Abstract

In this paper, we address a variant of a batch scheduling problem with identical parallel machines and non-anticipatory family setup times to minimize the total weighted completion time. We developed an ILS and a GRASP matheuristics to solve the problem using a constructive heuristic and two MIP-based neighborhood searches, considering two batch scheduling mathematical formulations. The problem derives from a ship scheduling problem related to offshore oil & gas logistics, the Pipe Laying Support Vessel Scheduling Problem (PLSVSP). The developed methods overcome the current solution approaches in the PLSVSP literature, according to experiments carried out on a benchmark of 72 instances, with different sizes and characteristics, in terms of computational time and solution quality. New best solutions are provided for all medium and large-sized instances, achieving a reduction of more than 10% in the objective function of the best case.

Keywords Parallel machine scheduling · Family scheduling · Batch scheduling · Matheuristic · Offshore industry logistics · Ship scheduling

1 Introduction

The discovery of Brazilian pre-salt fields in 2006, the largest one in recent years, duplicated Brazilian oil and gas reserves. These fields are located in ultra-deep waters, below the ocean salt deposits, exceeding 2,000 meters of water depth. Exploration and Production (E&P) are more challenging in this region in terms of technology and sustainability (Beltrao et al., 2009; Allahyarzadeh-Bidgoli et al., 2018; Haddad and Giuberti, 2010). Wells drilled in the Brazilian pre-salt basin are connected to surface platforms by flexible pipelines that better fits to the high water depths. Pipe Laying Support Vessels (PLSVs) are responsible for loading these pipelines at the port, transporting them to the wells site, laying them out in the ocean, and connecting them between the wells and platforms, allowing production to start (Speight, 2015; Clevelario et al., 2010). The PLSV Scheduling Problem (PLSVSP) consists of servicing a demand of sub-sea oil wells connections, finding the best schedule for a limited PLSV fleet, prioritizing the completion of wells with higher production levels.

The PLSVSP has been addressed previously in the literature. Queiroz and Mendes (2011) and Bremenkamp et al. (2016) simplified the problem by grouping pipeline connections to compose jobs, modeling it as a classic identical parallel machine scheduling problem. In the recent works of Cunha et al. (2017), Cunha et al. (2020), Abu-Marrul
et al. (2020), and Abu-Marrul et al. (2021), the problem is formulated as an identical parallel machine scheduling problem with batching. Cunha et al. (2017) and Cunha et al. (2020) applied heuristics to reschedule a small set of instances with similar characteristics to minimize the impacts caused by disruptions in a given schedule. Abu-Marrul et al. (2020) introduced three Mixed Integer Programming (MIP) formulations for the PLSVSP. They tested their approach on 72 instances generated from actual pre-salt data provided by a Brazilian company. A batch scheduling formulation using a dispatching rule to sequence operations within batches produced the best solutions, running for six hours. Abu-Marrul et al. (2021) developed several heuristics for building fast initial solutions for the PLSVSP. In our paper, we extend the works of Abu-Marrul et al. (2020) and Abu-Marrul et al. (2021), proposing a new variant on the batch scheduling formulation. We developed matheuristics using the constructive heuristic and the batch formulations to improve the solution quality and reduce computational time compared to previous works. An Iterated Local Search (ILS) and a Greedy Randomized Adaptive Search Procedure (GRASP) matheuristic for the PLSVSP are introduced, using two MIP-based neighborhood searches.

Matheuristics are hybrid approaches that combine concepts of metaheuristics and exact methods, being a growing field in the literature due to the improvement of computers and solvers (Thompson, 2018). Some researchers applied matheuristics to solve machine scheduling problems. Billaut et al. (2015) handled a single machine environment using a two-step approach, combining a beam search algorithm with a MIP-based neighborhood search. Regarding parallel machines environments, Ekici et al. (2019) applied a Tabu Search matheuristic, prohibiting some job-machine assignments during its execution. Ozer and Sarac (2019) developed a two-step approach combining a genetic algorithm with a MIP model. Woo and Kim (2018) proposed a two-step approach, grouping jobs in so-called buckets using metaheuristics and assigning them to machines by a mathematical model. Fanjul-Peyro et al. (2017) introduced matheuristics using constraint relaxation, limiting job-machine assignments, and using a MIP-based neighborhood search to optimize subsets of jobs. Other researchers applied matheuristics to solve flow-shop scheduling problems. Ta et al. (2018), Della Croce et al. (2014), and Della Croce et al. (2019) used MIP-based neighborhoods in positional scheduling formulations, solving sub-problems for a limited number of positions. Lin and Ying (2016) and Lin and Ying (2019) converted the flow-shop problem into a traveling salesman problem, building an initial heuristic solution and solving a mathematical model. To our knowledge, Mönch and Roob (2018) is the only work that applies matheuristics to a batch scheduling formulation. However, they do not apply MIP-based neighborhood search but use a two-step approach that combines a genetic algorithm to compose batches and a mathematical model to assign and sequence them on the machines. Matheuristics have also been successfully applied to many scheduling problems with realistic backgrounds. See, for instance the works of Martinelli et al. (2019), Kalinowski et al. (2020), and Grenouilleau et al. (2020), related to the scheduling of mining activities, rail network maintenance, and home health care services, respectively.

The PLSVSP can be modeled as a variant of an identical parallel machine scheduling problem with non-anticipatory family setup times and batching. In this analogy, vessels are machines, jobs represent wells, and pipeline connections are the operations. The non-anticipatory family setup times represent pipeline loading times, while PLSV voyages from the ports to the wells site are the batches. In this machine environment, a set of parallel machines is available to perform a given set of tasks. Machines are called identical when tasks processing times are fixed and, therefore, machine-independent (Pinedo, 2012). In most studies, tasks are called jobs, as each task requires only a single operation to complete. However, we represent tasks as operations, given that a set of operations must be finished to complete a job, in our problem. Besides, when family setup times are considered, tasks are grouped into families by similarity, and a setup time must be scheduled whenever a machine changes the execution of a task from one family to another. In this class of problems, the combination of one setup time and its subsequent tasks is called a batch. Also, when these setup times are non-anticipatory, the starting time of a batch cannot be anticipated, that is, it depends on the release of the tasks assigned to it. Parallel machine scheduling problems with family setup times are an important field in the scheduling literature. Shin and Leon (2004), Schaller (2014), Obeid et al. (2014), and Ciavotta et al. (2016) applied heuristics to solve parallel machine scheduling problems with realistic backgrounds but not dealing with non-anticipatory setup times. The reader can be referred to Allahverdi (2015) and Potts and Kovalyov (2000) for more details about scheduling problems with setup times considerations and batching.

The novelty of our research relies on the development of two new MIP-based neighborhood searches for batch scheduling formulations, testing its efficiency in an ILS, and a GRASP matheuristic procedures. We also provide an extension of a batch scheduling formulation for the PLSVSP, considering the sequence of operations inside batches as a model’s decision. Another contribution concerns the application to a problem with a realistic background, where the search for suitable solutions is crucial for sustainability in the company’s operation. The method developed in this work outperforms the current PLSVSP solution approaches in terms of solution quality and computational time when solving a benchmark of 72 PLSVSP instances developed from real-life data. Besides, dealing with a complex problem allows the use of the presented concepts on other batch scheduling problems by adjusting some formulation constraints. In ad-
dation, we contribute to the increase in the literature on matheuristics applied to parallel machine scheduling problems with batching.

The remainder of this paper is organized as follows. A description of the problem and its correspondence as an identical parallel machine scheduling problem is given in Section 2. Two mathematical formulations for the PLSVSP are provided in Section 3. In Section 4, a constructive heuristic for the PLSVSP is presented. The MIP-based neighborhood searches are introduced in Section 5 with the ILS and GRASP matheuristics described in Section 6. In Section 7, computational experiments are presented and discussed. Finally, the conclusion and some perspectives for future research are provided in Section 8.

2 Problem Description

The PLSVSP consists of scheduling a given PLSV fleet to meet a demand for pipeline connections in different subsea oil wells. The objective is to anticipate the completion of more productive wells (i.e., wells drilled in larger oil deposits). PLSVs are responsible for transporting the pipelines from the port to the wells site and connecting them between the wells and the production platforms. Each well includes a number of pipelines that must be connected to complete it, allowing its production to begin. Navigation times are disregarded in the problem due to the wells’ proximity in the Brazilian pre-salt basin, reducing the importance of vessel routing, thus defining a scheduling problem. The PLSV planners must define schedules of voyages for the vessels in which each voyage consists of the loading process of the pipelines at the port, followed by one or more pipeline connections to perform. Three main decisions are made when scheduling a PLSV fleet:

1. Definition of the voyages;
2. Assignment of the voyages to the vessels;
3. Sequencing of the voyages on each vessel.

In the first step, the planners must consider the family of each operation to compose the voyages. Each family represents a group of operations with a similar loading process at the port, and a voyage must only include operations of the same family. Moreover, the loading duration of each voyage is family-dependent. On the assignment step, the schedulers must check if a vessel is eligible and has enough space on the deck to perform all pipeline connections defined for a voyage since the fleet is heterogeneous. Finally, on the voyage sequencing step, the planners must consider the arrival of the pipelines at the port, which may occur on different days, and the day on which each vessel is available for starting their activities. The loading process can only begin when all pipelines defined for a voyage are available at the port. Thus, the defined sequence directly affects the vessels’ idleness and the delay in starting subsequent voyages. Due to the problem constraints and the concern with the solution’s quality, all activities from this decision process are carried out simultaneously.

The PLSVSP is described in Section 2.1 as an identical parallel machine scheduling problem with non-anticipatory family setup times and batching. We provide a mapping between the machine scheduling aspects and the PLSVSP context in Appendix A to help readers clarifying this relation.

2.1 Identical Parallel Machine Scheduling Approach

The notation and assumptions considered in the PLSVSP are described below, according to the machine scheduling theory.

1. There is a set $O$ of operations, where each operation $i$ has a processing time $p_i$, a release date $r_i$, a load occupancy $l_i$, and a family $f_i$.
2. There is a set $F$ of families where $s_f$ is the setup time for a family $f$.
3. All operations must be scheduled without preemption in a set $M$ of identical machines.
4. Each machine $k \in M$ is available to process operations from its release date $r_k$ and has a capacity $q_k$.
5. Machines are called identical due to the fixed processing times of the operations.
6. A subset $M_i$ defines the eligible machines for executing each operation $i$. Conversely, $O_k$ is a subset of operations $i \in O$ that a machine $k \in M$ is eligible to execute, i.e., $O_k = \{i \in O \mid k \in M_i\}$.
7. A family setup time is incurred on three occasions: while changing the execution of operations from different families, before the first operation on each machine, or when the machine’s capacity is reached.
8. We define Batch as a combination of one family setup time followed by a sequence with one or more operations from the same family.

9. The batching mode is Serial-Batching. Thus, the processing time of a batch is given by the sum of the operations’ processing times within the batch plus the setup time duration regarding the batch family.

10. The size of a batch, computed by the sum of the load occupancy of the operations within it, must respect the capacity of the machine assigned to execute it.

11. The setup times are non-anticipatory, i.e., a Batch can only start when all operations within it are released.

12. There is a set \( N \) of jobs where each job \( j \) is associated with a subset \( O_j \) of operations and has a weight \( w_j \) defining its priority.

13. We use \( N_i \) to identify a subset of jobs associated with operation \( i \), since in the PLSVSP, one operation might be related to several jobs simultaneously.

14. A job is completed when all of its associated operations are concluded. Thus, the completion time \( (C_j) \) of a job is the maximum completion time of the associated operations \( (C_j = \max_{i \in O_j} C_i) \). \( C_i \) is the completion time of operation \( i \).

15. The objective function is to minimize the total weighted completion time of all jobs, defined as \( \sum_{j \in N} w_j C_j \).

Figure 1: PLSV scheduling example with 15 operations, 5 Jobs and 4 machines. Adapted from [Abu-Marrul et al. (2020)].

