VNIVERSITAT DE VALÈNCIA

Facultat de Ciències Matemàtiques

Departament d'Estadística i Investigació Operativa

Programa de doctorat en Estadística i Optimització

DOCTORAL THESIS

# Improving container terminal efficiency: New models and algorithms for Premarshalling and Stowage Problems

*Author:*

CONSUELO PARREÑO TORRES

*Supervised by:*

Dr. Ramón Álvarez-Valdés Olaguíbel
Dr. Rubén Ruiz García

April 2020

*To them, my heaven and
my earth, my parents.*

# Resumen

El desarrollo del contenedor ha revolucionado el comercio marítimo de mercancías, permitiendo la manipulación de carga de diversos tipos y dimensiones con un costo reducido y disminuyendo el costo de importación de muchos productos, En la actualidad, aproximadamente el 90% de la carga no a granel en todo el mundo se transporta en buques portacontenedores, cuyas capacidades han llegado a sobrepasar los 20000 TEUs (*Twenty-foot Equivalent Unit*, unidad de medida correspondiente a un contenedor normalizado de 20 pies).

Las terminales de contenedores tienen que hacer frente al creciente volumen de carga transportada, al aumento del tamaño de las naves y a las alianzas de las navieras. En este contexto, deben competir por menos servicios de barcos cada vez más grandes. Para ello, deben aumentar su eficiencia, optimizando los recursos existentes. En esta tesis se estudian dos problemas de optimización combinatoria, el problema de premarshalling y el problema de la estiba, que surgen en el patio y en el muelle de las terminales de contenedores, antes y durante las operaciones de carga y descarga de los buques, y cuya resolución deriva en una disminución del tiempo de atraque y, por lo tanto, en un aumento de la eficiencia de las terminales.

El problema de premarshalling prepara el patio de contenedores antes de la llegada del buque, usando las grúas de patio cuando la carga de trabajo es mínima, con el fin de evitar un mayor número de recolocaciones a la llegada del buque y así acelerar los tiempos de servicio. El objetivo clásico de este problema ha sido reducir al mínimo el número de movimientos necesarios para eliminar los contenedores que bloquean la retirada de otros dentro de una bahía. De este modo, el número de movimientos se ha tomado como un indicador del tiempo de grúa. No obstante, en esta tesis se prueba que considerando como objetivo el tiempo real que la grúa emplea en realizar los movimientos, se puede reducir hasta un 24% el tiempo total empleado. Para la resolución de ambos problemas, el premarshalling con función objetivo clásica y el premarshalling con la nueva función objetivo, se han desarrollado diversos modelos matemáticos y algoritmos Branch and Bound con nuevas cotas superiores e inferiores, reglas de dominancia y algoritmos heurísticos integrados en el proceso de ramificación.

Por lo que respecta al problema de la estiba, se ha estudiado el problema multi-puerto que busca obtener un plan de estiba del barco de modo que se reduzca al mínimo el número total de movimientos improductivos en las operaciones de carga y descarga a lo largo de la ruta en la que presta servicio. Comenzamos estudiando el problema simplificado, en el que no se consideran restricciones de tamaño ni de peso de los contenedores, y progresivamente se van introducido restricciones más realistas, desarrollando modelos matemáticos, heurísticas, metaheurísticas y mateheurísticas. Estos procedimientos son capaces de resolver instancias de gran tamaño correspondientes a los barcos de mayor capacidad que se encuentran actualmente en el sector.

# Resum

El desenvolupament del contenidor ha revolucionat el comerç marítim de mercaderies, permetent la manipulació de càrrega de diversos tipus i dimensions amb un cost reduït i disminuint el cost d'importació de molts productes fins al punt que, de vegades, és més barat transportar les mercaderies a l'altra banda del món que produir-les localment. En l'actualitat, aproximadament el 90% de la càrrega no a granel a nivell mundial es transporta en vaixells portacontenidors, amb capacitats que han arribat a sobrepassar 20000 TEUs (*Twenty-foot Equivalent Unit*, unitat de mesura corresponent a un contenidor normalitzat de 20 peus).

Les terminals de contenidors han de fer front al creixent volum de càrrega transportada, a l'augment de les dimensions de les naus, i a les aliances de les navilieres. En aquest context, han de competir per menys serveis de vaixells cada vegada més grans. Per això, han d'augmentar la seva eficiència, optimitzant els recursos existents. En aquesta tesi s'estudien dos problemes d'optimització combinatòria, el problema de premarshalling i el problema de l'estiba, que apareixen al pati i al moll de les terminals de contenidors, abans i durant les operacions de càrrega i descàrrega dels vaixells. La seva resolució deriva en una disminució del temps d'atracada i, per tant, en un augment de l'eficiència de les terminals.

El problema de premarshalling prepara el pati de contenidors abans de l'arribada del vaixell, usant les grues de pati quan la càrrega de treball és mínima, per tal d'evitar un major nombre de recol·locacions quan arriba el vaixell i així accelerar els temps de servei. L'objectiu clàssic d'aquest problema ha sigut reduir al mínim el nombre de moviments necessaris per eliminar els contenidors que bloquegen la retirada d'altres dins d'una badia. D'aquesta manera, el nombre de moviments s'ha pres com un indicador del temps de grua. No obstant això, en aquesta tesi es prova que considerant com a objectiu el temps real que la grua empra al realitzar els moviments, es pot reduir fins a un 24 % el temps total emprat. Per a la resolució de tots dos problemes, el premarshalling amb funció objectiu clàssica i el premarshalling amb la nova funció objectiu, s'han desenvolupat diversos models matemàtics i algorismes Branch and Bound amb noves cotes superiors i inferiors, regles de dominància i algoritmes heurístics integrats en el procés de ramificació.

Pel que fa al problema de l'estiba, s'ha estudiat el problema multi-port que busca obtenir un pla d'estiba del vaixell per reduir al mínim el nombre total de moviments improductius en les operacions de càrrega i descàrrega al llarg de la ruta en la qual presta servei. Començant amb un problema simplificat, en el qual no es consideren restriccions de mida ni pes dels contenidors, s'han introduït progressivament restriccions més realistes, desenvolupat models matemàtics, heurístiques, metaheurístiques i mateheurístiques. Aquests procediments són capaços de resoldre instàncies de grans dimensions, els quals corresponen a vaixells de major capacitat que es troben actualment en el sector.

# Abstract

The development of containers has revolutionized maritime trade by making it possible to handle various types and sizes of cargo at a reduced cost, lowering the import cost of many products to such an extent that it is sometimes cheaper to transport goods to the other side of the world than to produce them locally. Nowadays, about 90 per cent of non-bulk cargo worldwide is carried on container ships with capacities exceeding 20,000 TEUs (Twenty-foot Equivalent Units).

Container terminals have to cope with the increase in the volumes of cargo transported, the ever-larger ships, and the consolidation of shipping companies. In this context, they have to compete for fewer calls of larger ships. Since they cannot simply increase the number of cranes indefinitely, they have to improve efficiency by optimizing the available resources. This thesis studies two combinatorial optimization problems, the premarshalling problem and the stowage problem. These problems arise in the yard and the seaside of container terminals, before and during the loading and unloading operations of the ships, and make it possible to reduce the berthing time and thus to increase container terminal efficiency.

The premarshalling problem prepares the container yard before the arrival of the ship, using the yard cranes when the workload at the terminal is at a minimum to rearrange the yard in order to avoid container relocations when the vessel arrives and to speed up the service times. The classic objective of this problem is to minimize the number of movements required to remove containers blocking the retrieval of others within a bay. Thus, the number of movements has been used as an indicator of crane time. However, this thesis shows that considering the real time that the crane takes to perform the movements as the target, the total time spent by the crane can be cut down up to 24 per cent. To solve both problems, premarshalling with the classic objective function and premarshalling with the new objective function, this thesis develops several mathematical models and branch and bound algorithms with new upper and lower bounds, dominance rules and heuristic algorithms integrated in the branching process.

With regard to the stowage problem, the multi-port problem is addressed, seeking to obtain a stowage plan for the ship so as to minimize the total number of unproductive moves in the loading/unloading operations along the trade route of the ship. We start with a simplified problem, in which no size and weight constraints are considered, and progressively introduce more realistic constraints, developing mathematical models, metaheuristics, and matheuristics. These procedures are able to solve very large instances, corresponding to the largest ships in service.

# Acknowledgements

The research presented in this thesis has been carried out under FPU Grant A-2015-12849, awarded by the Spanish Ministry of Science, Innovation, and Universities, and under projects DPI2014-53665-P, DPI2015-65895-R, and “OPTEP-Port Terminal Operations Optimization” (RTI2018-094940-B-I00), financed with FEDER funds. Many people have been directly or indirectly involved with the thesis during the course of its preparation and I would like to express my gratitude to them all.

First, I want to thank my supervisors, Ramón Álvarez-Valdés and Rubén Ruiz, for their unconditional support and unwavering trust in me and my abilities. They have been a perfect tandem and I cannot imagine a better team, on a scientific and personal level, to supervise a thesis. Many thanks, especially, to Ramón for his generosity and guidance and for always having good advice to offer, or a *"You can do whatever you want, but I..."*. I hope that one day I may be able to inspire someone with that love of research and teaching that you have instilled in me over all these years.

I would like to thank all the members of the Department of Statistics and Operations Research at the University of Valencia, and particularly my fellow doctoral students, those who have already finished and those who are still in the process of doing so. They have all made my daily work more enjoyable. This department is a family, with great scientists always giving of their best. I would also like to thank all the SOA group at the Polytechnic University of Valencia, and especially Gerardo, whose technical support has been indispensable for the completion of this thesis. I am very grateful to the supervisors of the research visits I have made while working on this project: Prof. Dr. Kevin Tierney and Prof. Dr. Leena Suhl at the University of Paderborn, Germany, and Prof. Dr. Greet Vanden Berghe and Dr. Hatice Çalik at the KU Leuven in Ghent, Belgium. They received me kindly and made me feel like one of their own students. Thanks to Kevin and Hatice for their contribution to this thesis; it has been a pleasure to meet and work with them. Thanks also to every member of the CODeS group, whose high personal and academic quality is a perfect reflection of their leader and her careful choice of each person she works with. Among the wonderful people I met during my research is Ari, who was, is, and always will be Paderborn to me.

Finally, thanks to everyone reading this document, and apologies to any-one who does not understand Spanish, but I want to end by expressing my gratitude to my family and friends in our mother tongue, so that it reaches each and every one of them.

