

HEURISTIC METHODS TO SOLVE THE BERTH ALLOCATION PROBLEM IN IRON ORE EXPORT TERMINALS

¹ Ana Beatriz A. Scarpino, ¹ Welber W. F. dos Reis

² Renato E. N. Moraes, ¹ Maria Claudia S. Boeres

¹ Departamento de Informática (DI/UFES) – Vitória/ES

² Departamento de Tecnologia Industrial (DTI/UFES)– Vitória/ES

anascarpino@gmail.com, welber91@gmail.com,

renato.moraes@ufes.br, boeres@inf.ufes.br

ABSTRACT

Port terminals processing large cargo vessels play an important role in bulk material supply chain. This paper addresses the Berth Allocation Problem (BAP) which aims at assigning and scheduling incoming vessels to berthing positions along the quay of a bulk terminal. It is well known that even simpler variants of BAP are NP-hard, and thus, metaheuristic approaches are more convenient than exact methods since they provide high quality solutions in reasonable computational time. Therefore, we propose one greedy randomized constructive heuristic joined with two local search procedures to create two greedy randomized adaptive search procedures (GRASP) metaheuristics to solve the BAP in dry bulk terminals. The results reached by GRASP algorithms are compared with the optimal values given in the BAP literature. Computational experiments show that the proposed algorithm is able to obtain most of the optimal solutions expending low computational time.

KEYWORDS. Berth allocation problem, Port logistics, Metaheuristics, GRASP

Topics: L&T – Logistics and Transports; IND – PO in Industry; MH – Metaheuristics; CO – Combinatorial Optimization.

1. Introduction

Dry bulk terminals (DBTs) are transport hubs for carrying dry bulk materials which are shipping unpackaged (without containers) in large quantities. Typical dry bulk materials, such as coal and iron ore, are essential for producing electric energy and steel. Facing the increased global demand for dry bulk materials [Lodewijks et al., 2007], there has been significant growth in dry bulk transport over the last few decades.

When considering loading and unloading processes, DBTs can be roughly divided [Zhu et al., 2018] into two types: export and import. Import terminals mainly receive varieties of bulk cargoes in differing quantities via incoming ships and deliver the loads to various clients via rail networks after storage. Export terminals receive varieties of bulk cargoes via incoming vehicles, such as trains and trucks, and they are then transported by ship after stockpiling. This study is motivated by the export terminal of the Port of Tubarão located at Vitória in the Brazilian state of Espírito Santo. Port of Tubarão is one of the largest port specialized in the export of iron ore in the world. It is owned and operated by the Brazilian mining company Vale¹, recognized as one of the world's largest iron ore producer. Port of Tubarão is capable of receiving Valemax ships, the biggest mineral vessels in the world, each with capacity for 400,000 tonnes of ore.

The Port of Tubarão operation can be divided into three management areas: train arrival and unloading, stockyard management, and ship berthing. Because the quay throughput capacity is

¹<http://www.vale.com>

often used as a benchmark during the terminal design process [Zhu et al., 2018], in this paper we focus on the Berth Allocation Problem (BAP). The berth allocation problem aims to optimally assign and schedule ships to berth areas along a quay. As berth space is very limited at Port of Tubarão terminal, and thousands of tons of material must be handled daily, an effective berth allocation is critical to the efficient management of the iron ore flow.

BAP problems can be classified as being either discrete (DBAP) or continuous (CBAP), both of which are NP-hard [Cordeau et al., 2005]. The discrete case considers that the quay is divided into several specific berthing positions and only one ship can be serviced at a time in each berth, regardless of its size. In the continuous case, there is no division of the platform and thus the ships can moor at any position.

The BAP can be further classified into static or dynamic variants [Mauri et al., 2016]. The static case assumes all ships already in port for the service, while the dynamic case allows ships to arrive at any time (estimated in advance). Since the static case does not fully correspond to reality [Mauri et al., 2016], we consider the dynamic case of the discrete BAP where the quay is a finite set of berthing points. Berthing points or berth segments herein will be called berths.

The DBAP studied consists in, given a set of ships, a set of berths, the handling time of each ship at each berth, the beginning (ship arrival time) and the ending time of the ship time window, and the opening and closing hour of each berth (availability berth time window), minimize the sum for each ship of the service time (the difference between the completion time and the arrival time of the ship in port) such that every ship must be assigned to only one berth and at most one ship can be assigned to the same position on the same berth in the same time. The solution must also satisfy the available time of the ships (berthing time is greater than the arrival time and departure time is smaller than its bound in service time – ship time window) and berths (the first ship moors at the berth after its starting operation time and the completion time of a ship is smaller than the berth time windows upper bound – berth time window).

We treat the problem as a Vehicle Routing Problem with Time Windows and Multiple Garages (VRPTWMG) following the approach reported by Cordeau et al. [2005]. More recently, Mauri et al. [2016], Ribeiro et al. [2016] and Nishi et al. [2017] solve the DBAP proposed by Cordeau et al. [2005]. Mauri et al. [2016] propose an Adaptive Large Neighborhood Search (ALNS) heuristic based on the principle of destruction and recreation of solutions. Ribeiro et al. [2016] addressed a DBAP in an iron ore DBT. Nishi et al. [2017] also present matheuristic based on a dynamic programming. A more detailed literature review is presented in Section 2.

It is well known that even simpler variants of DBAP are NP-hard, and thus, metaheuristic approaches are more convenient than exact methods since they provide high quality solutions in reasonable computational time. Therefore, this paper presents a metaheuristic to solve the DBAP. We propose a GRASP heuristic implementing a new randomized constructive heuristic to generate greedy solutions to the problem. The proposed GRASP improves the greedy solutions applying two local searches to them. The GRASP solutions are then compared with the optimal values given in the DBAP literature.

The remainder of this paper is organized as follows. Section 2 gives a literature review. In Section 3 the proposed model is detailed. The GRASP heuristic is described in Section 4. Computational results are presented in Section 5, followed by conclusions in Section 6.

2. Related Work

Problems related to ship scheduling and berth allocation have been extensively studied by researchers. The tactical berth allocation problem refers to the problem of assigning a set of vessels to a given berthing layout within a given time horizon. There could be several objectives such as the minimization of the service times of vessels, the minimization of the port stay time, the minimization of the number of rejected vessels, the minimization of the deviation between actual and planned berthing schedules, etc.

