



An efficient MILP-based decomposition strategy for solving large-scale scheduling problems in the shipbuilding industry

Natalia P. Basán¹ · Mariana E. Cóccola¹ · Alejandro García del Valle² · Carlos A. Méndez¹

Received: 31 August 2018 / Revised: 3 July 2019 / Accepted: 4 July 2019 / Published online: 11 July 2019
© Springer Science+Business Media, LLC, part of Springer Nature 2019

Abstract

This work presents a novel hybrid and systematic MILP-based solution approach for the resolution of multi-stage scheduling problems arising in the shipbuilding industry. The manufacturing problem involves the processing of a large number of sub-blocks and blocks, which should be rigorously produced and assembled with the aim of finalizing a project on time. Firstly, this paper presents three alternative rigorous MILP mathematical formulations relied on a continuous-time representation for solving the problem under study. Although the objective values reported by these exact optimization approaches outperform the results found through other solution techniques proposed in the literature to solve the same problem instances, the main drawback of the MILP models is the high computation time. Therefore, this work proposes an algorithm for solving the mathematical models in a decomposable way with the goal of accelerating the resolution times. The applicability of our proposal is demonstrated by effectively coping with several instances of a real-world case study dealing with the construction of a ship for the development of marine resources. Computational results show that the proposed decomposition method is able to obtain high-quality solutions in few seconds of CPU time for all examples considered.

Keywords Multi-stage scheduling problem · Shipbuilding process · MILP model · Decomposition strategy

✉ Carlos A. Méndez
cmendez@intec.unl.edu.ar

¹ INTEC (UNL –CONICET), Güemes 3450, 3000 Santa Fe, Argentina

² University of A Coruña, C/Mendizábal s/n, 15403 Ferrol, Spain

List of symbols

Indices

i	Product order (block or sub-block)
k	Processing unit
s	Processing stage
p	Time slot

Sets

I	Set of product orders
K	Set of processing units
S	Set of processing stages
P	Set of time slots
I^b	Set of blocks
I^{sb}	Set of sub-blocks
SB_i	Subset of sub-blocks that integrate a block $i \in I^b$
S^b	Available processing stages s to process block $i \in I^b$
S^{sb}	Available processing stages s to process sub-block $i \in I^{sb}$
S^a	Available processing stages s to assemble sub-blocks $i \in I^{sb}$
P_k	Set of time slots for processing unit k (slot-based continuous time formulation)
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	Constant for big-M constraints
$iter$	Number of block to be inserted at each iteration
$active_i$	Indicating if product i is active in the current iteration
sY_{ik}	Saving assignment decisions
$sW_{i'j's}$	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
Ts_{pk}	Start time of time slot p in processing unit k (slot-based continuous time formulation)
Tf_{pk}	Final time of time slot p in processing unit k (slot-based continuous time formulation)
MK	Makespan

Binary variables

W_{ipk}	Defining if product i is allocated to the time slot p of processing unit k (slot-based continuous time formulation)
-----------	---

$W_{i'i's}$	Defining if product i is processed before of product i' in processing stage s (global general precedence formulation)
$W_{i'i'k}$	Defining if product i is processed exactly before than i' in processing unit k (unit-specific direct precedence formulation)
Y_{ik}	Defining if product order i is processed in processing unit k
YF_{ik}	Defining if product order i is first processed in unit k

1 Introduction

Shipbuilding is a complex and long-term manufacturing process, which requires the coordination of a considerable number of limited resources. Traditionally, this process was carried out through a project-oriented approach. Due to a ship is generally constructed with few components, which are based on the same design but with a certain degree of customization, a modular approach began to be implemented several decades ago, taking into account the Lean principles and standardization processes. This modular approach uses an integrated modular design based on the prefabrication of blocks (or steel structures), which are then assembled in a block erection process to finally build the ship. The ship design determines the quantity of blocks to be produced and the size of each one of them. Figure 1 shows as a ship is built following the principles of the modular approach: the hull is divided into blocks and, in turn, each block is divided into sub-blocks.

According to Shin and Ciccantell (2009), the production of ships and marine facilities tends to be large scale and greatly non-standardized, so the shipbuilding process requires labor-intensive and technology intensive, resulting high expensive. These authors remark that the shipbuilding industry tends to have new orders for construction of ships concentrated in short periods according to the demand, which generally fluctuates wildly following the rise and decline of this industry in the world market.

The block assembly process becomes a bottleneck for the shipbuilders due to the complexity of the production system. This process requires a high degree of coordination among several resources in order to satisfy the complex production restrictions and strict storage policies. A delay in the completion of a project increases the

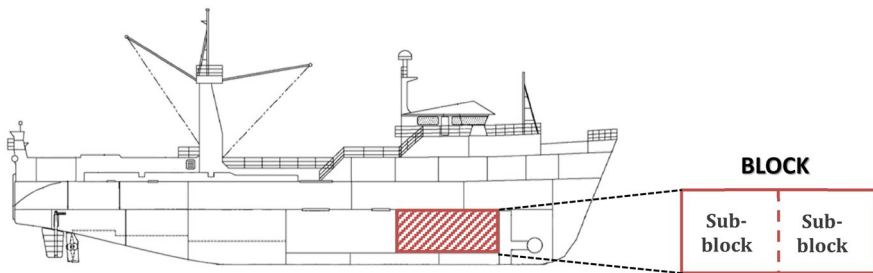


Fig. 1 Hull production—modular approach

risk of incurring penalties in delivery. Moreover, the most shipbuilding contracts provide the buyers with the right to cancel the contract in case of delayed delivery of the vessel.

Since the block assembly process represents more than half of the whole shipbuilding process, it would be very useful to have a decision-making tool that may determine the optimal scheduling of operations, reducing at minimum the man-hours (Cho et al. 1998). Although significant developments have emerged in recent decades about this problem, few works have been focused on the scheduling problem. The main challenge is to obtain a good solution that takes into account the wide range of operational constraints. Unlike other scheduling problems published in the literature, this production problem deals with assembly operations and the processing of different types of products.

In the last years, the researchers have been tried to solve the block assembly process through heuristics, metaheuristics, and simulation-based methods with the goal of providing feasible solutions with reasonable computational effort. Some contributions have been focused on representing the production process as a constraint satisfaction problem where the precedence relationships between operations are considered constraints (Kim et al. 2002; Seo et al. 2007). In addition, some heuristic algorithms have been used to determine the best scheduling that both improves the use of the area in the long term and minimizes the processing times of the blocks (Koh et al. 2008; Shang et al. 2017). Other contributions also appear in the modeling and simulation area. Lee et al. (2009), Zhuo et al. (2012), and Cebral-Fernandez et al. (2016) have developed three discrete event simulation models in order to obtain a schedule that maximizes the productivity while minimizing the total time for completing the operations. Afterwards, Basán et al. (2017) have proposed a simulation environment to evaluate the best flow of materials that minimizes the completion time of a ship through applying different heuristic rules. Although these simulation approaches are greatly useful to represent the dynamic behavior over time of the system, the quality of solutions reported by these models are highly dependent on the heuristic rules using within them.

On the other hand, several hybrid methods have also been reported in the literature for effectively coping with large size industrial cases. For instance, a research conducted by Xiong et al. (2015) have formulated a MIP (mixed integer programming) model for the hybrid assembly-differentiation flow shop scheduling problem. Since the NP-hardness of the problem, these authors propose two fast heuristics and three hybrid meta-heuristics for treating with medium and large-size instances. Zhuo et al. (2012) have developed a hybrid planning approach using discrete-event simulation and discrete spatial optimization to take into account both the programming and the spatial arrangement. In addition, Basán et al. (2017) introduced a novel hybrid simulation-based optimization method, where a MILP mathematical model and a discrete event simulation model are incorporated together into an algorithm in order to generate complete schedules with efficient computational times for large-sized shipbuilding scheduling problems. Despite these advancements, the quality of solutions found remains the main drawback of the hybrid approaches.

Robust and reliable solutions regarding to their quality, feasibility or optimality can be achieved through exact optimization approaches. Due to the block assembly

process consists of a set of products that go through different processing stages, the problem can be mathematically treated as a multi-stage system (Pinto and Grossmann 1995; Méndez et al. 2001; Kopanos et al. 2009; Castro et al. 2009), where the main objective is to find the optimal schedule that minimizes the time for completing all processing tasks. The mathematical model should consider constraints for the assignment of tasks to processing units and the sequencing of those tasks in each unit as in the classical multi-stage scheduling problem (Méndez et al. 2006; Maravelias and Sung 2009). Also, a set of new mathematical constraints should be defined for representing the assembly operations involved in the shipbuilding process. Recently, a continuous-time MILP formulation has been proposed by Basán et al. (2018) to determine the optimal schedule of a block assembly process considering up to ten blocks.

The main drawback of the MILP models is that they are characterized by high computation times. From the practical point of view, the industries prefer computational tools that report the optimal or at least, near-optimal solutions but in a quickly way. In an attempt to reduce this gap between theory and practice, the development of MILP-based hybrid approaches (Roslöf et al. 2002; Chu et al. 2014; Persson and Ölvander 2015; Tenne 2015; Fettaka et al. 2015; Larson and Wild 2016; Schweiger and Liers 2018) and techniques of decomposition-aggregation and improvement algorithms (Hasebe et al. 1991; Roslöf et al. 1999, 2001; Méndez and Cerdá 2003a; Kopanos et al. 2010; Aguirre et al. 2012; Cóccola et al. 2015; Fettaka et al. 2015) have emerged as alternative solution strategies for solving the complex scheduling problems arising in industrial environments. These techniques are generally based on rigorous mathematical programming approaches (Cerdá et al. 1997; Méndez et al. 2000, 2001, 2006; Méndez and Cerdá 2003b).

