Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

A unifying framework and a mathematical model for the Slab Stack Shuffling Problem

Giuseppe Bruno[a], Manuel Cavola[a*], Antonio Diglio[a] and Carmela Piccolo[a]

[a]*Università degli Studi di Napoli Federico II, Department of Industrial Engineering-DII, Italy*

C H R O N I C L E

Article history:
Received October 3 2022
Accepted October 28 2022
Available online
October, 28 2022

Keywords:
Slab stack shuffling
Steel industry
Shipbuilding
Warehouse management

A B S T R A C T

The Slab Stack Shuffling Problem (SSSP) consists of retrieving slabs, stored in stacks in a warehouse, to efficiently satisfy a processing order. The problem is relevant in the steel industry as the slab yard serves as a storage buffer between the continuous casting stage and the rolling mill. Notably, the SSSP also arises in cutting/assembly centres within the shipbuilding supply chain, where already rolled slabs must undergo further production stages. The different slabs managed in these facilities confer the problem novel practical features, such as the existence of slabs' typologies and deadlines, i.e., a maximum time beyond which their quality certifications expire and are no longer usable. In such a context, the goals of the present paper are twofold: (i) providing a comprehensive taxonomy of the main aspects involved in the problem; (ii) proposing an original mathematical formulation for the SSSP. Specifically, the model is cast as a bi-objective multi-period program, seeking to minimise the number of shuffles and expired slabs. Computational tests on randomly generated instances prove the relevance of the trade-off between the above-mentioned objectives and the impact of the yard's configuration on the retrieval process, suggesting the most suitable storage strategy to adopt under different operational settings.

1. Introduction

Stacking Problems (SPs) are a broad family of optimisation problems occurring in warehouses where *items* are stored in *stacks* (i.e., positioned on top of each other) and handled by cranes to fulfil an *order*. The order, in practice, lists the items to be picked up for further processing. As items are stacked in a last-in-first-out fashion, cranes can directly access only the upper item of each stack. Hence, if a lower item has to be retrieved, those placed upon it must be moved (temporarily or permanently) to other stacks. These operations are referred to as *shuffles* (or *reshuffles* or *relocations*). Since the latter are usually very time-consuming, SPs mainly involve optimising the shuffling workload or, more in general, an efficiency measure associated with the retrieval process (e.g., time and costs). SPs are particularly relevant in the steel industry, where they have been widely applied in port logistics to tackle, for instance, container handling (see Kim et al., 2000; Dekker et al., 2007). For an overview of models and solution methods for SPs, interested readers can refer to Caserta et al. (2012, 2020) and Lehnfeld and Knust (2014). As the latter reference outlines, changing operating conditions lead to different optimisation problems. *Loading Problems (LPs)*, for instance, deal with the storage of *incoming* items in the warehouse, seeking to minimise the expected shuffling workload associated with the retrieval process (the order is typically unknown when taking storage decisions – see, e.g., Jang et al., 2013). *Unloading Problems (UPs)*, instead, concern the retrieval process of items from the warehouse (*outgoing* items) when an actual processing order is disclosed (Forster and Bortfeldt, 2012; Kim et al., 2016). Therefore, in their generic setting, UPs aim to define the sequence in which items are retrieved to satisfy the order at the least shuffling workload. Note that *Combined Problems (CPs)* involving both loading and unloading operations are also considered in the literature (Avriel and Penn, 1993; Ding and Chou, 2015). Finally, *Premarshalling Problems (PPs)* occur if items are relocated across stacks in order to be retrieved following the sequence expressed in the order without additional shuffles (Bortfeldt and Forster, 2012; Huang and Lin, 2012).

* Corresponding author
E-mail: manuel.cavola@unina.it (M. Cavola)
ISSN 1923-2934 (Online) - ISSN 1923-2926 (Print)
2023 Growing Science Ltd.
doi: 10.5267/j.ijiec.2022.10.005

The present work focuses on UPs and, specifically, one of its variants, i.e., the *Slab Stack Shuffling Problem* (SSSP). In this problem, we consider a storage area, i.e., the slab yard, divided into spans and pitches, with each pitch hosting a slab stack. The distinctive feature of the SSSP is that the item to be retrieved is not a single and unique element and does not explicitly refer to a physical object per se (e.g., a container). Here, an item indicates a generic order request for a slab that must be chosen within a restricted set. The latter is known with the order information and is typically denoted as *candidate slabs set*. Hence, the problem consists of identifying *target slabs* satisfying the order (that is, slabs chosen among the candidate ones) and retrieving them at the least shuffling workload. In SSSPs, in practice, decisions pertain not only to the retrieval sequence to be performed, but also (and most importantly) to the specific elements to be retrieved.

Despite its importance, the SSSP has received little scholars' attention: indeed, to the best of our knowledge, only a few studies, that we detail in the next section, are found in the literature. Drawing upon the seminal paper by Tang et al. (2001), extant contributions focused on the proposal of more refined formulations and solution methods (Tang et al. 2002; Singh and Tiwari, 2004; Fernandes et al., 2012; Shi and Liu, 2021; Rajabi et al., 2022) or variants of the problem at hand (Cheng and Tang, 2010; Tang and Ren, 2010; Ren and Tang, 2010).

The SSSP naturally arises between the continuous casting stage and the hot rolling mill, which is the primary application context explored in the literature. Notably, the SSSP also emerges in cutting/assembly centres or shipyards (see Wang et al., 2008; Deng, 2014; Liyun et al., 2020), where stacked slabs that already underwent the rolling phase must be handled for the subsequent production stages. This problem mainly occurs in the shipbuilding sector, as slabs are the primary input materials for constructing ships' hulls. Shipbuilding plays a pivotal role in many countries' economies (Ferrari, 2012), and, as recent data highlight, it keeps this status regardless of the effects of the Covid-19 pandemic (BRS Group, 2021). Hence, the SSSP is of crucial importance in this industry since it could affect the operational and economic performance of the entire hull construction supply chain.

Interestingly, addressing SSSPs in cutting/assembly centres leads to considering novel practical features that depend on the different types of slabs managed. Indeed, as the output of the former rolling process, slabs in these facilities are thinner (4-70mm vs 100-250mm) and of well-defined shapes and sizes. For this reason, they are also referred to as *plates* and are classified in *typologies*, which are groups of slabs sharing specific physical/chemical properties (e.g., length, thickness, weight, steel quality). This fact has manifold implications.

Firstly, *stacking constraints* may exist. Thinner (and lighter) slabs can be accidentally handled together with thicker (and heavier) ones if stacked consecutively, which may delay the retrieval process. Additionally, slabs with higher length or width may bend if stacked upon shorter ones and, hence, be no longer usable (Wang et al., 2008). Besides, since slabs must be cut, according to prefixed cutting schemes, to produce hulls' subassemblies or welded on hulls' blocks (Zhong et al., 2013), each item is satisfiable only by slabs of a given typology. Consequently, candidate slabs' sets correspond to slabs' typologies. As we will discuss in Section 3, this may affect how the order is released; in these contexts, it may be specialised by listing the number of slabs required for each typology. Such a circumstance renders the organisation of the yard, in terms of slabs' distribution across the stacks, a problem of strategic relevance in cutting/assembly centres. Indeed, while it is true that candidate slabs' sets are disclosed only when the order is released, we can undoubtedly assert their realisations fall within the finite set of slabs' typologies (known ex-ante). Hence, one may think, for instance, to distribute slabs according to the frequency the different typologies are required (e.g., obtained using historical data) to minimise the shuffling workload. This problem would be meaningless in rolling mills, since: (i) due to the coarser nature of the slabs, even those with different characteristics may be suitable for an item; (ii) candidate sets are not related to slabs' features, but solely depend on the order itself. Finally, in cutting/assembly centres, slabs are also characterised by *temporary quality certifications* issued by dedicated bodies, named *classification societies*, responsible for verifying and certifying that the vessel's construction process complies with the required standards (Nersesian and Mahmood, 2010). Accordingly, slabs are associated with a *deadline* imposing a maximum time limit within which they can be used. In practice, these certifications introduce priority aspects in the retrieval process, thus raising the decision-makers sensitivity to expired slabs minimisation objectives.

The discussed background highlights the potentialities and importance of the SSSP, which, however, is not reflected in rich literature and leaves room for further improvements and extensions. Moreover, we note that the few studies on the topic do not offer a clear picture of the investigated problem to the reader, thus resulting in a very disaggregated body of knowledge for interested scholars.

In such a context, the contributions we aim at giving to the literature are threefold: (i) we develop a unifying theoretical framework, that is, a taxonomy of all the aspects involved in the SSSP, also including those related to the cutting/assembly centres; (ii) we extend the SSSP by proposing a novel mathematical programming model accounting for original features of the problem; (iii) we show the applicability of the introduced model through a computational experience on randomly generated instances.

The remainder of the paper is organised as follows. In Section 2, we review the literature on the SSSP, while in Section 3, we introduce and describe the proposed theoretical framework. Section 4 presents the proposed model for the SSSP, whose results are discussed in Section 5. Section 6 finally closes the paper with some conclusions and future research directions.