In Figure 1, a PLSV schedule example is depicted, consisting of 15 operations, assigned, sequenced, and forming batches on four machines. The batches on the first machine are depicted on the top. Setup times are depicted with \( t \) inside the batches, reducing the solution space of the problem. To consider the entire solution space, we extend the Batch-WSPT formulation by adding inner batch sequencing variables. In this formulation, which we refer to as Batch-S, the sequence of operations inside batches is an optimization decision.

3 Mathematical Formulations

In this section, we present two MIP formulations for the PLSVSP. The first one is a batch formulation with a Weighted Short Processing Time (WSPT) dispatching rule [Pinedo (2012)], introduced by [Abu-Marrul et al. (2020)]. This formulation, which we refer to as Batch-WSPT, uses the WSPT dispatching rule to sequence operations heuristically inside the batches, reducing the solution space of the problem. To consider the entire solution space, we extend the Batch-WSPT formulation by adding inner batch sequencing variables. In this formulation, which we refer to as Batch-S, the sequence of operations inside batches is an optimization decision.
3.1 Batch Formulation with WSPT Dispatching Rule

To use the WSPT dispatching rule within the batch formulation, Abu-Marrul et al. (2020) created the inner-batch precedence subsets $O_i$ (Equation 1). Each subset $O_i$ is composed by the operations $i \in O$ that will precede operation $i$ if both are scheduled in the same batch. These subsets consider an estimated weight $w_i$ for the operations to compute the WSPT rule ($w_i/p_i$) and other breaking tie rules to achieve a complete ordering between operations. The complete ordering ensures that for any subset of operations assigned to the same batch, the model will be able to identify the sequence between them. To estimate the weight $w_i$ for each operation $i \in O$, we consider a proportional rule by splitting the weight of each job into equal parts for each of its associated operations, computed as $w_i = \sum_{j \in N_i} w_j / |O_j|.$

\[
O_i = \left\{ i \in O \left| \left( \frac{w_i}{p_i} > \frac{w_i}{p_j} \right) \lor \left( \frac{w_i}{p_i} = \frac{w_i}{p_j} \land w_i > w_j \right) \lor \left( \frac{w_i}{p_i} = \frac{w_i}{p_j} = w_i \land i < j \right) \right. \right\} \forall i \in O \tag{1}
\]

A set $B$ of batches is used in the formulation, and the subsets $B_k \subseteq B$ are generated to limit the number of available batches on each machine $k$. Taking into account the eligibility constraints, we limit $|B_k| = |O_k|$. Each element $b \in B_k$ defines the position of a batch in the schedule of machine $k$. When solving the PLSVSP, the model decides which batches to use, the family of these batches, and which operations to assign to it.

The following binary variables are considered in the formulation:

\[
X_{ik}^b = \begin{cases} 
1 & \text{if operation } i \text{ is scheduled in the } b\text{-th batch of machine } k; \\
0 & \text{otherwise.}
\end{cases}
\]

\[
Y_{fk}^b = \begin{cases} 
1 & \text{if the } b\text{-th batch of machine } k \text{ is of family } f; \\
0 & \text{otherwise.}
\end{cases}
\]

Other continuous variables are considered in the formulation:

- $S_k^b$ – Starting time of the $b$-th batch on machine $k$.
- $P_k^b$ – Processing time of the $b$-th batch on machine $k$.
- $C_i$ – Completion time of operation $i$.
- $C_j$ – Completion time of job $j$.

The Batch-WSPT formulation is as follows:

\[
\min \sum_{j \in N} w_j C_j \tag{2}
\]

subject to

\[
\sum_{k \in M_i} \sum_{b \in B_k} X_{ik}^b = 1 \quad \forall i \in O \tag{3}
\]

\[
\sum_{f \in F} Y_{fk}^b \leq 1 \quad \forall k \in M, b \in B_k \tag{4}
\]

\[
X_{ik}^b \leq Y_{fk}^b \quad \forall i \in O, k \in M_i, b \in B_k \tag{5}
\]

\[
\sum_{i \in O} l_i X_{ik}^b \leq q_k \quad \forall k \in M, b \in B_k \tag{6}
\]

\[
P_k^b \geq \sum_{i \in O} p_i X_{ik}^b + \sum_{f \in F} s_f Y_{fk}^b \quad \forall k \in M, b \in B_k \tag{7}
\]

\[
S_k^b \geq r_k \quad \forall k \in M, b \in B_k \tag{8}
\]

\[
S_k^{b+1} \geq S_k^b + P_k^b \quad \forall k \in M, b \in B_k \tag{9}
\]

\[
S_k^b \geq r_i X_{ik}^b \quad \forall i \in O, k \in M_i, b \in B_k \tag{10}
\]
The following binary variable $Z_{ik}$ is added to sequence operations inside batches:

$$Z_{ii} = \begin{cases} 
1 & \text{if operation } i \text{ and } i' \text{ are scheduled in the same batch and } i \text{ precedes } i'; \\
0 & \text{otherwise.}
\end{cases}$$

The Batch-S formulation is as follows:

$$\min \sum_{i \in \mathcal{O}} \sum_{k \in \mathcal{B}_i} \sum_{b \in \mathcal{B}_k} C_{ij}$$

subject to

$$Z_{ii} + Z_{ii'} \geq X_{i}^{b} + X_{i'}^{b} - 1 \quad \forall i, i' \in \mathcal{O}, k \in \mathcal{M}_i, b \in \mathcal{B}_k \quad (17)$$

$$Z_{ii} + Z_{ii'} \leq 1 \quad \forall i, i' \in \mathcal{O} \mid \mu_{ii'} \quad (18)$$

$$Z_{ii} + Z_{ii'} + Z_{i'i} \leq 2 \quad \forall i, i', i'' \in \mathcal{O} \mid \mu_{ii''} \quad (19)$$

$$C_i \geq S_k^b + p_i + s_{f_i} + \sum_{i \in \mathcal{O}} p_i Z_{ii} - \left(1 - X_{i}^{b}\right) M \quad \forall i \in \mathcal{O}, k \in \mathcal{M}_i, b \in \mathcal{B}_k \quad (20)$$

$$Z_{ii} \in \{0, 1\} \quad \forall i \in \mathcal{O} \mid \mu_{ii} \quad (21)$$

Constraints (17) identify when operations $i$ and $i'$ are scheduled in the same batch. Constraints (18) ensure that only one of the variables that define the precedence between operations $i$ and $i'$ inside a batch will be considered.
guarantee a complete ordering between operations inside a batch. Constraints (20) replace Constraints (11). The completion time of an operation is now computed with the sequencing variable $Z_{i \hat{i}}$. Finally, Constraints (21) present variable $Z_{i \hat{i}}$ domains.

4 Constructive Heuristic

We use a constructive heuristic, introduced by Abu-Marrul et al. (2021), to build the initial solutions for the PLSVSP. The authors tested 19 heuristics, combining several dispatching rules and ways of estimating weights for the operations, showing that one of them outperformed the others. The heuristic named $\text{WMCT-WAVGA}$, is an algorithm that, at each iteration, chooses an operation according to a Weighted Minimum Completion Time (WMCT) dispatching rule, scheduling it in the machine that minimizes the total weighted completion time in the partial schedule. Let $C_k$ be the completion time of each machine $k$, and $T_i$ be the minimum completion time among the set $\mathcal{M}$, of eligible machines for each operation $i$, computed as $T_i = \min_{k \in \mathcal{M}} C_k$. Then, at each iteration, the operation with the highest priority value $\pi_i$ is selected, where $\pi_i$ is computed according to Equation (22). The weight $w_i$ of operation $i$ is estimated as $w_i = \sum_{j \in \mathcal{N}_i} w_j / |\mathcal{U}_j|$, where $\mathcal{U}$ is the set of unscheduled operations, and $\mathcal{U}_j \subseteq \mathcal{U}$ is a subsets of unscheduled operations associated with job $j$. In the first iteration, $\mathcal{U}$ is equal to the set $\mathcal{O}$ of operations. During the heuristic execution, the weights of unscheduled operations are adjusted at each iteration.

$$\pi_i = \frac{w_i}{\max(T_i, r_i) + p_i + s_i} \forall i \in \mathcal{U}$$

The method creates batches by assigning operations sequentially to the machines. Therefore, at each iteration, the algorithm selects the next operation to schedule and assigns a machine to the operation. The method then decides whether to insert the selected operation in the last batch (called current batch) or to create a new batch to insert it in the assigned machine. If a new batch is created, the selected operation is sequenced as the first one inside the new batch, i.e., after a new family setup time is also inserted in the machine schedule. Otherwise, the operation is scheduled as the last one in the current batch on the assigned machine. There are three situations that force the creation of a new batch on the selected machine: (1) when the machine is empty; (2) when the current batch on the machine is of a different family from the selected operation; (3) when the insertion of the operation in the current batch exceeds the machine capacity. The $\text{WMCT-WAVGA}$ heuristic, presented in Algorithm 1, defines a list of schedules $\sigma = (\sigma_1, \ldots, \sigma_k)$ containing operations and families for each machine $k$. The families represent the setup times, indicating the beginning of a new batch.

New variables and sets are used to store information regarding the solution during the heuristic procedure. We use $S_k, L_k, F_k,$ and $A_k$ to save the starting time, cumulative load, family, and set of assigned operations regarding the current batch on machine $k$. $\Delta_{ik}$ computes the delay of the starting time for the current batch on machine $k$ due to the insertion of operation $i$ (*caused by the operation’s release date and the non-anticipatory setup time consideration*). $C_i^{CB}$ and $C_i^{NB}$ represent the completion time of operation $i$, if inserted in the current batch or in a new batch on machine $k$, respectively. Finally, we consider two sets to store the feasible assignments and their costs: (1) $\mathcal{CB}$ stores the set of feasible assignments $c_{b_{ik}}$ of operation $i$ into the current batch on machine $k$ (to be feasible, the selected operation must respect the eligibility, family, and capacity constraints); (2) $\mathcal{NB}$ stores the set of feasible assignments $n_{b_{ik}}$ of operation $i$ into a new batch on machine $k$ (to be feasible, the selected operation must respect the eligibility constraint).

The algorithm starts by initializing the variables and sets (Lines 1-3). The method runs until all operations are scheduled (Line 4). The main loop (Lines 4-23) starts by computing the priority value ($\pi_i$) for all unscheduled operations (Line 5). In Line 6, the next operation $i^*$ to schedule is selected according to the value of $\pi_i$. The delay at the starting time of the current batch on machine $k$ is computed in Line 7. In Lines 8 and 9, the selected operation’s completion times are computed, considering its insertion in the current batch or in a new batch on the assigned machine, respectively. Next, the method creates sets of feasible assignments (Lines 10 and 11). In Line 12, the assignment with minimum cost is selected, identifying the best machine $k^*$ in Line 13. If the selected element belongs to the set $\mathcal{CB}$, operation $i^*$ is inserted in the current batch on machine $k^*$ (variables and sets are updated in Lines 15, and 16). Otherwise, a new batch on machine $k$ is created for operation $i$ (variables and sets are updated in Lines 18-20). The remaining variables and sets of the algorithm are updated in Line 22. Finally, the procedure returns the final solution $\sigma$ in Line 25.