There are also several spatial and temporal constraints involved in the BAP, which lead to a multitude of BAP formulations. The temporal attributes include the vessel arrival process, the start of service, the handling times of vessels, while the spatial attributes relate to the berth layout, the draft restrictions and others. Many researchers studied the berth allocation problem in container terminals (CTs), see for an extensive literature review [Bierwirth and Meisel, 2015].

In addition to the CT, the dry bulk cargo terminal is an important terminal type, of which the iron ore, coal, and grains terminals are the most common examples. The continuous process of loading the bulk material to the vessels and the fixed specialized equipment used are the differences between a DBT and a CT. Due to the high demand for energy and mineral resources many dry bulk terminals around the world are expanding and seriously increasing their capacity.

Various studies specifically addressed the BAP in DBTs. Umang et al. [2013] proposed several models and heuristics for hybrid BAP in bulk terminals with berths specializing in a range of cargo handling. In the hybrid BAP, as in the discrete case, the quay is divided in berths, but the big ships can occupy more than one position, thus allowing small ships to share his berth. Kordi et al. [2016] proposed a model and some heuristics for discrete and hybrid BAP implementing rules provided by the terminal operator. Some authors proposed models for the berth-allocation problem explicitly for bulk ports taking account of tidal constraints [Barros et al., 2011; Ernst et al., 2017].

In this paper, we consider the dynamic discrete BAP problem (DBAP) which could be applied in both CTs and DBTs. We treat the problem as a VRPTWMG following the approach reported by Cordeau et al. [2005]. They considered both discrete and continuous BAP to minimize the total completion times of the vessels. They presented mathematical formulations for the discrete case which is further improved upon by Mauri et al. [2008] using a column generation approach that delivers higher quality solutions in lesser computation time.

Also following the approach reported by Cordeau et al. [2005], Oliveira et al. [2012] proposed a Clustering Search (CS) hybrid metaheuristic to solve DBAP. The authors found the optimal solutions for all instances of DBAP (proved by Buhrkal et al. [2011]). The DBAP modeled as a VRPTWMG was recently studied by Mauri et al. [2016], Ribeiro et al. [2016] and Nishi et al. [2017]. Mauri et al. [2016] have proposed an adaptive large neighborhood search to solve the same problem. The authors have showed the effectiveness of the algorithm on a set of instances. Ribeiro et al. [2016] addressed a discrete BAP as those from Cordeau et al. [2005] in an iron ore DBT in which maintenance activities on the berths could also be scheduled. An adaptive large neighborhood search heuristic is applied to solve the instances and computational experiments are performed. Nishi et al. [2017] present a dynamic programming-based matheuristic to solve the DBAP. The proposed approach provides tight lower bounds by means of a Lagrangian decomposition approach based on the Lagrangian relaxation of the VRPTWMG accompanied by a dynamic programming algorithm for solving the subproblems. Ribeiro et al. [2016] and Nishi et al. [2017] also present a comprehensive review of past literature on the berth allocation problem.

In this research, we also propose a metaheuristic method to solve the DBAP. We use two different rearrangements of vessels [Oliveira et al., 2012] to define two neighborhoods of a given feasible solution and combine them with a new randomized constructive heuristic to create two greedy randomized adaptive search procedures (GRASP) metaheuristics to solve the DBAP.

3. Problem Definition

In this work, we study the discrete berth allocation problem proposed by Cordeau et al. [2005]. In this problem we are given a set of incoming vessels $\mathcal{V} = \{v_1, v_2, \dots, v_i\}$, $1 \leq i \leq |\mathcal{V}|$ and a set of berths $\mathcal{P} = \{B^1, B^2, \dots, B^k\}$, $1 \leq k \leq |\mathcal{P}|$. Each vessel $v_i \in \mathcal{V}$ must be assigned to a berth $B^k \in \mathcal{P}$, within its time window $[a_i, b_i]$, and within the berth time window $[o^k, c^k]$. Each berth can handle at most one vessel at a time.

For each vessel $v_i \in \mathcal{V}$, its handling time t_i^k depends on the berth $B^k \in \mathcal{P}$ where it is allocated. That is, the handling time of a given vessel differs from one berth to another. Moreover, some vessels may have forbidden berths in order to include water-depth or maintenance constraints.

The handling time is also given as a problem input. Finally, let $B^{k(i)}$ be the berth where ship v_i is currently allocated, we have $T_i^{k(i)}, k(i) \in \mathcal{P}$ and $v_i \in \mathcal{V}$, as the time when the ship v_i moored in the berth $B^{k(i)}$. The set of berthing times $T_i^{k(i)}, \forall i \in \mathcal{V}$, represents a BAP solution. A comprehensive description and mathematical formulations of the DBAP are provided by [Cordeau et al., 2005; Oliveira et al., 2012].

In this context, the service time of a vessel is defined as the time elapsed between its arrival to the port and the completion of its transshipment operations, $T_i^{k(i)} - a_i + t_i^{k(i)}$ (see Figure 1). Thus, the objective of the DBAP is to minimize the total service time $f_{DBAP} = \sum_{i \in \mathcal{V}} (T_i^{k(i)} - a_i + t_i^{k(i)})$ of the incoming vessels. The algorithm outputs a BAP solution $S = \{T_1^{k(1)}, T_2^{k(2)}, \dots, T_{|\mathcal{V}|}^{k(|\mathcal{V}|)}\}$ such that for each ship $v_i \in \mathcal{V}$, we know its allocated berth $B^{k(i)} \in \mathcal{P}$ and its berthing time $T_i^{k(i)}$.

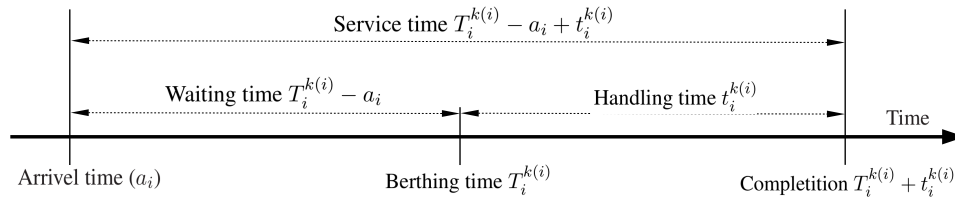


Figure 1: Representation of time variables.