Si alguien merece el mayor de los agradecimientos, esa es mi madre, gracias a la cual hoy estoy escribiendo estas líneas. Sinónimo de lucha y superación, de mujer fuerte, ha hecho frente a todas las adversidades con una entereza envidiable. Siempre ha sido mi mayor motor y si bien ella lo ha dado todo por mí, yo lo daría todo por ella. Gracias a mi padre, porque de mi corazón nunca se ha marchado, su memoria me hace mejorar y desear que pueda sentirse orgulloso de cómo he crecido y las metas que he alcanzado. Gracias también a mis hermanos. A Fran, porque si para mí eres un ejemplo como investigador, lo eres mucho más como persona, gracias por tu contribución en esta tesis a nivel científico y personal. A Alfon, por escucharme cuando no sé qué hacer y tener siempre un buen consejo, aunque hayamos crecido y ahora los viajes los hagas solo, sigo sin querer que te vayas de mi lado. Estoy muy orgullosa de vosotros y no puedo sentirme más agradecida de la familia que tengo. Gracias, gracias, gracias y gracias. Cómo olvidarme de los que llenan la casa de vitalidad, risas y alegría, mis sobrinos. Ellos te sacan una sonrisa en cualquier momento. Gracias Alfonso, María, Héctor, Pablo, Darío y Nora, os quiero muchísimo. Tampoco olvidarme de mis cuñadas, abuela, tíos, primos y demás familia. Todos vosotros hacéis aún más grande el concepto de familia.

Agradecer también a mis amigos de toda la vida, los que siempre han estado y siempre estarán ahí. Aunque cada uno esté en una ciudad distinta, Villarrobledo sigue siendo un punto clave para encontrarnos y que parezca que no ha pasado el tiempo. Gracias también a todos los amigos que la bonita ciudad de Valencia me ha regalado. A vosotras, chicas, gracias por entender y compartir mi veneno. Por último, gracias al mejor amigo, compañero de trabajo y, sobre todo, compañero de vida. Él es el encargado de alegrar todos y cada uno de mis días, levantándome en los días más tensos, pero también compartiendo conmigo los momentos más bonitos. Gracias Reu, porque tan solo pensar en ti me hace feliz.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **TEU** | 20-foot equivalent unit |
| **FEU** | 40-foot equivalent unit |
| **CPMP** | Container premarshalling problem |
| **CPMPCT** | Container premarshalling problem with crane time minimization objective |
| **CSPP** | Container stowage planning problem |
| **MBPP** | Master bay planning problem |
| **MP-MBPP** | Multi-port master bay planning problem |
| **SPP** | Slot planning problem |
| **ILP** | Integer linear programming |
| **MILP** | Mixed integer linear programming |
| **GRASP** | Greedy randomized adaptive search procedure |
| **RF** | Relax-and-Fix |
| **IF** | Insert-and-Fix |
| **FRF** | Fractional Relax-and-Fix |

# List of Premarshalling notation

## Common notation:

| | |
|---|---|
| $\mathcal{C}$ | Set of containers. |
| $\mathcal{S}$ | Set of stacks. $S := \|\mathcal{S}\|$ is the total number of stacks. |
| $\mathcal{H}$ | Set of tiers. $H := \|\mathcal{H}\|$ represents the highest tier of the stacks. |
| $\mathcal{P}$ | Set of priorities or group values. 1 is the highest priority and $P := \|\mathcal{P}\|$ the lowest priority. |
| $T$ | Upper bound for the number of moves. |

## Chapter 2 notation:

| | |
|---|---|
| $n_p$ | Number of containers of priority $P$. |
| $\kappa$ | Maximum difference in the numbers of containers in adjacent stacks. |
| $x_{shpt}$ | Binary decision variable, set to 1 if at time point $t$ there is a container in position $(s, h)$ whose priority is $p$. |
| $y_{skt}$ | Binary decision variable, set to 1 if a container is moved from stack $s$ to $k$ in the time segment between $t$ and $t + 1$. |
| $y_{skpt}$ | Binary decision variable, set to 1 if a container with priority $p$ is moved from stack $s$ to $k$ in the time segment between $t$ and $t + 1$. |
| $y_{shket}$ | Binary decision variable, set to 1 if a container is moved from stack $(s, h)$ to $(k, e)$ in the time segment between $t$ and $t + 1$. |
| $y_{shkept}$ | Binary decision variable, set to 1 if a container with priority $p$ is moved from stack $(s, h)$ to $(k, e)$ in the time segment between $t$ and $t + 1$. |
| $w_{st}$ | Binary decision variable, set to 1 if a container is moved from stack $s$ in the time segment between $t$ and $t + 1$. |
| $z_{st}$ | Binary decision variable, set to 1 if a container is moved to stack $s$ in the time segment between $t$ and $t + 1$. |
| $w_{spt}$ | Binary decision variable, set to 1 if a container with priority $p$ is moved from stack $s$ in the time segment between $t$ and $t + 1$. |
| $z_{spt}$ | Binary decision variable, set to 1 if a container with priority $p$ is moved to stack $s$ in the time segment between $t$ and $t + 1$. |

| | |
|---|---|
| $w_{sht}$ | Binary decision variable, set to 1 if a container is moved from stack $(s, h)$ in the time segment between $t$ and $t + 1$. |
| $z_{sht}$ | Binary decision variable, set to 1 if a container is moved to stack $(s, h)$ in the time segment between $t$ and $t + 1$. |
| $w_{shpt}$ | Binary decision variable, set to 1 if a container with priority $p$ is moved from stack $(s, h)$ in the time segment between $t$ and $t + 1$. |
| $z_{shpt}$ | Binary decision variable, set to 1 if a container with priority $p$ is moved to stack $(s, h)$ in the time segment between $t$ and $t + 1$. |
| $h_{min}$ | Integer decision variable, minimum number of containers in a stack at time $T$. |
| $h_{max}$ | Integer decision variable, maximum number of containers in a stack at time $T$. |
| $g_{shp}$ | Binary decision variable, set to 1 if a container of priority $p$ located at $(s, h)$ at time $T$ is on top of other container of priority $p$. |

# Chapter 3 notation:

| | |
|---|---|
| $S^{\mathrm{M}}$ | Set of misoverlaid stacks. |
| $S^{\mathrm{minM}}$ | Set of misoverlaid stacks with the minimum number of misoverlaying containers. $S^{\mathrm{minM}} := \{s \in S \mid n_s^{\mathrm{M}} = f^{\mathrm{M}}\}$. |
| $S^{\mathrm{N}}$ | Set of non-misoverlaid stacks. |
| $U$ | Set of stacks where the misoverlaying containers are "upside down" sorted, i.e., $\mathit{group}(s, h) \leq \mathit{group}(s, h + 1)$ for all misoverlaying tiers $h$. $U := \{s \in S^{\mathrm{M}} \mid n_s^{\mathrm{BG}} = 1\}$. |
| $U'$ | Set of stacks that would be upside down if one misoverlaying container were removed. We assume $U \subset U'$. |
| $n_s$ | Number of containers in stack $s$. |
| $n_s^{\mathrm{M}}$ | Number of misoverlaying containers in stack $s$. |
| $n_s^{\mathrm{N}}$ | Number of non-misoverlaying containers in stack $s$. |
| $n^{\mathrm{M}}$ | Total number of misoverlaying containers in the bay. |
| $f^{\mathrm{M}}$ | Minimum number of misoverlaying containers in misoverlaid stacks. $(f^{\mathrm{M}} := \min_{i \in S^{\mathrm{M}}} n_s^{\mathrm{M}})$. |
| $p_s^{\mathrm{top}}$ | Group value of the topmost container in stack $s$. If stack $s$ is empty, $p_s^{\mathrm{top}} := \lvert\mathcal{P}\rvert = P$. |
| $c_s^{\mathrm{top}}$ | Topmost container in stack $s$. If stack $s$ is empty, $c_s^{\mathrm{top}} = \varnothing$. |
| $m_s$ | Largest group value of the misoverlaying containers in stack $s \in S^{\mathrm{M}}$. If $s \in U$, $m_s = p_s^{\mathrm{top}}$. |
| $m_s^i$ | The group value of the $i$th largest misoverlaying group value in stack $s$, e.g., $m_s^1 = m_s$, $m_s^2$ is the second largest misoverlaying group value, etc. |
| $w_s$ | Smallest group value of non-misoverlaying containers in stack $s$. If stack $s$ is empty, $w_s := \lvert\mathcal{P}\rvert = P$. If $s \in S^{\mathrm{N}}$, $w_s = p_s^{\mathrm{top}}$. |

| | |
|---|---|
| $w_s^{\text{sec}}$ | Second smallest group value of non-misoverlaying containers in stack $s$. If all the non-misoverlaying containers in stack $i$ have the same group, $w_s^{\text{sec}} := \lvert\mathcal{P}\rvert = P$. |
| $n_s^{\text{BG}}$ | Minimum number of non-misoverlaid stacks necessary for repairing stack $s$ with only BG moves. |
| $p_{si}^{\text{BG}}$ | Minimum group value of the topmost container of the $i$th non-misoverlaid stack for repairing stack $s$. Sorted as $p_{s1}^{\text{BG}} \geq p_{s2}^{\text{BG}} \geq \cdots \geq p_{sn_s^{\text{c}}}^{\text{BG}}$. We assume $p_{si}^{\text{BG}} = 0$ for $i > n_s^{\text{BG}}$. |

## Chapter 4 notation:

| | |
|---|---|
| $ch$ | Container height [m]. |
| $cw$ | Container width [m]. |
| $vsep$ | Distance between the top level and the container at the topmost tier [m]. |
| $hsep$ | Distance between the first stack and the truck [m]. |
| $msep$ | Distance between containers [m]. |
| $vmax^{vl}$ | Hoisting speed loaded [m/s]. |
| $vmax^{vu}$ | Hoisting speed unloaded [m/s]. |
| $vmax^{rl}$ | Travelling speed loaded [m/s]. |
| $vmax^{ru}$ | Travelling speed unloaded [m/s]. |
| $d^v$ | Distance to max. hoisting speed [m]. |
| $d^r$ | Distance to max. travelling speed [m]. |
| $b_h$ | Twistlock time at level $h$ [s]. |
| $a^{\alpha\beta}$ | Acceleration of the vertical/horizontal, i.e, $v/h$, unloaded/unloaded, i.e, $u/l$, move. $\alpha \in \{v, h\}$ and $\beta \in \{u, l\}$. |
| $c_{sk}^0$ | Time spent to move the crane unloaded along the upper travel line from stack $s$ to $k$. |
| $c_{sk}^1$ | Time spent to move the crane loaded along the upper travel line from stack $s$ to $k$. |
| $v_h^0$ | Time spent to move the crane vertically loaded from/to the upper travel line to/from tier $h$. |
| $v_h^1$ | Time spent to move the crane vertically unloaded from/to the upper travel line to/from tier $h$. |
| $v_h$ | $v_h^0 + v_h^1$ |
| $t_{min}$ | Lower bound for the time spent to perform a move. |
| $\gamma$ | Number of blocking containers in the initial layout of the bay. $\eta_1, \ldots, \eta_\gamma$ are the tiers where they are initially stored and $\tau_1, \ldots, \tau_\gamma$ are the $\gamma$ highest tiers to which the $\gamma$ blocking containers in the initial layout of the bay could be moved on top of non-blocking containers without considering blocking containers. |
| $x_{shpt}$ | Binary decision variable, set to 1 if at point $t$ there is a container in position $(s, h)$ whose priority is $p$. |

| | |
|---|---|
| $w_{shpt}$ | Binary decision variable, set to 1 if a container with priority $p$ is moved from stack $(s, h)$ in the segment between $t$ and $t + 1$. |
| $z_{shpt}$ | Binary decision variable, set to 1 if a container with priority $p$ is moved to stack $(s, h)$ in the segment between $t$ and $t + 1$. |
| $l_{skt}$ | Binary decision variable, set to 1 if there is a move from stack $s$ to $k$ in the segment between $t$ and $t + 1$. |
| $u_{skt}$ | Binary decision variable, set to 1 if in the segment between $t$ and $t + 1$ there is a move from stack $k$ and the destination of the previous move was $s$. |