Note that there are no rules to control the sequence of operations inside the batches during the $\text{WMCT-WAVGA}$ heuristic execution. Since the $\text{Batch-WSPT}$ formulation (Section 3.1) generates these sequences heuristically, we need to update the subsets $O_i$ to use the constructed solutions in this formulation. Thus, we introduce a variable $\vartheta_{i \hat{i}} \in \{0, 1\}$ which
Algorithm 1: WMCT-WAVGA

1: \( C_k \leftarrow r_k, S_k \leftarrow r_k, L_k \leftarrow 0, F_k \leftarrow 0, A_k \leftarrow \emptyset, \sigma_k \leftarrow \emptyset \quad \forall k \in \mathcal{M}; \)
2: \( C_t \leftarrow \infty \quad \forall t \in \mathcal{O}; \)
3: \( U \leftarrow \mathcal{O}; \)
4: \textbf{while} \( U \neq \emptyset \) \textbf{do}
5: \[ w_i \leftarrow \sum_{j \in N_i} \frac{w_j}{|B_j|}, T_i \leftarrow \min_{k \in \mathcal{M}_i} C_k, \pi_i = \max(T_i, r_i) + p_i + s_{f_i} \quad \forall i \in U; \]
6: \textbf{Select operation} \( i^* \in U \) \textbf{that maximizes} \( \pi_i; \)
7: \( \Delta_{i^*k} \leftarrow \max(0, r_{i^*} - S_k) \quad \forall k \in \mathcal{M}_{i^*}; \)
8: \( C_{i^*k}^{CB} \leftarrow C_k + \Delta_{i^*k} + p_i \quad \forall k \in \mathcal{M}_{i^*}; \)
9: \( C_{i^*k}^{NB} \leftarrow \max(r_{i^*}, C_k) + s_{f_{i^*}} + p_i \quad \forall k \in \mathcal{M}_{i^*}; \)
10: \( CB \leftarrow \left\{ c_{b_{i^*k}} = w_i C_{i^*k}^{CB} + \sum_{k \in \mathcal{B}_k} w_i \Delta_{i^*k} \mid k \in \mathcal{M}_{i^*}, F_k = f_{i^*}, L_k + l_{i^*} \leq q_k \right\}; \)
11: \( NB \leftarrow \left\{ n_{b_{i^*k}} = w_i C_{i^*k}^{NB} \mid k \in \mathcal{M}_{i^*} \right\}; \)
12: \( b_{\text{min}} \leftarrow \min\{b : b \in (CB \cup NB)\}; \)
13: \textbf{Select} \( k^* \) \textbf{corresponding to} \( b_{\text{min}}; \)
14: \textbf{if} \( b_{\text{min}} \in CB \) \textbf{then}
15: \( S_{k^*} \leftarrow \max(r_{i^*}, S_k); C_{k^*} \leftarrow C_{i^*k}^{CB}; \)
16: \( L_{k^*} \leftarrow L_{k^*} + l_{i^*}; A_{k^*} \leftarrow A_{k^*} \cup \{i^*\}; \)
17: \textbf{else}
18: \( S_{k^*} \leftarrow \max(r_{i^*}, C_k); C_{k^*} \leftarrow C_{i^*k}^{NB}; \)
19: \( L_{k^*} \leftarrow l_{i^*}; A_{k^*} \leftarrow \{i^*\}; \)
20: \( \sigma_{k^*} \leftarrow \sigma_{k^*} \cup \{i^*\}; \)
21: \textbf{end}
22: \( \sigma_k \leftarrow \sigma_k \cup \{i^*\}; F_{i^*} \leftarrow f_{i^*}; C_{i^*} \leftarrow C_{i^*}; U \leftarrow U \setminus \{i^*\}; \)
23: \textbf{return} \( \sigma; \)

is equals 1 if a pair of operations \( (i, i^* \in \mathcal{O}) \) is scheduled in the same batch and \( i \) precedes \( i^* \) in the constructed solution, and zero otherwise. Thus, the new definition of the subsets \( \mathcal{O}_i \) is shown in Equation (23). Suppose that the solution depicted in Figure [1] was generated by the WMCT-WAVGA heuristic. Then, the schedule \( \sigma_k \) for the fourth machine \( (k = 4) \) would be \( \sigma_4 = \{f_{14}, f_{15}, f_6, f_{15}, 15, 7\} \), where \( f_{14}, f_6, \) and \( f_{15} \) indicates families 1, 2, and 3, respectively. In that case, \( \delta_{ii} = 1 \) only for \( i = 15 \) and \( i = 7 \), since this pair of operations are scheduled together in the last batch on machine 4.

\[
\mathcal{O}_i = \left\{ i \in \mathcal{O} \left| \delta_{ii} \lor \left( \delta_{ii} = \delta_{ii} \land \frac{w_i}{p_i} > \frac{w_i}{p_i} \right) \lor \left( \delta_{ii} = \delta_{ii} \land \frac{w_i}{p_i} = \frac{w_i}{p_i} \land w_i > w_i \right) \lor \left( \delta_{ii} = \delta_{ii} \land \frac{w_i}{p_i} = \frac{w_i}{p_i} \land w_i = w_i \land i < i \right) \right\} \quad \forall i \in \mathcal{O}
\]

(23)

In the next section, we present the MIP-based neighborhood searches to consider in the local search step of our methods.

5 MIP-based Neighborhood Searches

We consider two MIP-based neighborhood searches, named Batch Windows and Multi-Batches Relocate, making use of the batch formulations presented in Section [3] to decompose the PLSVSP into smaller problems that can be optimized more quickly than the complete problem. The idea is to limit the number of integer variables to optimize at each iteration, fixing the remaining ones from a feasible solution. As mentioned before, we consider the initial feasible solution, the one provided by the WMCT-WAVGA heuristic (Algorithm [1]). The approaches are detailed in the next sections.

In both methods, we use variables \( MB_k \in \mathbb{Z}_+ \) (Machine Batches) to limit the subset of batches to be considered on each machine \( k \in \mathcal{M} \) during the search. These variables are bounded by the size of the subsets \( B_k \) on each machine \( k \), thus \( 0 \leq MB_k \leq |B_k| \). Throughout the search, only batches with position \( b \leq MB_k \) are available to optimize. The
remaining batches are fixed without any operations inside them. To help to clarify the idea, we show in Figure 2 a graph representation with batches of the PLSV schedule illustrated in Figure 1 (Section 2). Each node represents a batch $b$ on a machine $k$, with the respective operations assigned to it described inside. Below each node, we show the starting time ($S_b^k$) and the completion time ($C_b^k$) of the respective batch. Note that the maximum number of batches on each machine $k$ is given by the total number of eligible operations. For instance, in machine 1, we have $|B_k| = 10$ (batches 1 to 10), following the subsets $O_k$ defined in Table 8 (Appendix B), with $O_1 = \{1, 2, 4, 5, 7, 9, 10, 11, 12, 14\}$. Nodes are labeled as: (1) Available batches (batches that can be used by the search procedures); (2) Unavailable batches (batches generated by the formulation but not available for the search procedure in a given iteration, according to variables $MB_k$). The variables $MB_k$ are updated during the procedures, allowing an unavailable batch to become available at some point in the search.

![Graph representation with batches of the PLSV scheduling example.](image)

Figure 2: Graph representation with batches of the PLSV scheduling example.

The $MB_k$ variables are initialized according to the $WMCT-WAVGA$ heuristic (Algorithm 1), considering the number of used batches plus one extra batch on each machine $k$. Let $\eta_b^k \in \{0, 1\}$ be a variable equal 1 if a batch $b$ on a machine $k$ is used in a given solution (i.e., contains at least one operation scheduled in it), and zero otherwise. Then, $MB_k$ are initialized according to Equation (24). Note that, on machine 1, we have four available batches (1 to 4), with $MB_1 = 4$, although the solution, depicted in Figures 1 and 2, only uses three batches on this machine. At each iteration of the MIP-based neighborhood searches, we check whether all available batches are being used on each machine $k$, updating $MB_k$ with Equation (24), if $\sum_{b \in B_k} \eta_b^k = MB_k$.

$$MB_k = 1 + \sum_{b \in B_k} \eta_b^k \quad \forall k \in \mathcal{M}$$

(24)

### 5.1 Batch Windows

In this approach, we limit the subset of batches to optimize based on a defined time range interval. At each iteration, we only optimize batches that are scheduled inside the range. The Range Size ($RS$) is defined as a fraction of the makespan ($C_{max} = \max_{i \in \mathcal{O}} C_i$) based on a given PLSV solution, computed as $RS = \lfloor \rho \times C_{max} \rfloor$, where $\rho \in [0, 1]$ is a parameter that defines the proportion of the makespan to consider. The complete search moves the optimization range from the end of the schedule to its beginning, with a step half the size of $RS$, ensuring overlap between iterations and that all batches are optimized at least once, totaling in $\lfloor C_{max}/(RS/2) \rfloor - 1$ iterations. We chose to move the search from the end of the schedule to its beginning based on preliminary experiments that showed advantages in this approach. We use $R_{begin}$ and $R_{end}$ to identify the beginning and the end of the optimization range to consider at each iteration.

Given the PLSV schedule example shown in Figure 1 (Section 2), we depict in Figure 3 how the optimization range defines the batch windows to optimize on each machine. We consider an optimization range of size 30 ($RS = 30$), which would result in a total of five iterations, with the following ranges ($R_{begin}, R_{end}$): Iteration 1 (60, 90); Iteration 2 (45, 75); Iteration 3 (30, 60); Iteration 4 (15, 45); Iteration 5 (0, 30). To save space, we only show iterations 1, 3, and 5, depicted in Figures 3(a), 3(b), and 3(c), respectively. In this example, we suppose the solution does not change during the search. On the right side of the figures, we show the solution graph representation with the batches...
on each machine, highlighting the ones to optimize at each iteration. Nodes are labeled as: (1) Fixed batches (batches not selected in the given iteration); (2) Batches to optimize (batches selected to optimize in the given iteration); (3) Unavailable batches (batches generated by the formulation but not available for the search procedure in the given iteration, according to variables $MB_k$).

![Diagram showing iterations and batch optimization](image-url)

(a) Iteration 1: $R_{begin} = 60$ and $R_{end} = 90$.

(b) Iteration 3: $R_{begin} = 30$ and $R_{end} = 60$.

(c) Iteration 5: $R_{begin} = 0$ and $R_{end} = 30$.

Figure 3: Example of three iterations in the Batch Windows neighborhood search, showing a PLSV schedule and the optimization range on the left side, and the graph representation with the batches to optimize highlighted on the right side.

Note that a given optimization range determines different sizes of batch windows to optimize, on each machine, due to the continuous variables that compute the starting time ($S^k_t$) and the completion time ($C^k_t$) of each available batch. For instance, in Iteration 3, depicted in Figure 3(b), on machines 1 and 2, there are three batches (1 to 3) to optimize, while on machines 3 and 4, there are two batches (2 and 3) to optimize. Let $O' \subseteq O$ be the subset of operations assigned to the selected batches to optimize in a given iteration. Then, $O' = \{1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 13, 15\}$ in iteration 5. The pseudo-code of the Batch Windows is shown in Algorithm 2.
Algorithm 2: Batch Windows (s, ρ, MBk)

1. $C_{max} \leftarrow \max_{i \in \mathcal{O}} C_i$, where $C_i$ is given by the PLSVSP solution $s$;
2. $RS \leftarrow \lfloor \rho \times C_{max} \rfloor$;
3. $R_{begin} \leftarrow \infty$;
4. $R_{end} \leftarrow C_{max}$;
5. while $R_{begin} > 0$ do
6. \hspace{1em} $C^b_k \leftarrow S^b_k + P^b_k$, $\forall k \in \mathcal{M}, b \in B_k$;
7. \hspace{1em} $R_{begin} \leftarrow \max(0, R_{end} - RS)$;
8. \hspace{1em} Create the subset $O'$ of operations assigned to batch $b$ on machine $k$ in which $S^b_k \leq R_{end}$ and $C^b_k \geq R_{begin}$ and $b \leq MB_k$ in solution $s$;
9. \hspace{1em} Solve Batch Formulation, starting from solution $s$, for a subset of variables $X^b_k$ in which $S^b_k \leq R_{end}$, $C^b_k \geq R_{begin}$, $b \leq MB_k$ and $i \in O'$;
10. \hspace{1em} $R_{end} \leftarrow R_{begin} + RS/2$;
11. \hspace{1em} Update variables $MB_k$ according to Equation 24 if $\sum_{b \in B_k} n^b_k = MB_k$;
12. end

Algorithm 2 starts by computing values for $C_{max}$, $RS$, $R_{begin}$, and $R_{end}$ (Lines 1-4), according to a given solution $s$ and the parameter $\rho$. The main loop of the algorithm (Lines 5-12) is repeated until $R_{begin}$ reaches zero. The completion time ($C^b_k$) of each batch $b$ on each machine $k$ is computed in Line 6. The algorithm updates $C^b_k$ and $R_{begin}$ (Lines 6 and 7), defining the subset $O'$ in Line 8. The Batch Formulation is solved for all variables $X^b_k$ of batches scheduled within the optimization range (Line 9). After solving the model, the value of $R_{end}$ is updated for the next iteration in Line 10. Finally, the procedure to update the number of available batches on each machine is executed in Line 11.

