# A generic feasibility-based heuristic scheme for the container pre-marshalling problem

Bo Jin[a], Andrew Lim[b,1], Ning Wang[c]

[a]*Department of Management Sciences, City University of Hong Kong, Tat Chee Avenue, Kowloon Tong, Hong Kong*
[b]*School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore 637371, Singapore*
[c]*Department of Information Management, School of Management, Shanghai University, Shanghai, China*

**Abstract**

This paper proposes a generic feasibility-based heuristic scheme for the container pre-marshalling problem, the problem of reorganizing containers inside a storage bay such that no more reshuffle is needed in the container retrieval process afterwards. The proposed heuristic scheme breaks out of the traditional heuristic strategy that handles containers in the reverse order of future retrievals. Instead, the concept of state feasibility is introduced, providing theoretical basis of the freedom of selecting the container to handle next. An implementation of the heuristic scheme, the greedy and speedy heuristic, is also proposed, embedded with four well designed heuristic techniques. Computational experiments are carried out to showcase the performance of the proposed heuristic.

## 1. Introduction

Since the commencement of containerization, the global usage of standardized containers has dramatically improved international trade. Containers enable the smooth flow of goods between multiple transportation modes without directly handling the freight. Nowadays, the stringent requirement of just-in-time operations from consignors prompts the entire container transportation industry to face more challenges.

*Email addresses:* msjinbo@cityu.edu.hk (Bo Jin), alim.china@gmail.com (Andrew Lim), ningwang@shu.edu.cn (Ning Wang)

[1]Andrew Lim is currently on no pay leave from Department of Management Sciences, City University of Hong Kong.

* See below.

The container yards are key components at maritime container terminals, which refer to space for the transshipment, handover, loading, consolidation, maintenance, and storage of containers. Some yards act as exchange venues for container transfers between different transportation modes, while others are used some act as caches for temporary storage, or and some act as warehouses for long-term storage.

Generally, a container yard is divided into several yard blocks, each of which A yard block consists of several parallel bays and A a bay is formed by a row of stacks. Usually the containers stored in the same bay has ve the same dimensions. In the container yard, E equipments such as rubber tyre gantry cranes, rail-mounted gantry cranes and or reach stackers in containes yards are frequently used. Figure 1 illustrates an example of a yard block.

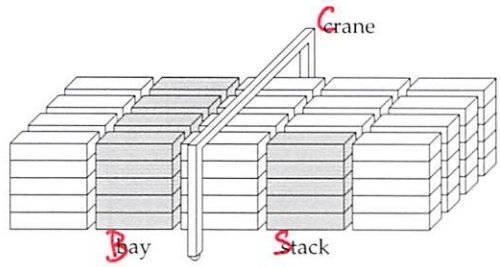


Figure 1: Yard block

Because Since containers in the same stack are organized follow the last-in-first-out manner, to retrieve lower containers, those other containers on top must first of it have to be relocated first. Such forced movements are is known as container *reshuffles*.

According to recent reviews [2, 14], terminals deal with three major decision problems related to container reshuffling are considered at terminals. The first problem is on the selection of storage locations for arriving containers. The second one is the pre-marshalling problem, which addresses the reorganization of that reorganize containers inside a storage area such that no more reshuffle is not required needed afterwards. The last one problem minimizes the total operational cost in the container retrieval process, measured by the number of relocations conducted carried out or the total operational time. The purpose of such studies aims is to improve the terminals' performance level S of the terminal, including their the throughput rate per berth or the turnaround time of vessels or road trucks [11].



* Container yards — the spaces dedicated to the transshipment, handover, loading, consolidation, maintenance, and storage of containers — are key components of maritime container terminals.

This paper investigates the *container pre-marshalling problem* (CPMP), ~~that reshuffles~~ containers within a single bay before they are retrieved, ~~such that all the containers can be~~ retrieved without ~~extra~~ reshuffles. It is assumed that the retrieval order of the containers is known beforehand and ~~no~~ container arrives at or leaves from the bay during pre-marshalling. ~~There are a few~~ ~~papers~~ proposing solution ~~approaches for~~ the CPMP, including integer programming [13], the corridor method [3], labelling algorithm [8], neighborhood search [12], tree search [1], multi-start randomized heuristic [6], and beam search [15].

Expósito-Izquierdo et al. [6] provide a multi-start randomized heuristic for the CPMP, ~~which is~~ the first container-oriented heuristic in the literature. At every step of the heuristic, the misplaced container with the latest retrieval order is fixed to a properly chosen slot. Our previous work [15] proposes the target-guided heuristic (TGH) and two beam search algorithms for the problem. The TGH also handles containers sequentially in the reverse order of future retrievals. This work provides the first comprehensive analysis on every situation that the heuristic may face during the pre-marshalling process, especially for dense instances where empty slots are limited.

~~The contributions of this paper are twofold.~~ ~~The first is that~~ a state feasibility concept and a generic feasibility-based heuristic scheme are proposed. ~~The feasibility-based heuristic scheme~~ breaks ~~out of~~ the traditional heuristic strategy, greatly enriching the methodology for the ~~problem~~. The second contribution is an implementation of the generic scheme, the *greedy and speedy heuristic* (GASH), in which four well-designed techniques are applied. The GASH performs outstandingly on both dense and loose instances, ~~with~~ comparison ~~to~~ existing heuristics.

The remainder of this paper is structured as follows. Section 2 lists the notation used throughout this paper and Section 3 gives a mathematical description of the problem. Section 4 introduces the concept of ~~the~~ state feasibility, which is the basis of the proposed generic heuristic scheme. Sections 6 and 7 introduce the proposed heuristic scheme, available techniques ~~for the heuristic scheme~~ and the GASH. Section 8 illustrates the results of computational experiments. The last section concludes this paper and ~~points out~~ future research directions.

## 2. Notation summary

The notation for describing the problem is ~~listed~~ as follows.

*I. Global constants.*

| | |
|---|---|
| $N$ | number of containers |
| $S$ | number of stacks |
| $H$ | height limitation of stacks |
| $P$ | number of priorities |
| $E$ | number of empty slots, $E = SH - N$ |
| $U$ | number of unreachable tiers, $U = \max\{H - E, 0\}$ |
| $\mathbb{C}$ | set of containers |
| $\mathbb{S}$ | set of stacks, $\mathbb{S} = \{1, \ldots, S\}$ |

*II. Common symbols and functions.*

| | |
|---|---|
| $\mathsf{L}$ | layout |
| $c$ | container |
| $s$ | stack index |
| $t$ | tier index |
| $p$ | priority value |
| $(s, t)$ | slot or the container located in |
| $h(s)$ | height of stack $s$, vectorized as $\boldsymbol{h}$ |
| $e(s)$ | number of empty slots in stack $s$, $e(s) = H - h(s)$ |
| $o(s)$ | orderly height of stack $s$, vectorized as $\boldsymbol{o}$ |
| $f(s)$ | fixed height of stack $s$, vectorized as $\boldsymbol{f}$ |
| $(\mathsf{L}, \boldsymbol{f})$ | state |
| $p(c)$ | priority value of container $c$ |
| $p(s, t)$ | priority value of container $(s, t)$ |
| $q(s, t)$ | capability of an occupied slot $(s, t)$ |
| $q^f(s)$ | capability of the top fixed slot in stack $s$, $q^f(s) = q(s, f(s))$ |

Resource - / demand - related

*III. ~~Resource/demand related~~ functions.*

| | |
|---|---|
| $d(p)$ | order-$p$ demand, or the number of unfixed containers with priority $p$ |

| | |
|---|---|
| $D(p)$ | order-$p$ accumulate demand, $D(p) = \sum_{\varphi=p}^{P} d(\varphi)$, vectorized as $\boldsymbol{D}$ |
| $r(p)$ | order-$p$ resource, $r(p) = \sum_{q^f(s)=p}(H - f(s))$ |
| $R(p)$ | order-$p$ accumulate resource, $R(p) = \sum_{\varphi=p}^{P} r(\varphi)$, vectorized as $\boldsymbol{R}$ |
| $\Delta(p)$ | order-$p$ surplus, $\Delta(p) = R(p) - D(p)$, vectorized as $\boldsymbol{\Delta}$ |

The following is the extra notation used ~~for~~ describing the proposed algorithms.

