An Efficient MILP-Based Decomposition Strategy for Solving Large-Scale Scheduling Problems

Natalia P. Basán[1], Mariana E. Cóccola[2], Carlos A. Méndez[3]

INTEC (UNL –CONICET), Güemes 3450, Santa Fe, 3000, Argentina
[1]nbasan@intec.unl.edu.ar
[2]marcoccola@santafe-conicet.gov.ar
[3]cmendez@intec.unl.edu.ar

Abstract. This paper presents a MILP-based decomposition algorithm for solving large-scale scheduling problems with assembly operations in flexible flow shop environments. First, a rigorous mixed-integer linear (MILP) formulation based on the general precedence notion is developed for the problem under study. Then, the MILP model is embedded within a decomposition algorithm in order to accelerate the resolution of large-size industrial problems. The proposed solution approach is tested on several examples derived from a real-world case study arising in a shipbuilding company.

Keywords: flexible flow shop, scheduling problem, assembly operations, MILP model, decomposition strategy.

1 Introduction

The flexible flow shop scheduling problem (FFSP) is a generalization of the classical flow shop problem (FSP), wherein all products follow the same flow processing line but all of them may not visit all processing stages. When some stage on the line performs an assembly operation, the problem is known as FFSP-A. Generally, the objective is to minimize the completion time of all products (makespan). This type of scheduling problem appears in many industrial applications such as automotive industry, paint companies, and shipbuilding industry, between others.

The FFSP-A is strongly NP-hard [1]. Consequently, real-world industrial problems lead to intractable model sizes when rigorous mathematical formulations are used. To overcome this drawback, this works presents a decomposition algorithm that allows finding high quality solutions with low computational effort even for large-size instances. The decomposition approach first obtains a good schedule, by using an insertion method, and then improves it by executing partial rescheduling actions. All decisions in the iterative procedure are taken by solving a MILP model featuring a reduced search space. Specifically, the mathematical formulation used in this paper was developed applying the general precedence notion, but other alternative approaches can be considered too. The applicability and efficiency of the solution strategy is tested by solving a challenging real-world problem.

adfa, p. 1, 2011.

2 Problem statement

The FFSP-A problem consists of a set of products $i \in I$ ($i = 1,2,\ldots,|I|$) processed through several consecutive operation stages $s \in S$ ($s = 1,2,\ldots,|S|$) with parallel identical units $k \in K_s$ working in parallel at each stage s. The subset S_i identifies all stages processing product i and the subsets $(S^a, S^a_{seq}) \subset S$ contains all stages performing assembly operations.

The final products obtained on the line are identified by subset I^f ($I^f \subset I$) and are created by assembling other intermediate products $i \in I^a$. The subset SA_i contains all subassemblies of product i. Note that $I = (I^f \cup I^a)$.

Either the non-intermediate storage (NIS) policy or the unlimited intermediate storage (UIS) policy between stages can be adopted. When a NIS strategy is used, each processing unit becomes intermediate storage if its processing has finished and the next step is not available yet.

3 Mathematical formulation

The problem constraints can be mathematically modeled using any of the continuous-time formulations that have been published in the literature for the short-term scheduling of multistage batch plants [2]. Particularly, the MILP model developed in this work and presented follow is based on the general precedence notion. It is worth to remark that some changes have been incorporated to the original proposal in order to consider the assembly operations.

This formulation generalizes the precedence concept and reduces by more than half the number of sequencing variables used by the model. This reduction is obtained by defining the sequencing binary variable $W_{ii's}$ just for all pair of products (i, i') with $i < i'$, processed at stage s. On the other hand, Y_{ik} is the assignment binary variable valuing 1 if task i is processed at unit k.

3.1 Nomenclature

Indices.

i	product order
k	processing unit
s	processing stage

Sets.