Firstly, this work proposes three new alternative continuous-time MILP formulations for the block assembly problem. The main goal is the minimization of makespan. Although these exact methods allow reaching solutions that outperform those ones achieved by evaluating alternative dispatching rules in a simulation model, such solutions are far from being the optimal ones. Moreover, the mathematical models show a high computational burden in the most examples solved. To develop a more suitable tool for industrial environments without losing the robustness exhibited by the exact optimization approaches of reaching good solutions, this paper proposes then a MILP-based decomposition algorithm that allows finding high-quality and practical solutions in few seconds of CPU time even for large-size problems.

All solution strategies proposed in this work are tested by solving several instances derived from a real-world shipbuilding problem. A detailed comparison of the computational performance of each approach is presented in order to highlight the advantages of the decomposition method.

The remainder of this paper is organized as follows. The block assembly process is described in Sect. 2. In Sect. 3, the problem statement is introduced. Afterwards, Sect. 4 presents three alternative MILP models to solve the multi-stage scheduling problem. Then, in Sect. 5, the development of the MILP-based decomposition procedure is explained in detail. Several complex problem instances of a real-world block assembly scheduling problem are presented and solved in Sect. 6. Finally, the concluding remarks follow in Sect. 7.

2 The shipbuilding process

The block assembly process may be treated as a flow shop scheduling problem, in which a set of sub-blocks and blocks must be assigned and sequenced on several processing units belonging to different processing stages. Having into account the ship design, the hull is divided into dozens of blocks of specific size (see Fig. 1). 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 drawn for the ship. Both blocks and sub-blocks are considered intermediate products in the modular approach used for constructing a ship, which contains other components such as pipes, supports, and electronic equipment.

At early stages of the shipbuilding process, steel plates and profiles are processed to form the sub-blocks, which are then assembled in a dedicate workshop to obtain large blocks as an intermediate product. After that, the blocks are sequentially processed on a set of workshops and finally, they are assembled in a dry dock, operation known as *Erection process*. Although the blocks should arrive at the dry dock with all the necessary equipment and systems effectively installed, several elements must be usually mounted during the block erection operation. The block assembly process is depicted in Fig. 2. In this picture, the stages processing sub-blocks are referenced as S^{sb} while those ones performing operations on the blocks are classified as S^b .

Firstly, the block assembly process begins with *Construction and Assembly operation*, where sheets and profiles are received, cut into small parts, and welded to form the smallest units of a ship (sub-blocks). The process continues with the *Pre-outfitting operation*, consisting in the installation of different components as pipes, brackets, and auxiliary elements in order to obtain the finished sub-blocks.

Once the sub-blocks are outfitted, they are assembled by welding operations to form the block structure. This stage is known as *Assembly*. The assembly of a block

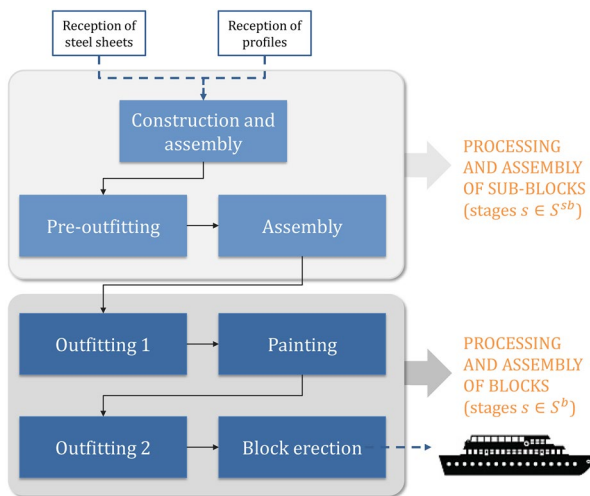


Fig. 2 The block assembly process

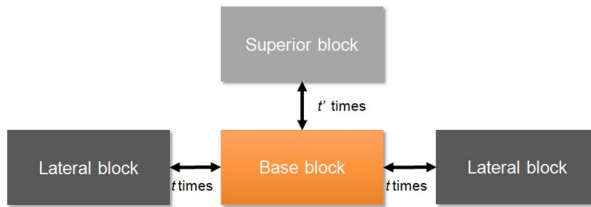


Fig. 3 Block erection assembly process

cannot begin until its sub-assemblies (sub-blocks) are ready. The next stages of the manufacturing process deal with the construction and transportation of the blocks. After assembling the sub-blocks, the *Outfitting 1 operation* is performed to outfit items like pipes and electrical and lighting lines inside the blocks. Then, these one are blasted and painted in the painting booths during *Painting operation*. During this action, the protection and design requirements of blocks are considered. After outfitting and painting stages, a second outfitting process is performed on the blocks before they will be moved to the dock (*Outfitting 2 operation*). All equipment that might be deteriorated in the painting stations, such as wires and electronic components, are again installed at this stage. Finally, the prefabricated and painted blocks are transported and positioned in the dry dock for assembling the ship. The *Block erection operation* is carried out to mount the blocks, one after another, according to a pre-predefined sequence. This assembly sequence is predefined by the company, taking into account if the unit to be assembled is a base block, a lateral block, or an upper block, as shown Fig. 3. The building of a ship always begins with a predefined base block and then, the remaining blocks are assembled and welded respecting the mentioned sequence. The processing time at *Erection* stage is modified taking into account the type of block to be mounted (lateral or superior block). Note that a block may be assembled just if the erection operations of its predecessor block have already been finished. Thus, if a block arrives at the dock before these operations end, it must wait.

3 Problem statement

According to the integrated modular design, the shipbuilding problem involves the processing of a set of product $i \in I$ through several stages $s \in S$, following a predefined and known production sequence. This manufacturing process is characteristic of the flexible flow shop (FFSP) environments, in which each stage s has a set $k \in K_s$ of identical processing units working in parallel. In the case of shipyard, each unit represents a workshop. Hence, the block assembly process may be mathematical represented as a FFSP problem with the following features:

- A set of products $i \in I$ should be processed by following a predefined sequence of stages $s \in S$.

- The set I is partitioned in two subsets: I^{sb} and I^b , where I^{sb} contains the products representing sub-blocks and I^b includes the blocks. Note that $I = I^b \cup I^{sb}$.
- Each processing stage s should be performed in a specific subset of units $k \in K_s$. In addition, each unit k is able to perform just one processing stage (dedicated unit).
- Each block $i \in I^b$ is made up of one or more predefined sub-blocks $i' \in SB_i$.
- Products $i \in I$ are not processed in all the stages $s \in S$ because there are several assembly operations.
- The first stages $s \in S^{sb}$ process sub-blocks while the last stages $s \in S^b$ perform operations on the blocks.
- Sub-blocks $i \in I^{sb}$ are assembled to form a block at intermediate stage $s \in S^a$.
- Blocks $i \in I^b$ are assembled in a dock to form the hull of the ship at stage $s \in S^{seq}$.
- The final assembly sequence on slipway (*Erection stage*), which determines the order in which the blocks must be mounted, is known a priori through the parameter Seq_i .
- Each workshop $k \in K$ can process one block (or sub-block) at a time and, likewise, a product i only can be processed at most in one workshop at each stage $s \in S$.
- The processing times pt_{is} are known with certainty remaining invariant over time.
- Transfer times of the blocks (or sub-blocks) between stages are considered negligible.
- Raw material are considered unlimited.
- Either the non-intermediate storage (NIS) policy or the unlimited intermediate storage (UIS) policy may be adopted. When a NIS strategy is used, each workshop becomes intermediate storage if its processing has finished and the next step is not available yet.

It is worth to remark that all assumptions listed above are realistic features of the block assembly process. The problem statement does not incorporate any simplification with regards to the real-data provided by the company.

4 Mathematical formulations

Having defined the block assembly process as a flexible flow shop scheduling problem, three continuous-time mathematical formulations of mixed integer linear mathematical programming (MILP) that have been published in the literature for the short-term scheduling of multistage batch plants are presented in this section (Méndez et al. 2006): (1) the first formulation is based on the concept of time-slots (Pinto and Grossmann 1995), (2) the second one is constructed following the immediate batch precedence concept (Cerdá et al. 1997), and (3) the last formulation is based on the general precedence concept (Méndez et al. 2001). These three monolithic approaches are widespread

used by the communities of Operations Research and Process Systems Engineering for solving the production problems arising in the short-term scheduling area.

It is worth noticing that the MILP models presented in this paper have been modified from their original proposals in the literature in order to adapt them to the shipbuilding problem. The problem goal is to determine the optimal production schedule that minimizes the time needed to construct a ship. In this way, the mathematical model should determine: (1) the assignment of products to workshops, (2) the processing sequence for any pair of product allocated at each workshop, and (3) the completion time of each product at each processing stage.

4.1 Objective function

The shipbuilding industry aims at increasing its productive efficiency and consequently, its profit, which is directly related to the time required to build a large-scale ship. A delay in the completion of a project increases the risk of incurring penalties in delivery. Moreover, the buyers have the right to cancel the shipbuilding contract in case of delayed delivery of the vessel. Hence, the makespan criterion (*MK*) has been chosen as optimization goal, as shown in Eq. (1).