*IV. Task-related symbols.*

| | |
|---|---|
| $c^*$ | target container |
| $s^+$ | stack of the target container |
| $t^+$ | tier of the target container |
| $s^-$ | aim stack |
| $b$ | number of blocking containers |
| $a$ | number of slots available for blocking containers |

*V. Extra symbols.*

| | |
|---|---|
| $s^{\mathrm{src}}$ | source stack (sender) of a move |
| $s^{\mathrm{dst}}$ | destination stack (receiver) of a move |
| $s^{\mathrm{tmp}}$ | interim stack for temporarily storing the target container |
| $\vec{v}$ | evaluation tuple of a move, lexicographically comparable |
| $\mathbb{R}$ | receiver set |
| $\mathbb{I}$ | set of potential interim stacks |
| $\mathbb{F}$ | set of potential interim stacks which are full currently |

*VI. Extra functions.*

| | |
|---|---|
| $g(s)$ | stable height of stack $s$, vectorized as $\boldsymbol{g}$ |
| $m(s)$ | messiness (largest unstable priority value) of stack $s$, $m(s) = \max_{g(s)<t\le h(s)} p(s,t)$ |
| $\alpha(s^+, s)$ | number of slots available after the move from stack $s^+$ to $s$ |
| $\delta(p, q)$ | demand between $p$ exclusive and $q$ inclusive, $\delta(p,q) = \sum_{\varphi=p+1}^{q} d(\varphi)$ |

## 3. Problem description

The container pre-marshalling problem (CPMP) is restricted to the bay size, or precisely the dimensions of the operating crane. An instance (problem input) includes an initial layout of $N$ containers, which are distributed in a single bay with $S$ stacks ($S \geq 3$) and $H$ tiers ($H \geq 2$), leaving $E$ empty slots ($E = SH - N, E \geq 2$).

In dense instances such that $E < H$, the bottom $H - E$ tiers of the bay are unreachable. Let $U = \max\{H - E, 0\}$ denote the number of unreachable tiers. As a simple fact, containers in the reachable tiers can be permuted into any wanted arrangement.

Let $\mathbb{S} = \{1, \ldots, S\}$ be the set of stacks. Hereafter for simplification of description, when a stack $s$ is mentioned without declaring its domain, it is assumed that $s \in \mathbb{S}$. The height of stack $s$ is denoted by $h(s)$ and $e(s) = H - h(s)$ denotes the number of empty slots in this stack. Note that the height of stacks should not exceed $H$.

The containers in the bay are categorized into $P$ groups, based on shipment destination, weight or some other factors. Each group is assigned a priority value from 1 to $P$, such that a smaller priority value indicates an earlier retrieval order (loading order to the vessel). The $t$-th slot (from the bottom up) of stack $s$ or the container located inside is denoted by a pair $(s, t)$. The priority value of a container $c$ or the container located in slot $(s, t)$ are denoted by $p(c)$ or $p(s, t)$.

In order to prevent confusion in the following expression, hereafter we only use "small/large priority value" to describe the retrieval order of a container, instead of "high/low priority" which is commonly used in the literature though.

A container is *orderly* if it is supported directly by the ground or another orderly container with equal or larger priority value; otherwise it is *disorderly*. The orderly height (number of orderly containers) of stack $s$ is denoted by $o(s)$. Other phrases referring to "orderly/disorderly" in the recent literature include "well/badly placed" [1], "well-located/non-located" [6] and "clean/dirty" [15]. Table 2 gives an example of a bay with $S = 5$, $H = 4$ and $N = 13$; containers are represented by boxes with their priority values marked inside. In addition, boxes with grey background are the disorderly containers.

Let us define the capability of an occupied slot $(s, t)$ by $q(s, t) = p(s, t)$ if the container inside

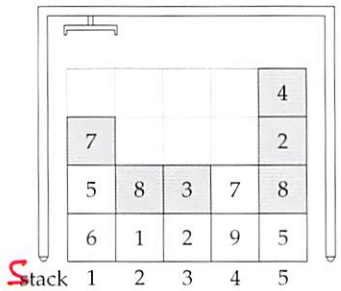


Figure 2: Container bay

is orderly, otherwise $q(s,t) = 0$. Specially, regarding the ground as an occupied slot at tier 0 with priority value $P$. Therefore, we can verify the placement of a container $(s,t)$ easily by checking if $p(s,t) \le q(s,t-1)$.

The objective of the CPMP is to find an optimized sequence with the fewest container moves, so as to convert the initial layout into a final layout, where all the containers are orderly.

## 4. State and feasibility

A state $(\mathsf{L}, \boldsymbol{f})$ is a pair consisting of a layout $\mathsf{L}$ and a *fix* vector $\boldsymbol{f}$. The vector $\boldsymbol{f}$ indicates the fixed height (number of fixed containers) of each stack in $\mathsf{L}$. Further moves on the fixed containers are not allowed. The fix status of the containers in a layout raises the issue of state feasibility.

**Proposition 1** (Necessary conditions for state feasibility)**.** *The necessary conditions for the feasibility of a state* $(\mathsf{L}, \boldsymbol{f})$ *are* $\boldsymbol{f} \le \boldsymbol{o}$ *and* $\Delta = \boldsymbol{R} - \boldsymbol{D} \ge \boldsymbol{0}$.

Proposition 1 gives the necessary conditions for state feasibility, where $\boldsymbol{R}$ and $\boldsymbol{D}$ are the accumulative resource and demand vectors, and $\Delta$ is the surplus vector. For $1 \le p \le P$, $R(p)$ is the order-$p$ accumulative resource, that is the total number of slots which can support containers with priority value $p$ (certainly, as well as less than $p$), and $D(p)$ is the order-$p$ accumulative demand, that is the number of unfixed containers with priority values not less than $p$. Let $d(p)$ be the number of unfixed containers with priority $p$, and $r(p) = \sum_{q^f(s)=p}(H - f(s))$ where $q^f(s) = q(s, f(s))$, we have $D(p) = \sum_{\varphi=p}^{P} d(\varphi)$ and $R(p) = \sum_{\varphi=p}^{P} r(\varphi)$. As a result, the non-negativity of the surplus vector $\Delta = \boldsymbol{R} - \boldsymbol{D}$ should be maintained in a feasible state.

For those states where fewer than $S - 2$ stacks are fully fixed, the necessary conditions are sufficient to ensure the feasibility of the stack. Taking the layout in Figure 2 as an example, Table 1 illustrates the computation for the surplus vector where $\boldsymbol{f} = \mathbf{1}$ is given. The supposed state is infeasible due to $\Delta(7) < 0$; that is, the containers with priority values 7, 8, and 9 cannot be well accommodated by enough unfixed slots.