I	set of product orders
K	set of processing units
S	set of processing stages
I^f	set of final products
I^{sa}	set of subassemblies or parts

SA_i	set of subassemblies of each block $i \in I^f$
S^f	available processing stages s to process final product $i \in I^f$
S^{sa}	available processing stages s to process subassemblies $i \in I^{sa}$
S^a	available processing stages s to assemble subassemblies $i \in I^{sa}$
K_s	set of parallel processing units$k$ in processing stage s

Parameters.

pt_{is}	processing time of product order i at stage s
M	big constant in big-M constraints
$iter$	number of product order to be inserted at each iteration
$active_i$	indicating if product order i is active in the current iteration
sY_{ik}	saving assignment decisions
$sW_{ii's}$	saving sequencing decisions
$BestSol$	saving the best solution found in the improvement stage
$CurrentSol$	saving the last solution found by the improvement stage

Continuous variables.

Ts_{is}	start time of product i in processing stage s
Tf_{is}	final time of product i in processing stage s
MK	makespan

Binary variables.

$W_{ii's}$	defining if product i is processed before of product i' in processing stage s
Y_{ik}	defining if product order i is processed in processing unit k

3.2 Constraints

The general precedence formulation for the problem under study includes the following sets of constraints.

As shown Eq. (1), the main goal is to minimize the total time required to obtain the final products.

$$minimize\ MK \tag{1}$$

$$\sum_{k \in K_s} Y_{ik} = 1 \qquad \forall\, i \in I, s \in S_i \tag{2}$$

$$Tf_{is} \geq Ts_{is} + tp_{is} \qquad \forall\, i \in I, s \in S_i \tag{3}$$

$$Ts_{\mathrm{is}} = Tf_{i(\mathrm{s}-1)} \qquad \forall\, i \in I, (s, (s-1)) \in S_i : s > 1 \tag{4}$$

$$Ts_{\mathrm{is}} \geq Tf_{i(\mathrm{s}-1)} \qquad \forall\, i \in I, (s, (s-1)) \in S_i : s > 1 \tag{5}$$

$$Ts_{\mathrm{is}} \geq Tf_{i'(\mathrm{s}-1)} \qquad \forall\, i \in I^f, i' \in SA_i, s \in (S^a \cap S_i,)\ (s-1) \in S_{i'} \tag{6}$$

$$Ts_{i's} \geq Tf_{is} \qquad \forall (i,i') \in I, s \in S^a_{seq}: Seq_i < Seq_{i'} \tag{7}$$

$$Ts_{i's} \geq Tf_{is} - M(1 - W_{ii's}) - M(2 - Y_{ik} - Y_{i'k})$$

$$\forall (i,i') \in I, s \in (S_i \cap S_{i'}), k \in K_s: i < i' \tag{8}$$

$$Ts_{is} \geq Tf_{i's} - MW_{ii's} - M(2 - Y_{ik} - Y_{i'k})$$

$$\forall (i,i') \in I, s \in (S_i \cap S_{i'}), k \in K_s: i < i' \tag{9}$$

$$MK \geq Tf_{is} \qquad \forall i \in I^f, s \in S_i: s = |S| \tag{100}$$

Eq. (2) defines the allocation constraint. Binary variable Y_{ik} takes 1 as value when product i is processed in unit k; otherwise, it is set to zero. Eq. (3) computes the ending time Tf_{is} of product i at stage s as its starting time Ts_{is} plus the associated processing time pt_{is}.The storage police between two consecutive stages is represented by Eq. (4) for NIS or Eq. (5) for UIS. Constraint (6) determines that the assembly of a product i in stage $s \in S^a$, with $s \epsilon S_i$, must begin after its associated sub-assemblies $i' \in SA_i$ have completed their processing in the previous stage. On the other hand, the assembly sequence in specific stages is determined by Eq. (7). This sequencing constraint forces the starting time of product order i' to be greater than the completion time of any product order i that is before at specific sequence ($Seq_i < Seq_{i'}$). Eqs. (8) and (9) define the sequencing constraints on a same unit k. Binary variable $W_{ii's}$ is the general precedence variable in stage s. Finally, Eq. (10) states a lower bound for the variable MK to be minimized.

