Advanced Algorithms for the Reclaimer Scheduling Problem with Sequence-Dependent Setup Times and Availability Constraints

Oualid Benbrik$^{[0009-0009-3404-4136]1}$, Rachid Benmansour$^{[0000-0003-2553-4116]1,2}$, Abdelhak Elidrissi$^{[0000-0002-5024-6610]3}$, and Angelo Sifaleras$^{[0000-0002-5696-7021]4}$

[1] SI2M Laboratory INSEA, Rabat, Morocco
{obenbrik,r.benmansour}@insea.ac.ma

[2] LAMIH CNRS UMR 8201, INSA Hauts-de-France, Polytechnic University of Hauts-de-France (UVHC), Campus Mont Houy, F-59313 Valenciennes Cedex 9, France

[3] Rabat Business School, International University of Rabat, Parc Technopolis, Rabat, Morocco
abdelhak.elidrissi@uir.ac.ma

[4] Department of Applied Informatics, University of Macedonia, School of Information Sciences, 156 Egnatias Str., 54636, Thessaloniki, Greece
sifalera@uom.gr

Abstract. Scheduling of reclaimers activities in dry bulk terminals significantly impact terminal throughput, a crucial performance indicator for such facilities. This study addresses the Reclaimer Scheduling Problem (RSP) while considering periodic preventive maintenance activities for reclaimers. These machines are integral for reclaiming dry bulk materials stored in stockyards, facilitating their loading onto vessels via ship-loaders. The primary aim of the objective function entails the minimization of the overall completion time, commonly referred to as the makespan. Since this problem is $\mathcal{NP}$-hard, we propose a novel greedy constructive heuristic. The solutions obtained from this heuristic serve as the starting point for an efficient General Variable Neighborhood Search (GVNS) algorithm to handle medium-scale instances resembling real stockyard configurations. Computational experiments are conducted by comparing the proposed methods across various problem instances. The results demonstrate that the developed GVNS, coupled with the constructive heuristic for initial solution finding, efficiently improves scheduling efficacy. Thus, it emerges as a new state-of-the-art algorithm for this problem.

Keywords: Reclaimer Scheduling, Bulk Ports, Sequence-Dependent Setup Times, Availability Constraints, Machine Eligibility Restrictions, Variable Neighborhood Search, Heuristic.

1 Introduction and Literature Review

Bulk terminals play a pivotal role in global trade by facilitating the efficient handling and storage of large quantities of commodities, such as coal, minerals, grains, raw materials, and so on. These terminals serve as crucial nodes in the logistics chain, ensuring the seamless flow of goods between various modes of transportation. The importance of bulk terminals cannot be overstated, given their pivotal role in maritime transport, which handles approximately 80% of the world's trade volume, as reported by the United Nations Conference on Trade and Development (UNCTAD 2022) [15]. Despite their indispensable contribution to global trade, bulk terminals have not received proportionate attention in the research literature when compared to container terminals.

While container terminals have been extensively studied, the operational challenges specific to bulk terminals have been relatively understudied. However, recent research is placing a growing emphasis on understanding and addressing the distinctive challenges faced by bulk terminals.

The overall configuration of dry bulk terminals involves a designated berth area where vessels anchor for the loading or unloading of materials, utilizing shiploaders or cranes. Complementing this, the terminal features a yard where bulk cargoes are managed, either through addition as stockpiles using stacker machines or complete reclamation using reclaimer machines, facilitating subsequent delivery to ships at the berths. The research at hand is prompted by a keen interest in the operational intricacies of bulk ports, with a specific emphasis on the Newcastle Coal Infrastructure Group (NCIG) terminal, a notable coal export terminal in Australia [7]. The NCIG stockyard incorporates diverse stockpads, each tailored with specific positions for unloaded coal. Rail tracks are strategically positioned between parallel stockpads, accommodating stacker-reclaimer (SR) machinery for effective material handling.

The effective scheduling of reclaimers constitutes crucial aspects of resource management in dry bulk terminals, directly influencing terminal throughput—a key performance indicator for these facilities. Despite its paramount significance, research on this subject is relatively underdeveloped, with a limited number of papers addressing the RSP. To the best of our knowledge, Hu and Yao [10] were the pioneers in addressing the SR scheduling problem at an iron ore terminal. They concentrated on minimizing the makespan for a given set of handling operations using a genetic algorithm (GA). Similarly, Angelelli et al. [1] conducted a study on bulk material reclamation in stockyards. They presented and analyzed multiple variants of an abstract scheduling problem for the reclaiming operations and demonstrated the $\mathcal{NP}$-hardness of these variants. Kalinowski et al. [11] extended the work presented by Angelelli et al. [1], relaxing the assumption that all stockpiles must be stacked at the beginning of the planning period. They further investigated the dynamic version of the problem, although they did not consider the setup times (i.e., traveling time) of reclaimers. Recently, Ünsal [16] delved into the RSP within a realistic world setting. He posited that the problem is a variant of the parallel machine scheduling problem and presented two versions—one with stacking operations and one without. The author developed an arc-time indexed Mixed Integer Programming (MIP) formulation to solve the problem.

The loading and unloading process at the yard-side presents risks to critical equipment like the SR, necessitating periodic preventive maintenance to prevent breakdowns and accidents [2]. This maintenance, including inspections, lubrication, and safety testing, is essential for terminal reliability but leads to downtime affecting stockpile handling. Benbrik et al. [3] pioneered the integration of preventive maintenance into reclaimer scheduling, developing mathematical formulations for the RSP. They explored two cases: one with two stockpads and one reclaimer, resulting in two novel formulations, and another with three stockpads and two reclaimers, leading to a unique model. Their formulations, solved using CPLEX, successfully handled small instances but struggled with medium ones. Therefore, this paper extends the scope of the second case of the configuration addressed in [3], with the primary objective of solving this problem with a real configuration of the stockyard involving multiple stockpads.

The main contributions of this paper are as follows:

- Investigation of a real configuration of a coal export terminal involving the minimization of the makespan.

- Development of a novel and innovative greedy constructive heuristic. Additionally, the design of an efficient GVNS metaheuristic for solving medium-sized instances of the problem within a reasonable computational time.
- Provision of empirical results from numerical experiments for reasonable computing times, considering both the literature and industrial practices.

The remaining sections of this paper are organized as follows. Section 2 presents the problem addressed in this study. Section 3 introduces a version of the MIP model previously developed for solving the problem. In Section 4, we describe the proposed greedy constructive heuristic procedures. Section 5 presents the GVNS approach employed in this research. Numerical experiments are conducted in Section 6. Finally, Section 7 concludes the paper by summarizing the findings and discussing future perspectives.

