Dear Editor,

We would like to express our gratitude for the great effort you and the anonymous referees have put in. We found most of the critiques and feedback 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 have removed feasibility check method which was in Section 4.2, as it is not the original contribution of this paper; we have mentioned it in our another paper. In addition, it is not closely related to the key ideas expressed in this paper. In the new version, Section 4 only talks about proposed new lower bounds. Since only the algorithm TGH in Section 5 uses the concept of movable (immovable) containers, we have moved this concept (Section 4.1 in old version) to the first part of Section 5.

2. We have revised Section 1 and Figure 2 to illustrate the applications of CPMPDS.

3. We added two sections (Section 4.3 and Section 7.1) to substantiate why the proposed lower bounds dominate the existing lower bound of Bortfeldt and Foster.

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

Regards,

Ning Wang

Bo Jin

Andrew Lim

Reviewer 1

1. In section 4.2.1, on page 8, line about 48 (the numbers on the left do not match the lines of the text in my version): The authors say "If a stack has zero immovable container, then add *H*-*im* to element *SG*. Notation *im* is the number of immovable containers in a stack...". In other words *im* = 0 in this case. Lemma 1 says, that *im* is the same for all stacks. In my opinion, in this case it should be *SG* = H × S, because there are zero immovable containers in all stacks. The formulation used by the authors seems to be right but misleading.

Answer: As Reviewer 2 suggests, the feasibility check is not the original contribution of this paper; it has appeared in our another paper. And the feasibility issue is not closely related to the key ideas of this paper, so we have removed the section about feasibility from this paper. But the suggestion of Reviewer 1 is right, thus we have revised the paper discussing feasibility accordingly.

2. In section 4.3.2, on page 11, *LP* number 4: The authors write "*xi* ϵ {0, 1}; *i* = 1,…,S". In my opinion it should be either *xi* ϵ {0, 1}; *i* = {1,…,S} or *xi* ϵ {0, 1}; *i* = 1,…,S.

Answer: We have revised the error in typing this expression and similar ones.

3. The authors present a lower bound for (CPMP) and (CPMPDS). For (CPMP) their lower bound consists of three parts. Two of them are already presented by Bortfeldt and Foster. The third part is new. The authors claim, that their lower bound dominates the lower bound of Bortfeldt and Foster, but they do not substantiate this claim. Further, they propose a maximum knapsack method to approximate their third part of the lower bound. Again, they claim, that even their approximated lower bound dominates the one of Bortfeldt and Foster without any substantiation.

Answer: We have added a Section (Section 4.3) to substantiate why our proposed lower bounds dominate that of Bortfeldt and Foster.

4. The authors say nothing about the quality of the lower bound. In my opinion, they should provide the calculated lower bound in the computational study. Otherwise it is not possible to see how useful this bound is.

Answer: We have added the comparison of two proposed lower bound computation methods with that of Bortfeldt and Foster. The results on three data sets are displayed in Section 7.1.

5. All computational results (except the ones for CPMPDS) are compared with the results of Bortfeldt and Foster. The problem is, that the authors use a much faster PC than Bortfeldt and Foster. This makes the comparison of runtimes useless. In my opinion, the authors need to test the algorithm of Bortfeldt and Foster on the same machine to get a fair comparison.

Answer: From the machine perspective, the environment of Bortfeldt and Foster (BF for short) is Intel Core 2 Duo processor clocked at 2 GHz and the code is written in C; while the environment of this paper is Intel Core i7 CPU clocked at 3.40 GHz and the code is written in Java. As programs in Java run slower than those in C and the speed of our CPU is not faster than twice of BF’s CPU, it can be inferred that our code should be no faster than BF’s code by twice. In Table 3, Table 4, and Table 5 of our paper, the running time of BS-G is less than half of BF2012, which indicates the efficiency of our algorithms.

In order to make our conclusion more convincing, we ran BF’s code on our machine. Since we are unable to get BF’s code, we reimplemented BF’s algorithm by ourselves (the code can be provided upon request). The comparison of two codes on data set BF is shown in the table below.