2. Literature background on the SSSP

As we pointed out in the previous section, and according to Lehnfeld and Knust (2014), the SSSP represents a well-defined and unique category of optimisation problems in the context of storage unloading operations. Therefore, in order to provide the reader with a comprehensive but focused overview of the topic, this section solely reviews the existing contributions on the SSSP. Tang et al. (2001) introduced the problem considering the real case of a rolling mill at a steel plant in Shanghai. This work proposed a non-linear recursive formula to evaluate the number of shuffles based on the following assumptions: (i) slabs had to be retrieved from the yard, according to the sequence expressed in the order; (ii) shuffled slabs had to be repositioned in their original stacks; (iii) each slab could belong to various candidate sets, which implied that certain slabs were suitable to satisfy multiple order items. The goal was to minimise the total number of shuffles. To solve the problem, the authors developed a two-stage heuristic in which an initial solution was generated, assigning the suitable slab in the higher position to each request, and then improved through a local search procedure. Computational experiments showed that the algorithm significantly reduces the number of shuffles w.r.t. the algorithm adopted in the company. Tang et al. (2002) and Singh and Tiwari (2004) proposed two different solution procedures for the above problem, under the hypothesis that each slab belonged to only one candidate set. In particular, Tang et al. (2002) introduced a modified genetic algorithm with ad-hoc population and genetic operators, further enhanced with a local search scheme. The authors showed that the proposed approach outperformed the algorithm by Tang et al. (2001) but suffered from premature convergence. To overcome this issue, Singh and Tiwari (2004) developed an improved parallel genetic scheme. Experiments on randomly generated instances proved the capability of the proposed procedure to improve the quality of the solutions. Tang and Ren (2010) addressed a variant of the SSSP where shuffled slabs were not repositioned in their initial stack but permanently moved on top of close ones. In particular, assuming the availability of enough space in the yard, these moves were allowed only towards stacks hosting no target slabs, thus avoiding further shuffles to fulfil the order. As these conditions affected the rule to update slabs' positions, an ad-hoc formula to compute shuffles was introduced. Besides, the problem involved slab yards with multiple spans served by dedicated cranes. The authors considered a deadline, i.e., a maximum time to satisfy the order, which, given the limited capacity of the cranes, fostered workload balancing among them. The objective function minimised the total retrieval time, given by the sum of the time to shuffle, lift, and move target slabs to the delivery point of each span. The authors modelled the problem as a non-linear program and proposed a heuristic algorithm based on segmented dynamic programming to solve it, which proved efficient in reducing cranes' workload.

For the same problem, Ren and Tang (2010) proposed an integer linear programming-based algorithm, which iteratively solved two subproblems, one related to slabs' selection and the other to crane's scheduling, until a feasible solution for the overall problem was found. Cheng and Tang (2010) studied a variant of the SSSP, in which the sequence in the order was no longer mandatory. Hence, slabs satisfying non-consecutive items could be retrieved sequentially. As in Tang and Ren (2010), shuffled slabs were positioned on stacks that did not host target ones. The model optimised both cranes' workload balancing and the number of shuffles. The authors proposed a scatter search algorithm to solve the problem and tested it on real data instances from a steel enterprise, showing a significant decrease in the shuffling workload compared to the program used in the company. Fernandes et al. (2012) proposed an explicit integer linear program for the original version of the SSSP by Tang et al. (2001). The authors showed that the running time of the model was comparable to the greedy algorithm used in the steel plant studied in that reference for small-sized instances but outperformed it, in terms of solutions' quality, as the size of the instances increased.

Based on a tailored indexing method of the slabs, Shi and Liu (2021) introduced a simplified integer linear programming model, which approximately solved the SSSP introduced in Tang et al. (2001). The solution of the approximate model was then improved through a Very Large-Scale Neighbourhood search (VLSN) algorithm coupled with a Branch-and-Bound (B&B) procedure. Computational experiments on randomly generated instances demonstrated the best performance of the proposed approach compared to constructive and classic local search heuristics. Recently, Rajabi et al. (2022) proposed novel integer programming models accounting for two different repositioning policies, i.e., where (i) shuffled slabs are relocated in their original stack after each retrieval or (ii) only when all the retrievals from the same stack are completed. The authors developed ad-hoc algorithms to generate initial feasible solutions, which were used to warm start the solution process (performed using an off-the-shelf solver). Some theoretical analyses were conducted to reduce the size of the problem and accelerate the solution procedure. Through extensive experiments, the authors showed their models were capable to solve large-sized instances in reasonable computational times.

Unlike the above-discussed studies, Wang et al. (2008), Deng (2014), and Liyun et al. (2020) addressed the SSSP in the case of shipyards. In these three references, a bilevel combinatorial optimisation model was considered, minimising the total retrieval time (Wang et al., 2008; Deng, 2014) or the number of shuffles (Liyun et al., 2020). Moreover, they also assumed that shuffled slabs could be moved to stacks hosting other target slabs. In particular, Liyun et al. (2020) imposed specific constraints to forbid shuffled slabs from being repositioned in their original stacks. Interestingly, these studies considered constraints on the maximum stacks' height and compatibility conditions - in terms of lengths and widths - between slabs stacked together. As concerns the solution approaches, the works proposed three different genetic algorithms, which proved effective in solving large instances in reasonable computational times.

In conclusion, our review highlights the broad spectrum of hypotheses, decisions and models that scholars have proposed on the topic despite the generally shared problem setting. Therefore, in order to provide a more unified view of the SSSP, a theoretical framework to classify the aspects and the variant of the problem is discussed in the next section.

3. A taxonomy of the SSSP

As the literature shows, the SSSP may present different characteristics, according to the considered assumptions and the application contexts. This section attempts to provide a theoretical framework hinged on a comprehensive classification of the elements involved in the problem. Other than those already formalised in the reference literature, novel aspects are also introduced to shed light on new problem settings that have been neglected in the literature. This way, we aim to highlight the variants covered by extant contributions and existing gaps. Our framework is based on the following pillars: *Order*, *Candidate slabs' sets*, *Shuffling method*, *Layout*, *Deadline constraints*, and *Objective Function*. We discuss the latter in Sections 3.1 – 3.6. Then, the classification of the analysed papers follows in Section 3.7. Before moving on, some useful notation is introduced. We denote by J the set of slabs, by F the set of stacks, and by H the set of spans. Additionally, let I be the set of items, i.e., order requests for processing purposes. Accordingly, we denote by J_i the candidate slabs associated with item $i \in I$, i.e., the set of slabs $j \in J$ that are suitable to satisfy it.

3.1 Order

We can classify orders into: (i) *sorted or non-sorted*; (ii) *single-period or multi-period*.

(i) *Sorted* or *non-sorted*

Orders are *sorted* if they indicate the mandatory sequence according to which slabs must be retrieved. This circumstance could reflect constraints related to the schedule of subsequent processing stages. Instead, in the *non-sorted* case, the sequence is non-binding. For the sake of clarity, an illustrative example – corresponding to an order with three items ($I = \{\mathrm{I}, \mathrm{II}, \mathrm{III}\}$) – is reported in Fig. 1. Candidate slabs' sets for each item are also indicated. For instance, slabs 1 or 6 are suitable to satisfy item I ($J_{\mathrm{I}} = \{1,6\}$). In this specific case, the problem consists of retrieving three target slabs, each one to be identified from sets {1,6}, {2,3,5}, {4}, respectively. If the order is sorted, one must satisfy item I first, then item II, and, finally, item III.

Items ($i \in I$)	I	II	III
Candidate Slabs' Sets (J_i)	{1,6}	{2,3,5}	{4}

Fig. 1. An example of order: the general case

(ii) Single-period or multi-period

The SSSP can be cast within a *multi-period* setting, involving a discrete set of time periods T (e.g., days or weeks), namely the *planning horizon*, where each item is associated with a time period $t \in T$. Hence, I^t denotes the set of items $i \in I$ to be satisfied in period $t \in T$. Note that items in period t (I^t, with $t \geq 2$) can be satisfied only after those in period $t - 1$ (I^{t-1}). Such a circumstance may reflect the knowledge of a predefined schedule of further production stages (and the corresponding slabs' requirements) in the longer term. Fig. 2 extends the example in Fig. 1 to a multi-period case involving five items and two time periods ($T = \{1,2\}$). Consistently with our notation, we have: $I^1 = \{\mathrm{I}, \mathrm{II}, \mathrm{III}\}$, and $I^2 = \{\mathrm{IV}, \mathrm{V}\}$. If the order is non-sorted, the sequence is not binding within each time period. However, one must always satisfy items I^1 first, and then, items I^2.

Time periods ($t \in T$)		1		2	
Items ($i \in I$)	I	II	III	IV	V
Candidate Slabs' Sets (J_i)	{1,6}	{2,3,5}	{4}	{2,8}	{5,7,9}

Fig. 2. An example of multi-period order

3.2 Candidate slabs' sets

Candidate slabs' sets can be classified as: (i) *typology-based* or *order-based*; (ii) *disjoint or not-disjoint*.

