

Manuscript Number: EJOR-D-15-00397R1

Title: A feasibility-based heuristic for the container pre-marshalling problem

Article Type: Innovative Application of OR

Section/Category: (O) Heuristics

Keywords: container pre-marshalling problem; feasibility-based heuristic

Corresponding Author: Dr. Ning Wang,

Corresponding Author's Institution: Shanghai University

First Author: Bo Jin

Order of Authors: Bo Jin; Andrew Lim; Ning Wang; Zizhen Zhang

Manuscript Region of Origin:

Abstract: This paper addresses the container pre-marshalling problem (CPMP) which rearranges containers inside a storage bay to a desired order.

By far, target-driven algorithms have relatively good performance among all algorithms; they have two key components: 1) containers are rearranged to their desired slots one by one in a certain order; and 2) rearranging one container is completed by a sequence of movements. Our algorithm improves the performance from the aforementioned two components.

Instead of rearranging containers in the descending order of their group labels (traditional idea), this paper determines the order as the algorithm goes on by an evaluation function. To efficiently explore the enlarged search space, we are the first one to come up with a concept of state feasibility to cut branches.

In addition, we improve the sequence of movements by techniques such as pre-extreme avoidance and tier protection.

Computational experiments showcase that the performance of our proposed heuristic is better than other heuristics.

Dear Editor and Reviewers,

We would like to express our gratitude for the great efforts that you and the anonymous referees have put in this paper. We found most of the critiques and feedbacks very constructive. We have strived to address all the issues as thoroughly as possible. Consequently, we have rewritten many sections of the manuscript. The main changes are summarized as follows:

1. We change the title of our paper to "A feasibility-based heuristic for the container pre-marshalling problem". Words "generic" and "scheme" are removed. As presented in point 4, the original idea is to separate framework and techniques, so that future scholars can take any part as they want. But we find it's hard to follow, hence Section 5-7 are compressed into one Section, which explains the heuristic as well as associate techniques. "Generic" and "scheme" now are useless.
2. We have consolidated Section 3 and 4 into Section 3. The new Section 3 defines the problem (in Section 3 of the old version) and gives problem-related notations (in Section 4 of the old version). Solution-related notations (in Section 4 of the old version) are moved to the algorithm and will be introduced when used.
3. We add a new Section 4 to introduce state, concepts and properties of states. "State" is an important concept for the algorithm development; in the meanwhile, the analysis of state properties is useful for further study on algorithms. Hence, we write the contents in an independent section.
4. We compress Section 5-7 into Section 5. The initial thought for us is to propose a framework and several techniques. To introduce them separately is to emphasize that every part can be used solely. Other scholars can use the framework combined with other techniques; the proposed techniques can also be used in other frameworks. Hence, Section 5 explains framework, Section 6 explains techniques and Section 7 illustrates the assembled heuristic. However, we find such a presentation is too hard to follow. Now we just explain the assembled heuristic, and the details of the framework and techniques are introduced after the heuristic in the same section (Section 5). Useful parts will still be adopted if some reader thinks that they are effective in the future research.
5. Most of the presentation in this paper is rewritten. Figures and examples are added for better understanding. For new ideas such as the tier-protection indicator and task types, we add some explanations to reveal the thinking process.

We hope our revision meets your satisfaction, and we look forward to your favorable response.

Regards,
Bo Jin

Andrew Lim
Ning Wang
Zizhen Zhang

Reviewer1

1. The authors need to clearly demonstrate adequate understanding of the relevant literature. The authors need to include a good literature survey to show exactly what is novel about their approach. In this regard, I would like a clear discussion on the current literature versus the contributions of the paper. Now, only a few references are considered in the literature review.

To the best of our knowledge, only seven works discuss the pre-marshalling problem. These seven works have been discussed in our revised version (Section 2). The contributions (innovations) have been listed in the last but one paragraph of Section 1: This paper makes two main contributions to the literature. One contribution is the idea behind the proposed heuristic. In the traditional algorithms, containers are allocated according to a certain order, such as the descending order of their groups [8]; in this research, the order to allocate containers is not fixed beforehand, but rather that it is decided as the algorithm goes on. This innovation undoubtedly enlarges the search space and decreases the search efficiency. To avoid efficiency decrease, we propose the concept of state feasibility to check the feasibility of allocating a certain container before we actually allocate it. The time complexity of check feasibility is only $O(G)$, here G is the number of groups. Our paper is the first work that uses feasibility to cut branches in CPMP algorithms. Another contribution is that the proposed heuristic performs outstandingly on both dense and loose instances, compared with existing heuristics.

2. In the conclusion section, the authors need to clearly discuss their theoretical contributions in Operations Research compared to those in related papers found in the literature. Additionally, the authors need to discuss and supply at least 4-5 solid and insightful future research directions for the operations research community.

The conclusion has been revised, and we have discussed our contributions as well as the future research road.

3. I would like authors to add a paragraph in the introduction discussing the optimization problem from an "operations research" point of view. The introduction does not clearly place the paper in that area.

We have revised the introduction session. The formal definition of the problem talked about is given in the last paragraph of Page 2, which infers that the problem is a typical optimization problem.

4. After reading the paragraph dedicated to describe the contributions of the paper in the

introduction, I think that most of the readers could not to obtain a clear idea of what the paper proposes. Specifically, only a set of novel terms are provided with no context in the research field. Consequently, the contributions must be clearly introduced.

The new contributions are illustrated in the last but one paragraph of Section 1 as well as the answer to Question 1.

5. It would be desirable to provide some including remarks to further discuss the proposed methods, for example, what are the main advantages and limitations in comparison with existing methods?

We have added our innovation in contribution part of Section 1 and comparison with other algorithms in Section 5.

Briefly speaking, our algorithm is a kind of target-driven algorithm. This kind of algorithm has been talked about in some extant works. Target-driven algorithm rearranges a certain container to its final slot by a series of movements, and the algorithm does this until all containers are rearranged. By far, target-driven is effective compared with other kinds of algorithm. On top of this, our algorithm optimizes the rearrangement order, and series of movements for rearranging one container. The basic idea is to use state feasibility to cut searching branch. Now our algorithm is the best among target-driven heuristics.

6. The abstract in current form is superficial. Specifically, the abstract must be rewritten to point out significance and impact of the paper for EJOR. For instance, check the following phrases: 'We propose a generic feasibility-based heuristic scheme for the CPMP, which is a departure from the traditional heuristic strategy. An implementation of the heuristic scheme, the greedy and speedy heuristic, is also proposed, in which four well designed heuristic techniques are applied.' I wonder why the scheme is generic? are other proposals that consider infeasibility? why is a departure from 'traditional heuristic strategy'?, etc. Answers for these and similar questions are not provided in the remainder of the paper.

We have reorganized the abstract. The key ideas of our algorithm are emphasized and explained.

The 'generic' selling point in the old version is to emphasize that our framework and techniques for container rearrangement can be separately used by other scholars. But now we do not emphasize their independence, so we remove the word "generic". Other proposals have never used feasibility to cut branches before. "Departure from traditional heuristic strategy" means we use feasibility which is different from others. The presentation makes the reviewers confusing, we have revised.

7. Moreover, the paper does not clearly communicate neither what has been done nor

what is its significance. It is extremely hard to follow the description of the heuristic scheme without providing illustrative examples and a suitable introduction of the notations used along the whole paper. In this regard, the authors must carry out a great effort in communicating their proposal clearly. I would say that in general it lacks many explanations and contextualization of the scheme.

What has been done: we propose a heuristic better than other heuristic and the significance lies in the idea behind the heuristic and the details such as state feasibility and tier-protection. We have stated this in Section 1 and 5.

We have rewritten the heuristic. We first add Section 4 to explain necessary concepts such as state feasibility, stability, etc. Then we consolidate Section 5-7 in the old version into one section (Section 5). The new section emphasizes the target-driven heuristic, which is composed of task selection and speedy task accomplishment procedure. The details of the two components are also explained in Section 5.2-5.4. Necessary examples and figures are added for better understanding.

8. I do not like the order of sections 3 and 4. Introducing notations concerning a problem that is not yet presented is very confusing. The reader must perform an effort to devise concepts such as 'order-p demand', 'interim stack', and so forth.

We remove Section 4 (Notations). For problem-related notations, we introduce them when introducing the problem in Section 3. For notations used for problem solving, we explain them when they are used.

Reviewer2

1. p. 7, l. 3 The authors should explain in greater detail what unreachable tiers are.
We have explained unreachable tiers in Page 11. Unreachable tiers are tiers that any movement sequence cannot reach. For a stack s , only the top $\sum_{i \neq s} e(i)$ containers can be moved to other stacks. The immovable number of containers (unreachable tiers) for any stack is $U = H - \sum_{i=1}^S e(i) = H - E$.
2. p. 8, l. 36, p. 9, l. 4-6 Before giving a proposition on necessary conditions for state feasibility the authors should define the term "feasible state". The notion "feasibility of stack(s)" must also be defined. The first sentence on p. 9 reads as an unproven claim. We add the definition of feasible state in Definition 3. A state (L, f) is feasible if there exists a movement sequence that can convert L to an orderly layout without moving fixed containers indicated by f .

"feasibility of stack(s)" is a typo, and it should be "feasibility of state". But the sentence has been removed from the paper.

The first sentence on p. 9 is the conclusion from "Wang, N. (2014). Optimization Study on Container Operations in Maritime Transportation. PhD thesis, City University of Hong Kong, Hong Kong" which gives the proof to this claim. The work can be retrieved from City University Library->E-resources->E-theses->By title, and we are preparing a paper for this claim. In this paper, we do not need sufficient conditions, we only need necessary conditions. Hence, we explain the necessary conditions on Page 7 and 8. The first sentence on p. 9 is removed.
3. p. 9, l. 42 Proposition 2 should be proved and illustrated by an example.
We have added the explanation and examples on Page 10. When $E \geq 2(H - 1)$, it has $N \leq SH - 2H + 2$, even though all containers are fixed, there are at most $S - 2$ stacks fully occupied with fixed containers: 1) if there are fewer than $S-2$ stacks fully occupied, the state is not a dead-end state; 2) if there are $S-2$ stacks fully occupied, there are at most two extra containers in the left two stacks. The state is not a dead-end state, either. For cases with $E < 2(H-1)$, there exist possibilities that the algorithm falls into a dead-end state.
4. p. 10, l. 25 Proposition 3 should be proved and illustrated by an example.
Proposition 3 does not have relationship with the content in this paper, therefore, we remove the proposition.
5. p. 12, l. 21-22 Please improve the formulation "If a container stack pair leads to an extreme state ..." since in my opinion rather the $S-2$ fully fixed stacks lead to an

extreme state.

The meaning of the quoted sentence is “if the resultant state after performing a container-stack pair is an extreme state...”. We rewrite this part on Page 13, and now the sentence is removed.

6. p. 12, l. 29-31 In the definition of a pre-tricky state the last condition (Sum of $f(s) < N - 2$) should be motivated.

