Proceedings of the 18th Conference on Computer Science and Intelligence Systems pp. 615–625
DOI: 10.15439/2023F2715
ISSN 2300-5963 ACSIS, Vol. 35

Comparative Analysis of Exact, Heuristic and Metaheuristic Algorithms for Flexible Assembly Scheduling

Octavian Maghiar, Teodora Selea, Adrian Copie, Flavia Micota, Mircea Marin
West University of Timişoara,
Department of Computer Science,
blvd. Vasile Pârvan, 4,
300223, Timişoara, Romania
Email: {octavian.maghiar98, teodora.selea, adrian.copie, flavia.micota, mircea.marin}@e-uvt.ro

***Abstract*—Real-world manufacturing scenarios usually lead to difficult assembly scheduling problems. Besides strict precedence constraints between jobs or operations, such problems incorporate constraints related to maintenance activities on working stations (machines) and specific setup times when different operations are executed on the same machine. This paper analyzes the performance of several approaches, based on mathematical programming and on (meta)heuristics, to solve flexible assembly scheduling problems characterized by an arbitrary tree-like structure of the operation network. In this context, a specific encoding of candidate solutions and some specific perturbation operators are proposed. The encoding and the operators allow the distribution of sub(batches) of operations on several machines which leads, for some assembly scheduling problems, to a significant decrease of the makespan.**

I. Introduction

SCHEDULING represents a class of optimization problems with significant practical impact. The most studied problem is the Job Shop Scheduling problem (JSSP) [1] aiming to find an assignment of a set of inter-related jobs on a set of working resources (machines) such that some performance measures are optimized.

Assembly production scheduling is a class of scheduling problems (as identified in [2]), where a product is obtained by assembling several components (also referred to as subassemblies, subproducts or make-parts) which are a result of either some production operations or other assembling steps. Hence, the final product is the result of a particular set of operations, which are inter-related according to a hierarchical structure. Numerous factors, such as the number of operations, the number of products, the number of machines, the complexity of product structure, and the number of constraints, can increase the scheduling problem's complexity. Therefore, ongoing research in the field of scheduling is always necessary.

As is mentioned in [3] flexible assembly job-shop scheduling is less studied than job-shop scheduling. Previous work on Assembly Production Scheduling includes effective heuristics, genetic algorithms [4], or machine learning-based solutions. In [5], the authors propose a heuristic, based on the concept of the critical path, for solving the problem of large assembly production having as objective the minimization of the makespan. Another heuristic that targets the problem of assembly scheduling by taking into account the splitting of the operations in several (sub)batches was introduced in [6]. The load is distributed among the available machines using the proposed heuristic, followed by actual scheduling based on the critical path approach. In [7], the authors also include a batch splitting procedure; however, it is followed by a genetic algorithm to perform the scheduling.

The aim of this paper is to analyze several assembly scheduling strategies that are flexible enough to accommodate specific characteristics of the production process, e.g. the maintenance activities. The main contributions of the paper include:

- specific encoding of candidate solutions and specific perturbation operators which ensure the feasibility of generated schedules and allow the distribution of sub(batches) of operations on several machines, leading to a significant decrease of the makespan;
- design and implementation of a highly configurable generator of assembly scheduling test problems;
- a comparative analysis of the performance of an exact solver, a heuristic based on the critical path concept, and two metaheuristic algorithms.

The rest of the paper is organized as follows. Section II proposes a motivating manufacturing scenario, while the particularities and the formal description of the scheduling problem are presented in Section III. A short review of recent works addressing similar problems is presented in Section IV. Sections V and VI provide details on the heuristic based on the critical path and present the proposed encoding, the algorithm for generating feasible initial schedules, the decoding and evaluation procedures as well as the proposed search operators used by the metaheuristic algorithms. The data generator structure, the experimental setup and results for some test

This work was supported by project POC/163/1/3/ "Advanced computational statistics for planning and monitoring production environments" (2022-2023)

problems are presented in Section VII, while Section VIII concludes the paper.

II. A REAL-WORLD SCENARIO

A real-world problem that triggered our study is the production of flexible metal tubes for car exhaust systems. The final product is composed of subproducts that are result of production (SP) or assembly (As) operations. The manufacture process of the product is described using the bill of materials (BOM), that contains all information used to produce an item: the raw materials (RM), the (sub)products, the quantity needed to by for each manufactured product (BOM for mill-tube production - Figures 1).

A mill tube is formed from two component tubes (referred as IT and OT in Figure 1) that are covered with a metal mesh. Interlock rings are used as a fastening system. The production process can be shortly described as follows: (1) the inner tube (subproduct IT) and the outer tube (subproduct OT) are produced from a metal sheet (RM) that has to be rolled, welded and cut on a specific machine (operations O_6 and O_7); (2) the resulting tubes are transformed in bellows (subproduct E) through a stuffing process (O_5); (3) meantime: (i) the metal mesh (product F), that covers the tube built in the hydroforming process (O_4), is produced from metal wire (O_{10}, O_9, and O_8); (ii) the interlocks (subproduct C) are produced also from metal wire (O_3).

The assembling process, that has as result the product B, is executed on assembling workstations (O_2) and it consists of: (i) the bellows are wrapped with the metallic mesh; (ii) the interlocks and garnish are added; (iii) the final product is pressed; (iv) some identification information is written on the product. Then a quality assurance control (O_1) is done and the product, A, is packed.

Beside the information related to BOM other production information, like the machine characteristics on which the operations are executed, must be provided. Table I contains the characteristics (setup-time, unit processing time) of the set of machines ($\{M_1, M_2, \ldots, M_{19}\}$) that are used in mill tube production.

III. PROBLEM DESCRIPTION

The result of an assembly manufacturing process is a product that consists of many components, each of the components being either manufactured or obtained by assembling several other (sub)components (also called make-parts). The operations involved in this process are either of manufacturing type or assembling type.

The main difference between these two types of operations is that a manufacturing operation usually requires only one previously produced component (and potentially several raw materials), while an assembling operation requires several other (sub)components which should be produced by the time the assembling operation starts. In the tree-like structure of an assembly (see Figures 1 and 2) the manufacturing operations correspond to nodes having only one child, while the assembly operations correspond to nodes having at least two children.