5.2 Multi-Batches Relocate

In this approach, we randomly select the batches to optimize at each iteration, not allowing the selection of batches already optimized in previous iterations. The complete search ends when each batch is optimized exactly once in one of the iterations. We compute the Number of Batches ($NB$) to optimize at each iteration based on a given parameter $\varphi \in [0, 1]$, as $NB = \lfloor \varphi \times \sum_{k \in \mathcal{M}} MB_k \rfloor$. The method runs with a total of $\lfloor \sum_{k \in \mathcal{M}} MB_k / NB \rfloor$ iterations.

An example of the Multi-Batches Relocate is depicted in Figure 4 using the graph representation shown in Figure 2. We consider $NB = 6$, generating a total of three iterations. At each iteration, we highlight the subset of batches to optimize. Nodes are labeled as: (1) Optimized fixed batches (batches already optimized in previous iterations); (2) Non-optimized fixed batches (batches not yet optimized but not selected in the given iteration); (3) Batches to optimize (randomly selected batches to optimize in the given iteration); (4) Unavailable batches (batches generated by the formulation but not available for the search procedure in the given iteration, according to variables $MB_k$).

![Figure 4: Example with three iterations of batches selection in the Multi-Batches Relocate.](image)

Note that at the end of the search, each batch is optimized precisely once in one of the iterations. A higher diversification can be seen in this neighborhood, as it allows the selection of non-sequential batches. One can also note that in the last iteration (Iteration 3), only five batches are selected, although we defined $NB = 6$. This is because these five
batches are the only ones not optimized at this point in the search. The pseudo-code of the Multi-Batches Relocate is shown in Algorithm 3. To describe the procedure, we use \( P \) to define the set of pairs machine/batch, where each element \((k, b)\) represents a specific batch \( b \) on a machine \( k \), thus \( P = \{(k, b) \mid k \in \mathcal{M}, b \in B_k\} \).

**Algorithm 3:** Multi-Batches Relocate \((s, \varphi, MB_k)\)

1. \( NB \leftarrow [\varphi \times \sum_{k \in \mathcal{M}} MB_k] \);
2. Initialize set \( P \) of pairs \((k, b)\) considering all machines \( k \in \mathcal{M} \) and their respective batches \( b \in B_k \);
3. while \( P \neq \emptyset \) do
   4. Select \( NB \) pairs machine/batch \((k, b)\) randomly from \( P \) to compose subset \( P' \);
   5. Create the subset \( O' \) of operations scheduled in the subset \( P' \) of pairs machine/batch \((k, b)\) on solution \( s \);
   6. Solve Batch Formulation, for the given solution \( s \), for a subset of variables \( X_{ik}^b \) in which \((k, b) \in P' \) and \( i \in O' \);
   7. Update variables \( MB_k \) according to Equation 24 if \( \sum_{b \in B_k} X_{ik}^b = MB_k \);
4. end

Let \( P' \subseteq P \) be the subset of selected pairs batch/machine on a given iteration. Algorithm 3 starts by computing the value of \( NB \) (Line 1) and initializing the set \( P \) (Line 2). The main loop of the algorithm (Lines 3-9) is repeated until there exist pairs batch/machine not optimized. Each iteration starts by randomly selecting \( NB \) pairs batch/machine from the set \( P \) to compose the subset \( P' \) (Line 4). The subset \( O' \) of operations assigned to any of the selected pairs batch/machine is created in Line 5. The Batch Formulation is solved, for a limited time, for all variables \( X_{ik}^b \), where \((k, b) \in P' \) and \( i \in O' \) (Line 6). Then, the set \( P \) and variables \( MB_k \) are updated in Lines 7 and 8, respectively.

## 6 Matheuristics

In this section, we present the matheuristics, which combines the MIP-based Neighborhood Searches (Section 5) and the GRASP heuristic (Algorithm 1) aiming to improve solutions continuously. Two well-known algorithm frameworks from the matheuristics literature are considered, the Iterated Local Search (ILS) and the Greedy Randomized Adaptive Search Procedure (GRASP). The following sections explain each method that we refer to as ILS-Math and GRASP-Math, respectively.

For the local search, we consider a Variable Neighborhood Descent (VND) algorithm (Hansen and Mladenovic [2003]), using the two MIP-based Neighborhood Searches described in Section 5.2, changing for the Batch Windows (Section 5.1) if no improvement is found. Every time an improved solution is found, we restart the local search, returning to the Multi-Batches Relocate. The local search stops when no improvement is found after running both MIP-based Neighborhood Searches completely. The sequence between the neighborhoods was defined based on preliminary experiments that showed a faster solution improvement using the Multi-Batches Relocate.

### 6.1 Iterated Local Search

ILS is a powerful tool for combinatorial optimization problems with a simple structure and very useful for practical experiments. The method starts by building an initial solution and improving it using a local search. Then, the main loop consists of perturbing the current solution with simple modifications and running the local search until a stopping criterion is reached (Lourenço et al. [2003]).

The ILS matheuristic (ILS-Math) is described in Algorithm 4. A parameter \( \Omega^{\max} \in \mathbb{Z}_+ \) defines the maximum number of iterations without improvement to execute, stopping the procedure whenever the value of the counter \( \Omega \) reaches \( \Omega^{\max} \). During the ILS execution, worst solutions may be accepted, according to an acceptance parameter \( \delta \in [0, 1] \). We use \( s^* \) to keep the best solution found among all iterations.

The perturbation phase of the ILS-Math consists of randomly swapping batches in a given PLSVSP solution. In this approach, named RandomBatchSwap, we compute the number of swaps \( (NS) \) to be performed based on a given parameter \( \omega \in [0, 1] \), where \( NS = [\omega \times \sum_{k \in \mathcal{M}} MB_k] \). Note that the number of swaps to be performed is a fraction of the total number of available batches, considering all machines. At each swap movement, two batches are selected, and their operations are exchanged. If an empty batch is selected, the movement consists of removing the operations of the batch with operations inside and inserting them in the empty one. We forbid the selection of two empty batches. The movement is executed by updating the value of the corresponding assignment variables \( X_{ik}^b \) of the selected batches.
Algorithm 4: ILS-Math ($\rho, \varphi, \omega, \delta, \Omega^{max}$)

1. Build an initial PLSVSP solution $s_0$ using the WMCT-WAVGA heuristic (Algorithm 1);
2. Initialize variables $MB_k$ based on $s_0$ according to Equation (24);
3. $s \leftarrow VND(s_0, \rho, \varphi, MB_k);
4. s* \leftarrow s;
5. \Omega \leftarrow 1;
6. while $\Omega \leq \Omega^{max}$ do
7. \quad $\Omega \leftarrow \Omega + 1;
8. s' \leftarrow RandomBatchSwap(s, \omega, MB_k);
9. s*' \leftarrow VND(s', \rho, \varphi, MB_k);
10. if $f(s*) < f(s*) \times (1 + \delta)$ then
11. \quad s \leftarrow s*;
12. \quad if $f(s) < f(s*)$ then
13. \quad \quad s* \leftarrow s*;
14. \quad \quad \Omega \leftarrow 1;
15. \quad end
16. else
17. \quad s \leftarrow s*;
18. end
19. end
20. Restart variables $MB_k$ based on the best solution $s^*$ according to Equation (24);
21. $s^* \leftarrow VND(s^*, \rho, \varphi, MB_k);
22. return $s^*$;

The ILS-Math starts by building an initial solution $s_0$ (Line 1), using the WMCT-WAVGA heuristic (Algorithm 1). In Line 2, the variable $MB_k$ is initialized (See Section 6). The VND is then performed on $s_0$ (Line 3), generating the current solution $s$, which is copied to $s^*$ (Line 4). The iteration counter $\Omega$ is initialized in Line 5. The main loop (Lines 6-19) is executed until $\Omega \leq \Omega^{max}$ and consists of repeatedly executing the Random Batch Swap (Line 8), followed by the VND (Line 9). In Line 10, the algorithm checks whether the objective value of the new solution $s^*$, given by $f(s^*)$, passes the acceptance criteria. If true, the current solution $s$ is updated to $s^*$ (Line 11), and the algorithm checks whether this solution is also better then $s^*$ (Line 12). If true, $s^*$ is updated (Line 13), and $\Omega$ is reinitialized (Line 14). When the new solution is not accepted, the best solution $s^*$ replaces $s$ (Line 17). Finally, an intensification step is executed by running the VND on the best solution $s^*$ (Lines 20-21).

6.2 Greedy Randomized Adaptive Search Procedure

GRASP is a multi-start method that combines a randomized constructive procedure followed by a local search, being successfully applied to many scheduling problems in the literature. For instance, we refer the reader to the papers of Bassi et al. (2012), Rodriguez et al. (2012), and Heath et al. (2013). In the constructive procedure, a Restricted Candidate List (RCL) is employed, and each element is randomly selected at each step (Resende and Ribeiro, 2019). To randomize the constructive procedure for the PLSVSP, we replace the operation selection step of the WMCT-WAVGA heuristic (Line 6 of Algorithm 1) by a Randomized Operation Selection, using a parameter $\alpha \in [0, 1]$ to define the greediness of the method. When $\alpha = 0$, the method builds the same solution of the WMCT-WAVGA heuristic, and when $\alpha = 1$, a completely randomized solution is generated. After computing the priority value $\pi_i$ for the set $U$ of unscheduled operations (Line 5 of Algorithm 1), we identify the minimum and maximum priority values, defined as $\pi_{min}$ and $\pi_{max}$, respectively. The RCL is created as $\text{RCL} = \{i \in U \mid \pi_i \geq \pi_{max} - \alpha(\pi_{max} - \pi_{min})\}$, and one operation $i^*$ is randomly selected from the RCL.

The pseudo-code of the GRASP matheuristic (GRASP-Math) is shown in Algorithm 5. We call Randomized Constructive Procedure, the WMCT-WAVGA heuristic (Algorithm 1) with the Randomized Operation Selection. A parameter $\Omega^{max} \in \mathbb{Z}_+$ defines the maximum number of iterations without improvement, while $s^*$ saves the best solution found among all iterations.

The GRASP-Math starts by initializing the $\Omega$ counter and $f(s^*)$ (Lines 1 and 2). The main loop is executed until $\Omega \leq \Omega^{max}$ (Lines 3-12). At each iteration, a new randomized solution $s$ is built (Line 5), with its $MB_k$ variables initialized (Line 6), and the VND is applied to this solution (Line 7), generating a new solution $s^*$. The algorithm checks whether the objective value of the new solution, given by $f(s^*)$, is better than $f(s^*)$ (Line 8). If true, $s^*$ is replaced by $s^*$, and the counter $\Omega$ is reinitialized (Lines 9-10). Finally, an intensification step is executed by running the VND on the best solution $s^*$ (Lines 13-14).
Algorithm 5: GRASP-Math (\(\rho, \varphi, \alpha, \Omega^{\max}\))

\begin{algorithmic}[1]
\State \(\Omega \leftarrow 1;\)
\State \(s^* \leftarrow \emptyset; f(s^*) \leftarrow \infty;\)
\While {\(\Omega \leq \Omega^{\max}\)}
\State \(\Omega \leftarrow \Omega + 1;\)
\State Build a PLSVSP solution \(s\) using the Randomized Constructive Procedure;
\State Initialize variables \(MB_k\) based on \(s\) according to Equation (24);
\State \(s'^* \leftarrow VND(s, \rho, \varphi, MB_k);\)
\If {\(f(s'^*) < f(s^*)\)}
\State \(s^* \leftarrow s'^*;\)
\State \(\Omega \leftarrow 1;\)
\EndIf
\EndWhile
\State Restart variables \(MB_k\) based on the best solution \(s^*\) according to Equation (24);
\State \(s^* \leftarrow VND(s^*, \rho, \varphi, MB_k);\)
\State return \(s^*;\)
\end{algorithmic}

6.3 Overview of the Methodology

The general pseudocode of the proposed methodology is described in Algorithm 6. The algorithm allows selecting different mathematical formulations to be used in the matheuristics’ main loop and the intensification step.