(i) Typology-based or order-based

This aspect is strictly related to the application context of the problem. In cutting/assembly centres or shipyards, slabs can be classified in a set K of typologies, each involving slabs sharing the same physical/chemical properties. We recall that, due to the specific processing occurring at these facilities, each item can be satisfied only by slabs of a given typology. Hence, we say that candidate slabs' sets are *typology-based* sets since they correspond to slabs' typologies. In practice, denoted by J_k the set of slabs of typology $k \in K$, it follows that, for each candidate set $J_i, i \in I$, there always exists only one $k \in K$, such that $J_i = J_k$. Note also that slabs belong to only one typology ($J_k \cap J_{k'} = \emptyset, \forall\ (k, k') \in K, k \neq k'$). Instead, slabs are not classified in typologies in rolling mills, and the candidate sets are known when the order is released. Therefore, we say that candidate sets are *order-based*. The example in Fig. 1 applies to both cases. However, as it happens in practice in cutting/assembly centres or shipyards, the order specifies slabs' typology for each item. Supposing that three typologies of slabs are available, say A, B, and C ($K = \{A, B, C\}$), and that one slab of each typology is required, the order shows as in Fig. 3. Then, the problem consists of retrieving three target slabs, each one to be identified from sets J_A, J_B, and J_C, respectively.

Items ($i \in I$)	I	II	III
Slabs' Typologies ($k \in K$)	A	B	C

Fig. 3. An example of order: typology-based candidate sets case

(ii) Disjoint or not-disjoint

In the *disjoint* case, each slab is suitable for only one item: hence, there is a one-to-one correspondence between items and slabs and a null intersection between any pair of candidate slabs' sets ($J_i \cap J_{i'} = \emptyset, \forall\ (i, i') \in I, i \neq i'$). Instead, in the *not-disjoint* case, some slabs may be used to satisfy the request for multiple items ($\exists\ (i, i') \in I: J_i \cap J_{i'} \neq \emptyset, i \neq i'$). Accordingly, the example in Fig. 1 illustrates a disjoint case. Note that the example in Fig. 3 also falls within the disjoint case since slabs, as we underlined, can belong to only typology. Fig. 2 shows, instead, a not-disjoint case, since candidate sets of items I and IV share slab 2, while slab 5 can satisfy items II and V.

We also highlight that, when candidate sets are typology-based, multiple items may require slabs of the same typology. In such a situation, we are, by definition, in a (particular) not-disjoint case; however, there is at least a pair of items with equivalent candidate sets ($\exists\ (i, i') \in I: J_i = J_{i'}$). In other words, the same typology $k \in K$ occurs more than once in the order. With reference to Fig. 3, this case would arise if typology A occurs for both items I and III. Under these conditions, quantities q_k can be associated with slabs' typologies $k \in K$. In particular, if the order is also non-sorted, it can be expressed only in terms of quantities required for each typology. Following the above example, we can ultimately say that an order in cutting assembly/centres or shipyards may be released as in Fig. 4a. A more general multi-period case is displayed in Fig. 4b. We consider the latter setting for the model presented in Section 4.

Slabs' Typologies ($k \in K$)	A	B	C
Quantities (q_k)	2	1	0

(a) Single-period

Time periods ($t \in T$)		1		$\cdots$		t		$\cdots$		$\|T\|$	
Slabs' Typologies ($k \in K$)	A	B	C	$\cdots$	A	B	C	$\cdots$	A	B	C
Quantities (q_k^t)	2	1	0	$\cdots$	1	1	2	$\cdots$	0	2	2

(b) Multi-period

Fig. 4. Examples of (non-sorted) order in cutting assembly/centres

3.3 Shuffling method

We can distinguish two types of shuffles: (i) *with-repositioning* or (ii) *without-repositioning*.

(i) With-repositioning shuffles

Intuitively, shuffles are *with-repositioning* if shifted slabs are temporarily moved to different stacks and then repositioned in the original one. In this case, we highlight that shifted slabs can be repositioned after every single retrieval or when all the retrievals from the same stack are completed. Hence, we distinguish between *non-consecutive-repositioning* and *consecutive-repositioning*, respectively.

(ii) Without-repositioning shuffles

Alternatively, if the relocation is permanent, shuffles are *without-repositioning.* From the literature, two sub-cases emerge, based on the criteria used to select the destination stack. One option is to move slabs on top of stacks hosting no target slabs.

In this case, shifted slabs play the role of "*barrier*" only once during the retrieval process (*one-time barrier*). In practice, they will not cause any further shuffle. Otherwise, shifted slabs may be moved towards any stack and might be reshuffled (*multiple-time barrier*).

3.4 Layout

The characteristics of the layout considered in the SSSP may differ based on two main elements: (i) *spans* and (ii) *stacks*.

(i) Spans

In general, a slab yard may be divided into a set F of spans. Depending on their number, we distinguish between *single-span* or *multiple-spans* yards. Typically, each span is served by a dedicated crane. Although it does not affect the overall number of shuffles, the use of multiple cranes may reduce the total retrieval time and induce considerations on cranes' workload balancing.

(ii) Stacks

The layout can also be characterised by: *constrained* or *unconstrained stacks*; *typology-dedicated* or *random* stacks. In practice, we say that stacks are *constrained* if specific conditions rule the stacking process. These can be due to incompatibilities between slabs (e.g., different lengths or weights), or other physical requirements (e.g., maximum stack height). Of course, we have *unconstrained* stacks when no such conditions apply.

In cutting/assembly centres or shipyards, stacks can also be dedicated to hosting predefined slabs' typologies (*typology-dedicated* stacks). Such a choice may reflect a precise storage strategy, which can significantly affect the retrieval process, as we discuss in sections 4 and 5. Regardless of the application context, we say that stacks are *random* when no predefined schemes drive the stacking process.

3.5 Deadline constraints

The SSSP may be further enriched with *deadline constraints* referring to *orders* and *slabs*. In particular, the *order deadline* indicates a time limit, say d, within which the overall retrieval process has to be completed. Note that deadlines may also be associated with items ($d_i, i \in I$). Hence, in this case, the order deadline can be calculated as the maximum deadline across all the items ($d = \max_{i \in I}\{d_i\}$). *Slabs' deadlines* are typical of cutting/assembly centres or shipyards. In these contexts, slabs have a *temporary quality certification* testifying their compliance with the standards required for the hulls' production process. Thus, each slab has a deadline ($d_j, j \in J$), corresponding to the duration of the certification itself, beyond which the slab is no longer retrievable.

3.6 Objective Function

The SSSP mainly involve optimising efficiency measures associated with the retrieval process. As shuffles are very time-consuming operations, a typical objective function minimises the total *number of shuffles* or the *total retrieval time.* In yards with multiple spans, efficiency-oriented objectives can also seek to optimise *cranes' workload balancing* obtained by ensuring that each crane handles (approximately) the same number of slabs. Finally, when slabs' deadlines exist, the *number of expired slabs* also emerge as a relevant cost function to be minimised.

3.7 Papers' classification

Table 1 classifies the extant studies (i.e., those discussed in Section 2) according to the proposed theoretical framework for the SSSP. Briefly, we note that most of these works considered sorted orders and given their diffused application in rolling mills context, typology-based candidate sets and layout characterised by random stacks. Also, they mainly focused on shuffles' minimisation objectives. The above table highlights the relatively high number of variants proposed for the SSSP compared to the number of papers; in the authors' opinion, this fact further supports the need to classify them more systematically. Besides, the table sheds light on the (three) main novelties of the present work: (i) the multi-period setting given to the SSSP, due to the presence of multi-period orders; (ii) the consideration of slabs' deadlines and, hence, expired slabs minimisation objectives; (iii) the consideration of typology-related stacks. We will emphasise the latter aspect in our computational experiments to discuss the impact of the yard's organisation on the retrieval process (Section 5.3).

The proposed optimisation model is presented next.

Table 1
Papers' classification

				Context										
				Rolling Mills								*Shipyards*		*Cutting/assembly centres*
				Tang et al. (2001)	Tang et al. (2002), Singh and Tiwari (2004)	Ren and Tang (2010)	Tang and Ren (2010)	Cheng and Tang (2010)	Fernandes et al. (2012)	Shi and Liu (2021)	Rajabi et al. (2022)	Wang et al. (2008), Deng (2014)	Liyun et al. (2020)	This paper
Order	**Sequence**		*Sorted*	√	√	√	√		√	√	√			
			Non-sorted					√			√	√	√	√
	Planning horizon		*Single-period*	√	√	√	√	√	√	√	√	√	√	
			Multi-period											√
Candidate slabs' sets	**Relationship with typologies/order**		*Typology-based*									√	√	√
			Order-based	√	√	√	√	√	√	√	√			
	Relationship among candidate sets		*Disjoint*	√			√	√	√	√	√			
			Not-Disjoint		√	√					√	√	√	√
Shuffling method	**With-repositioning**		*Consecutive*								√			√
			Non-consecutive	√	√				√		√			
	Without-repositioning		*One-time barrier*			√	√	√						
			Multiple-time barrier									√	√	
Layout	**Spans**		*Single-span*	√	√				√	√	√	√	√	√
			Multiple-spans			√	√	√						
	Stacks	Slabs' typologies	*Typology-dedicated*											√
			Random	√	√	√	√	√	√	√	√	√	√	
		Stacking rules	*Constrained*									√	√	√
			Unconstrained	√	√	√	√	√	√	√	√			
Deadline constraints	Order deadline					√	√							
	Slabs' deadline													√
Objective function	Number of shuffles (to be minimized)			√	√			√	√	√	√		√	√
	Retrieval time (to be minimised)					√	√					√		
	Span workload (to be balanced)							√						
	Expired slabs (to be minimised)													√

4. Optimisation model

With reference to a generic a cutting/assembly centre, we consider the presence of a set J of slabs, belonging to a finite set K of typologies, distributed across a set F of stacks in a single span yard, served by a single crane. The order specifies the number of slabs required for each typology over a planning horizon T ($q_k^t, k \in J, t \in T$ – see Fig. 4b). Each slab $j \in J$ is associated with a deadline (d_j) indicating the maximum time period $t \in T$ within which it can be retrieved. In particular, a fictitious deadline ($d_j = |T| + 1$) is considered for slabs not expiring within the planning horizon. We further assume that shuffled slabs are repositioned in their stacks after all the retrievals from the same stack have been completed. The problem aims at identifying the target slabs satisfying the order seeking to minimise two objectives: the total number of shuffles and the number of expired slabs. To model the problem, the following additional notation is introduced:

Sets and parameters

$J' \subset J$	subset of slabs expiring during the planning horizon T ($J' = \{j \in J: d_j \leq \lvert T \rvert\}$);
f_j	stack hosting slab $j \in J$;
p_j^0	initial position of slab $j \in J$ in stack f_j (note that positions are indexed from the top to the bottom of the stack).