We report an example of the berth allocation problem. Figure 2 shows a feasible solution for the DBAP, for which the handling times (t_i^k), berths time window ($[o^k, c^k]$) and vessels time window ($[a_i, b_i]$) are summarized in Table 1. In the dynamic case, the vessels can arrive at any time along the planning horizon. Note that idle times may appear in the berth's time window. Berths 1 and 2 are opened in times 4 and 3, and closed in times 20 and 18, respectively. Vessels v_2 and v_4 cannot moor in berth 2.

Vessels time window						Berths time window			Handling time t_i^k					
i	1	2	3	4	5	k	1	2	i	1	2	3	4	5
a_i	5	6	3	4	11	o^k	4	3	t_i^1	1	6	4	4	6
b_i	16	18	12	22	20	c^k	20	18	t_i^2	2	—	1	—	5

Table 1: Vessels time window, berths time window and handling time.

In the DBAP solution presented in Figure 2, vessels v_4, v_1 and v_2 are assigned to berth 1, while vessels v_3 and v_5 are assigned to berth 2. Vessel v_4 , which arrives at time 4, is assigned to berth 1 at this time and it stays for 4 time units as indicated in Table 1. The next vessel assigned to berth 1 is v_1 , that moors at time 8 and stays 1 time unit. Vessel v_2 is then assigned to berth 1 after vessel v_1 has finished. Note that, since vessel v_2 arrives at time 6 and is assigned to berth 1 at time 9, it has to wait for 3 time units. Then, it requires six more time units to perform the loading and unloading operations. Therefore, the total time vessel v_2 stays at the port is equal to 9 time units, which are counted in the objective function value. The same analysis can be considered to determine the stays of vessels v_3 and v_5 at the port.

The BAP solution can be represented as $S = \{T_1^1 = 8, T_2^1 = 9, T_3^2 = 3, T_4^1 = 4, T_5^2 = 11\}$. In this example, the vessels v_1 to v_5 stay at the port for 4, 9, 1, 4 and 5 time units, respectively and the objective function value is equal to $f_{DBAP} = 23$ time units.

4. A reactive GRASP for the BAP

DBAP is a NP-hard problem, as stated in Cordeau et al. [2005], motivating the use of metaheuristics for its resolution. Metaheuristics are intelligent local search procedures with the

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
Berth 1																						
Berth 2																						

Figure 2: Berth allocation solution example $S = \{T_1^1 = 8, T_2^1 = 9, T_3^2 = 3, T_4^1 = 4, T_5^2 = 11\}$.

ability to explore the space of solutions and escape from local optimums.

The Greedy Randomized Adaptive Search Procedure (GRASP) [Prais and Ribeiro, 2000] is an important metaheuristic with a wide variety of applications. It consists of a multi-start iterative search strategy that, in each iteration, constructs a greedy randomized solution which is submitted as initial to a local search procedure. Constructive strategies could be randomized by using a Restricted Candidate List (RCL). Given a set $X = \{x^1, x^2, \dots, x^{|X|}\}$ of all candidate elements to compose a solution and g a function to compute the cost of each one of them, let $\underline{g}(X) = \min_{1 \leq j \leq |X|} \{g(x^j)\}$ and $\bar{g}(X) = \max_{1 \leq j \leq |X|} \{g(x^j)\}$ be, respectively, the minimum and maximum greedy cost over all feasible candidate elements to compose a solution. Thus, the restricted candidate list RCL is formed by all feasible elements $x^y \in X$, $1 \leq y \leq |X|$, such that $g(x^y) \leq \underline{g}(X) + \alpha(\bar{g}(X) - \underline{g}(X))$, with $0.0 \leq \alpha \leq 1.0$. The case $\alpha = 0.0$ corresponds to a pure greedy algorithm, while $\alpha = 1.0$ is equivalent to a random construction. An element x^y is randomly selected from RCL to be part of the solution. The RLC size is determined by the parameter α , which can assume a constant or self-adjusted value based on the quality of the solutions found in previous iterations. The GRASP with the latter type of parameter is denoted reactive GRASP [Prais and Ribeiro, 2000]. Following the multi-start process, different initial solutions are created, in order to promote diversification. The best solution among all iterations is the algorithm output.

In this section we propose and describe a reactive GRASP strategy to solve the DBAP, the $GRASP_{BAP}$. Section 4.1 describes an algorithm to determine the berth and the berthing time of each ship. The $GRASP_{BAP}$ constructive and local search procedures are presented in section 4.2 and 4.3, respectively, and its whole pseudocode, in section 4.4.

4.1. Programming Heuristic

The programming heuristic described in this section is based in Mauri et al. [2008]. It sorts, by the ships' arrival time (a_i) on the port, the handling vessel sequence of each berth B^k , $1 \leq k \leq |\mathcal{P}|$, from the input set of berths $\mathcal{P} = \{B^1, B^2, \dots, B^{|\mathcal{P}|}\}$ together with berth attributes (the berths time window $[o^k, c^k]$) and the input set of vessels $\mathcal{V}$ together with vessels attributes (the ships time window $[a_i, b_i]$ and its handling time $t_i^k, \forall k \in \mathcal{P}$).

The ordered berth handling sequence given by programming heuristic is useful to improve the objective function since the handling of the first incoming ships reduce their waiting time and, consequently, the objective function. The programming heuristic also compute the berthing times of each ship $v_i \in \mathcal{V}$ allocated to the berth $B^{k(i)} \in \mathcal{P}$. Thus, each initial (obtained from the constructive algorithm) or improved solution (obtained from the local search procedure) is submitted to this heuristic before the calculation of f_{DBAP} .

The algorithm pseudocode is reproduced in Algorithm 1 and denoted as PH_{DBAP} . As input, the algorithm receives the set of berths $\mathcal{P}$ together with berth attributes (the berths time window $[o^k, c^k]$) and a set of vessels $\mathcal{V}$ together with vessels attributes (the ships time window $[a_i, b_i]$ and its handling time $t_i^k, \forall k \in \mathcal{P}$). The algorithm outputs a DBAP solution $S = \{T_1^{k(1)}, T_2^{k(2)}, \dots, T_{|\mathcal{V}|}^{k(|\mathcal{V}|)}\}$ such that for each ship $v_i \in \mathcal{V}$, we know its allocated berth $k(i) \in \mathcal{P}$ and its berthing time $T_i^{k(i)}$.