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

4.2 The slot-based continuous time formulation

In this mathematical formulation, a set of time-slots $p \in P_k$ is postulated for each processing unit k in order to allocate in them the blocks or sub-blocks to be processed. A binary variable W_{ipk} is used to determine if product i is assigned to time-slot p of unit k .

$$\sum_{k \in K_s} \sum_{p \in P_k} W_{ipk} = 1 \quad \forall ((i \in I^{sb}, s \in S^{sb}) \cup (i \in I^b, s \in S^b)) \tag{2}$$

$$\sum_{\substack{(i \in I^{sb}, s \in S^{sb}) \\ \cup (i \in I^b, s \in S^b)}} W_{ipk} \leq 1 \quad \forall s \in S, k \in K_s, p \in P_k \tag{3}$$

$$\sum_{i \in I} W_{i(p+1)k} \leq \sum_{i \in I} W_{ipk} \quad \forall k \in K_s, p \in P_k, (p+1) \in P_k \tag{4}$$

$$Tf_{is} \geq Ts_{is} + pt_{is} \quad \forall ((i \in I^{sb}, s \in S^{sb}) \cup (i \in I^b, s \in S^b)) \tag{5}$$

$$Tf_{pk} \geq Ts_{pk} + \sum_{\substack{(i \in I^{sb}, s \in S^{sb}) \\ \cup (i \in I^b, s \in S^b)}} W_{ipk} pt_{is} \quad \forall s \in S, k \in K_s, p \in P_k \tag{6}$$

$$Ts_{is} = Tf_{i(s-1)} \quad \forall ((i \in I^{sb}, s \in S^{sb}) \cup (i \in I^b, s \in S^b)) : s > 1 \tag{7a}$$

$$Ts_{is} \geq Tf_{i(s-1)} \quad \forall ((i \in I^{sb}, s \in S^{sb}) \cup (i \in I^b, s \in S^b)) : s > 1 \tag{7b}$$

$$Tf_{pk} \leq Ts_{(p+1)k} \quad \forall k \in K, p \in P_k, (p+1) \in P_k \tag{8}$$

$$Tf_{is} - Tf_{pk} \geq -M(1 - W_{ipk}) \quad \forall ((i \in I^{sb}, s \in S^{sb}) \cup (i \in I^b, s \in S^b)), k \in K_s, p \in P_k \tag{9a}$$

$$Tf_{is} - Tf_{pk} \leq M(1 - W_{ipk}) \quad \forall ((i \in I^{sb}, s \in S^{sb}) \cup (i \in I^b, s \in S^b)), k \in K_s, p \in P_k \tag{9b}$$

$$Ts_{is} - Ts_{pk} \geq -M(1 - W_{ipk}) \quad \forall ((i \in I^{sb}, s \in S^{sb}) \cup (i \in I^b, s \in S^b)), k \in K_s, p \in P_k \tag{10a}$$

$$Ts_{is} - Ts_{pk} \leq M(1 - W_{ipk}) \quad \forall ((i \in I^{sb}, s \in S^{sb}) \cup (i \in I^b, s \in S^b)), k \in K_s, p \in P_k \tag{10b}$$

$$Ts_{is} \geq Tf_{i'(s-1)} \quad \forall i \in I^b, i' \in SB_i, s \in S^a \tag{11}$$

$$Ts_{i's} \geq Tf_{is} \quad \forall (i, i') \in I^b, s \in S^a_{seq} : Seq_i < Seq_{i'} \tag{12}$$

$$W_{ipk} - W_{i',p+1,k} = 0 \quad \forall (i, i') \in I^b, k \in K_s, s \in S^a_{seq}, p \in P_k, (p+1) \in P_k : Seq_i < Seq_{i'} \tag{13}$$

$$MK \geq Tf_{is} \quad \forall i \in I^b, s \in |S| \tag{14}$$

Equation (2) assures that every block $i \in I^b$ is processed at each stage $s \in S^b$ on exactly one time-slot p of some unit $k \in K_s$; in case of $i \in I^{sb}$, the same condition must be accomplished but in this case for every stage $s \in S^{sb}$. In turn, Eq. (3) defines that each time-slot p of unit $k \in K_s$ can at most be assigned to one product i corresponding to stage s . To avoid empty positions between the time slots postulated for a unit k , the allocation of any product i to slot $p + 1$ requires that at least other processing task i will be assigned to slot p . This constraint is expressed through Eq. (4).

On the other hand, the timing decisions are determined by Eqs. (5)–(7). The completion time of product order i at stage s , Tf_{is} , and the finish time of slot p in unit k , Tf_{pk} , are computed by Eqs. (5) and (6), respectively, as the starting time plus the processing time pt_{is} associated to the product order i assigned to slot p . The earliest time for starting the processing of product order i at stage s , Ts_{is} , is determined by Eq. (7). Equation (7a) is

used when the non-intermediate storage policy is assumed by the scheduler; otherwise, when an UIS policy is adopted, Eq. (7b) becomes active. Since the time-slots defined for every unit k are sequentially arranged over time, the starting time of slot $p + 1$ requires that the processing of slot p be finished, which is expressed through Eq. (8). Note that if the processing of product i at stage s is accomplished on slot $p \in P_k$ with $k \in K_s$, i.e. $W_{ipk} = 1$, then it must be fulfilled that $Tf_{is} = Tf_{pk}$ and $Ts_{is} = Ts_{pk}$. These conditions are forced to be satisfied by Eqs. (9) and (10). For assembly stage $s \in S^a$, Eq. (11) determines that a block i should not start to be constructed until their specific sub-blocks $i' \in SB_i$ have finished its processing in the above stage ($s - 1$).

For the last processing stage $s \in S_{seq}^a$, in where the blocks are mounted following a predefined sequence, either Eq. (12) or Eq. (13) may be used to represent the ship assembly operation. Finally, Eq. (14) states a lower bound for the variable MK to be minimized.

4.3 The immediate precedence formulation

When the block assembly process is mathematically represented through the immediate precedence notion, three sets of binary variables are defined: YF_{ik} denoting that product i is the first being processed in unit k , Y_{ik} determining that block (or sub-block) i is processed in k but not in the first place, and $W_{i'k}$ denoting that task i is processed right before task i' in unit k . The constraints defining this continuous-time representation are follows.

$$\sum_{k \in K_s} (Y_{ik} + YF_{ik}) = 1 \quad \forall ((i \in I^{sb}, s \in S^{sb}) \cup (i \in I^b, s \in S^b)) \tag{15}$$

$$\sum_{i \in I} YF_{ik} \leq 1 \quad \forall k \in K \tag{16}$$

$$W_{i'k} \leq Y_{ik} + YF_{ik} \quad \forall (i, i') \in I, k \in K : i \neq i' \tag{17}$$

$$W_{i'k} \leq Y_{i'k} \quad \forall (i, i') \in I, k \in K : i \neq i' \tag{18}$$

$$YF_{ik} + \sum_{i' \in I: i \neq i'} W_{i'ik} \leq 1 \quad \forall i \in I, k \in K \tag{19}$$

$$\sum_{i' \in I: i \neq i'} W_{i'ik} = Y_{ik} \quad \forall i \in I, k \in K \tag{20}$$

$$\sum_{i' \in I: i \neq i'} W_{i'k} \leq 1 \quad \forall i \in I, k \in K \tag{21}$$

$$Tf_{is} \geq Ts_{is} + pt_{is} \quad \forall ((i \in I^{sb}, s \in S^{sb}) \cup (i \in I^b, s \in S^b)) \tag{22}$$

$$Ts_{is} = Tf_{i(s-1)} \quad \forall ((i \in I^{sb}, s \in S^{sb}) \cup (i \in I^b, s \in S^b)) : s > 1 \tag{23a}$$

$$Ts_{is} \geq Tf_{i(s-1)} \quad \forall ((i \in I^{sb}, s \in S^{sb}) \cup (i \in I^b, s \in S^b)) : s > 1 \quad (23b)$$

$$Ts_{i's} \geq Tf_{i,s} - M(1 - W_{i'k}) \\ \forall ((i, i') \in I^{sb}, s \in S^{sb}) \cup ((i, i') \in I^b, s \in S^b), \quad k \in K_s : i \neq i' \quad (24)$$

$$Ts_{is} \geq Tf_{i'(s-1)} \quad \forall i \in I^b, i' \in SB_i, s \in S^a \quad (25)$$

$$Ts_{i's} \geq Tf_{is} \quad \forall (i, i') \in I^b, s \in S^a_{seq} : Seq_i < Seq_{i'} \quad (26)$$

$$W_{i'k} = 1 \quad \forall (i, i') \in I^b, k \in K_s, s \in S^a_{seq} : Seq_i = Seq_{i'} - 1 \quad (27)$$

$$MK \geq Tf_{is} \quad \forall i \in I^b, s \in |S| \quad (28)$$

Equation (15) assures that every sub-block $i \in I^{sb}$ goes through just one unit $k \in K_s$ at each stage $s \in S^{sb}$ and that every block $i \in I^b$ is processed in just one unit $k \in K_s$ at each stage $s \in S^b$. In addition, Eq. (16) enforces the condition that at most one product i is the first processed in every unit k . When a unit k is not used, $YF_{ik} = 0 \forall i \in I$. For sequencing decisions, Eqs. (17) and (18) impose that the sequencing variable $W_{i'k}$ can be active only if the pair (i, i') are processed in the same unit k (i.e., $Y_{ik} + YF_{ik} = 1$ and $Y_{i'k} = 1$). Moreover, Eq. (19) assures that a product i might be processed in unit k either in the first place or after another product i' while Eq. (20) states that every time a product i is not the earliest processed at the assigned unit k , i.e. $Y_{ik} = 1$, it must feature a direct predecessor i' in the processing sequence of k . On the other hand, Eq. (21) states that at most a single product i' can be scheduled immediately after other product i unless i will be the last in the processing sequence of unit k .

For timing constraints, Eq. (22) computes the ending time Tf_{is} of product i at stage s as its starting time Ts_{is} plus its processing time pt_{is} . The earliest time for starting the processing of product i at stage s is determined by Eqs. (23)–(25). Firstly, a product i cannot begin to be processed at stage s until its processing has finished in previous stage $(s - 1)$. This condition is forced by Eq. (23a) or Eq. (23b), depending on the storage policy adopted by the scheduler. The sequencing constraint Eq. (24) determines that the starting time $Ts_{i's}$ of a product i' at processing stage s must be greater than the ending time Tf_{is} of its immediate predecessor i ($W_{i'k} = 1$ for any $k \in K_s$). Moreover, Eq. (25) determines that a block $i \in I^b$ can start to be mounted at stage $s \in S^a$ after their associated sub-blocks $i' \in SB_i$ have completed their processing in the previous stage. To force the assembling sequence on slipway, it may be used either the timing variables Tf_{is} or the sequencing variables $W_{i'k}$, as is expressed through Eqs. (26) and (27), respectively. Finally, the makespan value is computed by Eq. (28).