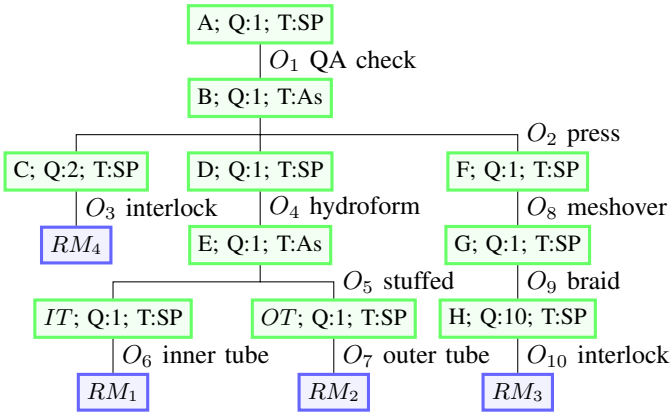


Fig. 1. Bill of materials networks for mill tube scenario. Each node contains information regarding the obtained (sub)product: name (A, B, C, ...) quantity (Q:number), type (corresponding to a production process - SP or to an assembly process - As)

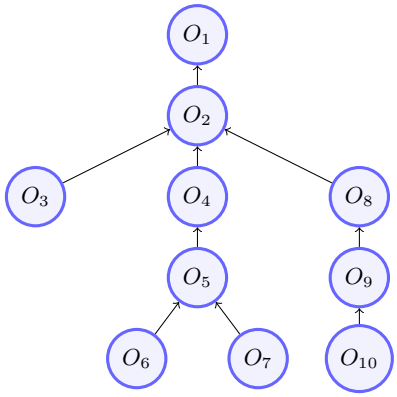


Fig. 2. Operation network for mill tube example

The Assembly Scheduling Problem (ASP) aims to schedule all types of operations on a set of machines in such a way that all make-parts are finalized before they are required for an assembly operation. The main particularity of this problem with respect to the standard Job Shop Scheduling Problem is that the precedence relation between operations corresponding to one assembly product (i.e. one job) is not a total order relation.

This allows the extension of one assembly product scheduling to several products by just considering a dummy assembly operation that would virtually group all products.

Therefore, in the following, we will consider the operations corresponding to all products (jobs) being grouped in one set of operations on which there is a partial order precedence relation.

A. Characteristics of the assembly scheduling problem

The class of assembly scheduling problems addressed in this paper is characterized by:

- all required raw materials are available, thus at the beginning of the scheduling time horizon all operations that do not have predecessors can start;

TABLE I
PRODUCTS, OPERATIONS, AND ELIGIBLE MACHINES FOR MILL TUBE EXAMPLE

Product	Operation	Machine (ID, setup time, unit production time)
A	O_1 QA check	(M_1, 600s, 2s)
B	O_2 press	(M_2, 600s, 40s), (M_3, 600s, 40s), (M_4, 500s, 40s), (M_5, 600s, 40s), (M_6, 500s, 40s)
C	O_3 interlock	(M_7, 600s, 4s), (M_{19}, 600s, 4s)
D	O_4 hydroform	(M_8, 800s, 1s), (M_9, 800s, 1s), (M_{10}, 800s, 1s), (M_{11}, 700s, 2s), (M_{12}, 700s, 2s)
E	O_5 stuffed	(M_{13}, 1000s, 5s)
IT	O_6 inner tube	(M_{14}, 900s, 2s),
OT	O_7 outer tube	(M_{15}, 900s, 2s),
F	O_8 meshoverline	(M_{16}, 900s, 12s), (M_{17}, 900s, 12s)
G	O_9 braid	(M_{18}, 900s, 30s)
H	O_{10} interlock braid	(M_7, 600s, 4s), (M_{19}, 600s, 4s)

- at a given moment, a machine can process only one operation and once started, an operation cannot be interrupted (non-preemptive);
- the operations can be grouped in batches and (producing a batch of components or products) started, it cannot be interrupted;
- once established, the size of a batch is not modified;
- for a given operation only one batch of executions can be scheduled on a machine (re-entrance is not allowed);
- there is no cost for transferring components between machines;
- no setup is required if the operations (corresponding to different batches of *same type* of (sub)components or products) are scheduled in sequence on the same machine;
- the setup times are sequence-independent and non-anticipatory (the machine is available and the operation is ready to be started);
- the maintenance activities are considered as machine-assigned operations with fixed starting time and duration.

B. Formal description

1) Notations and input data:

- Set of n operations: $\{O_j | j = \overline{1,n}\}$;
- Set of m machines: $\{M_i | i = \overline{1,m}\}$;
- t_{ji} = time to execute on machine M_i one unit of the (sub)component which is a result of operation O_j;
- s_{ji} = setup time for executing the operation O_j on machine M_i;
- Q_j = total number of executions of operation O_j derived from the quantities specified in the BOM structure;
- F = matrix of eligibility, an $n \times m$ matrix specifying which machines are eligible for each operation, i.e.

$$F_{ji} = \begin{cases} 1 & \text{if } O_j \text{ can be executed on } M_i \\ 0 & \text{if } O_j \text{ cannot be executed on } M_i \end{cases} \quad (1)$$

- R = array of n entries which can be used to identify the subset of maintenance operations, i.e.

$$\mathcal{O}_m = \{O_j | R_j > -1\},$$

in which case R_j denotes the maintenance starting time (SM_i) on the machine M_i which satisfies the condition $F_{ji} = 1$. More specifically, the elements of R are defined as:

$$R_j = \begin{cases} SM_i & \text{if } O_j \text{ is a maintenance operation for } M_i \\ -1 & \text{if } O_j \text{ is not a maintenance operation} \end{cases} \quad (2)$$

The maintenance starting times are fixed and known before the scheduling process is started.

- $\pi(j) = \{k | O_k \text{ is a child of } O_j \text{ in the operation network}\}$ describes the direct predecessor relation, i.e. $\pi(j)$ is the set of indices of operations which directly precedes O_j, thus they should be finalized before the starting moment of O_j (one operation can have several direct predecessors);
- $\sigma(j) = k$ where $k \in \pi(j)$ (one operation can have only one direct successor) and $\sigma(j) = -1$ if O_j is a final operation (its result is a final product);
- Deadline for the final operations: $D_j \in [0, T]$ for any $j \in \{1, \ldots, n\}$ such that $\sigma(j) = -1$.

2) Decision variables:

For $j = \overline{1,n}$ and $i = \overline{1,m}$:

- Assignment matrix:

$$A_{ji} = \begin{cases} 1 & \text{if } O_j \text{ is executed on } M_i \\ 0 & \text{if } O_j \text{ is not executed on } M_i. \end{cases} \quad (3)$$

- Batch splitting matrix:

$$B_{ji} = \begin{cases} b_{ji} & \text{if } O_j \text{ is executed on } M_i \\ 0 & \text{if } O_j \text{ is not executed on } M_i \end{cases} \quad (4)$$

where $b_{ji} \in \mathbf{N}$ denotes the number of consecutive executions of operation O_j on machine M_i leading to the production of a batch of b_{ji} (sub)components that are specific to operation O_j;

- $S_{ji} \in [0, T)$: starting time of O_j on M_i (T is the time horizon for the assembly production);
- $C_{ji} \in [0, T]$: completion time of O_j on M_i.