# List of Stowage notation

## Common notation:

| | |
|---|---|
| $\Omega$ | Container ship capacity, expressed in terms of the total number of slots. |
| $\mathcal{N}$ | Index set of ports. $\mathcal{N} = \{1, \ldots, N\}$. |
| $\mathcal{B}$ | Index set of bays. $\mathcal{B} = \{1, \ldots, B\}$. |
| $\mathcal{C}^b$ | Index set of rows in bay $b$. $\mathcal{C}^b = \{1, \ldots, C^b\}$. |
| $\mathcal{S}$ | Index set of stacks on the ship. $\mathcal{S} = \{1, \ldots, S\}$ where $S = \sum_{b \in \mathcal{B}} \sum_{c \in \mathcal{C}^b} (b \times c)$. |
| $\mathcal{R}^{bc}$ | Index set of tiers in bay $b$, row $c$. $\mathcal{R}^{bc} = \{1, \ldots, R^{bc}\}$ |
| $\mathcal{R}$ | Index set of tiers when the number of tiers is constant for each bay and row. $\mathcal{R} = \{1, \ldots, R\}$. |

## Chapter 5 notation:

| | |
|---|---|
| $T_{ij}$ | Number of containers going from port $i$ to port $j$. |
| $x_{ijrs}$ | Binary decision variable, set to 1 if a $j$-container is stowed in slot $(r, s)$ after loading operations at port $i$. |
| $y_{ijrs}$ | Binary decision variable, set to 1 if a $j$-container stowed in slot $(r, s)$ at port $i - 1$ is unproductively moved at port $i$. |

## Chapter 6 notation:

| | |
|---|---|
| $\mathcal{O}$ | Ports where loading operations can occur. $\mathcal{O} = \{1, \ldots, N-1\} \subseteq \mathcal{N}$ |
| $\mathcal{O}^r$ | Ports where relocations can occur. $\mathcal{O}^r = \{2, \ldots, N-1\} \subseteq \mathcal{O} \subseteq \mathcal{N}$ |
| $\mathcal{D}^i$ | Set of possible destination ports for containers loaded at port $i$. $\mathcal{D}^i = \{i+1, \ldots, N\}$ |
| $\mathcal{P}$ | Set of cells. $\mathcal{P} = \{p = (b, c, r) : b \in \mathcal{B}, c \in \mathcal{C}^b, r \in \mathcal{R}^{bc}\}$ |
| $\mathcal{X}$ | Index set of slots. $\mathcal{X} = \{1, 2\}$ |
| $\mathcal{HP}$ | Set of slots belonging to the bow end. |
| $\mathcal{LP}$ | Set of slots belonging to the stern end. |
| $\mathcal{WP}$ | Set of slots belonging to the starboard side. |
| $\mathcal{YP}$ | Set of slots belonging to the port side. |

| | |
|---|---|
| $\mathcal{A}$ | Areas of the ship, depending on their longitudinal and lateral position: port quarter (back left), starboard quarter (back right), port bow (front left), and starboard bow (front right). $\mathcal{A} = \{\{\mathcal{LP}\} \cap \{\mathcal{YP}\}, \{\mathcal{LP}\} \cap \{\mathcal{WP}\}, \{\mathcal{HP}\} \cap \{\mathcal{YP}\}, \{\mathcal{HP}\} \cap \{\mathcal{WP}\}\}, \quad \lvert\mathcal{A}\rvert = 4$ |
| $\mathcal{M}$ | Set of weight classes (light, medium and heavy). |
| $\mathcal{S}$ | Set of container sizes (20' and 40'). |
| $\mathcal{T}$ | Set of 20' containers. |
| $\mathcal{F}$ | Set of 40' containers. |
| $\mathcal{T}_i$ | Set of 20' containers loaded at port i. |
| $\mathcal{F}_i$ | Set of 40' containers loaded at port i. |
| $T_{ij}$ | Number of 20' containers going from port $i$ to port $j$. |
| $F_{ij}$ | Number of 40' containers going from port $i$ to port $j$. |
| $T_{ij}^m$ | Number of 20' containers of weight class $m$ going from port $i$ to port $j$. |
| $F_{ij}^m$ | Number of 40' containers of weight class $m$ going from port $i$ to port $j$. |
| $E_i$ | Number of empty slots after loading operations at port $i$. |
| $W_i$ | Total weight on board after loading operations at port $i$. |
| $w_m^T$ | Weight of a 20' container of class $m$. |
| $w_m^F$ | Weight of a 40' container of class $m$. |
| $W^a$ | Total weight on board in area $a$. |
| $Q_1$ | Maximum percentage of weight deviation allowed between bow and stern ends. |
| $Q_2$ | Maximum percentage of weight deviation allowed between starboard and port sides. |
| $t_{ij}(p,x)$ | Binary decision variable, set to 1 if one 20' container with destination port $j$ is stowed in slot $(p,x)$ after the loading operations at port $i$. |
| $rt_i(p,x)$ | Binary decision variable, set to 1 if one 20' container stowed in slot $(p,x)$ at port $i-1$ is relocated at port $i$. |
| $e_i(p,x)$ | Binary decision variable, set to 1 if slot $(p,x)$ is empty after the loading operations at port $i$. |
| $t_{ijm}(p,x)$ | Binary decision variable, set to 1 if one 20' container of weight class $m$ with destination port $j$ is stowed in slot $(p,x)$ after the loading operations at port $i$. |
| $f_{ij}(p)$ | Binary decision variable, set to 1 if one 40' container with destination port $j$ is stowed in cell $p$ after the loading operations at port $i$. |
| $rf_i(p)$ | Binary decision variable, set to 1 if one 40' container stowed in cell $p$ at port $i-1$ is relocated at port $i$. |
| $f_{ijm}(p)$ | Binary decision variable, set to 1 if one 40' container of weight class $m$ with destination port $j$ is stowed in cell $p$ after the loading operations at port $i$. |

| | |
|---|---|
| $e_i(p)$ | Binary decision variable, set to 1 if cell $(p)$ is empty after the loading operations at port $i$. |

# Chapter 1

# Motivation and scope of research

This chapter introduces the reader to the framework of containerized maritime trade, contextualizing and motivating the development of the thesis. First, the history of maritime trade is briefly described and the container trade and the situation of the container industry are discussed. Then the main operational research problems that arise in container terminals are described, the problems that have been studied are listed, and finally an outline of the thesis is given.

## 1.1 Brief history of maritime trade and containerization

The use of water for transporting cargo dates back to the beginning of time, when logs were thrown downstream and a small load was tied to them for trade. The log raft is probably the first step in the evolution of wooden ships. This invention could have happened more than once, given the simplicity of the idea and the universal distribution of log rafts; however, it is a question that can never be solved (Hornell, 1946). The first transport routes were established by the 3rd century BCE[1] along the Arabian Sea and, simultaneously, along the Mediterranean, where Romans started to transport bulk commodities by building large fleets of commercial ships. Romans realized that it was cheaper to transport goods by sea than by road. In fact, a Roman ship could cross the Mediterranean in one month at a cost of one-sixtieth of the cost of land routes (Scarre, 1995). They therefore expanded their trade routes to the Indian Ocean. By the 7th - 13th centuries, the Arab Empire developed trade routes through Asia, Africa and Europe. Although in those years no single ship could go directly from Britain to China, goods such as silk, gold or spices moved regularly across the sea and between empires as far apart as Rome and China (Stein, 2017). In the Age of Discovery (15th - 19th centuries), major advances in navigation and shipbuilding enabled Europeans to cross the Atlantic and also to make numerous voyages across the Indian and Pacific Oceans.

Even though humanity has always been shipping goods across the seas from one land to another, prior to the 1970s loading and unloading ships was an expensive and labor-intensive task, making it unprofitable to transport many types of cargo overseas. The introduction

[1]BCE: Before the Common Era

and standardization of containers revolutionized seaborne trade, to the extent that by 1973 container ship operators in the United States, Europe and Asia were already carrying an average of 4 million twenty-foot equivalent units per year. Port efficiency skyrocketed, drastically reducing shipping costs, and resulted in a shift of production to countries with cheaper manufacturing costs. Due to the change it entailed in the way manufacturing and trade were conducted, containerization is seen as one of the main drivers of globalization. Containerized trade has lowered the import costs of many products to such an extent that it is sometimes cheaper to transport the goods to the other side of the world than to produce them locally.

Containerization is a system of standardized transport that uses intermodal containers with uniform dimensions. These containers can easily be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another without being opened. The idea for an intermodal container came to a young trucker named Malcolm McLean (1913-2001) in 1937 while he was waiting to deliver bales of cotton to the port of New Jersey.

> *"I had driven my trailer truck up from Fayetteville, North Caroline with a load of cotton bales that were to go on an American Export ship tied up at the dock. For some reason or another, I had to wait most of the day to deliver the bales, and as I sat there, I watched all those people muscling each crate and bundle off the trucks and into the slings that would lift them into the hold of the ship. On board the ship, every sling would have to be unloaded by the stevedores and its contents put in the proper place in the hold. What a waste in time and money! Suddenly, the thought occurred to me: Wouldn't it be great if my trailer could simply be lifted up and placed on the ship without its contents being touched?"*

Nearly two decades later, on April 26, 1956, he carried fifty-eight containers on the converted oil tanker *Ideal-X* which sailed from the port of Newark and successfully reached Houston within six days. This journey would mark the birth of containerized trade. During the early years, each shipping company built its own containers with various sizes and corner fittings, McLean's SeaLand Industries was still using 35-foot containers, while industry rival Matson was using 24-foot containers. Because of pressure from the United States government for a more efficient and standardized way of transporting goods in the Vietnam War, McLean finally published its patent on the revolutionary corner fittings for container shipping and a set of rules for the standardization of shipping containers was agreed. The ISO[2] standards were published between 1968 and 1970 by the International Maritime Organization (IMO). These standards promote the development of intermodal container shipping by making containers suitable for transporting cargo by different means, such as truck, rail or ship, and allow for more consistent loading, transport and unloading of goods in ports around the world, thus saving time and resources.

The basic unit of measurement for containers is currently the TEU, 20-foot equivalent unit, and refers to a standardized container 20' long, 8' wide, and 8'6" high. The unit of measurement for containers with the same width but twice the length, even more common today, is the FEU, 40-foot equivalent unit. One FEU equals two TEUs (see Figure 1.1).