4 The MILP-based decomposition algorithm

The computational efficiency of the full space approach presented in the above section or any other rigorous formulation is rapidly deteriorated when increasing the problem size. For industrial applications, the solvers report solutions with a high gap after several CPU hours. This weakness can be overcome by solving the mathematical model several times but considering a reduced search space at each iteration. Even though this solution strategy does not guarantee the optimality of the solution found, it allows reporting practical solutions with reasonable computational time.

The decomposition method presented here is based on the strategy of first obtaining an initial solution (constructive stage) and then, gradually enhance it by applying several rescheduling iterations (improvement stage). The general structure of the algorithm is given in Fig. 1. Note that both algorithmic stages have as core the general precedence MILP model presented previously. At this point, it is worth mentioning that other alternative mathematical formulations [3] may also be easily adapted to the proposed decomposition strategy.

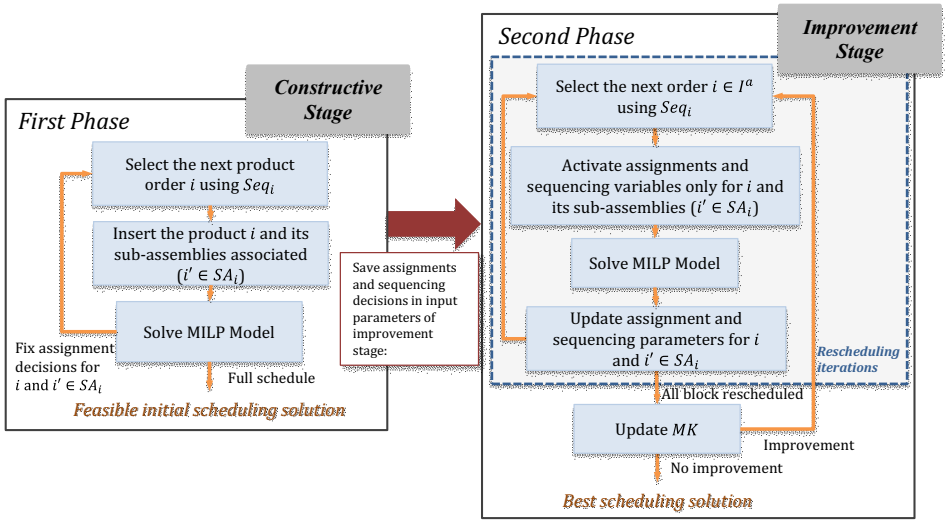

Fig. 1. Overview of the iterative MILP-based algorithm.

4.1 First phase: Constructive step

The first phase of decomposition algorithm aims at generating an initial full schedule with low computational effort. The constructive method is based on the insertion technique presented by Kopanos et al. [4] for solving large-scale pharmaceutical scheduling problems. These authors propose to insert (schedule) the products one-by-one in an iterative mode. Since the FFSP-A problem includes assembly operations, a product $i \in I^f$ and its sub-assemblies $i' \in SA_i$ must be inserted and scheduled at each iteration. Every time the MILP model (1)-(8) is solved, the binary variables Y_{ik} and $W_{ii's}$ for the new products scheduled are fixed at their optimal values. The pseudo-code for the constructive step is given in Fig. 2. The scalar $iter$ identifies the number of final product $i \in I^b$ to be inserted in the current iteration while the boolean parameter $active_i$ is true when the product i (final product or subassembly) is selected for scheduling.

One key point to consider in the constructive stage is to define the order in which the products will be inserted. The insertion criterion should be determined according to the problem features [5] [6], for example, it can follow the lexicographic order or be based on a specific sequence. The aim should always be to find a good initial scheduling solution in a short computational time.