C. Constraints

- (C1) All operations are assigned on eligible machines:

$$\sum_{j=1}^{n} \sum_{i=1}^{m} (1 - F_{ji}) A_{ji} = 0 \quad (5)$$

- (C2) The assignment and batch splitting matrices are consistent (an operation assigned to a machine should be

executed at least once and the number of executions of an operation on a machine is nonzero only if the operation is assigned to that machine):

$$B_{ji} > 0, \quad \forall j, i \text{ such that } A_{ji} = 1 \tag{6}$$

$$B_{ji} = 0, S_{ji} = 0, C_{ji} = 0 \quad \forall j, i \text{ such that } A_{ji} = 0$$

- (C3) The completion time of any operation O_j is smaller than the starting time of its succeeding operation $O_{\sigma(j)}$:

$$C_{ji_1} \leq S_{\sigma(j)i_2} \tag{7}$$

$\forall j \in \{1, \ldots, n\}, \forall i_1, i_2 \in \{1, \ldots, m\}, A_{ji_1}{=}A_{\sigma(j)i_2}{=}1$

- (C4) The time intervals corresponding to operations executed on the same machine are disjoint:

$$[S_{j_1 i}, C_{j_1 i}) \cap [S_{j_2 i}, C_{j_2 i}) = \emptyset, \quad \forall j_1 \neq j_2, \tag{8}$$

$\forall i \in \{1, \ldots, m\}$ such that $A_{j_1 i} = A_{j_2 i} = 1$

- (C5) Completion time of a batch of operations:

$$C_{ji} = S_{ji} + s_{ji} + B_{ji} \cdot t_{ji} \tag{9}$$

$\forall j \in \{1, \ldots, n\}, i \in \{1, \ldots, m\},$ such that $A_{ji} = 1$

- (C6) The sum of all batch sizes corresponding to an operation equals the total quantity which should be produced by that operation:

$$\sum_{i=1, A_{ji}=1}^{m} B_{ji} = Q_j, \quad \forall j \in \{1, \ldots, n\} \tag{10}$$

- (C7) All final operations are finalized before the corresponding deadlines:

$$C_{ji} \leq D_j, \quad \forall j \in \{1, \ldots, n\}, i \in \{1, \ldots, m\} \tag{11}$$

such that $\sigma(j) = -1, A_{ji} = 1$.

- (C8) The starting time for maintenance operations is fixed:

$$S_{ji} = R_j, \forall j \in \{1, \ldots, n\}, i \in \{1, \ldots, m\} \tag{12}$$

such that $R_j \neq -1, A_{ji} = 1$.

D. Objective function

The goal of the scheduling is to minimize the makespan, C_{max}, defined as:

$$C_{max} = \max_{j \in E} \max_{i \in \{1, \ldots, m\}} \{C_{ji} \mid A_{ji} = 1\} \tag{13}$$

where $E = \{j | \sigma(j) = -1\}$ is the set of final operations. Thus, the optimization problem is:

$$\min_{A,B,S,C} C_{max} \tag{14}$$

IV. Related work

The authors of [8] addressed the problem of flexible assembly job-shop scheduling with lot streaming. The analyzed production structure is a two-stage one, characterized by the presence of an assembly stage at the end of a flexible job shop. The jobs in the first stage are processed in large batches which might lead to waiting time. The proposed approach is to split the batch into sub-batches, thus the initial scheduling problem is split into two sub-problems: batch splitting and batch scheduling. The proposed solution is based on an Artificial Bee Colony (ABC) algorithm using a population of candidates encoded based on four one-dimensional arrays: (i) an array containing the number of (sub)batches corresponding to each job; (ii) an array containing the size of each (sub)batch; (iii) an array encoding the sequence of all (sub)batches of operations; (iv) an array specifying the machine on which each (sub)batch of operations is assigned. The size of each candidate solution depends on the number of (sub)batches.

The idea of using sub-batches of unequal sizes is exploited also in our study but in the more general context when the production process involves several assembly stages.

An assembly scheduling problem involving several assembly stages which induce additional precedence constraints between jobs is approached in [3] where a genetic algorithm (GA) is used to solve it. Each element of the population evolved by the GA consists of two parts: (i) a one-dimensional array used to encode the machine assignment; (ii) a two-dimensional array for encoding the sequence of operations based on the concept of level in the operations' network. A schedule is constructed through a decoding process in which the operations are allocated to the first time slot where they fit, in the order given by the structure of the two-dimensional array.

With respect to the structure of the operations' network, the approach proposed in [3] is consistent with the particularities of the problem addressed in this paper, but it does not allow batch size control and to distribute sub-batches of operations to different machines.

In [9] a lot streaming technique, that splits jobs into sub-jobs, is applied on an assembly job shop scheduling problem (AJSP). The authors split the problem into two sub-problems (i) determine a sub-lot split; (ii) solve AJSP problem. In order to solve the sub-lot split, a genetic algorithm is used that uses a matrix, that stores on each line the number of lots for each operation and the quantity for each lot. In order to resolve the assembly problem four simple dispatching rules are used. The lots are scheduled by shortest/longest processing time, earliest due date, minimal slack time (difference between the due date and the total processing time of the operation) such that the constraints derived from the BOM are satisfied. In [10] the authors extend the work for [9] and use genetic algorithm and particle swarm optimization metaheuristics to solve also the second problem, by incorporating into the solutions the information related to the order of operation on each machine.

The approaches proposed in [9], [10] are different from the

ones discussed in this paper by the fact that the operations can be execute on a sub-set of machines not on all available machines.

V. Heuristic based on the critical path

In [5], the authors introduced the Lead Time Evaluation and Scheduling Algorithm (LETSA) that uses a critical path heuristic in order to construct a scheduling. Based on the operation networks, the LETSA algorithm creates a list of feasible operations, $\mathcal{F}$, i.e. operations for which their successors have been already scheduled. In the beginning, $\mathcal{F}$ contains the operations generating the final product(s).

The scheduling process consists of several steps (see Algorithm 1) starting with the identification of critical paths that originate in all operations of $\mathcal{F}$. For an operation O_j, a critical path is a sequence $[O_j = O_{l_1}, O_{l_2}, \ldots, O_{l_r}]$ such that $O_{l_q} = \sigma(O_{l_{q+1}})$ and $\sum_{q=1}^{r} t_{l_q *}$ is maximal. The notation t_{j*} refers to the execution time of operation O_j on the slowest machine (if the machines would have different execution times).