Algorithm 6: General Methodology

\begin{algorithmic}[1]
\State Choose a matheuristic framework between GRASP-Math and ILS-Math.
\State Select a mathematical formulation between Batch-WSPT and Batch-S.
\State Run the chosen matheuristic’s main loop using the selected mathematical formulation.
\State Choose a mathematical formulation between Batch-WSPT and Batch-S.
\State Initialize the variables of the chosen mathematical formulation according to the best solution found so far.
\State Run the VND algorithm using the chosen mathematical formulation.
\State Return the best-found solution.
\end{algorithmic}

The combination of the developed matheuristics’ frameworks (ILS-Math and GRASP-Math) and the batch scheduling formulations (Batch-WSPT and Batch-S), following the defined methodology, generates three variants of each matheuristic, which we refer to as ILS-Math\(_1\), ILS-Math\(_2\), ILS-Math\(_3\), GRASP-Math\(_1\), GRASP-Math\(_2\), and GRASP-Math\(_3\), described below:

- ILS-Math\(_1\): ILS-Math with Batch-WSPT
- ILS-Math\(_2\): ILS-Math with Batch-S
- ILS-Math\(_3\): ILS-Math with Batch-WSPT and Batch-S
- GRASP-Math\(_1\): GRASP-Math with Batch-WSPT
- GRASP-Math\(_2\): GRASP-Math with Batch-S
- GRASP-Math\(_3\): GRASP-Math with Batch-WSPT and Batch-S

The ILS-Math\(_3\) and the GRASP-Math\(_3\) consider the Batch-WSPT formulation for their main loop, changing for the Batch-S formulation at the intensification step.

7 Computational Experiments

In this section, we present the computational experiments conducted to assess the performance of the proposed matheuristics. We compare them with the Mathematical Formulations (Section 3) running independently. The computational experiments were performed on a machine with an Intel i7-8700K CPU of 3.70GHz and 64 GB of RAM running Linux. All methods were coded using C++ language solved by CPLEX 12.8 solver running in a single thread with MIP emphasis set to finding hidden feasible solutions. We limit the execution time to one second per CPLEX call in the matheuristics. Thus, sub-problem optimizations are interrupted within the defined time-limit even if the optimal solutions have not been reached.
To evaluate the solution’s quality, we compare the solutions provided by each method with the Best-Know Solutions (BKS) from the literature in terms of the Relative Percentage Deviation (RPD), computed according to Equation (25). TWCT\textsuperscript{Method} designates the total weighted completion time obtained with one run of a selected method on a PLSVSP instance, while TWCT\textsuperscript{BKS} is the total weighted completion time for the Best-Know Solution for the same instance. In all analyzes, we compare the matheuristics, running each one ten times per instance, with the solutions of Batch-WSPT (Section 3.1) and Batch-S (Section 3.2) formulations, running within a 6-hour time-limit. We limit the memory allocation to 10 GB for each method execution to allow multiple runs simultaneously using the available CPUs.

\[
RDP = 100 \times \frac{TWCT_{\text{Method}} - TWCT_{\text{BKS}}}{TWCT_{\text{BKS}}} \tag{25}
\]

The discussion of the results focuses only on comparing the ILS-Math\textsubscript{3} and the GRASP-Math\textsubscript{3} matheuristics with the pure mathematical formulations since these approaches presented the best results among the proposed matheuristic variants in a preliminary analysis, shown in Appendix C. We also included the complete results for each instance, considering all methods, in Appendix D.

### 7.1 Instances Description

The experiments were conducted on the benchmark of 72 PLSVSP instances\textsuperscript{1} developed by Abu-Marrul et al. (2020) in which the number of operations to schedule varies as \(|O|\in\{15, 25, 50\}\), and the number of machines varies as \(|M|\in\{4, 8\}\). The number of jobs is defined as \(|N| = |O|/3\), and the number of families is defined as \(|F| = 3\). The instances generation process, defined by the authors, with the distributions used, is detailed below.

Operations are randomly assigned to families, and family setup times are drawn from a discrete uniform distribution \(U(5, 10)\). Processing times of operations follow a discrete uniform distribution \(U(1, 30)\), and their occupancy values are drawn from another discrete uniform distribution \(U(1, 100)\) with step size 10. Job’s weights follows a discrete uniform distribution \(U(1, 50)\). Capacities of machines are drawn from a discrete uniform distribution \(U(0, MR)\), where \(MR\) is the maximum release date, defined as \(MR = \lfloor \alpha \cdot \sum_{i\in O}(p_i + s_f_i)/|M| \rfloor\). The release factor \(\alpha \in \{0.25, 0.5, 0.75\}\) defines different values for \(MR\). An eligibility factor \(\beta \in \{0.7, 0.9\}\), defines the probability of a machine being eligible to process each operation. Finally, a factor \(\gamma \in \{0.05, 0.15\}\) defines the probability of a job being associated with an operation.

### 7.2 Parameter Tuning

For parameter tuning, we used 12 medium-sized instances (25 operations) and 12 large-sized instances (50 operations), selected at random, corresponding to one-third of the total number of instances available in the benchmark. The Batch-WSPT formulation was used within the matheuristics during the parameterization. First, we set parameters related to the MIP-based Neighborhood Searches (Section 5), to further define each specific matheuristic parameter. To establish the neighborhoods parameters, we ran five times each neighborhood individually, using the solution generated by the \(\text{WMCT-WAVGA}\) heuristic (Algorithm 1) as a warm start. We define the ranges for the Batch Windows (Section 5.1), and for the Multi-Batches Relocate (Section 5.2) parameters as \(\rho \in [0.1, 0.5]\), and \(\varphi \in [0.1, 0.5]\), respectively, considering a step of size 0.05. After setting values for \(\rho\), and \(\varphi\), the ILS-Math was executed five times without accepting worse solutions. We used \(\Omega_{\text{max}} = 10\), limiting \(\omega \in [0.05, 0.15]\) with a step size of 0.05, to define the perturbation parameter. With \(\omega\) set, we ran the ILS-Math again five more times, limiting \(\delta \in [0.00, 0.15]\) with a step size of 0.05, to define the worst solution acceptance rate. The same steps were executed to define the greediness factor of the GRASP-Math, limiting \(\alpha \in [0.05, 0.15]\), with a step size of 0.05. The final values for the parameters are shown in Table 1.

### 7.3 Results Analysis and Discussion

#### 7.3.1 Average Values for the RPD and Computational Time

In the first analysis, shown in Table 2, we compare the four methods (Batch-WSPT, Batch-S, ILS-Math\textsubscript{3} and GRASP-Math\textsubscript{3}) in terms of the average RPD (RDP) and the average computational time (\(\text{time}\)), in seconds, for

\textsuperscript{1}The PLSVSP instances benchmark is available online Abu-Marrul et al. (2019)
Matheuristics for a Parallel Machine Scheduling Problem

### Table 1: Parameters definition

| Algorithm                        | Parameter | Description | Domain   | Value |
|----------------------------------|-----------|-------------|----------|-------|
| Batch Windows                    | $\rho$    | The proportion of the makespan ($C_{max}$) to optimize at each iteration | [0, 1]    | 0.20  |
| Multi-Batches Relocate           | $\varphi$ | The proportion of batches to optimize at each iteration | [0, 1]    | 0.30  |
| Random Batch Swap                | $\omega$  | The proportion of batch swaps to execute at each perturbation | [0, 1]    | 0.10  |
| ILS-Math                         | $\delta$  | Worse solutions acceptance rate | [0, 1]    | 0.00  |
| GRASP-Math                       | $\alpha$  | Greediness factor for the randomized constructive procedure | [0, 1]    | 0.10  |
| ILS-Math and GRASP-Math          | $\Omega_{max}$ | Maximum number of iterations without improvement | $\mathbb{Z}_+$ | 10    |

Each instance group. We use the grouping scheme defined by Abu-Marrul et al. (2020) for this analysis, in which each group, represented by the combination of the number of operations and machines ($|O|$ and $|M|$), comprises 12 instances. The best result for each criterion in each group is highlighted in bold.

### Table 2: Average values for the RPD and computational time distributions for each method in each instance group.

| $|O|$ | $|M|$ | Batch-WSPT | Batch-S | ILS-Math$_3$ | GRASP-Math$_3$ |
|-----|-----|-----------|---------|--------------|---------------|
|     |     | RPD | time | RPD | time | RPD | time | RPD | time |
| 15  | 4   | 0.49 | 20414 | **0.00** | 20725 | 0.42 | 14 | 0.30 | 26 |
|     | 8   | 0.04 | 18000 | **0.01** | 18000 | 0.08 | 46 | 0.01 | 54 |
| 25  | 4   | 0.70 | 21600 | 0.81 | 21600 | -0.17 | **123** | -0.27 | 186 |
|     | 8   | 0.23 | 21600 | 0.62 | 21600 | -0.20 | **244** | -0.27 | 297 |
| 50  | 4   | -0.27 | 19978 | ![†](Note) | 0.60 | 21600 | **-3.02** | **1052** | -2.65 | 1159 |
|     | 8   | -1.83 | 17380 | ![†](Note) | 1.90 | 13901 | **-4.09** | **990** | -3.80 | 1066 |

All instances -0.11 | ![†](Note) | 0.66 | 19571 | ![†](Note) | **-1.16** | **412** | -1.12 | 465 |

† CPLEX execution interrupted before 21,600 seconds (time limit) for some instances in this group, for this method, due to memory problems.

Note that the average relative percentage deviations are low for instances with 15 operations, with an $\overline{RPD}$ below 1% for all methods. Regarding group 15–4, the Batch-S formulation presented the smallest $\overline{RPD}$ with 0.00, that is, it reaches the BKS in all runs, but with an average computational time of 20,725 seconds. The average computational times for the MIP formulations in this group are close to the limit of 21,600 seconds (6 hours) previously defined. ILS-Math$_3$ runs with the least average computational time in this group (14 seconds). However, concerning the $\overline{RPD}$, GRASP-Math$_3$ is the best matheuristic approach ($\overline{RPD} = 0.30$), without significantly increasing computational time spent by ILS-Math$_3$. Similar behavior can be seen in group 15–8. GRASP-Math$_3$ presented equivalent results with Batch-S ($\overline{RPD} = 0.01$), but consuming less than a minute to achieve it. In contrast, Batch-S run for an average of 18,000 seconds.

For medium-sized groups (25–4 and 25–8), the matheuristics outperform the MIP formulations in terms of solution quality, with considerably less computing time. The MIP formulations maintained a good quality of the solutions, below 1% for the $\overline{RPD}$ in both groups, but running until the time limit in all executions ($\overline{time} = 21,600$). Between the matheuristics, GRASP-Math$_3$ outperforms ILS-Math$_3$ in terms of solution quality, with $\overline{RPD} = -0.31$ in group 25–4, and $\overline{RPD} = -0.27$ in group 25–8, but with 51% extra computational time needed on average in the former group (186 seconds against 123 seconds), and 22% in the latter (297 seconds against 244 seconds).

For large-sized groups, we can see that Batch-S loses performance, with the worst values for the $\overline{RPD}$ in both groups (0.60 in group 50–4 and 1.90 in group 50–8), resulting in a clear advantage for Batch-WSPT. Again, the matheuristics dominate the MIP formulations in both groups with lower values for the $\overline{RPD}$ and $\overline{time}$. However, in these groups, ILS-Math$_3$ dominates GRASP-Math$_3$, in terms of solution quality and computational time. It can be noted that the $\overline{time}$ for the MIP formulations on these groups is smaller than the limit of 21,600. This is due to memory issues, with
the solver interrupting the execution of 6 instances, in each formulation, before reaching the time limit. Appendix D details the complete results, indicating these instances.