[2]International Organization for Standardization

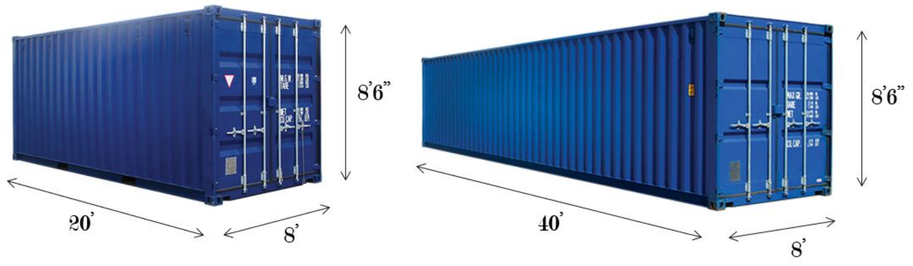


Figure 1.1: Dimensions of standard 20' (left) and 40' (right) containers.

Containers of these sizes, 20' and 40', account for about 90% of all container shipping. Shorter and longer containers (8', 10', 45', 48', 53') can also be found in the sector, though they cannot be shipped in the same way as 20' and 40' containers nor can they be loaded on container ships, except for 45' containers, which can be loaded, but only on deck. In contrast to standard containers, which have a maximum height of 8'6", there are containers of similar width and length but with a height of up to 9'6" (*high-cube*), which makes them ideal for light, bulky and large cargoes. Although it is uncommon to find 20' high-cube containers, 40' high-cube containers are often used.

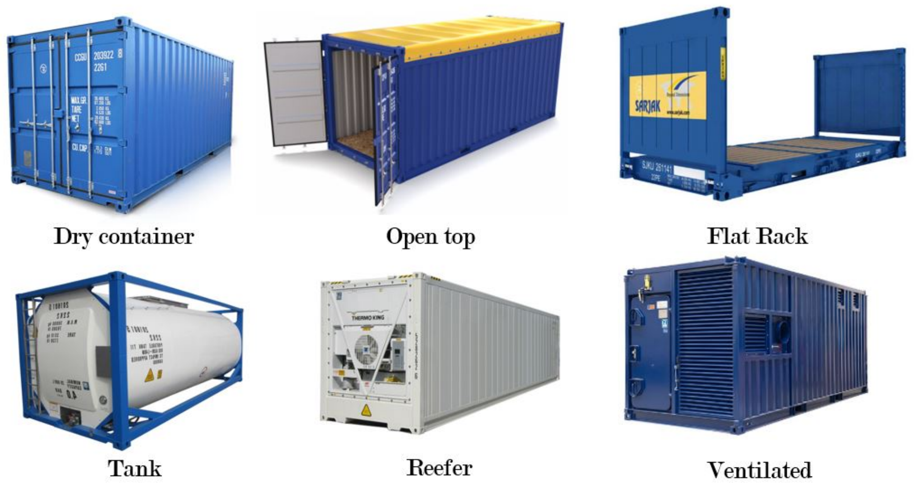


Figure 1.2: Some of the most common types of shipping containers.

The diversity of the cargo transported requires the use of different types of containers for special purposes or needs. Some of the most common types of containers are shown in Figure 1.2. The most widely used are *general purpose* or *dry van* containers for cargo without special requirements. For cargo that exceeds the dimensions of a shipping container in length, width, height, or all of these, *Out of Gauge* containers are used, such as *open top* containers, with the top open and covered with a strong rubber tarpaulin rather than a solid roof. *Flat rack* containers have end walls but no sides, so they are used for transporting goods with unusual shapes as these can stick out through the open sides. Liquid materials, both hazardous and non-hazardous, are mostly transported in *tank* containers. Perishable goods such as fruit and vegetables are transported in *reefer* containers. Reefer containers can be plugged into the power supply on ships or have clip-on generators attached. Cargo that needs ventilation in transit, such as coffee, is transported in *ventilated* containers, also known as *coffee* containers.

## 1.2 International maritime trade

Nowadays, more than 80% of world cargo in volume and 70% in value is transported by sea, making international maritime transport the lifeblood of world trade and global economy. Every year since 1968, the United Nations Conference on Trade and Development (UNCTAD) has published a Review of Maritime Transport, which provides an analysis of structural and cyclical changes affecting seaborne trade, ports and shipping, as well as an extensive collection of statistical information. According to the latest report, a historic high of 11 billion tonnes transported by sea was reached in 2018 and the volumes transported are forecast to grow by 2.6% in 2019. However, the world seaborne trade lost momentum compared to 2017, increasing by 2.7% instead of 4.1%, below the historical average of 3.0% from 1970 to 2017. The main factors in the slowdown were trade tensions and protectionism[3], the departure of the United Kingdom from the European Union, the economic transition in China, geopolitical turmoil, supply disruptions such as those in the oil sector, and other country-specific developments

Figure 1.3 shows the trends in international maritime trade between 1980 and 2018 by cargo type. Over that period, sea transport increased by 197.1%. With respect to the type of cargo transported, tanker trade (oil and gas) increased by 71%, main bulk commodities (iron ore, grain and coal) increased by 428%, other dry cargo (other than major bulks) increased by 139%, and finally containerized cargo increased the most, by 1782%. Considering a shorter and more recent, therefore more representative, period from 2010 to 2018, total sea transport grew by 31%, with containerized cargo again achieving the greatest increase, of around 49%. The favourable growth trend was expected to continue in the coming years with maritime trade increasing at an annual rate of 3.4% between 2019 and 2024, driven in particular by growth in container, dry bulk and tanker trade, with a compound annual growth rate expected to be 4.5%, 3.9%, and 2.2%, respectively. Nonetheless, the new strain of coronavirus, COVID-19, hitting the global economy during the first months of 2020, may substantially alter these predictions, and although the most recent statistics point to a significant downturn in global trade, the full consequences and impact will not be known for several months.

The answer to the question of who generates trade, which means where the cargo is loaded and unloaded, has historically been the developing countries. These countries have been the main exporters, with about two-thirds of maritime trade originating in their territories. In 2018, this trend continued and developing countries accounted for most global maritime trade flows, both in terms of exports (goods loaded), with 58.8% of the total, and in terms of imports (goods unloaded), with 64.5% of the total. In contrast, only 34.7% of exports and

[3]Protectionism is the economic policy of restricting imports from other countries through methods such as tariffs on imported goods, import quotas, and a variety of other government regulations.

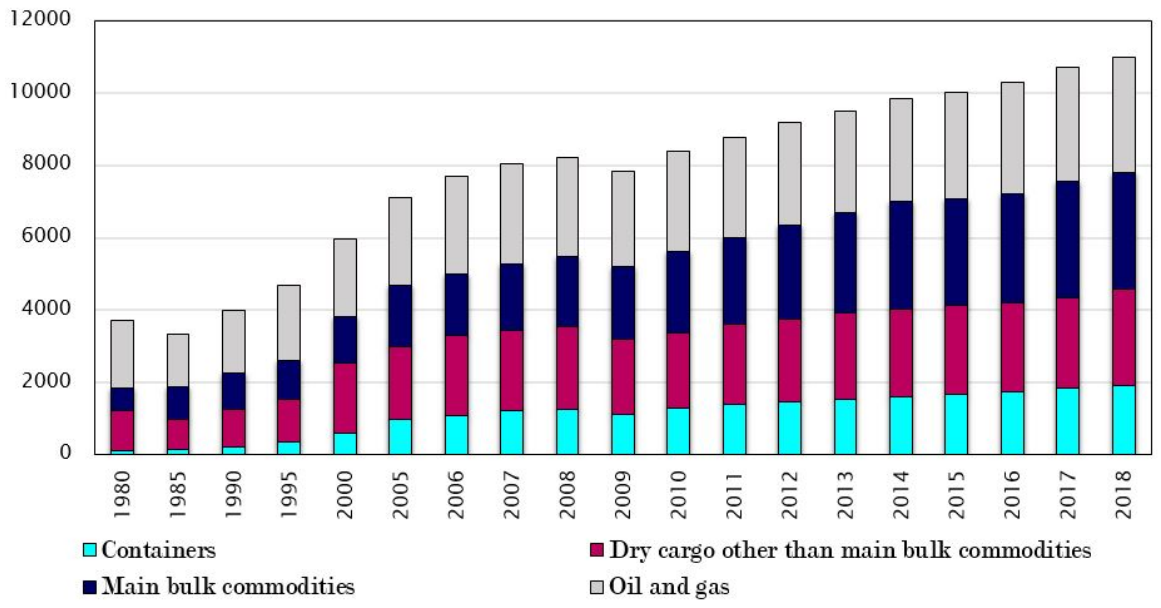


Figure 1.3: International maritime trade by cargo type in million of tons.
Source: Data taken from the Review of Maritime Transport, 2018 and 2019 issues (UNCTAD, 2018, 2019).

34.7% of imports were produced in developed countries. The remaining exports and imports, distributed in significantly smaller percentages, were produced in transition countries.

## 1.3 Containerized trade

According to UNCTAD (2019), containerized trade accounted for 24% of total dry cargo shipments and reached volumes of around 152 million TEUs in 2018. Container ports handled 793.3 million TEUs in 2018 and 752.2 million in 2017 (UNCTAD, 2018). The difference is equivalent to the container cargo throughput of the port of Shangai, which is the largest container centre in the world in 2018 based on the annual ranking by the Journal Of Commerce (JOC), shown in Table 1.1.

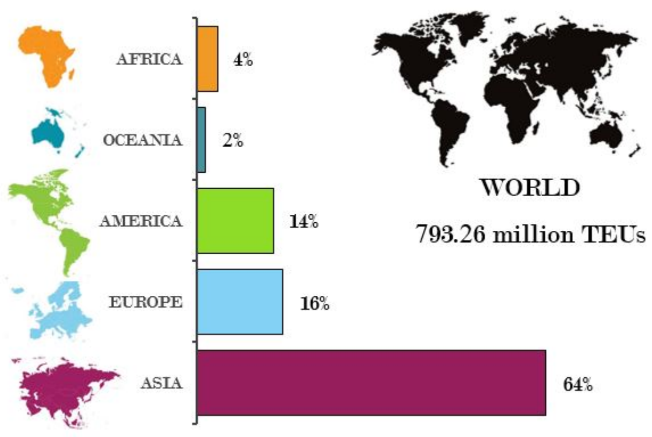


Figure 1.4: Percentage share of world container port throughput per region.
Source: Data obtained from UNCTAD secretariat calculations UNCTAD (2019).