Decision variables

x_j^t	binary variables equal to 1 if slab $j \in J$ is retrieved at time period $t \in T$;
s_f^t	non-negative decision variables accounting for the number of shuffles occurring at stack $f \in F$ in time period $t \in T$.

The model, denoted by (M_1) is as follows:

$$\min \quad z = \sum_{t\in T}\sum_{f\in F} s_f^t + \lambda \sum_{j\in J'} \left(1 - \sum_{t=1}^{d_j} x_j^t\right) \tag{1}$$

subject to:

$$s_f^t \geq \; p_j^0 x_j^t - \sum_{l\in J:(p_l^0 \leq p_j^0 \cap f_j = f_l)} \sum_{m=1}^{t} x_l^m \qquad f \in F.\; t \in T. j \in J: f_j = f \tag{2}$$

$$\sum_{j\in J_k} x_j^t = q_k^t \qquad k \in K, t \in T \tag{3}$$

$$\sum_{t\in T} x_j^t \leq 1 \qquad j \in J \tag{4}$$

$$x_j^t = 0 \qquad j \in J', t \in \{\min\{d_j + 1; |T|\}, \dots, |T|\} \tag{5}$$

$$x_j^t \in \{0,1\} \qquad j \in J, t \in T \tag{6}$$

$$s_f^t \geq 0 \qquad f \in F, t \in T \tag{7}$$

Objective Function (1) minimises the sum of two terms: the total number of shuffles ($\sum_{t\in T}\sum_{f\in F} s_f^t$) and the number of expired slabs ($\sum_{j\in J'} (1 - \sum_{t=1}^{d_j} x_j^t$). A non-negative coefficient λ is employed weighting the relative importance of the latter objective. Note that setting $\lambda = 0$ [∞] leads to minimising the number of shuffles' [expired slabs] only. Otherwise, a trade-off solution is sought. We emphasise that λ represents the number of additional shuffles one is willing to perform to reduce the number of expired slabs by one unit. Constraints (2) count the number of shuffles s_f^t occurring at stack f in period t. In general, the shuffles needed to retrieve slab j in period t is given by the number of slabs stacked above it in that period. Clearly, this value equals the position of slab j in period t minus one; in turn, this can be calculated as the initial position of slab j (p_j^0) minus the number of slabs initially located above it (i.e., $l \in J: p_l^0 < p_j^0$) that have already been taken up to period t. Hence, should slab j be retrieved in period t ($x_j^t = 1$), the RHS of Constraints (2) expresses the resulting number of shuffles. The value of s_f^t is then obtained as the number of shuffles needed to retrieve the lowest-stacked slab, that is, the highest number of shuffles occurring across slabs hosted in stack f in period t. Note that, according to Constraints (2) and (7), we have: $s_f^t = \max\{0, \max_{j\in J: f_j = f}\{p_j^0 x_j^t - \sum_{l\in J:\left(p_l^0 \leq p_j^0 \cap f_j = f_l\right)} \sum_{m=1}^{t} x_l^m\}\}$. Constrains (3) ensure the satisfaction of the order since they guarantee that, in each time period $t \in T$, the number of slabs required for each typology $k \in K$ (q_k^t) is retrieved. Constraints (4) assure that each slab j can be retrieved at most once during the planning horizon. Constraints (5) forbid retrieving already expired slabs. Finally, Constraints (6) and (7) define the domain of the introduced decision variables.

We note that we can go even further in terms of our formulation. Indeed, relying upon the fact that slabs cannot be retrieved once expired (Constraints (5)) and when the typology k they belong to is not required in period t (i.e., $q_k^t = 0$, see Constraints (3)), we can write a more compact model considering only those decision variables that are strictly necessary. To this end, let us denote by k_j the typology of slab j. Accordingly, it is straightforward to derive that the x-variables need to be introduced, for each slab j, only for time periods $t \in \{1, .., \min\{d_j, |T|\}\}$ such that $q_{k_j}^t \neq 0$. Besides, we can also restrict the s-variables, for each stack f, to time periods $t \in T$ in which at least one of the slabs' typologies it hosts, say K_f, is required (i.e., $t \in T: \sum_{k\in K_f} q_k^t \neq 0$). Model (M_1) can be then modified into the following mathematical program (M_2):

$$min \quad z = \sum_{t\in T}\sum_{f\in F} s_f^t + \lambda \sum_{j\in J'} \left(1 - \sum_{t\in\{1,..,d_j\}:q_{k_j}^t\neq 0} x_j^t\right) \qquad (1')$$

subject to

$$s_f^t \geq p_j^0 x_j^t - \sum_{l\in J:(p_l^0\leq p_j^0 \cap f_j = f_l)} \sum_{m\in\{1,..,\min\{d_j,m\}\}:q_{k_l}^m\neq 0} x_l^m \qquad f\in F, t\in T, \; j\in J:(f_j = f \cap d_j \geq t) \qquad (2')$$

$$\sum_{j\in J_k:d_j\geq t} x_j^t = q_k^t \qquad k\in K, t\in T: q_k^t \neq 0 \qquad (3')$$

$$\sum_{t\in\{1,..,\min\{d_j,|T|\}\}:q_{k_j}^t} x_j^t \leq 1 \qquad j\in J \qquad (4')$$

$$x_j^t \in \{0,1\} \qquad j\in J, \; t\in\{1,..,\min\{d_j,|T|\}\}: q_{k_j}^t \neq 0 \qquad (6')$$

$$s_f^t \geq 0 \qquad f\in F, t\in T: \sum_{k\in K_f} q_k^t \neq 0 \qquad (7')$$

We implemented model (M_2) for the computational experiments reported in Section 5.

4.1 An optimal decomposition strategy

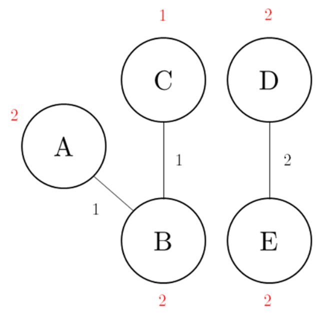
In general terms, we can describe the yard by the pair $(\alpha|\beta)$, where α indicates the number of slabs' typologies, and β the number of stacks. Hence, in our terminology, the above-introduced models apply to a generic $(|K|\,|\,|F|)$-*yard*. However, the knowledge of the yard's configuration, in terms of slabs' distribution (and, hence, their typologies) across stacks is particularly relevant. Indeed, it allows obtaining the optimal solution of the problem by aggregating the optimal solutions of a series of restricted subproblems, each focusing on a limited subset of stacks and typologies. In practice, an optimal decomposition strategy of the problem is applicable. To explain this fact, let us consider a simple example of a $(5|5)$-*yard*, i.e., with five typologies ($K = \{A,B,C,D,E\}$) and five stacks ($F = \{1,2,3,4,5\}$). Besides, let us consider a *typology/stack assignment matrix*, say A, whose generic element a_{kf} is a binary label equal to 1 if slabs of typology $k \in K$ are hosted in stack $f \in F$, 0 otherwise (Fig. 5a). Accordingly, we say that two typologies are *adjacent* if they are hosted together in at least one stack. Therefore, we can introduce an *adjacency matrix* E, whose generic element e_{kl} (with $k \neq l$) is equal to 1 if typologies k and l are adjacent, i.e., if $F_k \cap F_l \neq \emptyset$, being denoted by F_k and F_l the set of stacks they are hosted in. In our example, we can say that A is adjacent to B, B is adjacent to both A and C, while D is adjacent to E (Fig. 5b). At this point, it is possible to represent a generic yard as a graph $G = (K,E)$ with vertices representing the typologies K and E containing all the direct links between the typologies (a link exists for every pair of adjacent typologies). Each node has a weight (in red) indicating the number of stacks it is hosted in. Also, edges are associated with weights indicating the number of stacks shared by the typologies they connect. Fig. 5c depicts the underlying graph of the considered example. From this picture, it is possible to notice that the graph has two connected components, given by $G_1 = \{A,B,C\}$ and $G_2 = \{D,E\}$. In essence, we can recognise two separate (sub)-yards: a $(3|3)$-*yard*, involving typologies $\{A,B,C\}$ and the corresponding set of stacks $\{1,2,3\}$, and a $(2|2)$-*yard* involving the remaining sets of typologies ($\{D,E\}$) and stacks ($\{4,5\}$). In our problem, this fact implies that handling slabs of typologies D or E, does not influence retrieving slabs of typologies A, B and/or C.
Hence, we can conclude that solving model (M_2) for the considered $(5|5)$-*yard* corresponds to solving two separate subproblems, i.e., one for each connected component, say $M_2(G_1)$ and $M_2(G_2)$.

