constrained problem structure, incorporating tournament selection and a feasibility-preserving crossover operator tailored to the positional constraint.
3. We validate both approaches on 16 benchmark instances derived from real industrial data from the UNIPACK corrugated cardboard facility in Sfax, Tunisia, ranging from 8 to 200 jobs.
4. We provide a systematic comparative analysis of the MIP and GA in terms of solution quality and computational efficiency, demonstrating the complementary strengths of exact and heuristic methods for problems of different scales.

The remainder of the paper is structured as follows: Section II reviews the relevant literature. Section III provides the problem description and notations. Section IV presents the MIP model. Section V details the metaheuristic approach. Section VI reports computational results. Section VII concludes with implications and future directions.

2. Literature Review

This section surveys the relevant literature across three interconnected streams: hybrid flow-shop scheduling, HFS with machine eligibility and shared/multifunctional machines, and integrated cutting stock and scheduling.

2.1. Hybrid flow-shop scheduling

The HFS problem was introduced by [1] and later formalized by [6], who established its NP-hard nature for even two stages. [10] provided one of the earliest comprehensive surveys of the HFS, cataloguing problem variants and solution methods. The seminal work of [7] proved the NP-hardness of the preemptive two-stage multiprocessor flow shop. The survey by [14] remains the most comprehensive treatment of HFS, covering exact models, construction heuristics, local search, and metaheuristics.

Exact MIP approaches for HFS have been developed for various special cases. [4] proposed MIP models and metaheuristics for the unrelated parallel machine scheduling problem with additional resources, which is closely related to our problem. However, most exact models in the literature do not simultaneously handle machine eligibility, shared machines, and the positional constraint introduced in our work.

Metaheuristic approaches dominate the HFS literature for large instances. Recent studies continue to demonstrate the effectiveness of advanced metaheuristics and hybrid optimization frameworks for increasingly complex hybrid flow shop scheduling problems [11, 20], addressing variants with additional realistic constraints such as lag times, transportation, and unloading delays [8]. Research efforts have focused on integrating evolutionary algorithms with local search procedures, adaptive parameter control mechanisms, and decomposition-based optimization techniques to improve solution quality and scalability [3, 11, 15, 20, 21]. These developments confirm the continuing relevance of metaheuristic approaches when exact methods become computationally intractable for realistic industrial-scale instances.

Genetic algorithms have been widely applied: [2] proposed an efficient GA for HFS with multi-processor task problems, while [19] developed a cooperative adaptive GA for HFS with sequence-dependent setup times. Particle swarm optimization (PSO), Tabu Search (TS), and simulated annealing (SA) have also been studied for HFS variants [9, 16, 17]. The present paper focuses on GA, motivated by its demonstrated effectiveness on combinatorial scheduling problems with complex constraints.

2.2. HFS with machine eligibility and shared machines

Machine eligibility (or capability) constraints, where only a subset of machines can process a given job at each stage, significantly increase the complexity of the HFS. [5] studied unrelated parallel machine scheduling with machine eligibility and optional machine constraints, a related problem that shares the eligibility structure with our two-stage formulation. The problem becomes substantially harder when combined with unrelated parallel machines, since processing times vary across machines.

Shared or multifunctional machines capable of

performing operations from multiple stages represent a less-studied but practically important variant. When a machine serves as both a printing and cutting unit (as in our problem), its allocation between stages must be managed carefully to avoid availability conflicts. To the best of our knowledge, the combination of shared machines with a positional constraint that prevents job splitting across stages on the same machine has not been studied before. This constitutes a key novelty of our formulation.

2.3. Integrated cutting stock and scheduling

The integration of cutting stock problems with production scheduling has received growing attention. In cutting stock problems, the goal is typically to determine an optimal set of cutting patterns minimizing waste. When the resulting patterns must then be scheduled on production machines, a two-level optimization problem arises. Several authors like [12, 18] have studied this integration in the paper, metal, and glass industries. Our work extends this line of research to the corrugated cardboard context with polygonal items, where a bin-packing pre-stage generates patterns that serve as input to the HFS scheduling problem.

2.4. Research gap

Despite the rich literature on HFS, no prior work simultaneously addresses:

- A two-stage HFS with unrelated parallel machines,
- Shared bi-functional machines with availability disjunction across stages,
- A positional constraint for shared machines enforcing sequential processing without preemption,
- Machine eligibility for each job at each stage,
- Integration with a real bin-packing pre-stage, generating the job set.

The present paper fills this gap through a comprehensive MIP formulation and GA, validated on industrial data.

3. Problem Statement and Notations

This section formally defines the primary problem variant under consideration. Our empirical investigation centers on corrugated cardboard manufacturing, specifically conducted at "UNIPACK".

3.1. A pre-solved bin-packing problem

Before addressing the scheduling problem, it is assumed that a preceding bin-packing problem has been solved to provide input data regarding the jobs. This preliminary problem involves a set of polygonal items for production (refer to Figure 1) packed within a set of two-dimensional rectangular bins. A mixed-integer programming model minimizes the number of bins required to cover the full demand for items. This model extends the classical bin-packing problem formulation with rectangular items (see Annex 1). The approach divides polygons into rectangular shapes and imposes specific constraints to efficiently reconstruct the polygons in the problem solution. The output of this preliminary problem, for a given demand, serves as the input data for the scheduling problem addressed in the subsequent subsection.

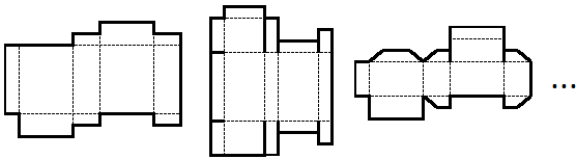

Fig.1. Example of required cardboard items with polygon shapes

3.2. Cutting operations scheduling problem as a special hybrid flow shop

The proposed formulation is particularly suitable for production environments characterized by highly heterogeneous customer demands and a limited number of repeated item types, as illustrated by the distinction between heterogeneous demand and demand with several repeated items in Figures 2 and 3.

In this scenario, items are assigned independent numbering, and based on the solution to the prior bin-packing problem, each resulting bin is associated with a cutting pattern. These bins, filled with items, are treated as a set of jobs to be processed on a designated set of machines. A key advantage is that demand is fulfilled immediately upon completion of pattern scheduling, eliminating surplus production

Two categories of jobs are considered, each involving up to two primary operations:

- The first subset comprises jobs requiring only cutting (one operation, Stage 2 only).
- The second subset involves jobs requiring printing followed by cutting (two operations, Stages 1 and 2 sequentially).

Additionally, the workshop is equipped with three types of machines:

- Type 1: machines exclusively designed for printing.
- Type 2: machines dedicated solely to cutting.