We have rewritten Definition 7. In the new definition, we discuss the resultant layouts after performing a task $c \rightarrow s$ with $f(s)=H-1$. The resultant layouts are different when $\sum_{s \in \mathbb{S}} f(s) = N$, $\sum_{s \in \mathbb{S}} f(s) = N - 1$, $\sum_{s \in \mathbb{S}} f(s) = N - 2$, and $\sum_{s \in \mathbb{S}} f(s) < N - 2$. It shows that only when $\sum_{s \in \mathbb{S}} f(s) < N - 2$, the resultant layout has a chance to generate dead-end states.

Moreover, we change the name of “trick state” to “dead-end state”.

7. p. 12, l. 40 ff This paragraph is hard to understand. Maybe an example would help.

We further explain the tier-protection indicator on Page 14 and 15. The tier-protection indicator is to balance the trade-off between the freedom of task selection and the surplus loss. In principle, a container can be moved to any valid stack. But if a container with small group value occupies an (relatively) empty stack, then the loss of the surplus vector is large. After the target container c is fixed to the target slot $(s, f(s)+1)$, the surplus $\Delta(\phi)$ is reduced by $H - f(s)$, for $\phi \in (g(c), g(s, f(s))]$. The larger the spread between $g(c)$ and $g(s, f(s))$ is, the more the surplus vector will lose.

As shown in the following example, if container 1 (container with group value 1) is moved on container 3 (stack 5 tier 3), the supply of group 3 is reduced by 2 as 1 becomes the highest orderly container of stack 5. The cumulative supplies $G(3)$ and $G(2)$ are reduced by 2 accordingly.

	4	6			
	5	6	2	1	
	8	7	4	4	3
	8	10	11	3	5
	12	11	7	10	5
Stack	1	2	3	4	5

8. Numerical experiments The authors should provide runtimes for all compared solution methods and for the CVS instances as well as the BF instances (even if the run times for GASH are very small). In the case of the BF instances they should include the heuristic by Bortfeldt and Forster (2012) in the comparison.

We add the runtimes of the three algorithms for both CVS and BF data sets. The algorithm of BF is a tree search; therefore we don't compare it with our algorithm.

Minors:

1. p. 8: $f \leq 0$ or better $f = 0$?

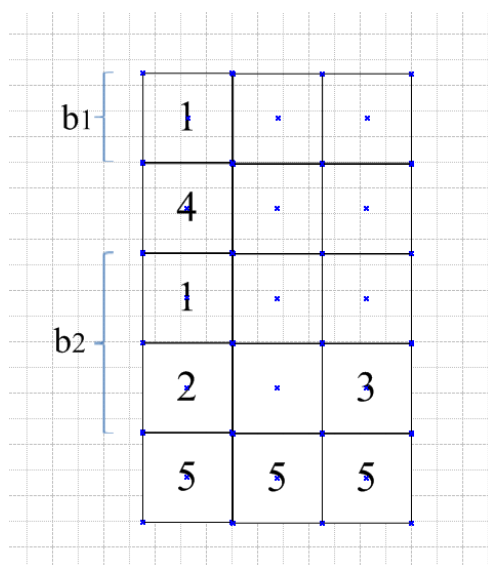
In the old version, $f \leq \mathbf{o}$, here \mathbf{o} is the vector composed by numbers of orderly containers in stacks. In the new version, we have defined that fixed containers are orderly and not allowed to be moved in the future. Hence, $f \leq \mathbf{o}$ is useless and removed.

2. p 10: "solubility" - write "resolvability"

Proposition 3 has been removed since it has no relationship with the main contents of the paper. We have illustrated and explained it in "Wang, N. (2014). Optimization Study on Container Operations in Maritime Transportation. PhD thesis, City University of Hong Kong, Hong Kong" and our another paper.

3. p 15: "blocking containers above and below c^* " - blocking containers below?

We have explained blocking containers on Page 16. Blocking containers refer to those above target containers or target slots. In the internal task, $c^*(s,t)$ is moved to a slot $f(s)+1$ under c^* . Hence, the number of containers above c^* is b_1 , and the number of containers between tier t (excluded) and $f(s)+1$ (included) is b_2 . In the following example, container with group label 4 located at (1,4) needs fixed into stack 1. b_1 and b_2 containers are marked by brackets.



4. p 16: "realized implementation" - "implementation" would suffice.

The new version has removed the statement.

Reviewer3

1. Improve and ease the presentation, especially Section 6. Maybe, including examples and pictures might help in achieving this goal. Similarly, while the precision and rigor of Algorithms 1-6 is appreciated, the author should rethink the way in which the ideas behind some of them (especially Algorithms 4 and 5) is currently presented.

We have moved the contents in Section 6 to Section 5.4. We add examples and figures for bettering understanding, such as the examples for pre-extreme states and tier-protection.

The main content is to accomplish a task based on different types. The algorithms for internal and external types are presented in Algorithm 2 and 7, respectively. In Algorithm 3-6, they present all the functions appeared in Algorithm 2 and 7. As the functions are many, we explain them one by one, especially the motivations and advantages.

2. Proposition 2 at page 9 on the existence of "tricky states": The proposition states that a tricky states exists if the number of empty slots is less than $2(H-1)$. However, at lines 38 and 39, the authors are mentioning that what can cause the existence of a tricky state is the inability to reshuffle unfixed containers. In other words, could it happen that, despite $E < 2(H - 1)$ being true, a tricky states does not exist (for example, in the case in which the number of unfixed containers in the remaining two stacks is less than E)? It seems to me that the existence of a tricky state must depend on the number of unfixed containers in the "extreme" state. Please, clarify this doubt.

We didn't present our idea clearly in the old version. The reviewer's thought is right, the existence of a tricky state (now called dead-end state) depends on f . Proposition 2 wants to justify the possibility of entering dead-end state of a layout before the algorithm actually starts.

We have restated Proposition 2: "No dead-end state can be generated whatever movements and fixed vectors are applied to cases with $E \geq 2(H - 1)$. For cases with $E < 2(H - 1)$, there exist possibilities that the algorithm falls into a dead-end state."

Given a layout L , the algorithm proposed in our paper starts from fixed vector $f=U \cdot 1$, and fixes containers one by one (changes f in this process). For a layout L with $E < 2(H-1)$, it has possibilities that the algorithm encounters a dead-end state; for a layout L with $E \geq 2(H - 1)$, there is no chance for the layout to enter a dead-end state in the process of the algorithm. The example is also given on Page 10.

3. Page 12, lines 49-52: The concept of "bottom tiers protection" is quite interesting. However, one might wonder what is the effect of different choices of parameters value, namely 2 and S . Would computational results and running times change significantly if these two parameters were modified? How?

We have added a subsection (Section 6.1) discussing the experiment on tier-protection indicator. It shows that the tier-protection indicator is effective, as the performance with the indicator is better than that without the indicator (the setting (0,N)). Moreover, the settings with intermediate parameters have the best performance, and tight and loose parameters have worse performance. If the parameters are too tight, the instances even do not have a solution.

4. The authors should present the running times of the proposed algorithm on both benchmark sets (if possible, along with the running time of the two benchmark heuristics).

The running times of the three heuristics (TGH, LGVFH, FBH) are added. Specially, TGH and LGVFH are implemented by us.

5. Please, provide details of the implementation: Machine configuration (speed, ram, OS, etc.), language and compiler used, etc.

We added the machine information in the first paragraph of Section 6 “Experiments were conducted on a PC with Intel Core i7 CPU clocked at 3.40 GHz with Windows 7 operating system. The code was written in Java.”

Minors

1. Page 15, line 6 : The accomplishment for an internal task→The accomplishment of an internal task?
2. Page 15, line 6: “tack” →”task”

We have amended typos of Point 1 and 2.

3. Why is the gap of Tables 2 and 3 measured with respect to the number of containers? Why is this gap preferred over a standard improvement measure (e.g., gap = (gash-literature)/literature). For example, for the first row of table 2, to give a

measure of the goodness of the GASH approach, I would use: $\gamma = \frac{LPVFH - GASH}{LPVFH} =$

$\frac{11.25 - 11.28}{11.25} = -0.003$. In this case, the γ value would indicate the net improvement, in

percentage, (in this specific case, loss) of the proposed method with respect to the best result from the literature.

We have removed the original gap, and compare the performance of our algorithm with TGH and LGVFH. The formulation used is the one pointed out by the reviewer

$\gamma = \frac{TGH - FBH}{LPVFH}$. In the new version, GASH is renamed FBH, shorted for

Feasibility-Based Heuristic.

Reviewer4

1. First, the presentation needs to be enhanced. The current manuscript has many loose ends, making it hard to follow. For example, page 7 stated that "containers in the reachable tiers can be permuted into any wanted arrangement", and in page 10, it reads "it is a natural fact that when $S \geq 3$, the reachable tiers can be permuted into any wanted placement." The condition " $S \geq 3$ " was not mentioned when the property was first stated on page 7. Moreover, this property needs a careful proof, or at least some justification.

In the new version, we have removed the statement "containers in the reachable tiers can be permuted into any wanted arrangement". The fact is for $S \geq 3$, the containers in reachable tiers can be permuted into any wanted arrangement. The conclusion is given in "Wang, N. (2014). Optimization Study on Container Operations in Maritime Transportation. PhD thesis, City University of Hong Kong, Hong Kong". The proof is detailed given in this work (can be retrieved from City University Library->E-resources->E-theses->By title), and we are preparing a paper for this claim now. As the conclusion has no relationship with the contents in this paper, we remove this conclusion to avoid misunderstanding.

2. Also related to this issue, the three propositions given in the manuscript need proofs as well.

The proofs for the first two propositions are given in the new version. And the proposition 3 is removed from the new version.

3. Another example appears on page 12, section 6.2. The first line in this section mentioned a "container-stack pair", which seems to be a "task" to me. More careful and precise writing could help the reader understand the issue more easily.

We used container-stack pair to refer to a task, and this is not precise. A container-pair is a valid task only when it satisfies four conditions listed in Section 5.2. This is our fault.

As we consolidated Section 5-7, the content is moved to Page 13. We have revised the imprecise writing.

4. Similar situations occur when "solvable" seems to refer to the same concept as "feasible", and when the term "tricky state" pops out without definition. In general, the presentation makes the results appear far more complicated than they really are.

"Solvable" appeared in Proposition 3. Proposition 3 has no relationship with the main contents in this paper, hence we removed Proposition 3.

"Tricky states" is renamed "dead-end states" for better understanding. In the old version, we define "tricky states" in the text under Definition 1. Maybe, it is not

apparent. In the new version, we define “dead-end” states in Definition 6. All the discussions about the “dead-end” states are after the definition.

5. The second concern refers to sections 6 and 7, where the techniques are developed and assembled to form the greedy and speedy heuristic, GASP. Five algorithms are presented in detail, but insights are insufficient. In other words, the manuscript described in detail what the algorithms do, but did not provide sufficient information to explain to the readers why they are designed that way. On top of all these, the manuscript should justify the basic idea of fixing the containers one by one. This idea seems reasonable to me, but it needs a careful justification. After all, the purpose of publication is to contribute to the body of knowledge, in addition to providing pseudo code for programmers to follow

The initial thought for us is to propose a framework and several techniques, and introduce them separately to demonstrate that every part can be used solely. Other scholars can use the framework combined with other techniques. The proposed techniques can also be used in other frameworks. Hence, Section 5 explains framework, Section 6 explains techniques and Section 7 explains the assembled heuristic.