It should be mentioned that if a machine satisfying all conditions specified in Step S3 of Algorithm 1 is not found, then the tentative execution interval is shifted toward the left until the first availability interval is reached. In this way, a machine is always identified.

The problem addressed in [5] is slightly different from that addressed in this paper because the operations are executed in a work-centers that contain one or more *identical* machines. In our approach, the machines are not necessarily identical, so the following adaptations were done: (i) in order to calculate the critical path in Step S1 of Algorithm 1, the maximal execution times over all eligible machines are used; (ii) the list of potential machines analyzed at Step S3 of Algorithm 1 is limited to eligible machines characterized by the smallest execution times for the corresponding operation. This has been done with the aim of increasing the chance to generate schedules with smaller makespan.

VI. Metaheuristic approaches

For the metaheuristic algorithms used in the experimental analysis, the encoding and decoding procedures, as well as the search operators are described in the following.

A. Encoding and search space

According to [8], a flexible assembly scheduling with batch splitting involves several decisions: (i) in which sequence are executed the operations on each machine; (ii) on which machine is executed each sub-batch corresponding to each operation; (iii) which is the size of each sub-batch for each operation.

In order to incorporate the information required by these decisions we propose the following encoding for a candidate solution:

- a list of distinct operation indices, $L = [o_{(1)}, \ldots, o_{(n)}]$ with $o_{(l)} \in \{1, \ldots, n\}$, such that for any $1 \leq l, r \leq n$, if $o_{(l)} \in \pi(o_{(r)})$ then $l < r$;
- the batch-splitting matrix B_{ji} defined as in Eq. (4).

The order of operations given by list L corresponds to a topological order of the nodes in the operation network. L provides the order in which the operations are dispatched to machines, during the decoding step, but not necessarily the order in which they are executed in the production stage. More specifically, if $O_{o(l)}$ and $O_{o(r)}$ belong to the same branch in the operation tree-like network, then $O_{o(l)}$ will be executed before $O_{o(r)}$, but if they belong to different branches then $O_{o(l)}$ will be dispatched before $O_{o(r)}$, but not necessarily executed before $O_{o(r)}$. It should be noted that the maintenance operations are not explicitly included in the encoding, as their starting times and durations are fixed. They are taken into consideration only during the evaluation step.

An operation list, L, corresponding to the operation network described in Figure 2 is illustrated in Figure 3 where the arrows highlight the direct predecessor relations between operations. As follows from this figure, such an operation list corresponds to a topological order of the operation network. This ordering is not unique. In fact, for an oriented tree $\mathcal{T}$ with n nodes the total number of topological orderings is $n! / \prod_{v \in \mathcal{T}} \mu(\mathcal{T}_v)$ where v denotes a node, $\mathcal{T}_v$ denotes the (sub)tree rooted in the node v and $\mu(\mathcal{T}_v)$ denotes the number of nodes in the (sub)tree rooted in v. For the tree corresponding to the operation network from Figure 2 the number of distinct topological orderings is $10!/(10 \cdot 9 \cdot 4 \cdot 3 \cdot 3 \cdot 2) = 560$.

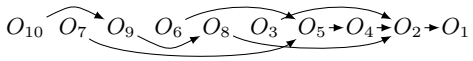


Fig. 3. An operation list corresponding to the tree-like BOM described in Figure 1 which illustrates the topological order of the operations.

Taking into account both the operation list and the batch size matrix, B (see Table II), the proposed encoding allows solving batch splitting and batch sequencing problems simultaneously and corresponds to a structure of size $n + n \cdot m$. This size is larger than that corresponding to traditional encoding based on two one-dimensional arrays of n elements (operation sequence array and machine assignment array). However this traditional encoding does not allow to distribute batches of operations on parallel machines.

B. Generation of initial candidates

The metaheuristics start from one or several initial candidates, which are further modified with the aim of improving its/their quality. The generation of initial candidates should take into account the encoding rules and the structural constraints which should be satisfied by a feasible candidate, i.e. precedence constraints induced by the operation network and constraints related to the number of components to be produced (induced by the corresponding bill of materials). The construction of the list L, corresponding to the topological

Algorithm 1 LETSA Algorithm

Input: n, m, $F_{1..n\times1..m}$, $O_{1..n}$, $t_{1..n\times1..m}$, $s_{1..n\times1..m}$
Output: $A_{1..n\times1..m}$, $S_{1..n}$, $C_{1..n}$

$\mathcal{F} \leftarrow \{O_j | \sigma(j) = -1\}$
while $\mathcal{F} \neq \emptyset$ **do**
  (S1): select $O_j \in \mathcal{F}$ s.t. O_j belongs to a *critical path*
  (S2): set the *tentative completion time* for O_j: $C_{j*} \leftarrow S_{\sigma(j)k}$, where k satisfies $A_{\sigma(j)k} = 1$
  (S3): find M_i such that $F_{ji} = 1$, M_i is available in $[C_{j*} - t_{ji}, C_{j*})$ and maximizes $C_{j*} - t_{ji}$
  (S4): update the assignment matrix and the completion and starting times: $A_{ji} \leftarrow 1$; $C_{ji} \leftarrow C_{j*}$; $S_{ji} \leftarrow C_{ji} - t_{ji}$
  (S5): update the list of feasible operations: $\mathcal{F} \leftarrow \mathcal{F} \backslash \{O_j\} \cup \pi(j)$
end while
return A, S, C

TABLE II
BATCH SIZE MATRIX (B) FOR MILL TUBE EXAMPLE

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}	M_{16}	M_{17}	M_{18}	M_{19}
O_1	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
O_2	0	2	2	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0
O_3	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	10
O_4	0	0	0	0	0	0	0	2	2	2	2	2	0	0	0	0	0	0	0
O_5	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0	0
O_6	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0	0
O_7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0	0	0
O_8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	5	0	0
O_9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	10	0
O_{10}	0	0	0	0	0	0	50	0	0	0	0	0	0	0	0	0	0	0	50

order, starts from the root node, O_1, and the frontier, $\mathcal{F}$, consisting of its directly preceding operations, $\pi(O_1)$, and select, at each step, a random element from the current frontier to be added in the front of list L. As soon as an operation is included in L, it is removed from $\mathcal{F}$ which is instead extended with all operations which directly precede the operation included in L. In this way, one of the possible topological orderings of the nodes in the operation network is generated. The matrix B can be initialized using random decisions concerning both the selection of machines from the eligible list and the choice of the batch size. The `random` function used in Algorithm 2 generates an integer value in $\{0, 1, \ldots, q\}$. A minimum batch, min_{Q_j} size of 10% of operation quantity is used in order to restrict small batch sizes.