Table 1: Computation for the surplus vector.

| $p$ | $d(p)$ | $D(p)$ | $r(p)$ | $R(p)$ | $\Delta(p)$ |
|---|---|---|---|---|---|
| 1 | 0 | 8 | 3 | 15 | 7 |
| 2 | 1 | 8 | 3 | 12 | 4 |
| 3 | 1 | 7 | 0 | 9 | 2 |
| 4 | 1 | 6 | 0 | 9 | 3 |
| 5 | 1 | 5 | 3 | 9 | 4 |
| 6 | 0 | 4 | 3 | 6 | 2 |
| 7 | 2 | 4 | 0 | 3 | −1 |
| 8 | 2 | 2 | 0 | 3 | 1 |
| 9 | 0 | 0 | 3 | 3 | 3 |

**Definition 1** (Extreme state)**.** *A state* $(\mathsf{L}, \boldsymbol{f})$ *is an extreme state if* $|\{s \in \mathbb{S} : f(s) = H\}| \geq S - 2$.

However, in an *extreme* state, i.e., $S - 2$ or more stacks are fully fixed, it needs an additional condition. Considering an extreme state, except for $S - 2$ fully fixed stacks, if the remaining two stacks are unable to be converted into orderly stacks by simply reshuffling unfixed containers from one to the other, we say it is a *tricky* state.

**Proposition 2** (Existence of tricky states)**.** *Tricky states exist if* $E < 2(H - 1)$.

## 5. Generic feasibility-based heuristic scheme

The heuristic firstly constructs the initial state according to the unreachable tiers. Starting from the initial state, the heuristic repeatedly fixes a target container $c^*$ to a properly chosen slot $(s^-, f(s^-) + 1)$, until all of the containers are fixed.

The procedure of the heuristic scheme is concisely given in Algorithm 1. Detailed explanation of the state initialization and valid tasks will be given in the remainder of this section.

**Algorithm 1:** Feasibility-based heuristic scheme.

```
1 begin
2     (L, f) := (L^0, U1);
3     repeat N − S U times do
4         (c*, s⁻) := the chosen valid task;
5         Accomplish (c*, s⁻) on L;
6         f(s⁻) := f(s⁻) + 1;
```

### 5.1. *Initialization*

The initial state is generated with the initial layout $\mathsf{L}^0$ and the initial fix vector $U\mathbf{1}$. It is a natural fact that when $S \geq 3$, the reachable (upper $H - U$) tiers can be permuted into any wanted placement, meanwhile the bottom $U$ tiers are unreachable. With the help of the state feasibility concept, we can check the solubility of an instance by checking the feasibility of the initial state.

**Proposition 3** (Instance solubility)**.** *A CPMP instance is solvable if the induced initial state is feasible.*

### 5.2. *Valid task*

At every step of the generic heuristic scheme, a task is determined and then accomplished through a sequence of moves. A task is a fix assignment that aims to fix a chosen target container to the lowest unfixed slot of a chosen aim stack.

A container–stack pair $(c, s)$ is a valid task for the current state $(\mathsf{L}, f)$ only if (necessary but not sufficient)

- $c$ is unfixed,
- $f(s) < H$,
- $p(c) \leq q^f(s)$, and
- $\Delta(\varphi) \geq H - f(s)$, for $p(c) < \varphi \leq q^f(s)$.

The last condition ensures the non-negativity of the resulting surplus vector, however, the feasibility of the resulting state is not sufficiently ensured, due to the existence of tricky states.

There are three ~~kinds of~~ methods ~~to~~ resolve ~~the~~ the tricky state issue.

1. Prevent from entering a tricky state when deciding the next task;
2. Design an ideal task accomplishment procedure, ~~which is able to make~~ the resulting extreme state feasible;
3. Allow entering a tricky state and resolve the regression issue.

Our previous work [15] adopts the last method ~~to deal with the issue~~. The regression issue involves temporarily moving an already fixed container during the task accomplishment, and moving it back to its originally fixed slot. The regression issue ~~will be further~~ discussed in Section 6.4.4.

In this paper, we introduce a tricky state avoidance technique (cf. Section 6.2), ~~With~~ that ~~technique~~, the heuristic would never enter a tricky state, saving unnecessary moves when solving dense instances.

## 6. Available techniques

In this section, we introduce four available techniques for the feasibility-based heuristic scheme.

### *6.1. Container stability*

Considering an orderly container $(s, t)$ in a feasible state $(\mathsf{L}, f)$, we say it is *stable* if the revised state $(\mathsf{L}, f')$ is feasible where $f'(s) = t$ and $f'(i) = f(i)$ for $i \neq s$. Denoting the stable height (number of stable containers) of stack $s$ by $g(s)$, we have $\boldsymbol{f} \leq \boldsymbol{g} \leq \boldsymbol{o} \leq \boldsymbol{h}$. If the top container of stack $s_1$, $c = (s_1, h(s_1))$ will be stable after being moved to a non-full stack $s_2$, we say stack $s_2$ can *stabilize* container $c$.

Figure 3 gives an example of state, where unstable containers are labelled with grey background.

An orderly but unstable container $(s, t)$ with $g(s) < t \leq o(s)$ indicates that its orderliness is valueless ~~since~~ it definitely needs reshuffles. Using the concept of container stability instead of container orderliness is more precise ~~since~~ an orderly yet unstable container has no opportunity to be fixed to its slot. For example, when a blocking container is becoming orderly yet unstable onto a destination stack, the attractiveness of such move should be reconsidered.

Note that the stability of containers should be recomputed after some container ~~being~~ fixed.

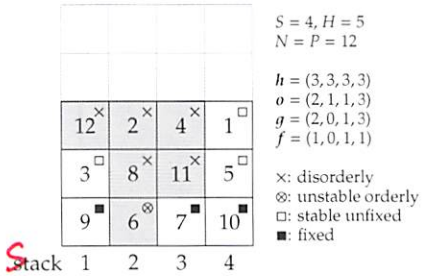


Figure 3: Stability of containers

### 6.2. *Tricky state avoidance*

When talking about selecting the next task, it is inevitable to face the tricky state. If a container–stack pair leads to an extreme state, the feasibility of the resulting state is unpredictable unless a complicated accomplishment procedure is designed. If unfortunately a tricky state is resulted, the remainder of the pre-marshalling work has to take the regression issue into consideration.

**Definition 2** (Pre-tricky state)**.** *A state is a pre-tricky state if* $|\{s \in \mathbb{S} : f(s) = H\}| = S - 3$, $|\{s \in \mathbb{S} : f(s) = H - 1\}| \geq 1$ *and* $\sum_{s \in \mathbb{S}} f(s) < N - 2$.

When the current state is *pre-tricky*, we eliminate the container–stack pairs $(c, s)$, in order to prevent from entering a tricky state.

### 6.3. *Bottom tiers protection*

In the feasibility-based heuristic scheme, the target container can be freely chosen as long as the task is valid, not limited to the largest priority value. After the target container $c^*$ is fixed to the aim slot $(s^-, f(s^-) + 1)$, the surplus $\Delta(\varphi)$ is reduced by $H - f(s^-)$, for $p(c^*) < \varphi \leq q^f(s^-)$.

The bottom tiers protection technique balances the trade-off between the target freedom and the surplus loss, by eliminating the container–stack pairs $(c, s)$ such that $f(s) < 2$ and $\delta(p(c), q^f(s)) \geq S$, where $\delta(p, q) = \sum_{\varphi=p+1}^{q} d(\varphi)$. That is, the affected demand quantity $\delta(p(c), q^f(s))$ should not reach the threshold $S$ in the bottom two tiers that are protected.

Moreover, the number of tiers protected and the threshold can also be adjusted optionally.

*6.4. Speedy task accomplishment procedure*

After the task is decided by specific rules, it is accomplished by a sequence of moves, resulting in a new state. The speedy task accomplishment procedure (STAP) provides a structured framework to finish the given task with the fewest moves, providing customizable interfaces with several user-defined functions.