- Type 3: bi-functional machines capable of both printing and cutting. When a job is assigned to a Type 3 machine, printing and cutting are performed sequentially (not simultaneously) on the same machine, with a negligible transition time between the two sub-operations. This is a tandem arrangement: the cutting sub-operation begins immediately upon completion of printing on the same machine. Importantly, if a job starts printing on a Type 3 machine and completes cutting on the same machine, it must maintain the same position in that machine's job sequence across both stages. This positional constraint detailed in the MIP formulation prevents job splitting and preemption, ensuring uninterrupted tandem processing.

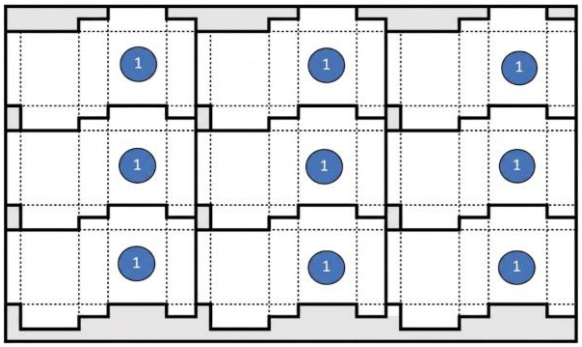

Fig.2. Example of a cutting pattern (bin with packed items) for demand with several repeated item types

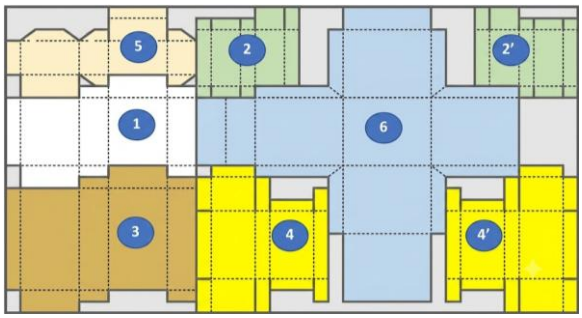

Fig.3. Example of a cutting pattern for strongly heterogeneous demand with few repeated items

This problem can be conceptualized as a two-stage hybrid flow shop incorporating setup constraints (transfer times and changeover times), machine eligibility, and the novel shared-machine positional constraint. For each machine type, unrelated parallel machines are available, exhibiting distinct speed performances based on the employed technology. Two alternatives exist for jobs requiring printing followed by cutting (See Figure 4):

- Alternative 1: The job is processed on two distinct machines, a printing-enabled machine (Type 1 or 3) at Stage 1, followed by a cutting-enabled machine (Type 2 or 3) at Stage 2. A transfer time applies when transitioning between machines.
- Alternative 2: The job is fully processed on one bi-functional machine (Type 3), performing printing then cutting tandem-sequentially on the same machine, with the positional constraint enforced.

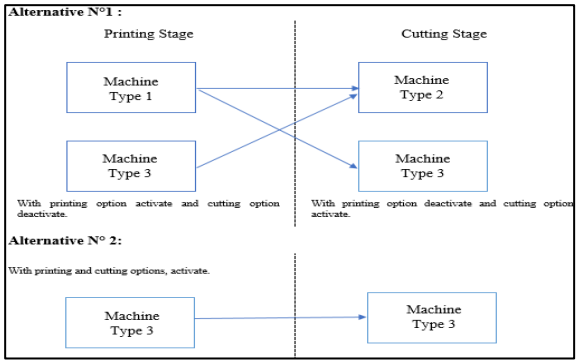

Fig.4. Description of the two processing stages

To formulate a precise mathematical representation of the problem, we introduce the following data notations:

3.3. Notations

$\mathcal{I} = \{1, \dots, I\}$: Set of items types of index $i \in \mathcal{I}$;
$\mathcal{M} = \{1, \dots, m\}$: Set of machines of index $j \in \mathcal{M}$;
$\mathcal{K} = \{1, \dots, K\}$: Set of patterns (jobs) with indices k, k', covering exactly all required items $i \in \mathcal{I}$ and as the set of rectangular bins with assigned cutting schemes for each bin, calculated from the optimal or close-to-optimal solution of a related bin packing problem;

$$a_k = \begin{cases} 1, & \text{if k must be printed,} \\ 0, & \text{otherwise} \end{cases}, \text{ for } k \in \mathcal{K};$$

$$b_j = \begin{cases} 1, & \text{if machine j is enabled to print only.} \\ 2, & \text{if machine j is enabled to cut only.} \\ 3, & \text{if machine j is enabled to both print and then cut with a neglectable waiting time between the two processes.} \end{cases},$$

for $j \in \mathcal{M}$;
Note: b_j this is a fixed parameter used to partition the machine set at model instantiation; it does not appear as a decision variable in any constraint.

$$e_{j,k,s} = \begin{cases} 1, & \text{if machine j is capable (eligible) to process job k on stage s,} \\ 0, & \text{otherwise} \end{cases}$$

$\mathcal{K}, s = 1, 2$; Note that when $e_{j,k,s} = 0$, we set $p_{k,s,j} = +\infty$.
$\mathcal{M}_{k,s}$: Subset of machines j in $\mathcal{M}$ capable to process

pattern k at stage s (i.e., $e_{j,k,s} = 1$). Specifically:

- For $s = 1$, $\mathcal{M}_{k,s} = \emptyset$ if $a_k = 0$; and $\mathcal{M}_{k,s} \subseteq \{j \in \mathcal{M}, b_j = 1 \text{ or } 3\}$ if $a_k = 1$.
- For $s = 2$, $\mathcal{M}_{k,s} \subseteq \{j \in \mathcal{M}, b_j = 2 \text{ or } 3\}$.

$\mathcal{M}_k$: Subset of machines in $\mathcal{M}$ capable to process pattern k, i.e., $\mathcal{M}_k = \mathcal{M}_{k,1} \cup \mathcal{M}_{k,2}$, for $k \in \mathcal{K}$;

$\mathcal{K}_j$: Subset of patterns in $\mathcal{K}$ that can be processed on machine j, for $j \in \mathcal{M}$;

$SC_{k'kj}$: Changeover time to switch from pattern k' to pattern k on machine j.

$ST_{kj'j}$: Transfer time to transfer pattern k from machine j' to machine j.

p_{kjs}: Processing time of pattern k on machine j in stage s.

w_k: Unitary flowtime cost of pattern k.

W: Unitary makespan-related labor time cost of the schedule.

U: A sufficiently large positive parameter (big M) used in linearization.

$\mathcal{T} = \{1, \dots, T\}$: Set of positions in a job sequence on a machine, with indices t;

3.4. Decision variables

$$x_{kjt} = \begin{cases} 1, \text{if pattern k is } t^{th} \text{ processed pattern on machine j,} \\ 0, \text{otherwise.} \end{cases}$$

$$y_{kjs} = \begin{cases} 1, \text{if pattern k is assigned to machine j} \\ \qquad \text{in the processing stage s,} \\ 0, \text{otherwise.} \end{cases}$$

C_{kj} Completion time of pattern k on machine j.