2 Problem Description

This paper addresses the scheduling problem related to the reclamation of stockpiles using a set of identical reclaimer machines, denoted as $\mathcal{M} = \{M_1, M_2 \ldots, M_m\}$. The operational layout consists of parallel stockpads on the yard-side of a dry bulk export terminal, represented by $\mathcal{P} = \{\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{m+1}\}$. Each reclaimer machine is mounted on a rail track between two adjacent stockpads. Let $\mathcal{P}_z = \{n_{(z-1)} + 1, \ldots, n_z\}$ represent the set of stockpiles in stockpad $\mathcal{P}_z$, with $n_0 = 0$ and n_z denoting the number of stockpiles in $\mathcal{P}_z$. The set $\mathcal{N} = \{J_1, J_2, \ldots, J_n\}$ encompasses all stockpiles across all stockpads, where $n = \sum_{z=1}^{m+1} n_z$. Each stockpile i, where $i \in [\![1, n]\!]$, possesses a length denoted by L_i. The time required to reclaim a stockpile, p_i, is determined as the ratio of its length to the reclamation speed s (i.e., $p_i = L_i/s$). Introducing sequence-dependent setup times, denoted as $t_{i,j}$, accounts for the travel time between two consecutive stockpiles. The setup time is the duration between completing the reclamation of $J_i \in \mathcal{N}$ and commencing the reclamation of the subsequent stockpile $J_j \in \mathcal{N}$. We assume $t_{0,j} = 0$, signifying no setup before processing the first reclaiming job. Additionally, the triangle property holds for setup times, ensuring $t_{i,l} + t_{l,j} \geq t_{i,j}$ for any three distinct jobs J_i, J_j, and J_l. Furthermore, strict adherence to the eligibility restrictions of the machines is enforced; each machine possesses the capability to pivot its boom, facilitating the processing of adjacent stockpiles along the rail track, while reclamation of stockpiles from other stockpads is not allowed. On a reclaimer machine M_k, stockpile reclamation occurs during the interval between consecutive preventive periodic maintenance activities, with the length denoted as T_k. Each maintenance activity has a duration of σ. Reclaiming tasks are prohibited during maintenance activities, and no breakdowns occur after maintenance. The problem is denoted as Reclaimer Scheduling Problem with Preventive Periodic Maintenance Activities (RSP-PPMA), with the objective of finding a feasible schedule to minimize the makespan. In this problem, we refer to each reclaimer as a machine, and each operation of reclaiming stockpile as a job.

Importantly, this paper expands upon the previous work conducted by Benbrik et al. [3]. The main notations used to describe the problem are listed in Table 1.

Figure 1 displays a graphical representation of a feasible solution to the addressed RSP-PPMA. In the figure, green rectangles represent T_k, indicating the duration between two consecutive maintenance activities on machine $M_k \in \mathcal{M}$. The maintenance activities are denoted by PM in yellow rectangles, with σ representing the duration of a maintenance activity. Jobs are scheduled within batches denoted as $\{B_1, B_2, \ldots, B_b, \ldots, B_n\}$, each having a duration of $T_k + \sigma$ $\forall k \in [\![1, m]\!]$.

Table 1: Notations.

Sets and Indices	
m	Number of machines
n_z	Number of jobs in stockpad z
n	Number of jobs ($n = \sum_{z=1}^{m+1} n_z$)
$\mathcal{M}$	Set of machines ($M_1, M_2, \ldots, M_k, \ldots, M_m$)
$\mathcal{P}$	Set of stockpads ($\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_z, \ldots, \mathcal{P}_{m+1}$)
$\mathcal{P}_z$	Set of jobs in stockpad z ($\mathcal{P}_z = \{n_{(z-1)} + 1, \ldots, n_z\}$ with $n_0 = 0$)
$\mathcal{N}$	Set of jobs in all stockpads ($\mathcal{N} = \{J_1, J_2, \ldots, J_n\}$)
Parameters	
A	Large positive integer
p_i	Processing time of job $J_i \in \mathcal{N}$
$t_{i,j}$	Travel time of machine between stockpile J_i and stockpile J_j, i.e., setup time
T	Time interval between two consecutive maintenance activities
σ	Duration of a maintenance activity
Decision variables	
$C_{i,k}$	The completion time of job $J_i \in \mathcal{N}$ on machine $M_k \in \mathcal{M}$
C_{max}	Maximum completion time (makespan)

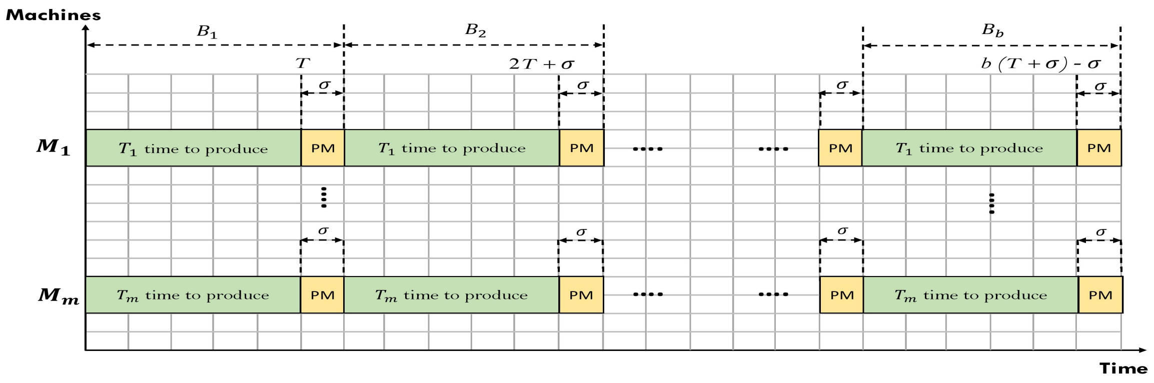

Fig. 1: Graphical representation of a feasible solution.

3 Mathematical Formulation

This section introduces a complex version of the MIP formulation previously proposed by Benbrik et al. [3], tailored for addressing the RSP-PPMA. The model is specialized for scenarios involving three parallel stockpads and two reclaimer machines. The scheduling strategy involves grouping jobs into batches denoted as $\mathcal{B} = \{B_1, B_2, \ldots, B_b, \ldots, B_n\}$ for the two reclaimer machines, with the overarching objective of minimizing the makespan. Each batch on machine $M_k \in \mathcal{M}$ has a capacity constraint denoted as T_k, and the time allocated for each maintenance activity is represented by σ. Throughout this formulation, the assumption $T_k = T, \forall k \in [\![1, m]\!]$ is adopted, where the notation

$[\![X,Y]\!]$ is used to indicate the interval of all integers between X and Y included. This problem can be seen, as a variant of the parallel machine scheduling problem [16].