4.4 The general precedence formulation

In contrast to the previous model, 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 $W_{i'i's}$ just for all pair of products (i, i') with $i < i'$, processed at stage s . The variable $W_{i'i's}$ takes a meaning only when both products (i, i') are assigned to the same processing unit $k \in K_s$, i.e. $Y_{ik} = Y_{i'k} = 1$. In this case, $W_{i'i's}$ becomes equal to 1 whenever product i is processed before i' in the processing sequence of unit k , otherwise it takes value 0, indicating that i' is processed before i in unit k . If both product orders are not assigned to the same unit $k \in K_s$, the value of $W_{i'i's}$ is negligible. On the other hand, Y_{ik} is the assignment binary variable valuing 1 if task i is processed at unit k . The general precedence formulation for the problem under study includes the following sets of constraints.

$$\sum_{k \in K_s} Y_{ik} = 1 \quad \forall ((i \in I^{sb}, s \in S^{sb}) \cup (i \in I^b, s \in S^b)). \tag{29}$$

$$Tf_{is}^f \geq Ts_{is} + pt_{is} \quad \forall ((i \in I^{sb}, s \in S^{sb}) \cup (i \in I^b, s \in S^b)) \tag{30}$$

$$Ts_{is} = Tf_{i(s-1)} \quad \forall ((i \in I^{sb}, s \in S^{sb}) \cup (i \in I^b, s \in S^b)) : s > 1 \tag{31a}$$

$$Ts_{is} \geq Tf_{i(s-1)} \quad \forall ((i \in I^{sb}, s \in S^{sb}) \cup (i \in I^b, s \in S^b)) : s > 1 \tag{31b}$$

$$Ts_{i's} \geq Tf_{is} - M(1 - W_{i'i's}) - M(2 - Y_{ik} - Y_{i'k}) \tag{32}$$

$$\forall (((i, i') \in I^{sb}, s \in S^{sb}) \cup ((i, i') \in I^b, s \in S^b)), k \in K_s : i < i'$$

$$Ts_{is} \geq Tf_{i's} - MW_{i'i's} - M(2 - Y_{ik} - Y_{i'k}) \tag{33}$$

$$\forall ((i, i') \in I^{sb}, s \in S^{sb}) \cup ((i, i') \in I^b, s \in S^b), k \in K_s : i < i'$$

$$Ts_{is} \geq Tf_{i'(s-1)} \quad \forall i \in I^b, i' \in SB_i, s \in S^a \tag{34}$$

$$Ts_{i's} \geq Tf_{is} \quad \forall (i, i') \in I^b, s \in S_{seq}^a : Seq_i < Seq_{i'} \tag{35}$$

$$MK \geq Tf_{is} \quad \forall i \in I^b, s \in |S| \tag{36}$$

Equation (29) defines the allocation constraint. The timing constraints are determined by Eqs. (30) and (31), which are similar to Eqs. (22) and (23) previously explained for the immediate precedence notion. The sequencing constraints on a same processing unit k are expressed through Eqs. (32) and (33). The sub-blocks assembly operation at stage $s \in S^a$ is represented by Eq. (34). On the other hand,

the assembly sequence on slipway is determined by Eq. (35). This sequencing constraint forces the starting time of block i' to be greater than the completion time of any block i that is before at erection sequence ($Seq_i < Seq_{i'}$). Refer to Méndez et al. (2006) and Cóccola et al. (2014) for more details about the general precedence concept.

5 The MILP-based decomposition algorithm

Although the mathematical formulations presented in above section are capable to represent and solve the block assembly process, the practice shows that realistic industrial applications are either intractable or result in poor solutions when rigorous optimization approaches are used. The computational efficiency of the rigorous MILP methods is quickly deteriorated when increasing the problem size because of the thousands of binary variables involved. To overcome this limitation, we propose a solution strategy based on a decomposition algorithm that allows finding good quality solutions with very low CPU time even for large-size problems.

The decomposition method is based on the strategy of first obtaining an initial solution (constructive stage), and then gradually enhancing it by applying several rescheduling iterations (improvement stage). Both algorithmic stages have as core the general precedence MILP model defined by Eqs. (29)–(36). Thus, the procedure exploits the benefits of combining the robustness of the MILP models with the flexibility of the heuristic rules. At this point, it is worth mentioning that any alternative mathematical formulation may be easily adapted to the proposed decomposition strategy, but in general, the general precedence representation shown the best computational performance.

The general structure of the algorithm is given in Fig. 4. Note that a feasible initial scheduling solution is obtained from inserting the blocks one by one in the scheduled and fixing the decision variables at their optimum values. A block $i \in I^b$ and its sub-assemblies $i' \in SB_i$ are scheduled at each iteration of the constructive stage. Once determining a starting solution, this one is iteratively improved by executing several rescheduling actions in order to find a high-quality scheduling solution with minimal computational effort. At each improvement iteration, the assignment and sequencing decisions are left free just for the block $i \in I^b$ (and its sub-assemblies) that is being rescheduled. Such decomposition strategy significantly reduces the MIP solver search space at each model execution, accelerating its resolution. When no improvements are obtained for the objective function after rescheduling all blocks, the current schedule is reported as the best solution by the procedure.

5.1 Constructive step

The first phase of the decomposition algorithm aims at rapidly generating an initial full schedule for the problem under study. Here, the production problem is solved iteratively through the schedule of a subset of products at each solver execution. The constructive method is based on the insertion technique presented by Kopanos et al.

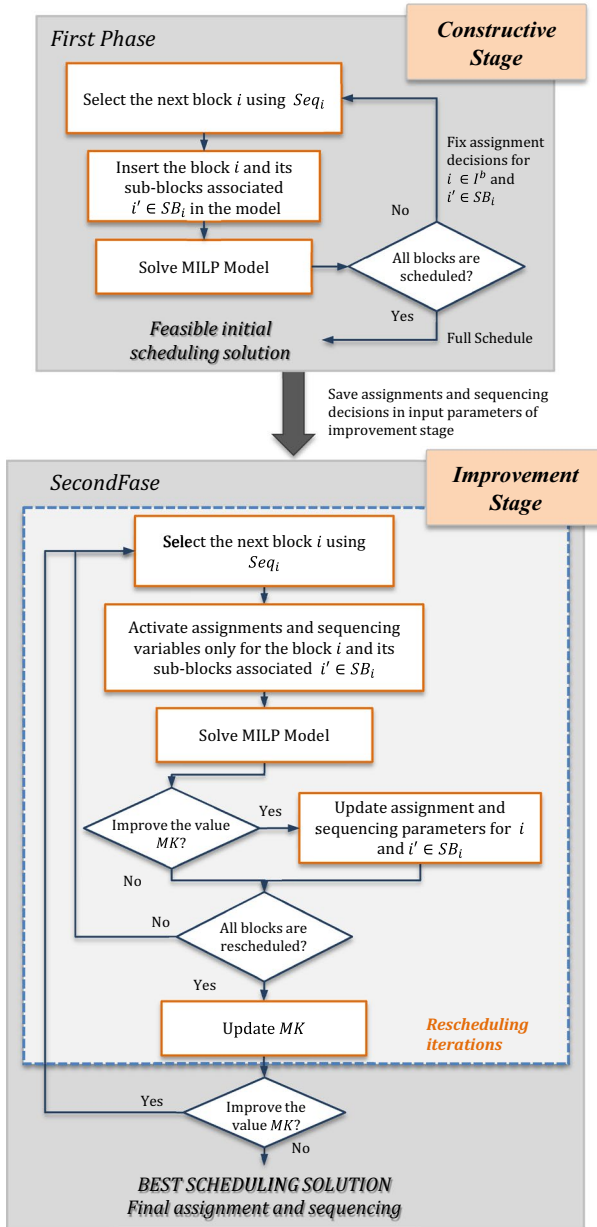


Fig. 4 Overview of the iterative MILP-based algorithm

(2010) for solving large-scale pharmaceutical scheduling problems: the orders are inserted (scheduled) one-by-one in an iterative mode by solving the original MILP model. The sequence in which the products are inserted may vary according to

the characteristics of the problem (Roslöf et al. 2001, 2002). The insertion criterion should be always defined taking into account the features of productive system. Here, it is proposed to insert the products according to the assembly sequence on slipway, known through parameter Seq_i . This guarantees to find a fairly good starting solution.

For programming the constructive step, the user should be also defined the number of products to be scheduling simultaneously in order to maintain a reduced search space at each solver execution. In our proposal, just one block $i \in I^b$ and its sub-assemblies $i' \in SB_i$ are assigned and sequenced at each iteration, using Eqs. (29)–(36). The constructive stage ends when all blocks have been scheduled.

The pseudo-code for the constructive step is given in Fig. 5. The scalar $iter$ identifies the number of block $i \in I^b$ to be inserted in the current iteration while the boolean parameter $active_i$ is true when the product i (block or sub-blocks) is selected for scheduling. Once a new block and its sub-blocks are scheduled, the MILP model (29)–(36) is solved. After solver execution, the assignment variables Y_{ik} for these new insertions are fixed at their optimal values. Note that variables Y_{ik} for the blocks (and sub-blocks) that have already been scheduled in previous iterations are also fixed and only the timing decisions can be modified for these products. In other words, the values of variable Y_{ik} for already scheduled products cannot be modified in the following iterations.

It is worth to remarks that the sequencing decisions are ignored in the constructive stage and it is assumed that the production sequence at each processing stage must follow the assembly order required on slipway. This assumption is made for favoring the problem resolution. Thus, Eqs. (32) and (33) of general precedence formulation are replaced by Eq. (37).

$$Ts_{is} \geq Tf_{i's} - M(2 - Y_{ik} - Y_{i'k}) \quad \forall(((i, i') \in I^{sb}) \vee ((i, i') \in I^b)), s \in S_i, k \in K_s : Seq_i < Seq_{i'} \quad (37)$$

The constructive step solution ends to be computed when all products (blocks and sub-blocks) have been scheduled. The best value obtaining for the makespan is saved in parameter $CurrentSol$, while the values of binary variables Y_{ik} and $W_{i'i's}$ are copied in parameters sY_{ik} and $sW_{i'i's}$, respectively. These data become the starting point of the improvement stage.