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline A & 1 & 1 & 0 & 0 & 0 \\ B & 0 & 1 & 1 & 0 & 0 \\ C & 0 & 0 & 1 & 0 & 0 \\ D & 0 & 0 & 0 & 1 & 1 \\ E & 0 & 0 & 0 & 1 & 1 \end{array}$$

(a) Slabs' typologies /stacks assignment matrix (*A*)

$$\begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & & 1 & 0 & 0 & 0 \\ B & & & 1 & 0 & 0 \\ C & & & & 0 & 0 \\ D & & & & & 1 \\ E & & & & & \end{array}$$

(b) Adjacency matrix (*E*)



(c) The underlying graph of the yard (*G*)

Fig. 5. Yard's representation

In general, the solution of a SSSP, formulated through model (M_2), defined over a yard with a set of K typologies and a F of stacks, involves three main steps:

Step 1. Compute the connected components of the underlying graph G of the yard. Denote their number by q. Note that this can be done in polynomial time using a breadth[depth]-first-search algorithm;

Step 2. Solve model $M_2(G_r)$ for $r \in \{1,..,q\}$, where $M_2(G_r)$ is defined over sets $K = G_r$ and $F = \cup_{k \in G_r} F_k$. Denote the corresponding vector of decision variables in the optimal solutions by $x_r^\star$ and $s_r^\star$, and their objective function by $z_r^\star$;

Step 3. Compute the vector of decision variables of the optimal solution to problem (M_2) as $x^\star = \cup_{r \in \{1,..,q\}} x_r^\star$, and $s^\star = \cup_{r \in \{1,..,q\}} s_r^\star$, and the corresponding objective function $z^\star = \sum_{r \in \{1,..,q\}} z_r^\star$.

5. Computational experiments

This section reports on the computational tests performed to test model (M_2). Our goal here is to evaluate the impact on the retrieval process determined by:

(i) The parameter λ, that is, the relative importance given to the minimisation of the number of expired slabs (Experiment 1);

(ii) The organisation of the yard, i.e., the consideration of various slabs' typologies distributions across stacks (Experiment 2).

To these ends, two different tests on randomly generated instances are realized: $(1|1)$-*yard* instances for Experiment 1; $(3|3)$-*yard* instances for Experiment 2. We first detail our instances' generation procedure in Section 5.1. Then, we present results for Experiments 1 and 2 in Sections 5.2 and 5.3, respectively.

5.1 Instances' generation procedure

Given the absence of benchmark instances for our problem, we developed a tailored generation procedure, that is described next. Each instance needs the following data: (i) the number of typologies ($|K|$), the number of stacks ($|F|$), the number of time periods ($|T|$); (ii) the distribution of slabs' typologies across stacks; (iii) the order, i.e., the number of slabs required for each typology $k \in K$ and for each time period $t \in T$ (q_k^t); (iv) the number of slabs per typology ($|J_k|$), and, hence, the total number of slabs ($|J| = \sum_{k \in K} |J_k|$); (v) the deadlines for each slab $j \in J$ (d_j); (vi) the stack hosting each slab $j \in J$ (f_j) and the corresponding position of slab j in stack f_j (p_j^0). We generated the data as follows:

(i) For Experiment 1, we considered: $|F| = 1$, $|K| = 1$, and $|T| = \{5, 10, 15\}$. For Experiment 2, instead, we set: $|F| = 3$, $|K| = 3$, and $|T| = 8$.

(ii) The distribution of slabs' typologies across stacks is obviously considered for Experiment 2 only. We considered eight different configurations, corresponding to various typologies/stacks allocations (see Fig. 5a). We refer the reader to Section 5.3 for further details. Note that this phase provides necessary information to be used in step (vi).

(iii) To create the order, based on the above information, we defined two additional parameters: $\bar{t}_k$, indicating the maximum number of periods slabs of typology k may be required; q_k^{max}, representing the maximum number of slabs requestable for each typology k in each time period. In practice, $\bar{t}_k$ and r_k account for the *frequency* and *intensity* of typologies in the order, respectively. For Experiment 1, we considered: $\bar{t}_k = \bar{t} = |T|$, and $q_k^{max} = q^{max} = 4$. For Experiment 2, we distinguish two cases, whether the above-introduced parameters are constant values (Experiment 2a) or vary by typology (Experiment 2b). In particular, for Experiment 2a, we considered: $\bar{t}_k = \{7, 7, 7\}$, and $q_k^{max} = \{5, 5, 5\}$; for Experiment 2b, instead, we set: $\bar{t}_k = \{8, 4, 2\}$, and $q_k^{max} = \{10, 5, 3\}$, to simulate a case where typologies have different turnover rates. The order was then generated in two steps: first, by randomly extracting for each typology, from a uniform discrete distribution in the interval $\{1, .., |T|\}$, the $\bar{t}_k$ periods in which it can be required; then, by extracting the number of slabs required for typology k in the $\bar{t}_k$ periods (q_k^t), from a uniform discrete distribution in the interval $\{0, .., q_k^{max}\}$. Note that, in this step, we also automatically have information of the typology of each slab j (k_j).

(iv) In order to ensure the availability of enough slabs in the yard to satisfy the order, we calculated the number of slabs per typology $k \in K$ ($|J_k|$) as: $|J_k| = \bar{t}_k q_k^{max}$.

(v) As regards the deadlines, we proceeded as follows. First, based on the already generated order, we needed to guarantee, in each time period t and for each typology k, the availability of at least q_k^t slabs expiring beyond t. Indeed, this condition is sufficient to ensure satisfying the order period by period. To this end, for each time period t and for each typology k, we associated q_k^t slabs with a deadline between t and $t + 2$ (randomly extracted). Then, for the reminding $|J| - \sum_{t \in T} \sum_{k \in K} q_k^t$ slabs, deadlines were assigned by considering a 70% probability of expiring beyond the planning horizon ($d_j = |T| + 1$), and a 30% probability of expiring within the time horizon. In the latter case, a value was extracted from a discrete uniform distribution in the interval $\{1, .., |T|\}$.

(vi) To determine slabs' positions and their stacks, we initially considered all the $|J|$ slabs as hosted in a single stack. This was obtained by assigning each slab with a random value (without repetition) in the interval $\{1, .., |J|\}$. Then, we swapped top-stacked slabs (i.e., those in the top half of the stack) with bottom ones (bottom half) until obtaining that at least 60% of the slabs with a deadline $d_j \leq \left\lceil\frac{|T|}{2}\right\rceil$ were in the bottom half of the stack. This was performed to obtain a more realistic situation in which "first-in" slabs, that reasonably have earlier deadlines, are mostly at the bottom of the stack. For Experiment 1, this procedure gives the initial positions (p_j^0) in the single stack we are considering. For Experiment 2, we randomly assigned each slab (beginning from the bottom one) to one of the stacks it can be hosted in (see step (ii)), and their positions were updated accordingly.

The described procedure was coded and implemented in MATLAB R2021A. The characteristics of the generated instances are summarised in Table 2. Note that, for Experiment 1, we generated 50 random instances for each pair ($|T|, |J|$). For Experiment 2, we generated 10 random instances for each of the eight different configurations of the yard we considered. Hence, our test bed comprises 310 instances: 150 for Experiment 1, and 160 for Experiment 2. These instances are available to anyone interested upon writing to the authors.

Table 2
Characteristics of the generated instances

		$\lvert T\rvert$	$\lvert K\rvert$	$\lvert F\rvert$	$\lvert J\rvert$	$\bar{t}_k$	q_i^{max}	*No. of yard's configurations*	*No. of instances*
Experiment 1		5	1	1	20	5	4	1	50
		10	1	1	40	10	4	1	50
		15	1	1	60	15	4	1	50
Experiment 2	*2a*	8	3	3	105	7,7,7	5,5,5	8	80
	2b	8	3	3	106	8,4,2	10,5,3	8	80

All the experiments were solved using IBM ILOG CPLEX Optimization Studio 12.9 software with default parameters settings on an Intel(R) Core(TM) i7-8550U CPU with a frequency of 1.80 GHz and 16 GB of RAM.

5.2 Experiment 1

In this experiment, we try to evaluate the effect of parameter λ, i.e., the relative importance of expired slabs minimisation objective, on the number of shuffles. First, we present two detailed solutions to get a deeper understanding of the solutions we are obtaining. Then, we discuss the overall results.