The constructive stage procedure ends when all products have been scheduled. Next, the initial solution is sent to the next algorithmic phase (improvement stage) using the parameters sY_{ik} and $sW_{ii's}$, which indicate the assignment and sequencing decisions taken by the constructive stage.

```
Set iter = 1, active_i = false
WHILE iter = Seq_i and i ∈ I^b
    LOOP (i ∈ I^b and iter = Seq_i )
        active_i = true
        LOOP (i' ∈ SB_i)
            active_i' = true
        END LOOP
    END LOOP
    SOLVE  MILP model
    LOOP (i ∈ I^b and iter = Seq_i )
        LOOP (s ∈ S^b)
            LOOP (k ∈ K_s)
                FIX variable Y_ik
            END LOOP
        END LOOP
        LOOP (i' ∈ SB_i)
            LOOP (s ∈ S^sb)
                LOOP (k ∈ K_s)
                    FIX variable Y_i'k
                END LOOP
            END LOOP
        END LOOP
    END LOOP
    iter = iter + 1
END WHILE
CurrentSol = MK
sY_ik = Y_ik
LOOP ((i, i') ∈ I^b and Seq_i < Seq_i')
    LOOP (s ∈ S^b)
        IF (i < i') THEN
            sW_ii's = 1
        ELSE
            sW_ii's = 0
        ENDIF
    END LOOP
END LOOP
LOOP ((i, i') ∈ I^sb and Seq_i ≤ Seq_i'))
    LOOP (s ∈ S^sb)
        IF (i < i') THEN
            sW_ii's = 1
        ELSE
            sW_ii's = 0
        ENDIF
    END LOOP
END LOOP
```

Fig. 2. Pseudo-code for the constructive step.

4.2 Second phase: Improvement step

Taking as starting point the assignments sY_{ik} and sequencing decisions $sW_{ii's}$ obtained as initial solution in the constructive step, this second phase applies the strategy of rescheduling each product $i \in I$ in a sequential way to try to improve the current solution. In other words, reassignment and reordering decisions are iteratively taken for each product i and its sub-assemblies $i' \in SA_i$. The improvement stage is executed sequentially until no improvement can be achieved to the makespan. The pseudo-code for the improvement step is given in Fig. 3.

At first, the parameter $BestSol$ is initialized with a big value that should be greater than the makespan found in the constructive step. Then, the procedure starts to iterate over the set I^f following the sequence parameter Seq_i; the parameter $iter$ indicates the next product order that will be rescheduled.

A boolean parameter $active_i$ is used for determining the subset of products i that can be rescheduled at each iteration. When a final product $i \in I^f$ is chosen, the boolean parameter $active_i$ is set to true for the final product i and its subassemblies $i' \in SA_i$. The MILP formulation (1)-(8) activates only the binary variables Y_{ik} and $W_{ii's}$ for products i with $active_i = true$. Reassignment to other units is not allowed for products with parameter $active_i$ set to false. Furthermore, their relative position in the processing sequence remains unchanged.

This decomposition strategy allows reducing the number of binary variables of the mathematical formulation with regards to the full space approach, and reduces drastically the CPU time needed to solve the model. Note that solving the full space approach is equivalent to setting $active_i = true \ \forall i \in I$.

Every time a rescheduling action is executed, the current solution is updated. Once the rescheduling step was applied for all products, the procedure checks the makespan achieved. If the solution found ($CurrentSol$) is better than the best solution obtained until that moment ($BestSol$), the algorithm updates the makespan ($BestSol = CurrentSol$) and goes to execute the improvement step for all products again. Otherwise, the algorithm ends and reports the current solution as the best solution found for the problem under study.

```
Set BestSol = M
WHILE CurrentSol < BestSol
    BestSol = CurrentSol
    iter = 1
    WHILE iter = Seq_i and i ∈ I^b
        active_i = false
        LOOP (i ∈ I^b and iter = Seq_i )
            RELEASE variables Y_ik
            active_i = true
            LOOP (i' ∈ SB_i)
                RELEASE variables Y_ik
                active_i' = true
            END LOOP
        END LOOP
        SOLVE MILP model (29) – (36)
        CurrentSol = MK
        LOOP (i ∈ I^b and iter = Seq_i )
            LOOP (s ∈ S^b)
                LOOP (k ∈ K_s)
                    FIX variable Y_ik
                END LOOP
            END LOOP
            LOOP (i' ∈ SB_i)
                LOOP (s ∈ S^sb)
                    LOOP (k ∈ K_s)
                        FIX variable Y_i'k
                    END LOOP
                END LOOP
            END LOOP
        END LOOP
        LOOP ((i, i') ∈ I^b and i < i' and
            (active_i = true or active_i' = true))
            LOOP ( s ∈ S^b)
                FIX variable W_ii's
            END LOOP
        END LOOP
        LOOP ((i, i') ∈ I^sb and i < i' and
            (active_i = true or active_i' = true))
            LOOP ( s ∈ S^sb)
                FIX variable W_ii's
            END LOOP
        END LOOP
    iter = iter + 1
    END WHILE
END WHILE
```

Fig. 3. Pseudo-code for the improvement step.