5.2 Improvement step

Starting from the solution given by the first phase, the improvement step aims at improving iteratively the current solution though several rescheduling decisions. The procedure iterates over each element of the set I^b in order to perform reassignment or reordering actions on every block $i \in I^b$ and its sub-blocks $i' \in SB_i$, using the MILP model (29)–(36), until no improvement can be achieved to the objective function (makespan MK). Note that a simultaneous rescheduling of all blocks results impractical because it would be equivalent to solve the full space approach. Hence, a key point for the scheduler is to determine the number of blocks to be simultaneous

```

Set  $iter = 1, active_i = false$ 
WHILE  $iter = Seq_i$  and  $i \in I^b$ 
  LOOP ( $i \in I^b$  and  $iter = Seq_i$ )
     $active_i = true$ 
    LOOP ( $i' \in SB_i$ )
       $active_{i'} = true$ 
    END LOOP
  END LOOP
  SOLVE MILP model (29) – (31), (34) – (37)
  LOOP ( $i \in I^b$  and  $iter = Seq_i$ )
    LOOP ( $s \in S^b$ )
      LOOP ( $k \in K_s$ )
        FIX variable  $Y_{ik}$ 
      END LOOP
    END LOOP
    LOOP ( $i' \in SB_i$ )
      LOOP ( $s \in S^{sb}$ )
        LOOP ( $k \in K_s$ )
          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 ( $(i, i') \in I^b$  and  $Seq_i < Seq_{i'}$ )
  LOOP ( $s \in S^b$ )
    IF ( $i < i'$ ) THEN
       $sW_{ii's} = 1$ 
    ELSE
       $sW_{ii's} = 0$ 
    ENDIF
  END LOOP
END LOOP
LOOP ( $(i, i') \in I^{sb}$  and  $Seq_i \leq Seq_{i'}$ )
  LOOP ( $s \in S^{sb}$ )
    IF ( $i < i'$ ) THEN
       $sW_{ii's} = 1$ 
    ELSE
       $sW_{ii's} = 0$ 
    ENDIF
  END LOOP
END LOOP

```

Fig. 5 Pseudo-code for the constructive step

rescheduled at each iteration. In this case, the lowest number is adopted, i.e. one at a time, in order to reduce at minimum the resolution times of the MILP model. Thus, the number of iterations in each improvement loop is equal to the number of blocks: $|I^b|$.

The pseudo-code for the improvement step is given in Fig. 6. Firstly, 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^b following the sequence parameter Seq_i ; the parameter *iter* indicates the next block that will be rescheduled. When a block $i \in I^b$ is chosen, the boolean parameter $active_i$ is set to true for the block i and its sub-assemblies $i' \in SB_i$. The MILP model (29)–(36) can take reassignment and resequencing decisions just on products $i \in I$ with $active_i = true$. For the remaining products, those with $active_i = false$, only timing variables (Ts_{is} and Tf_{is}) can be modified. Reassignment to other units is not allowed for blocks (and sub-blocks) with parameter $active_i$ set to false. Furthermore, their relative position in the processing sequence remains unchanged. The binary variables Y_{ik} and $W_{i't_s}$ are fixed for the products with $active_i = false$. This resolution strategy reduces drastically the number of binary variables with regards to the full space approach, which is equivalent to solve the MILP model (29)–(36) with $active_i = true \forall i \in I$. Every time a rescheduling iteration terminates, the current solution is updated.

Once the rescheduling step is applied for all blocks, the procedure checks the improvement reached for the objective function. If the current makespan (*CurrentSol*) is better than the best solution found until now (*BestSol*), the algorithm updates the makespan ($BestSol = CurrentSol$) and goes to start the rescheduling iterations again. Otherwise, the solution procedure ends and reports the best solution found for the problem under study.

6 Computational results and discussion

In this section, several problem instances are introduced in order to evaluate and compare the computational performance of the rigorous MILP models and the decomposition procedure. These testing examples arise in the shipbuilding industry for the development of marine resources, specifically for the offshore oil and gas industry. The aim is to validate the performance of the proposed decomposition algorithm and compare its computational efficiency with regards to the exact optimization approaches when the same examples are solved.

The experimentation section is organized as follows. Firstly, ten problem instances with different features are presented. Afterwards, the inherently complexity of problems addressed and model statistics for the exact optimization approaches are shown. Finally, the MILP-based strategy is tested on all examples and a comparison between the alternative solution approaches is shown. All solution strategies are implemented in GAMS 24.9.2 with CPLEX 12.7.1.0 as MIP solver and run on a PC with four-core Intel Xeon X5650 Processor (2.6 GHz). The termination criterion imposed for each MIP solver execution was either 0% integrality gap or 3600 s of CPU time.

```

Set  $BestSol = M$ 
WHILE  $CurrentSol < BestSol$ 
   $BestSol = CurrentSol$ 
   $iter = 1$ 
  WHILE  $iter = Seq_i$  and  $i \in I^b$ 
     $active_i = false$ 
    LOOP ( $i \in I^b$  and  $iter = Seq_i$ )
      RELEASE variables  $Y_{ik}$ 
       $active_i = true$ 
      LOOP ( $i' \in SB_i$ )
        RELEASE variables  $Y_{ik}$ 
         $active_{i'} = true$ 
      END LOOP
    END LOOP
    SOLVE MILP model (29) – (36)
     $CurrentSol = MK$ 
    LOOP ( $i \in I^b$  and  $iter = Seq_i$ )
      LOOP ( $s \in S^b$ )
        LOOP ( $k \in K_s$ )
          FIX variable  $Y_{ik}$ 
        END LOOP
      END LOOP
      LOOP ( $i' \in SB_i$ )
        LOOP ( $s \in S^{sb}$ )
          LOOP ( $k \in K_s$ )
            FIX variable  $Y_{i'k}$ 
          END LOOP
        END LOOP
      END LOOP
    END LOOP
    LOOP ( $(i, i') \in I^b$  and  $i < i'$  and
      ( $active_i = true$  or  $active_{i'} = true$ ))
      LOOP ( $s \in S^b$ )
        FIX variable  $W_{ii's}$ 
      END LOOP
    END LOOP
    LOOP ( $(i, i') \in I^{sb}$  and  $i < i'$  and
      ( $active_i = true$  or  $active_{i'} = true$ ))
      LOOP ( $s \in S^{sb}$ )
        FIX variable  $W_{ii's}$ 
      END LOOP
    END LOOP
  END WHILE
   $iter = iter + 1$ 
END WHILE
END WHILE

```

Fig. 6 Pseudo-code for the improvement step

6.1 Real-world case study

The real-world case of study arises in a shipyard with 41 workshops and one dry dock, which are used for the construction of the vessels. The manufacturing process involves 7 processing stages ($s \in S$), each of them with K_s processing units working in parallel, as shown in Fig. 7. Note that the last processing stage s_7 has just one unit, representing the dry dock where the ship is mounted. An interesting feature of this manufacturing process is that there are two assembly stages: in the first one, stage s_3 , each block is formed by welding one or more sub-blocks while in the second one, stage s_7 , the mounting of these blocks is carried out to build the ship. As depicted Fig. 7, every block can be formed by a different quantity of sub-assemblies, for instance, the block i_{n+1} is formed by two sub-blocks (i_1 and i_2), the block i_{n+2} is made from just one sub-block (i_3), while the block i_{n+3} has three sub-assemblies (i_4 , i_5 , and i_6). Other problem characteristic is that the processing times at each stage depend on the type of product that is processed. It is worth to remark that due to confidentiality issues, we cannot disclose the data of this real-world case study.

The Gantt chart of the optimal schedule for a toy example involving the processing of 8 sub-blocks and 4 blocks is given in Fig. 8. In this picture, each block $i \in I^b$ and its sub-assemblies $i' \in SB_i$ are depicted with the same color. In the right of Fig. 8, it is shown as the timing constraints (34) and (35) of the general precedence formulation are used for enforcing the assembly operations. Note that, for instance,

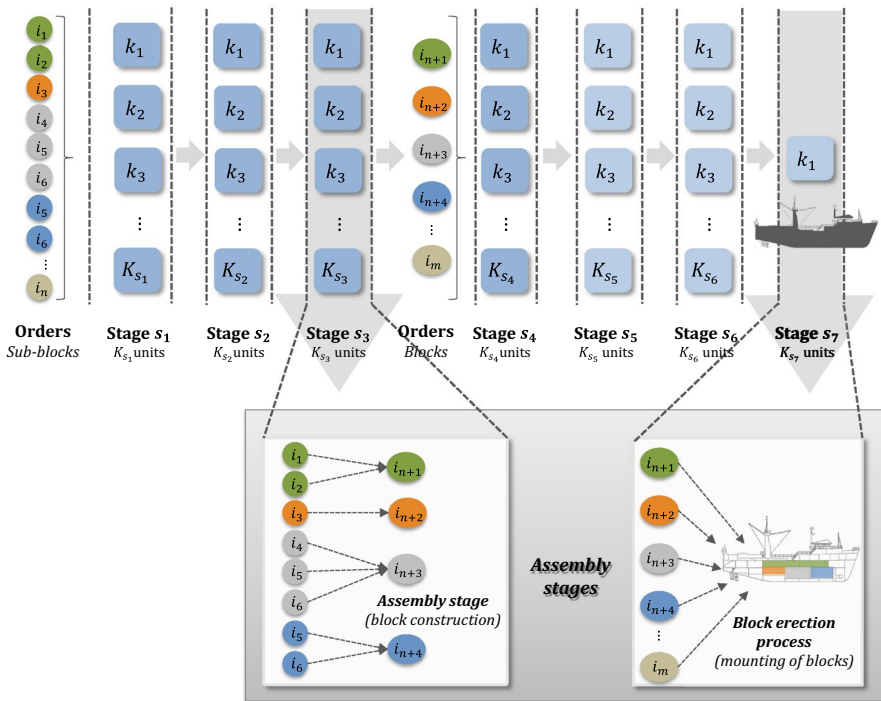


Fig. 7 The Shipbuilding process with assembly and processing stages

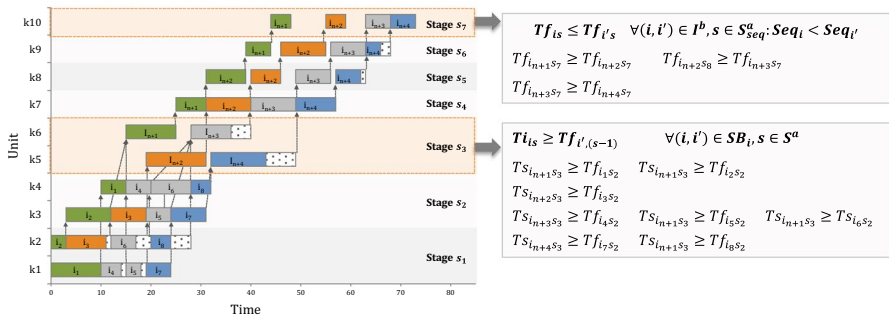


Fig. 8 Gantt chart of a block assembly problem involving 8 sub-blocks and 4 blocks

the earliest starting time of block i_{n+3} at stage s_3 should be greater than the finishing times of its sub-assemblies $i_4, i_5,$ and i_6 at stage s_2 .