Given the set of berths $\mathcal{P} = \{B^1, B^2, \dots, B^k\}$, $1 \leq k \leq |\mathcal{P}|$, the loop of lines 1 to 3 sorts in ascending order each berth B^k by ships arrival time on the port using function $\text{Order_Ships}(B^k)$.

Algorithm 1 $PH_{DBAP}()$

Input: A set of berths $\mathcal{P}$ with berth attributes (the berths time window $[o^k, c^k]$) and a set of vessels $\mathcal{V}$ with vessels attributes (the ships time window $[a_i, b_i]$ and its handling time $t_i^k, \forall k \in \mathcal{P}$).

Output: A DBAP solution S

```
1: for  $k \leftarrow 1, \dots, |\mathcal{P}|$  do
2:    $B^k \leftarrow \text{Order\_Ships}(B^k)$ ;
3: end for
4:  $S \leftarrow \emptyset$ ;
5: for  $i \leftarrow 1, \dots, |\mathcal{V}|$  do
6:   if  $v_i$  is the first ship of berth  $B^{k(i)}$  then
7:      $T_i^{k(i)} \leftarrow \max(a_i, o^{k(i)})$ ;
8:   else
9:      $T_i^{k(i)} \leftarrow \max(a_i, T_{\text{Prec}(i)}^{k(i)} + t_{\text{Prec}(i)}^{k(i)})$ ;
10:  end if
11:   $S \leftarrow S \cup T_i^{k(i)}$ ;
12: end for
13: return  $S$ 
```

Given the example of Table 1 and the set of berths $\mathcal{P} = \{B^1 = \{v_1, v_2, v_4\}, B^2 = \{v_3, v_5\}\}$ (see Figure 4) built by Algorithm 2 (to be discussed in next section), Figure 3 shows the ordered set of berths processed by function $\text{Order_Ships}()$. Notice that the berthing times are not defined yet. We have only the berth assignment and the sequence of vessels in each berth. After applying function $\text{Order_Ships}(B^k)$ in all berths, we have the updated set $\mathcal{P} = \{B^1 = \{v_4, v_1, v_2\}, B^2 = \{v_3, v_5\}\}$ as showed in Figure 3.

In line 4 an empty partial solution S is initialized. The loop of lines 5 to 12 scans each ship $v_i \in \mathcal{V}$, computing its berthing time $T_i^{k(i)}$ which indicates when the ship v_i is attended in berth $B^{k(i)}$. If v_i is the first ship of berth $B^{k(i)}$ (test in line 6), the $T_i^{k(i)}$ is calculated in line 7 as the maximum between the arriving time of the ship v_i (a_i) and the opening time ($o^{k(i)}$) of berth $B^{k(i)}$, otherwise we assign the sum of the berthing ($T_{\text{Prec}(i)}^{k(i)}$) and the processing times ($t_{\text{Prec}(i)}^{k(i)}$) of the immediately preceding ship allocated to berth $B^{k(i)}$ given by function $\text{Prec}(i)$ in line 9. The partial solution S is updated in line 11 and it is returned as a complete feasible solution in line 13.

Given the example of Table 1 and the ordered set $\mathcal{P} = \{B^1 = \{v_4, v_1, v_2\}, B^2 = \{v_3, v_5\}\}$ as showed in Figure 3, Figure 2 shows the feasible solution $S = \{T_1^1 = 8, T_2^1 = 9, T_3^2 = 3, T_4^1 = 4, T_5^2 = 11\}$ built by the loop from line 5 to 12 in Algorithm 1.

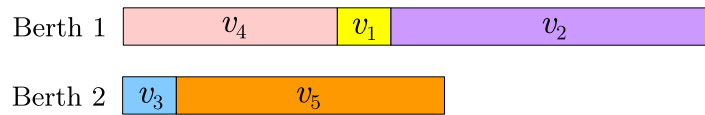


Figure 3: Ordered berths example $\mathcal{P} = \{B^1 = \{v_4, v_1, v_2\}, B^2 = \{v_3, v_5\}\}$.

4.2. Constructive Heuristic

In this work, we propose a new greedy randomized algorithm to distribute ships over the berths, in order to create initial solutions to submit to the local search procedures. We use it as the Construction Heuristic (CH) of $GRASP_{BAP}$. The CH algorithm uses Algorithm 1 to define ship berthing times. The CH pseudocode is presented in Algorithm 2. As input, the algorithm receives the parameter $\alpha \in (0.0, 1.0]$, the set of berths $\mathcal{P}$ with berths attributes and the set of incoming vessels $\mathcal{V}$ with its attributes. The algorithm outputs a DBAP solution S .

Algorithm 2 $CH_{DBAP}()$

Input: $\alpha \in (0.0, 1.0]$, $\mathcal{P}$, $[o^k, c^k] : \forall B^k \in \mathcal{P}$, $\mathcal{V}$, $[a_i, b_i] : \forall v_i \in \mathcal{V}$, $t_i^k : \forall v_i \in \mathcal{V}$ and $\forall k \in \mathcal{P}$.

Output: A DBAP solution S .

```
1: for  $k \leftarrow 1, \dots, |\mathcal{P}|$  do
2:    $B^k \leftarrow \emptyset$ ;
3: end for
4: for  $i \leftarrow 1, \dots, |\mathcal{V}|$  do
5:    $B^{rs} \leftarrow \text{RandomBerthSelection}(\mathcal{P}, \alpha)$ ;
6:    $B^{rs} \leftarrow B^{rs} \cup v_i$ ;
7:    $\text{Update}(\mathcal{P}, B^{rs})$ ;
8: end for
9:  $S \leftarrow PH_{DBAP}(\mathcal{P}, \mathcal{V})$ ;
10: return  $S$ 
```

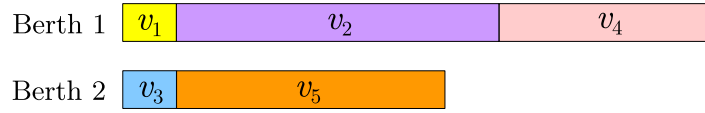


Figure 4: Berth assignment example $\mathcal{P} = \{B^1 = \{v_1, v_2, v_4\}, B^2 = \{v_3, v_5\}\}$.