Cmax The makespan.

4. The Proposed MIP

In this section, we propose the following mixed-integer programming model to solve the problem in an exact way based on the above parameter and decision variable notations. The objective function is about minimizing the production costs, including job flowtime costs and the makespan-related labor time costs as follows:

$$\text{Min} \sum_{k \in \mathcal{K}} \left(w_k \sum_{j \in \mathcal{M}_{k,2}} C_{kj} \right) + W.\text{Cmax} \qquad \text{(Obj)}$$

The objective function combines weighted job flow-time costs (first term) and makespan-related labor costs (second term).

The constraints are organised into four groups:

- Assignment and sequencing,
- Completion time on Type 1 and Type 2 machines,
- Completion time on Type 3 bi-functional machines,
- Makespan.

5. Assignment and Sequencing Constraints

Constraints (1.1) and (1.2) ensure each job is assigned to exactly one machine at each required stage:

$$\sum_{j \in \mathcal{M}_{k,s}} y_{kjs} = a_k, \forall k \in \mathcal{K}, s = 1 \qquad (1.1)$$

$$\sum_{j \in \mathcal{M}_{k,s}} y_{kjs} = 1, \forall k \in \mathcal{K}, s = 2 \qquad (1.2)$$

Constraints (2) and (3) manage sequence positions:

$$\sum_{k \in \mathcal{K}_j} x_{kjt} \le 1, \forall j \in \mathcal{M}, \forall t \in \mathcal{T}:$$

At most one job per position. (2)

$$\sum_{t \in \mathcal{T}} x_{kjt} \le 1, \forall k \in \mathcal{K}, \forall j \in \mathcal{M}_k:$$

At most one position per job per machine. (3)

Constraints (4.1) to (4.4) link assignment variables y to position variables x. The conditions $b_j \in \{1, 2, 3\}$ in these constraints are parameter-based index set restrictions, not products of variables and therefore do not introduce non-linearity. At model instantiation, the machine set M is partitioned into $M_1 = \{j: b_j = 1\}$, $M_2 = \{j: b_j = 2\}$ and $M_3 = \{j: b_j = 3\}$, and each constraint is generated only for the relevant subset:

$$\sum_{t \in \mathcal{T}} x_{kjt} \ge y_{kjs}, \forall s = 1,2;\ \forall k \in \mathcal{K}, \forall j \in \mathcal{M}_{k,s} \qquad (4.1)$$

$$\sum_{t \in \mathcal{T}} x_{kjt} \le \sum_{\substack{s=1,2 \\ j \in \mathcal{M}_{k,s}}} y_{kjs}, \forall k \in \mathcal{K}, \forall j \in \mathcal{M}_k, b_j = 3 \qquad (4.2)$$

$$\sum_{t \in \mathcal{T}} x_{kjt} \le y_{kjs}, \forall k \in \mathcal{K}, s = 1, \forall j \in \mathcal{M}_{k,s}, b_j = 1 \qquad (4.3)$$

$$\sum_{t \in \mathcal{T}} x_{kjt} \le y_{kjs}, \forall k \in \mathcal{K}, s = 2, \forall j \in \mathcal{M}_{k,s}, b_j = 2 \qquad (4.4)$$

Constraint (4.4) is generated only for cutting-only machines ($b_j = 2$) because:

- Printing-only machines ($b_j = 1$) are covered by (4.3),
- Bi-functional machine ($b_j = 3$) require the combined constraint (4.2) to allow a single sequence position to account for both sub-operations. All constraints remain linear, as b_j is a fixed parameter.

Constraints (5) forbid empty positions located between two filled positions in the different job sequences.

$$\sum_{k \in \mathcal{K}_j} x_{kjt} \le \sum_{k \in \mathcal{K}_j} x_{kj,t-1}, \forall j \in \mathcal{M}, \forall t \in \mathcal{T} \backslash \{1\} \qquad (5)$$

5.1. Completion time constraints on type 1 and type 2 machines

Constraints (6) to (9) govern completion times on non-shared machines ($b_j \neq 3$):

$$C_{kj} \le U y_{kjs}, \forall k \in \mathcal{K},\ \forall s = 1,2,\ \forall j \in \mathcal{M}_{k,s},\ b_j \neq 3 \qquad (6)$$

$$C_{kj} \geq p_{kjs} \cdot y_{kjs}, \forall k \in \mathcal{K}, \forall s = 1,2, \forall j \in \mathcal{M}_{k,s}, b_j \neq 3 \quad (7)$$

$$C_{kj} \geq C_{k'j} + SC_{k'kj} + p_{kjs} + U(x_{j\,k'\,t-1} + x_{kjt} - 2) \quad \forall kk' \in \mathcal{K} \;\; k \neq k' \;\; \forall s = 12 \;\; \forall j \in \mathcal{M}_{ks} \cap \mathcal{M}_{k's} \;\; b_j \neq 3 \;\; \forall t \in \mathcal{T}\backslash\{1\} \quad (8)$$

$$C_{kj} \geq C_{kj'} + ST_{kj'j} + p_{kj2} + U(y_{kj2} + y_{kj'1} - 2) \quad \forall k \in \mathcal{K} \;\; \forall j \in \mathcal{M}_{k2} \;\; \forall j' \in \mathcal{M}_{k1} \;\; j \neq j' \;\; \forall t \in \mathcal{T}\backslash\{1\} \quad (9)$$

5.2. Completion time constraints on type 3 (bi-functional) machines

Constraints (10) to (12) govern completion times on shared bi-functional machines (b_j= 3). Recall that on Type 3 machines, printing and cutting are performed sequentially without interruption (strict sequential processing). This implies a hard resource disjunction where the bi-functional machine cannot execute Stage 1 and Stage 2 operations concurrently, requiring explicit state transitions. The positional constraint ensures the same sequence position is used for both sub-operations:

$$C_{kj} \leq U(e_{j,k,1}y_{kj1} + e_{j,k,2}y_{kj2}), \;\; \forall k \in \mathcal{K}, \forall j \in \mathcal{M}_k, \;\; b_j = 3 \quad (10)$$

$$C_{kj} \geq e_{j,k,1}p_{kj1}y_{kj1} + e_{j,k,2}p_{kj2}y_{kj2}, \forall k \in \mathcal{K}, \forall j \in \mathcal{M}_k, \;\; b_j = 3 \quad (11)$$

$$C_{kj} \geq C_{k'j} + SC_{k'kj} + e_{j,k,1}p_{kj1}y_{kj1} + e_{j,k,2}p_{kj2}y_{kj2} + U(x_{j,k',t-1} + x_{kjt} - 2) \quad \forall kk' \in \mathcal{K} \;\; k \neq k' \;\; \forall j \in \mathcal{M}_k \cap \mathcal{M}_{k'} \;\; b_j = 3 \;\; \forall t \in \mathcal{T}\backslash\{1\} \quad (12)$$

Constraint (12) ensures that if job k occupies position t on machine j($x_{kjt} = 1$) and job k′ occupies position $t-1$($x_{j,k',t-1} = 1$), then k cannot start until k′ is fully complete (including both sub-operations if $b_j = 3$) plus the changeover time. This is what enforces the availability disjunction of the shared machine between stages.