C. Decoding and evaluation

From a candidate solution, represented by the one-dimensional array L and the two-dimensional array B, a schedule is obtained through decoding. The decoding process, described in Algorithm 3 allows also the computation of the makespan value, and it is activated any time a candidate solution has to be evaluated. The operations are scheduled based on the order specified in the operations list (L). If a part of the quantity (B_{ji}) of operation O_j is planned to be scheduled on machine M_i, the decoding procedure identifies the time interval in which it can be executed on that machine. The operation is scheduled as close as possible to the completion time of all of its preceding operations ($\pi(j)$). This means that for each machine the completion time of the last scheduled operation is dynamically updated and further used to establish the starting time of next operation to be scheduled on that machine. Thus the order defined by L is preserved on each machine.

It should be mentioned that the value of the makespan is influenced by the following decisions:

- split the batch of operations scheduled in a time interval which overlaps with a maintenance interval in such a way that the maximum possible amount operation products is executed before the maintenance activity and the remaining ones immediately after the maintenance ($opt\_maintenance$ flag is activated in Algorithm 3) ;
- no setup is required when two consecutive batches involving the same operation/product are scheduled (opt_setup flag is activated in Algorithm 3).

D. Search operators

Both trajectory and population-based metaheuristics require operators which generate new candidate solutions from existing ones. The metaheuristic algorithms involved in the analysis we conducted involve two main types of operators which generate new candidate solutions:

- *Mutation-like.* This operator generates a new candidate solution in the neighborhood of an existing one by applying a perturbation strategy.
- *Crossover-like.* A new feasible candidate solution is constructed by combining information from two existing candidate solutions.

Algorithm 2 Initialization of a feasible candidate

Input: n, m, $F_{1..n\times 1..m}$, $Q_{1..n}$
Output: $L_{1..n}$, $B_{1..n\times 1..m}$

$L \leftarrow [1]$ // index of the root operation
$\mathcal{F} \leftarrow \pi(1)$ // direct predecessors of the root operation
while $\mathcal{F} \neq \emptyset$ **do**
  $j \leftarrow \texttt{select}(\mathcal{F})$ // random selection from $\mathcal{F}$
  $L \leftarrow \texttt{prepend}(L, j)$ // add in the front of the list
  $\mathcal{F} \leftarrow \mathcal{F} \backslash \{j\} \cup \pi(j)$ // update the frontier
end while
for $j \leftarrow 1..n$ **do**
  $q \leftarrow Q_j$; $i \leftarrow 1$
  while $(i \leq m)$ and $(q > 0)$ **do**
    if $F_{ji} = 1$ **then**
      $B_{ji} \leftarrow \texttt{max}(min_{Q_j}, \texttt{random}(0, q))$
      $q \leftarrow q - B_{ji}$
      $i_* \leftarrow i$
    end if
    $i \leftarrow i + 1$
  end while
  if $q > 0$ **then**
    $B_{ji_*} \leftarrow B_{ji_*} + q$
  end if
end for
return L, B

The way of action of these operators is shortly presented in the following, the guiding idea being to preserve the feasibility.

1) Mutation : Let us consider a candidate solution encoded by (L, B). A new candidate is generated in the neighbourhood of (L, B) by following the steps:

- randomly select $o_{(q)} \in L$;
- search for the largest $l \in \{1, \ldots, n\}$ such that $o_{(l)} \in \pi(o_{(q)})$ and for the smallest $r \in \{1, \ldots, n\}$ such that $o_{(q)} \in \pi(o_{(r)})$; if operation $o_{(q)}$ does not have predecessors then $l = 1$ and if $o_{(q)}$ does not have a successor then $r = n$;
- randomly select an insertion position $p \in \{l, l+1, \ldots, r\}$ and insert the element $o_{(q)}$ on position p in L;
- if there are several eligible machines for operation $o_{(q)}$, then randomly select two of them and move the batch (totally or partially) from the source to the destination machine; the decision to perturb B is taken with a given probability.

It is easy to observe that all operations in the sublist of L delimited by l and r ($L_{l..r}$) do not contain operations that are in a precedence relation with $o_{(q)}$, meaning that the perturbed candidate solution is still feasible.

It should be also mentioned that if the list of eligible machines for operation $o_{(q)}$ and the lists of machines corresponding to the operations in the sublist $L_{l..r}$ are disjoint, then any insertion of $o_{(q)}$ in another position of the sublist will have no impact on the makespan of the schedule obtained by decoding the candidate solution. On the other hand, if $o_{(q)}$ is one of the final assembly operations, then $l = r$ and the perturbation will be ineffective.

Algorithm 3 Decoding and evaluation of a solution

Input: $L_{1..n}$, $B_{1..n\times 1..m}$, $R_{1..n}$,D
Output: $S_{1..n}, C_{1..n}, C_{max}$ // start time, completion time

$S_{1..n} \leftarrow D$; $C_{1..n} \leftarrow 0$
$M^{FT}[1..m] \leftarrow 0$ // current makespan per machine
for $h \leftarrow 1..n$ **do**
  $j \leftarrow L_h$ // schedule operation O_j
  for $i \in \{1, \ldots, m\}$ such that $B_{ji} > 0$ **do**
    if $opt\_setup = True$ and O_j is identical with the last operation scheduled on M_i **then**
      $st \leftarrow 0$
    else
      $st \leftarrow s_{ij}$
    end if
    $ST \leftarrow \max(\max\{C_k | k \in \pi(j)\}, M_i^{FT})$
    $ET \leftarrow B_{ji} \cdot t_{ji}$
    if there exists k such that $R_k > 0$ and $[ST, ST + st + ET) \cap [R_k, R_k + t_{ki}) \neq \emptyset$ **then**
      if $opt\_maintenance = True$ **then**
        $C_j \leftarrow \max\{ST + st + ET + t_{ki}, C_j\}$
      else
        $ST \leftarrow R_k + t_{ki}$
        $C_j \leftarrow \max\{ST + st + ET, C_j\}$
      end if
    else
      $C_j \leftarrow \max\{ST + st + ET, C_j\}$
    end if
    $S_j \leftarrow \min\{ST, S_j\}$
  end for
end for
$C_{max} \leftarrow \max_{j=\overline{1,n}} C_j - \min_{j=\overline{1,n}} S_j$
return C_{max}

Let us consider the L-part encoding corresponding to the example illustrated in Figure 3: $L = [10, 7, 9, 6, 8, 3, 5, 4, 2, 1]$. If the selected element is $o_{(5)} = 8$ then $l = 4$ and $r = 8$, thus there are four alternative insertion positions for the value 8. However, as the machines M_{16} and M_{17} are not used by the other operations, all of these perturbations will be without impact on the makespan.