Algorithm 2 starts emptying each berth B^k , $1 \leq k \leq |\mathcal{P}|$, from the input set of berths $\mathcal{P} = \{B^1, B^2, \dots, B^k\}$ in the loop from line 1 to 3. Next, each berth is filled iteratively by the allocation of a ship v_i in the loop from line 4 to 8. In line 5, function `RandomBerthSelection()` receives the current set of berths $\mathcal{P}$ and the random parameter α , and it returns the berth B^{rs} to allocate the current vessel v_i . The berth selection is guided by a greedy function based on the number of ships currently allocated in each berth in $\mathcal{P}$. The greedy function is defined as $g(\mathcal{P}) = \min_{1 \leq k \leq |\mathcal{P}|} \{|B^k|\}$, where $|B^k|$ is the number of vessels currently allocated in berth B^k . Berth B^{rs} is picked at random from the RCL formed by all feasible elements $B^k \in \mathcal{P}$, $1 \leq k \leq |\mathcal{P}|$, such that $|B^k| \leq \underline{g}(\mathcal{P}) + \alpha(\bar{g}(\mathcal{P}) - \underline{g}(\mathcal{P}))$, as discussed previously. Next, vessel v_i is assigned to berth B^{rs} and the set $\mathcal{P}$ is updated in lines 6 and 7, respectively.

When the loop from line 4 to 8 finishes we have all ships allocated to exactly one berth in $\mathcal{P}$. Given the input example of Table 1, Figure 4 shows a pure greedy construction ($\alpha = 0.0$) which finds the set of berths $\mathcal{P} = \{B^1 = \{v_1, v_2, v_4\}, B^2 = \{v_3, v_5\}\}$. In line 9, we call Algorithm 1 passing $\mathcal{P}$ as input. Then, Algorithm 1 assigns berthing time to all ships and returns a feasible solution to be stored in S . Algorithm 2 returns feasible solution S in line 10.

4.3. Local Search Algorithms

In this section we describe the local search procedure used in $GRASP_{DBAP}$. The algorithm implemented is a first-improvement local search. The neighborhood $N(S)$ of a solution S is the set of all new solutions S' generated by the application of a movement m over S , i.e., $S' = m(S)$. We implemented two different neighborhoods, considering m as *move* or *swap*.

In the context of DBAP, a *move* application over a solution S transfers a vessel v_i from a berth B^k to another berth B^l , $k \neq l$. Figure 5 illustrates this movement: given the solution S showed in Figure 2, the ship v_1 is moved from B^1 to B^2 afterwards but still respecting both ship and berth constraints (see Table 1). The *swap* movement chooses two berths B^k and B^l , $k \neq l$, and change the ships allocated to them. This movement is showed in Figure 6 using solution S showed in Figure 2 as initial solution and exchanging v_1 with v_5 . As the generation of the initial solution, these movements ensure that each ship is assigned only to berths that can serve them. More

details about these neighborhoods are presented in [Oliveira et al., 2012]. Algorithm 3 presents the pseudocode of the local search procedure used in the $GRASP_{DBAP}$. The neighborhood to be applied is fixed by input $neigh$.

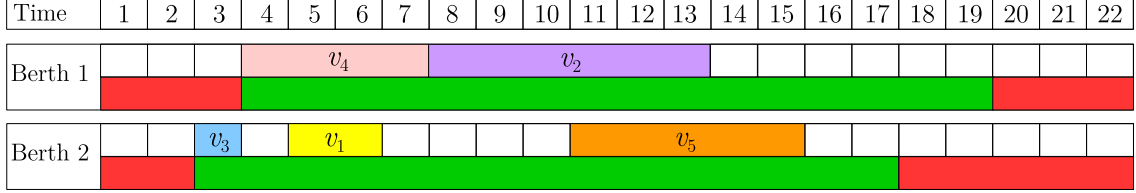


Figure 5: LS_{DBAP} example using the *move* movement.

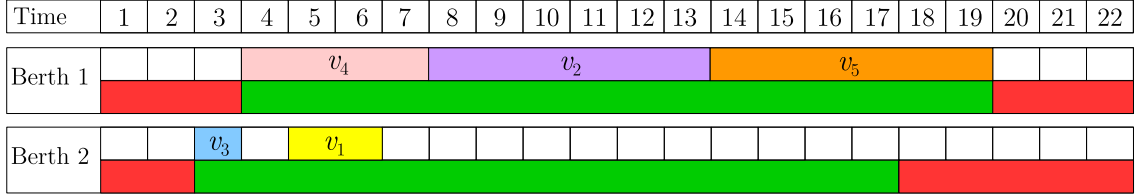


Figure 6: LS_{DBAP} example using the *swap* movement.

As input, Algorithm 3 receives the initial solution S , the parameter $neigh$, the set of berths $\mathcal{P}$ with berths attributes and the set of incoming vessels $\mathcal{V}$ with its attributes. The algorithm outputs the best DBAP solution found in the neighborhood. It starts assigning the initial solution S to the best and current solutions and defining the two possible neighborhoods (lines 1 and 2). While it is possible to improve the best solution (loop of lines 3 to 8), it searches a solution better than $S_{current}$ in the neighborhood defined by parameter $neigh = \{move, swap\}$ (line 4). If the current solution $S_{current}$ found in line 4 is better than the best solution found so far S^* (test in line 5), the best solution S^* is updated with $S_{current}$ in line 6. The algorithm outputs the best solution found S^* .

Algorithm 3 $LS_{DBAP}()$

Input: $S, neigh, \mathcal{P}, [o^k, c^k] : \forall B^k \in \mathcal{P}, \mathcal{V}, [a_i, b_i] : \forall v_i \in \mathcal{V}, t_i^k : \forall v_i \in \mathcal{V} \text{ and } \forall k \in \mathcal{P}$.