The binary decision variables in this formulation are denoted as follows:

$$x_{i,j} = \begin{cases} 1 & \text{if job } J_j \text{ follows job } J_i \text{ in the sequence} \\ 0 & \text{otherwise} \end{cases}$$

$$y_{i,k} = \begin{cases} 1 & \text{if job } J_i \text{ is processed on machine } M_k \in \mathcal{M} \\ 0 & \text{otherwise} \end{cases}$$

$$\alpha^k_{i,b} = \begin{cases} 1 & \text{if job } J_i \text{ is in batch } b \in \mathcal{B} \text{ on machine } M_k \in \mathcal{M} \\ 0 & \text{otherwise} \end{cases}$$

$$
\begin{array}{lll}
(MIP)\min C_{max} & & (1)\\
\text{s.t. } C_{max} \geq C_{i,k} & \forall i \in [\![1,n]\!], \forall k \in [\![1,m]\!] & (2)\\
y_{i,1} = 1 & \forall i \in \mathcal{P}_1 & (3)\\
y_{i,2} = 1 & \forall i \in \mathcal{P}_3 & (4)\\
y_{i,1} + y_{i,2} = 1 & \forall i \in \mathcal{P}_2 & (5)\\
C_{i,k} \geq p_i y_{i,k} & \forall i \in [\![1,n]\!], \forall k \in [\![1,m]\!] & (6)\\
x_{i,j} + x_{j,i} \geq y_{i,k} + y_{j,k} - 1 & \forall i,j \in [\![1,n]\!], i \neq j, \forall k \in [\![1,m]\!] & (7)\\
C_{i,k} \leq A y_{i,k} & \forall i \in [\![1,n]\!], \forall k \in [\![1,m]\!] & (8)\\
C_{j,k} + A(3 - x_{i,j} - y_{i,k} - y_{j,k}) \geq C_{i,k} + t_{i,j} + p_j & \forall i,j \in [\![1,n]\!], i < j, \forall k \in [\![1,m]\!] & (9)\\
C_{j,k} + t_{j,i} + p_i \leq C_{i,k} + A(2 - y_{i,k} - y_{j,k} + x_{i,j}) & \forall i,j \in [\![1,n]\!], i < j, \forall k \in [\![1,m]\!] & (10)\\
\sum_{b=1}^{n} \alpha^k_{i,b} = y_{i,k} & \forall i \in [\![1,n]\!], \forall k \in [\![1,m]\!] & (11)\\
C_{i,k} \geq (b-1)\alpha^k_{i,b}(T_k + \sigma) + t_{n+1,i}\alpha^k_{i,b} + p_i y_{i,k} & \begin{array}{l}\forall i \in [\![1,n]\!], \forall b \in [\![2,n]\!],\\ \forall k \in [\![1,m]\!]\end{array} & (12)\\
C_{i,k} \leq b\alpha^k_{i,b} T_k + (b-1)\sigma - t_{i,n+1} + A(1 - \alpha^k_{i,b}) & \forall i,b \in [\![1,n]\!], \forall k \in [\![1,m]\!] & (13)\\
C_{i,k} \geq 0 & \forall i \in [\![1,n]\!], \forall k \in [\![1,m]\!] & (14)\\
x_{i,j} \in \{0,1\} & \forall i,j \in [\![1,n]\!] & (15)\\
y_{i,k} \in \{0,1\} & \forall i \in [\![1,n]\!], \forall k \in [\![1,m]\!] & (16)\\
\alpha^k_{i,b} \in \{0,1\} & \forall i,b \in [\![1,n]\!], \forall k \in [\![1,m]\!] & (17)
\end{array}
$$

The objective function (1) aims to minimize the makespan of each reclaimer machine. Constraint set (2) ensures that the makespan of an optimal schedule is not less than the completion time of all jobs that have been executed on each machine. The sets of constraints (3)-(5) impose eligibility restrictions on the reclaimer machines. Constraints (3) and (4) specify that all jobs in the stockpad $\mathcal{P}_1$ and the last stockpad $\mathcal{P}_3$ are processed on machines M_1 and M_2, respectively. Constraint set (5) guarantees that each job J_i in stockpad $\mathcal{P}_2$ is assigned to exactly one reclaimer machine. Constraint set (6) calculates the completion time of job J_i on each machine. Constraint set (7) ensures that no two jobs J_i and J_j can overlap in time. Constraints (8)-(10) specify that a job can be processed only if the machines are available. Constraints (9) and (10) indicate that no two jobs J_i and J_j scheduled on the same reclaimer machine (i.e., $y_{i,k} = y_{j,k} = 1$) can overlap in time. Constraint set (11) ensures that each job J_i is assigned to exactly one batch on a corresponding machine M_k. Constraints (12) and (13) guarantee that every job J_i processed by machine M_k must be executed within batch B_b of this machine. Constraint (12) ensures that in batch B_b of machine M_k, with $b \in [\![2, n]\!]$, the scheduling of any job J_i in this batch is performed after the end of preventive maintenance and the setup time $t_{(n+1),i}$ spent after maintenance. Additionally, constraint (13) requires that each job J_i processed by machine M_k should be finished before the starting time of preventive maintenance activity and its related setup time $t_{i,(n+1)}$. Constraint set (14) defines the completion time of job J_i on machine M_k as a positive continuous variable. Constraints (15), (16), and (17) define the variables $x_{i,j}$, $y_{i,k}$, and $\alpha^k_{i,b}$ as binaries.

4 Greedy Constructive Heuristic Procedure

In this section, we present a novel constructive heuristic algorithm tailored for solving the RSP-PPMA (Algorithm 1). Constructive methods, widely used for generating feasible solutions from scratch in optimization problems, serve as the foundation for our innovative approach. The algorithm begins with an initialization phase, assigning jobs from the first and last stockpads, $\mathcal{P}_1$ and $\mathcal{P}_{m+1}$, to machines M_1 and M_m, respectively. The core of our constructive heuristic procedure (Phase 2) is an iterative process that traverses each intermediate stockpad $\mathcal{P}_z$, where $z \in [\![2, m]\!]$. Within this phase, a job J_i is selected from a specific position (pos_i) within stockpad $\mathcal{P}_z$. The selected job is simultaneously allocated to both the current machine M_z and the preceding machine M_{z-1}, adhering to eligibility constraints. Allocation decisions depend on the comparison of the last completion times for jobs assigned to M_z and M_{z-1}. Based on this evaluation, the job either remains at M_z or is transferred to M_{z-1}, striving for a locally optimal job-machine allocation. This iteration continues until all positions within the stockpad have been considered. Moving to Phase 3, the algorithm generates prioritized job sequences π_k for each machine M_k, where $k \in [\![1, m]\!]$. This prioritization is achieved by sorting jobs using the Longest Processing Time (LPT) rule. The resulting prioritized sequences π_k are then concatenated, forming an ordered list that guides the creation of the final schedule π as a feasible solution to the RSP.