6.2 Problem instances

The applicability and effectiveness of the proposed solution strategy are illustrated by efficient solving a set of realistic instances derived from the original case study presented above. Since both the quality of the solution and the computational time to find it depend on the number of products (blocks and sub-blocks) involved in the scheduling problem, 10 examples are defined by varying both the amount of products to be processed and the storage policy adopted by the scheduler (NIS or UIS). All problem instances are summarized in Table 1. The expression $N \times M$ refers to a ship constructed with N blocks and M sub-blocks. The simplest problem addressed involves 10 sub-blocks and 5 blocks, while the biggest one deals with a ship that is built from 50 sub-blocks and 25 blocks. For all examples, it is assumed that each block has two sub-assemblies. This means that twice as many products go through the processing stages $s \in S^{sb}$ regarding to stages $s \in S^b$. The amount of products

Table 1 Problem instances

Problem	Blocks \times sub-blocks ($N \times M$)	Storage policy
P.01	5 \times 10	UIS
P.02	5 \times 10	NIS
P.03	10 \times 20	UIS
P.04	10 \times 20	NIS
P.05	15 \times 30	UIS
P.06	15 \times 30	NIS
P.07	20 \times 40	UIS
P.08	20 \times 40	NIS
P.09	25 \times 50	UIS
P.10	25 \times 50	NIS

$(N \times M)$ to be assigned and sequenced in the available processing units is closely related to the model complexity, which directly impacts on the resolution times. The optimization goal is the minimization of makespan for all problem instances.

6.3 Testing the exact optimization approaches

In order to underline the complexity of the problem addressed, all examples are first solved by using the three rigorous MILP model proposed in this paper. The best solution found within the predefined time limit of 1 CPU hour (3600 CPUs), the integrality gap and the CPU time requirement of each alternative are given in Table 2. In addition, Table 3 reports the problem sizes in terms of binary and continuous variables, and linear constraints. Note that the number of binary variables is strongly incremented by increasing the number of products to be processed. The information provided by Tables 2 and 3 allows making a comparative analysis between the alternative models, highlighting the resolution complexity of the instances considered when they are solved with full-space approaches.

From Table 2, it follows that the optimal solution is reported by the MILP models just for the least size instances (P.01 and P.02). In the remaining examples, the MIP solver terminates without reaching the optimal solution because the time limit is exceeded. For these cases, feasible but poorly solutions are reported. Note that the bigger size of the model, the higher the integrality gap. Also, the worse results are obtained for examples considering NIS policy when comparing them with those ones reported for the UIS policy.

The worst computational performance is exhibited by the unit-specific direct precedence formulation. For examples P.06, P.08, and P.10, this approach is not even capable of reporting a feasible solution within the time limit. In contrast, the other two mathematical formulations are at least capable of returning a suboptimal solution with certain integrality gap within the predefined CPU time. For remaining examples, the immediate precedence formulation shows bigger integrality gaps than the other two models. This situation is directly related to the high number of binary variables used by the formulation.

On the other hand, Table 2 also shows that the general precedence formulation exhibits a slightly better computational performance than the time-slots based model, up to 3%, in all examples considered. Particularly, a solution of 319.1 days with an integrality gap of 47.4% is reported by the general precedence approach for the most complex example P.10 (25 blocks, 50 sub-blocks and NIS policy), featuring a model size of 61,376 equations, 31,700 binary variables, and 502 continuous variables (see Table 3), while the time-slots based model (140,277 equations, 17,375 binary variables, and 18,866 continuous variables) reaches a solution of 321.4 days with an integrality gap of 47.7% after 3600 s of CPU time.

Despite the high integrality gap reported by the general precedence model for problem instance P.10, it is worth to remark that the solution found (319.1 days) improves considerably the time currently employed by the shipyard for delivering a ship of size 25×50 . A few years ago, the production schedule used by the company featured a makespan of about a year and a half. Afterwards, this

Table 2 Computational statistics for the three alternative exact optimization approaches

Problem	Slot-based continuous time formulation			Unit-specific direct precedence formulation			Global general precedence formulation		
	MILP sol.	GAP (%)	CPU time (s)	MILP sol.	GAP (%)	CPU time (s)	MILP sol.	GAP (%)	CPU time (s)
P.01	126.3	0	8.1	126.3	0	2.1	126.3	0	2.3
P.02	126.3	0	21.9	126.3	0	2.1	126.3	0	2.1
P.03	160.7	13.1	3600	170.6	18.0	3600	160.1	12.7	3600
P.04	162.8	13.4	3600	161.2	13.3	3600	161.4	13.4	3600
P.05	207.0	25.9	3600	258.8	40.8	3600	202.4	24.3	3600
P.06	211.5	27.5	3600	NA ^a	–	3600	210.6	27.2	3600
P.07	231.1	27.8	3600	425.8	60.8	3600	229.2	27.2	3600
P.08	240.4	30.6	3600	NA ^a	–	3600	240.2	30.6	3600
P.09	299.8	40.0	3600	823.1	78.1	3600	290.8	37.9	3600
P.10	321.4	47.7	3600	NA ^a	–	3600	319.1	47.4	3600

^aA feasible solution has not been generated within 3600 CPUs

Table 3 Model sizes for the exact optimization approaches

Problem	Slot-based continuous time formulation			Unit-specific direct precedence formulation			Global general precedence formulation		
	Bin Vars	Cont Vars	Eqs	Bin Var	Cont Vars	Eqs	Bin Var	Cont Vars	Eqs
P.01	485	871	4441	2780	7397	21,907	1390	102	2326
P.02	485	871	4441	2780	7397	21907	1390	102	2326
P.03	2130	2832	17,876	10,510	28,742	86,547	5255	202	9626
P.04	2130	2832	17,876	10,510	28,742	86,547	5255	202	9626
P.05	5550	6715	45,931	23,190	64,037	193,962	11,595	302	21,901
P.06	5550	6715	45,931	23,190	64,037	193,962	11,595	302	21,901
P.07	10,720	12,266	87,571	40,820	113,282	344,152	20,410	402	39,151
P.08	10,720	12,266	87,571	40,820	113,282	344,152	20,410	402	39,151
P.09	17,375	18,866	140,277	64,650	176,727	585,867	31,700	502	61,376
P.10	17,375	18,866	140,277	64,650	176,727	585,867	31,700	502	61,376

total time was reduced to nearly 1 year by using the simulation models that were developed by Basán et al. (2017) and Cebral-Fernandez et al. (2016).

Taking into account that the shipbuilding is a long-term production process, involving several months, it is not strictly necessary to have a solution strategy that provides a good solution in less than 1 h of CPU time. When solving the realistic example P.10 without setting a time limit, the solution found by the time-slots formulation is 288.6 days with an integrality gap of 37.5%, while the general precedence model reaches a makespan of 284.7 days with an integrality gap of 36.6%. In both cases, the MIP solver ended after several hours of CPU time because of “*out of memory*” error. This solution represents a significant saving for the company because the total time required for delivering a vessel is reduced up to approximately 20%, from 1 year to 285 days. The best solution found by the general precedence model for example P.10 is shown in Fig. 9. In this picture, each block $i \in I^b$ and its sub-assemblies $i' \in SB_i$ 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 .

Despite the good solutions found by the exact optimization approaches (excepting the immediate precedence formulation) with regards to other solution strategies, the computational results given in Tables 2 and 3 demonstrate that the main drawback of continuous time models is their high computation time. However, the robustness offered by these MILP models can be exploited resolving them in a decomposable way in order to find better solutions in considerably less computational time.