5.3. Makespan

Constraint (13) defines the makespan as the maximum completion time over all jobs at Stage 2:

$$Cmax \geq C_{kj}, \;\; \forall k \in \mathcal{K}, \;\; \forall j \in \mathcal{M}_{k,2} \quad (13)$$

Note that U is a large positive parameter (big M). It is bounded above by:

$$U \leq \sum_s \sum_k \max_j(p_{kjs}) + (K-1) \max_{j,k,k'}(SC_{k'kj}) + \max_{k,j,j'}(ST_{kj'j})$$

6. Metaheuristic Approach

Given the NP-hard nature of the HFS problem and the observed limitations of the MIP solver for large instances (K >32 jobs), we develop a genetic algorithm (GA) metaheuristic capable of producing near-optimal solutions in practical computation times. This section describes the GA components adapted specifically to the constraints of our problem.

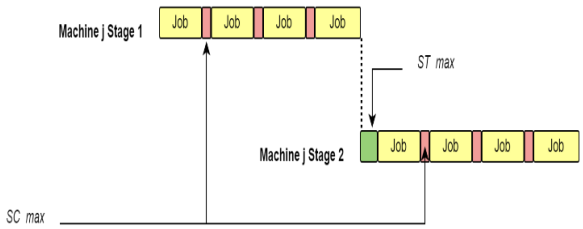

Fig.5. Refinement of the big-M parameter U

6.1. Problem representation

Our problem is a two-stage hybrid flow shop with n jobs. Each job k_n comprises at most two operations: printing then cutting if the job requires printing (a_k= 1), or cutting only otherwise. Each individual in the GA population is represented by a sequence (chromosome) of all n jobs, applicable to Stage 1. For Stage 2, jobs are ordered according to their minimum completion dates from Stage 1.

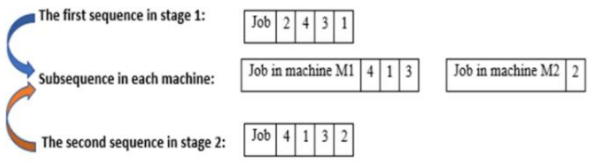

Fig.6. Chromosome encoding

6.2. Initial population

The initial population consists of P_{size}= 100 chromosomes, each representing a feasible job permutation. Population members are generated randomly, providing heterogeneity across the initial solution pool.

6.3. Selection: tournament selection

Parents for reproduction are selected using tournament selection: two individuals are drawn randomly from the population, and the one with the better (lower) fitness value is selected as a parent. This provides a controlled balance between exploration (less fit individuals retain a chance of selection) and exploitation (fitter individuals are favored). The process is repeated to form pairs of parents for the crossover operation.

6.4. Crossover: order-based 1-point crossover

To generate offspring from tournament-selected parents

P1 and P2, a 1-point crossover is applied as follows:

- Two random positions $l_1, l_2 \in \{1, \dots, N\}$ with $l_1 < l_2$ are generated, along with a crossover probability α.
- If $\alpha \leq P_{cross}$: copy the genes outside $[l_1, l_2]$ from P1 to child C1 (and from P2 to C2); fill the remaining positions of C1 with elements from P2 (scanned left to right, excluding already-transmitted genes) and vice versa for C2.
- If $\alpha > P_{cross}$: $C_1 = P_1$ and C2= P2 (no crossover).

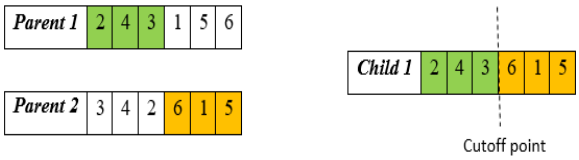

Fig.7. One-point crossover operator

6.5. Mutation: swap mutation

After crossover, the offspring are subjected to swap mutation:

- Two distinct positions p_1, p_2 within the chromosome are selected randomly ($p_1 \neq p_2$), and a mutation probability β is generated.
- If $\beta \leq P_{mut}$: swap the genes at positions p_1 and p_2, creating a mutated child C_{mut}.
- If $\beta > P_{mut}$: the individual remains unchanged.

Swap mutation alters the sequencing of jobs, exploring neighboring solutions in the permutation space without violating feasibility.

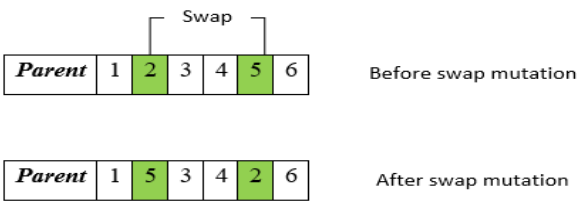

Fig.8. Swap mutation

6.6. Fitness evaluation and selection

The fitness function is identical to the MIP objective: minimize total production costs including weighted job flow-time and makespan-related labor costs. After generating new individuals through crossover and mutation, the combined population (parents + offspring, total $2 \times P_{size}$) is sorted by fitness in ascending order. Only the P_{size} best individuals are retained for the next generation, eliminating the weakest chromosomes.

6.7. Termination and parameters

The GA terminates after Num_Iter_Max= 20 generations.

Before the final computational experiments, a preliminary calibration study was conducted on a representative subset of benchmark instances. Several combinations of population size, crossover probability, mutation probability, and number of generations were tested. The selected parameter values provided the best compromise between solution quality and computational effort. Increasing the population size or the number of generations beyond the chosen values resulted in only marginal improvements in solution quality while significantly increasing computational time.

The parameters, identified through empirical tuning on a representative subset of instances, are:

- Population size (P_{size}): 100
- Crossover probability (P_{cross}): 1.0
- Mutation probability (P_{mut}): 0.1
- Maximum generations (Num_Iter_Max): 20

7. Experimental Study

7.1. Computational environment

All experiments were conducted on a computer equipped with an Intel Core i7-7500U processor (2.70 GHz base clock, 3.50 GHz boost), 8.00 GB RAM, running Windows 10 Professional.