The preventive maintenance assignment scheduler algorithm (PMAS) (Algorithm 2) is devised to incorporate Preventive Maintenance (PM) considerations into the initial job sequence π generated by the constructive heuristic procedure. In Phase 4 of the Algorithm 1, PMAS is applied to seamlessly integrate PM and calculate the makespan with the final sequence π_{PM}. The input includes the initial sequence π and the batch duration representing the time between two consecutive preventive maintenance activities (i.e., T). The algorithm iterates through each machine M_k, applying the algorithm 1 to generate individual sequences π_k. Subsequently, the completion times for each job in π_k are calculated, and jobs exceeding the batch duration are moved to the next batch. The algorithm maintains track of completion times, and the last completion time of each machine is reported (i.e., makespan). The final sequence π_{PM} is generated by incorporating the PM strategy, ensuring that jobs are appropriately scheduled within batches. This is achieved by updating π_{PM} through the union operation with the sequences $\pi_{k,\mathrm{PM}}$ for each machine M_k. The output includes π_{PM} and the makespan C_{max}, providing a feasible solution to the RSP that seamlessly integrates both job sequencing and PM considerations.

Algorithm 1 Constructive heuristic algorithm (Constructive)

```
    Input: $z, n_z, \mathcal{P}_z, m$
 1: Phase 1: Initialization
 2: if $z = 1$ then
 3:     Assign($M_1$, $\mathcal{P}_1$)
 4: end if
 5: if $z = m + 1$ then
 6:     Assign($M_m$, $\mathcal{P}_{(m+1)}$)
 7: end if
 8: for $k = 2$ to $m - 1$ do
 9:     $\pi_k \leftarrow \emptyset$
10: end for
11: Phase 2: Constructive Heuristic Procedure
12: $z \leftarrow 2$ , $pos \leftarrow 1$
13: while $\left(pos \leq n_z\right)$ do                                    ▷ $z \in [\![2, m]\!]$
14:     for each $z \in [\![2, m]\!]$ do
15:         Choose the job $J_i$ situated at the position $pos$ within the stockpad $\mathcal{P}_z$.
16:         Assign( $M_{z-1}$, $J_i$)
17:         Assign( $M_z$, $J_i$)
18:         if $\Big(C_{max}(M_z) < C_{max}(M_{z-1})\Big)$ then
19:             Assign($M_z$, $J_i$ )
20:             Remove($M_{z-1}$, $J_i$)                                 ▷ Local search procedure
21:         else
22:             Assign($M_{z-1}$, $J_i$ )
23:             Remove($M_z$, $J_i$)
24:         end if
25:     end for
26:     $pos \leftarrow pos + 1$
27: end while
28: Phase 3: Schedule jobs using the $LPT$ rule
29: $\pi \leftarrow \emptyset$
30: for each $k \in [\![1, m]\!]$ do
31:     $\pi_k \leftarrow$ Sort($M_k$, $LPT$)
32:     $\pi \leftarrow \pi \cup \pi_k$
33: end for
34: Phase 4: Integration of preventive maintenance and makespan calculation using PMAS
35: $(\pi_{\mathrm{PM}}, C_{max}) \leftarrow$ PMAS($\pi$, $T$)
    Output: $\pi_{\mathrm{PM}}, C_{max}$
```

Algorithm 2 Preventive maintenance assignment scheduler algorithm (PMAS)

```
    Input: $\pi$, $T$                                ▷ Initial sequence and batch duration
 1: $\pi_{\mathrm{PM}} \leftarrow \emptyset$                                ▷ Initialize final sequence
 2: for $k = 1$ to $m$ do
 3:     $\pi_k \leftarrow$ Constructive($\mathcal{N}, \mathcal{P}, \mathcal{M}$)        ▷ Individual sequences for each $M_k$
 4:     $C_{max} \leftarrow 0$
 5:     $batch_k \leftarrow 1$
 6:     for each $J_i \in \pi_k$ do
 7:         Calculate the completion time $C_{i,k}$ of job $J_i$
 8:         if $C_{i,k} > batch_k \times T_k$ then
 9:             Move the current job $J_i$ to the next batch
10:             Update the completion time $C_{i,k}$
11:             $batch_k \leftarrow batch_k + 1$
12:         end if
13:     end for
14:     Report the last completion time $C_{i,k}$ of machine $M_k$
15:     if $C_{i,k} > C_{max}$ then
16:         Update $C_{max}$ with the last completion time $C_{i,k}$
17:     end if
18:     $\pi_{\mathrm{PM}} \leftarrow \pi_{\mathrm{PM}} \cup \pi_{k,\mathrm{PM}}$                      ▷ Sequence with PM
19: end for
    Output:
20: Final sequence $\pi_{\mathrm{PM}}$ and Makespan $C_{max}$
```

5 General Variable Neighborhood Search

Advanced optimization techniques, such as Simulated Annealing (SA), Tabu Search (TS), and Variable Neighborhood Search (VNS), constitute sophisticated metaheuristic approaches. These methods are strategically crafted to navigate beyond local optima within the search space while steering other heuristics. The VNS is a metaheuristic method initially introduced by Mladenović and Hansen [12]. It incorporates a local search procedure and dynamically adjusts neighborhood structures throughout the solution process. Over time, VNS has transcended its status as just a metaheuristic, transforming into a general framework for heuristic development. Numerous variants have emerged from the original schema [9], rendering this methodology a robust and potent tool in the optimization context. The VNS method has been widely employed in a diverse set of optimization problems (see Sifaleras and Konstantaras [14]; Benmansour and Sifaleras [5]; Benmansour et al.[4]; Benmansour et al.[6]; Elidrissi et al. [8]).

In this paper, we utilize the primary framework of the methodology known as GVNS. However, as mentioned earlier, various well-known variants exist alongside the general design. Some of the classical and widely adopted ones include Basic VNS (BVNS), Reduced VNS (RVNS), and Variable Neighborhood Descent (VND). For a recent survey on VNS, we direct the reader to the work by Hansen et al. [9]. The GVNS proposed in this paper for the RSP-PPMA generally incorporates advanced local search procedures, such as pipe VND, sequential VND, cyclic VND, and others, in the algorithm's improvement phase (cf. Hansen et al., [9]).