But we find such a presentation is too hard to follow. Now we just explain the assembled heuristic, and the details of the framework and techniques are introduced after the heuristic in the same section (Section 5). Useful parts will still be adopted if some reader thinks they are effective in the future research.

For the idea behind fixing containers one by one, we explain it in Section 5.1: The advantage of fixing containers one by one lies in twofold: 1) movements are more target-oriented. Some works move a container each time, but they don't have a clear aim. Hence, it is difficult to measure the benefit of moves and moreover, moves are blind: a container can be unnecessarily moved from stack a to b in a round and then moved from stack b to a after several rounds. 2) fixed containers are not allowed to be moved. But unfixed orderly containers are allowed to be moved. This can distinguish fixed and unfixed orderly containers. Even though both are orderly, the statuses of both kinds of containers are totally different. In works which does not distinguish fixed and unfixed containers, some containers which do not need moved (fixed containers in our context) are involved in the movements.

For the idea behind other techniques such as tier-protection, pre-extreme are added in the new version.

- We deal with the container pre-marshalling problem (CPMP).
- Concepts of state feasibility, stability are introduced.
- A feasibility-based heuristic for the CPMP is proposed.
- Computational experiments prove the effectiveness of our approach.

A feasibility-based heuristic for the container pre-marshalling problem

Bo Jin^a, Andrew Lim^{a,1}, Ning Wang^{b,*}, Zizhen Zhang^c

^a*Department of Management Sciences, City University of Hong Kong, Tat Chee Avenue, Kowloon Tong, Hong Kong*

^b*Department of Information Management, School of Management, Shanghai University, Shanghai, China*

^c*School of Mobile Information Engineering, Sun Yat-Sen University*

Abstract

This paper addresses the container pre-marshalling problem (CPMP) which rearranges containers inside a storage bay to a desired order. By far, target-driven algorithms have relatively good performance among all algorithms; they have two key components: 1) containers are rearranged to their desired slots one by one in a certain order; and 2) rearranging one container is completed by a sequence of movements. Our algorithm improves the performance from the aforementioned two components. Instead of rearranging containers in the descending order of their group labels (traditional idea), this paper determines the order as the algorithm goes on by an evaluation function. To efficiently explore the enlarged search space, we are the first one to come up with a concept of state feasibility to cut branches. In addition, we improve the sequence of movements by techniques such as pre-extreme avoidance and tier protection. Computational experiments showcase that the performance of our proposed heuristic is better than other heuristics.

Keywords: container pre-marshalling problem, feasibility-based heuristic

1. Introduction

Since the commencement of containerization, the global use of standardized containers has dramatically improved international trade. Containers enable the smooth flow of goods between

*Corresponding author. Tel: +86 18930400865

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

¹Andrew Lim is currently on no pay leave from City University of Hong Kong.

multiple transportation modes without directly handling the freight. Over the years, stringent requirements such as just-in-time operations from consignors have created additional challenges for the container transportation industry.

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

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

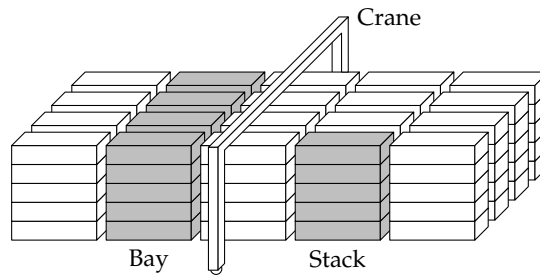


Figure 1: Yard block.

Containers in the same stack are organized in a last-in-first-out matter. To retrieve containers placed in lower levels, containers on top of them must be relocated to other places (other slots or the yard) first. Such forced movements are known as container reshuffles. Container pre-marshalling problem (CPMP) addresses the reorganization of containers inside a storage area such that no reshuffling is further required when containers are retrieved. Hence, CPMP is to improve terminals' performance level, including their throughput rate per berth and the turnaround time of vessels or road trucks [13].

The CPMP can be formally defined as follows: containers are initially placed in several stacks.

1
2
3 These stacks form a bay. Each container is marked with a group label. CPMP is to rearrange
4 the containers within the bay, so that containers in each stack are placed in a descending order of
5 group labels from bottom to top. The objective is to carry out the rearrangement with the fewest
6 number of container movements. This paper investigates the CPMP, and it is assumed that the
7 retrieval order of the containers is known beforehand and no container arrives at or leaves from
8 the bay during pre-marshalling.
9

10
11 This paper makes two main contributions to the literature. One contribution is the idea behind
12 the proposed heuristic. In the traditional algorithms, containers are allocated according to a certain
13 order, such as the descending order of their groups [8]; in this research, the order to allocate
14 containers is not fixed beforehand, but rather that it is decided as the algorithm goes on. This
15 innovation undoubtedly enlarges the search space and decreases the search efficiency. To avoid
16 efficiency decrease, we propose the concept of state feasibility to check the feasibility of allocating
17 a certain container before we actually allocate it. The time complexity of check feasibility is
18 only $O(G)$, here G is the number of groups. Our paper is the first work that uses feasibility to
19 cut branches in CPMP algorithms. Another contribution is that the proposed heuristic performs
20 outstandingly on both dense and loose instances, compared with existing heuristics. The better
21 performance is thanks to the techniques used, such as state stability, dead-end states avoidance
22 and tier-protection indicator which will be explained in Section 5.
23
24

25
26 The remainder of this paper is structured as follows. Section 2 reviews existing approaches
27 in the literature. Section 3 gives a mathematical description of the problem and lists the notation
28 used throughout this paper. The concept of state, state feasibility and stable state are introduced
29 in Section 4 Comprehensive description of the heuristic and associate techniques are explained in
30 Sections 5. Section 6 illustrates the results of our computational experiments, and the last section
31 concludes this paper and identifies future research directions.
32
33

34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 **2. Recent work**

52
53 According to recent reviews [3,16], problems related to or caused by container reshuffles at
54 terminals include three major kinds of decision problems. The first problem is the container relo-
55 cation problem [12] which minimizes the total operational cost in the container retrieval process.
56
57
58
59
60
61
62
63
64
65

The operational cost is commonly measured by the number of relocations conducted or the total operational time. The second is the container pre-marshalling problem addressed in this paper. The third problem is container stacking problem which decides the selection of storage locations or sequences the movements of cranes for arriving containers [6].

To the best of our knowledge, only seven works study the solutions to the pre-marshalling problem, which is rather few compared with other problems in ports, such as berth allocation problems and quay crane scheduling problems. The aforementioned two problems have been studied in more than 120 publications just since 2009 [1]. One reason behind is that the variants of pre-marshalling problem by far are few. Known variants appear in Wang et al. [18] which considers truck lanes and in Huang & Lin [10] which requires containers of different groups be separately located in the final layout. In the following, the only seven works will be discussed in sequence.

Lee & Hsu [15] develop an integer programming model for the CPMP. In this work, the problem is formulated as a multi-commodity network flow problem. The overall network is divided into several subnetworks, with each subnetwork representing an intermediate layout. The nodes in a subnetwork correspond to the slots that accommodate containers, and the commodities correspond to the containers stored in the bay. Every valid flow in the network represents a solution to the pre-marshalling problem. The model provides an innovative viewpoint to the problem; however, its performance is limited because the network is too large even for a small instance.

Caserta & Voß [4] provide a greedy heuristic, the corridor method, for solving the problem. The heuristic selects the direction of movements in a randomized manner according to the attractiveness of available successors confined by the corridor. The attractiveness is measured by an estimated number of additional relocations needed for the particular successor. A local improvement procedure is also conducted to accelerate the heuristic process.

A neighborhood search is proposed by Lee & Chao [14], which repeatedly modifies the current solution until some termination condition is met. Unlike other existing solution-construction approaches, the neighborhood search is required to start from a pre-generated initial solution. A feasible solution can be further improved by a four-step procedure, and the diversity of the neighborhood is raised by multiple subroutines. The main drawback of the approach is the unreliability

of random solution modifications, i.e., the feasibility of the resultant new solutions are not always ensured.

Bortfeldt & Forster [2] describe a tree search procedure for solving the problem. In the tree search structure, solutions are constructed by compound moves instead of single moves. Moves are classified into four types, and only the most promising ones are adopted in the branching scheme.

There are two studies in the literature regarding heuristic approaches. They adopt the same heuristic strategy that containers are handled in reverse of their retrieval order.

Expósito-Izquierdo et al. [8] provide the first group-oriented heuristic for the CPMP. Their method iteratively handles containers in a descending order of container group values. After handling all containers with a specific group value, a stack filling process is applied to reduce the number of disorderly containers in the bay.

Huang & Lin [10] solved two variants of CPMP by a heuristic algorithm. One variant is commonly seen which allows containers of different groups to mix within a bay. The other variant is newly proposed, which requires containers of different groups be separately located in the final layout.

Wang et al. [18] propose a target-guided heuristic (TGH) and two beam search algorithms. The TGH also handles containers in a descending order of the container group values. The method to determine container handle order and the way to handle a container are improved. This work can solve instances of different densities well, especially dense instances with fewer empty slots.

3. Problem description and notations

The CPMP is restricted to the bay size, or more precisely, the dimensions of the operating cranes. As shown in Figure 2, 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 ($H \geq 2$) with E empty slots ($E = SH - N$, $E \geq 2$) left.

Each container is labeled with a group value $g \in [1, G]$. A container is *orderly* if it is supported directly by the ground or another orderly container with equal or larger group value; otherwise, it is *disorderly*. Other phrases which have the same meaning of “orderly/disorderly” in recent literature include “well/badly placed” [2], “well-/non-located” [8], and “clean/dirty” [18].

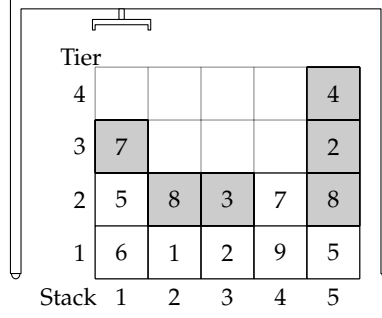


Figure 2: A container bay.

Figure 2 gives an example of a bay for which $S = 5$, $H = 4$, and $N = 13$. Containers are represented by boxes with their group values marked inside. In addition, boxes with gray background are disorderly containers. The objective of the CPMP is to find an optimized sequence with the fewest container movements, by which all containers are rearranged to be orderly. The movement sequence is called “applied to” the CPMP case.

To better describe the solution to CPMP, we represent layout-related items mathematically. The tier and stack numbers of the bay are labeled from bottom to up and left to right, respectively. Let $\mathbb{S} = \{1, \dots, S\}$ 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 s . Note that the height of stacks should not exceed H . The orderly height (number of orderly containers) of s is denoted by $o(s)$.

The slot positioned at t -th tier of stack s is denoted by (s, t) . The group value of a container c is denoted by $g(c)$. Likewise, the group value of a container located in slot (s, t) is denoted by $g(s, t)$.