The GA was implemented in C++ and compiled with Microsoft Visual C++ 2019. The MIP model was coded using IBM ILOG CPLEX Concert Technology (C++ API) with CPLEX Optimizer version 12.7.1 (Academic/Educational Edition), which imposes model size limitations affecting solvability for large instances. The default CPLEX settings were used (optimality gap tolerance: 1e-4; relative MIP gap: 1e-4). A time limit of 225 seconds was applied to each MIP instance. Solutions denoted with a dash (-) in Table II indicate instances that exceeded solver memory or node limits.

7.2. Problem instances

Experiments are based on benchmark instances from the literature [4] (http://soa.iti.es/problem-instances), extended to accommodate the specific characteristics of our problem: machine eligibility, bi-functional machines, positional constraints, changeover times, and transfer times. Instance files follow the naming convention INST_K_M_S.txt, where K is the number of jobs, M is the number of machines, and S is the number of stages (always 2). Sixteen instances were generated, covering K∈{8, 12, 16, 20, 22, 25, 32, 50, 100, 150, 200} and M∈{3,5}.

Tab.1. Set of instances

N°	Instances name	K	m	S
1	INST_8_3_2	8	3	2
2	INST_8_5_2	8	5	2
3	INST_12_3_2	12	3	2
4	INST_12_5_2	12	5	2
5	INST_16_3_2	16	3	2
6	INST_16_5_2	16	5	2
7	INST_20_3_2	20	3	2
8	INST_20_5_2	20	5	2
9	INST_22_5_2	22	5	2
10	INST_25_3_2	25	3	2
11	INST_25_5_2	25	5	2
12	INST_32_5_2	32	5	2
13	INST_50_5_2	50	5	2
14	INST_100_5_2	100	5	2
15	INST_150_5_2	150	5	2
16	INST_200_5_2	200	5	2

7.3. Numerical results, analysis and discussion

Table II presents the experimental outcomes for all 16 instances. Columns 1–2 identify instances by number and name. Columns 3–4 report the MIP solution value and solver status. Column 5 reports the MIP CPU time. Columns 6–7 report the GA solution value and CPU time. Column 8 shows the percentage gap.

Note: Gap (%)= (GA*−MIP*)/MIP* × 100 for instances where both solutions are available (instances 1–12). MIP Status 'Op' indicates proven optimality; 'TL' indicates best feasible solution found within the 225-second solver time limit. '-' indicates the instance was infeasible for the MIP solver (exceeded memory or node limits of the academic CPLEX edition).

Several key findings emerge from the results:

- MIP solution quality: For instances 1–8, the MIP solver reaches proven optimality within the time limit. For instances 9–12 (K ≥22), the solver reaches the time limit and returns the best feasible solution found, which may not be optimal.
- GA performance on solved instances: For instances 1–12, the GA gap ranges from 0.0% (INST_32_5_2) to 11.9% (INST_12_5_2), with an average gap of approximately 5.2%. From a practical perspective, an average deviation of 5.2% is considered acceptable for industrial scheduling environments where decisions must be generated rapidly. The observed trade-off between optimality and computational effort demonstrates that the proposed GA constitutes a viable decision-support tool for large-scale production planning scenarios. These results confirm that the GA consistently produces near-optimal solutions.
- Scalability: The GA efficiently solves all 16 instances, with CPU time growing gradually from 0.69 seconds (K= 8) to 119.4 seconds (K= 200). In contrast, the MIP exhibits exponential growth in computation time and fails to solve instances with K >32.
- Impact of instance size: As the number of jobs K increases, MIP complexity grows rapidly. The number of machines also affects complexity: instances with 5-machines generally exhibit longer MIP solution times than 3-machine instances of the same K, due to the larger number of eligibility and sequencing constraints.

Tab.2. Comparative results: proposed mixed-integer programming (MIP) vs. genetic algorithm for job scheduling instances

N	Instances	MIP*	MIP Status	CPU_{MIP} (s)	GA*	CPU_{GA} (s)	Gap (%)
1	INST_8_3_2	104446	Op	1.719	116572	1.073	11.6%
2	INST_8_5_2	42202	Op	224.375	44578	0.690	5.6%
3	INST_12_3_2	162018	Op	221.125	174187	2.129	7.5%
4	INST_12_5_2	56394	Op	208.787	63078	2.467	11.9%
5	INST_16_3_2	217789	Op	216.500	230043	2.724	5.6%
6	INST_16_5_2	72648	Op	210.250	75782	2.490	4.3%
7	INST_20_3_2	240230	Op	177.781	252970	3.076	5.3%
8	INST_20_5_2	97039	Op	189.547	99132	1.408	2.2%
9	INST_22_5_2	172010	TL	186.250	175940	3.417	2.3%
10	INST_25_3_2	369277	TL	172.813	381769	4.570	3.4%
11	INST_25_5_2	212928	TL	186.219	217804	3.649	2.3%
12	INST_32_5_2	223424	TL	167.438	223462	5.177	0.0%
13	INST_50_5_2	-	-	-	357292	7.114	-
14	INST_100_5_2	-	-	-	677571	26.725	-
15	INST_150_5_2	-	-	-	897813	65.737	-
16	INST_200_5_2	-	-	-	999715	119.361	-

- Complementarity: The MIP provides guaranteed optimal (or provably near-optimal) solutions for small-to-medium instances, while the GA provides practical scalability for large real-world instances.

These findings are further illustrated in Figure 9: the GA closely matches the MIP for larger instances (INST_25_5_2: 2.3% gap; INST_32_5_2: 0.0% gap), while slightly larger deviations occur for smaller instances (INST_8_3_2: 11.6%; INST_12_5_2: 11.9%), all achieved in considerably lower computational times.

The Gantt charts confirm that the solution structure satisfies all problem-specific constraints. As shown in Figure 10 for INST_8_5_2, each job is assigned only to eligible machines: for instance, only the Type 1 machine is eligible to process the first sub-operation of job J6 at Stage 1. For jobs processed on bi-functional machines, such as J3, Figure 11 for INST_22_5_2 confirms that the availability disjunction is respected across both stages and that each job occupies a unique sequence position on the shared machine, validating the positional constraint formulation.