5.2.1 Detailed solutions

We focus on two instances with $|T| = 10$ and $|J| = 40$. We generated the Pareto front for these instances by applying the well-known "ϵ-constraint method" (Mavrotas, 2009). First, we obtained the upper and lower bounds on the number of expired shuffles, say UB_{ES} and LB_{ES}, by solving model (M_2) for $\lambda = 0$ and $\lambda = |T| \times |J|$, respectively. Note that the latter settings also provide the lower and upper bounds on the number of shuffles (LB_S and UB_S). Then, we iteratively solved the model, setting $\lambda = 0$, by adding the following Constraint:

$$\sum_{j\in J'}\left(1-\sum_{t\in\{1,..,d_j\}:q_{k_j}^t\neq 0} x_j^t\right)\leq ES_i \tag{8}$$

where ES_i is the maximum number of slabs that can expire in the generic i-th iteration. This value is calculated as $ES_i = ES_{i-1}^\star - 1$, with $ES_{i-1}^\star$ denoting the number of expired slabs in the optimal solution of the preceding iteration. Clearly, in the first iteration, we set $ES_1 = UB_{ES} - 1$, and proceeded until reaching the lower bound LB_{ES}. In Fig. 6, the Pareto front obtained for the two instances are displayed.

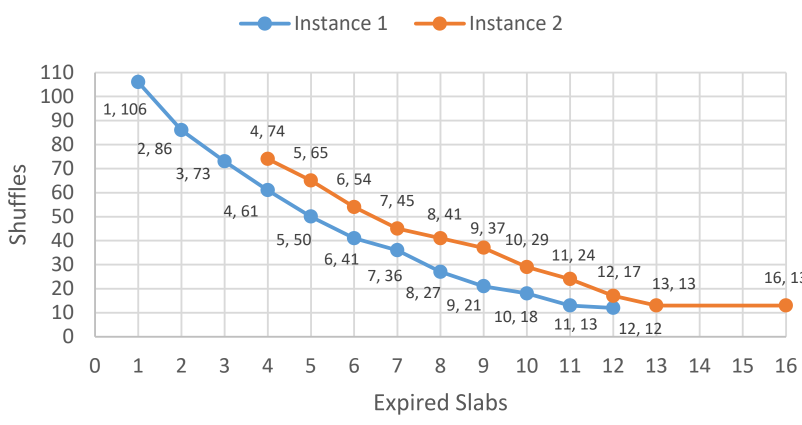


Fig. 6. Pareto front of the considered instances

The first aspect that emerges as relevant to our analysis is the variety of efficient solutions a decision-maker may consider (13 in Instance 1 and 11 in Instance 2). Hence, the relative importance of the two objectives plays a role in identifying the most suitable trade-off solution between the envisaged goals. Indeed, their conflicting nature is also evident, as reducing the number of expired slabs generally increases the number of shuffles. Interestingly, we note that, especially in the case of Instance 1, the marginal variation of the number of shuffles, that is, the number of additional shuffles needed to reduce the number of expired slabs by (at least) one unit, is higher for lower values of the number of expired slabs. We emphasise that this value corresponds to the slope of the Pareto front between two consecutive solutions. This fact implies that, as long as the number of expired slabs is relatively high, the produced solutions are almost indistinguishable for the decision-maker in terms of shuffles. On the contrary, a decision-maker interested in minimising expired slabs may evaluate solutions whose corresponding shuffling workload can differ significantly. Although less evident, this trend is confirmed when looking at Instance 2. This preliminary evidence highlights that the trade-off between the two objectives is relevant and tends to smooth as the number of expired slabs increases.

i. *Extensive results*

As noted in Section 5.1, we tested our model on 50 randomly generated instances for each considered pair ($|T|, |J|$). Moreover, we solved it by varying λ to partially reproduce the Pareto front for the tested instances. In particular, we considered $\lambda = 0$ and $\lambda = |T| \times |J|$ to obtain the extreme points of the Pareto front. Also, for each instance, we set $\lambda \in \{\Lambda, \frac{\Lambda}{2}, \frac{\Lambda}{4}\}$, to obtain three intermediate solutions. Consistently with the notation used in Section 5.2.1, we set $\Lambda = \left\lceil \frac{UB_S - LB_S}{UB_{ES} - LB_{ES}} \right\rceil$. In practice, Λ represents the slope of the line passing through the above-mentioned extreme points, having coordinates (LB_{ES}, UB_S) and (UB_{ES}, LB_S), and respectively associated with the solutions obtained for $\lambda = |T| \times |J|$ and $\lambda = 0$. Note that Λ varies by each tested instance. In total, we performed 750 experiments: three pairs ($|T|, |J|$), 50 random instances for each pair ($|T|, |J|$), and five values for λ. Tables 3 and 4 summarise the obtained results. In particular, Table 3 reports the minimum, maximum, and average objective function values (number of shuffles and expired slabs) calculated across the 50 tested instances for each pair ($|T|, |J|$) and each value of λ. Similarly, Table 4 indicates the corresponding computational times (in seconds) to solve our instances up to proven optimality.

Table 3
Results of Experiment 1: number of shuffles and expired slabs

			Shuffles					*Expired slabs*				
			λ					λ				
$\|T\|$	$\|J\|$		$\|T\| \times \|J\|$	Λ	$\Lambda/2$	$\Lambda/4$	0	$\|T\| \times \|J\|$	Λ	$\Lambda/2$	$\Lambda/4$	0
5	20	*Min*	10	2	0	0	0	0	0	2	2	3
		Max	55	38	15	15	14	7	7	10	10	10
		Avg	27.08	14.16	5.22	3.88	3.40	1.46	3.04	5.08	5.64	6.08
10	40	*Min*	51	17	5	0	0	0	3	8	8	10
		Max	199	102	46	38	36	9	13	16	17	18
		Avg	99.92	48.92	22.32	17.38	16.32	4.20	8.04	11.68	12.92	13.82
15	60	*Min*	89	36	12	7	7	1	8	14	17	19
		Max	311	143	77	73	69	15	22	26	27	27
		Avg	183.56	83.24	38.22	29.36	27.36	8.54	14.86	19.8	21.52	22.62

Table 4
Results of Experiment 1: CPU times (sec.)

			λ				
$\|T\|$	$\|J\|$		$\|T\| \times \|J\|$	Λ	$\Lambda/2$	$\Lambda/4$	0
5	20	*Min*	0.16	0.17	0.18	0.17	0.16
		Max	0.90	1.46	1.11	34.00	37.00
		Avg	0.40	0.42	0.35	1.00	2.23
10	40	*Min*	0.22	3.55	2.74	1.64	0.62
		Max	9.95	678.19	55.88	46.02	22.79
		Avg	2.66	41.66	12.69	8.43	4.48
15	60	*Min*	46.58	27.98	12.00	11.86	4.08
		Max	110.01	41250.87	6448.24	1077.70	362.09
		Avg	81.36	5088.26	372.05	177.14	39.72

From these Tables, we can observe that the above-discussed findings are confirmed. Indeed, for each pair $(|T|, |J|)$, the trade-off between the objectives keeps being evident, as they show two opposite trends by the values of λ. Moreover, we can again point out that the marginal variation in the number of shuffles is especially relevant for higher values of λ. For instance, if we focus on the pair $(15, 60)$, we observe that, on average, performing two additional shuffles allows reducing the number of expired slabs by one unit (namely, when λ increases from 0 to $\Lambda/4$). However, the (average) marginal shuffling workload increases significantly with λ: indeed, reducing the number of expired slabs by about five units (namely, when λ increases from $\Lambda/2$ to Λ), calls for 45 additional shuffles. In the extreme case, i.e., when λ increases from Λ to $|T| \times |J|$, 100 further shuffles are averagely needed to have six expired slabs less.

Besides, from Table 4, we also observe that the computational times increase with the size of the problem. However, most of the instances can be solved within acceptable CPU times. Indeed, only in five out of 250 cases, the $(10, 40)$-instances required more than one minute to be optimally solved. Instead, running times exceeded one hour in 16 cases for the $(15, 60)$-instances (15 for $\lambda = \Lambda$ and $\lambda = \Lambda/2$). Nevertheless, we underline that in 75% of the cases, these larger-sized instances are solved within less than five minutes. Remarkably, we notice the highest computational times for $\lambda = \Lambda$ and $\lambda = \Lambda/2$; possibly, such values accentuate the trade-off between the two objectives, thus making the tested instances "harder" to solve under these settings.

5.3 Experiment 2

This experiment aims at evaluating the impact of slabs' distributions on the retrieval process. To this end, we tested our model on instances under eight different cases, which we detail in Section 5.3.1. Afterwards, we present the obtained results in Section 5.3.2.

5.3.1 Slabs' distributions

For these tests, we recall, we considered $(3|3)$-*yard* instances and eight different distributions of the three slabs' typologies ($K = \{A, B, C\}$) across the three stacks ($F = \{1, 2, 3\}$). The considered distributions are detailed in Table 5, while the corresponding underlying graphs are displayed in Fig. 7. The last column of Table 5 also informs on the optimal problem decomposition induced by each distribution (as we explained in Section 4.1). For instance, when focusing on the first distribution (ID 1), we note that the problem can be decomposed into three subproblems, each focusing on a $(1|1)$-*yard*. Indeed, in this case, we have three dedicated stacks (one per typology). Accordingly, we report the notation $(1|1) \times 3$, to indicate that a $(1|1)$-*yard* problem has to be solved three times. On the contrary, distribution 8 does not allow for any decomposition since each stack hosts all the typologies. Note, in fact, that the underlying graph has no connected subcomponents. We denote this case by $(3|3) \times 1$. The other are intermediate cases. Distribution 4, for example, induces two subproblems: one for a $(1|2)$-*yard* (two stacks are dedicated to typology A); one for a $(2|1)$-*yard* (typology B e C share the same stack). Therefore, we report the notation $(1|2) \times 1 \cup (2|1) \times 1$.