Common functions of the STAP are defined in Algorithm 2 and the user-defined functions include

- `EvalMove`$(s^{\text{src}}, s^{\text{dst}})$ returns the penalty of the move from stack $s^{\text{src}}$ to $s^{\text{dst}}$;
- `Interim`$(\mathbb{I})$ selects the interim stack from set $\mathbb{I}$; and
- `InterimFull`$(\mathbb{F})$ selects the interim stack from set $\mathbb{F}$ which is composed by full stacks.

Function `Relocate`$(s^{\text{src}}, k, \mathbb{R})$ performs $k$ relocations from a *sender* stack $s^{\text{src}}$ to a *receiver* set $\mathbb{R}$. For every top container of the sender, the destination stack $s^{\text{dst}}$ is properly selected from $\mathbb{R}$ according to the evaluations by the user-defined function `EvalMove`$(s^{\text{src}}, s^{\text{dst}})$. The evaluations of moves are represented by numerical tuples, which are lexicographically comparable.

Function `BiReceiver`$(s^{\text{src}}, k_1, \mathbb{R}_1, k_2, \mathbb{R}_2)$ relocates $k_1$ and $k_2$ containers from one sender stack $s^{\text{src}}$ to two receiver sets, $\mathbb{R}_1$ and $\mathbb{R}_2$, respectively. Similarly, function `BiSender`$(s_1^{\text{src}}, k_1, s_2^{\text{src}}, k_2, \mathbb{R})$ performs relocations from two senders, stacks $s_1^{\text{src}}$ and $s_2^{\text{src}}$, to the same receiver set $\mathbb{R}$, the respective quantities are $k_1$ and $k_2$. The moving order of the two top containers from two senders is determined by the smaller evaluation tuple.

Function `MoveNeed`$(c, s)$ returns the actual number of moves performed by the STAP to accomplish the given task $(c, s)$.

Denote the target container by $c^*(s^+, t^+)$ and the aim stack by $s^-$. The task is *immediate* if it is located in the aim slot already; *internal* if the aim slot is below the target container in the same stack; or *external* if the aim slot is in a different stack.

In the following, the STAP is introduced according to the type of the next task to accomplish.

**Algorithm 2:** Speedy task accomplishment procedure: Common functions.

```
1  procedure Move(s^src, k, s^dst)
2      Move k containers from stack s^src to stack s^dst;
3  procedure Relocate(s^src, k, ℝ)
4      repeat k times do
5          ℝ' := {s ∈ ℝ : h(s) < H};
6          s^dst := arg min_{s∈ℝ'} EvalMove(s^src, s);
7          Move(s^src, 1, s^dst);
8  procedure BiReceiver(s^src, k_1, ℝ_1, k_2, ℝ_2)
9      i := k_1, j := k_2;
10     repeat k_1 + k_2 times do
11         if j = 0 then
12             Relocate(s^src, i, ℝ_1);
13             i := 0;
14         else if i = 0 then
15             Relocate(s^src, j, ℝ_2);
16             j := 0;
17         else
18             ℝ'_1 := {s ∈ ℝ_1 : h(s) < H};
19             ℝ'_2 := {s ∈ ℝ_2 : h(s) < H};
20             s_1^dst := arg min_{s∈ℝ'_1} EvalMove(s^src, s);
21             v⃗_1 := EvalMove(s^src, s_1^dst);
22             s_2^dst := arg min_{s∈ℝ'_2} EvalMove(s^src, s);
23             v⃗_2 := EvalMove(s^src, s_2^dst);
24             if v⃗_1 ≤^lex v⃗_2 then
25                 Move(s^src, s_1^dst) i := i − 1;
26             else
27                 Move(s^src, s_2^dst) j := j − 1;
28 procedure BiSender(s_1^src, k_1, s_2^src, k_2, ℝ)
29     i := k_1, j := k_2;
30     repeat k_1 + k_2 times do
31         if j = 0 then
32             Relocate(s_1^src, i, ℝ);
33             i := 0;
34         else if i = 0 then
35             Relocate(s_2^src, j, ℝ);
36             j := 0;
37         else
38             ℝ' := {s ∈ ℝ : h(s) < H};
39             s_1^dst := arg min_{s∈ℝ'} EvalMove(s_1^src, s);
40             v⃗_1 := EvalMove(s_1^src, s_1^dst);
41             s_2^dst := arg min_{s∈ℝ'} EvalMove(s_2^src, s);
42             v⃗_2 := EvalMove(s_2^src, s_2^dst);
43             if v⃗_1 ≤^lex v⃗_2 then
44                 Move(s_1^src, 1, s_1^dst);
45                 i := i − 1;
46             else
47                 Move(s_2^src, 1, s_2^dst);
48                 j := j − 1;
49 function MoveNeed(c, s)
50     if (c, s) is immediate then
51         return 0;
52     else
53         return actual number of moves needed by the
           STAP to accomplish the task (c, s);
```

### 6.4.1. *Immediate task*

An immediate task does ~~not need any~~ require a move ~~since~~ because the target container is already located in the aim slot; that is, $\texttt{MoveNeed}(c^*, s^-) = 0$ for an immediate task $(c^*, s^-)$.

### 6.4.2. *Internal task*

For an internal task, let $a$ denote the number of empty slots in $\mathbb{S} \setminus \{s^+\}$ with the exclusion of the highest non-full stack; that is, $a = E - e(s^+) - \min\{e(s) > 0 : s \neq s^+\}$. If multiple stacks are with the maximum height, only one is excluded. Let $b_1$ and $b_2$ denote the numbers of blocking containers above and below $c^*$ in stack $s^+$, respectively. The pseudo-code of the STAP for an internal task is given in Algorithm 3, which is divided into three situations:

- I1: $a \geq b_2$;
- I2: $a < b_2$ & $|\{s \neq s^+ : h(s) < H\}| > 1$; and
- I3: $a < b_2$ & $|\{s \neq s^+ : h(s) < H\}| = 1$.

**Algorithm 3:** Speedy task accomplishment procedure: Internal task.

**target container**: $c^*(s^+, t^+)$
**aim slot**: $(s^+, f(s^+) + 1)$
**slot supply**: $a = E - e(s^+) - \min\{e(s) > 0 : s \neq s^+\}$
**blocking above**: $b_1 = h(s^+) - t^+$
**blocking below**: $b_2 = t^+ - f(s^+) - 1$

**function** $\alpha(s^+, s^{\text{dst}})$
  $e^{\min} := \min\{e(s) > 0 : s \neq s^+\}$;
  $e^{\text{sec}} := \min\{e(s) > e^{\min} : s \neq s^+\}$;
  $k^{\min} := |\{s \neq s^+ : e(s) = e^{\min}\}|$;
  **if** $e(s^{\text{dst}}) > e^{\min}$ **then**
    **return** $E - e(s^+) - e^{\min} - 1$;
  **else if** $e^{\min} \geq 2$ **then**
    **return** $E - e(s^+) - e^{\min}$;
  **else if** $k^{\min} = 1$ **then**
    **return** $E - e(s^+) - e^{\text{sec}} - 1$;
  **else**
    **return** $E - e(s^+) - 2$;

**case I1:** $a \geq b_2$
  **repeat** $b_1$ **times do**
    $\mathbb{R} := \{s \neq s^+ : h(s) < H,\ \alpha(s^+, s) \geq b_2\}$;
    `Relocate`$(s^+, 1, \mathbb{R})$;
  $\mathbb{I} := \{s \neq s^+ : h(s) < H,\ E - e(s^+) - e(s) \geq b_2\}$;
  $s^{\text{tmp}} :=$ `Interim`$(\mathbb{I})$;
  `Move`$(s^+, 1, s')$;
  `Relocate`$(s^+, b_2, \mathbb{S} \setminus \{s^+, s^{\text{tmp}}\})$;
  `Move`$(s^{\text{tmp}}, 1, s^+)$;
  // `I1:` $b_1 + b_2 + 2$ `moves`