4. State and State Feasibility

Definition 1 (Fixed containers). *A fixed container is an orderly container and not allowed to be moved in the following movement sequence.*

If a container is a fixed container, then the containers under it must be fixed containers.

Definition 2 (State). *A state (L, f) is a pair composed of a layout L and a fix vector f .*

The fix vector f indicates the number of fixed containers (fixed height) in each stack of L . $f(s)$ is the s th element. Fixed and unfixed containers are separated by a line (a skyline), as shown in Figure 3. In Figure 3(a), $f = (2, 3, 1)$; in Figure 3(b), $f = (2, 3, 0)$.

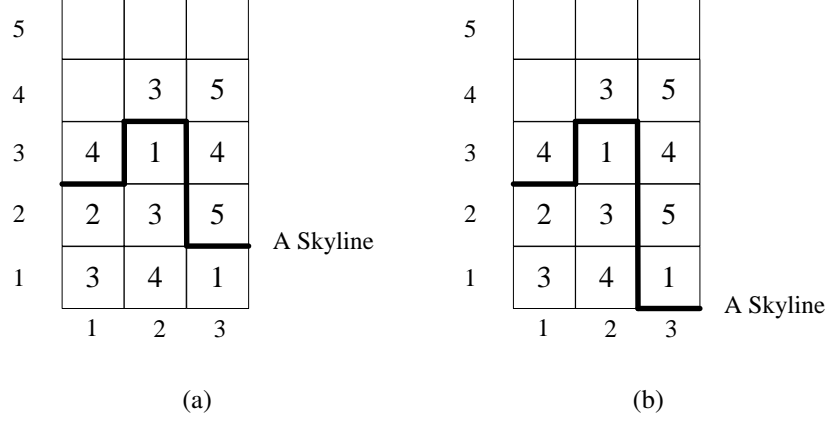


Figure 3: Fix vector and skyline

It is noteworthy that if two states have the same layout but different fix vectors, then they are two different states, just like Figure 3(a) and 3(b).

Definition 3 (Feasibility of a state). *A state (L, f) is feasible if there exists a movement sequence that can convert L to an orderly layout without moving fixed containers indicated by f .*

The necessary condition for state feasibility has been talked about in Wang [17]. To make the content self-contained, we briefly explain it in this paper. Containers above the skyline can be moved, therefore slots and containers above the skyline now are separately considered. In one stack s , suppose the smallest group label under the skyline is g , then $H - f(s)$ slots above the skyline can only be placed with containers with group label no larger than g , or the stack is disorderly. In such a situation, we say stack s has $H - f(s)$ slots with group label g . If the skyline of stack s is the ground, then s has H slots with group label G . For example, in Figure 3(b), stack 1, 2, 3 have 3, 2, 5 slots with group label 2, 1, 5, respectively. The total number of slots with group label g in the whole bay is denoted as $r(g)$ (resource of group g). Similarly, the number of unfixed containers with group label g is denoted as $d(g)$ (demand of group g). For a feasible state, the slots

available for group g must be more than the demand of group g . That is,

$$\sum_{i=g+1}^G r(i) - \sum_{i=g+1}^G d(i) + r(g) \geq d(g)$$

which is equivalent to

$$\sum_{i=g}^G r(i) \geq \sum_{i=g}^G d(i)$$

Denote $R(g) = \sum_{i=g}^G r(i)$ and $D(g) = \sum_{i=g}^G d(i)$. \mathbf{R} and \mathbf{D} are the two G -dimension vectors with elements $R(g)$ and $D(g)$, respectively. The above condition is expressed as $\Delta = \mathbf{R} - \mathbf{D} \geq \mathbf{0}$. Here Δ is called ‘surplus vector’.

Proposition 1 (Necessary condition for state feasibility). *A necessary condition for a feasible state (L, f) is $\Delta = \mathbf{R} - \mathbf{D} \geq \mathbf{0}$.*

Take the layouts in Figure 3(a) and 3(b) as an example, and Table 1 shows the surplus vectors for both cases. Case a is infeasible as only $\Delta(1) \geq 0$; while the feasibility of Case b is not sure since the above conditions are necessary conditions. Intuitively, it can be seen that none of the three stacks in Figure 3(a) can accommodate containers with group label 5 orderly, and this is the reason why Case a is infeasible.

Table 1: Computation for the surplus vector.

Case a						Case b					
g	$d(g)$	$D(g)$	$r(g)$	$R(g)$	$\Delta(g)$	g	$d(g)$	$D(g)$	$r(g)$	$R(g)$	$\Delta(g)$
1	0	5	6	9	3	1	1	6	2	10	4
2	0	5	3	3	-2	2	0	5	3	8	3
3	1	5	0	0	-5	3	1	5	0	5	0
4	2	4	0	0	-4	4	2	4	0	5	1
5	2	2	0	0	-2	5	2	2	5	5	3

It is noteworthy that if only one container c is unfixed in a state (L, f) , and the surplus vector $\Delta \geq 0$, then at most one movement is need to convert the layout to a feasible layout. As $\Delta \geq 0$, and $D[g(c)] = d[g(c)] = 1$, we have $S[g(c)] > 1$, i.e., there is at least one slot which can accommodate c orderly. We just need c to the slot.

Definition 4 (Container stability). *Given a feasible state, a disorderly container is unstable. For any orderly container c in slot (s, t) , try to fix container c and those underneath it; if the resultant state remains feasible, c is stable. Otherwise, c is unstable.*

According to the definition, an unstable container cannot be fixed in the current slot even if it is orderly, because the resultant state will be infeasible. Unstable must to be moved in the future. Denote the stable height (number of stable containers) of stack s by $sh(s)$, then we have $f \leq sh \leq o \leq h$. If a container c is stable in stack s , we say stack s **stabilizes** container c .

Figure 4 gives an example of a state. Fixed containers are labeled with a solid square. Unstable containers are highlighted with gray background. Notice that the container (2, 1) with group value 6 is orderly but unstable.

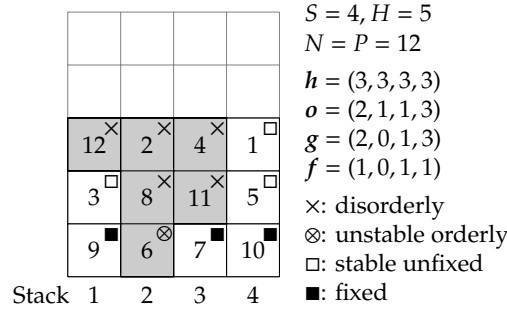


Figure 4: Stability of containers.

An orderly but unstable container indicates that its orderliness is valueless because it definitely requires at least one reshuffle. Using the concept of container stability instead of container orderliness is more precise. For example, when selecting the placement of a blocking container, if the container is orderly but unstable after moving to a stack, then the attractiveness of such a movement should be reconsidered.

Note that the stability of containers should be recomputed after any container is fixed.

Definition 5 (Extreme state). *A state (L, f) is called an extreme state if $|\{s \in \mathbb{S} : f(s) = H\}| = S - 2$.*

The definition of extreme state is to emphasize the especial case where $S - 2$ stacks are fully fixed. In such a case, unfixed containers can only be moved between two stacks, say a and b ; the case is feasible if and only if one of the following conditions is satisfied:

1. Both stacks a and b are orderly;
2. Stack a is orderly, and stack b is disorderly. Disorderly containers in b can be allocated to a and a is still orderly after the allocation.

Definition 6 (Dead-end state). *A state is called a dead-end state if it is an infeasible extreme state.*

The dead-end states are infeasible, therefore we should avoid generating them. We have the following proposition:

Proposition 2 (Existence of dead-end states). *No dead-end state can be generated whatever movements and fixed vectors are applied to cases with $E \geq 2(H - 1)$. For cases with $E < 2(H - 1)$, there exist possibilities that the algorithm falls into a dead-end state.*

When $E \geq 2(H - 1)$, it has $N \leq SH - 2H + 2$, even though all containers are fixed, there are at most $S - 2$ stacks fully occupied with fixed containers: 1) if there are fewer than $S-2$ stacks fully occupied, the state is not a dead-end state; 2) if there are $S-2$ stacks fully occupied, there are at most two extra containers in the left two stacks. The state is not a dead-end state, either. For cases with $E < 2(H - 1)$, there exist possibilities that the algorithm falls into a dead-end state. As shown in Figure 5(a), it has $E \geq 2(H - 1)$, and it has no chance to generate dead-end states. In Figure 5(b) with $E < 2(H - 1)$, when $f = (0, 0, 5, 5)$, the state is a dead-end state.

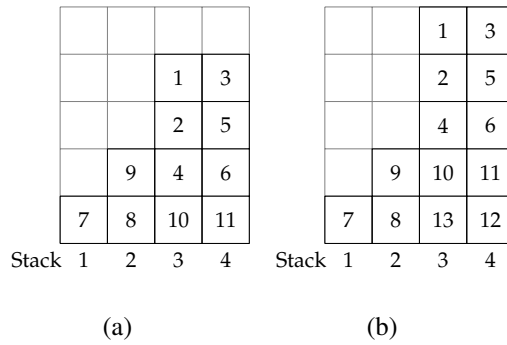


Figure 5: Existence of dead-end states

5. Feasibility-based heuristic

In this section, a feasibility-based heuristic (FBH) is developed for solving the CPMP. The proposed heuristic is a target-driven algorithm. Target-driven algorithms have appeared in other works[8,18]: generally, containers (targets) are rearranged to certain slots one by one, and each rearrangement is achieved by a sequence of movements. The heuristic can be implemented solely, then it is a greedy algorithm; the heuristic can also be the major component of other frameworks, such as GRASP and beam search. Target-driven algorithms are effective among all existing algorithms, and our algorithm in this work is more effective. We optimize the order of rearranging containers as well as the sequence of movements in one rearrangement.

The optimization is due to the concept of state feasibility. This is why our algorithm is called feasibility-based heuristic. Before really rearranging a certain container, the feasibility of rearrangement is checked. If the rearrangement is infeasible, then we give up this rearrangement. Due to the feasibility, we can explore larger space without sacrificing efficiency: the order to rearrange containers is not fixed beforehand, but rather that it is decided as the algorithm goes on.

5.1. Heuristic framework

According to Wang [17], in dense instances such that $E < H$, for a stack s , only the top $\sum_{i \neq s} e(i)$ containers can be moved to other stacks. The immovable number of containers (unreachable tiers) for any stack is $U = H - \sum_i e(i) = H - E$. Containers in the bottom $H - E$ tiers of the bay are unreachable, that is, no sequence can move these containers. In loose instances such that $E \geq H$, all containers in a stack are movable. Let $U = \max\{H - E, 0\}$ denotes the number of unreachable tiers for any instances.

The feasibility-based heuristic first constructs the initial state according to the unreachable tiers. In the initial state, L is the initial layout L^0 and containers in unreachable tiers are fixed, i.e., $f = U \cdot \mathbf{1}$, where $\mathbf{1}$ is a S -dimension all-ones vector.