5.1 Neighborhood Structures

The efficiency of the GVNS metaheuristic relies on defining suitable neighborhood structures. In this paper, we categorize permutation-based neighborhoods into two types: intra-machine and inter-machine. Intra-machine neighborhoods concentrate on single-reclaimer machine impacts, comprising the Intra-machine exchange (N_1), Intra-machine insert (N_2), and Intra-machine reverse (N_3) neighborhoods. The Intra-machine exchange involves swapping positions of two jobs $\pi^{J_i}_{k,\mathrm{PM}}$ and $\pi^{J_j}_{k,\mathrm{PM}}$ on a single machine $M_k \in \mathcal{M}$. Intra-machine insert moves a job within the same machine, which involves taking a job from its current position and inserting it into another position within the same machine $M_k \in \mathcal{M}$. Intra-machine reverse chooses two jobs $\pi^{J_i}_{k,\mathrm{PM}}$ and $\pi^{J_j}_{k,\mathrm{PM}}$ and reverses the order of the jobs between them on a machine. Inter-machine neighborhoods, affecting two machines, consist of the Inter-machine exchange (N_4) and Inter-machine insert (N_5). The Inter-machine exchange chooses jobs $\pi^{J_i}_{k-1,\mathrm{PM}}$ and $\pi^{J_j}_{k+1,\mathrm{PM}}$, where $J_i, J_j \in P_k \cup P_{k+1}$, and exchanges them, while Inter-machine insert removes a job from one machine and inserts it into another, adhering to eligibility restrictions.

5.2 Variable Neighborhood Descent

We propose incorporating the neighborhood structures detailed in Section 5.1 collectively within the context of a VND heuristic to locally refine a given solution π_{PM}. The general framework of the VND is delineated in Algorithm 3. It initiates with an initial solution π_{PM} derived from a constructive heuristic, extensively discussed in Section 4. Subsequently, the algorithm persistently endeavors to construct an improved solution from the current state π_{PM} by exploring its neighborhood $N_l(\pi_{\mathrm{PM}})$. Thus, the effectiveness of the VND variants proposed in this study relies on the sequence of five neighborhood structures (N_1, N_2, N_3, N_4, N_5) and the chosen search strategy (*first* or *best* improvement), known as the neighborhood change step. The approach chosen for transitioning between neighborhoods is the pipe strategy. The steps of this strategy are outlined in Algorithm 4. Following preliminary tests, we opted for the best improvement in the search strategy, and in our proposed GVNS, the selected neighborhood order is $N_4(\pi_{\text{PM}}), N_5(\pi_{\text{PM}}), N_3(\pi_{\text{PM}}), N_1(\pi_{\text{PM}}), N_2(\pi_{\text{PM}})$.

Algorithm 3 Variable Neighborhood Descent VND

```
   Data: $\pi_{\text{PM}}, l_{max}$
   Result: $\pi_{\text{PM}}$
1: while There is no improvement do
2:     $l \leftarrow 1$
3:     while $l \leq l_{max}$ do
4:         $\pi'_{\text{PM}} \leftarrow$ Local Search($\pi_{\text{PM}}, N_l$)
5:         ChangeNeighborhood-Pipe($\pi_{\text{PM}}, \pi'_{\text{PM}}, l$)
6:     end while
7: end while
8: return $\pi_{\text{PM}}$
```

Algorithm 4 ChangeNeighborhood-Pipe ($\pi_{\text{PM}}, \pi'_{\text{PM}}, l$)

```
1: if $\left(C_{max}(\pi'_{\text{PM}}) < C_{max}(\pi_{\text{PM}})\right)$ then
2:     $\pi_{\text{PM}} \leftarrow \pi'_{\text{PM}}$
3: else
4:     $l \leftarrow l + 1$
5: end if
6: return $\pi_{\text{PM}}$
```

5.3 Shake Strategy

The shaking phase, pivotal for escaping local optima during convergence, is integral to the algorithm's effectiveness (Mladenović and Hansen, [12]). This phase systematically generates k random jumps from the current solution π_{PM}. Based on our experiments, we adopted a diversification approach involving the random selection from a set of predefined neighborhood structures, namely N_3, N_1, and N_2. Subsequently, we apply the selected structure k times, where $1 \leq k \leq k_{max}$. It's worth noting that introducing additional neighborhood structures in the shaking method has been observed to detrimentally impact result quality. The procedures of the shaking phase are outlined in Algorithm 5.

Algorithm 5 Shaking

```
    Data: $\pi_{\text{PM}}, k$
1:  $p \leftarrow$ randomInteger(1, 3)
2:  for $j = 1$ to $k$ do
3:      if ($p = 1$) then
4:          Generate a random $\pi'_{\text{PM}} \in N_3(\pi_{\text{PM}})$
5:      end if
6:      if ($p = 2$) then
7:          Generate a random $\pi'_{\text{PM}} \in N_1(\pi_{\text{PM}})$
8:      end if
9:      if ($p = 3$) then
10:         Generate a random $\pi'_{\text{PM}} \in N_2(\pi_{\text{PM}})$
11:     end if
12: end for
13:     return $\pi'_{\text{PM}}$
```

5.4 GVNS for the RSP-PPMA

In this section, we present the overall pseudocode of GVNS as it is implemented to solve the RSP-PPMA, which is presented in Algorithm 6. This scheme has three input parameters: the initial solution (π_{PM}), the maximum perturbation level (k_{max}), and the maximum computing time (T_{max}). The parameters (T_{max}) and (k_{max}), determined after preliminary experimentation, will be provided in Section 6.2. The diversifcation and intensifcation ability of GVNS relies on the shaking phase and VND, respectively. Shaking step of GVNS consists of three neighbohood structures N_3, N_1, and N_2. In the VND step, the five proposed neighborhood structures are used. The stopping criterion is a CPU time limit T_{max}. It is worth noting that the construction of the initial solution lies outside the GVNS framework. Typically, this initial solution can be generated randomly, following the common practice in the VNS community. However, a more sophisticated constructive procedure, as supported by literature (see Sánchez-Oro et al. [13]), can significantly enhance the quality of the best solution. A well-designed starting point is often more promising than a simple random solution. In Section 4, we detailed the constructive procedures proposed for the RSP-PPMA, which furnish the initial solutions for the GVNS.