One can note that, in all groups, the GRASP approach requires more computational time than ILS due to the method’s restart characteristic. This characteristic also affects the quality of the method’s solutions when the size of the instances increases. We can also observe that the average computational time for the matheuristics grows considerably when the size of problems grows, which, in our case, directly impacts the execution time of our matheuristics.

### 7.3.2 RPD Distribution for Different Instances Aspects

In this analysis, we evaluate the impacts on the solution quality of each method with the increasing number of machines and operations. Figures 5(a), 5(b), and 5(c) depict the distribution of the RPDs for instances with 15, 25, and 50 operations to schedule, respectively. Next, Figures 6(a) and 6(b) show the distribution of the RPDs for instances with 4, and 8 machines, respectively.

![Figure 5: Boxplots of the RPD distributions for each method, considering different number of operations.](image)

Regarding instances with 15 operations (Figure 5(a)), all methods are competitive (RPDs closest to zero). However, Batch-WSPT struggles to achieve the BKS due to the reduced solution space considered by the formulation. Note that the matheuristics have better performance than the MIP formulations when 25 and 50 operations are considered (Figures 5(b) and 5(c)). It is interesting to see the distributions’ behavior when the number of operations increases. For the MIP formulations, we can see that Batch-S performs better on small instances, while Batch-WSPT shows advantages when the number of operations increases. Note that on medium-sized instances (25 operations), they have similar performance. The consideration of extra variables and constraints to sequence operations within batches proved to be a good strategy for small instances since it considers the complete solution space of the problem, however affecting the performance of Batch-S when the number of operations increases. Regarding the matheuristics, we can see that they maintain similar distributions regardless of the number of operations, but with ILS-Math3 improving its performance, compared to GRASP-Math3 in instances with 50 operations.

The same behavior can be observed in the analysis by the number of machines (Figures 6(a) and 6(b)), with the matheuristics showing better distribution of RPDs. However, we can observe less dispersed distributions due to the consideration of instances with 15 operations in both groups. The MIP formulations maintained the same behavior, with Batch-S losing performance when the number of machines grows.

### 7.3.3 Comparison with the BKS Values

Now, we analyze the methods, in Table 3, regarding the achievement or improvement of the Best-Know Solutions from the literature, defined by Abu-Marrul et al. (2020). The results indicate the number of instances in which the BKS was achieved or improved (#Inst.), its percentage regarding the complete set of 72 PLSVSP instances (%Inst.), and the respective percentage of runs of achievement or improvement (%Runs). The RPD for the subsets of not achieved and not improved solutions are included. Finally, we show the minimum RPD ($RDP^-$) found by each method when the BKS is improved.
Matheuristics for a Parallel Machine Scheduling Problem

### Table 3: Analysis regarding BKS solutions from the literature.

| Method      | Achieved BKS\(^\dagger\) | Not Achieved | Improved BKS | Not Improved |
|-------------|--------------------------|--------------|--------------|--------------|
|             | #Inst. | %Inst. | %Runs | RPD\(|\) | #Inst. | %Inst. | %Runs | RPD\(|\) | RPD\(|\) |
| Batch-WSPT  | 42     | 58.33  | 58.33 | 1.16 | 23     | 31.94  | 31.94 | -5.17 | 0.71 |
| Batch-S     | 41     | 56.94  | 56.94 | 2.06 | 13     | 18.06  | 18.06 | -4.08 | 1.08 |
| ILS-Math\(_3\) | 69     | 95.83  | 81.94 | 0.72 | 40     | 55.56  | 49.72 | -10.96 | 0.26 |
| GRASP-Math\(_3\) | 72     | 100.00 | 86.11 | 0.59 | 40     | 55.56  | 50.56 | -9.72 | 0.17 |

\(^\dagger\) This subset includes solutions that have also improved the BKS.

Note that all methods achieve the BKS in more than 50% of instances, with GRASP-Math\(_3\) reaching it in all instances (\#Inst. = 72). The %Inst. and %Run has the same value for each MIP formulation, as we only run it once for each instance. Again, the matheuristics have better performance than the MIP formulations. Note that the \(\text{RPD}\) for runs where BKS was not found is less than 1% for the matheuristics and more than 1% for the MIP formulations, with the worst performance for Batch-S formulation (\(\text{RPD} = 2.06\%\)). It can be seen that the matheuristics ILS-Math\(_3\) and GRASP-Math\(_3\) achieve the BKS in 81.94% and 86.11% of the executions, respectively. The same advantage for the matheuristics can be seen for improved solutions, with both methods improving the BKS in about 50% of runs. The \(\text{RPD}\) is low for runs that have not improved the BKS (0.26% for ILS-Math\(_3\), and 0.17% for GRASP-Math\(_3\)), as it includes runs that have achieved the BKS. With these results, we can see that 40 new best solutions are found by the matheuristics, with an improvement of 10.96% in the objective value of the solution in the best case (\(\text{RPD}\) of ILS-Math\(_3\)). The specific instances with enhanced solutions are indicated in Appendix D. Considering that 26 instances have the proven optimal solutions (see Table 12), 40 new best solutions have been defined in 46 possible instances.

#### 7.3.4 Statistical Analysis for the RPD Distributions

To improve our discussion and validate what we highlighted during the previous analyses, we applied a statistical evaluation comparing the RPD distributions between the methods, considering the complete set of 72 PLSVSP instances. First, we tested the distributions’ normality with the Shapiro-Wilk test, which shows that the RPDs do not follow a normal distribution. Then, we ran the pairwise Wilcoxon rank-sum test with Hommel’s \(p\)-values adjustment. We also included the Analysis of Variance (ANOVA) with the Tukey HSD (honestly significant difference) test to compare the method’s RPD distributions. We ran both tests with a confidence level of 0.05. In Table 4, we present the \(p\)-value for each pair of methods, also including some statistics on the RPD distributions. Significantly better results are highlighted in bold.

The tests confirm that the matheuristics yield significantly better results than the MIP formulations. Note that no statistical significance can be seen when comparing one MIP formulation with another, nor when comparing the
Table 4: The mean and standard deviation of the RPD distributions for each method, and $p$-values from pairwise Wilcoxon rank-sum and ANOVA with Tukey HSD tests, with a 0.05 confidence level, considering all 72 PLSVSP instances.

| Method       | $\text{RPD}$ | $\sigma$ | Wilcoxon $p$-value (ANOVA–Tukey $p$-value) |
|--------------|--------------|----------|------------------------------------------|
| Batch-WSPT   | -0.10        | 1.78     | Batch-WSPT                                |
| Batch-S      | 0.66         | 2.10     | Batch-S                                  |
| GRASP-Math3  | -1.16        | 2.18     | 0.00 (0.00)                              |
| ILS-Math3    | -1.12        | 2.15     | 0.00 (0.00)                              |

Matheuristics between each other. Wilcoxon and ANOVA–Tukey agreed in all cases despite discrepancies in the $p$-values.

7.3.5 Average RPD Evolution Analysis

As we evaluate solutions for a realistic process, it is important to analyze the final solutions of the proposed methods and their evolution over time, in case the company needs faster solutions. Based on this, we show in Figure 7 the evolution of the solution for the two best approaches, ILS-Math3 and GRASP-Math3, illustrating the trade-off between the quality of the solution and the time spent for achieving it. For better visualization, the computational time axis is on a $\log_{10}$ scale. As the deviations and computational times are low for small-sized instances, the curve's shape is more defined by the medium and large-sized instances.

Figure 7: Average RPD (vertical axis) evolution over the computational time (horizontal axis – $\log_{10}$ scale) for each matheuristic, running on the complete set of 72 PLSVSP instances.

Note that both matheuristics surpass the BKS on average (that is, the current method used to solve the PLSVSP), crossing the zero line, in less than 1 minute (38 seconds for ILS-Math3, and 49 seconds for GRASP-Math3). The ILS-Math3 matheuristic dominates the GRASP-Math3 but reaching an equivalent average RPD after crossing the zero line. This analysis reinforces the advantages of using the matheuristics concept instead of running the MIP formulations, showing that the additional time-limit criteria can be included, if necessary, without significantly impacting the quality of the solution.

8 Conclusions

In this paper, we studied a ship scheduling problem related to offshore oil & gas logistics, in which a company needs to schedule its fleet of Pipe Laying Support Vessels (PLSVs). These vessels are responsible for connecting sub-sea oil wells to the platforms, allowing production to begin. The company intends to anticipate the start of production in the most promising wells. The PLSV Scheduling Problem (PLSVSP) can be seen as a variant of an identical parallel machine scheduling problem with batching and non-anticipatory family setup times to minimize the total weighted
completion time. In this analogy, the vessels are the machines, the jobs are the wells, and each job includes several connection operations to be performed.

We introduced an ILS and a GRASP matheuristics, using two MIP-based neighborhood searches and a constructive heuristic to solve the PLSVSP. Two MIP formulations are considered, resulting in three variants of each method. The first is a batch formulation, which uses a WSPT dispatching rule to sequence operations within batches, while the second is a new formulation that considers sequencing within batches as a model decision. The results show that the matheuristics outperform the pure mathematical programming models in terms of computational time and solution quality. Among the matheuristics, a small advantage can be observed for two variants that combine the use of two batch formulations for the PLSVSP. The analysis shows that our new proposed formulation with sequencing variables helps the matheuristics to improve the quality of the solution. Moreover, new best solutions are provided for 40 of the 46 possible instances (without proven optimal solutions) on the PLSVSP instances benchmark.

The present study reinforces the importance of hybrid methods and their applicability in practical and theoretical contexts. The concept of splitting the problem to solve sub-problems is a relevant approach, especially when the size of the problem increases. It is worth mentioning that the approach is closely related to the nature of the PLSVSP in the studied company. In this environment, management guidelines for dealing with the problem change rapidly due to political and operational issues, requiring a quick adjustment in decision support tools. The use of mathematical formulation facilitates these adjustments, allowing the inclusion or exclusion of constraints without the need for substantial computational development in the algorithms. Nevertheless, regular metaheuristics based on local search procedures can be an interesting research direction due to the complexity of the problem, allowing other researchers working with similar or simplified versions of the PLSVSP to benefit from these studies. Moreover, the offshore environment where the PLSVs operate is surrounded by uncertainties caused by several aspects, such as climate changes (affecting ocean conditions), the complexity of operations, crew experience, and others, with significant impacts on the operating companies’ planning. Thus, another attractive field for future research would be to understand the most impacted aspects of the problem and design stochastic optimization tools for the PLSVSP with uncertainties.

Acknowledgements

This study was financed in part by PUC-Rio, by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001 and by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) under Grant numbers 403863/2016-3, 306802/2015-5, 425962/2016-4 and 313521/2017-4.

A Parallel machine and PLSVSP definitions

This section presents a mapping between the addressed identical parallel machine scheduling problem and its correspondence with the PLSVSP aspects (Table 5).

B Details of the example instance

The details of the instance used to present a PLSV schedule example (Figure 1), are given below. The details of the instance used to present a PLSV schedule example (Figure 1), are given below. The instance is composed by 15 operations (|O| = 15), 5 jobs (|N| = 5) and 4 machines (|M| = 4). Table 6 shows the processing time (p_i), the release date (r_i), the family (f_i), the load occupancy (l_i), the set of associated jobs (N_i), and the eligible set of machines (M_i) of each operation i. Table 7 presents the weight (w_j), and the subset of associated operations (O_j) of each job j. Table 8 shows the release date (r_k), and the capacity (q_k) of each machine k, together with the subset of operations (O_k) that each machine k is eligible to execute. Table 9 presents the setup time (s_f) of each family f.

C Matheuristics Comparative Analysis

In this section, we compare the results for the matheuristic variations. In Table 10 the results for the ILS-Math variations are shown in terms of the average RPD (RPD) and average computational time (time). The same results but for the GRASP-Math variations are presented in Table 11.