Starting from the initial state, the heuristic repeatedly fixes a target container c^* to a target stack s^* . Stack s^* has $f(s^*)$ fixed containers; hence, c^* is fixed at tier $f(s^*) + 1$. The algorithm continues until all containers are fixed. The procedure of the heuristic is concisely described in Algorithm 1.

Algorithm 1: Feasibility-based heuristic.

```
1 begin
2    $(L, f) := (L^0, U \cdot \mathbf{1});$ 
3   repeat  $N - SU$  times do
4      $(c^* \rightarrow s^*) :=$  the chosen valid task;
5      $L =$  resultant layout after moving  $c^*$  to  $s^*$ ;
6      $f(s^*) := f(s^*) + 1;$ 
```

The procedure of choosing a target container c^* and a target stack s^* and the procedure of moving c^* to s^* will be explained in the following section.

The advantage of fixing containers one by one lies in twofold: 1) movements are more target-oriented. Some works such as [2] move a container each time, but they don't have a clear aim. Hence, it is difficult to measure the benefit of moves and moreover, moves are blind: a container can be unnecessarily moved from stack a to b in a round and then moved from stack b to a after several rounds. 2) fixed containers are not allowed to be moved. But unfixed orderly containers are allowed to be moved. This can distinguish fixed and unfixed orderly containers. Even though both are orderly, the statuses of both kinds of containers are totally different. In works which does not distinguish fixed and unfixed containers, some containers which do not need moved (fixed containers in our context) are involved in the movements.

5.2. Valid task

At every step of the heuristic, a task determines an unfixed container (target container) and a destination stack (target stack). A container-stack pair $c \rightarrow s$ is a valid task for the current state (L, f) only if

- c is unfixed,
- $f(s) < H$,
- $g(c) \leq g(s, f(s))$, and
- $\Delta(\varphi) \geq H - f(s), \varphi \in (g(c), g(s, f(s))]$.

The first condition is easy to understand. The second condition ensures that s has a slot to accommodate c . The third condition makes sure that the group label of c is no larger than that of

the highest fixed container of s , here, $g(c)$ and $g(s, f(s))$ denotes the group labels of c and highest fixed container of s , respectively. The last condition ensures the non-negativity of the resultant surplus vector. If c is moved to s , then the $g(c)$ th element of demand vector $d[g(c)]$ decreases 1, i.e., $d'[g(c)] = d[g(c)] - 1$, while other elements do not change. In the meanwhile, for the resource vector, $r'[g(s, f(s))] = r[g(s, f(s))] - (H - f(s))$ while $r'[g(c)] = r[g(c)] + H - f(s) - 1$, and other elements do not change. Then for $\varphi \in (g(c), g(s, f(s))]$, the feasible condition in the new layout is $D'(\varphi) = \sum_{g \geq \varphi} d'(g) = D(\varphi) \leq R'(\varphi) = \sum_{g \geq \varphi} r'(g) = R(\varphi) - (H - f(s))$, i.e., $\Delta(\varphi) \geq H - f(s)$.

Pairs satisfy the above conditions compose a set \mathbb{T} . However, the above conditions are only necessary conditions, so the feasibility of the resultant state is not sufficiently ensured, due to the possibility of dead-end states.

There are three methods for resolving the dead-end state issue.

1. Prevent from entering an extreme state when deciding the next task;
2. Design an ideal task accomplishment procedure that makes the resultant extreme state feasible;
3. Allow entering a dead-end state and relabel a fixed container as unfixed to run away from the dead-end state.

Our previous work [18] adopts the last method listed above. When the algorithm enters into a dead-end state, the algorithm relabels a fixed container as unfixed and moves it to a temporary slot; then the state becomes non-dead-end. After taking a sequence of movements, the relabeled container is recovered to its original slot. The disadvantage of such a choice is that running away from existing dead-end states costs many movements and these movements are unnecessary per se. A more choice is to avoid resulting in dead-end states when choosing valid tasks.

Definition 7 (Pre-extreme state). *A feasible state is a pre-extreme state if $|\{s \in \mathbb{S} : f(s) = H\}| = S - 3$, $|\{s \in \mathbb{S} : f(s) = H - 1\}| \geq 1$ and $\sum_{s \in \mathbb{S}} f(s) < N - 2$.*

The algorithm only explores state (L, f) which satisfy the necessary condition for state feasibility (see Proposition 1). When only three stacks have available slots while the other $S - 3$ stacks

are fully fixed ($|s \in \mathbb{S} : f(s) = H| = S - 3$), and at least one stack has only one available slot left $|s \in \mathbb{S} : f(s) = H - 1| \geq 1$. Let us investigate the validity of performing a task $c \rightarrow s$ with $f(s) = H - 1$ based on the number of remaining unfixed containers.

1. If $\sum_{s \in \mathbb{S}} f(s) = N$, all the containers are fixed, the algorithm terminates with an orderly layout.

2. If $\sum_{s \in \mathbb{S}} f(s) = N - 1$, only one container remains unfixed. Performing the task will make the layout feasible.

3. If $\sum_{s \in \mathbb{S}} f(s) = N - 2$, two containers remain unfixed. Suppose the last two unfixed containers are a and b , and the valid task is $a \rightarrow s$ with $f(s) = H - 1$. When generating valid tasks, the fourth condition $\Delta(\varphi) \geq H - f(s), \varphi \in (g(c), g(s, f(s))]$ must be satisfied, i.e., the resultant layout L' after task $a \rightarrow s$ satisfies $\Delta \geq 0$ and only b is unfixed. Hence, L' can be converted into a feasible layout. Performing task $a \rightarrow s$ does not result in a dead-end state.

4. If $\sum f(s)_{s \in \mathbb{S}} \leq N - 3$, 3 or more containers remain unfixed. If the next task is to fix a container to the stack s such that $f(s) = H - 1$, the resultant may be a dead-end state. Taking Figure 6 as an example. Containers with a solid square are fixed containers. For the task $1 \rightarrow 3$, it is valid as the surplus vector of the resultant state is no less than zero. But the resultant state becomes dead-end. Based on the above discussion, we can see that when $\sum_{s \in \mathbb{S}} f(s) < N - 2$, there

	4	6			
	5	6	2	1	
	8	7	4	4	3
	8	10	11	3	5
	12	11	7	10	5
Stack	1	2	3	4	5

Figure 6: Pre-extreme state.

is a chance to generate dead-end state.

When the current state is a pre-extreme state, we eliminate the container-stack pairs $c \rightarrow s$ such that $f(s) = H - 1$ from \mathbb{T} , so that the algorithm avoids entering a dead-end state.

5.3. Task evaluation

Every valid task $c \rightarrow s \in \mathbb{T}$ is evaluated by a tuple with six elements: the tier-protection indicator, the number of moves required by the STAP, the number of stable containers that need to be moved from the aim stack $sh(s) - f(s)$, the affected demand, fixed height of the aim stack $f(s)$, $-g(c)$. The tuples of valid tasks are compared lexicographically, and the task with the minimum tuple is selected as the next task. The six elements in the tuple will be explained in the following.

The tier-protection indicator is to balance the trade-off between the freedom of task selection and the surplus loss. In principle, a container can be moved to any valid stack. But if a container with small group value occupies an (relatively) empty stack, then the loss of the surplus vector is large: after the target container c is fixed to the target slot $(s, f(s) + 1)$, the surplus $\Delta(\varphi)$ is reduced by $H - f(s)$, for $\varphi \in (g(c), g(s, f(s))]$. The larger the spread between $g(c)$ and $g(s, f(s))$ is, the more the surplus vector will lose. For example, in Figure 6, if container 1 (container with group value 1) is moved on container 3 (stack 5 tier 3), the supply of group 3 is reduced by 2 as 1 becomes the highest orderly container of stack 5. The cumulative supplies $G(3)$ and $G(2)$ are reduced by 2 accordingly.

If a pair $c \rightarrow s$ satisfies $f(s) < P1$ and affected demand $\sum_{\varphi=g(c)+1}^{g(s, f(s))} d(\varphi) \geq P2$, then the task cannot be carried out and is deleted from the evaluation list. As customizable parameters, the number of tiers protected $P1$ and the threshold value $P2$ can be adjusted optionally.

The number of moves required by the STAP is the movements for moving c to s , the number is precise while in [18] the number is just an estimation. The process to calculate the number is introduced in Section 5.4. The affected demand is $\sum_{\varphi=g(c)+1}^{g(s, f(s))} d(\varphi)$.

5.4. Speedy task accomplishment procedure

After the task is decided, a speedy task accomplishment procedure (STAP) is carried out to accomplish the task, resulting in a new state. STAP is a move sequence which is based on task types.

Let the target container be denoted by c^* (located in (s^+, t^+)) and the target stack by s^* . The task is **immediate** if c^* is just on the highest fixed container in s^* , i.e., $s^+ = s^*$ and $t^+ = f(s^*) + 1$;

internal if c^* is above but not on the highest fixed container in s^* , i.e., $s^+ = s^*$ and $t^+ > f(s^*) + 1$; and **external** if c^* and s^* are in different stacks, i.e., $s^+ \neq s^*$.

5.4.1. Immediate task

An immediate task does not require any move because the target container is already located in the target slot.

5.4.2. Internal task

For an internal task, let a denote the number of empty slots except those in s^+ and 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. Blocking containers refer to those above target containers or target slots. The accomplishment procedure of an internal task is given in Algorithm 2.

Algorithm 2: Accomplish an internal task.

<p>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: $b_2 = t^+ - f(s^+) - 1$</p> <p>1 case I1: $a \geq b_2$ 2 repeat b_1 times do 3 $\mathbb{R} := \{s \neq s^+ : h(s) < H, \alpha(s^+, s) \geq b_2\};$ 4 Relocate($s^+, 1, \mathbb{R}$); 5 $\mathbb{I} := \{s \neq s^+ : h(s) < H, E - e(s^+) - e(s) \geq b_2\};$ 6 $s^{\text{tmp}} := \text{Interim}(\mathbb{I});$ 7 Move($s^+, 1, s'$); 8 Relocate($s^+, b_2, \mathbb{S} \setminus \{s^+, s^{\text{tmp}}\}$); 9 Move($s^{\text{tmp}}, 1, s^+$); 10 // I1: $h(s^+) - f(s^+) + 1$ moves</p>	<p>10 case I2: $a < b_2 \ \& \ \{s \neq s^+ : h(s) < H\} > 1$ 11 Relocate($s^+, b_1, \mathbb{S} \setminus \{s^+\}$); 12 $s_1^{\text{tmp}} := \text{Interim}(\mathbb{S} \setminus \{s^+\});$ 13 Move($s^+, 1, s_1^{\text{tmp}}$); 14 $k_2 := E - e(s^+) - e(s_1^{\text{tmp}}) - 1;$ 15 Relocate($s^+, k_2, \mathbb{S} \setminus \{s^+, s_1^{\text{tmp}}\}$); 16 Find s_2^{tmp} s.t. $s_2^{\text{tmp}} \notin \{s^+, s_1^{\text{tmp}}\} \ \& \ h(s_2^{\text{tmp}}) < H;$ 17 Move($s_1^{\text{tmp}}, 1, s_2^{\text{tmp}}$); 18 Move($s^+, b_2 - k_2, s_1^{\text{tmp}}$); 19 Move($s_2^{\text{tmp}}, 1, s^+$); 20 // I2: $h(s^+) - f(s^+) + 2$ moves</p> <p>20 case I3: $a < b_2 \ \& \ \{s \neq s^+ : h(s) < H\} = 1$ 21 Find s' s.t. $s' \neq s^+ \ \& \ h(s') < H;$ 22 $s^{\text{tmp}} := \text{InterimFull}(\mathbb{S} \setminus \{s^+, s'\});$ 23 BiSender($s^{\text{tmp}}, 1, s^+, b_1, \{s'\}$); 24 Move($s^+, 1, s^{\text{tmp}}$); 25 Move($s^+, b_2, s'$); 26 Move($s^{\text{tmp}}, 1, s^+$); 27 // I3: $h(s^+) - f(s^+) + 2$ moves</p>
--	--

The accomplishment procedure is carried out based on the relationship between available empty slots and the number of blocking containers. The relationship determines how containers are relocated.