Output: Best DBAP solution found S^* .

```

1:  $S^* \leftarrow S_{current} \leftarrow S$ ;
2:  $\mathcal{N} = \{N_{move}, N_{swap}\}$ ;
3: while  $S_{current} = S^*$  do
4:    $S_{current} \leftarrow SearchNeighborhood(S_{current}, \mathcal{N}, neigh, \mathcal{P}, \mathcal{V})$ ;
5:   if  $f_{DBAP}(S_{current}) < f_{DBAP}(S^*)$  then
6:      $S^* \leftarrow S_{current}$ ;
7:   end if
8: end while
9: return  $S^*$ 

```

Algorithm 4 The algorithm $GRASP_{BAP}$ ()

Input: $neigh, MaxIter, niter, RandomSeed, \mathcal{A}, \beta, \mathcal{P}, [o^k, c^k] : \forall B^k \in \mathcal{P}, \mathcal{V}, [a_i, b_i] : \forall v_i \in \mathcal{V}, t_i^k : \forall v_i \in \mathcal{V} \text{ and } \forall k \in \mathcal{P}$.

Output: Best DBAP solution found S_{best} .

```
1:  $S_{best} \leftarrow \emptyset; f_{DBAP}(S_{best}) \leftarrow +\infty;$ 
2:  $iter \leftarrow 1; no\_improvement \leftarrow 0;$ 
3:  $\alpha \leftarrow \text{SetAlpha}(\mathcal{A}, RandomSeed);$ 
4: while  $iter \leq MaxIter$  and  $no\_improvement \leq niter$  do
5:    $S_1 \leftarrow CH_{DBAP}(\mathcal{P}, \mathcal{V}, \alpha);$ 
6:    $S_2 \leftarrow LS_{DBAP}(S_1, neigh, \mathcal{P}, \mathcal{V});$ 
7:   if  $f_{DBAP}(S_2) < f_{DBAP}(S_{best})$  then
8:      $S_{best} \leftarrow S_2;$ 
9:      $no\_improvement \leftarrow 0;$ 
10:  end if
11:   $\alpha \leftarrow \text{UpdateAlpha}(\beta, \mathcal{A}, \alpha);$ 
12:   $iter \leftarrow iter + 1;$ 
13:   $no\_improvement \leftarrow no\_improvement + 1;$ 
14: end while
15: return  $S_{best}$ 
```

4.4. The $GRASP_{BAP}$

The whole pseudo-code of the algorithm $GRASP_{BAP}$ is presented in Algorithm 4. As input, it receives the parameters $neigh = \{move, swap\}$, $MaxIter$, $niter$, $RandomSeed$, a discrete set $\mathcal{A} = \{\alpha_1, \dots, \alpha_{10}\}$ such that $\alpha_j \in (0.0, 1.0]$, $1 \leq j \leq 10$, a parameter $1 < \beta < MaxIter$, the set of berths $\mathcal{P}$ with berths attributes and the set of incoming vessels $\mathcal{V}$ with its attributes. Parameter $neigh$ defines the neighborhood, $move$ or $swap$, to be searched by Algorithm 3. The algorithm outputs the best DBAP solution found.

Algorithm 4 starts emptying the best solution found so far S_{best} and setting its objective function $f_{DBAP}(S_{best})$ with a huge value in line 1. The algorithm also initialize the iteration $iter$ and no improvement $no_improvement$ counters in line 2. The reactive parameter α is initialized using function $\text{SetAlpha}()$ which returns a random choice from set $\mathcal{A}$ guided by the $RandomSeed$. The loop from line 4 to 14 runs while the total number of iterations ($MaxIter$) or the maximum number of iterations without improvement ($niter$) are not reached.

The GRASP presented in Algorithm 4 explores the solution space towards the best one possible. For it, in each GRASP iteration, a feasible DBAP solution S_1 is construct with CH_{DBAP} (see Algorithm 2) in line 5 and submit S_1 as input to LS_{DBAP} (see Algorithm 3) in line 6. Algorithm LS_{DBAP} returns the best solution S_2 found in the neighborhood of S_1 . In the following, if S_2 is better than the best solution found so far (S_{best}), S_{best} is updated and the no improvement counter is reset (lines 7 to 10). In line 11, function $\text{UpdateAlpha}()$ sets parameter α using the reactive strategy described in Prais and Ribeiro [2000] with the probability distribution being updated after every β iterations. At the end of the loop, the iteration and no improvement counters are incremented (lines 12 and 13). The best over all solutions obtained at each loop iteration is returned as the answer of the algorithm (line 15).

5. Computational Results

The $GRASP_{DBAP}$ algorithm was tested over 30 instances, each with $N = 60$ vessels and $M = 13$ berths (named i01 ~ i30). These are random instances considering the context of Gioia Tauro Port, Italy, generated and made available in Cordeau et al. [2005]. In this section, we present and discuss the results of the computational experiments performed.

The $GRASP_{DBAP}$ was executed 10 times for each instance, with different seeds for the generation of random numbers, considering $MaxIter$ and $niter$ as 1000 and 10000, respectively. The parameter α was set by using the reactive strategy described in Prais and Ribeiro [2000] with the probability distribution being updated after every $\beta = 20$ iterations. In the tests for $GRASP_{DBAP}$, we investigated two distinct neighborhood sets for LS_{DBAP} , explored respectively with MOVE and SWAP, described in Section 4.3.

The code was written in C language, compiled using *gcc* version 4.8.4 and executed in an Intel Core C2D machine with a 2.93GHz clock and 2 GB of RAM memory running Linux Ubuntu 14.04 LTS for all the experiments.

Table 2 presents the results obtained by the $GRASP_{DBAP}$ algorithm, using MOVE ($GRASP^{move}$) and SWAP ($GRASP^{swap}$) in its local search procedure. The first column of this table indicates the instances identification. The second column shows the objective function value found by the greedy randomized algorithm with $\alpha = 0$. In the next four columns, we have information about $GRASP^{move}$: the best (*Min*) and average over ten executions (*Avg*) of the objective function values found, with their respective gains (*gain (%)*), with respect to the greedy value, computed as $(f_{DBAP}^{GRASP} - f_{DBAP}^{greedy})/f_{DBAP}^{greedy}$. Also, we have the CPU time of the best solution execution and the average CPU time over all executions. Similar information about $GRASP^{swap}$ is pointed out in the next four columns. The last two columns depict the optimal values reported in the literature (Opt.) and the gap between the best solution found by $GRASP^{swap}$ and the optimal (gap (%)), computed as $(f_{DBAP}^{GRASP} - \text{optimal})/\text{optimal}$.