Figure 1.4 represents the percentage share per continent in world container port throughput. Asia is the continent with the largest presence in container handling, with 64%. Far behind is Europe with 16%, then America with 14%, and finally Africa and Oceania with 4% and 2%. The importance of Asia is mainly due to China, which, including Hong Kong and Taiwan Province of China, handled 260.7 million TEUs, one-third of total container port-handling activity. China has been a key player in the container trade over the last decade and remains so, with 7 of the 10 busiest ports in the world located in its territory. Container port traffic in Europe, supported by trade between China and the European Union, grew steadily during 2018, at a rate of 5.5%, above the average global growth of 4.7%. With regard to the Spanish role in the sector, Spain is currently one of the top ten most important countries worldwide in container port throughput, occupying tenth position with a total of 17.2 million TEUs recorded in 2018, 36.5% more than 2010. It is the European country with the second greatest container port-handling activity, surpassed only Germany, with 19.6 million TEUs.

Table 1.1 shows how containerized cargo handling remains concentrated in a few major ports. The top 10 ports handle 244.13 million TEUs, 31% of the total, and all of them are in Asia. Shanghai remained the busiest container port worldwide in 2018, with a total of 42 million TEUs handled, 2 more million (4.4%) than in 2018. This port alone handled 7.6 million TEUs more than the three main European ports, Rotterdam, Antwerp and Hamburg, combined. If we focus on the top 20 ports, only five non-Asian ports appear, the three main European ports already mentioned together with the ports of Los Angeles and Long Beach in the United States. The port of Valencia is the fifth European port, behind Rotterdam, Antwerp, Hamburg, and Bremen, and is the leading Spanish port. It occupies position 30 in the JOC ranking of the 50 busiest container ports. The other Spanish ports that appear in this ranking are Algeciras, in position 34, and Barcelona, in position 46.

## 1.4 Container shipping industry

The first 50 years of containerization have seen a remarkable evolution of container ships, which have advanced from the *Ideal X* to the current vessels with capacities exceeding 20,000 TEU (e.g., the MSC Mina and MSC Gülsün) and an average tonnage of 83,122 dwt[4]. Container ships nowadays represent the third most important fleet in terms of total cargo-carrying capacity, behind dry bulk carriers and oil tankers, with 13.1% of the total. However, since container ships transport goods of higher unit value and travel at higher speeds, they account for more than half of total maritime trade by monetary value (UNCTAD, 2018).

Out of all the ships delivered for trade in 2018, 23.5% were container ships, with demand for container ships larger than 15,000 TEUs growing by 33% in 2018 (Clarkson, 2019). The increasing size of ships, with the entry of mega-container ships into an oversupplied market, together with trade tensions, an increase in protectionism and changes in environmental regulations, have increased the imbalance between trade and fleet supply capacity, causing overall freight rates to fall in 2018-2019. While the demand for cargo may be affected by the new challenges and additional costs of complying with the new IMO 2020 regulations,

[4]DWT: Deadweight tonnage

Table 1.1: Leading ports in container cargo throughput 2018, in millions of TEUs.

| Rank | Port | Country | 2018 |
|---|---|---|---|
| 1 | Shanghai | China | 42.01 |
| 2 | Singapore | Singapore | 36.66 |
| 3 | Ningbo-Zhoushan | China | 26.35 |
| 4 | Shenzhen | China | 25.73 |
| 5 | Guangzhou | China | 21.92 |
| 6 | Busan | South Korea | 21.66 |
| 7 | Hong Kong | China | 19.60 |
| 8 | Qingdao | China | 19.31 |
| 9 | Tianjin | China | 16.00 |
| 10 | Jebel Ali | United Arab Emirates | 14.95 |
| 11 | Rotterdam | Netherlands | 14.51 |
| 13 | Antwerp | Belgium | 11.10 |
| 17 | Los Angeles | United States | 9.46 |
| 19 | Hamburg | Germany | 8.77 |
| 20 | Long Beach | United States | 8.01 |
| 30 | Valencia | Spain | 5.18 |
| 34 | Algeciras | Spain | 4.77 |
| 46 | Barcelona | Spain | 3.42 |

Source: The JOC Top 50 Global Container Ports 2018 (www.joc.com).

reducing sulphur content in fuel oil from 3.5% to 0.5%, capacity management will be critical in reconciling slow demand growth, high supply capacity, and high operating costs.

To cope with the difficult market conditions, the global container shipping industry has tended towards consolidation since the bankruptcy of the Hanjin container line in the Republic of Korea in 2016. According to UNCTAD (2019), the top 10 container shipping lines have achieved a combined market share of 90%, with a deployed capacity of about 96.4 million TEUs. These main lines belong to three major alliances: 2M Alliance, which includes MSC, Maersk and HMM, with a total of 223 ships and a total capacity of around 2.4 million TEUs, Ocean Alliance, which includes CMA-CGM, Cosco Group, OOCL and Evergreen, and has 323 ships with a total capacity of around 3.5 million TEUs, and The Alliance, including Hapag Lloyd, NYK, Yang Ming, MOL and K-Line, with a fleet of 241 ships and a capacity of around 3.3 million TEUs. UNCTAD highlights two opposing sides to container market consolidation. On the one hand, by consolidating and joining alliances, container lines can reduce costs, better manage vessel capacity and increase efficiency. Shippers, the clients of an alliance, can benefit if lower fares and better service provision are achieved. On the other hand, shippers, trade and ports can also be negatively affected because of the increase in competition, limited supply, abusive market power, and higher rates and prices.

The increasing size of ships and the creation of strategic liner shipping alliances have triggered a new dynamic in which shipping lines have greater bargaining power and influence over the container terminals, which must compete for fewer services from larger ships. In 2018, the median time in port of container ships was 16.8 hours, about seven hours less than the global median of 23.5 hours for all ships. Figure 1.5 shows the global distribution of port calls by container ships (represented by the size of the circles) and the average time spent in port (the color of the circles). The dominance of the Asian continent and, more precisely, of China can be clearly seen. It can also be seen that the average time in port of container ships is longer than the median time in practically all countries. This is due to the existence of statistical outliers corresponding to ships that remain in port for weeks or even months, for example for repairs.

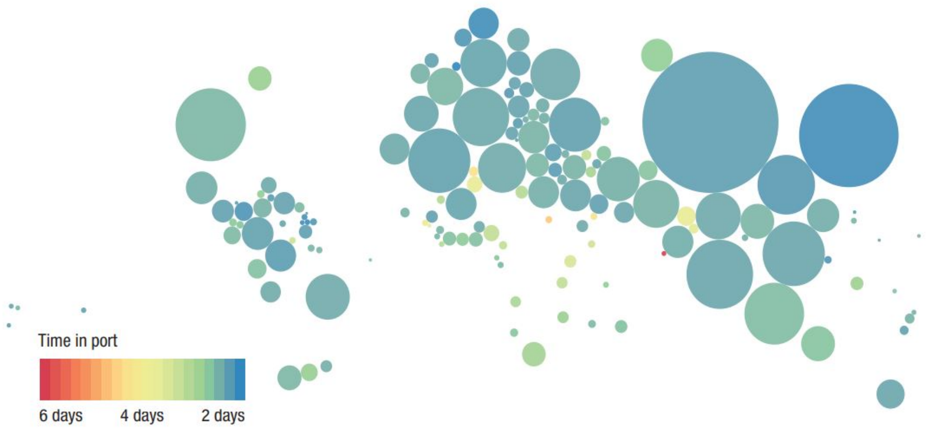


Figure 1.5: Container ship port calls and time in port (days), 2019.
Source: UNCTAD secretariat calculations (http://stats.unctad.org/teu) UNCTAD (2019).

Countries with shorter delivery times are more attractive to shippers and carriers. Therefore, the number of port calls will tend to be higher. However, the relationship also works in the opposite direction: countries with more trade and more port calls will also produce more revenue to invest in efficient port operations and will tend to reduce their delivery times.

## 1.5 Operational Research and container terminals

Container terminals mainly serve as an interface between different modes of transport, such as domestic rail or truck transport and offshore shipping (Günther and Kim, 2005). Zhao et al. (2020) divide the services offered by container terminals into three stages: the arrival of containers, the storage of containers, and the departure of containers. Figure 1.6 illustrates the classic layout of a container terminal, consisting of three different areas: *Seaside, Yard* and *Landside Area* (Böse, 2011).

- The *Seaside Area* is the sea-land connection area within a maritime terminal, where vessels are moored to unload containers arriving at the terminal as their final destination or in transit and to load containers leaving the terminal. Loading/unloading operations are performed by quay cranes whose productivity is mainly measured by the number of movements per hour.

- The *Yard Area* is a large space where inbound containers are stored before being transferred to the hinterland (import containers), and outbound containers are also stored before being loaded on the ships (export containers). This area has to ensure rapid loading and unloading of ships and has become the bottleneck of terminals.

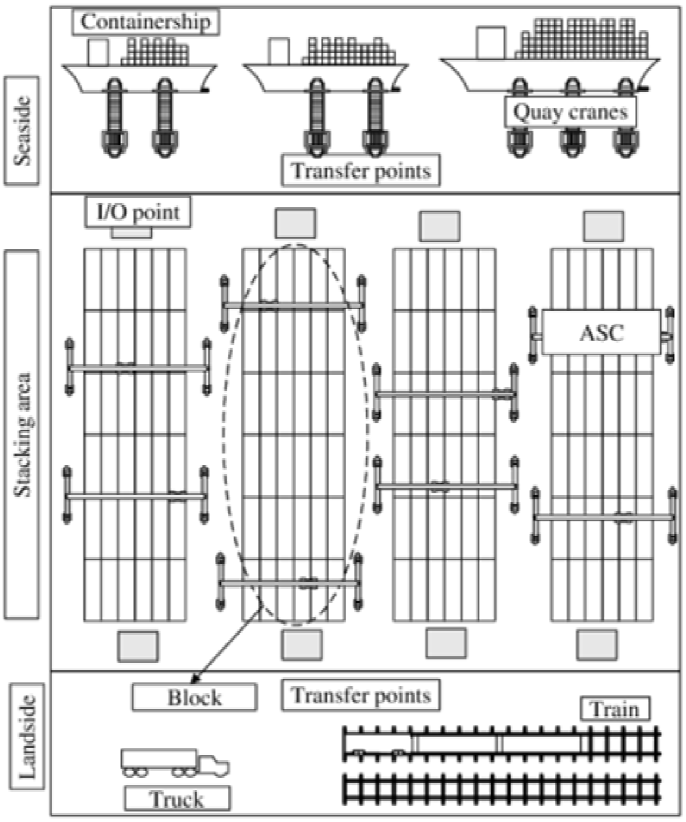


Figure 1.6: Schematic top view layout of a container terminal (Gharehgozli et al., 2014).

- The *Landside Area* is the area where containers are loaded on or unloaded from external trucks and trains. Trucks account for most of the movement and their loading and unloading involve peak hours and highly variable requirements. In contrast, trains are more flexible and allow high levels of performance.