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

In case I1, b_1 containers above c^* are relocated one by one. \mathbb{R} represents the set of available destination stack for the top container which blocks c^* . A stack s is available if $s \neq s^+, h(s) < H, \alpha(s^+, s) \geq b_2$. Here α function in Algorithm 3 indicates how many slots are left for b_2 .

Algorithm 3: Alpha function.

```

1  function  $\alpha(s^+, s^{\text{dst}})$ 
2      //  $s^+ \neq s^{\text{dst}}$ 
3       $e^{\min} := \min\{e(s) > 0 : s \neq s^+\};$ 
4       $e^{\text{sec}} := \min\{e(s) > e^{\min} : s \neq s^+\};$ 
5       $k^{\min} := |\{s \neq s^+ : e(s) = e^{\min}\}|;$ 
6      if  $e(s^{\text{dst}}) > e^{\min}$  then
7          return  $E - e(s^+) - e^{\min} - 1;$ 
8      else if  $e^{\min} \geq 2$  then
9          return  $E - e(s^+) - e^{\min};$ 
10     else if  $k^{\min} = 1$  then
11         return  $E - e(s^+) - e^{\text{sec}} - 1;$ 
12     else
13         return  $E - e(s^+) - 2;$ 

```

In the process of relocating containers from b_1 , it need consider reserving slots for containers from b_2 . Because c^* is above b_2 . After retrieving b_1 , c^* is relocated before containers in b_2 . If the relocation is to put c^* and b_2 in the same stack, then c^* needs additional moves to be retrieved. Hence, we reserve the stack with the minimum empty slots s_{\min} for c^* , and the slots from $\{s \in S : s \neq s^+, s \neq s_{\min}\}$ for b_2 . Of course, containers in b_1 can also occupy s_{\min} and $\{s \in S : s \neq s^+, s \neq s_{\min}\}$, and definitely $|\{s \in S : s \neq s^+\}| \geq b_1 + b_2$. But the number of containers from b_1 which occupy $\{s \in S : s \neq s^+, s \neq s_{\min}\}$ needs computation, in order to satisfy b_2 first. Hence, when deciding whether a stack s^{dst} can be the destination of a container c from b_1 , α function is used to compute if c is placed to s^{dst} how many slots left for b_2 . And if the left slots are not enough for b_2 , c cannot be moved to s^{dst} .

If $e(s^{\text{dst}}) > e^{\min}$, that is, s^{dst} is not the highest non-full stack. e^{\min} is reserved for c^* . Slots of s^+ cannot be used. If c is moved to s^{dst} , left slots for b_2 is $E - e(s^+) - e^{\min} - 1$.

If $e(s^{\text{dst}}) = e^{\min} \geq 2$, that is s^{dst} is the highest non-full stack and it has at least 2 empty slots. If c is moved to $e(s^{\text{dst}})$, the stack can also accommodate c^* , hence, left slots for b_2 is $E - e(s^+) - e^{\min}$.

If $e(s^{\text{dst}}) = e^{\min} = 1$ and $k^{\min} = 1$, that is, s^{dst} is the unique highest stack with only one empty slot. If c is moved to s^{dst} , the stack with the second highest height will reserve for c^* in the next round. The slots left for b_2 becomes $E - e(s^+) - e^{\text{sec}} - 1$.

If $e(s^{\text{dst}}) = e^{\min} = 1$ and $k^{\min} \geq 2$, another stack with the one empty slot accommodates c^* in the future. If c is moved to s^{dst} , the slots left for b_2 becomes $E - e(s^+) - 2$.

The container relocation is illustrated in Algorithm 4, which relocates k containers from a sender stack s^{src} to \mathbb{R} . For each of the top k containers of the sender, the destination stack s^{dst} is properly selected from \mathbb{R} according to the evaluations by function $\text{EvalMove}(s^{\text{src}}, s^{\text{dst}})$. Function $\text{Move}(s^{\text{src}}, k, s^{\text{dst}})$ performs k relocations from stack s^{src} to stack s^{dst} . The function $\text{EvalMove}(s^{\text{src}}, s^{\text{dst}})$ and $\text{Move}(s^{\text{src}}, k, s^{\text{dst}})$ are also illustrated in Algorithm 4.

Algorithm 4: Relocation function.

<pre> 1 function Relocate($s^{\text{src}}, k, \mathbb{R}$) 2 repeat k times do 3 $\mathbb{R}' := \{s \in \mathbb{R} : h(s) < H\};$ 4 $s^{\text{dst}} := \arg \min_{s \in \mathbb{R}'} \text{EvalMove}(s^{\text{src}}, s);$ 5 $\text{Move}(s^{\text{src}}, 1, s^{\text{dst}});$ 6 function $\text{Move}(s^{\text{src}}, k, s^{\text{dst}})$ 7 $\text{Move } k \text{ containers from stack } s^{\text{src}} \text{ to stack } s^{\text{dst}};$ </pre>	<pre> 8 function $\text{EvalMove}(s^{\text{src}}, s^{\text{dst}})$ 9 $c := (s^{\text{src}}, h(s^{\text{src}}));$ 10 $g := g(c);$ 11 $q := q(s^{\text{dst}}, h(s^{\text{dst}}));$ 12 case $sh(s^{\text{dst}}) = h(s^{\text{dst}})$ & stack s^{dst} can stabilize c 13 return $\langle 1, \sum_{\varphi=g+1}^q d(\varphi) \rangle;$ 14 case $sh(s^{\text{dst}}) < h(s^{\text{dst}})$ & $g \geq m(s^{\text{dst}})$ 15 return $\langle 2, g - m(s^{\text{dst}}) \rangle;$ 16 case $g(s^{\text{dst}}) < h(s^{\text{dst}})$ & $g < m(s^{\text{dst}})$ 17 return $\langle 3, m(s^{\text{dst}}) - g \rangle;$ 18 case $g(s^{\text{dst}}) = h(s^{\text{dst}})$ & stack s_2 cannot stabilize c 19 return $\langle 4, q \rangle;$ </pre>
---	--

Let us define the capability of an occupied slot (s, t) by $q(s, t) = g(s, t)$ if the container inside is orderly, otherwise $q(s, t) = 0$; specifically, regarding the ground as an occupied slot at tier 0 with group value G . The evaluations of moves are represented by numerical tuples, which are lexicographically comparable. $\text{EvalMove}(s^{\text{src}}, s^{\text{dst}})$ returns the penalty of the move from stack s^{src} to s^{dst} . Define the *messiness* of stack s by $m(s) = \max_{sh(s) < t \leq h(s)} g(s, t)$, that is the largest group value among the unstable containers in stack s . The preferences of selecting a stack s^{dst} as the destination stack for a blocking container are as follows.

1. If stack s^{dst} is entirely stable and can stabilize c , the minimum affected demand is preferred;
2. If stack s^{dst} is not entirely stable and $g(c) \geq m(s^{\text{dst}})$, the minimum gap between $m(s^{\text{dst}})$ and $g(c)$ is preferred;
3. If stack s^{dst} is not entirely stable and $g(c) > m(s^{\text{dst}})$, the minimum gap between $g(c)$ and $m(s^{\text{dst}})$ is preferred;
4. If stack s^{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 consider the messiness of the destination stack. Messiness refers to the largest group value of the unstable containers in a stack; larger messiness implies a higher urgency of reshuffles. The last preference indicates that an entirely stable stack should be protected from being ruined.

Function `EvalMove` returns a tuple, the first element indicates the type of the value and the second element indicates the scale of the value.

In cases I1 and I2, an interim stack is selected to temporarily store the target container (function `Interim(I)`). The selection prefers stacks that are not entirely stable, with the largest messiness, then entirely stable stacks with the smallest group of orderly containers. The most unattractive stack for receiving blocking containers is selected as the interim stack. In cases I3 and E4 (in Section 5.4.3), the interim stack is selected by the minimum group value of the top containers of these full stacks, which is function `InterimFull(F)`.

Algorithm 5: Interim and InterimFull functions.

<pre> 1 function Interim(I) 2 $I_1 := \{s \in I : sh(s) < h(s) < H\};$ 3 $I_2 := \{s \in I : sh(s) = h(s) < H\};$ 4 if $I_1 \neq \emptyset$ then 5 return $\arg \max_{s \in I_1} m(s);$ 6 else 7 return $\arg \min_{s \in I_2} q(s, h(s));$ </pre>	<pre> 8 function InterimFull(F) 9 $F_1 := \{s \in F : f(s) \leq sh(s) < h(s) = H\};$ 10 $F_2 := \{s \in F : f(s) < sh(s) = h(s) = H\};$ 11 if $F_1 \neq \emptyset$ then 12 return $\arg \min_{s \in F_1} g(s, h(s));$ 13 else 14 return $\arg \min_{s \in F_2} g(s, h(s));$ </pre>
--	--

Function `BiSender` ($s_1^{\text{src}}, k_1, s_2^{\text{src}}, k_2, \mathbb{R}$) performs relocations from two senders, stacks s_1^{src} and s_2^{src} , to the same receiver set \mathbb{R} , and the respective quantities are k_1 and k_2 . The moving order of the

two top containers from two senders is determined by the evaluation tuple. The one with smaller tuple is moved first. In Algorithm 6, \vec{v}_1 and \vec{v}_2 are tuple results of function EvalMove.

Likewise, function BiReceiver($s^{\text{src}}, k_1, \mathbb{R}_1, k_2, \mathbb{R}_2$) in the external case relocates k_1 and k_2 containers from one sender stack s^{src} to two receiver sets, \mathbb{R}_1 and \mathbb{R}_2 , respectively.

Algorithm 6: BiSender and BiReceiver functions.