2) Crossover : The crossover-like perturbation aims to generate a new feasible candidate solution by using information from two existing ones, usually called parents. The feasibility is preserved if the idea of precedence operation crossover is used: some elements are taken from one parent, while from the other parent is used to order in which the remaining elements are placed. Let us consider $L = [o_{(1)}, \ldots, o_{(n)}]$ and $L' = [o'_{(1)}, \ldots, o'_{(n)}]$ and the corresponding batch size matrices B and B'. The steps followed to construct a new feasible candidate, (L^{new}, B^{new}), are:

- randomly select $l < r$ from $\{1, \ldots, n\}$;
- transfer from L to L^{new} all elements with indices be-

tween l and r;
- scan the elements of L' and for each element $o'_{(l)}$ which is not yet in L^{new} append it to
  - a prefix list L_P, if $o'_{(l)}$ is the predecessor (not necessarily direct) of at least one element of L^{new};
  - a suffix list L_S if there is no element in L^{new} such that $o'_{(l)}$ is its predecessor.
- the new candidate solution is obtained by joining L_P, L^{new} and L_S;
- the rows of B^{new} corresponding to the operations taken in the first step from L will be identical to the corresponding rows from B, while the other rows are taken from B'.

Let us consider, in the case of mill tube example, two candidate solutions: $L = [6, 10, 7, 5, 9, 4, 3, 8, 2, 1]$ and $L' = [10, 7, 9, 6, 8, 3, 5, 4, 2, 1]$ In the case when $l = 4$ and $r = 9$, applying the crossover operator leads to $L^{new} = [10, 7, 6, 5, 9, 4, 3, 8, 2, 1]$ which could be also generated from L by mutation (insertion of the first element on the third position). However, the crossover-like perturbation allows the generation of new candidates which would not be generated through one-step mutation. For instance, for the same L and L' from the above example, if $l = 2$ and $r = 5$ one obtains $L^{new} = [6, 10, 7, 5, 9, 8, 3, 4, 2, 1]$ which could not be obtained by mutation neither from L nor from L'. Ensuring the feasibility of the crossover result requires checking the precedence constraints, thus this operator induces a computational cost larger than that of mutation.

VII. Experimental analysis

A. Data generator

Despite the increasing interest in ASP, there are no benchmarks for the general assembly scheduling problem. Some of the works addressing the flexible assembly scheduling problem [8], [11] use benchmarks that have been originally proposed for flexible job-shop scheduling problems, e.g. Kacem benchmark [12] and Fattahi benchmark [13]. Recently, Talens et al. [14] proposed two extensive sets of instances for the 2-stage assembly scheduling problem, one corresponding to the case of one assembly machine and the other one corresponding to the case of several assembly machines.

Since the problems included in these benchmarks do not capture all characteristics of flexible assembly scheduling, we designed a problem generator that allows the generation of a large variety of assembly scheduling problems characterized by different operation networks and different sets of eligible machines.

The problem generator has a simple architecture, as illustrated in Figure 4. The generator uses:

- a *pool of products* that represent the entities included in the bill of materials corresponding to a client order. Currently, the generator uses a list of 200 of fictitious entities (which can be interpreted as products, components or make-parts), characterized by randomly generated names;

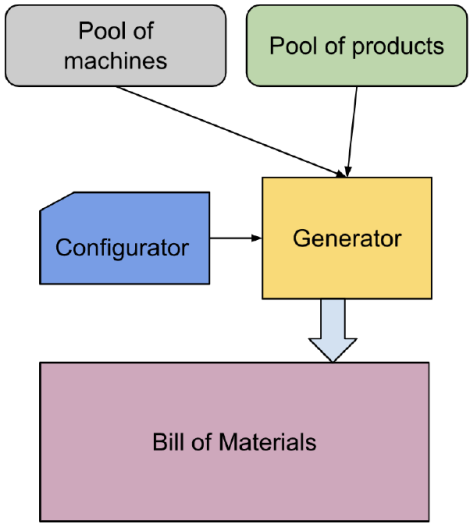


Fig. 4. Architecture of the assembly scheduling problems generator

- a *pool of machines* that can be used to execute the operations. Currently, the generator uses a list of 20 available machines.

The characteristics of a problem instance are specified in a configuration file that contains the values of the parameters describing the BOM structure.

In order to provide a high level of flexibility to the generated BOMs, the configurator module can be loaded with a multitude of parameters such as the number of levels in BOM, meaning the number of operations needed to be completed to create the final product, a maximum number of children for each node, meaning that a certain product is the result of assembling one or more simpler parts. One can specify the size of the product pool that is considered for this generator and the time horizon for scheduling. Also, the quantity of the final products can be provided. Another important characteristic that is supported is related to the maintenance applied to the machines involved in the product fabrication. The generator supports the setting of maintenance time intervals, the *Overall Equipment Effectivness* (OEE) for the machines and also one can specify the setup time for the machines if there is a need to change its purpose to support the processing of another product. The values can be generated using either an uniform distribution over the range of feasible values (the default case) or a truncated normal distribution.

The generator produces a JSON file containing the description of a tree-like structure corresponding to a BOM with a specified depth and a variable number of children, each node in the tree representing a specific product and including the generated set of eligible machines. The root node corresponds to the final product, while the children correspond to components that eventually will compose the final product.

B. Test problems

The experimental analysis is based on several problems instances/sets that have been generated such that various characteristics of the problem are emphasized:

- *A real-world case study:* the mill tube problem incorporating three orders of sizes 800, 320, and 160, each order requiring the production of the specified number of tubes according to the BOM described in Figure 1.
- *A deep BOM structure:* it is characterized by branches of up to 50 nodes in the operation network and a branching factor of 2 (for an operation there are at most two operations that directly precede it). The problem instance used in experiments contains 437 nodes.
- *A set of BOM structures with a variable number of operations:* it contains 15 problem instances corresponding to operation networks with a specified depth of 3 and a branching factor ranging between 2 and 16. The number of operations varies between 7 and 273. The quantity corresponding to each component (make-part) in the BOM is set to 10 (in this way each component on to the third level has a quantity equal to 1000). This set is characterized by rather wide structures and it was used to analyze the influence of the number of operations on the performance of exact, heuristic, and metaheuristic methods.