Table: Comparison of codes by BF and us on data set BF

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| group | solved | our solution | our time (s) | BF’s solution | BF’s time (s) |
| BF1 | 20 | 29.1 | <1 | 29.1 | <1 |
| BF2 | 20 | 36 | <1 | 36 | <1 |
| BF3 | 20 | 29.1 | <1 | 29.1 | <1 |
| BF4 | 20 | 36 | <1 | 36 | <1 |
| **BF5** | **20** | **42.1** | **4.6** | **41.6** | **8** |
| **BF6** | **20** | **49.75** | **<1** | **49.4** | **4.9** |
| BF7 | 20 | 42.9 | 5.55 | 42.9 | 26.3 |
| BF8 | 20 | 51.05 | 11.8 | 50.6 | 24.7 |
| **BF9** | **20** | **51.6** | **13.75** | **51.8** | **30.8** |
| BF10 | 20 | 59.75 | 11.3 | 59.8 | 21.9 |
| **BF11** | **20** | **54.55** | **20.3** | **52.2** | **44.6** |
| **BF12** | **20** | **60.9** | **15.45** | **60.2** | **28.2** |
| **BF13** | **13** | **104.08** | **60** | **84.6** | **60** |
| **BF14** | **5** | **136.4** | **60** | **105.6** | **60** |
| **BF15** | **2** | **116.5** | **60** | **95.5** | **60** |
| **BF16** | **1** | **128** | **60** | **109.8** | **60** |
| BF17 | 20 | 36.3 | <1 | 36.3 | 3.5 |
| BF18 | 20 | 45 | <1 | 45 | <1 |
| BF19 | 20 | 36.45 | 2.85 | 36.5 | <1 |
| BF20 | 20 | 45 | <1 | 45 | <1 |
| BF21 | 20 | 51.6 | 15.3 | 51.7 | 31.5 |
| **BF22** | **20** | **61.1** | **5** | **60.9** | **7.6** |
| **BF23** | **20** | **51.85** | **23.2** | **51.5** | **25.5** |
| **BF24** | **20** | **61.45** | **17.5** | **61.3** | **18.2** |
| **BF25** | **20** | **64** | **36.45** | **62.8** | **45** |
| **BF26** | **20** | **78.5** | **31.6** | **74** | **30.8** |
| **BF27** | **20** | **70.9** | **54** | **64** | **55.9** |
| **BF28** | **20** | **77.5** | **35.05** | **74.9** | **44.7** |
| **BF29** | **9** | **138.56** | **60** | **106.6** | **60** |
| **BF30** | **7** | **170** | **60** | **128.5** | **60** |
| **BF31** | **4** | **138.75** | **60** | **115.2** | **60** |
| **BF32** | **3** | **168.67** | **60** | **132.3** | **60** |

Column ‘solved’ indicates the number of instances in a group that can be solved by our code. Columns ‘our solution’ (‘our time’) and ‘BF’s solution’ (BF’s time) are the average solutions (time) taken over solved instances by our and BF’s codes, respectively. Rows in bold indicate that the results by our and BF’s codes are inconsistent. The inconsistency exists because some details of the algorithm are not revealed in BF’s paper, and there may be optimization techniques implemented by BF when coding. From the table, it can be seen that result by BF’s code is better than that by our code. Disadvantage to BF does not exist when comparing our algorithms BS-G and BS-G with BF’s algorithm implemented by BF.

Furthermore, running time is not the main focus of CPMP. The quality measurement of CPMP algorithms is the ability to find better solutions in a reasonable time, i.e., 60s or the same scale. To this end, we remove the time comparison with BF’s algorithm in the new version of this paper.

Reviewer 2:

1. I actually doubt that the dummy stack extension of the CPMP (CPMPDS) can be applied in practice. The 'dummy stack' is placed at the truck lane that spans the whole length of a block. Since there is only one such lane (see Fig. 2), trucks cannot bypass a dummy container stack. Hence, it is impossible to serve any trucks at a block while the pre-marshalling is performed at one of the block's bays. I expect that this severely disturbs the operations of truck handling at a container terminal and that the pre-marshalling process may actually lower the productivity of the terminal.

Answer: The scenario mentioned by the reviewer is only one possible scenario. There are at least three scenarios where the CPMPDS is useful.