<pre> 1 function BiSender($s_1^{\text{src}}, k_1, s_2^{\text{src}}, k_2, \mathbb{R}$) 2 $i := k_1, j := k_2$; 3 repeat $k_1 + k_2$ times do 4 if $j = 0$ then 5 Relocate($s_1^{\text{src}}, i, \mathbb{R}$); 6 $i := 0$; 7 else if $i = 0$ then 8 Relocate($s_2^{\text{src}}, j, \mathbb{R}$); 9 $j := 0$; 10 else 11 $\mathbb{R}' := \{s \in \mathbb{R} : h(s) < H\}$; 12 $s_1^{\text{dst}} := \arg \min_{s \in \mathbb{R}'} \text{EvalMove}(s_1^{\text{src}}, s)$; 13 $\vec{v}_1 := \text{EvalMove}(s_1^{\text{src}}, s_1^{\text{dst}})$; 14 $s_2^{\text{dst}} := \arg \min_{s \in \mathbb{R}'} \text{EvalMove}(s_2^{\text{src}}, s)$; 15 $\vec{v}_2 := \text{EvalMove}(s_2^{\text{src}}, s_2^{\text{dst}})$; 16 if $\vec{v}_1 \leq^{\text{lex}} \vec{v}_2$ then 17 Move($s_1^{\text{src}}, 1, s_1^{\text{dst}}$); 18 $i := i - 1$; 19 else 20 Move($s_2^{\text{src}}, 1, s_2^{\text{dst}}$); 21 $j := j - 1$; </pre>	<pre> 22 function BiReceiver($s^{\text{src}}, k_1, \mathbb{R}_1, k_2, \mathbb{R}_2$) 23 $i := k_1, j := k_2$; 24 repeat $k_1 + k_2$ times do 25 if $j = 0$ then 26 Relocate($s^{\text{src}}, i, \mathbb{R}_1$); 27 $i := 0$; 28 else if $i = 0$ then 29 Relocate($s^{\text{src}}, j, \mathbb{R}_2$); 30 $j := 0$; 31 else 32 $\mathbb{R}'_1 := \{s \in \mathbb{R}_1 : h(s) < H\}$; 33 $\mathbb{R}'_2 := \{s \in \mathbb{R}_2 : h(s) < H\}$; 34 $s_1^{\text{dst}} := \arg \min_{s \in \mathbb{R}'_1} \text{EvalMove}(s^{\text{src}}, s)$; 35 $\vec{v}_1 := \text{EvalMove}(s^{\text{src}}, s_1^{\text{dst}})$; 36 $s_2^{\text{dst}} := \arg \min_{s \in \mathbb{R}'_2} \text{EvalMove}(s^{\text{src}}, s)$; 37 $\vec{v}_2 := \text{EvalMove}(s^{\text{src}}, s_2^{\text{dst}})$; 38 if $\vec{v}_1 \leq^{\text{lex}} \vec{v}_2$ then 39 Move($s^{\text{src}}, s_1^{\text{dst}}$); 40 $i := i - 1$; 41 else 42 Move($s^{\text{src}}, s_2^{\text{dst}}$); 43 $j := j - 1$; </pre>
---	--

The total number of moves needed in case I1 is $h(s^+) - f(s^+) + 1$, and the number of moves in case I2 and case I3 is $h(s^+) - f(s^+) + 2$. All are noted in the comment of Algorithm 2.

Note that in case I2, the target container c^* is moved to the new interim stack s_2^{tmp} from the first interim stack s_1^{tmp} when there is only one empty slot remaining in $\mathbb{S} \setminus \{s^+, s_1^{\text{tmp}}\}$. This can be modified so that c^* can be moved to a new interim stack s_2^{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 is a full stack in $\mathbb{S} \setminus \{s^+\}$ in case I2, the task can also be completed in a similar way as that used in case I3, with the same operational cost.

5.4.3. 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 describing the accomplishment for an external task is given in Algorithm 7, with four situations considered,

- 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 7: Accomplish an external task.

<p>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: $a = E - e(s^+) - e(s^*)$</p> <p>1 case E1: $a \geq b_1 + b_2$ 2 BiSender($s^+, b_1, s^*, b_2, \mathbb{S} \setminus \{s^+, s^*\}$); 3 Move($s^+, 1, s^*$); 4 // E1: $b_1 + b_2 + 1$ moves</p> <p>4 case E2: $b_1 + 1 \leq a < b_1 + b_2$ 5 $k_2 := a - 1 - b_1$; 6 BiSender($s^+, b_1, s^*, k_2, \mathbb{S} \setminus \{s^+, s^*\}$); 7 Find s^{tmp} s.t. $s^{\text{tmp}} \notin \{s^+, s^*\}$ & $h(s^{\text{tmp}}) < H$; 8 Move($s^+, 1, s^{\text{tmp}}$); 9 Move($s^-, b_2 - k_2, s^+$); 10 Move($s^{\text{tmp}}, 1, s^*$); 11 // E2: $b_1 + b_2 + 2$ moves</p>	<p>11 case E3: $1 \leq a < b_1 + \min\{1, b_2\}$ 12 $k_1 := a - 1$; 13 BiReceiver($s^+, k_1, \mathbb{S} \setminus \{s^+, s^*\}, b_1 - k_1, \{s^*\}$); 14 Find s^{tmp} s.t. $s^{\text{tmp}} \notin \{s^+, s^*\}$ & $h(s^{\text{tmp}}) < H$; 15 Move($s^+, 1, s^{\text{tmp}}$); 16 Move($s^*, b_1 - k_1 + b_2, s^+$); 17 Move($s^{\text{tmp}}, 1, s^*$); 18 // E3: $2b_1 + b_2 - a + 3$ moves</p> <p>18 case E4: $a = 0 < b_1 + b_2$ 19 $s^{\text{tmp}} := \text{InterimFull}(\mathbb{S} \setminus \{s^+, s^*\})$; 20 BiSender($s^{\text{tmp}}, 1, s^+, b_1, \{s^*\}$); 21 Move($s^+, 1, s^{\text{tmp}}$); 22 Move($s^*, b_1 + b_2 + 1, s^+$); 23 Move($s^{\text{tmp}}, 1, s^*$); 24 // E4: $2b_1 + b_2 + 4$ moves</p>
---	--

6. Computational results

In this section, the proposed feasibility-based heuristic (FBH) is compared to two benchmark heuristic approaches. One is the target-guided heuristic (TGH) proposed in our previous work [18], and the other is a similar implementation to the LPFH proposed by [8]. Since the LPFH only

discusses cases with enough empty slots (cases I1 and E1 in our statement) and involves random decisions, it is difficult to implement a deterministic version. Thus, we developed a substitute instead, referred to as the largest group value first heuristic (LGVFH). We implemented both TGH and LGVFH. Experiments were conducted on a PC with Intel Core i7 CPU clocked at 3.40 GHz with Windows 7 operating system. The code was written in Java.

The LGVFH selects the next target container from unfixed containers with the largest group value, and then accomplish it by the STAP with the same user-defined functions used as that in the FBH. Necessary pseudo-code for the LGVFH is given in Algorithm 8.

Algorithm 8: Largest group value first heuristic.

<pre> 1 begin 2 $(L, f) := (L^0, U \cdot 1)$; 3 foreach $g = G, \dots, 1$ do 4 $\mathbb{C}_g :=$ containers with group value g; 5 $\mathbb{A}_g := \emptyset$; // set of aim stacks 6 while $\mathbb{C}_g \neq \emptyset$ do 7 $(c^*, s^*) :=$ 8 $\arg \min_{c \in \mathbb{C}_g, f(s) < H} \text{MoveNeed}(c, s)$; 9 Accomplish (c^*, s^-) by the STAP; 10 $f(s^*) := f(s^*) + 1$; 11 $\mathbb{C}_g := \mathbb{C}_g \setminus \{c^*\}$; 12 $\mathbb{A}_g := \mathbb{A}_g \cup \{s^*\}$; 13 foreach $s^* \in \mathbb{A}_g$ do 14 Fill(s^*); </pre>	<pre> 14 function Fill(s) 15 while $h(s) < H$ do 16 $\mathbb{S}' := \emptyset$; 17 foreach $i \in \{i \neq s : sh(i) < h(i)\}$ do 18 $c := (i, h(i))$; 19 if stack s can stabilize c then 20 $\mathbb{S}' := \mathbb{S}' \cup \{i\}$; 21 if $\mathbb{S}' \neq \emptyset$ then 22 $s' := \arg \max_{i \in \mathbb{S}'} g(i, h(i))$; 23 Move($s', 1, s$); 24 else 25 break while; </pre>
--	--

Here, function $\text{MoveNeed}(c, s)$ is the actual number of moves needed by the STAP to accomplish the task (c, s) . Function $\text{Fill}(s)$ is achieved by fulfilling s with those unstable containers in other stacks which s stabilizes.

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 literature, researchers usually use the same data sets when solving the CPMP [2,4,8,18] and the CRP [5,7,9,11,12].

6.1. Configuration of $P1$ and $P2$

This section shows the effect of the tier-protection indicator. We selected six settings of the two parameters $(P1, P2)$, which is $(0, N)$, $(H/4, N/2)$, $(H/3, N/6)$, $(2H/3, N/8)$, $(H, N/10)$. We made $P1(P2)$ proportional to $H(N)$, since $f(s)$ (affected demand) is related to height (number of containers).

The six settings are in a loose-intermediate-tight order, and algorithms will filter out more tasks when using back settings. The result is shown in Table 2. The first column indicates the parameter setting. The second and third columns indicate the average movements of CVS and BF data sets, respectively. Parameter $(0, N)$ means that there is no tier-protection indicator as no task is filtered out. As the setting becomes tight, the performance of both CVS and BF is improved, until a peak is achieved. But when the setting becomes extremely tight, the performance decreases. When the setting is $(H, N/10)$, some instances of CVS even cannot find a solution; therefore, we make the unit with “Nil”. The instances of BF are loose, that is, the empty slots are sufficient. Solutions can be found even if under setting $(H, N/10)$, but the performance is worse than that with intermediate setting.

Table 2: Configuration of $P1$ and $P2$

$(P1, P2)$	CVS	BF
$(0, N)$	52.15	71.69
$(H/4, N/2)$	50.43	71.21
$(H/3, N/4)$	45.82	69.05
$(H/2, N/6)$	45.32	69.51
$(2H/3, N/8)$	46.33	71.23
$(H, N/10)$	Nil	73.44

6.2. Results for CVS instances

Caserta et al. [5] present the complete CVS data set (named after the authors’ surnames, Caserta, Voß and Sniedovich) originally for the CRP. The CVS instances are classified into 21 groups,

each consisting of 40 instances. The stacks of the initial layout have the same height in a CVS instance. The number of containers per stack in the initial layout is denoted by K , hence $N = SK$. It is worth noting that the stack height limitation is not specified in the original data. The 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 3 illustrates the computational results on CVS instances by the TGH, the LGVFH, and the FBH. The values under the “move” headings represent average number of moves for every CVS group, whereas the values under the “time (ms)” headings are the run time in ms. The values under the “improvement” heading are the improvements compared to TGH and LGVFH in percentage. The results showcase that the FBH surpasses the benchmark heuristics by more than $10\% \times N$ moves on most instance groups. Especially on the narrowest group CVS 10-6, the FBH outperforms the TGH and the LGVFH by 319.38% and 55.54%, respectively.