Figure 1.7 shows an outline of the different planning problems that arise in a container terminal according to the area in which these problems occur. It follows the classification of problems proposed by Meisel (2009) who differentiates between *strategic*, *tactical*, and *operational planning problems*, depending on the planning horizon. Strategic decisions often last for years and are very time-consuming and expensive. This level involves decisions such as the location and configuration of a new terminal layout, but also decisions regarding the types and quantity of equipment to be purchased. Tactical decisions often last for months, or at least weeks, and are the general criteria for operation. This level involves decisions such as the use of the terminal space, and general rules are defined for the allocation of resources to tasks, as well as the route of the horizontal transport equipment. Finally, operational decisions last for days, hours, or even seconds. This level involves decisions on matters such as operational management, i.e., the assignment of specific equipment to carry out each activity and the individual work plan for each piece of equipment. Some decisions need to be made at the start of operations and others while the processes are running,

Within container terminals there is a wide variety of problems whose importance in a sector as demanding as shipping has attracted the attention of the scientific community

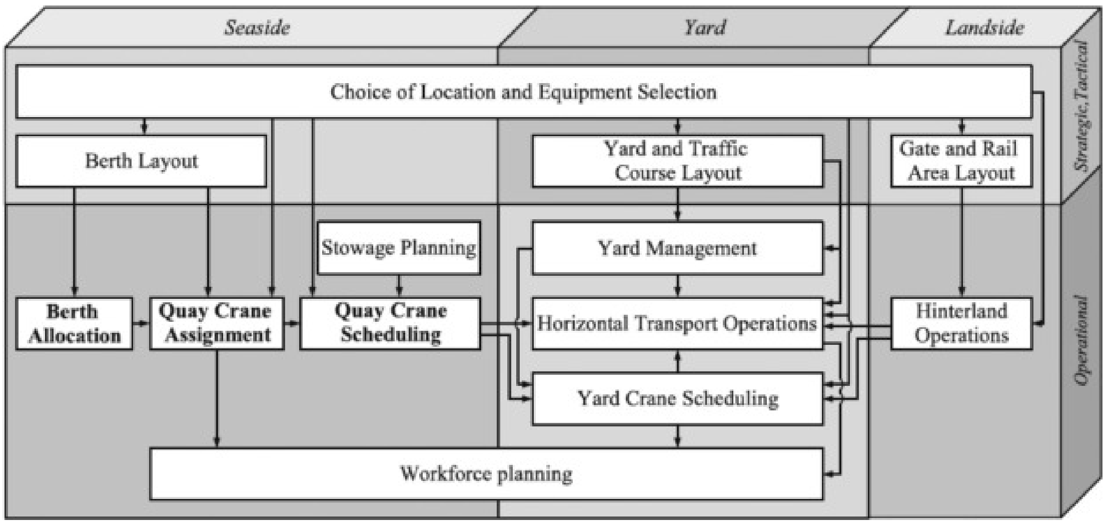


Figure 1.7: Planning problems in container terminals (Meisel, 2009).

in recent decades. Several authors address strategic and tactical decisions (Petering, 2011; Cartenì and De Luca, 2012; Roy and de Koster, 2014; Prayogo et al., 2017; Huang et al., 2019; Gharehgozli et al., 2019). Most of these authors focus on simulation models. A detailed review of the research literature available on the application of simulation models in ports and container terminals is provided by Dragović et al. (2017). We focus here on operational planning problems, requiring decision-making whose effects last from seconds up to days, briefly describing each problem and classifying them according to the area in which they arise.

## 1.5.1 Seaside area operational planning problems

Operational optimization problems in the seaside area have been carefully reviewed in Meisel (2009); Bierwirth and Meisel (2010); Carlo et al. (2015). There are four main planning problems in this area:

- The *Berth Allocation Problem* seeks to assign a berthing position and a berthing time to each vessel, taking into account the basic characteristics of the berths and the vessels. Since the goal is to provide fast and achievable services, the objective function most used in the literature consists of minimizing the sum of waiting and handling times of vessels.

- The *Quay Crane Assignment Problem* seeks to assign a set of quay cranes to each vessel. Minimizing delays or maximizing early departures are objective functions commonly used for this problem.

- The *Quay Crane Scheduling Problem* seeks to assign and sequence the loading and unloading operations of the vessels to a predetermined number of quay cranes. The problem is typically addressed as a scheduling problem, where the quay cranes are the machines and the jobs correspond to individual containers, groups of containers, or sections of the vessel. Minimizing makespan of the quay crane schedule is often the objective of this problem because it corresponds to the handling time of the vessel.

- The *Stowage Problem* seeks to determine the exact position of the containers aboard the ship. Since this problem is extensively studied in this thesis, it will be further described in Chapters 5 and 6.

### 1.5.2 Yard area operational planning problems

Detailed reviews of operational planning problems in the storage yards of container terminals are provided by Zhen et al. (2013); Carlo et al. (2014). We divide them into three groups.

- *Yard Space Management* considers the problems related to the allocation, marshalling and retrieval of containers within the yard. There are three stages in a container's life cycle within the yard. First, it is placed on a stack, using assignment strategies, such as stacking containers destined for the same section of a ship together. The goal in this stage is to avoid containers blocking the retrieval of other containers. However, as container yards are dynamic environments with many containers entering and leaving, it is difficult to sort containers perfectly in advance. In the second stage, containers wait to be extracted. In this period, the container premarshalling problem, thoroughly studied in this thesis, provides decision support on how to re-order containers whose retrieval was or has become blocked by others. In the third stage, containers are extracted.

- *Horizontal Transport Operations* considers three main decisions: assignment of vehicles to quay cranes, assignment of transport orders to vehicles (the vehicle dispatching problem), and the routing of vehicles and control of traffic.

- *Yard Crane Scheduling* considers yard crane dispatch (deciding which route the yard cranes will take within a block) and yard crane deployment (deciding how many cranes will be deployed in each block and how the yard cranes will be moved between blocks).

### 1.5.3 Landside area operational planning problems

Operational problems arising in the landside area are detailed in Steenken et al. (2004). They include the scheduling of appointment times of trucks and trains, the assignment of vehicles to loading/unloading operations, the scheduling of gantry cranes that serve trains and trucks, and the routing of vehicles operating in that area.

## 1.6 Scope of this thesis

The need to handle more containers in the same timeframe puts pressure on both berthing and yard operations. Container terminals cannot simply increase the number of available cranes indefinitely, so the focus is on optimizing existing resources to reduce berthing times. An effective way of speeding up the loading/unloading operations of ships at the container terminal is to use the idle time before the arrival of a ship for sorting the stored containers in advance. In this context, the premarshalling problem consists in rearranging the containers in each bay of a container yard in the order in which they will be required later. With sorted bays, loading/unloading operations are significantly faster, as there is no longer a need to make unproductive moves in the bays once ships are berthed. Another effective way of speeding up the loading/unloading operations of ships is to provide efficient stowage plans describing in which position each container should be loaded on the ship. These plans have to satisfy high-level constraints, related to the stability of the ship, as well as low-level constraints related to the way in which containers have to be stacked, the weight distribution, and other specific conditions such as those regulating containers with dangerous products. Container ships, whose capacity can exceed 20,000 TEUs, sail from port to port loading/unloading thousands of containers. Considering such large volumes, maximizing efficiency in container handling is crucial to the global economy, as well as to the environment, since reducing unnecessary port operations results in a greatly reduced carbon footprint. In this context, the multi-port container ship stowage problem consists in determining the position of the containers loaded at each port of the ship's route so as to minimize the number of unproductive moves necessary to unload the containers at their port of discharge.

This thesis mainly focuses on these two problems: the *Container Premarshalling Problem* (CPMP) and the *Multi-port Container Stowage Planning Problem* (Multi-port CSPP), considering new generalizations, objective functions and constraints that bring the problems studied in the scientific literature closer to the reality of the case studies in practice. Exact techniques, such as integer programming models and other more sophisticated exact methods such as branch and bound algorithms, are used throughout this thesis, but also heuristic techniques, such as constructive, matheuristic, and metaheuristic algorithms. The aim of the thesis is twofold: to contribute to improving the efficiency of maritime container terminals, and to bring new ideas and knowledge to the field of Operational Research, which could potentially be useful in addressing other combinatorial problems of a similar nature.

### 1.6.1 Outline

This thesis is divided into seven chapters as follows:

**Chapter 1. Motivation and scope of research.** This chapter introduces the reader to the framework of containerized maritime trade, contextualizing and motivating the development of the thesis. First, the history of maritime trade is briefly described and the container trade and the situation of the container industry are discussed. Then the main operational research problems that arise in container terminals are described, the problems that have been studied are listed, and finally an outline of the thesis is given.

**Chapter 2. Integer programming models for the container premarshalling problem.** This chapter addresses the premarshalling problem by developing and testing integer linear programming models. Two alternative families of models are presented, as well as an iterative solution procedure that does not depend on a difficult-to-obtain upper bound. An extensive computational analysis is carried out over several well-known datasets from the literature. The chapter is based on the following published paper:

> Parreño-Torres, C., Alvarez-Valdes, R., Ruiz, R., 2019. Integer programming models for the premarshalling problem. European Journal of Operational Research. 274, 142 - 154. doi: 10.1016/j.ejor.2018.09.048.

**Chapter 3. A branch-and-bound approach for large premarshalling problems.** This chapter proposes a branch-and-bound algorithm with new components for solving large and difficult premarshalling instances. The existing lower bounds are strengthened and two new lower bounds that use a relaxation of the premarshalling problem to provide tight bounds in specific situations are introduced. The chapter also introduces generalized dominance rules to reduce the search space, and a memoization heuristic that finds feasible solutions quickly. The algorithm is evaluated to standard benchmarks of premarshalling instances, as well as to a new data set to avoid over-adaptation to available data. It is based on the following published paper:

> Tanaka, S., Tierney, K., Parreño-Torres, C., Alvarez-Valdes, R., Ruiz, R., 2019. A branch and bound approach for large premarshalling problems. European Journal of Operational Research 278, 211 - 225. doi: 10.1016/j.ejor.2019.04.005

**Chapter 4. Minimizing the crane time in premarshalling problems.** This chapter shows that the number of movements needed to rearrange the bay in the premarshalling problem does not correlate with the time the crane takes to perform the movements. Since time is shown to be a more realistic objective for measuring process efficiency, the problem of minimizing crane times is addressed by developing two exact approaches to solve it: an integer linear mathematical model, and a branch-and-bound algorithm, with new upper and lower bounds, dominance criteria, and a heuristic procedure to repair the feasibility of the solutions. An extensive computational study of standard benchmarks shows the performance of both approaches. This chapter is based on the following forthcoming paper, already accepted for publication:

> Parreño-Torres, C., Alvarez-Valdes, R., Ruiz, R., Tierney, K., 2020. Minimizing the crane time in premarshalling problems. Transportation Research Part E 137, Art. ID 101917. doi: 10.1016/j.tre.2020.101917.