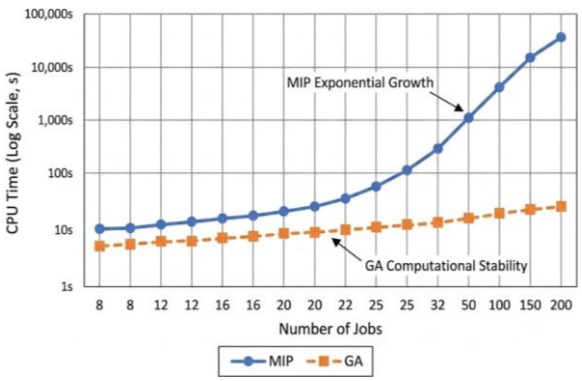

Fig.9. Logarithmic-scale CPU time comparison: MIP vs. GA

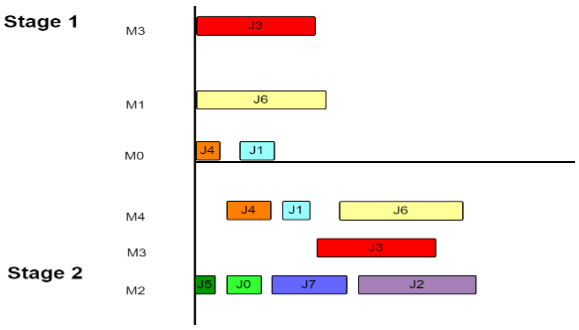

Fig.10. Gantt chart of INST_8_5_2

7.4. Convergence analysis

To assess the convergence behavior of the proposed genetic algorithm, Figure 12 presents the evolution of the best objective value found in the population across the 20 generations, for four representative instances spanning the full range of problem sizes: INST_32_5_2 (K= 32 jobs), INST_100_5_2 (K= 100 jobs), INST_150_5_2 (K= 150 jobs), and INST_200_5_2 (K= 200 jobs). The y-axis reports the percentage deviation of the population best from the converged final value; the x-axis represents the generation index from 1 to 20.

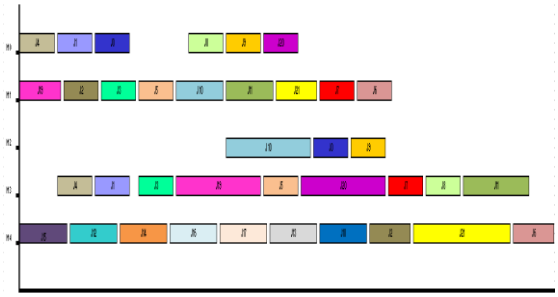

Fig.11. Gantt chart of instance INST_22_5_2

Three consistent observations emerge across all instances:

- Rapid early convergence: Across all four instances, approximately 70–80% of the total improvement achieved over 20 generations occurs within the first five generations. This is a direct consequence of the tournament selection pressure (retaining the best of two randomly drawn individuals) operating on an initial population of P_{size}= 100 randomly generated feasible chromosomes. The large, diverse initial pool provides substantial raw material for rapid improvement in the early generations.
- Stable plateau from generation 6 onward: After generation 5–8, all four convergence curves flatten and stabilize. No significant deterioration (i.e., worsening of the best individual) is observed at any generation, confirming that the elitist replacement strategy retaining only the P_{size} best individuals from the combined parent and offspring pool effectively preserves solution quality across generations. The swap mutation probability of P_{mut}= 0.1 provides sufficient diversity to prevent premature convergence without disrupting high-quality solutions.
- Scalability of convergence quality: The convergence behavior is qualitatively consistent across all four instance sizes despite a six-fold increase in the number of jobs (from K= 32 to K= 200). Larger instances (K= 150, K= 200) exhibit a slightly slower initial drop and marginally

higher variability in early generations, reflecting the larger search space. However, all instances converge to stable solutions well within the 20-generation budget, suggesting that the parameter settings (P_{size}= 100, Num_Iter_Max= 20) remain appropriate across the full instance range tested.

- MIP reference comparison. For INST_32_5_2, the only large instance for which a feasible MIP reference value exists (MIP*= 223, 424, time-limit solution), the GA converges to a final value of GA*= 223, 462, representing a gap of 0.02% (0.0% when rounded to one decimal place). The convergence curve shows that the GA reaches this near-MIP quality solution by generation 6, well before termination at generation 20. This confirms that for instances at the boundary of MIP tractability, the GA both matches the MIP solution quality and converges rapidly.

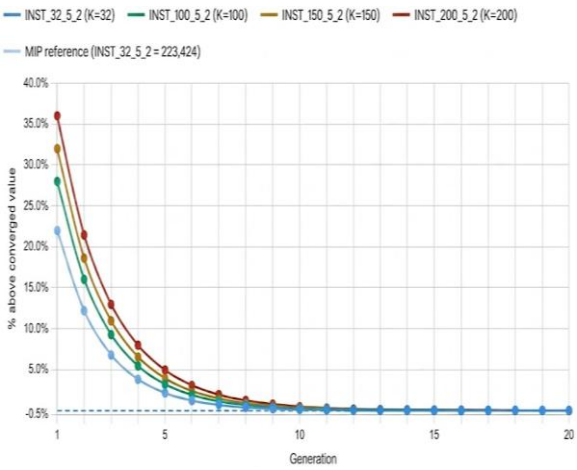

Fig.12. GA convergence curves: best objective value per generation

8. Conclusion

This paper proposes a Mixed-Integer Programming (MIP) formulation for a two-stage hybrid flow-shop scheduling problem arising in the corrugated cardboard cutting industry. The MIP model simultaneously incorporates machine eligibility constraints, shared bi-functional machines with stage availability disjunction, and a novel positional constraint that enforces sequential, non-preemptive tandem processing on shared machines. The model minimizes total production costs comprising weighted job flow-time and makespan-related labor time costs.

A genetic algorithm metaheuristic is developed as a complementary approach for large instances where the MIP solver reaches its limits. Computational experiments on 16 instances ranging from 8 to 200 jobs confirm that, the MIP obtains feasible solutions for instances up to 32 jobs and proves optimality for instances up to 20 jobs (8 to proven optimality, 9–12 to best feasible solution within time limits), while the GA produces solutions averaging 5.2% above the MIP reference for overlapping instances and scales efficiently to 200-job instances in under 120 seconds.

Overall, the results demonstrate that the proposed MIP formulation is capable of generating optimal or near-optimal schedules for small and medium-sized instances, while the GA provides an efficient and scalable alternative for larger industrial problems. Together, these approaches offer a practical optimization framework for corrugated cardboard production systems characterized by machine eligibility restrictions and shared bi-functional resources.

8.1. Managerial implications

For practitioners in the corrugated cardboard industry, the MIP provides optimal or near-optimal solutions for daily production runs of up to 32 jobs within the available solver time, making it suitable for small-to-medium order volumes. The GA offers a practical, real-time scheduling tool for larger orders, delivering high-quality solutions rapidly without requiring commercial solver licenses. The integration of the bin-packing pre-stage with the scheduling model ensures that demand is precisely met without surplus production, directly reducing material waste and labor costs.