6.3. Results for BF instances

Bortfeldt & Forster [2] 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 groups G 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 4 illustrates the computational results on BF instances by the three heuristics. The values under the move headings represent average number of moves for every BF group, whereas the values under the time (ms) headings are the run time in ms. The values under the improvement heading are the improvements compared to TGH and LGVFH in percentage.

Table 5 shows the computational results based on the dimension of density and stack height. It shows that the instance density N/SH (or bay utilization) and the height of the bay H are key factors 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 FBH outperforms these two benchmark heuristics.

Table 3: Results for CVS instances.

CVS K-S	TGH		LGVFH		FBH		improvement	
	move	time (ms)	move	time (ms)	move	time (ms)	TGH	LGVFH
CVS 3-3	12.95	0.43	11.25	0.45	11.28	0.53	12.93%	-0.22%
CVS 3-4	12.18	0.20	12.23	0.15	10.80	0.18	11.29%	11.66%
CVS 3-5	12.78	0.10	13.45	0.20	12.08	0.18	5.48%	10.22%
CVS 3-6	14.38	0.18	14.88	0.33	12.98	0.23	9.74%	12.77%
CVS 3-7	16.00	0.13	16.58	0.23	14.75	0.15	7.81%	11.01%
CVS 3-8	16.55	0.15	17.08	0.28	15.65	0.38	5.44%	8.35%
CVS 4-4	23.35	0.05	21.93	0.08	21.88	0.15	6.32%	0.23%
CVS 4-5	26.73	0.10	26.48	0.18	23.08	0.18	13.66%	12.84%
CVS 4-6	27.58	0.08	27.20	0.23	24.75	0.48	10.24%	9.01%
CVS 4-7	29.93	0.08	31.23	0.38	27.63	0.10	7.69%	11.53%
CVS 5-4	44.83	0.08	35.83	0.18	35.08	0.08	21.75%	2.09%
CVS 5-5	42.40	0.05	36.50	0.25	35.33	0.23	16.69%	3.22%
CVS 5-6	50.63	0.13	43.08	0.23	39.88	0.23	21.24%	7.43%
CVS 5-7	48.83	0.13	46.95	0.73	41.68	0.28	14.64%	11.24%
CVS 5-8	56.68	0.20	51.83	0.63	47.50	0.48	16.19%	8.35%
CVS 5-9	57.50	0.08	55.65	1.08	50.45	0.90	12.26%	9.34%
CVS 5-10	62.80	0.53	60.88	1.30	54.63	0.80	13.02%	10.27%
CVS 6-6	74.33	0.13	57.85	0.28	55.23	0.50	25.7%	4.54%
CVS 6-10	88.63	0.43	79.73	1.88	75.60	1.65	14.7%	5.17%
CVS 10-6	332.25	0.30	173.95	1.08	140.63	0.85	57.57%	19.16%
CVS 10-10	302.90	0.90	190.50	5.05	179.23	3.65	40.83%	5.92%
Average	64.48	0.21	48.81	0.72	44.29	0.58	16.44%	8.29%

Table 4: Results for BF instances.

BF	S	H	N	G	B	TGH		LGVFH		FBH		improvement	
						move	time (ms)	move	time (ms)	move	time (ms)	TGH	LGVFH
1	16	5	48	10	29	29.10	2.85	29.55	5.05	29.15	4.80	-0.17%	1.35%
2	16	5	48	10	36	36.00	1.35	36.60	2.90	36.00	3.65	0	1.64%
3	16	5	48	20	29	29.45	0.85	30.90	2.65	29.35	3.30	0.34%	5.02%
4	16	5	48	20	36	36.00	0.55	37.20	1.55	36.15	1.60	-0.42%	2.82%
5	16	5	64	13	39	48.50	1.55	53.00	2.90	46.30	2.65	4.54%	12.64%
6	16	5	64	13	48	57.55	0.95	62.85	3.45	55.50	2.65	3.56%	11.69%
7	16	5	64	26	39	53.55	0.85	57.15	3.10	49.95	2.25	6.72%	12.6%
8	16	5	64	26	48	60.00	0.80	66.90	2.90	57.60	2.40	4%	13.9%
9	16	8	77	16	47	60.35	1.55	62.15	4.35	56.30	3.90	6.71%	9.41%
10	16	8	77	16	58	62.15	7.90	69.50	4.65	61.55	3.85	0.97%	11.44%
11	16	8	77	31	47	61.25	1.15	63.70	4.20	55.00	3.35	10.2%	13.66%
12	16	8	77	31	58	63.45	1.10	68.50	4.15	61.45	3.30	3.15%	10.29%
13	16	8	103	21	62	107.45	1.75	110.85	6.45	96.50	5.55	10.19%	12.95%
14	16	8	103	21	78	124.75	2.55	134.90	5.95	116.05	4.75	6.97%	13.97%
15	16	8	103	42	62	110.60	2.90	110.40	9.45	99.45	6.50	10.08%	9.92%
16	16	8	103	42	78	133.35	1.55	137.40	7.40	115.45	5.85	13.42%	15.98%
17	20	5	60	12	36	36.50	0.90	37.35	2.30	36.50	2.15	0	2.28%
18	20	5	60	12	45	45.00	0.80	45.00	2.15	45.20	1.90	-0.44%	-0.44%
19	20	5	60	24	36	36.80	0.75	38.50	2.90	36.75	1.95	0.14%	4.55%
20	20	5	60	24	45	45.00	2.00	45.70	3.00	45.10	2.80	-0.22%	1.31%
21	20	5	80	16	48	61.65	1.65	65.65	4.10	56.55	3.25	8.27%	13.86%
22	20	5	80	16	60	67.90	1.30	74.50	4.10	65.55	3.25	3.46%	12.01%
23	20	5	80	32	48	61.10	1.40	65.75	4.75	55.25	4.05	9.57%	15.97%
24	20	5	80	32	60	70.95	1.95	76.65	5.80	68.00	4.25	4.16%	11.29%
25	20	8	96	20	58	69.80	3.90	73.60	8.60	66.00	6.75	5.44%	10.33%
26	20	8	96	20	72	74.35	3.60	81.75	7.55	75.75	7.15	-1.88%	7.34%
27	20	8	96	39	58	71.85	2.95	73.65	8.50	65.65	6.70	8.63%	10.86%
28	20	8	96	39	72	76.30	2.10	83.55	8.35	76.50	6.25	-0.26%	8.44%
29	20	8	128	26	77	118.65	3.60	128.65	11.15	115.85	8.80	2.36%	9.95%
30	20	8	128	26	96	143.05	3.60	155.15	11.05	129.60	9.75	9.4%	16.47%
31	20	8	128	52	77	128.15	3.30	128.80	15.15	115.85	10.85	9.6%	10.05%
32	20	8	128	52	96	147.30	2.70	157.00	13.60	134.10	10.45	8.96%	14.59%
Average						72.75	2.08	76.96	5.75	68.44	4.71	5.92%	11.08%

Table 5: Summary on BF instances.

density	TGH		LGVFH		FBH	
	$H = 5$	8	5	8	5	8
0.6	36.73	67.44	37.60	72.05	36.78	64.78
0.8	60.15	126.66	65.31	132.89	56.84	115.36

7. Conclusions

The container pre-marshalling deals with how to rehandle containers in a bay so that the containers are placed in a determined order. By far, only seven works talk about solutions to this problem. In this paper we present a feasibility-based heuristic to solve the container pre-marshalling problem. The proposed heuristic can be implemented solely or combined with other frameworks. The main innovation is the concept of state feasibility, which checks feasibility of states before really searching those states. Thanks to this concept, the search space is enlarged compared with extant methods, whereas the search efficiency is guaranteed. Numerical experiments on two commonly used data sets show that our proposed method outperforms other methods.

Our algorithm can be inserted into many heuristic frameworks such as GRASP, LNS and beam search. How these combinations work is worthy of study in the future. A major challenge raised in the feasibility-based heuristic is the trade-off between the freedom of selecting target containers and the waste of resource surplus. In this paper, the bottom tiers protection technique is proposed to balance the trade-off. As a future work, more efforts should be dedicated to better balance the trade-off. Moreover, the feasibility of a state is essentially based on a lower bound of current state. If tighter lower bound is found, the performance of the algorithm can be improved. Therefore, finding a better lower bound of current state is a good research question. In this paper, we assume that the containers in the bay are determined and no container comes in or goes out during the pre-marshalling. In the future work, it can be discussed that how to handle dynamic yards in which containers come in or go out from time to time. To some extent, the new setting is similar to container stacking problem, as the close relationship of container pre-marshalling and container stacking problem, the methods proposed for the container pre-marshalling problem may provide a new perspective for the container stacking problem.

Acknowledgments

This research was partially supported by the National Natural Science Foundation of China (Grant no. 71371114, 71301102).

References

- [1] Bierwirth, C. & 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.
- [2] Bortfeldt, A. & Forster, F. (2012). A tree search procedure for the container pre-marshalling problem. *European Journal of Operational Research*, 217(3), 531–540.
- [3] 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.
- [4] 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*, volume 5484 of *Lecture Notes in Computer Science* (pp. 788–797). Springer Berlin Heidelberg.
- [5] Caserta, M., Voß, S., & Sniedovich, M. (2011). Applying the corridor method to a blocks relocation problem. *OR Spectrum*, 33(4), 915–929.
- [6] Dayama, N. R., Krishnamoorthy, M., Ernst, A., Narayanan, V., & Rangaraj, N. (2014). Approaches for solving the container stacking problem with route distance minimization and stack rearrangement considerations. *Computers & Operations Research*, 52, Part A, 68 – 83.
- [7] 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.
- [8] 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.
- [9] Forster, F. & Bortfeldt, A. (2012). A tree search procedure for the container relocation problem. *Computers & Operations Research*, 39(2), 299–309.
- [10] Huang, S.-H. & Lin, T.-H. (2012). Heuristic algorithms for container pre-marshalling problems. *Computers & Industrial Engineering*, 62(1), 13 – 20.
- [11] 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*, volume 7906 of *Lecture Notes in Computer Science* (pp. 181–190). Springer Berlin Heidelberg.
- [12] 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.
- [13] 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*, volume 220 of *International Series in Operations Research & Management Science* (pp. 43–73). Springer International Publishing.
- [14] Lee, Y. & Chao, S.-L. (2009). A neighborhood search heuristic for pre-marshalling export containers. *European Journal of Operational Research*, 196(2), 468–475.

- 1
2
3
4 [15] Lee, Y. & Hsu, N.-Y. (2007). An optimization model for the container pre-marshalling problem. *Computers &*
5 *Operations Research*, 34(11), 3295–3313.
6
7 [16] Lehnfeld, J. & Knust, S. (2014). Loading, unloading and premarshalling of stacks in storage areas: Survey and
8 classification. *European Journal of Operational Research*, 239(2), 297–312.
9
10 [17] Wang, N. (2014). *Optimization Study on Container Operations in Maritime Transportation*. PhD thesis, City
11 University of Hong Kong, Hong Kong.
12
13 [18] Wang, N., Jin, B., & Lim, A. (2015). Target-guided algorithms for the container pre-marshalling problem.
14 *Omega*, 53, 67–77.
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65

LaTeX Source Files

[Click here to download LaTeX Source Files: gash.tex](#)