**case I2:** $a < b_2$ & $|\{s \neq s^+ : h(s) < H\}| > 1$
  `Relocate`$(s^+, b_1, \mathbb{S} \setminus \{s^+\})$;
  $s_1^{\text{tmp}} :=$ `Interim`$(\mathbb{S} \setminus \{s^+\})$;
  `Move`$(s^+, 1, s_1^{\text{tmp}})$;
  $k_2 := E - e(s^+) - e(s_1^{\text{tmp}}) - 1$;
  `Relocate`$(s^+, k_2, \mathbb{S} \setminus \{s^+, s_1^{\text{tmp}}\})$;
  Find $s_2^{\text{tmp}}$ s.t. $s_2^{\text{tmp}} \notin \{s^+, s_1^{\text{tmp}}\}$ & $h(s_2^{\text{tmp}}) < H$;
  `Move`$(s_1^{\text{tmp}}, 1, s_2^{\text{tmp}})$;
  `Move`$(s^+, b_2 - k_2, s_1^{\text{tmp}})$;
  `Move`$(s_2^{\text{tmp}}, 1, s^+)$;
  // `I2:` $b_1 + b_2 + 3$ `moves`

**case I3:** $a < b_2$ & $|\{s \neq s^+ : h(s) < H\}| = 1$
  Find $s'$ s.t. $s' \neq s^+$ & $h(s') < H$;
  $s^{\text{tmp}} :=$ `InterimFull`$(\mathbb{S} \setminus \{s^+, s'\})$;
  `BiSender`$(s^{\text{tmp}}, 1, s^+, b_1, \{s'\})$;
  `Move`$(s^+, 1, s^{\text{tmp}})$;
  `Move`$(s^+, b_2, s')$;
  `Move`$(s^{\text{tmp}}, 1, s^+)$;
  // `I3:` $b_1 + b_2 + 3$ `moves`
  // `(regression issue not yet included)`
  Resolve regression issue (cf. Section 6.4.4);

In case I1, a new function $\alpha(s^+, s)$ is defined similar to $a$, representing the number of empty slots in $\mathbb{S} \setminus \{s^+\}$ with the exclusion of the highest non-full stack, after the top blocking container of stack $s^+$ is moved to stack $s$. That is, when the blocking containers above the target container are being relocated, the detection of enough slots is performed, so as to prevent unnecessary additional move.

Note that in case I2, the target container $c^*$ is moved to the new interim stack $s_2^{\text{tmp}}$ from the first interim stack $s_1^{\text{tmp}}$ when there is only one empty slot remaining in $\mathbb{S} \setminus \{s^+, s_1^{\text{tmp}}\}$. This can

be modified that $c^*$ can be moved to a new interim stack $s_2^{\text{tmp}}$ earlier as long as the empty slots in $\mathbb{S} \setminus \{s^+, s_2^{\text{tmp}}\}$ are enough for the remaining blocking containers in stack $s^+$. Moreover, if there exists a full stack in $\mathbb{S} \setminus \{s^+\}$ in case I2, the task can also be completed in the similar way like case I3, without raising operational cost.

### 6.4.3. *External task*

For an external task, the number of empty slots in $\mathbb{S} \setminus \{s^+, s^-\}$ is denoted by $a$; that is, $a = E - e(s^+) - e(s^-)$. Let $b_1$ and $b_2$ denote the numbers of blocking containers above the target container $c^*$ and the aim slot $(s^-, f(s^-))$, respectively. The pseudo-code of the STAP for an external task is given in Algorithm 4, including four situations:

- E1: $a \geq b_1 + b_2$;
- E2: $b_1 + 1 \leq a < b_1 + b_2$;
- E3: $1 \leq a < b_1 + \min\{1, b_2\}$; and
- E4: $a = 0 < b_1 + b_2$.

**Algorithm 4:** Speedy task accomplishment procedure: External task.

**target container**: $c^*(s^+, t^+)$
**aim slot**: $(s^-, f(s^-) + 1)$
**blocking above target**: $b_1 = h(s^+) - t^+$
**blocking in aim stack**: $b_2 = h(s^-) - f(s^-)$
**slot supply**: $a = E - e(s^+) - e(s^-)$

**case E1:** $a \geq b_1 + b_2$
  `BiSender`$(s^+, b_1, s^-, b_2, \mathbb{S} \setminus \{s^+, s^-\})$;
  `Move`$(s^+, 1, s^-)$;
  // `E1:` $b_1 + b_2 + 1$ `moves`

**case E2:** $b_1 + 1 \leq a < b_1 + b_2$
  $k_2 \coloneqq a - 1 - b_1$;
  `BiSender`$(s^+, b_1, s^-, k_2, \mathbb{S} \setminus \{s^+, s^-\})$;
  Find $s^{\text{tmp}}$ s.t. $s^{\text{tmp}} \notin \{s^+, s^-\}$ & $h(s^{\text{tmp}}) < H$;
  `Move`$(s^+, 1, s^{\text{tmp}})$;
  `Move`$(s^-, b_2 - k_2, s^+)$;
  `Move`$(s^{\text{tmp}}, 1, s^-)$;
  // `E2:` $b_1 + b_2 + 2$ `moves`

**case E3:** $1 \leq a < b_1 + \min\{1, b_2\}$
  $k_1 \coloneqq a - 1$;
  `BiReceiver`$(s^+, k_1, \mathbb{S} \setminus \{s^+, s^-\}, b_1 - k_1, \{s^-\})$;
  Find $s^{\text{tmp}}$ s.t. $s^{\text{tmp}} \notin \{s^+, s^-\}$ & $h(s^{\text{tmp}}) < H$;
  `Move`$(s^+, 1, s^{\text{tmp}})$;
  `Move`$(s^-, b_1 - k_1 + b_2, s^+)$;
  `Move`$(s^{\text{tmp}}, 1, s^-)$;
  // `E3:` $2b_1 + b_2 - a + 3$ `moves`

**case E4:** $a = 0 < b_1 + b_2$
  $s^{\text{tmp}} \coloneqq$ `InterimFull`$(\mathbb{S} \setminus \{s^+, s^-\})$;
  `BiSender`$(s^{\text{tmp}}, 1, s^+, b_1, \{s^-\})$;
  `Move`$(s^+, 1, s^{\text{tmp}})$;
  `Move`$(s^-, b_1 + b_2 + 1, s^+)$;
  `Move`$(s^{\text{tmp}}, 1, s^-)$;
  // `E4:` $2b_1 + b_2 + 4$ `moves`
  // `(regression issue not yet included)`
  Resolve regression issue (cf. Section 6.4.4);

*6.4.4. Regression issue*

In cases I3 and E4, if the top container of the full stack selected (cf. line 34 in Algorithm 3 and line 19 in Algorithm 4) is not fixed, the task is done without any side effect. However, if the top container has been fixed, the container should be put back to its fixed slot, requiring additional moves.

The regression issue only occurs in tricky states, and it can be avoided if the tricky state avoidance technique (cf. Section 6.2) is applied.

## 7. Greedy and speedy heuristic

The proposed greedy and speedy heuristic (GASH) is a realized implementation of the generic feasibility-based heuristic scheme. Starting with the initial state, the next task is selected based on a list of evaluation factors.

The GASH employs all the techniques introduced above, i.e., the container stability concept, the tricky state avoidance and bottom tiers protection techniques, and the STAP. The pseudo-code of the GASH together with the user-defined functions for the STAP are given in Algorithm 5.