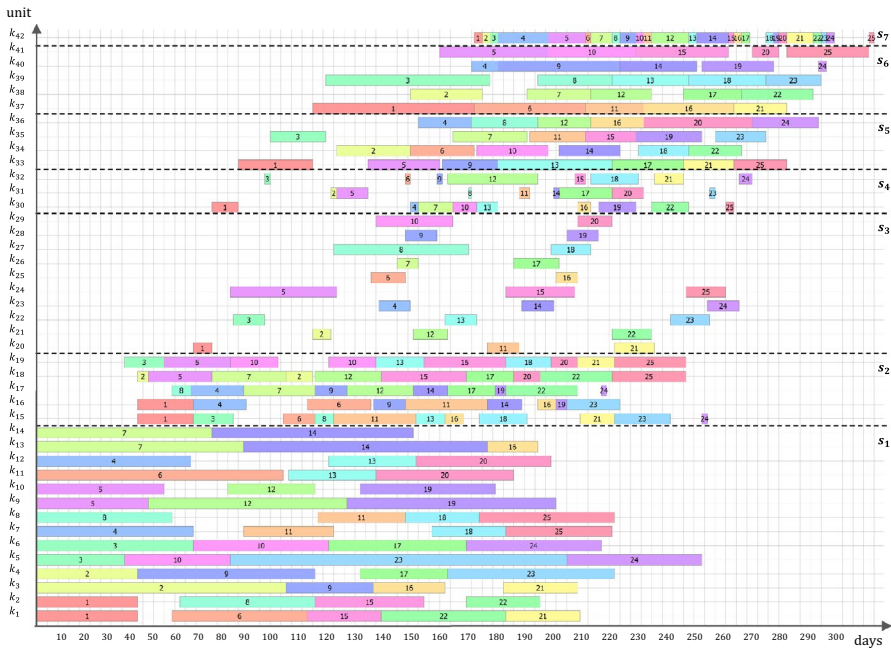


Fig. 9 Gantt chart of the best solution found by the general precedence approach for example P.10 (problem structure 25 × 50 under NIS policy)

Table 4 Improvement-based algorithm solutions found

Problem	Constructive stage		Improvement stage			Improvement percent (%)
	Stage 1 solution	CPU time (s)	Best solution	Total CPU time (s)	Iter.	
P.01	144.0	0.6	126.3	2.3	3	12.3
P.02	144.4	0.7	126.3	2.1	3	12.5
P.03	176.3	1.5	160.0	11.1	6	9.2
P.04	177.9	1.5	160.3	6.8	4	9.9
P.05	239.1	2.1	200.2	16.3	6	16.3
P.06	241.9	1.9	202.7	24.1	10	16.2
P.07	248.8	2.8	221.0	37.6	15	11.2
P.08	255.4	3.1	228.6	41.2	13	10.5
P.09	301.1	4.6	262.8	49.8	11	12.7
P.10	298.3	4.3	270.5	56.6	15	9.3

6.4 Testing the iterative MILP-based algorithm

For developing a comparative analysis, the ten examples proposed in Table 1 have been also solved through the MILP-based decomposition strategy. The computational results reported by the algorithm are detailed in Table 4. For every example solved, this table shows the best solution found at each algorithmic stage (*MK*) together with the accumulated CPU time. The number of improvement iterations and the percent of enhancement obtained from the starting solution are also given in Table 4.

As can be seen in Table 4, the heuristic of scheduling the products one by one following the assembly sequence Seq_i presents a good performance and is capable of generating a good initial solution in less of 5 s of CPU time for all examples solved. Moreover, these starting solutions are always improved by the algorithm in the second stage, consuming in total less than 60 s of computational time. The constructive stage solution is always enhanced at the improvement stage, obtaining an improvement percentage of around 12%, in average. For industrial-size example P.10 dealing with 25 blocks, 50 sub-blocks, and a NIS policy, the constructive step converges to a solution of 298.3 days, which is illustrated in Fig. 10. 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 given in Fig. 10, the final solution

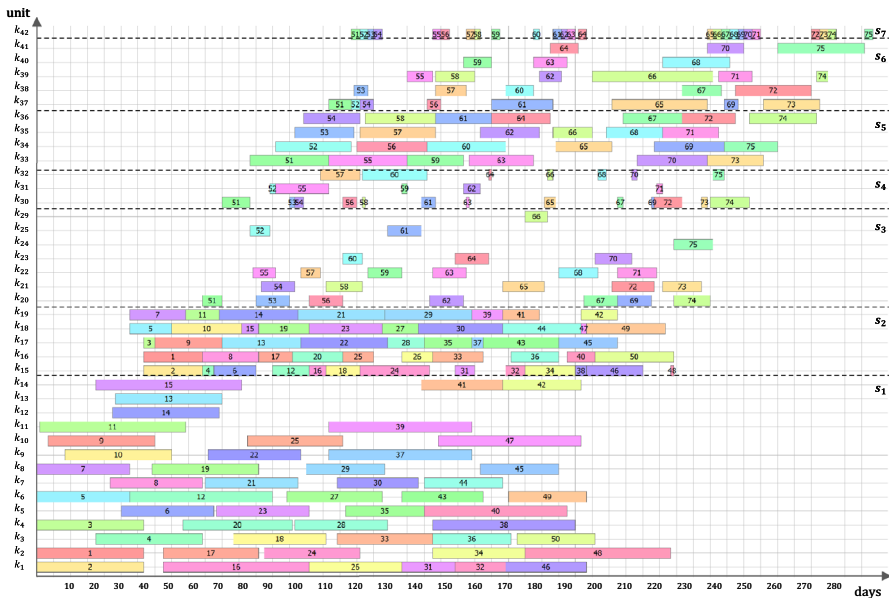


Fig. 10 Gantt chart of the initial schedule reported by the MILP-based strategy for P.10 (problem structure 25×50 under NIS policy)

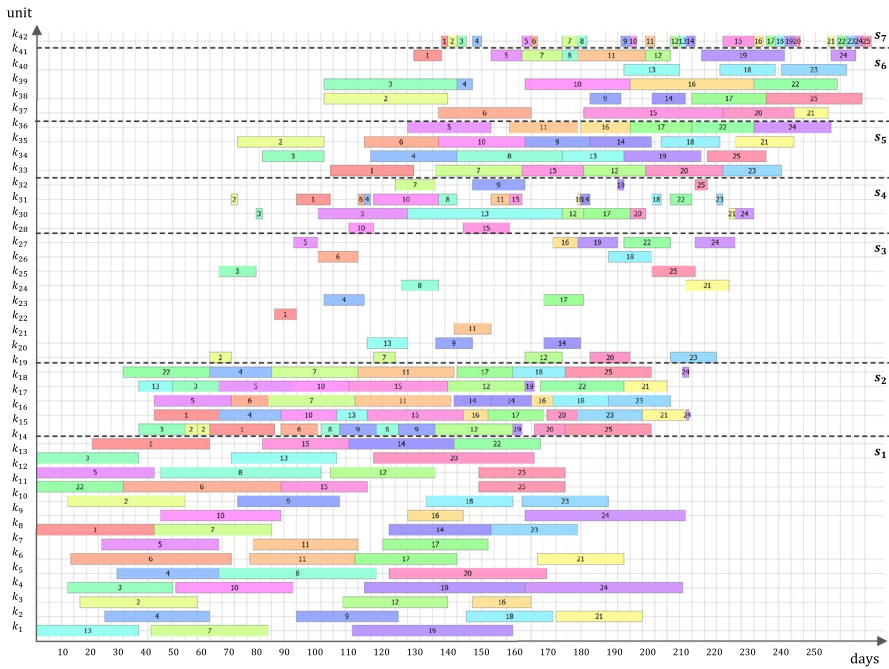


Fig. 11 Gantt chart of the best schedule reported by the MILP-based strategy for P.10 (problem structure 25×50 under NIS policy)

Table 5 Comparison between the exact MILP formulation and the decomposition method

Problem	Global general precedence formulation			Decomposition algorithm		Percent of enhancement (%)
	Solution	GAP	CPU time (s)	Solution	CPU time (s)	
P.01	126.3	0	2.3	126.3	2.3	0
P.02	126.3	0	2.2	126.3	2.1	0
P.03	160.1	12.7	3600	160.0	11.1	0.1
P.04	161.4	13.4	3600	160.3	6.8	0.7
P.05	202.4	24.3	3600	200.2	16.3	1.1
P.06	210.6	27.2	3600	202.7	24.1	3.8
P.07	229.2	27.2	3600	221.0	37.6	3.6
P.08	240.2	30.6	3600	228.6	41.2	4.8
P.09	290.8	37.9	3600	262.8	49.8	9.6
P.10	319.1	47.4	3600	270.5	56.6	15.2

depicted in Fig. 11 is reported by the procedure. The makespan is enhanced 9.3% from 298.3 to 270.5 days.

On the other hand, a comparative table between the best results obtained through the exact optimization strategy and the decomposition algorithm for all examples considered is given in Table 5. Specifically, the results reported by the global general precedence based model are used in this table because this formulation shows the best computational performance regarding to the other two exact alternatives (see Table 2).

As can be seen in Table 5, the decomposition strategy converges to the optimal solution in examples P.01 and P.02, validating the solution approach. For the other cases, the proposed algorithm reports, in few seconds of CPU time, better solutions than those found by the MILP model after 3600 s of computational time. For realistic example P.10, the global general precedence approach reports a solution featuring a makespan of 319.1 days with an integrality gap of 37.9% after 1 h of CPU time, while the decomposition algorithm reports a makespan of 270.5 days in just 56.6 CPU seconds. Moreover, this solution outperforms that one obtained after several hours of CPU time when the MILP model is executed without setting a time limit (see Fig. 9). The schedule given in Fig. 11 represents the best solution found for problem instance P.10; a reduction of approximately 26% is achieved in the total makespan with respect to the currently schedule used by the company.

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 15.2% 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 1 month of work in the productive system and hence, a significant savings are obtained by the company, avoiding penalties for delay in the delivery of the vessels. 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.

7 Conclusions

This paper has presented a systematic MILP-based iterative algorithm for solving a complex scheduling problem of the shipbuilding industry. Moreover, this proposal is arisen after than three alternative rigorous MILP formulations, which are based on the notion of time-slot, global general precedence, and unit-specific direct precedence were developed for representing and solving the problem under study.

From the experimentation developed, it was demonstrated that the rigorous formulations converged to the optimal solution only when small problem instances are considered. The computational efficiency of these full-space approaches is rapidly deteriorated as the problem size increased. In order to overcome this limitation, an efficient hybrid solution strategy, which has a MILP scheduling formulation as its core, is proposed for the solution of this type of challenging real-world problem. It combines the robustness of the traditional MILP approach based on general precedence formulation with the advantages of heuristic procedures. The solution

strategy is based on a systematic decomposition methodology to first obtain an initial solution (constructive stage), and then gradually enhancing it by applying several rescheduling iterations (improvement stage).