Scenario 1. The cranes are idle. Pre-marshalling is usually performed in cranes’ idle time, which has been mentioned by past literature.

Scenatio 2. In the truck lane, the crane which serves trucks (Crane A) is ahead of the crane which performs pre-marshalling (Crane B). When trucks get out of the block, pre-marshalling does not stand on their ways because truck lanes are unidirectional. In addition, the bays from which Crane A retrieves containers are near to each other and Crane A does not need to move to bays behind Crane B because containers destined for the same place are usually stored near to each other.

Scenario 3. There is more than one truck lane. According to the paper “Storage yard operations in container terminals: Literature overview, trends, and research directions” by Carlos et al. in EJOR, 2014 which is recommended by the reviewer 2, there may be more than one truck lane beside a block, which makes pre-marshalling possible while serving trucks.

2. According to the references, several parts of Section 4 stem from Wang et al. (2013). It needs to be stated more clearly, which parts of the material are reproduced from Wang et al. and which parts are actually new contributions of the OMEGA submission.

Answer: In the old version, the lower bound computation is the original contribution of this paper, while the feasibility check method is reproduced from Wang et al. (2013). As the feasibility check method is not closely related to the key ideas expressed in this paper, we decide to remove that part from this paper. Section 4 only talks about lower bounds. Since only the algorithm TGH in Section 5 uses the concept of movable (immovable) containers, we have moved this concept to the first part of Section 5.

3. The paper provides comprehensive computational experiments but, from my perspective, the most relevant research questions are not covered by the experimental evaluation:

* The authors propose new lower bounds and they claim that these bounds are better than those proposed by other researchers. However, the quality of these bounds is not evaluated against each other.

Answer: We have added the comparison of two proposed lower bound computation methods with that of Bortfeldt and Foster. The results on three data sets are displayed in Section 7.1.

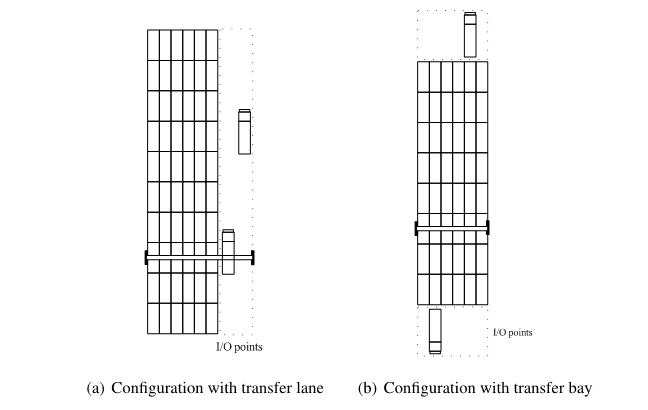
* The heuristics are compared with each other. An objective evaluation of their capabilities would require to take into account proven optimal solution or lower bounds. Actually, the bounds proposed in this paper do not appear at all in the computational study.

Answer: We are unable to obtain the optimal solutions to all instances due to the computational difficulty, so we add *LBDFS* of instances of and their gaps with resultant solutions in the last two columns of Table 3, Table 4, and Table 5, respectively, for the reference of solution quality. For BF data groups, BS-G and BS-B can solve six and eight groups to optimality as resultants solutions is equal to lower bounds.

* This paper proposed to extent the CPMP to the CPMPDS. However, there is no experiment, which shows how many container moves a terminal can actually save from switching to the more complex problem. Therefore, it remains an open question, whether (and to which extent) the CPMPDS provides a benefit for a terminal.

Answer: First, CPMPDS does exist in the real applications such as the three scenarios that we mentioned in the answer to Question 1 of Reviewer 2. In this regard, our intention of this paper is to depict real applications and solve them instead of extending CPMP to a more complex CPMPDS just for publication.

Second, seen from the terminal design perspective, Reviewer 2 suspects the benefits that CPMPDS provides. The reviewer may misunderstand the intention to solving CPMPDS. Actually, both CPMPDS and CPMP are solved before the final terminal layout is determined. As shown in the figure below, if the terminal layout is designed as in Figure (a), then CPMPDS is used; if the terminal layout is designed as Figure (b), then CPMP is used. When designing a terminal layout, it is definitely necessary to simulate the two layouts and then select the better one. That which one of the two layouts is better depends on many parameters, such as turnover rate, equipment efficiency, size of the terminal, traffic control etc.