C. Methods and control parameters

The methods involved in the comparative analysis are:

- An *exact solver* (CPLEX) used to solve the problem described in section III-B. The solver is executed using 32 threads and CPLEX control parameters have been used with their default values. It should be mentioned that besides the problem described in section III-B which corresponds to the case when batch splitting is applied, a simplified version that does not require the B matrix as decision variables is also analyzed (referred to as standard in Tables III and IV).
- An implementation of the *LETSA heuristic* as it is described in section V. The standard variant does not use batch splitting while the variant with batch splitting (BS) applies a load-balancing strategy in order to distribute the operations per eligible machines. It should be mentioned that LETSA heuristic starts the construction of the schedule from the final operation, while the other heuristics start from the leaf operations in the operation network. Since the maintenance intervals influence the solution quality, the solution generated by LETSA is shifted such that the leaf operations are scheduled closer to the start time. LETSA does not require control parameters.
- A *Tabu Search (TS)* algorithm based on the mutation-like perturbation described in Section VI-D1. It should be mentioned that when a new candidate (neighborhood element) is constructed, the batch-size matrix (B) is perturbed with a probability equal to 0.15. The neighborhood size is set to 100 and the tabu-list size to 25. If the current candidate solution is not improved in the last 50 iterations then it is replaced with an element selected from the current neighborhood. The implementation uses 32 parallel search processes.
- A *Genetic Algorithm (GA)* which evolves a population of 100 elements by applying the same perturbation as in TS (with the same mutation probability for the batch-size matrix), the crossover operator described in Section VI-D2 and proportional selection. The crossover operator does not use control parameters.

It should be mentioned that the TS and GA implementations are adapted starting from the JSSP implementation available at [1].

All experiments have been run on a machine with 64 vCPUs and 256 GB RAM. The execution timeout was set to 1 hour for CPLEX and 3 minutes for LETSA, TS and GA. For TS and GA the reported makespan is the average value of 30 independent runs.

D. Results and discussion

The results obtained for the mill tube study case are presented in Table III which contains the makespan values (in hours) corresponding to various methods. Since the BOM structure is rather simple and most of the operations use distinct machines the optimal solution is obtained by CPLEX, TS and GA without batch splitting (BS). On the other hand, since the number of products and make-parts corresponding to all orders is rather large, the batch-splitting strategy improves the makespan by around 40% in the case of CPLEX, TS and GA, and around 20% in the case of LETSA.

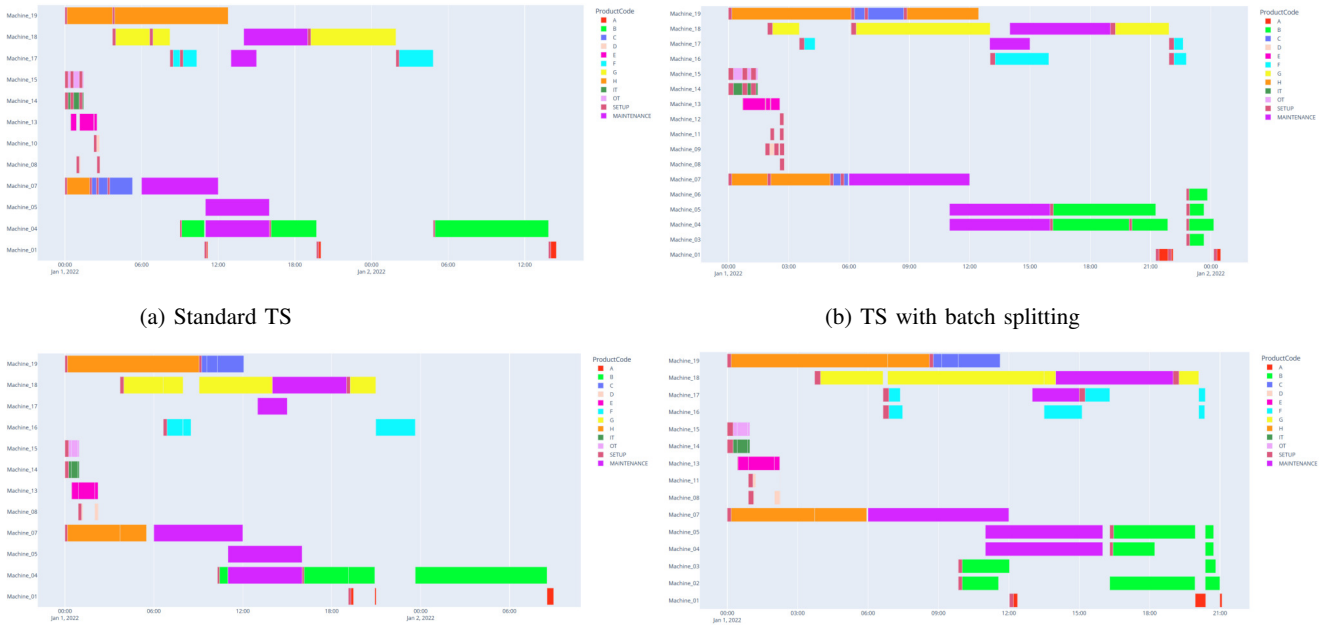
The control of the maintenance intervals and setup times corresponding to successive execution of batches of identical operations can further improve the makespan but not significantly. The Gantt charts (Figure 5) illustrate the fact that the benefit of BS is influenced by the number of machines on which the batch of operations can be distributed.

The positive impact of batch splitting is illustrated also in the case of the operation networks with deep structure (see Table IV), particularly in the case of TS. The poorer behavior of GA can be explained by the fact that the crossover operator is more time expensive than mutation and, because of the imposed limit of time, the exploration of the search space is limited. It should be noted that in this case, CPLEX could not provide a solution in the allocated amount of time (one hour).

TABLE III
MAKESPAN VALUES (IN HOURS) FOR THE TUBE MILL PROBLEM. VARIANTS OF THE ALGORITHMS: STANDARD, BS (WITH BATCH SPLITTING), MSC (WITH CONTROL ON THE MAINTENANCE INTERVALS AND SETUP TIMES)

	CPLEX	LETSA	TS	GA
standard	38.47	40.53	38.47 $\pm$ 0	37.42 $\pm$ 0.01
BS	21.87	31.23	22.07$\pm$ 0	24.20 $\pm$ 0.75
MSC	38.30	35.22	38.30 $\pm$ 0	32.98 $\pm$ 0
BS + MSC	21.69	30.37	**21.31 $\pm$ 0.15**	21.71 $\pm$ 0.52

[1] https://job-shop-schedule-problem.readthedocs.io/en/stable/index.html



(a) Standard TS

(b) TS with batch splitting

(c) TS with MSC

(d) TS with batch splitting and MSC

Fig. 5. Gantt charts for the Tube mill problem generated using Tabu Search

TABLE IV
MAKESPAN VALUES (IN DAYS) FOR THE PROBLEM INSTANCE WITH DEEP STRUCTURE. VARIANTS OF THE ALGORITHMS: STANDARD, BS (WITH BATCH SPLITTING), MSC (WITH CONTROL ON THE MAINTENANCE INTERVALS AND SETUP TIMES)