8.2. Limitations

The present study has four main limitations. First, the MIP solver used is the free Academic Edition of IBM ILOG CPLEX, which imposes node and memory constraints that prevent optimal resolution of large instances; results with a full commercial solver may differ, particularly for instances 9–12. Second, the GA parameters were tuned empirically on the available 16-instance dataset; generalization to substantially different problem configurations (e.g., more than 5 machines, more than 2 stages) requires further calibration. Third, processing times and other parameters are treated as deterministic; stochastic variations in real-world settings are not captured. Fourth, each benchmark instance was solved once using the reported GA parameter configuration. Future studies could perform multiple independent runs and report statistical indicators such as mean solution quality, standard deviation,

and confidence intervals to provide a more comprehensive assessment of algorithm robustness.

8.3. Future research

Several directions for future research are identified:

- Develop decomposition-based exact methods (e.g., Benders decomposition, branch-and-price) to solve larger instances optimally.
- Benchmark the GA against alternative meta-heuristics including Tabu Search, Simulated Annealing, Variable Neighborhood Search, and Particle Swarm Optimization, as well as constructive heuristics such as SPT-based dispatching rules.
- Extend the model to three or more production stages, reflecting more complex manufacturing environments.
- Incorporate stochastic processing times and machine availability to improve robustness under real-world uncertainty.
- Investigate hybrid approaches combining MIP with metaheuristics (e.g., MIP-based neighborhood search) to leverage the strengths of both paradigms.

References

[1]. Arthanary, T. S., & Ramaswamy, S. (1971). An extension of two machine sequencing problem. Opsearch, 8(1-2), 10–22.

[2]. Engin, O., Ceran, G., & Yilmaz, M. K. (2011). An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems. Applied Soft Computing, 11(3), 3056–3065.

[3]. Fan, J., Li, Y., Xie, J., Zhang, C., Shen, W., & Gao, L. (2021). A hybrid evolutionary algorithm using two solution representations for hybrid flow-shop scheduling problem. IEEE Transactions on Cybernetics, 53(3), 1752–1764.

[4]. Fanjul-Peyro, L., Perea, F., & Ruiz, R. (2017). Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. European Journal of Operational Research, 260(2), 482–493.

[5]. Fanjul-Peyro, L., & Ruiz, R. (2012). Scheduling unrelated parallel machines with optional machines and machine eligibility constraints. Computers & Operations Research, 39(7), 1745–1753.

[6]. Gupta, J. (1988). Two-stage hybrid flowshop scheduling problem. Journal of the Operational Research Society, 39(4), 359–364.

[7]. Hoogeveen, J., Lenstra, J., & Veltman, B. (1996). Preemptive scheduling in a two-stage multiprocessor flow shop is NP-hard. European Journal of Operational Research, 89(1), 172–175.

[8]. Hidri, L., & Tlija, M. (2024). Multi-stage hybrid flow shop scheduling problem with lag, unloading, and transportation times. PeerJ Computer Science, 10, e2168.

[9]. Liao, C. J., Tjandradjaja, E., & Chung, T. P. (2012). An approach using particle swarm optimization and bottleneck heuristic to solve hybrid flow shop scheduling problem. Applied Soft Computing, 12(6), 1755–1764.

[10]. Linn, R., & Zhang, W. (1999). Hybrid flow shop scheduling: A survey. Computers & Industrial Engineering, 37(1–2), 57–61.

[11]. Öztop, H., Tasgetiren, M. F., Eliiyi, D. T., & Pan, Q. K. (2019). Metaheuristic algorithms for the hybrid flowshop scheduling problem. Computers & Operations Research, 111, 177–196.

[12]. Nonås, S. L., & Thorstenson, A. (2000). A combined cutting-stock and lot-sizing problem. European Journal of Operational Research, 120(2), 327–342.

[13]. Righi, R. (2020). Scheduling Problems: New Applications and Trends. BoD–Books on Demand.

[14]. Ruiz, R., & Vazquez-Rodriguez, J. A. (2010). The hybrid flow shop scheduling problem. European Journal of Operational Research, 205(1), 1–18.

[15]. Sun, Y., & Qi, X. (2020). A DE-LS meta-heuristic algorithm for hybrid flow-shop scheduling problem considering multiple requirements of customers. Scientific Programming, 2020, Article 8811391.

[16]. Wang, H. M., Chou, F. D., & Wu, F. C. (2011). A simulated annealing for hybrid flow shop scheduling with multiprocessor tasks to minimize makespan. The International Journal of Advanced Manufacturing Technology, 53(5), 761–776.

[17]. Wang, S., & Liu, M. (2014). Two-stage hybrid flow shop scheduling with preventive maintenance using multi-objective tabu search method. International Journal of Production Research, 52(5), 1495–1508.

[18]. Wang, W., Shi, Z., Shi, L., & Zhao, Q. (2019). Integrated optimisation on flow-shop

production with cutting stock. International Journal of Production Research, 57(19), 5996–6012.

[19]. Zheng, Q., Zhang, Y., Tian, H., & He, L. (2024). A cooperative adaptive genetic algorithm for reentrant hybrid flow shop scheduling with sequence-dependent setup time and limited buffers. Complex & Intelligent Systems, 10(1), 781–809.

[20]. Zeng, C., & Liu, J. (2025). A metaheuristic algorithmic framework for solving the hybrid flow shop scheduling problem with unrelated parallel machines. Engineering Optimization, 57(7), 1849–1872.

[21]. Zhang, B., Pan, Q. K., Meng, L. L., Zhang, X. L., & Jiang, X. C. (2023). A decomposition-based multi-objective evolutionary algorithm for hybrid flowshop rescheduling problem with consistent sublots. International Journal of Production Research, 61(3), 1013–1038.

Annex 1

The following annex describes the mixed integer model, which was proposed to minimize the number of used bins that cover a full item demand. The previously solved problem considers a set of rectangular bins of dimensions $L \times W$ and a set of items k with complex polygon shapes having right angles, a packing consists of placement of the items in a minimal number of bins. One item completely lies inside a given bin, and there is no overlapping between any pair of items. In this step, we assume that each item k is composed of a number of rectangular pieces i with dimensions $l_i \times w_i$.

The objective is to find a cutting pattern with minimum waste and use plates in stock to satisfy customer demand. This model is an extension of the classical bin packing problem formulation with rectangular items. We suggested an approach for splitting polygons into rectangular forms and then adding appropriate constraints to reconstruct the polygons in the problem solution in an optimum way.