Table 5
Considered slabs' distributions and deriving problem decomposition

Distribution ID	*Stack 1*	*Stack 2*	*Stack 3*	*Problem decomposition*
1	A	B	C	$(1\|1) \times 3$
2	AB	C	C	$(2\|1) \times 1 \cup (1\|2) \times 1$
3	AC	B	B	$(2\|1) \times 1 \cup (1\|2) \times 1$
4	BC	A	A	$(2\|1) \times 1 \cup (1\|2) \times 1$
5	AB	AB	C	$(2\|2) \times 1 \cup (1\|1) \times 1$
6	AC	AC	B	$(2\|2) \times 1 \cup (1\|1) \times 1$
7	BC	BC	A	$(2\|2) \times 1 \cup (1\|1) \times 1$
8	ABC	ABC	ABC	$(3\|3) \times 1$

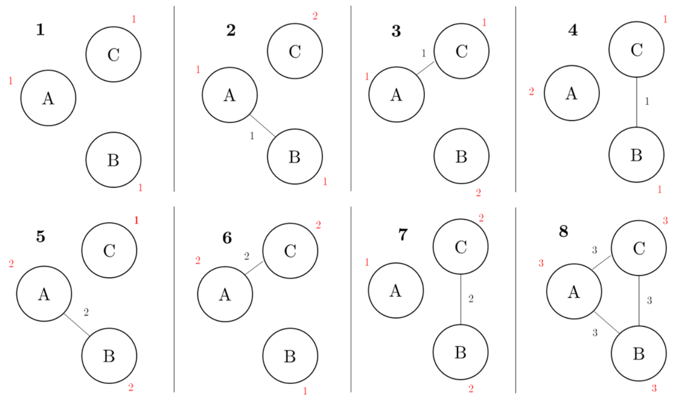
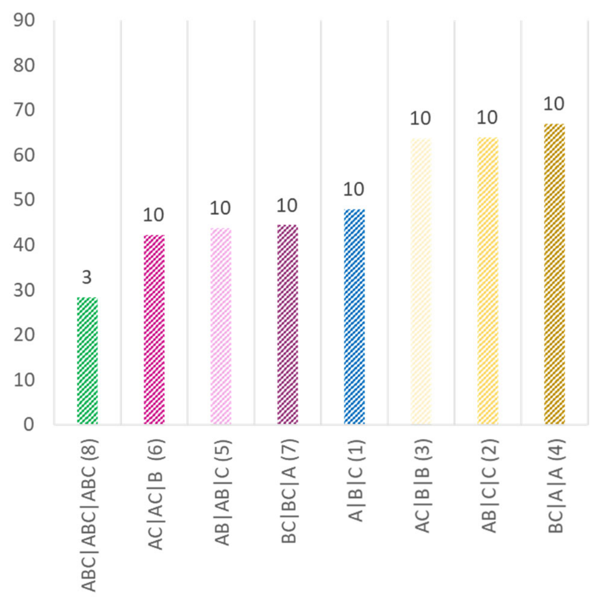


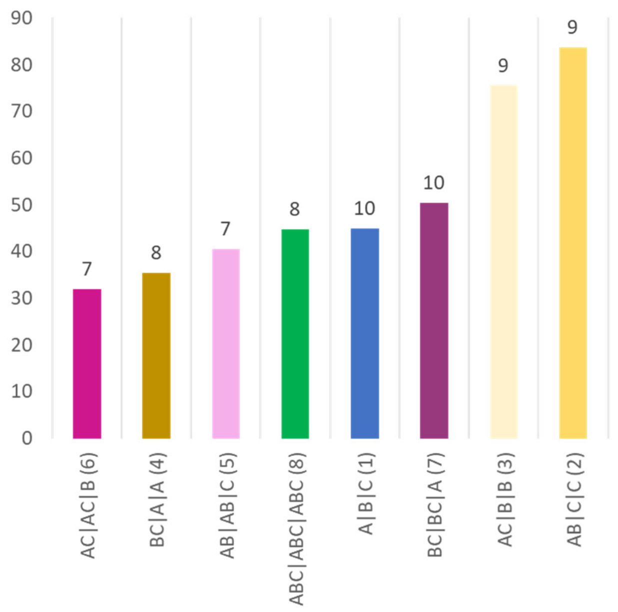
Fig. 7. The underlying graphs of the yard by slabs' distribution

5.3.2 Results

As described in Section 5.1 – step (iii), we distinguished two cases for our computational tests, depending on whether typologies have the same turnover rates (Experiment 2a) or not (Experiment 2b). In the latter case, we assumed turnover rates to reduce progressively from typology A to C. For both experiments, we generated 10 random instances for each yard configuration. Hence, we performed 160 tests in total. In all these tests, we set $\lambda = 0$, i.e., our focus is on shuffles' minimisation only. Fig. 8 reports on the average number of shuffles found, for each configuration, in Experiment 2a (Fig. 8a) and 2b (Fig. 8b). Table 6, instead, informs on the average computational times (in seconds). Specifically, these values were calculated by considering only those instances (out of ten) that were solved optimally. We indicate the number of solved instances on top of each bar in Fig. 8. Besides, in these pictures, we sorted slabs' distributions in non-decreasing order of the displayed indicator.



(a) Experiment 2a



(b) Experiment 2b

Fig. 8. Results of Experiment 2: average number of shuffles

Table 6
Results of Experiment 2: average computational times (sec.)

Distribution ID	*Stack 1*	*Stack 2*	*Stack 3*	*Experiment 2a*	*Experiment 2b*
1	A	B	C	4.41	71.11
2	AB	C	C	29.77	742.19
3	AC	B	B	27.40	117.55
4	BC	A	A	79.67	1435.98
5	AB	AB	C	161.26	916.72
6	AC	AC	B	202.50	322.00
7	BC	BC	A	815.91	71.01
8	ABC	ABC	ABC	36465.55	3064.61

From these tests, several conclusions can be drawn. If we focus on Experiment 2a, an interesting result is that adopting a "fully-dedicated" yard's configuration, i.e., with each stack hosting only one typology (distribution 1, $A|B|C$), is not the best strategy to pursue when turnover rates are similar. This is reasonably due to the effects of slabs' deadlines. Indeed, in principle, such configuration leads to the minimal shuffling workload, as one could handle the top-stacked slabs if deadlines are absent. However, this is not our case, and, as we underlined, "first-in" slabs (with earlier deadlines) are mainly at the bottom of their stacks. Consequently, satisfying the order meeting deadlines requirements determines a relatively high number of shuffles (equal to 47.90). Notably, this effect appears to mitigate under more "mixed strategies". Indeed, we notice that the number of shuffles tends to reduce when only one stack is dedicated (i.e., distributions 5, 6, 7), reaching its minimum (28.33) in correspondence of the most "randomised" case, namely distribution 8 ($ABC|ABC|ABC$). This evidence gets confirmed upon observing that more "dedicated" configurations (2, 3, and 4), where two stacks host only one typology, show the highest (average) number of shuffles (64.00, 63.80, 67.00, respectively). Reasonably, under these hypotheses, mixing typologies fosters the handling of more slabs from the same stack, thus allowing to (i) perform various retrievals consecutively and (ii) reduce the number of shuffles needed for order requests occurring in subsequent periods. Also, mixed-yard's configurations allow distributing slabs with close deadlines across various stacks, which may help reduce the shuffling workload, especially if obsolescence costs are not considered ($\lambda = 0$). However, we underline that mixed-yard's configurations increase the size of the problem and its computational burden: note, in fact, that we were unable to solve only seven instances for this experiment, all corresponding to configuration 8. This reflects in very high computational times, which are generally acceptable or even negligible in the other cases (see Table 6, column 5). Results are different when focusing on Experiment

2b. Since typology A has the highest turnover rate, it is not surprising that yards dedicating two stacks to lower required ones (namely, B and C) show the highest average number of shuffles (50.40, 75.56, and 83.78 for configurations 7, 3, and 2, respectively). Note that the latter is particularly high if no stack is solely dedicated to A. Again, we note that a "fully-dedicated" configuration (1) is not preferable and that mixing strategies emerge as viable to hedge against inefficiencies and meet slabs' deadlines. However, differently from Experiment 2a, a "random" configuration (8) is not the best choice: possibly, in this case, less required typologies play the role of barriers to retrieve frequently required ones more often, thus increasing the shuffling workload. Nevertheless, dedicating more stacks to the most frequently required typology (configurations 4 and 5) or, at least, mixing it with the least required ones (configuration 6) leads to the best results in terms of shuffles. Indeed, distributing typology A across various stacks makes it easier to access its slabs, thus reducing the shuffles needed to satisfy (large portions of) the order. Also, we highlight that instances appear harder to solve in this experiment, as we cannot obtain the optimal solutions for 12 out of 80 instances. Notably, these are not restricted to a single configuration (as in Experiment 2a), perhaps indicating that instances' "solvability" is affected more by numerical complexities than the size of the problem. Still, the latter leads to higher computational times (about 50 minutes for configuration 8), which are overall acceptable also in this experiment (see Table 6, columns 6).