Function `ValidTasks()` returns $\mathbb{T}$, the set of valid tasks for the current state. Every task $(c, s) \in \mathbb{T}$ is then evaluated by function `EvalTask`$(c, s)$. The evaluation of a container–stack pair $(c, s)$ is a six-tuple, that starts with the key indicator, followed by the actual number of moves required by the STAP, the stable containers that need to be moved in the aim stack, the affected demand, fixed height of the aim stack, and the opposite value to the target container's priority value. At every step of the GASH, the next task is determined by the lexicographically minimum evaluation six-tuple.

The user-defined functions required in the STAP are also specified. Denoting the blocking container to be moved by $c$ and the potential destination stack by $s^{\text{dst}}$, the preferences of the potential move is as follows, where $m(s) = \max_{g(s)<t\le h(s)} p(s,t)$ is the *messiness* of stack $s$.

1. If stack $s^{\text{dst}}$ is entirely stable and can stabilize $c$, the minimum affected demand is preferred;
2. If stack $s^{\text{dst}}$ is not entirely stable and $p(c) \ge m(s^{\text{dst}})$, the minimum gap between $m(s^{\text{dst}})$ and $p(c)$ is preferred;

**Algorithm 5:** Greedy and speedy heuristic.

```
1  begin
2      (L, f) := (L^0, U1);
3      repeat N − S U times do
4          T := ValidTasks();
5          (c*, s⁻) := arg min_{(c,s)∈T} EvalTask(c, s);
6          Accomplish (c*, s⁻) by the STAP;
7          f(s⁻) := f(s⁻) + 1;
8  function ValidTasks()
9      T := ∅;
10     foreach (c, s) ∈ C × S do
11         if c is unfixed
12         & f(s) < H
13         & p(c) ≤ q^f(s)
14         & Δ(φ) ≥ H − f(s), for p(c) < φ ≤ q^f(s) then
15             T := T ∪ {(c, s)};
16     return T;
17 function EvalTask(c, s)
18     p := p(c);
19     q := q^f(s, f(s));
20     χ := 0 ; // key indicator
21     if f(s) < 2 & δ(p, q) ≥ S then
           // bottom tiers protection
22         χ := ∞;
23     if (L, f) is pre-tricky & f(s) = H − 1 then
           // tricky state avoidance
24         χ := ∞;
25     n := MoveNeed(c, s);
26     return ⟨χ, n, g(s) − f(s), δ(p, q), f(s), −p(c)⟩;
27 function EvalMove(s^src, s^dst)
28     c := (s^src, h(s^src));
29     p := p(c);
30     q := q(s^dst, h(s^dst));
31     case g(s^dst) = h(s^dst) & stack s^dst can stabilize c
32         return ⟨1, δ(p, q)⟩;
33     case g(s^dst) < h(s^dst) & p ≥ m(s^dst)
34         return ⟨2, p − m(s^dst)⟩;
35     case g(s^dst) < h(s^dst) & p < m(s^dst)
36         return ⟨3, m(s^dst) − p⟩;
37     case g(s^dst) = h(s^dst) & stack s_2 cannot stabilize c
38         return ⟨4, q⟩;
39 function Interim(I)
40     I_1 := {s ∈ I : g(s) < h(s) < H};
41     I_2 := {s ∈ I : g(s) = h(s) < H};
42     if I_1 ≠ ∅ then
43         return arg max_{s∈I_1} m(s);
44     else
45         return arg min_{s∈I_2} q(s, h(s));
46 function InterimFull(F)
47     F_1 := {s ∈ F : f(s) ≤ g(s) < h(s) = H};
48     F_2 := {s ∈ F : f(s) < g(s) = h(s) = H};
49     F_3 := {s ∈ F : f(s) = g(s) = h(s) = H};
50     if F_1 ≠ ∅ then
51         return arg min_{s∈F_1} p(s, h(s));
52     else if F_2 ≠ ∅ then
53         return arg min_{s∈F_2} p(s, h(s));
54     else  // cause regression issue
55         return arg min_{s∈F_3} p(s, h(s));
```

3. If stack $s^{\text{dst}}$ is not entirely stable and $p(c) > m(s^{\text{dst}})$, the minimum gap between $p(c)$ and $m(s^{\text{dst}})$ is preferred;

4. If stack $s^{\text{dst}}$ is entirely stable but cannot stabilize $c$, the minimum $q(s^{\text{dst}}, h(s^{\text{dst}}))$ is preferred.

The first preference indicates that stabilizing a blocking container reduces the total number of unstable containers in the bay. The second and third preferences ~~take~~ consider the messiness of the destination stack ~~into consideration~~. ~~The~~ M messiness refers to the largest priority value of the unstable containers in a stack, with larger messiness representing a higher urgency of reshuffling. The last preference indicates that an entirely stable stack should be protected from being ruined.

In cases I1 and I2, an interim stack is selected ~~for~~ temporarily storing the target container. The selection prefers stacks that are not entirely stable with the largest messiness, then entirely stable stacks with the smallest capability. ~~That is,~~ the most unattractive stack for receiving blocking containers is selected as the interim stack. In cases I3 and E4, the interim stack is selected by the minimum priority value of the top containers of these full stacks.

Note that ~~since~~ the tricky state avoidance technique is applied, the regression issue would never happen in the GASH, ~~though~~ it is still considered in function `InterimFull(`$\mathbb{F}$`)` to complete the whole logic.

## 8. Computational experiments

To show the performance of the GASH, we compare it to two LPF heuristics. The first one is the *target-guided heuristic* (TGH) [15] and the ~~other~~ is a *largest priority (value) first heuristic with (post-task) filling* (LPFF).

The pseudo-code of the LPFF is given in Algorithm 6. The LPFF selects the target container from unfixed containers with the largest priority value, and then accomplishes it by the STAP; the user-defined functions used here are the same as ~~that~~ in the GASH. After ~~the~~ completion ~~of~~ the current task, the filling process is carried out. ~~That is,~~ among those stacks whose top containers are unstable and can become stable if ~~being~~ moved to the aim stack. If multiple filling choices exist, the container with the largest priority value is chosen. The idea of the LPFF is similar to the randomized heuristic proposed by Expósito-Izquierdo et al. [6]. However, because their heuristic only discusses cases with enough empty slots (cases I1 and E1 in our statement), it is difficult to implement a deterministic version of their heuristic.

The problem instance format of the CPMP is the same as that of the *container relocation problem* (CRP, a.k.a. the blocks relocation problem), the problem of minimizing the total number of container moves for retrieving containers from the initial layout. In the ~~recent~~ literature, researchers usually use the same data sets when solving the CPMP [1, 3, 6, 15] and the CRP [4, 5, 7, 9, 10].

Caserta et al. [4] present the complete CVS ~~date~~ set (named after the authors' surnames, Caserta, Voß ~~&~~ Sniedovich) originally for the CRP. The CVS instances are classified into 21

**Algorithm 6:** Largest priority first heuristic with filling.

```
1  begin
2      (L, f) := (L^0, U1);
3      repeat N − S U times do
4          T := LargestPriorityTasks();
5          (c*, s⁻) := arg min_{(c,s)∈T} MoveNeed(c, s);
6          Accomplish (c*, s⁻) by the STAP;
7          f(s⁻) := f(s⁻) + 1;
8          Fill(s⁻);
9  function LargestPriorityTasks()
10     T := ∅;
11     p^max := max{p(c) : c is unfixed};
12     foreach (c, s) ∈ C × S do
13         if c is unfixed & p(c) = p^max & f(s) < H then
14             T := T ∪ {(c, s)};
15     return T;
16 function Fill(s⁻)
17     while h(s⁻) < H do
18         S' := ∅;
19         for s ∈ {s ≠ s⁻ : g(s) < h(s)} do
20             c := (s, h(s));
21             if stack s⁻ can stabilize c then
22                 S' := S' ∪ {s};
23         if S' ≠ ∅ then
24             s' := arg max_{s∈S'} p(s, h(s));
25             Move(s', 1, s⁻);
26         else
27             break while;
```

groups, each consisting of 40 instances. The stacks of the initial layout ~~are with~~ have the same height in a CVS instance. The number of containers per stack in the initial layout is denoted by $K$, hence $N = S\,K$. It is worth noting that the stack height limitation is not specified in the original data. The ~~recent~~ researchers add two extra tiers above the initial layout; that is $H = K + 2$.