Algorithm 6 General variable neighborhood search GVNS

```
    Data: $\pi_{\mathrm{PM}}, k_{max}, T_{max}$
    Result: $\pi_{\mathrm{PM}}$
1:  while $CPU \leq T_{max}$ do
2:      $k \leftarrow 1$
3:      while $k \leq k_{max}$ do
4:          $\pi'_{\mathrm{PM}} \leftarrow$ Shaking($\pi_{\mathrm{PM}}, k$)
5:          $\pi''_{\mathrm{PM}} \leftarrow$ VND($\pi'_{\mathrm{PM}}$)
6:          if $\left(C_{max}(\pi''_{\mathrm{PM}}) < C_{max}(\pi_{\mathrm{PM}})\right)$ then
7:              $\pi_{\mathrm{PM}} \leftarrow \pi''_{\mathrm{PM}}$
8:              $k \leftarrow 1$
9:          else
10:             $k \leftarrow k + 1$
11:         end if
12:     end while
13: end while
14: return $\pi_{\mathrm{PM}}$
```

6 Computational Results

To evaluate and showcase the effectiveness of the proposed algorithm on problem instances of different sizes, extensive computational experiments were carried out. The MIP model for the RSP-PPMA, as employed in a prior study by Benbrik et al. [3], was implemented using the CPLEX 22.1 MIP solver with default configurations. Simultaneously, all other algorithms were coded in C++.

During the experiments, a personal computer with an Intel(R) Core(TM) i7-7700HQ CPU operating at 2.8 GHz and 8 GB of RAM was used. The MIP formulations are analyzed based on the following metrics: the objective value (*Opt*) of the test instances solved to optimality within 1800 s, the time required for solving these optimally solved instances (*CPU*) in seconds (s), the objective function value of the instances unsolved within 1800 s (instances with feasible solutions), denoted as *Best Integer*, and the optimality gap for the test instances which could not be solved within 1800 s, denoted as *Gap(%)*. Importantly, optimal solutions were only achievable for small instances with $n = 15$ jobs and $m = 2$ stockpads due to the $\mathcal{NP}$-hard nature of the RSP-PPMA. As a result, the constructive heuristics and GVNS versions were adapted for medium instances. It is crucial to emphasize that, to mitigate the influence of stochastic variations, 10 runtime executions of the GVNS were performed for each problem instance. Consequently, the Best (Best.), Maximum (Max.), and Average (Avg.) objective function values were determined from these 10 runs and reported. Additionally, the average computation times (CPU) were calculated based on the 10 runs, with each run's computation time corresponding to the moment when the best solution encountered during that specific run was identified.

6.1 Benchmark Instances

The characteristics of the test instances derived from the scheduling environment of the NCIG terminal in Australia. These instances possess the following characteristics:

- The processing times of reclaiming stockpiles p_i, the setup times $t_{i,j}$, the time interval T, and the duration of a maintenance activity σ are generated following the approach proposed by Benbrik et al. [3]. Specifically, $p_i \sim U(60, 140)$, $t_{i,j} = \beta \times \min(p_i, p_j)$, where $\beta \sim U(0.05, 0.15)$, $\sigma \sim U(20, 90)$, and $T = \max\left(\max_{i \in \mathcal{N}} p_i, 4 \times \sum_{i=1}^{n} p_i/n\right)$.
- The number of jobs in the stockpads is categorized into two sets of test problem instances. For small problem instances, the number of jobs n is chosen from $\{10, 15\}$, while the number of machines m is fixed at 2 (i.e., $\mathcal{P} = \mathcal{P}_3$). In the case of medium problem instances, the number of jobs n varies from 30 to 100. Specifically, n takes values from $\{30, 40, 50, 60\}$ when m equals 3 (i.e, $\mathcal{P} = \mathcal{P}_4$). It is essential to note that, for each combination of values (n, m), a total of 10 distinct problem instances were generated for both the small and medium-scale cases.
- In total, 60 unique problem instances were generated for every combination of values $(n, T, \sigma, \mathcal{P})$. These instances were evenly distributed across the two problem categories, with 20 instances designated for the small problem set, and 40 instances assigned to the medium problem set.

6.2 Tuning Parameters

A series of experiments were conducted to identify optimal parameter values for the GVNS algorithm. The algorithm relies on two key tuning parameters: k_{max} denotes the maximum perturbation level, and T_{max} represents the maximum time allotted to the GVNS. After preliminary experimentation, a thoughtful selection was made for the parameter configuration. Specifically, k_{max} is set to 20, chosen for its ability to strike a balance between solution quality and computational time (CPU). For small-sized instances $\big(n \in \{10, 15\}, T, \sigma, \mathcal{P} = \mathcal{P}_3\big)$, T_{max} is set to the computation time required to find an optimal solution using the CPLEX solver. For medium-sized instances, T_{max} is determined by the formula $T_{max} = \big(n \times m\big)/5$, indicating a polynomial increase in time with the growth of jobs and machines.

6.3 An Analysis of the Effectiveness of the Proposed Constructive Heuristic for Small Problems

In this section, we conduct an analysis of the performance of the developed greedy constructive heuristics for small-scale instances. For this heuristic, we calculate the percentage deviation for every problem instance from its optimum using the following formula:

$$Dev = 100 \times \left(\frac{C^{H}_{max} - opt}{opt} \right)$$

Here, C_{max}^{H} represents the makespan achieved by the greedy constructive heuristic, and *opt* denotes the optimum value obtained through the MIP formulation proposed in Benbrik et al. [3].

As anticipated, the computational experiment results presented in Table 2 demonstrate that the *Dev* values of the proposed greedy constructive heuristic, aimed at minimizing the makespan for small problem instances, generally fall within the range of 1.78 % to 13.29 % for the combination of values ($n = 10, T, \sigma, \mathcal{P} = \mathcal{P}_3$), and 7 % to 22.72 % for the combination of values ($n = 15, T, \sigma, \mathcal{P} = \mathcal{P}_3$).

In addition to evaluating the efficacy of heuristic based on their performance in addressing medium problem instances, the obtained outcomes for small-scale problems are deemed reasonably satisfactory and promising. This heuristic, furthermore, can be regarded as a robust initial solution for the GVNS metaheuristics.

Table 2: Evaluation of the heuristic algorithm for the RSP-PPMA in small scale instances for $\mathcal{P} = \mathcal{P}_3$.