**Chapter 5. Solution strategies for a multi-port container stowage planning problem.** This chapter presents an integer programming model for the multi-port container ship stowage problem and proposes several sets of valid constraints that bring its LP-relaxation closer to an integer solution. Moreover, it presents a GRASP algorithm that generates stowage plans with a minimal number of unproductive moves in a high percentage of medium and large size instances. An extended computational analysis has been performed in which the efficiency of the integer programming models and the GRASP algorithm are tested. The contributions of this chapter have been published in the following paper:

Parreño-Torres, C., Alvarez-Valdes, R., Parreño, F, 2019. Solution strategies for a multi-port container ship stowage problem. Mathematical Problems in Engineering. Art. ID 9029267. doi: 10.1155/2019/9029267.

**Chapter 6. Mathematical models and matheuristics for a generalized multi-port container stowage planning problem.** This chapter investigates how the complexity of the multi-port container ship stowage problem changes with and without the consideration of container size and weight constraints. For this purpose, integer programming formulations are proposed for the general problem, as well as some special cases with identical container size and/or identical weights, and their performance is evaluated on randomly generated small and medium-scale instances. To solve large-scale problems, the chapter presents three matheuristics, namely a Relax-and-Fix, an Insert-and-Fix, and a Fractional Relax-and-Fix heuristic, algorithms that use the special structure of the formulations. It also presents a constructive procedure that gives the solver an initial solution. Several experiments are conducted to solve the approaches. This chapter is based on the following paper already submitted to a recognized journal in the area:

Parreño-Torres, C., Çalik, H., Alvarez-Valdes, R., Ruiz, R., 2020. Mathematical models and matheuristics for a generalized multi-port container stowage planning problem. Submitted to Computers & Operations Research.

**Chapter 7. Conclusions and future work.** This chapter presents the contributions of the thesis and outlines the lines of research and future work that could be derived from it.

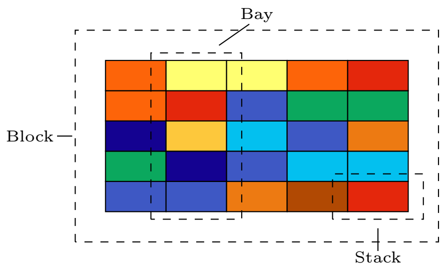
# Chapter 2

# Integer programming models for the container premarshalling problem

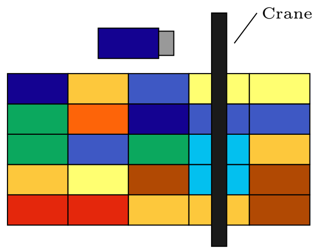
## 2.1 Introduction

To service larger-sized ships quickly, terminal operators use cranes over long working hours and shifts, which leads to overutilization of port capacity on some days and underutilization on others (Drewry Maritime Research, 2016). A good yard arrangement before the arrival of ships contributes significantly to softening workload peaks and increasing terminal productivity.

The container yard is a large space in which inbound containers are stored before being transferred to the hinterland and outbound containers are also stored before being loaded onto ships. The yard is divided into several blocks composed of parallel bays. Each bay has the same number of stacks of containers and each stack has a maximum number of tiers up to a height fixed by the height of the crane. As a consequence of stacking, to reach a specific container it may be necessary to rehandle containers on top of it. Figure 2.1 shows the top view of a container yard scheme with 2 blocks, 5 bays per block, and 5 stacks per bay. Subfigure 2.1a shows the basic elements of the problem: the block, the bay, and the stack. Subfigure 2.1b shows the interaction of block, crane, and truck. Yard cranes move along the block on top of containers. Trucks move along the bays to the bay in which a container has to be loaded or unloaded.



(a) Basic elements.



(b) Interactions.

Figure 2.1: Top view of a container yard scheme.

Premarshalling is an important mechanism to prepare containers for their onward journey, since delays to trucks, trains or ships could result in misoverlays even in a previously well-planned layout. When the workload in a terminal is low, bays are rearranged so that containers leaving the stacks early do not block containers leaving later. This process speeds up service times once the ship arrives. The classic objective of the container premarshalling problem (CPMP), is to minimize the number of container moves required to transform the initial layout of a bay into a final layout without any containers blocking the removal of other containers. We refer to containers that block the removal of others as *blocking* or *misoverlaying*[1] containers.

This chapter addresses the CPMP by developing and testing integer linear programming models. Besides the conceptual advantage of providing a deeper understanding of the problem, integer models are useful for three main reasons: (*i*) they are easy to implement for practitioners, (*ii*) they can be solved by commercial solvers, which have shown a dramatic increase in performance in recent years, often by five or even more orders of magnitude (Bixby, 2002), and are constantly improving, and (*iii*) they are very flexible, allowing constraints to be added or removed to meet the requirements of the specific problem being solved.

Our study of integer models for the premarshalling problem has two phases. In a first phase, we develop a series of models in which, besides the variables describing the configuration of the bay, the moves required to rearrange the bay are described by just one type of variable. The models are increasingly complex, improving the relationship of the variables and the linear relaxation at the expense of enlarging the set of variables. In a second phase, variables describing the moves are split into two types, one indicating the origin of the move and another indicating the destination of the move. This alternative strategy is shown to be much more efficient, in terms of the number of optimal solutions obtained and running times, in an extensive computational study on several datasets from the literature.

The remainder of the chapter is arranged as follows. Section 2.2 reviews the relevant literature for the premarshalling problem. A formal description of the problem is presented in Section 2.3. Sections 2.4 and 2.5 describe the mathematical models developed and Section 2.6 provides an iterative solution procedure to solve the models, avoiding the need for an upper bound to build them. The models are thoroughly tested computationally and compared with existing models in Section 2.7. Finally, Section 2.8 considers possible extensions of the models to cope with more realistic conditions and Section 2.9 highlights the conclusions.

## 2.2 Literature review

Achieving a good yard plan has become a primary objective of many container port terminals. This is the main reason why academic work in this field has been intense in recent years. Lehnfeld and Knust (2014) identified various problems in storage areas arranged in stacks, including premarshalling, re-marshalling, and relocation problems. All of them are post-stacking problems that deal with minimizing the future loading time of the ship. While in

[1] *Misoverlaid* containers are those that are blocked by others; *Non-misoverlaying* containers are those that are not blocking any other container; and *Non-misoverlaid* containers are those that are not blocked by any container.

the premarshalling and re-marshalling problems the number of containers remains constant during the reshuffling, in the relocation problem the number of containers decreases, i.e., containers are retrieved at the same time. In other words, the relocation problem deals with the management of the bay once the ship has berthed and while it is being loaded. The three problems are surveyed by Caserta et al. (2011). The re-marshalling problem is also studied by Kang et al. (2005); Kang et al. (2006); Choe et al. (2011). Recent papers in block relocation include exact approaches (Tanaka and Mizuno, 2018; Zhu et al., 2019; Quispe et al., 2018; Tanaka and Voß, 2019) and different heuristic approaches, such as tree search procedures (Zhang et al., 2016), Beam Search (Bacci et al., 2019), GRASP (Jovanovic et al., 2019a), and Ant Colony Optimization (Jovanovic et al., 2019b).

Several papers related to premarshalling propose heuristic methods for solving the problem. Lee and Chao (2009) present a neighborhood search process based on a combination of improvement moves. An extension of this research is proposed by Huang and Lin (2012), who also include another version of the premarshalling problem in which at the end of the reshuffling each type of container should be located in pre-defined cells. Hence, the objective is to find the minimum sequence of moves for sorting the containers in a specific way. Caserta and Voß (2009) propose a metaheuristic approach using the paradigm of the corridor method developed by Sniedovich and Voß (2006). Bortfeldt and Forster (2012) develop a tree search procedure and an algorithm to obtain a lower bound for the problem. In addition to a multi-start technique for the problem using the heuristic called Lowest Priority First (LPFH), Expósito-Izquierdo et al. (2012) propose an instance generator able to construct instances with different difficulty levels. More recently, Jovanovic et al. (2017) have presented an extension of the LPFH whose main idea is to provide different heuristics for each stage of the initial algorithm and mix them in a multi-heuristic approach. Target-guided algorithms are proposed by Wang et al. (2015) and by Wang et al. (2017). Wang et al. (2015) also present an extension of the problem that considers a dummy stack next to the bay where containers can be temporarily stored during the reshuffling of the bay. A genetic approach is proposed by Gheith et al. (2016) and a biased random-key genetic method by Hottung and Tierney (2016). The most recent heuristic method for solving the premarshalling problem is a deep learning tree search (Hottung et al., 2020), which uses a tree search guided with branching and bounding decisions made with deep neural networks.

A significantly smaller group of papers address the problem using ad-hoc exact methods. An A* algorithm is presented by Expósito-Izquierdo et al. (2012) in which a simple lower bound, counting the number of blocking containers, is used. Tierney et al. (2017) present an A* technique that includes branching rules, symmetry breaking, and the tighter lower bound proposed by Bortfeldt and Forster (2012). A branch and price approach based on column generation is introduced by van Brink and van der Zwaan (2014) and a branch and bound whose branching procedure is guided by a heuristic algorithm is proposed by Zhang et al. (2015). Tanaka and Tierney (2018) propose an improvement branch and bound algorithm guided by a heuristic algorithm and a tighter lower bound together with dominance rules that allow a greater number of branches to be cut. Tanaka et al. (2019), fully described in Chapter 3, also propose a branch and bound guided by a heuristic algorithm with new components for solving difficult instances. This approach optimally solves many more instances than previous exact works and finds optimal solutions to nearly all problems of an industrially relevant size (up to seven tiers), making heuristic methods for premarshalling only necessary for solving very large instances faster.

Concerning mathematical models, Lee and Hsu (2007) developed an integer programming model based on a multi-commodity flow formulation. In this model, time is discretized into time segments separated by time points. The slots in the bay are represented by nodes in the network and the possible moves for each container by arcs, with the moves of containers in each time segment being represented by flows in the network. They illustrate the behaviour of their model on one layout, including several extensions of the basic formulation. More recently, de Melo da Silva et al. (2018) propose a unified model for the premarshalling and the container relocation problem. They also discretize time into time segments separated by time points and consider two types of variables: variables describing the state of the bay and variables defining the movements. An upper bound is required and it is obtained using a simple heuristic. In this chapter, published in Parreño-Torres et al. (2019b), we study the performance and limits of the mathematical formulations for the premarshalling problem through an exhaustive computational analysis of the models proposed by Lee and Hsu (2007), by de Melo da Silva et al. (2018), and of the several formulations we developed.

We finally review other related problems such as the loading of vehicles on a "roll-on roll-off" ship (Hansen et al., 2017). In this problem, vehicles are stored in columns where they can potentially block each other, however the two dimensional structure of the problem and heterogeneous shapes of the vehicles lead to vastly different solution procedures than for premarshalling. Other problem is the stacking of steel coils (Tang et al., 2012, 2015), however coils can be stacked on multiple coils beneath it, leading to a different structure for symmetry breaking and dominance rules. Stack loading problems are considered for train cars and assembly lines, among other applications, in Boysen and Emde (2016), and the work of Galle et al. (2018) that considers the container relocation problem when deciding where to store and relocate containers in the yard. The blocks world problem from the Artificial Intelligence community, AI community, is also similar to the CPMP in that a set of blocks (containers) ought to be manipulated from above without height or stacks number constraints (Gupta and Nau, 1992; Slaney and Thiébaux, 2001).