The CVS instances can be considered typical dense CPMP instances. Table 2 illustrates the computational results on CVS instances by the GASH, the TGH and the LPFF. The values under ~~headings~~ the "move" represent average numbers of moves for every CVS group, ~~while~~ whereas the values under ~~headings~~ the "gap%" are the ratios of the difference between each LPF heuristic and the GASH to the number of containers $N$. The results showcase that the GASH leads the LPF heuristics by

more than $10\% \times N$ moves on most instance groups. Especially on the narrowest group CVS 10-6, the outperformance of the GASH against the TGH and the LPFF reaches 319.38% and 55.54%, respectively.

Table 2: Computational results: CVS instances.

| CVS *K*-*S* | GASH move | TGH | | LPFF | |
|---|---|---|---|---|---|
| | | move | gap% | move | gap% |
| CVS 3-3 | 11.28 | 12.95 | 18.61 | 11.25 | −0.28 |
| CVS 3-4 | 10.80 | 12.18 | 11.46 | 12.23 | 11.88 |
| CVS 3-5 | 12.08 | 12.78 | 4.67 | 13.45 | 9.17 |
| CVS 3-6 | 12.98 | 14.38 | 7.78 | 14.88 | 10.56 |
| CVS 3-7 | 14.75 | 16.00 | 5.95 | 16.58 | 8.69 |
| CVS 3-8 | 15.65 | 16.55 | 3.75 | 17.08 | 5.94 |
| CVS 4-4 | 21.88 | 23.35 | 9.22 | 21.93 | 0.31 |
| CVS 4-5 | 23.08 | 26.73 | 18.25 | 26.48 | 17.00 |
| CVS 4-6 | 24.75 | 27.58 | 11.77 | 27.20 | 10.21 |
| CVS 4-7 | 27.63 | 29.93 | 8.21 | 31.23 | 12.86 |
| CVS 5-4 | 35.08 | 44.83 | 48.75 | 35.83 | 3.75 |
| CVS 5-5 | 35.33 | 42.40 | 28.30 | 36.50 | 4.70 |
| CVS 5-6 | 39.88 | 50.63 | 35.83 | 43.08 | 10.67 |
| CVS 5-7 | 41.68 | 48.83 | 20.43 | 46.95 | 15.07 |
| CVS 5-8 | 47.50 | 56.68 | 22.94 | 51.83 | 10.81 |
| CVS 5-9 | 50.45 | 57.50 | 15.67 | 55.65 | 11.56 |
| CVS 5-10 | 54.63 | 62.80 | 16.35 | 60.88 | 12.50 |
| CVS 6-6 | 55.23 | 74.33 | 53.06 | 57.85 | 7.29 |
| CVS 6-10 | 75.60 | 88.63 | 21.71 | 79.73 | 6.88 |
| CVS 10-6 | 140.63 | 332.25 | 319.38 | 173.95 | 55.54 |
| CVS 10-10 | 179.23 | 302.90 | 123.68 | 190.50 | 11.28 |
| Average | 44.29 | 64.48 | 38.37 | 48.81 | 11.26 |

Bortfeldt & Forster [1] introduce 32 groups of CPMP instances (referred to as BF instances), each group consisting of 20 instances. In BF instances, the bay size is $S = 16$ or 20 and $H = 5$ or 8. The number of containers $N$ is either $0.6 \times SH$ or $0.8 \times SH$, the number of priorities $P$ is either $0.2 \times N$ or $0.4 \times N$, and the number of disorderly containers $B$ is either $0.6 \times N$ or $0.75 \times N$ in the initial layout.

The BF instances can be considered typical loose CPMP instances. Table 3 illustrates the computational results on BF instances by the three heuristics. The values under headings "ratio%" are the relative gap between per heuristic and the number of disorderly containers initially $B$, and the values under headings "gap%" are the differences between the "ratio" columns of the GASH

and each LPF heuristic.

Table 3: Computational results: BF instances.