Note that results for the ILS-Math are quite similar for all variations. However, we can see an advantage for the ILS-Math_3 approach when the size of the instances increases (instances with 25 and 50 operations). Regarding the
Table 5: The mapping between the parallel machine scheduling problem definitions and its correspondences on the PLSVSP.

| Name     | Machine scheduling definition                                                                 | PLSVSP correspondence                                                                 |
|----------|---------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------|
| \( O \) | Operations                                                                                 | Pipeline connections                                                                   |
| \( N \) | Jobs                                                                                       | Wells                                                                                  |
| \( M \) | Machines                                                                                   | PLSVs                                                                                  |
| \( F \) | Families                                                                                   | Groups of pipeline connections with similar loading process at the port                |
| \( M_i \) | Machine eligibility subset                                                                  | Subset of vessels eligible to execute a pipeline connection                              |
| \( O_{pk} \) | Subset of operations a machine is eligible to execute                                      | Subset of connections a vessel can carry                                                |
| \( O_j \) | Subset of operations composing a job                                                        | Subset of connections required to enable a well to start producing                     |
| \( N_i \) | Subset of jobs associated to an operation                                                   | Subset of wells that depends on a connection to be able to produce                     |
| \( p_i \) | Processing time of an operation                                                            | Time taken to perform a pipeline connection                                             |
| \( r_i \) | Release date of an operation                                                                | Arrival date of a pipeline at the port                                                 |
| \( l_i \) | Load occupancy of an operation                                                              | Pipeline occupancy on the deck of the ship                                             |
| \( f_i \) | Family of an operation                                                                       | Group of a pipeline connection                                                        |
| \( r_k \) | Release date of a machine                                                                    | Availability date of a vessel                                                         |
| \( q_k \) | Capacity of a machine                                                                       | Vessel's deck capacity                                                                  |
| \( w_j \) | Weight of a job                                                                            | Production potential of a well                                                        |
| \( C_i \) | Completion time of an operation                                                             | Time when a connection ends                                                            |
| \( C_j \) | Completion time of a job                                                                     | Time when a well is fully connected and able to start producing                       |
| \( s_f \) | Setup times of a family                                                                     | Time spent at port loading pipelines for a specific group of operations                |

Batch: Sequence of operations from the same family sharing the same setup time

Table 6: Operations aspects regarding the example instance.

| Operation (i) | \( p_i \) | \( r_i \) | \( f_i \) | \( l_i \) | \( N_i \) | \( M_i \) |
|---------------|---------|--------|--------|--------|----------|----------|
| 1             | 23      | 5      | 1      | 90     | [5]      | {1,4}    |
| 2             | 5       | 17     | 1      | 90     | [5]      | {1,3,4}  |
| 3             | 25      | 16     | 3      | 70     | [2,3,4]  | {2,3,4}  |
| 4             | 10      | 9      | 2      | 50     | [1]      | {1,2,3,4}|
| 5             | 4       | 18     | 1      | 60     | [3]      | {1,2,3,4}|
| 6             | 8       | 17     | 2      | 80     | {1,3}    | {2,4}    |
| 7             | 29      | 15     | 3      | 60     | [5]      | {1,2,3,4}|
| 8             | 8       | 5      | 3      | 0      | [2]      | [2]      |
| 9             | 17      | 9      | 3      | 40     | [1,5]    | {1,2,4}  |
| 10            | 25      | 0      | 1      | 40     | [2,5]    | {1,2,3,4}|
| 11            | 4       | 3      | 1      | 30     | [3]      | {1,2,3,4}|
| 12            | 15      | 0      | 1      | 20     | [1,3]    | {1,3,4}  |
| 13            | 7       | 16     | 2      | 60     | [3]      | {3,4}    |
| 14            | 7       | 7      | 1      | 90     | [4,5]    | {1,4}    |
| 15            | 18      | 15     | 3      | 20     | [3]      | {4}      |

Table 7: Jobs aspects regarding the example instance.

| Job (j) | \( w_j \) | \( O_j \) |
|---------|---------|----------|
| 1       | 46      | {6,9,11,12} |
| 2       | 40      | {3,8,10}  |
| 3       | 39      | {3,4,5,6,12,13,15} |
| 4       | 13      | {3,14}    |
| 5       | 3       | {1,2,7,9,10,14} |
Table 8: Machines aspects regarding the example instance.

| Machine \((k)\) | \(r_k\) | \(q_k\) | \(O_k\) |
|----------------|--------|--------|--------|
| 1              | 13     | 90     | \{1,2,4,5,7,9,10,11,12,14\} |
| 2              | 0      | 80     | \{3,4,5,6,7,8,9,10,11\}    |
| 3              | 0      | 90     | \{2,3,4,5,7,10,11,12,13\}  |
| 4              | 1      | 90     | \{1,2,3,4,5,6,7,9,10,11,12,13,14,15\} |

Table 9: Family setup time duration regarding the example instance.

| Family \((f)\) | \(s_f\) |
|---------------|--------|
| 1             | 5      |
| 2             | 7      |
| 3             | 9      |

complete set of instances, ILS-Math\(_3\) advantage is clearer with it dominating the other approaches in terms of the \(R^PD\) with \(-1.16\%\).

Similar behavior can be noted for the GRASP-Math variations. GRASP-Math\(_3\) presented the best overall \(R^PD\) of \(-1.119\%\) with a small difference in the average computational time for the best approach in this criterion (465 seconds against 456 seconds for the GRASP-Math\(_1\)).

Table 10: Average results for ILS-Math variations.

| \(|O|\) | \(|M|\) | ILS-Math\(_1\) | ILS-Math\(_2\) | ILS-Math\(_3\) |
|--------|--------|----------------|----------------|----------------|
|        |        | \(R^PD\) | time | \(R^PD\) | time | \(R^PD\) | time |
| 15     | 4      | 0.61  | 14   | 0.33  | 18   | 0.42  | 14   |
|        | 8      | 0.11  | 46   | 0.03  | 49   | 0.08  | 46   |
| 25     | 4      | 0.06  | 123  | -0.13 | 143  | -0.17 | 123  |
|        | 8      | -0.15 | 247  | -0.18 | 267  | -0.20 | 244  |
| 50     | 4      | -2.78 | 1054 | -2.74 | 984  | -3.02 | 1052 |
|        | 8      | -3.99 | 972  | -3.86 | 994  | -4.09 | 990  |

| All instances | -1.02 | 409 | -1.09 | 409 | -1.16 | 412 |

D Results by Instance

In this section, we present the total weighted completion time (objective function) found by each method for the benchmark of 72 PLSVSP instances with the respective computational times (Table 12). The first two columns indicate the name of the instance and its BKS from the literature (optimal solutions are indicated with an asterisk), defined by [Abu-Marrul et al. (2020)]. Then, the upper bound (ub) and the computational time (time) for each formulation are presented, followed by the results of each matheuristic variation. The matheuristic results are presented in the following order: minimum total weighted completion time (min), average total weighted completion time (avg), and average computational time (\(\text{time}\)), among the runs. We highlighted, in bold, the best solution for each instance. We suppress computational times that have reached the time limit (21,600 seconds) defined for the MIP formulations.

To save space, we shorten the instance names. For example, in the benchmark set, an instance named PLSV\(_{o15\_n5\_q3\_m4\_111\_1}\), indicates a total of 15 operations (\(o15\)), associated with 5 jobs (\(n5\)), divided into 3 families (\(q3\)), to be scheduled on 4 machines (\(m4\)). Since the number of jobs depends on the number of operations (calculated as \(|N| = |O|/3\)), following the instance generation procedure defined by [Abu-Marrul et al. (2020)], we delete it from the instance name. We also suppress the number of families and the last digit because they are equal to 3 and 1 in all instances.
Table 11: Average results for GRASP-Math variations.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| Instance | BKS | Batch-WiPT | Batch-S | ILS-Math1 | ILS-Math2 | GRASP-Math1 | GRASP-Math2 | GRASP-Math3 |
|----------|-----|------------|---------|----------|----------|-------------|-------------|-------------|
| 15-14-11| 6903*  | 6927      | 6903   | 6927   | 6927  | 6903  | 6903  | 6903  |
| 15-14-12| 7717*  | 7849      | 7736.4 | 7736.4 | 7736.4 | 7736.4 | 7736.4 | 7736.4 |
| 15-14-13| 7498    | 7498      | 7389   | 7389   | 7389   | 7389   | 7389   | 7389   |
| 15-14-14| 7747    | 7747      | 7345.6 | 7345.6 | 7345.6 | 7345.6 | 7345.6 | 7345.6 |
| 15-14-15| 8608    | 8608      | 8268.0 | 8268.0 | 8268.0 | 8268.0 | 8268.0 | 8268.0 |
| 15-14-16| 10964   | 10964 15064 | 10964   | 10964   | 10964   | 10964   | 10964   | 10964   |
| 15-14-17| 10685*  | 10685    | 10685  | 10685  | 10685  | 10685  | 10685  | 10685  |
| 15-14-18| 8822*   | 8822     | 8822   | 8822   | 8822   | 8822   | 8822   | 8822   |
| 15-14-19| 6579*   | 6579     | 6681.4 | 6681.4 | 6681.4 | 6681.4 | 6681.4 | 6681.4 |
| 15-14-20| 5929*   | 5929     | 5929   | 5929   | 5929   | 5929   | 5929   | 5929   |
| 15-14-21| 4545    | 4545     | 4545   | 4545   | 4545   | 4545   | 4545   | 4545   |
| 15-14-22| 10964   | 10964 15064 | 10964   | 10964   | 10964   | 10964   | 10964   | 10964   |
| 15-14-23| 10685*  | 10685    | 10685  | 10685  | 10685  | 10685  | 10685  | 10685  |
| 15-14-24| 8822*   | 8822     | 8822   | 8822   | 8822   | 8822   | 8822   | 8822   |
| 15-14-25| 6579*   | 6579     | 6681.4 | 6681.4 | 6681.4 | 6681.4 | 6681.4 | 6681.4 |
| 15-14-26| 5929*   | 5929     | 5929   | 5929   | 5929   | 5929   | 5929   | 5929   |
| 15-14-27| 4545    | 4545     | 4545   | 4545   | 4545   | 4545   | 4545   | 4545   |
| 15-14-28| 10964   | 10964 15064 | 10964   | 10964   | 10964   | 10964   | 10964   | 10964   |
| 15-14-29| 10685*  | 10685    | 10685  | 10685  | 10685  | 10685  | 10685  | 10685  |
| 15-14-30| 8822*   | 8822     | 8822   | 8822   | 8822   | 8822   | 8822   | 8822   |
| 15-14-31| 6579*   | 6579     | 6681.4 | 6681.4 | 6681.4 | 6681.4 | 6681.4 | 6681.4 |
| 15-14-32| 5929*   | 5929     | 5929   | 5929   | 5929   | 5929   | 5929   | 5929   |
| 15-14-33| 4545    | 4545     | 4545   | 4545   | 4545   | 4545   | 4545   | 4545   |
| 15-14-34| 10964   | 10964 15064 | 10964   | 10964   | 10964   | 10964   | 10964   | 10964   |
| 15-14-35| 10685*  | 10685    | 10685  | 10685  | 10685  | 10685  | 10685  | 10685  |
| 15-14-36| 8822*   | 8822     | 8822   | 8822   | 8822   | 8822   | 8822   | 8822   |
| 15-14-37| 6579*   | 6579     | 6681.4 | 6681.4 | 6681.4 | 6681.4 | 6681.4 | 6681.4 |
| 15-14-38| 5929*   | 5929     | 5929   | 5929   | 5929   | 5929   | 5929   | 5929   |
| 15-14-39| 4545    | 4545     | 4545   | 4545   | 4545   | 4545   | 4545   | 4545   |
| 15-14-40| 10964   | 10964 15064 | 10964   | 10964   | 10964   | 10964   | 10964   | 10964   |