6. Conclusion

The Slab Stack Shuffling Problem (SSSP) is a problem of practical relevance in the steel industry, consisting of retrieving slabs stored in stacks in a warehouse to satisfy a processing order. The problem has been mainly addressed in the rolling mills' context, and different variants have been proposed in the literature to tackle it. However, as we noted, the problem emerges also in cutting/assembly centres, especially within the shipbuilding supply chain, where large steel slabs - that have already undergone the rolling phase - must be handled and cut for the subsequent production stages. Addressing the SSSP in these facilities opens to considering novel features that motivated the research effort of the present paper. Specifically, the work had a twofold objective: (i) proposing a theoretical framework to systematise the main elements of the problem; (ii) developing a novel mathematical model for the SSSP. In particular, the latter extends previous formulations by accounting for original characteristics, namely slabs' deadlines and different yard's design strategies induced by stacking constraints and yard space limitations. Moreover, we cast the model as a bi-objective multi-period program seeking to minimise: (i) the number of shuffles, assumed as a proxy of the overall retrieval process cost; (ii) the number of expired slabs, considered as a proxy of obsolescence costs.

Our first set of computational experiments on randomly generated instances involving one slabs' typology and one stack demonstrated the relevance of the trade-off between the two objectives. Specifically, they showed that a decision-maker might evaluate various efficient solutions that can significantly differ in shuffles if relatively high importance is given to obsolescence costs.

The second set of experiments confirmed the relevance of deadlines' considerations. Indeed, our results suggest that a "fully-dedicated" yard's configuration, i.e., where each stack hosts only one slabs' typology, does not seem the best choice to achieve the least shuffling workload. If typologies have the same turnover rates, our experiments indicate that a decision-maker should prefer more mixed-strategies. The latter likely allow more consecutive retrievals from the same stacks, thus making the overall process more efficient. However, such a strategy fails if turnover rates vary significantly by typology. Under these circumstances, too much mixing may hinder handling the most frequently required slabs. One may opt for "hybrid" solutions in such cases, dedicating more stacks to fast-moving typologies and mixing the rest. Nevertheless, if space is an issue, our results confirm that mixing fastest and slowest-moving typologies leads to good performances. Of course, this paper is not free from limitations, which call for further research directions. Firstly, computational experiments should be extended to larger-sized and real-world instances to confirm the above insights. However, although often acceptable, the relatively high computational times reveal that our model might not accomplish this goal. Hence, the development of tailored solutions approaches - possibly exploiting the mathematical structure of the problem - would represent a natural extension of the work. As our literature review shows (Section 2), various algorithms have been devised to tackle the SSSP efficiently. Hence, one may argue why we do not resort to nor benchmark the proposed algorithm against such procedures. While we acknowledge this shortcoming, we feel the need to underline that our focus here is more on the modelling side than the algorithmic one and that the novel features we considered would likely make existing algorithms not very adequate for the investigated case. Also, a further consideration is due. Relying upon the optimal decomposition argument discussed in Section 4.1, one may expect computational times to increase "linearly" with the number of separable subproblems. Accordingly, computational times may still be relatively acceptable for larger instances, especially when handling decisions are planned in the longer term (and not made – almost – in real-time). Of course, we are aware that this observation may not hold due to numerical complexities. Hence, extending the study to cases involving a higher number of stacks and slabs' typologies and - above all - more complex yards' configurations is a necessary next step. Besides, since the yard's configuration significantly affects the retrieval process's efficiency, another line of research worth investigating may focus on developing and solving an integrated model for a *Combined Problem* involving slabs' storage decisions. The latter may assume a higher relevance under uncertainty on orders' requests, which also calls for stochastic extensions of the problem.

References

Avriel, M., & Penn, M. (1993). Exact and approximate solutions of the container ship stowage problem. *Computers & Industrial Engineering*, *25*(1-4), 271-274.

BRS GROUP, (2021). Shipping and Shipbuilding Markets, Annual Review. URL: https://www.brsbrokers.com [accessed on Oct 18, 2021].

Bortfeldt, A., & Forster, F. (2012). A tree search procedure for the container pre-marshalling problem. *European Journal of Operational Research*, *217*(3), 531-540.

Caserta, M., Schwarze, S., & Voß, S. (2012). A mathematical formulation and complexity considerations for the blocks relocation problem. *European Journal of Operational Research*, *219*(1), 96-104.

Caserta, M., Schwarze, S., & Voß, S. (2020). Container rehandling at maritime container terminals: A literature update. *Handbook of Terminal Planning*, 343-382.

Cheng, X., & Tang, L. (2010). A scatter search algorithm for the slab stack shuffling problem. In *International Conference in Swarm Intelligence* (pp. 382-389). Springer, Berlin, Heidelberg.

Dekker, R., Voogd, P., & Van Asperen, E. (2007). Advanced methods for container stacking. In *Container terminals and cargo systems* (pp. 131-154). Springer, Berlin, Heidelberg.

Deng, X. Y. (2014). A parallel optimisation algorithm for steel plate pick-up operation scheduling problem. *International Journal of Simulation Modelling*, *13*(3), 323-334.

Ding, D., & Chou, M. C. (2015). Stowage planning for container ships: A heuristic algorithm to reduce the number of shifts. *European Journal of Operational Research*, *246*(1), 242-249.

Fernandes, E. F. A., Freire, L., Passos, A. C., & Street, A. (2012). Solving the non-linear slab stack shuffling problem using linear binary integer programming. *EngOpt*.

Ferrari, C. (2012). Cantieristica navale: caratteristiche e tendenze di un mercato globale. *Impresa Progetto-Electronic Journal of Management*, *3*.

Forster, F., & Bortfeldt, A. (2012). A tree search procedure for the container relocation problem. *Computers & Operations Research*, *39*(2), 299-309.

Huang, S. H., & Lin, T. H. (2012). Heuristic algorithms for container pre-marshalling problems. *Computers & Industrial Engineering*, *62*(1), 13-20.

Jang, D. W., Kim, S. W., & Kim, K. H. (2013). The optimization of mixed block stacking requiring relocations. *International Journal of Production Economics*, *143*(2), 256-262.

Kim, K. H., Park, Y. M., & Ryu, K. R. (2000). Deriving decision rules to locate export containers in container yards. *European Journal of Operational Research*, *124*(1), 89-101.

Kim, Y., Kim, T., & Lee, H. (2016). Heuristic algorithm for retrieving containers. *Computers & Industrial Engineering*, *101*, 352-360.

Lehnfeld, J., & Knust, S. (2014). Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. *European Journal of Operational Research*, *239*(2), 297-312.

Liyun, X., Zhongyu, S., & Liansheng, Y. (2020). Steel plate scheduling optimisation in shipbuilding based on storage area partition. *Procedia CIRP*, 93, 1001-1006.

Mavrotas, G. (2009). Effective implementation of the ε-constraint method in multi-objective mathematical programming problems. *Applied Mathematics and Computation*, *213*(2), 455-465.

Nersesian, R., & Mahmood, S. (2010). International Association Of Classification Societies. In *Handbook of Transnational Economic Governance Regimes* (pp. 765-774). Brill Nijhoff.

Rajabi, P., Moslehi, G., & Reisi-Nafchi, M. (2022). New integer programming models for slab stack shuffling problems. *Applied Mathematical Modelling*, *109*, 775-796

Ren, H., & Tang, L. (2010). Modeling and an ILP-based algorithm framework for the slab stack shuffling problem considering crane scheduling. In *2010 International Conference on Computing, Control and Industrial Engineering* (Vol. 2, pp. 3-6).

Shi, Y., & Liu, S. (2021). Very Large-Scale Neighborhood Search for Steel Hot Rolling Scheduling Problem With Slab Stack Shuffling Considerations. *IEEE Access*, *9*, 47856-47863.

Singh, K. A., & Tiwari, M. K. (2004). Modelling the slab stack shuffling problem in developing steel rolling schedules and its solution using improved Parallel Genetic Algorithms. *International Journal of Production Economics*, *91*(2), 135-147.

Tang, L., Liu, J., Rong, A., & Yang, Z. (2001). An effective heuristic algorithm to minimise stack shuffles in selecting steel slabs from the slab yard for heating and rolling. *Journal of the Operational Research Society*, *52*(10), 1091-1097.

Tang, L., Liu, J., Rong, A., & Yang, Z. (2002). Modelling and a genetic algorithm solution for the slab stack shuffling problem when implementing steel rolling schedules. *International Journal of Production Research*, *40*(7), 1583-1595.

Tang, L., & Ren, H. (2010). Modelling and a segmented dynamic programming-based heuristic approach for the slab stack shuffling problem. *Computers & Operations Research*, *37*(2), 368-375.

Wang, G., Jin, C., & Deng, X. (2008, September). Modeling and optimisation on steel plate pick-up operation scheduling on stackyard of shipyard. In *2008 IEEE International Conference on Automation and Logistics* (pp. 548-553). IEEE.

Zhong, Y., Xue, K., & Shi, D. (2013). Assembly unit partitioning for hull structure in shipbuilding. *Computer-Aided Design*, *45*(12), 1630-1638.