The performance of the proposed methodology has been deeply evaluated by solving several instances arising in a shipbuilding industry, comparing the results obtained with those ones found by the exact MILP models. Computational results showed that feasible and good-quality solutions can be efficiently found in short computational time by the decomposition algorithm, outperforming the pure exact optimization approaches and other solution techniques published in the literature.

It would be of great interest for us to compare the results of our proposal with other search techniques, such as genetic-algorithm-based techniques, among others. Although the data of the real-world case of study has not been provided in this paper by confidentiality issues, the problem structure is clearly explained throughout the manuscript, enabling to other authors to prove other solution techniques using data that can be generated randomly. Moreover, as the MILP formulations used in this paper can be adapted to any type of multi-stage scheduling problem, we also invite our colleagues to apply the decomposition strategy for solving other practical situations.

Acknowledgements The authors gratefully acknowledge the financial support from CONICET under Grant PIP 112 20150100641, from Universidad Nacional del Litoral under Grant CAI+D 2016 PIC 50420150100101LI and from ANPCyT under Grant PICT-2014-2392.

References

- Aguirre AM, Méndez CA, Gutierrez G, De Prada C (2012) An improvement-based MILP optimization approach to complex AWS scheduling. *Comput Chem Eng* 47:217–226. <https://doi.org/10.1016/j.compchemeng.2012.06.036>
- Basán NP, Achkar VG, Méndez CA, Garcia-del-valle A (2017) A heuristic simulation-based framework to improve the scheduling of blocks assembly and the production process in shipbuilding. *Winter Simul Conf WSC F134102:3218–3229*. <https://doi.org/10.1109/WSC.2017.8248040>
- Basán NP, Achkar VG, Garcia-del-valle A, Méndez CA (2018) An effective continuous-time formulation for scheduling optimization in a shipbuilding. *Iberoam J Ind Eng* 10:34–48
- Castro PM, Harjunkoski I, Grossmann IE (2009) New continuous-time scheduling formulation for continuous plants under variable electricity cost. *Ind Eng Chem Res* 48:6701–6714. <https://doi.org/10.1021/ie900073k>
- Cebal-Fernandez M, Crespo-Pereira D, Garcia-Del-Valle A, Rouco-Couzo M (2016) Improving planning and resource utilization of a shipbuilding process based on simulation. In: 28th European Modeling and Simulation Symposium, EMSS 2016, pp 197–203
- Cerdá J, Henning GP, Grossmann IE (1997) A Mixed-integer linear programming model for short-term scheduling of single-stage multiproduct batch plants with parallel lines. *Ind Eng Chem Res* 36:1695–1707. <https://doi.org/10.1021/ie9605490>
- Cho KK, Oh SJ, Ryu KR, Choi HR (1998) An integrated process planning and scheduling system for block assembly in shipbuilding. *CIRP Ann Manuf Technol* 47:419–422
- Chu Y, You F, Wassick JM (2014) Hybrid agent-based method for scheduling of complex batch processes. *Comput Chem Eng* 60:277–296. <https://doi.org/10.1109/ACC.2014.6858592>
- Cóccola ME, Cafaro VG, Méndez CA, Cafaro DC (2014) Enhancing the general precedence approach for industrial scheduling problems with sequence-dependent issues. *Ind Eng Chem Res* 53:17092–17097. <https://doi.org/10.1021/ie500803p>

- Cóccola ME, Dondo R, Méndez CA (2015) A MILP-based column generation strategy for managing large-scale maritime distribution problems. *Comput Chem Eng* 72:350–362. <https://doi.org/10.1016/j.compchemeng.2014.04.008>
- Fattaka S, Thibault J, Gupta Y (2015) A new algorithm using front prediction and NSGA-II for solving two and three-objective optimization problems. *Optim Eng* 16:713–736. <https://doi.org/10.1007/s11081-014-9271-9>
- Hasebe S, Hashimoto I, Ishikawa A (1991) General reordering algorithm for scheduling of batch processes. *J Chem Eng Jpn* 24:483–489. <https://doi.org/10.1252/jcej.24.483>
- Kim H, Kang J, Park S (2002) Scheduling of shipyard block assembly process using constraint satisfaction problem scheduling of shipyard block assembly process using constraint satisfaction problem. *Asia Pac Manag Rev* 7:119–138
- Koh S, Eom C, Jang J, Choi Y (2008) An improved spatial scheduling algorithm for block assembly shop in shipbuilding company. In: 2008 3rd international conference on innovative computing information and control. IEEE, pp 253–253
- Kopanos GM, Laínez JM, Puigjaner L (2009) An efficient mixed-integer linear programming scheduling framework for addressing sequence-dependent setup issues in batch plants. *Ind Eng Chem Res* 48:6346–6357. <https://doi.org/10.1021/ie801127t>
- Kopanos GM, Méndez CA, Puigjaner L (2010) MIP-based decomposition strategies for large-scale scheduling problems in multiproduct multistage batch plants: a benchmark scheduling problem of the pharmaceutical industry. *Eur J Oper Res* 207:644–655. <https://doi.org/10.1016/j.ejor.2010.06.002>
- Larson J, Wild SM (2016) A batch, derivative-free algorithm for finding multiple local minima. *Optim Eng* 17:205–228. <https://doi.org/10.1007/s11081-015-9289-7>
- Lee K, Shin JG, Ryu C (2009) Development of simulation-based production execution system in a shipyard: a case study for a panel block assembly shop. *Prod Plan Control* 20:750–768. <https://doi.org/10.1080/09537280903164128>
- Maravelias CT, Sung C (2009) Integration of production planning and scheduling: overview, challenges and opportunities. *Comput Chem Eng* 33:1919–1930. <https://doi.org/10.1016/j.compchemeng.2009.06.007>
- Méndez CA, Cerdá J (2003a) Dynamic scheduling in multiproduct batch plants. *Comput Chem Eng* 27:1247–1259. [https://doi.org/10.1016/S0098-1354\(03\)00050-4](https://doi.org/10.1016/S0098-1354(03)00050-4)
- Méndez CA, Cerdá J (2003b) An MILP continuous-time framework for short-term scheduling of multipurpose batch processes under different operation strategies. *Optim Eng* 4:7–22. <https://doi.org/10.1023/A:1021856229236>
- Méndez C, Henning G, Cerdá J (2000) Optimal scheduling of batch plants satisfying multiple product orders with different due-dates. *Comput Chem Eng* 24:2223–2245. [https://doi.org/10.1016/S0098-1354\(00\)00584-6](https://doi.org/10.1016/S0098-1354(00)00584-6)
- Méndez CA, Henning GP, Cerdá J (2001) An MILP continuous-time approach to short-term scheduling of resource-constrained multistage flowshop batch facilities. *Comput Chem Eng* 25:701–711. [https://doi.org/10.1016/S0098-1354\(01\)00671-8](https://doi.org/10.1016/S0098-1354(01)00671-8)
- Méndez CA, Cerdá J, Grossmann IE et al (2006) State-of-the-art review of optimization methods for short-term scheduling of batch processes. *Comput Chem Eng* 30:913–946. <https://doi.org/10.1016/j.compchemeng.2006.02.008>
- Persson JA, Ölvander J (2015) Optimization of the complex-RFM optimization algorithm. *Optim Eng* 16:27–48. <https://doi.org/10.1007/s11081-014-9247-9>
- Pinto JM, Grossmann IE (1995) A continuous time mixed integer linear programming model for short term scheduling of multistage batch plants. *Ind Eng Chem Res* 34:3037–3051. <https://doi.org/10.1021/ie00048a015>
- Roslöf J, Harjunkoski I, Westerlund T, Isaksson J (1999) A short-term scheduling problem in the paper-converting industry. *Comput Chem Eng* 23:S871–S874. [https://doi.org/10.1016/S0098-1354\(99\)80214-2](https://doi.org/10.1016/S0098-1354(99)80214-2)
- Roslöf J, Harjunkoski I, Björkqvist J et al (2001) An MILP-based reordering algorithm for complex industrial scheduling and rescheduling. *Comput Chem Eng* 25:821–828. [https://doi.org/10.1016/S0098-1354\(01\)00674-3](https://doi.org/10.1016/S0098-1354(01)00674-3)
- Roslöf J, Harjunkoski I, Westerlund T, Isaksson J (2002) Solving a large-scale industrial scheduling problem using MILP combined with a heuristic procedure. *Eur J Oper Res* 138:29–42. [https://doi.org/10.1016/S0377-2217\(01\)00140-0](https://doi.org/10.1016/S0377-2217(01)00140-0)
- Schweiger J, Liers F (2018) A decomposition approach for optimal gas network extension with a finite set of demand scenarios. *Optim Eng* 19:297–326. <https://doi.org/10.1007/s11081-017-9371-4>

- Seo Y, Sheen D, Kim T (2007) Block assembly planning in shipbuilding using case-based reasoning. *Expert Syst Appl* 32:245–253. <https://doi.org/10.1016/j.eswa.2005.11.013>
- Shang Z, Gu J, Ding W, Duodu EA (2017) Spatial scheduling optimization algorithm for block assembly in shipbuilding. *Math Probl Eng* 2017:1–10. <https://doi.org/10.1155/2017/1923646>
- Shin K, Ciccantell PS (2009) The steel and shipbuilding industries of south korea: rising east ASIA and globalization. *Statew Agric Land Use Baseline XV*:167–192. <https://doi.org/10.1017/cbo9781107415324.004>
- Tenne Y (2015) An adaptive-topology ensemble algorithm for engineering optimization problems. *Optim Eng* 16:303–334. <https://doi.org/10.1007/s11081-014-9260-z>
- Xiong F, Xing K, Wang F (2015) Scheduling a hybrid assembly-differentiation flowshop to minimize total flow time. *Eur J Oper Res* 240:338–354. <https://doi.org/10.1016/j.ejor.2014.07.004>
- Zhuo L, Huat DCK, Wee KH (2012) Scheduling dynamic block assembly in shipbuilding through hybrid simulation and spatial optimisation. *Int J Prod Res* 50:5986–6004. <https://doi.org/10.1080/00207543.2011.639816>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.