5 Computational results

The MILP-decomposition algorithm is applied for the solution of a complex case study arising in a shipbuilding company, which constructs ships for the development of marine resources, specifically for the offshore oil and gas industry. This real-world FFSP-A problem involves 7 processing stages, each one with K_s processing units working in parallel, as is shown in Fig. 4. A ship is built using dozens of blocks of specific size. A block is the largest construction unit of a ship. In turn, each block is assembled from one or more sub-blocks, which are composed of steel plates according to the design drawing for the ship. Both blocks and sub-blocks are considered intermediate products in the ship, which contains other components such as pipes, supports, and electronic equipment. From Fig. 4, it follows that stage s_1-s_2 process sub-blocks, which are then assembled in stage s_3 to form the block. The last processing stages on the line (s_4-s_6) perform operations on the blocks, which are finally transported and positioned in a dry dock (stage s_7) for assembling the ship. Note that this manufacturing process includes two assembly stages: in the first one, each block is constructed by one or more sub-blocks while in the second one the mounting of these blocks is carried out to build the ship.

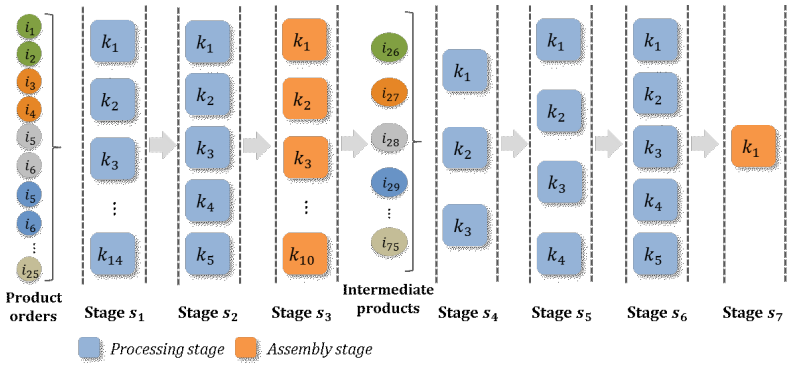

Fig. 4. FFSP-A process – case study.

From the original case study, 10 problem instances were derived in order to test the computational performance of the decomposition algorithm when facing different problem sizes. Alternative storage policy, UIS and NIS, were considered for each problem size. Moreover, it is assumed that the blocks are formed by two sub-blocks. All experimental studies were implemented in GAMS 24.9.2 with CPLEX 12.6.3.0 as MIP solver and run on a PC with four-core Intel Xeon X5650 Processor (2.6 GHz). Besides, the termination criterion imposed for the solution of all problem instances has been either 0% optimality gap or 3600 sec of CPU time.

Table 1 presents a comparison of both the results reported by MILP model and those reached by the decomposition algorithm. The expression $N \times M$ refers to a ship constructed with N blocks and M sub-blocks. The smallest problem addressed involves 10 sub-blocks and 5 blocks, while the biggest one deals with a ship built with 50 sub-blocks and 25 blocks. From Table 1, it follows that when the amount of blocks

exceeds the number of 5, the full space approach does not find the optimal solution within the time limit specified, reporting a high integrality gap for all examples. Instead, near-optimal solutions, sometimes the optimal one, are found by the solution strategy for all problem instances in few seconds of CPU time. For the more complex instance, P.10, the algorithm finds a solution of 270.5 days after 56.6 seconds, reaching an improvement of 16.5% with regards to the solution reported by the MILP model after 3600 seconds of CPU time.

Table 1. Comparison between exact MILP formulation and MILP-based algorithm statistic.