Inst.	Greedy	$GRASP^{move}$				$GRASP^{swap}$				Opt.	gap (%)
	f_{DBAP}	f_{DBAP} (gain(%))		Time (s)		f_{DBAP} (gain (%))		Time (s)		f_{DBAP}	
		Min	Avg	Min	Avg	Min	Avg	Min	Avg		
i01	2001	1411 (-0,29)	1415 (-0,29)	58,69	49,47	1409 (-0,30)	1411 (-0,29)	112,42	87,10	1409	0,00
i02	1725	1261 (-0,27)	1261 (-0,27)	25,01	32,74	1261 (-0,27)	1261 (-0,27)	49,36	51,23	1261	0,00
i03	1555	1129 (-0,27)	1130 (-0,27)	32,68	42,07	1129 (-0,27)	1129 (-0,27)	53,65	60,21	1129	0,00
i04	1852	1303 (-0,30)	1304 (-0,30)	31,05	36,89	1302 (-0,30)	1302 (-0,30)	54,87	71,42	1302	0,00
i05	1599	1207 (-0,25)	1207 (-0,25)	23,25	27,36	1207 (-0,25)	1207 (-0,25)	47,74	52,35	1207	0,00
i06	1807	1261 (-0,30)	1261 (-0,30)	26,57	34,72	1261 (-0,30)	1261 (-0,30)	55,50	65,61	1261	0,00
i07	1899	1279 (-0,33)	1279 (-0,33)	42,64	33,98	1279 (-0,33)	1279 (-0,33)	49,79	54,72	1279	0,00
i08	1805	1300 (-0,28)	1303 (-0,28)	52,34	43,52	1299 (-0,28)	1299 (-0,28)	50,66	56,71	1299	0,00
i09	2082	1445 (-0,31)	1445 (-0,31)	27,45	37,36	1444 (-0,31)	1444 (-0,31)	63,24	67,46	1444	0,00
i10	1737	1213 (-0,30)	1213 (-0,30)	52,37	48,23	1213 (-0,30)	1213 (-0,30)	53,41	58,69	1213	0,00
i11	1939	1372 (-0,29)	1376 (-0,29)	30,73	44,69	1368 (-0,29)	1369 (-0,29)	82,37	82,59	1368	0,00
i12	1786	1332 (-0,25)	1336 (-0,25)	28,66	42,41	1325 (-0,26)	1326 (-0,26)	76,15	89,90	1325	0,00
i13	1958	1360 (-0,31)	1360 (-0,31)	26,84	43,02	1360 (-0,31)	1361 (-0,30)	63,93	65,14	1360	0,00
i14	1764	1233 (-0,30)	1233 (-0,30)	24,07	36,72	1233 (-0,30)	1233 (-0,30)	48,32	50,79	1233	0,00
i15	1643	1295 (-0,21)	1295 (-0,21)	29,18	45,21	1295 (-0,21)	1295 (-0,21)	48,48	58,64	1295	0,00
i16	2100	1370 (-0,35)	1374 (-0,35)	53,61	40,91	1365 (-0,35)	1368 (-0,35)	121,72	96,24	1364	0,07
i17	1756	1283 (-0,27)	1283 (-0,27)	22,78	23,40	1283 (-0,27)	1283 (-0,27)	40,55	45,79	1283	0,00
i18	1941	1345 (-0,31)	1346 (-0,31)	60,63	43,07	1345 (-0,31)	1345 (-0,31)	56,62	58,00	1345	0,00
i19	2012	1372 (-0,32)	1377 (-0,32)	27,19	37,39	1368 (-0,32)	1370 (-0,32)	110,60	89,65	1367	0,07
i20	2013	1330 (-0,34)	1334 (-0,34)	55,48	44,40	1328 (-0,34)	1329 (-0,34)	109,72	76,20	1328	0,00
i21	2026	1345 (-0,34)	1346 (-0,34)	34,71	43,81	1341 (-0,34)	1341 (-0,34)	67,27	87,98	1341	0,00
i22	1878	1328 (-0,29)	1333 (-0,29)	40,30	34,99	1326 (-0,29)	1326 (-0,29)	56,58	80,71	1326	0,00
i23	1768	1266 (-0,28)	1267 (-0,28)	40,00	38,79	1266 (-0,28)	1266 (-0,28)	55,46	58,19	1266	0,00
i24	1682	1260 (-0,25)	1260 (-0,25)	43,72	45,04	1260 (-0,25)	1260 (-0,25)	52,89	65,62	1260	0,00
i25	1888	1382 (-0,27)	1383 (-0,27)	49,26	44,42	1376 (-0,27)	1376 (-0,27)	60,79	74,04	1376	0,00
i26	1714	1324 (-0,23)	1326 (-0,23)	31,00	39,29	1318 (-0,23)	1319 (-0,23)	57,96	93,44	1318	0,00
i27	1681	1261 (-0,25)	1261 (-0,25)	28,27	36,53	1261 (-0,25)	1261 (-0,25)	51,22	54,06	1261	0,00
i28	1872	1360 (-0,27)	1364 (-0,27)	83,08	52,61	1360 (-0,27)	1361 (-0,27)	82,05	91,52	1359	0,07
i29	1719	1283 (-0,25)	1284 (-0,25)	44,71	40,92	1280 (-0,26)	1280 (-0,26)	57,81	72,18	1280	0,00
i30	1866	1354 (-0,27)	1360 (-0,27)	49,21	43,42	1344 (-0,28)	1345 (-0,28)	73,82	90,15	1344	0,00

Table 2: Objective function value found by the greedy randomized with $\alpha = 0$, the best and average over ten executions of the objective function values found by $GRASP^{move}$ and $GRASP^{swap}$ with their respective gains (%) with respect to the greedy value and CPU times, optimum value for each instance and the gap of the $GRASP_{DBAP}$ results to the optimum.

The results indicate the CPU time spent by the algorithm $GRASP_{DBAP}$ to achieve the best solution (Min), using the SWAP movement was, in average, of 65 seconds, almost twice as much time spent using the MOVE movement, which was 39 seconds. However, in terms of solution quality, the solution obtained by the algorithm was, in average, 0.15% worse than the optimal with MOVE, while using SWAP, the results were approximately equal to the optimal value: only for the instances i16, i19 and i28, the best solution found is far only 0.07% from the optimum. With respect to the average values computed over the 10 executions, we can highlight that they are very closely to the minimum, indicating the robustness of the algorithm for this scenario.