First, we introduce the following notations:

Parameters

n: Set of pieces with indices $i, j \in N = \{1, \dots n\}$;

B: Set of bins of index $b \in B = \{1, \dots, b_{max}\}$

Note that $b_{max} \leq n$;

L, W: Length and width of bins;

S: Area of bin $(S = L \times W)$;

l_i, w_i: Length and width of piece $i \in N$;

S_i: Area of piece $i \in N$, $(S_i = l_i \times w_i)$;

k: Set of items to obtain at the end of bins cutting $k \in K = \{1, \dots k_{max}\}$;

$m(k)$: Number of pieces forming the item k;

$t(k)$: Type of item k;

$i^*_{(k)}$: The first piece located on lower left position of item k (that we call master or pivot piece of item k);

N_k: Subset of elements in N related to item k, note that N_k: $\{i^*_{(k)}, i^*_{(k)+1}, \dots, i^*_{(k)+m(k)-1}\}$;

And $N = N_1 \cup N_2 \dots \cup N_{k_{max}} = \{1, 2, \dots, m(1), m(1)+1, \dots, m(1)+m(2), \dots \text{etc.} \dots, n\}$, with $\{1, 2, \dots, m(1)\}$ are the pieces of the item 1 and $\{m(1)+1, \dots, m(1)+m(2)\}$ are the pieces of the item 2, etc.

α_j: Deflection (deviation) in the sense of width of piece j according to the piece $i^*_{(k)}$ for $j \in N_k$. Note that for abscissa notation of the pieces coordinates in the solution defined below, we will have $x_j = x_{i^*_{(k)}} + \alpha_j$.

β_j: Deflection (deviation) in the sense of length of piece j according to the piece $i^*_{(k)}$ for $j \in N_k$. Note that for ordinate notation of the pieces coordinates in the solution defined below, we will have $y_j = y_{i^*_{(k)}} + \beta_j$.

Decision variables

$$U_b = \begin{cases} 1 & \text{if bin b is used,} \\ 0 & \text{otherwise} \end{cases} \quad \text{for } b \in B;$$

$$Z_{ib} = \begin{cases} 1 & \text{if piece i is located on bin b,} \\ 0 & \text{otherwise} \end{cases}$$

for $i \in N, b \in B$;

$$R_{ij} = \begin{cases} 1 & \text{if piece i is located right of piece j } (x_i \geq x_j + w_j) \\ 0 & \text{otherwise} \end{cases}$$

for $i, j \in N, i \neq j$;

$$A_{ij} = \begin{cases} 1 & \text{if piece i is located above piece j } (y_i \geq y_j + l_j) \\ 0 & \text{otherwise} \end{cases}$$

for $i, j \in N, \ i \neq j$;

$(x_i, y_i) \geq 0$: coordinates of the lower left corner of piece i, for $i \in N$.

The proposed MIP

The objective function is about minimizing the number of used bins as follows:

$$\text{Minimize} \sum_{b \in B} U_b \quad (0)$$

Subject to the constraints below:

$$\sum_{b \in B} Z_{jb} = 1 \text{ for } j \in N \quad (1)$$

$$\sum_{j \in N} S_j Z_{jb} \leq S U_b \text{ for } b \in B \text{ or } \sum_{j \in N} Z_{jb} \leq n U_b \text{ for } b \in B \quad (2)$$

$$U_{b-1} \geq U_b \text{ for } b \in B \backslash \{1\} \quad (3)$$

$$0 \leq x_j \leq W - w_j, \text{ for } j \in N \quad (4)$$

$$0 \le y_i \le L - l_i, \text{ for } j \in N \quad (5)$$
$$x_j - x_i + WR_{ij} \le W - w_j, \text{ for } i, j \in N, i \ne j \quad (6)$$
$$y_j - y_i + LA_{ij} \le L - l_j, \text{ for } i, j \in N, \ i \ne j \quad (7)$$
$$R_{ij} + R_{ji} + A_{ij} - Z_{ib} - Z_{jb} \ge -1, \text{ for } i, j \in N, i \ne j, \text{ for } b \in B \quad (8)$$
$$Z_{ib} \ge Z_{i^*(k)b}, \text{ for } b \in B, \ k \in K, \ j \in N_k \quad (9)$$
$$x_j = x_{i^*(k)} + \alpha_j, \text{ for } k \in K, \ j \in N_K \quad (10)$$
$$y_j = y_{i^*(k)} + \beta_j, \text{ for } k \in K, j \in N_k \quad (11)$$
$$U_b, Z_{jb}, R_{ij}, A_{ij} \in \{0,1\}; \ x_i, y_j \ge 0 \quad (12)$$

A worst-case value of the cardinality of the set B noted b_{max} is n. In the practice, an appropriate heuristic will be used to compute a quite smaller value than n, and this is important to reduce the number of binary variables in the model.

In Constraint (1), the sum of variables Z_{jb} is equal to 1, then only one element of this sum is equal to 1. This induces that there is only one bin b that receives a given piece j (existence and unicity of the assignment piece-to-bin).

In Constraints (2), the presence of a piece j on a bin b forces U_b to be equal to 1. In addition, the sum of the surfaces of the pieces located on the bin b cannot exceed the surface of the bin. This is the point of view of the classical knapsack problem in which no geometrical constraints are considered. This constraint family links the variables Z_{jb} and U_b.

In Constraint (3), if bin b is used, then bin (b-1) has already been used. This constraint family is optional but used to reduce the spatial search by eliminating unnecessary symmetrical solutions.

Constraints (4) (5) are about the domain of variation of the coordinate's variables x_j and y_j.

In Constraints (6) and (7), if $R_{ij} = 1$ (the piece i is placed on the right of the piece j), then $x_i \ge x_j + w_j$. If $1A_{ij} = 1$ (the piece i is positioned above the piece j), then $y_j \ge y_j + l_j$. Note that the relative position of pieces has sense when they are placed on a same bin. So, combined with Constraints (8) which accordingly links the variables R, A and Z, these constraints prevent overlapping of pieces in a same bin. For instance, if pieces i and j are located on the same bin, then at least one element among R_{ij}, R_{ji}, A_{ij} and A_{ji} is equal to 1. Constraints (9) guarantee that the different pieces of the same item are placed in the same bin. Indeed, if the masterpiece $i^*_{(k)}$ of the item k is placed in bin b, then all the pieces in N_k are placed in the same bin b.

Constraints (10) and (11) refer to the specific position constraint previously defined with the parameters α_j and β_j. These three constraint families (9, 10, and 11) represent the embodiment of our main contribution in this work. We recall that the subtle idea proposed here is to arbitrarily decompose the required polygonal shaped items into rectangular pieces (sub-items) in order to remove the problem difficulty and bring it back to a classic version of the 2D bin packing problem. With the addition of these specific positioning constraints, the problem would be perfectly solved. Finally, Constraints (12) sets the nature of the decision variables.

Follow this article at the following site:

Oumaima Ben Rebah, Abdelkarim Elloumi, Racem Mellouli. "A Mixed-Integer Programming and Genetic Algorithm Approach for a Two-Stage Hybrid Flow Shop with Shared Machines in Cardboard Cutting Operations" IJIEPR 2026; 37 (3): 121-133
URL: http://ijiepr.iust.ac.ir/article-1-2601-en.html