Problem	Size $N \times M$	Storage policy	MILP Model			MILP-based strategy			Enhanced solution (%)
			MILP solution	GAP (%)	CPU Time (s)	Initial solution	Best solution	Total CPU (s)	
P.01	5 × 10	UIS	126.3	0	2.3	144.0	126.3	2.3	0
P.02	5 × 10	NIS	126.3	0	2.2	144.4	126.3	2.1	0
P.03	10 × 20	UIS	160.1	12.7	3600	176.3	160.0	11.1	0.1
P.04	10 × 20	NIS	161.4	13.4	3600	177.9	160.3	6.8	0.7
P.05	15 × 30	UIS	202.4	24.3	3600	239.1	200.2	16.3	1.1
P.06	15 × 30	NIS	210.6	27.2	3600	241.9	202.7	24.1	1.9
P.07	20 × 40	UIS	229.2	27.8	3600	248.8	221.0	37.6	4.4
P.08	20 × 40	NIS	240.2	30.6	3600	255.4	228.6	41.2	4.8
P.09	25 × 50	UIS	290.8	37.9	3600	301.1	262.8	49.8	9.6
P.10	25 × 50	NIS	323.8	44.3	3600	298.3	270.5	56.6	16.5

The best solution found by the general precedence model for example P.10 is shown in Fig. 5. In this picture, each block $i \in I^f$ and its sub-assemblies $i' \in SA_i$ are depicted with the same color and labeled according to the value of parameter Seq_i. This helps to the reader to easily visualize the block assembly operation at stage s_3. Moreover, the processing stages are separated through dashed lines. The Gant chart shows as the blocks are orderly processed in stage s_7, following the assembly sequence given by parameter Seq_i.

On the other hand, for industrial-size example P.10, the constructive step converges to a solution of 298.3 days. In this starting solution, the products assigned to the same processing unit are sequenced according to the value of parameter Seq_i in all processing stages, not only at stage s_7 (dry dock). When this condition is relaxed in the improvement stage and reassignment and reordering actions are iteratively applied on the schedule, the final solution depicted in Fig. 6 is reported by the procedure. The makespan is enhanced 9.3% from 298.3 to 270.5 days.

Finally, it is worth to remark that, although the iterative approach does not assure the optimality of the solutions reported, it is capable of reaching solutions that are up to 16.5% better than those found by the exact approach with significant less computation effort. It is important to emphasize that, this improvement in the schedule allows reducing one month of work in the productive system and hence, a significant savings are obtained by the company.

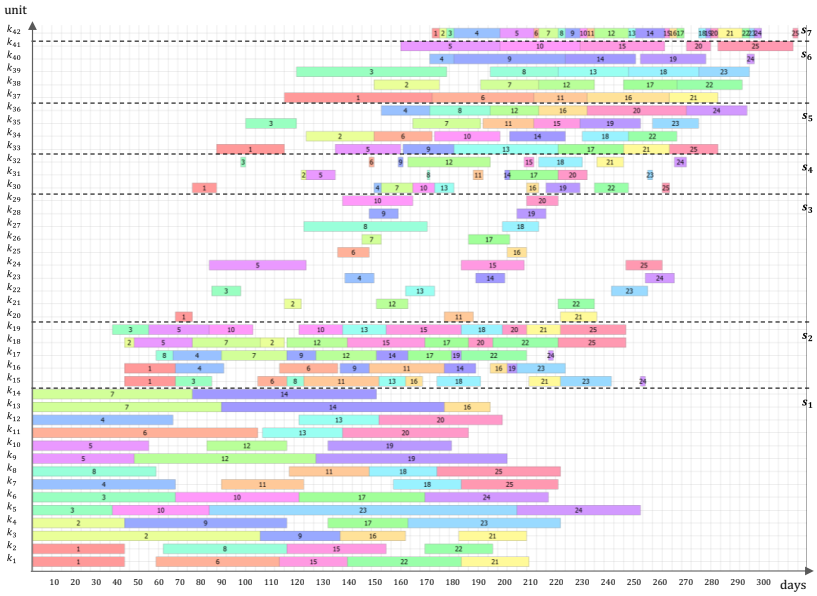

Fig. 5. Gantt chart of the best solution found by the general precedence approach for example P.10 (problem structure 25×50 under NIS policy).