| BF | $S$ | $H$ | $N$ | $P$ | $B$ | GASH | | TGH | | | LPFF | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | move | ratio% | move | ratio% | gap% | move | ratio% | gap% |
| 1 | 16 | 5 | 48 | 10 | 29 | 29.15 | 0.52 | 29.10 | 0.34 | −0.17 | 31.20 | 7.59 | 7.07 |
| 2 | 16 | 5 | 48 | 10 | 36 | 36.00 | 0.00 | 36.00 | 0.00 | 0.00 | 37.80 | 5.00 | 5.00 |
| 3 | 16 | 5 | 48 | 20 | 29 | 29.35 | 1.21 | 29.45 | 1.55 | 0.34 | 31.20 | 7.59 | 6.38 |
| 4 | 16 | 5 | 48 | 20 | 36 | 36.15 | 0.42 | 36.00 | 0.00 | −0.42 | 37.45 | 4.03 | 3.61 |
| 5 | 16 | 5 | 64 | 13 | 39 | 46.30 | 18.72 | 48.50 | 24.36 | 5.64 | 54.20 | 38.97 | 20.26 |
| 6 | 16 | 5 | 64 | 13 | 48 | 55.50 | 15.63 | 57.55 | 19.90 | 4.27 | 65.15 | 35.73 | 20.10 |
| 7 | 16 | 5 | 64 | 26 | 39 | 49.95 | 28.08 | 53.55 | 37.31 | 9.23 | 57.35 | 47.05 | 18.97 |
| 8 | 16 | 5 | 64 | 26 | 48 | 57.60 | 20.00 | 60.00 | 25.00 | 5.00 | 67.30 | 40.21 | 20.21 |
| 9 | 16 | 8 | 77 | 16 | 47 | 56.30 | 19.79 | 60.35 | 28.40 | 8.62 | 60.50 | 28.72 | 8.94 |
| 10 | 16 | 8 | 77 | 16 | 58 | 61.55 | 6.12 | 62.15 | 7.16 | 1.03 | 67.50 | 16.38 | 10.26 |
| 11 | 16 | 8 | 77 | 31 | 47 | 55.00 | 17.02 | 61.25 | 30.32 | 13.30 | 61.65 | 31.17 | 14.15 |
| 12 | 16 | 8 | 77 | 31 | 58 | 61.45 | 5.95 | 63.45 | 9.40 | 3.45 | 68.25 | 17.67 | 11.72 |
| 13 | 16 | 8 | 103 | 21 | 62 | 96.50 | 55.65 | 107.45 | 73.31 | 17.66 | 104.45 | 68.47 | 12.82 |
| 14 | 16 | 8 | 103 | 21 | 78 | 116.05 | 48.78 | 124.75 | 59.94 | 11.15 | 124.55 | 59.68 | 10.90 |
| 15 | 16 | 8 | 103 | 42 | 62 | 99.45 | 60.40 | 110.60 | 78.39 | 17.98 | 107.65 | 73.63 | 13.23 |
| 16 | 16 | 8 | 103 | 42 | 78 | 115.45 | 48.01 | 133.35 | 70.96 | 22.95 | 131.40 | 68.46 | 20.45 |
| 17 | 20 | 5 | 60 | 12 | 36 | 36.50 | 1.39 | 36.50 | 1.39 | 0.00 | 38.50 | 6.94 | 5.56 |
| 18 | 20 | 5 | 60 | 12 | 45 | 45.20 | 0.44 | 45.00 | 0.00 | −0.44 | 46.00 | 2.22 | 1.78 |
| 19 | 20 | 5 | 60 | 24 | 36 | 36.75 | 2.08 | 36.80 | 2.22 | 0.14 | 39.30 | 9.17 | 7.08 |
| 20 | 20 | 5 | 60 | 24 | 45 | 45.10 | 0.22 | 45.00 | 0.00 | −0.22 | 46.15 | 2.56 | 2.33 |
| 21 | 20 | 5 | 80 | 16 | 48 | 56.55 | 17.81 | 61.65 | 28.44 | 10.63 | 64.95 | 35.31 | 17.50 |
| 22 | 20 | 5 | 80 | 16 | 60 | 65.55 | 9.25 | 67.90 | 13.17 | 3.92 | 77.50 | 29.17 | 19.92 |
| 23 | 20 | 5 | 80 | 32 | 48 | 55.25 | 15.10 | 61.10 | 27.29 | 12.19 | 66.10 | 37.71 | 22.60 |
| 24 | 20 | 5 | 80 | 32 | 60 | 68.00 | 13.33 | 70.95 | 18.25 | 4.92 | 78.60 | 31.00 | 17.67 |
| 25 | 20 | 8 | 96 | 20 | 58 | 66.00 | 13.79 | 69.80 | 20.34 | 6.55 | 72.80 | 25.52 | 11.72 |
| 26 | 20 | 8 | 96 | 20 | 72 | 75.75 | 5.21 | 74.35 | 3.26 | −1.94 | 80.40 | 11.67 | 6.46 |
| 27 | 20 | 8 | 96 | 39 | 58 | 65.65 | 13.19 | 71.85 | 23.88 | 10.69 | 72.80 | 25.52 | 12.33 |
| 28 | 20 | 8 | 96 | 39 | 72 | 76.50 | 6.25 | 76.30 | 5.97 | −0.28 | 82.55 | 14.65 | 8.40 |
| 29 | 20 | 8 | 128 | 26 | 77 | 115.85 | 50.45 | 118.65 | 54.09 | 3.64 | 122.75 | 59.42 | 8.96 |
| 30 | 20 | 8 | 128 | 26 | 96 | 129.60 | 35.00 | 143.05 | 49.01 | 14.01 | 149.25 | 55.47 | 20.47 |
| 31 | 20 | 8 | 128 | 52 | 77 | 115.85 | 50.45 | 128.15 | 66.43 | 15.97 | 128.00 | 66.23 | 15.78 |
| 32 | 20 | 8 | 128 | 52 | 96 | 134.10 | 39.69 | 147.30 | 53.44 | 13.75 | 149.35 | 55.57 | 15.89 |
| Average | | | | | | 68.44 | 19.37 | 72.75 | 26.05 | 6.67 | 75.71 | 31.83 | 12.45 |

The relative gap between the solution value and the number of disorderly containers in the initial layout is a good measure of the heuristic performance. Table 4 shows that the instance density $N/SH$ (or bay utilization) and the height of the bay $H$ are key factors ~~to~~ in the number of moves needed for pre-marshalling. In other words, denser or higher instances are more difficult to solve. The computational results on BF instances also prove that the GASH outperforms these

two LPF heuristics.

Table 4: Summary on BF instances.

| | GASH ratio% | | TGH gap% | | LPFF gap% | |
|---|---|---|---|---|---|---|
| density | $H = 5$ | 8 | 5 | 8 | 5 | 8 |
| 0.6 | 0.78 | 10.91 | −0.10 | 5.18 | 4.85 | 10.50 |
| 0.8 | 17.24 | 48.55 | 6.97 | 14.64 | 19.65 | 14.81 |

## 9. Conclusions

In this paper we present a generic heuristic scheme based on a new concept of state feasibility for solving the container pre-marshalling problem. The proposed heuristic scheme is an innovative departure ~~which breaks out of~~ from the traditional largest priority (value) first (LPF) strategy. Furthermore, a realized instance of the heuristic scheme, the greedy and speedy heuristic is also proposed. We analyze the computational results on two commonly used data sets, ~~with comparison to two LPF heuristics~~ and compare the proposed scheme with two ~~other~~ LPF heuristics.

The proposed feasibility-based heuristic scheme brings a high degree of freedom to the methodology of heuristics. In the future, feasibility-based meta-heuristics will be developed.

## References


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

[2] Carlo, H. J., Vis, I. F. A., & Roodbergen, K. J. (2014). Storage yard operations in container terminals: Literature overview, trends, and research directions. *European Journal of Operational Research*, 235(2), 412–430. Maritime Logistics.

[3] Caserta, M. & Voß, S. (2009). A corridor method-based algorithm for the pre-marshalling problem. In M. Giacobini, A. Brabazon, S. Cagnoni, G. A. Di Caro, A. Ekárt, A. I. Esparcia-Alcázar, M. Farooq, A. Fink, & P. Machado (Eds.), *Applications of Evolutionary Computing*, vol. 5484 of *Lecture Notes in Computer Science* (pp. 788–797). Springer Berlin Heidelberg.

[4] Caserta, M., Voß, S., & Sniedovich, M. (2011). Applying the corridor method to a blocks relocation problem. *OR Spectrum*, 33(4), 915–929.

[5] Expósito-Izquierdo, C., Melián-Batista, B., & Moreno-Vega, J. M. (2014). A domain-specific knowledge-based heuristic for the blocks relocation problem. *Advanced Engineering Informatics*, 28(4), 327–343.

[6] Expósito-Izquierdo, C., Melián-Batista, B., & Moreno-Vega, M. (2012). Pre-marshalling problem: Heuristic solution method and instances generator. *Expert Systems with Applications*, 39(9), 8337–8349.

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

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

[9] Jin, B., Lim, A., & Zhu, W. (2013). A greedy look-ahead heuristic for the container relocation problem. In M. Ali, T. Bosse, K. V. Hindriks, M. Hoogendoorn, C. M. Jonker, & J. Treur (Eds.), *Recent Trends in Applied Artificial Intelligence*, vol. 7906 of *Lecture Notes in Computer Science* (pp. 181–190). Springer Berlin Heidelberg.

[10] Jin, B., Zhu, W., & Lim, A. (2015). Solving the container relocation problem by an improved greedy look-ahead heuristic. *European Journal of Operational Research*, 240(3), 837–847.

[11] Kim, K. H. & Lee, H. (2015). Container terminal operation: Current trends and future challenges. In C.-Y. Lee & Q. Meng (Eds.), *Handbook of Ocean Container Transport Logistics*, vol. 220 of *International Series in Operations Research & Management Science* (pp. 43–73). Springer International Publishing.

[12] Lee, Y. & Chao, S.-L. (2009). A neighborhood search heuristic for pre-marshalling export containers. *European Journal of Operational Research*, 196(2), 468–475.

[13] Lee, Y. & Hsu, N.-Y. (2007). An optimization model for the container pre-marshalling problem. *Computers & Operations Research*, 34(11), 3295–3313.

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

[15] Wang, N., Jin, B., & Lim, A. Target-guided algorithms for the container pre-marshalling problem. *Omega*. `http://dx.doi.org/10.1016/j.omega.2014.12.002`.