The number of iterations needed to obtain the best solution in the best $GRASP_{DBAP}$ execution is displayed, for each instance, in Figure 7. We can observe that both GRASP versions converged to the best solution found in less than 1500 iterations, except for i28, that executed 2241 iterations. $GRASP^{move}$ showed a slower convergence than $GRASP^{swap}$, once only for 10 instances, it takes less than 200 iterations to obtain the best solution, while $GRASP^{swap}$ needed this much, for the same purpose, for 20 instances.

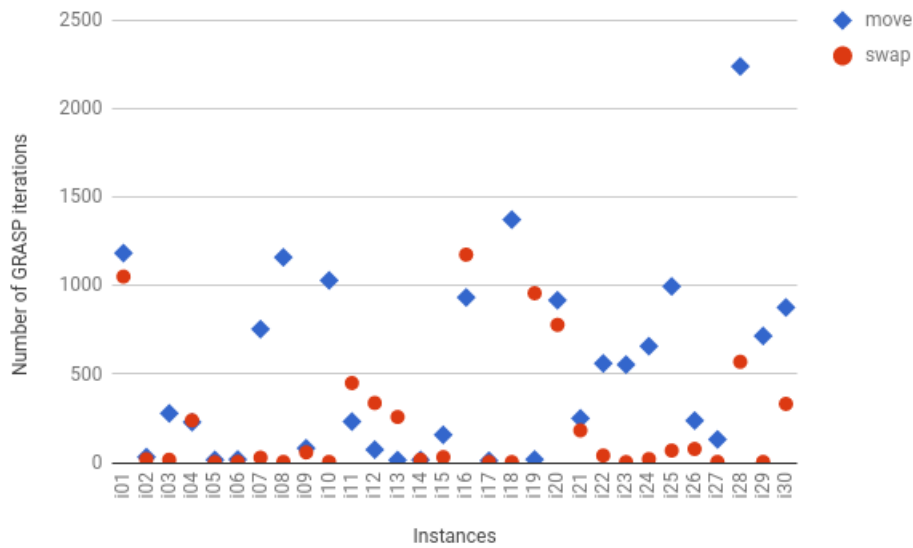


Figure 7: Number of iterations to find the best GRASP solution

6. Conclusion

Operating and managing a bulk cargo terminal is certainly a capital intensive project with huge investment in infrastructure, equipment and employees. Even a small increase in the number of serving of ships arriving at the terminal may bring significant improvement in the financial returns. In this article we studied the discrete variant of the Berth Allocation Problem in the export instance of maritime terminals that involves berthing vessels into the quay for transferring materials and proposed a GRASP algorithm to solve this problem.

Computational results showed that both GRASP versions are competitive strategies to the problem addressed, once they achieved the optimal solution in almost all the instances, with reasonable times. The $GRASP^{swap}$ accomplished, in average, better results than $GRASP^{move}$. Using the SWAP movement, only three instances did not achieve the optimum, but were very closely to it. Future work includes to investigate other local searches and strategies to aggregate to the GRASP and apply to others sets of instances.

Acknowledgements

The work of Renato E. N. Moraes was partially supported by CNPq (National Council for Scientific and Technological Development), grant number 118501/2017-8.

References

- Barros, V. H., Costa, T. S., Oliveira, A. C., and Lorena, L. A. (2011). Model and heuristic for berth allocation in tidal bulk ports with stock level constraints. *Computers & Industrial Engineering*, 60(4):606 – 613.
- Bierwirth, C. and Meisel, F. (2015). A follow-up survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research*, 244(3):675 – 689.
- Buhrkal, K., Zuglian, S., Ropke, S., Larsen, J., and Lusby, R. (2011). Models for the discrete berth allocation problem: A computational comparison. *Transportation Research Part E: Logistics and Transportation Review*, 47(4):461 – 473.
- Cordeau, J.-F., Laporte, G., Legato, P., and Moccia, L. (2005). Models and tabu search heuristics for the berth-allocation problem. *Transportation Science*, 39(4):526–538.
- Ernst, A. T., Oğuz, C., Singh, G., and Taherkhani, G. (2017). Mathematical models for the berth allocation problem in dry bulk terminals. *Journal of Scheduling*, 20(5):459–473.
- Kordi, S., Davidovi, T., Kova, N., and Dragovi, B. (2016). Combinatorial approach to exactly solving discrete and hybrid berth allocation problem. *Applied Mathematical Modelling*, 40(21): 8952 – 8973.
- Lodewijks, G., Schott, D., and Ottjes, J. (2007). Modern dry bulk terminal design. *Bulk Solids Handling*, 27(6):364.
- Mauri, G. R., Oliveira, A. C. M., and Lorena, L. A. N. (2008). A hybrid column generation approach for the berth allocation problem. In van Hemert, J. and Cotta, C., editors, *Evolutionary Computation in Combinatorial Optimization*, vol. 4972 of *LNCS*, p. 110–122. Springer Berlin Heidelberg.
- Mauri, G. R., Ribeiro, G. M., Lorena, L. A. N., and Laporte, G. (2016). An adaptive large neighborhood search for the discrete and continuous berth allocation problem. *Computers & Operations Research*, 70:140 – 154.
- Nishi, T., Okura, T., Lalla-Ruiz, E., and Voß, S. (2017). A dynamic programming-based matheuristic for the dynamic berth allocation problem. *Annals of Operations Research*.
- Oliveira, R. M., Mauri, G. R., and Lorena, L. A. N. (2012). Clustering search for the berth allocation problem. *Expert Systems with Applications*, 39(5):5499 – 5505.
- Prais, M. and Ribeiro, C. C. (2000). Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing*, 12:164–176.
- Ribeiro, G. M., Mauri, G. R., de Castro Beluco, S., Lorena, L. A. N., and Laporte, G. (2016). Berth allocation in an ore terminal with demurrage, despatch and maintenance. *Computers & Industrial Engineering*, 96:8 – 15.
- Umang, N., Bierlaire, M., and Vacca, I. (2013). Exact and heuristic methods to solve the berth allocation problem in bulk ports. *Transportation Research Part E: Logistics and Transportation Review*, 54:14 – 31.
- Zhu, B., Zhou, Q., Tian, Y., and Chen, K. (2018). Analysis method of terminal throughput capacity for coal export terminals. *Journal of Waterway, Port, Coastal, and Ocean Engineering*, 144: 04017036.