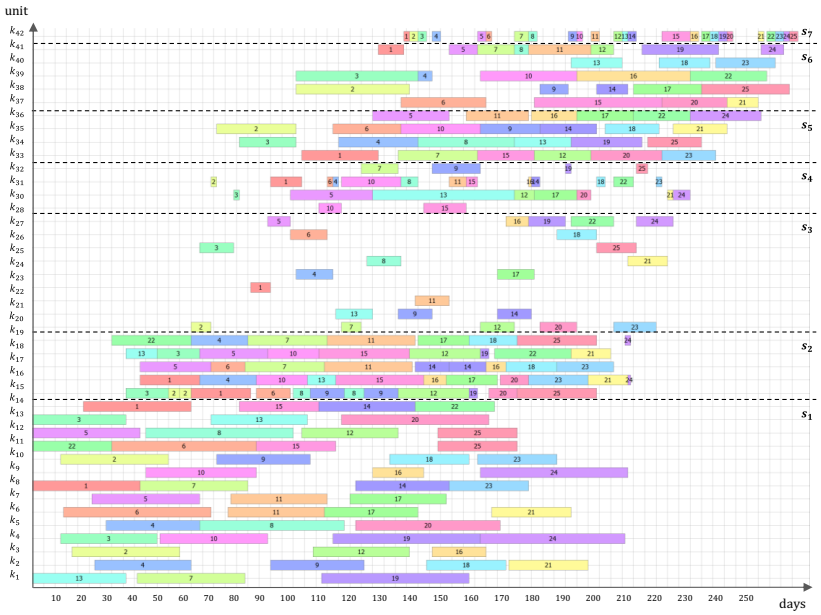

Fig. 6. Gantt chart of the best schedule reported by the MILP-based strategy for example P.10 (problem structure 25×50 under NIS policy).

6 Conclusions

A MILP-based iterative solution algorithm for solving industrial-scale FFSP-A problems has been presented in this work. The procedure was based on a MILP scheduling formulation rely on the general precedence notion. The performance of the proposed methodology has been deeply evaluated by solving several instances derived from a real-world case of study. Computational results showed that high-quality solutions can be efficiently found by the algorithm in short computational time, outperforming the rigorous optimization approach. The significant difference in the computational burden presented by both approaches is due to the iterative strategy allows decomposing the full problem into smaller sub-problems, which are solved iteratively.

References

1. Pinedo, M. L.: Scheduling: Theory, Algorithms, and Systems. Fifth Edit, Scheduling: Theory, Algorithms, and Systems. Fifth Edit. New York: Springer. (2016)
2. Méndez, C. A., Henning, G. P., Cerdá, J.: An MILP continuous-time approach to short-term scheduling of resource-constrained multistage flowshop batch facilities, Computers & Chemical Engineering, 25(4–6), 701–711 (2001)
3. Méndez, C. A., Cerdá J., Grossmann, I.E., Harjunkoski, I., Fahl, M.: State-of-the-art review of optimization methods for short-term scheduling of batch processes, Computers and Chemical Engineering, 30(6–7), 913–946 (2006)
4. Kopanos, G. M., Méndez, C. A., Puigjaner, L.: MIP-based decomposition strategies for large-scale scheduling problems in multiproduct multistage batch plants: A benchmark scheduling problem of the pharmaceutical industry, European Journal of Operational Research, 207(2), 644–655 (2010)
5. Roslöf, J., Harjunkoski, I., Bjorkqvist, J., Karlsson, S., Westerlund T.: An MILP-based reordering algorithm for complex industrial scheduling and rescheduling, Computers & Chemical Engineering, 25(4–6), 821–828 (2001)
6. Roslöf, J., Harjunkoski, I., Westerlund, T., Isaksson, J.: Solving a large-scale industrial scheduling problem using MILP combined with a heuristic procedure, European Journal of Operational Research, 138(1), 29–42 (2002)