Problem Instance				MIP				Constructive Heuristic		
				Objective value						
$\mathcal{P}$	n	T	σ	*Opt*	*Best integer*	*Gap* (%)	*CPU* (sec)	C_{max}^{H}	*CPU* (sec)	*Dev* (%)
	10	370.00	22	**550.70**	-	0.0	1.36	583.70	0.001	5.99
	10	378.00	54	**627.90**	-	0.0	2.87	639.90	0.001	1.91
	10	366.61	31	**557.45**	-	0.0	7.07	602.45	0.001	8.07
	10	378.61	47	**612.40**	-	0.0	1.40	639.60	0.001	4.44
	10	295.20	60	**525.10**	-	0.0	0.69	594.90	0.001	13.29
	10	355.40	87	**672.90**	-	0.0	0.83	684.90	0.001	1.78
	10	323.80	65	**580.50**	-	0.0	2.02	627.50	0.001	8.10
	10	339.80	81	**606.80**	-	0.0	1.01	664.00	0.001	9.43
	10	424.40	22	**650.10**	-	0.0	1.41	684.30	0.001	5.26
$\mathcal{P}_3$	10	404.00	60	**582.45**	-	0.0	1.27	621.90	0.001	6.77
	15	321.00	87	**813.00**	-	0.0	3.51	952.85	0.001	17.20
	15	395.86	31	**837.11**	-	0.0	87.20	1003.21	0.001	19.84
	15	397.43	22	**920.51**	-	0.0	145.25	984.91	0.001	7.00
	15	349.71	60	**893.03**	-	0.0	824.38	1025.83	0.001	14.87
	15	416.43	31	**874.23**	-	0.0	54.61	1052.41	0.001	20.38
	15	394.00	58	**880.20**	-	0.0	120.50	1063.85	0.001	20.86
	15	365.29	65	**859.58**	-	0.0	1050.22	946.82	0.001	10.15
	15	393.43	22	**804.98**	-	0.0	56.23	986.11	0.001	22.50
	15	396.29	58	**884.69**	-	0.0	28.75	1085.67	0.001	22.72
	15	402.14	47	**870.94**	-	0.0	370.20	1039.74	0.001	19.38
		Avg.		730.23	-	0.0	138.04	824.22	0.001	11.99

6.4 Assessing the Efficiency and Impact of GVNS Metaheuristics for Small Problems

The performance evaluation of the GVNS algorithm applied to solving the RSP-PPMA in small-scale instances is presented comprehensively in Table 3.

To quantify the percentage deviation for every problem instance from its optimum, we compute the percentage deviation (*Dev*) using the formula:

$$Dev = 100 \times \left(\frac{C_{max}^{\text{GVNS}} - opt}{opt} \right)$$

Here, C_{max}^{GVNS} represents the Best makespan obtained by the modified algorithm—GVNS (i.e., GVNS with the constructive heuristic as the initial solution), while *opt* denotes the optimum value obtained through the MIP formulation proposed by Benbrik et al. [3]. The objective of this analysis is to gain insights into the efficacy of this algorithm in finding optimal or near-optimal solutions. A detailed examination of the objective function values reveals that the GVNS algorithm demonstrates competitive performance across the considered problem instances. These results underscore the superior effectiveness of GVNS in identifying optimal solutions for the RSP-PPMA in small-scale instances. Additionally, it is crucial to highlight the computational time aspect. While the MIP formulation provides optimal solutions, the associated CPU times are considerably longer compared to the GVNS approaches. For instance, in the case of the problem instance ($n = 15, T = 365.29, \sigma = 65, \mathcal{P} = \mathcal{P}_3$), the MIP model took 1050.22 seconds to find the optimal solution, whereas GVNS identified the same solution in 3.67 seconds. The deviation column (*Dev*) also provides valuable insights into the optimality of the solutions. Notably, the last row of the table indicates the average performance across all problem instances, revealing consistently low deviations from the optimum solution, with an average deviation of 0.13 %. This indicates that the solutions produced by GVNS are highly reliable and close to optimality, further affirming its effectiveness in solving small-scale instances of the BWRS problem.

Table 3: Evaluation of the GVNS algorithm for the RSP-PPMA in small-scale instances for $\mathcal{P} = \mathcal{P}_3$.

Problem Instance				*MIP*				GVNS				
				Objective value								
$\mathcal{P}$	n	T	σ	*Opt*	*Best integer*	*Gap* (%)	*CPU* (sec)	Best.	Max.	Avg.	*CPU* (sec)	*Dev* (%)
	10	370.00	22	**550.70**	-	0.0	1.36	**550.70**	550.70	550.70	1.63	0.0
	10	378.00	54	**627.90**	-	0.0	2.87	**627.90**	627.90	627.90	0.67	0.0
	10	366.61	31	**557.45**	-	0.0	7.07	**557.45**	557.45	557.45	0.44	0.0
	10	378.61	47	**612.40**	-	0.0	1.40	**612.40**	612.40	612.40	0.14	0.0
	10	295.20	60	**525.10**	-	0.0	0.69	**525.10**	546.10	527.20	0.58	0.0
	10	355.40	87	**672.90**	-	0.0	0.83	**672.90**	674.90	673.30	0.15	0.0
	10	323.80	65	**580.50**	-	0.0	2.02	**580.50**	590.50	585.04	1.14	0.0
	10	339.80	81	**606.80**	-	0.0	1.01	**606.80**	615.90	607.71	0.62	0.0
	10	424.40	22	**650.10**	-	0.0	1.41	**650.10**	665.10	651.60	0.31	0.0
$\mathcal{P}_3$	10	404.00	60	**582.45**	-	0.0	1.27	**582.45**	582.45	582.45	0.67	0.0
	15	321.00	87	**813.00**	-	0.0	3.51	825.20	926.41	903.86	9.54	1.48
	15	395.86	31	**837.11**	-	0.0	87.20	**837.11**	837.11	837.11	4.34	0.0
	15	397.43	22	**920.51**	-	0.0	145.25	**920.51**	923.96	921.20	3.29	0.0
	15	349.71	60	**893.03**	-	0.0	824.38	**893.03**	893.03	893.03	7.22	0.0
	15	416.43	31	**874.23**	-	0.0	54.61	**874.23**	874.23	874.23	5.65	0.0
	15	394.00	58	**880.20**	-	0.0	120.50	**880.20**	891.95	887.10	3.00	0.0
	15	365.29	65	**859.58**	-	0.0	1050.22	**859.59**	946.82	876.57	3.67	0.0
	15	393.43	22	**804.98**	-	0.0	56.23	**804.98**	817.03	806.89	5.31	0.0
	15	396.29	58	**884.69**	-	0.0	28.75	**884.69**	993.67	921.44	3.66	0.0
	15	402.14	47	**870.94**	-	0.0	370.20	877.24	880.99	879.11	1.98	0.72
	Avg.			730.23	-	0.0	138.04	731.15	750.43	738.81	2.70	0.13

6.5 Evaluating the Enhancement of Solutions from the Proposed Constructive Heuristic with Metaheuristics for Medium-Scale Problems

In our computational analysis, we integrated a greedy constructive heuristic to generate initial solutions for the GVNS algorithm. This integration aimed to explore potential improvements in the heuristic's effectiveness when utilized within the metaheuristic framework.