	LETSA	TS	GA
standard	36.67	36.48 ± 0.87	37.41 ± 0
BS	34.06	27.28 ± 0.46	36.43 ± 0.11
MSC	35.22	35.47 ± 0.23	36.65 ± 0.19
BS + MSC	32.51	**25.79** ± **0.35**	35.17 ± 0.20

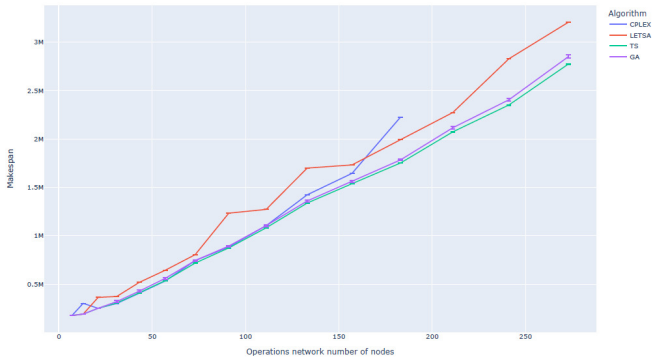


Fig. 6. Scalability results for standard variants of CPLEX, LETSA, TS, and GA for the 15 BOM structures

To analyze the scalability of the investigated methods we used the set of BOM structures with variable number of operations in the context when a set of 10 machines are available and an operation can be executed on at most 5 machines with different or similar characteristics.

From Figure 6 it can be observed that the exact solver was able to find solutions for problems having up to 183 operations in a time interval of one hour. TS and GA heuristics outperform LETSA heuristic, and TS is slightly better than GA.

VIII. CONCLUSIONS AND FURTHER WORK

The particularities of the addressed assembly scheduling problems required the incorporation of some specific decision variables and constraints. Most mathematical programming models used in flexible job-shop scheduling include binary variables which encode the order between any two operations scheduled on the same machine that leads to $n^2 \cdot m$ binary variables (in the case of n operations and m machines). The proposed mathematical programming model avoids the usage of such a large number of binary variables, but it uses instead the starting and completion time values to enforce the precedence constraints.

The proposed problem description and the candidate solution encoding allow specifying the distribution of (sub)batches of identical operations over several machines which led to a significant reduction in the makespan, particularly in the case of large orders.

The strategy that takes into account the maintenance time intervals and removes the unnecessary setup activities proved also to be beneficial but to a lesser extent.

As the operator inspired by the precedence preserving order-based crossover proved to be computationally intensive, we will further investigate other operators which preserve the feasibility of candidate solutions. Since the critical path heuristic

incorporated in LETSA generates relatively good solutions in a fraction of the time required by the metaheuristic algorithms, a further step would be to consider the sub-optimal solution produced by LETSA among the initial candidate solutions for the metaheuristic algorithms.

We also plan to conduct a systematic scalability analysis based on sets of test problems including various typologies of operation networks and interactions between operations with respect to the lists of eligible machines, as reflected in the corresponding conjunctive graphs of the scheduling problem.

REFERENCES

[1] H. Xiong, S. Shi, D. Ren, and J. Hu, "A survey of job shop scheduling problem: The types and models," *Computers & Operations Research*, vol. 142, 2022. doi: https://doi.org/10.1016/j.cor.2022.105731

[2] T. Morton and D. W. Pentico, *Heuristic scheduling systems: with applications to production systems and project management*. John Wiley & Sons, 1993, vol. 3.

[3] W. Lin, Q. Deng, W. Han, G. Gong, and K. Li, "An effective algorithm for flexible assembly job-shop scheduling with tight job constraints," *Int. Trans. Oper. Res.*, vol. 29, no. 1, pp. 496–525, 2022. doi: 10.1111/itor.12767. [Online]. Available: https://doi.org/10.1111/itor.12767

[4] Q. Liu, X. Li, H. Liu, and Z. Guo, "Multi-objective metaheuristics for discrete optimization problems: A review of the state-of-the-art," *Applied Soft Computing*, vol. 93, p. 106382, 2020.

[5] A. Agrawal, G. Harhalakis, I. Minis, and R. Nagi, "'just-in-time'production of large assemblies," *IIE transactions*, vol. 28, no. 8, pp. 653–667, 1996.

[6] S.-G. Dastidar and R. Nagi, "Batch splitting in an assembly scheduling environment," *International Journal of Production Economics*, vol. 105, no. 2, pp. 372–384, 2007.

[7] H.-y. Wang, Y.-w. Zhao, X.-l. Xu, and W.-L. Wang, "A batch splitting job shop scheduling problem with bounded batch sizes under multiple-resource constraints using genetic algorithm," in *2008 IEEE Conference on Cybernetics and Intelligent Systems*. IEEE, 2008, pp. 220–225.

[8] X. Li, J. Lu, C. Yang, and J. Wang, "Research of flexible assembly job-shop batch–scheduling problem based on improved artificial bee colony," *Frontiers in Bioengineering and Biotechnology*, vol. 10, p. 909548, 2022.

[9] F. Chan, T. Wong, and L. Chan, "Lot streaming for product assembly in job shop environment," *Robotics and Computer-Integrated Manufacturing*, vol. 24, no. 3, pp. 321–331, 2008. doi: https://doi.org/10.1016/j.rcim.2007.01.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0736584507000063

[10] ——, "The application of lot streaming to assembly job shop under resource constraints," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 14 852–14 857, 2008. doi: https://doi.org/10.3182/20080706-5-KR-1001.02514 17th IFAC World Congress. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667016413790

[11] A. Maoudj, B. Bouzouia, A. Hentout, A. Kouider, and R. Toumi, "Distributed multi-agent scheduling and control system for robotic flexible assembly cells," *J. Intell. Manuf.*, vol. 30, no. 4, pp. 1629–1644, 2019. doi: 10.1007/s10845-017-1345-z. [Online]. Available: https://doi.org/10.1007/s10845-017-1345-z

[12] I. Kacem, S. Hammadi, and P. Borne, "Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 32, no. 1, pp. 1–13, 2002. doi: 10.1109/TSMCC.2002.1009117

[13] P. Fattahi, M. S. Mehrabad, and F. Jolai, "Mathematical modeling and heuristic approaches to flexible job shop scheduling problems," *J Intell Manuf*, vol. 18, p. 331–342, 2007. doi: https://doi.org/10.1007/s10845-007-0026-8

[14] C. Talens, P. Perez-Gonzalez, V. Fernandez-Viagas, and J. M. Framiñan, "New hard benchmark for the 2-stage multi-machine assembly scheduling problem: Design and computational evaluation," *Comput. Ind. Eng.*, vol. 158, p. 107364, 2021. doi: 10.1016/j.cie.2021.107364. [Online]. Available: https://doi.org/10.1016/j.cie.2021.107364