Third, we compare the performance of two layouts in terms of the number of movements. TODO

Further comments:

1. I do not understand the meaning of the sentence 'Existing algorithms for the CPMP have not taken the dummy stack into consideration, hence they cannot be implemented directly at terminals using gantry cranes.'. Why is it impossible for these terminals to implement CPMP methods? They cannot benefit from the expected advantages of CPMPDS but, of course, they can use CPMP methods to improve terminal performance.

Answer: We expressed our intention in a wrong statement. We have revised the quoted sentence as ‘Existing algorithms for the CPMP have not taken dummy stacks into consideration, hence they cannot be implemented directly at terminal layouts like Figure 2(a).’ We also revised the sixth paragraph of Section 1 and Figure 2 to make our presentation more clear.

2. Is the beam search significantly different from the corridor method proposed by Caserta & Voss (2009)? Actually, both methods do restrict the 'sight field' when looking for a promising container movement, don't they?

Answer: As we have described in the second paragraph of Section 2, Caserta & Voß (2009) (CV for short) provided a greedy heuristic based on the paradigm of corridor and roulette wheel. The corridor reduces movement choices for a certain layout and the roulette wheel provides randomness when making choices. The probability of selecting an alternative is proportional to its attractiveness. The algorithm first builds a corridor with respect to the current layout to determine the destination stacks of a specific misplaced container. New layouts are then yielded by conducting the movements in the corridor. The attractiveness of each new layout is evaluated by an estimated number of needed relocations. A local improvement scheme is also conducted to accelerate the search process.

In depth, the following components of CV and our algorithms are different.

1. Structure: CV’s algorithm is a greedy algorithm which expands one node each time; in our paper, we propose one greedy algorithm and then combine it with beam search frameworks.

In the following, we compare the difference between CV’s and our greedy algorithm (TGH).

2. Next node: in CV’s, the next node is a movement which relocate one topmost container to the top of another stack; in TGH, the next node is a giant move (several movements) which aims at making a container (not necessary topmost) clean; this container can be moved to the same stack as its original stack.

3. Selection of next node: in CV’s, a certain container is determined, and then possible destination stacks are evaluated and selected based on the determined container; in TGH, container and stack are regarded as a pair; they are evaluated and selected as a whole.

3.1. Container selection: in CV’s, roulette-wheel mechanism is used to randomly select the topmost container of a stack such that the probability of selecting a stack is proportional to the number of dirty containers within the stack itself. The determined container must in the next node. In TGH, the target is clearer, that is, only containers with largest group labels can be the next node. A set of such containers are selected for further evaluation.

3.2. Candidate stack selection: in CV’s, a set of stacks is selected as candidate stacks based on corridor which in essence is an evaluation function; in TGH, candidate container and stack are evaluated as a whole be another evaluation function different from that of CV’s.

3.3. Selection of next node: in CV’s, they grade the determined container and each stack in the candidate set by the number of dirty containers. Then only the first half stacks with highest scores are allowed to enter roulette-wheel procedure, in which the next node is determined by the roulette-wheel. In TGH, no roulette-wheel is involved. The container stack pair with the highest score is selected as the next node.

4. Other techniques: CV’s deploys two local search schemes to make sequences in stacks sorted. In our TGH, we devised a fulfillment technique to achieve similar intention. In addition, as one expansion in TGH is a giant move, we also spare our effort in designing an efficient giant move.

5. Theoretical conclusion: in CV, the authors call readers’ attention that their algorithm does not guarantee a solution, whereas TGH guarantees a final solution.

From the above analysis, we can see that our algorithm TGH is significantly different from that of CV. To be honest, we use the idea of ‘sight filed’ in our algorithm, but it is a common practice for algorithms solving large scale problems, or it is hard to narrow the solution space without it. Frameworks used in large scale problems are similar, which highlights the importance of effective detailed components in the frameworks.