To quantify the enhancement achieved, we compute the percentage improvement (*Imp*) using the formula:

$$Imp = 100 \times \left(\frac{C^{H}_{max} - C^{\text{GVNS}}_{max}}{C^{\text{GVNS}}_{max}} \right)$$

Here, C_{max}^{H} represents the makespan obtained by the constructive heuristic, while C_{max}^{GVNS} denotes the Best makespan achieved by the modified algorithm—GVNS incorporating the constructive heuristic.

The comprehensive results are presented in Table 4. The table illustrates the average values obtained across all problem instances. Based on the observed variation in the average *Imp* values across different problem instances, ranging from 8.83 % to 14.40 %, it can be concluded that the modified algorithm—GVNS, effectively enhances the solutions derived from the constructive heuristic. Notably, the last row of the table demonstrates an average improvement (*Imp*) of 12.11 %, providing a benchmark for the effectiveness of GVNS in enhancing solutions derived from the constructive heuristic, especially for medium-sized instances. Additionally, it's noted that the average *Imp* values tend to decrease as the number of jobs increases. This trend suggests that the proposed heuristic remains quite competitive when compared to the metaheuristic approach.

Table 4: Average *Imp* values for enhancing solutions from the proposed constructive heuristic with metaheuristic

Problem Instance		Constructive Heuristic		GVNS				
$\mathcal{P}$	n	C_{max}^{H}	CPU(sec)	Best.	Max.	Avg.	CPU(sec)	Imp(%)
$\mathcal{P}_4$	30	1299.35	0.001	1146.44	1180.8	1160.38	14.09	13.38
	40	1828.26	0.001	1599.15	1649.38	1616.26	15.65	14.40
	50	2390.59	0.001	2138.20	2179.92	2153.14	22.50	11.84
	60	2689.82	0.001	2476.98	2536.86	2501.34	29.53	8.83
	Avg.	2052.01	0.001	1840.19	1886.74	1857.78	20.44	12.11

7 Conclusion

In this paper, we addressed the problem of scheduling stockpile reclamation considering the PPMA in bulk ports. The objective function considered was to find a feasible schedule which minimizes the latest completion time (i.e., makespan). Given the $\mathcal{NP}$-hard nature of the problem, a novel greedy constructive heuristic has been devised. This heuristic relies on iterative job allocation to machines and prioritized sequencing, all while considering the integration of PPMA into the scheduling process. Consequently, it ensures the appropriate scheduling of jobs within batches. The solutions generated through constructive procedures serve as excellent initial foundations for GVNS algorithm, tailored to handle medium-sized instances with up to 60 jobs and 3 machines (i.e., $\mathcal{P} = \mathcal{P}_4$). Computational experiments conducted on 60 new instances demonstrate that for small-sized instances, GVNS algorithm outperform the MIP formulation in terms of the computing time required to find an optimal solution. Furthermore, for medium-sized instances, GVNS consistently yields superior solutions compared to the proposed constructive heuristic.

Potential future research directions could involve several aspects. Firstly, there is a need to develop further metaheuristic algorithms to allow for a comprehensive comparison and evaluation of the proposed GVNS algorithm. Secondly, exploring the stochastic version of the problem would be relevant. Lastly, exploring the integration of advanced optimization techniques, such as multi-objective optimization methods, could offer valuable insights into addressing the complexities of the RSP-PPMA problem. Moreover, it would be beneficial to develop lower and upper bounds for this problem to facilitate a more thorough comparison of the effectiveness of the proposed algorithms.

References

1. Angelelli, E., Kalinowski, T., Kapoor, R., Savelsbergh, M.W.: A reclaimer scheduling problem arising in coal stockyard management. Journal of Scheduling **19**, 563–582 (2016)
2. Belov, G., Boland, N.L., Savelsbergh, M.W., Stuckey, P.J.: Logistics optimization for a coal supply chain. Journal of Heuristics **26**(2), 269–300 (2020)
3. Benbrik, O., Benmansour, R., Elidrissi, A.: Mathematical programming formulations for the reclaimer scheduling problem with sequence-dependent setup times and availability constraints. Procedia Computer Science **232**, 2959–2972 (2024). https://doi.org/10.1016/j.procs.2024.02.112
4. Benmansour, R., Braun, O., Hanafi, S., Mladenovic, N.: Using a variable neighborhood search to solve the single processor scheduling problem with time restrictions. In: International Conference on Variable Neighborhood Search. pp. 202–215. Springer (2018)
5. Benmansour, R., Sifaleras, A.: Scheduling in parallel machines with two servers: the restrictive case. In: Variable Neighborhood Search. ICVNS 2021. Lecture Notes in Computer Science. vol. 12559, pp. 71–82. Springer (2021)
6. Benmansour, R., Todosijević, R., Hanafi, S.: Variable neighborhood search for the single machine scheduling problem to minimize the total early work. Optimization Letters **17**(9), 2169–2184 (2023)
7. Boland, N.L., Savelsbergh, M.W.: Optimizing the hunter valley coal chain. In: Supply chain disruptions: theory and practice of managing risk, pp. 275–302. Springer (2011)
8. Elidrissi, A., Benmansour, R., Sifaleras, A.: General variable neighborhood search for the parallel machine scheduling problem with two common servers. Optimization Letters **17**(9), 2201–2231 (2023)
9. Hansen, P., Mladenović, N., Todosijević, R., Hanafi, S.: Variable neighborhood search: basics and variants. EURO Journal on Computational Optimization **5**(3), 423–454 (2017)
10. Hu, D., Yao, Z.: Stacker-reclaimer scheduling in a dry bulk terminal. International Journal of computer integrated manufacturing **25**(11), 1047–1058 (2012)
11. Kalinowski, T., Kapoor, R., Savelsbergh, M.W.: Scheduling reclaimers serving a stock pad at a coal terminal. Journal of Scheduling **20**, 85–101 (2017)
12. Mladenović, N., Hansen, P.: Variable neighborhood search. Computers & operations research **24**(11), 1097–1100 (1997)
13. Sánchez-Oro, J., Pantrigo, J.J., Duarte, A.: Combining intensification and diversification strategies in vns. an application to the vertex separation problem. Computers & Operations Research **52**, 209–219 (2014)
14. Sifaleras, A., Konstantaras, I.: A survey on variable neighborhood search methods for supply network inventory. In: Bychkov, I., Kalyagin, V.A., Pardalos, P.M., Prokopyev, O. (eds.) Network Algorithms, Data Mining, and Applications. NET 2018. Springer Proceedings in Mathematics & Statistics. vol. 315, pp. 71–82. Springer (2018)
15. UNCTAD: Review of maritime transport. United Nations Conference on Trade and Development (2022), http://www.unctad.org.
16. Ünsal, Ö.: Reclaimer scheduling in dry bulk terminals. IEEE Access **8**, 96294–96303 (2020)