Table 12: Complete results by instance regarding the PLSVSP benchmark.
Table 12 – continued from previous page

| Instance | BKS | Batch-WSPT | Batch-S | ILS-Math | GRASP-Math | GRASP-Math | GRASP-Math |
|----------|-----|------------|--------|----------|------------|------------|------------|
|          | ub time | ub time | avg | Time | min avg | Time | min avg | Time |
| 50-4-112 | 102246 | 102201 | 102283 | 98719.3 | 1165 | 98116 | 98668.3 | 1127 |
| 50-4-121 | 65231 | 65185 | 65378 | 63947.6 | 820 | 63395 | 63648.4 | 1038 |
| 50-4-131 | 92825 | 92011 | 91497 | 91478.9 | 1165 | 91341 | 91666.2 | 1170 |
| 50-4-132 | 99080 | 99690 | 98343 | 98593.0 | 803 | 98382 | 98782.8 | 1095 |
| 50-4-211 | 84383 | 82439 | 81604 | 80866.1 | 1160 | 80254 | 80681.2 | 1110 |
| 50-4-212 | 87761 | 83410 | 81350 | 82024.4 | 1210 | 81145 | 81713.8 | 1177 |
| 50-4-221 | 106430 | 105371 | 104586 | 101723 | 102670.7 | 1177 | 101794 | 102557.0 | 1322 |
| 50-4-222 | 105136 | 104714 | 107833 | 129575 | 130260.5 | 1177 | 130160 | 131671.3 | 1234 |
| 50-4-231 | 89865 | 85913 | 87968 | 85701.7 | 997 | 85456 | 85827.3 | 1401 |
| 50-4-232 | 131034 | 133194 | 134424 | 130959 | 131425 | 1177 | 130160 | 131671.3 | 1234 |
| 50-8-112 | 34381 | 33354 | 35691 | 35441.7 | 997 | 33402 | 33669.0 | 1029 |
| 50-8-121 | 31944 | 30906 | 32771 | 30393 | 30564.7 | 997 | 30276 | 30493.0 | 1127 |
| 50-8-131 | 60738 | 59588 | 60242 | 59540.0 | 673 | 59377 | 59538.9 | 857 |
| 50-8-132 | 65103 | 64848 | 66000 | 65190 | 65484.5 | 968 | 59410 | 59651.2 | 951 |
| 50-8-211 | 48355 | 48910 | 48461 | 48519.3 | 1074 | 45596 | 46078.9 | 899 |
| 50-8-212 | 42961 | 41870 | 43002 | 4111 | 41497 | 890 | 41101 | 41496.5 | 1077 |
| 50-8-221 | 35471 | 36247 | 5467 | 36632 | 54950.0 | 673 | 59111 | 59483.5 | 765 |
| 50-8-222 | 56600 | 54512 | 57001 | 52738 | 53360.1 | 997 | 52976 | 53645.4 | 968 |
| 50-8-231 | 54080 | 53303 | 53941 | 53894 | 53404.6 | 1138 | 53892 | 54108.0 | 1187 |
| 50-8-232 | 62858 | 62385 | 6017 | 67546 | 61796.8 | 856 | 61852 | 62099.2 | 724 |

† Execution interrupted before 21,600 seconds (time limit) due to memory limit.
References

Ricardo L Carneiro Beltrao, Cristiano Leite Sombra, Antonio Carlos VM Lage, Jose Roberto Fagundes Netto, Carlos Cunha Dias Henriques, et al. Pre-salt santos basin-challenges and new technologies for the development of the pre-salt cluster, santos basin, brazil (otc-19880). In Offshore Technology Conference, Houston, Texas, USA, 4-7 May 2009. doi:10.4043/19880-MS

Ali Allahyarzadeh-Bidgoli, Leandro Oliveira Salviano, Daniel Jonas Dezan, Silvio de Oliveira Junior, and Jurandir Itizo Yanagihara. Energy optimization of an FPSO operating in the brazilian pre-salt region. Energy, 164:390–399, 2018.

Eduardo Haddad and Ana Carolina Giuberti. Economic impacts of pre-salt on a regional economy: the case of espirito santo, brazil. In 50th Congress of the European Regional Science Association: “Sustainable Regional Growth and Development in the Creative Knowledge Economy”, Jönköping, Sweden, 19-23 August 2010. European Regional Science Association (ERSA).

James G Speight. Handbook of Offshore Oil and Gas Operations. Elsevier, 2015.

Judimar Clevelario, Fabio Pires, Claudio Barros, and Terry Sheldrake. Flexible pipe systems configurations for the pre-salt area. In ASME 2010 29th International Conference on Ocean, Offshore and Arctic Engineering, pages 457–464. American Society of Mechanical Engineers Digital Collection, 2010.

Maciel Manoel Queiroz and Andre Bergsten Mendes. Heuristic approach for solving a pipe layer fleet scheduling problem. In Enrico Rizzuto and C Guedes Soares, editors, Sustainable Maritime Transportation and Exploitation of Sea Resources, chapter 9, pages 1073–1080. Taylor & Francis Group, London, 2011.

L. H. Bremenkamp, V. A. M. C Cunha, Silvio Hamacher, Luciana Pessoa, and Paulo Cesar Ribas. Modelo de programação matemática para apoio à decisão da programação de embarcações PLSV. In XLVIII Simpósio Brasileiro de Pesquisa Operacional, pages 2574–2584, sep 2016. URL http://www.din.uem.br/sbpo/sbpo2016/pdf/156478.pdf (in Portuguese).

Victor Cunha, Iuri Santos, Silvio Hamacher, and Luciana Pessoa. An ils heuristic for a real-life ship scheduling problem. In The 12th edition of the Metaheuristics International Conference (MIC), pages 552–554, Barcelona, Spain, 4-7 July 2017.

Victor Cunha, Iuri Santos, Luciana Pessoa, and Silvio Hamacher. An ILS heuristic for the ship scheduling problem: application in the oil industry. International Transactions in Operational Research, 27(1):197–218, 2020.

Victor Abu-Marrul, Rafael Martinelli, and Silvio Hamacher. Scheduling Pipe Laying Support Vessels with Non-Anticipatory Family Setup Times and Intersections between Sets of Operations. International Journal of Production Research, (Pre-proof), aug 2020.

Victor Abu-Marrul, Davi Mecler, Rafael Martinelli, Silvio Hamacher, and Irina Gribkovskaia. Heuristics for Scheduling Pipe-laying Support Vessels: An Identical Parallel Machine Scheduling Approach. In 17th International Workshop on Project Management and Scheduling, 2021.

Jonathan M Thompson. Exact or metaheuristic methods or a bit of both—the rise of matheuristics. Keynote Papers, page 32, 2018.

Jean-Charles Billaut, Federico Della Croce, and Andrea Grosso. A single machine scheduling problem with two-dimensional vector packing constraints. European Journal of Operational Research, 243(1):75–81, 2015.

Ali Ekici, Milad Elyasi, Okan Örsan Özener, and Merve Burcu Sarıkaya. An application of unrelated parallel machine scheduling with sequence-dependent setups at vestel electronics. Computers & Operations Research, 111:130–140, 2019.

Emine Akyol Ozer and Tugba Sarac. MIP models and a matheuristic algorithm for an identical parallel machine scheduling problem under multiple copies of shared resources constraints. Top, 27(1):94–124, 2019.

Young-Bin Woo and Byung Soo Kim. Matheuristic approaches for parallel machine scheduling problem with time-dependent deterioration and multiple rate-modifying activities. Computers & Operations Research, 95:97–112, 2018.

Luis Fanjul-Peyro, Federico Perea, and Rubén Ruiz. Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. European Journal of Operational Research, 260(2):482–493, 2017.

Quang Chieu Ta, Jean-Charles Billaut, and Jean-Louis Bouquard. Matheuristic algorithms for minimizing total tardiness in the m-machine flow-shop scheduling problem. Journal of Intelligent Manufacturing, 29(3):617–628, 2018.

Federico Della Croce, Andrea Grosso, and Fabio Salassa. A matheuristic approach for the two-machine total completion time flow shop problem. Annals of Operations Research, 213(1):67–78, 2014.
Federico Della Croce, Andrea Grosso, and Fabio Salassa. Minimizing total completion time in the two-machine no-idle no-wait flow shop problem. *Journal of Heuristics*, 2019. doi:10.1007/s10732-019-09430-z.

Shih-Wei Lin and Kuo-Ching Ying. Optimization of makespan for no-wait flowshop scheduling problems using efficient matheuristics. *Omega*, 64:115–125, 2016.

Shih-Wei Lin and Kuo-Ching Ying. Make-span optimization in a no-wait flowline manufacturing cell with sequence-dependent family setup times. *Computers & Industrial Engineering*, 128:1–7, 2019.

Lars Mönch and Sebastian Roob. A matheuristic framework for batch machine scheduling problems with incompatible job families and regular sum objective. *Applied Soft Computing*, 68:835–846, 2018.

Rafael Martellini, Jean Collard, and Michel Gamache. Strategic planning of an underground mine with variable cut-off grades. *Optimization and Engineering*, pages 1–47, 2019.

Thomas Kalinowski, Jason Matthews, and Hamish Waterer. Scheduling of maintenance windows in a mining supply chain rail network. *Computers & Operations Research*, 115:104670, 2020.

Florian Grenouilleau, Nadia Lahrichi, and Louis-Martin Rousseau. New decomposition methods for home care scheduling with predefined visits. *Computers & Operations Research*, 115:104855, 2020.

Michael Pinedo. *Scheduling. Theory, algorithms, and systems*. Springer, 2012.

H. J. Shin and V. Jorge Leon. Scheduling with product family set-up times: an application in TFT LCD manufacturing. *International Journal of Production Research*, 42(20):4235–4248, 2004. ISSN 0020-7543. doi:10.1080/00207540410001708461

Jeffrey E. Schaller. Minimizing total tardiness for scheduling identical parallel machines with family setups. *Computers & Industrial Engineering*, 72:274–281, jun 2014. ISSN 03608352. doi:10.1016/j.cie.2014.04.001

Ali Obeid, Stéphane Dauzère-Pérès, and Claude Yugma. Scheduling job families on non-identical parallel machines with time constraints. *Annals of Operations Research*, 213(1):221–234, feb 2014. ISSN 0254-5330. doi:10.1007/s10479-012-1107-4

Michele Ciavotta, Carlo Meloni, and Marco Pranzo. Speeding up a Rollout algorithm for complex parallel machine scheduling. *International Journal of Production Research*, 54(16):4993–5009, aug 2016. ISSN 0020-7543. doi:10.1080/00207543.2016.1157276

Ali Allahverdi. The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2):345–378, oct 2015. ISSN 03772217. doi:10.1016/j.ejor.2015.04.004

Chris N Potts and Mikhail Y Kovalyov. Scheduling with batching: A review. *European Journal of Operational Research*, 120(2):228–249, 2000. doi:10.1016/j.ejor.2006.06.060

Pierre Hansen and Nenad Mladenović. Variable neighborhood search. In *Handbook of metaheuristics*, pages 145–184. Springer, 2003.

Helena R Lourenço, Olivier C Martin, and Thomas Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

Hugo Vinícius Bassi, Virgílio Jose Martins Ferreira Filho, and Laura Bahiense. Planning and scheduling a fleet of rigs using simulation–optimization. *Computers & Industrial Engineering*, 63(4):1074–1088, 2012.

Francisco J Rodríguez, Christian Blum, Carlos García-Martínez, and Manuel Lozano. GRASP with path-relinking for the non-identical parallel machine scheduling problem with minimising total weighted completion times. *Annals of Operations Research*, 201(1):383–401, 2012.

Susan K Heath, Jonathan F Bard, and Douglas J Morrice. A GRASP for simultaneously assigning and sequencing product families on flexible assembly lines. *Annals of Operations Research*, 203(1):295–323, 2013.

Mauricio G. C. Resende and Celso C. Ribeiro. *Greedy Randomized Adaptive Search Procedures: Advances and Extensions*, chapter 6, pages 169–220. Springer International Publishing, Cham, 2019. ISBN 978-3-319-91086-4. doi:10.1007/978-3-319-91086-4_6 URL https://doi.org/10.1007/978-3-319-91086-4_6

Victor Abu-Marrul, Rafael Martinelli, and Silvio Hamacher. Instances for the PLSV scheduling problem: An identical parallel machine approach with non-anticipatory family setup times, 2019. URL https://doi.org/10.17771/PUCRio.ResearchData.45799