3. The sentence 'A heuristic algorithm and two advanced beam search algorithms are elaborated in Section 5 and Section 6 respectively.' sounds like the beam search wouldn't be a heuristic. However, since these methods are also heuristics, the sentence needs to be rephrased.

Answer: We have rephrased the above sentence as ‘A greedy algorithm and two advanced beam search algorithms are elaborated in Section 5 and Section 6, respectively.’ in the paper.

4. The recent survey 'Storage yard operations in container terminals: Literature overview, trends, and research directions' by Carlos et al. in EJOR, 2014, might be relevant for this research and should be included into section 2.

Answer: We have added the information provided by the above reference in Section 1 and Section 2.

5. Figures 3, 4 and 5 are used for illustrating the problem. However, there is hardly any link between these figures and the text. I recommend to stronger connect them by either adding labels to the figures (e.g. to show H, G, g(c) etc. in Fig. 3) or to add more details to the text (e.g. saying that H=6 in Figure 3.). This would help readers to follow the verbal explanations.

Answer: We have added more explanation of Figure 3 in the text as ‘Figure 3 shows a layout with *S*=5, *H*=6, *G*=9 and duplicate group labels’. In the old version, Figure 4 explained Theorem 1, we have removed Theorem 1 and Figure 4 in the new version. Figure 5 of the old version becomes Figure 4 of the new version. We have rewritten the explanation of new Figure 4 as ‘For example, in Figure 4, the bold line presents a skyline of the layout with *tn*(*SL*2) =3 and g(*SL*2)=1.’

6. In Figure 4 there are cases 1 and 2, but I did not find a reference to these two cases in the text.

Answer: We have removed Figure 4 as it is not closely related to the content of this paper.

7. Section 4.2.1: In order to make sure that an instance is feasible, it has to hold that the immovable containers are all clean already in the initial layout. Is this ensured by (2) or is this an additional requirement which needs to be mentioned in 4.2.1?

Answer: We have removed Section 4.2 in the old version as it’s not the original contribution of this paper.

8. Since set AS is introduced on page 14, line 13, I recommend to use '\sum\_{s \in AS}' in the next line.

Answer: We have revised the paper as suggested by Reviewer 2 in Section 5.3.1 and Section 5.3.2.

9. The concept of 'fulfillment' (page 16-17) is not clear to me. Maybe, this part can be rewritten to make the idea more clear.

Answer: We have rewritten Section 5.4 to make ‘fulfillment’ clearer. The main idea is: after selecting a destination *s* for relocating *c*, if *s* is clean and can be clean after placing *c* on top of it, an extra action called ‘fulfillment’ is carried out. Denote the topmost container of *s* as *dc*. If there exists a set of topmost dirty containers with their group labels between *g*(*c*) (exclusive) and *g*(*dc*) (inclusive), the one with the largest group label is selected and moved to the destination stack. If fulfillment is carried out successfully, relocating *c* to *s* is stopped, and the relocation procedure for *c* is invoked again.

10. Section 6: Please describe clearly, whether the two beam search methods can be applied to both, the CPMP and the CPMPDS, as described or whether there are any modifications required if switching from one problem to the other.

Answer: We have mentioned that our algorithms can be applied to solve both CPMP and CPMPDS in Section 6.3, which is the difference of our algorithms with existing algorithms. In the new version, we also add the applicability of our algorithms in the first paragraph of Section 6.

11. Page 21: The phrase '… solved to the optimality …' is weird. First, there are no lower bounds reported which justify this statement. Second, Fig. 8 reports even lower values than those in Fig. 7.

Answer: We have added lower bounds and gaps in Table 5. Second, the objective of CPMP/CPMPDS is to minimize number of movements, therefore, the smaller the value, the better. Figure 7 and Figure 8 (Figure 6 and Figure 7 in the new version) report the results by BS-G (giant move) and BS-B (baby move), respectively. In general, BS-B has better performance than BS-G, therefore that values of Figure 8 is smaller is conventional.

12. I'm not a native speaker but I've got the impression that this paper could benefit from checking grammar and language. There are also a couple of obvious typos (page 12: 'lest' > 'least', page 15: 'snlot' > 'nslot', page 19: 'bench mark' > 'benchmark').

Answer: We have hired an English editor to proofread the manuscript.