## 2.3 Problem description and notation

The premarshalling problem considered here consists in determining a minimal sequence of moves to transform the initial layout of a bay into a final layout without containers that block the removal of others. Although the CPMP is now common in the literature, we discuss the assumptions of the problem:

- *Single bay assumption.* The CPMP is solved for a single bay, as it is assumed that moving the crane between bays not only consumes time, but also potentially violates safety constraints due to crane-crane or human-crane interactions.

- *Uniform move cost assumption.* As only the number of moves is considered, all crane moves have an equivalent cost or duration that is not entirely true in practice, as some moves may be between stacks that are far apart.

- *Single crane assumption.* There is just one crane that can only move one container at one time and all containers are of the same type.

- *Full information assumption.* We assume that all the groups of the containers are known in advance. Since delays are common in port operations, in reality the groups of some containers might change, potentially causing new misoverlays even after premarshalling.

We formally define $\mathcal{C}$ as the set of containers, $\mathcal{S}$ as the set of stacks of the bay, where $S = |\mathcal{S}|$ is the total number of stacks, and $\mathcal{H}$ as the set of tiers, where $H = |\mathcal{H}|$ represents the highest tier of the stacks. The bay can be seen as an $H \times S$ matrix, where $H$ represents the highest tier of stacks and $S$ the number of stacks. Obviously, the bay can accommodate at most $H \times S$ containers. However, if the bay is at 100% occupancy, intra-bay moves of containers are not possible, so the occupancy level will always be lower than the total capacity. Each of the positions in the bay can be described by a tuple $(s, h)$ where $s \in \mathcal{S}$ and $h \in \mathcal{H}$. The order in which containers will be moved out of the bay is known in advance. We therefore assign a priority to each container $p \in \{1, ..., P\} = \mathcal{P}$, where 1 is the highest priority and $P$ the lowest. It is possible that more than one container may share the same priority, that is, within a set of containers with the same priority the loading sequence is irrelevant. We denote $n_p$ as the total number of containers of priority $p$ in the bay.

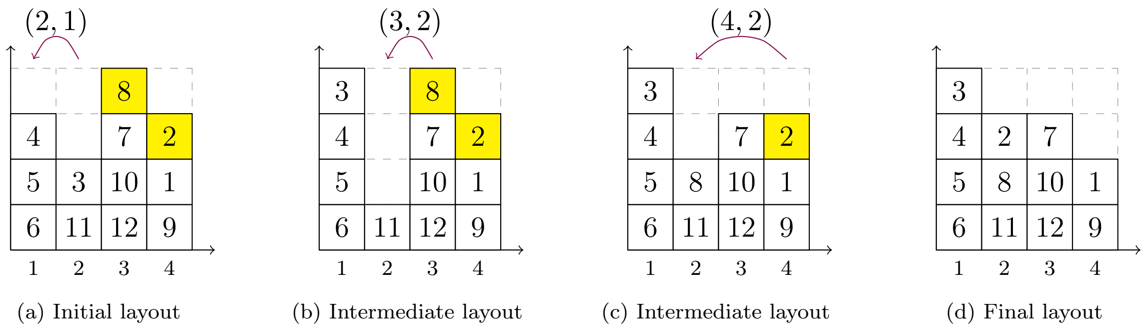


Figure 2.2: Example solution to the premarshalling problem. Blocking containers are shown in yellow.

A solution is a sequence of moves of the form $(s, k) : s \neq k \in \{1, \ldots, S\}$ where $s$ is the origin stack to which the topmost container to be moved belongs and $k$ is the destination stack where the container will also be placed at the top. Given a solution $sol = (s_1, k_1) \ldots (s_i, k_i) \ldots (s_n, k_n)$, the function $p_i(s, h)$ gives the priority of the container stored at position $(s, h)$ after the $i$-th move. The bay is considered ordered after move $n$ if $p_n(s, h) \geq p_n(s, h + 1)$ for all $s \in \mathcal{S}$ and $h \in \mathcal{H} \setminus \{H\}$. Figure 2.2 gives an example solution to the premarshalling problem. Subfigure 2.2a shows a bay with 4 tiers, 4 stacks, and 12 containers. Each container is represented by a box and the number in the box is its priority. This layout is not ordered in such a way that containers can be retrieved without any reshuffling. Containers with priorities 2 and 8 block containers with higher priorities.

An optimal solution for this instance is the sequence of moves $sol = (2,1),(3,2),(4,2)$. Subfigures 2.2b and 2.2c show the intermediate layouts associated with a solution that reaches the final layout of Subigure 2.2d without any blocking container.

## 2.4 Integer linear models with one variable set to describe the container movements

In our models, time is discretized into time segments separated by $T$ time points with the assumption that at most one move can be made in each time segment. Each time point corresponds to a layout of the bay. If the layout at time $t$ is different from that at time $t+1$, a move has taken place in the time segment between $t$ and $t+1$ with $t \in \{1,\dots,T\} = \mathcal{T}$. The first time point corresponds to the initial layout of the bay, and at the last time point $T$ the bay should be arranged, needing no further moves. The value of $T$ has to be strictly greater than the optimal number of moves necessary to order the bay. Otherwise, the problem will be unfeasible. The way in which we set this value is discussed in Section 2.6.

In this section we describe a first set of integer models, based on the use of two types of variables. Variables $x$ describe the layout of the bay at each period and variables $y$ describe the move of one container from one stack to another, defining the transition from one time point to the next. Different ways of defining variables $y$ define different models.

### 2.4.1 A model with 3-indexed $y$ variables, IP3

The two types of binary variables involved in the first integer programming model, IP3, are:

$$x_{shpt} = \begin{cases} 1, & \text{If at time point } t \text{ there is a container in stack } s \text{, tier } h \text{, whose priority is } p; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T}$$
$$y_{skt} = \begin{cases} 1, & \text{If in the time segment between } t \text{ and } t+1 \text{ a move from stack } s \text{ to } k \text{ occur;} \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}, \quad \forall k \in \mathcal{S}\setminus\{s\}, \quad \forall t \in \mathcal{T}\setminus\{T\}$$

Variables $y_{skt}$ represent the simplest way of describing a move between stacks at a given time segment.

The initial layout of the bay is given. Then, we initialize variables $x_{shct}$ at time point $t = 1$ $\forall s, h, c$. Thus, they are parameters of the model. The premarshalling problem can be formulated in the IP3 model as follows:

$$\textbf{Min} \quad \sum_{s=1}^{S}\sum_{\substack{k=1\\k\neq s}}^{S}\sum_{t=1}^{T-1} y_{skt} \tag{2.1}$$

$$\sum_{s=1}^{S}\sum_{h=1}^{H} x_{shpt} = n_p \qquad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T} \tag{2.2}$$

$$\sum_{s=1}^{S}\sum_{\substack{k=1\\k\neq s}}^{S} y_{skt} \leq 1 \qquad \forall t \in \mathcal{T} \setminus \{T\} \tag{2.3}$$

$$\sum_{s=1}^{S}\sum_{\substack{k=1\\k\neq s}}^{S} y_{skt+1} \leq \sum_{s=1}^{S}\sum_{\substack{k=1\\k\neq s}}^{S} y_{skt} \qquad \forall t \in \mathcal{T} \setminus \{T-1, T\} \tag{2.4}$$

$$\sum_{p=1}^{P} x_{s1pt} \leq 1 \qquad \forall s \in \mathcal{S}, \quad \forall t \in \mathcal{T} \setminus \{1\} \tag{2.5}$$

$$\sum_{p=1}^{P} x_{sh+1pt} \leq \sum_{p=1}^{P} x_{shpt} \qquad \forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H} \setminus \{H\}, \quad \forall t \in \mathcal{T} \setminus \{1, T\} \tag{2.6}$$

$$x_{shpt} + \sum_{k=1}^{P} x_{sh+1kt} \leq 1 + x_{shpt+1} \qquad \forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H} \setminus \{H\}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T} \setminus \{T\} \tag{2.7}$$

$$x_{shpt} \leq x_{shpt+1} + \sum_{\substack{k=1\\k\neq s}}^{S} y_{skt} \qquad \forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T} \setminus \{T\} \tag{2.8}$$

$$x_{shpt+1} \leq x_{shpt} + \sum_{\substack{k=1\\k\neq s}}^{S} y_{kst} \qquad \forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T} \setminus \{T\} \tag{2.9}$$

$$\sum_{k=p}^{P} x_{sh+1kT} \leq \sum_{k=p}^{P} x_{shkT} \qquad \forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H} \setminus \{H\}, \quad \forall p \in \mathcal{P} \tag{2.10}$$

$$\sum_{\substack{s=1\\s\neq k}}^{S} y_{skt} + \sum_{\substack{s=1\\s\neq k}}^{S} y_{kst+1} \leq 1 \qquad \forall k \in \mathcal{S}, \quad \forall t \in \mathcal{T} \setminus \{T\} \tag{2.11}$$

$$x_{shpt} \in \{0,1\}, \quad y_{skt} \in \mathcal{R}^{+} \qquad \forall s,k \in \mathcal{S} : s \neq k, \quad \forall h \in \mathcal{H}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T} \tag{2.12}$$

The objective function (2.1) minimizes the total number of moves between stacks necessary to arrange the bay. The number of containers of each priority remains constant over time by constraints (2.2). Constraints (2.3) force each time segment to have at most one move. By (2.4), moves are assigned to the earliest possible times, so if there are no moves in the time segment between $t$ and $t+1$, there will be no moves in any later time segment. The assumption that each slot can contain at most one container is forced by constraints (2.5) for the lowest tier. For the remaining tiers, this assumption is forced by constraints (2.6) and (2.7). Constraints (2.6) also ensure that if tier $h+1$ of a stack $s$ is occupied, all the other slots in lower tiers of stack $s$ are also occupied. In addition, constraints (2.7) ensure that containers that do not occupy the topmost position at a stack cannot be moved between two successive time points. In order to control topmost containers, we have constraints (2.8) and (2.9). On the one hand, if there was a container with priority $p$ in slot $(s,h)$ at time point $t$ but it is not there at time point $t+1$, there has been a move whose origin was stack $s$ in the time segment between $t$ and $t+1$ (constraints (2.8)). On the other hand, if there is a container with priority $p$ in slot $(s,h)$ at time point $t+1$ but it was not there at time point $t$, there has been a move whose destination was stack $s$ in the time segment between $t$ and $t+1$ (constraints (2.9)). The bay is already arranged at time $T$ by constraint (2.10), otherwise the problem is infeasible.

If in the time segment between $t$ and $t+1$ there is a move from stack $s$ to stack $k$, in the following time segment there should not be a move from stack $k$ to another stack $r$ because these moves are dominated by a single move from $s$ to $r$. An example can be seen in Figure 2.3, in which the pair of moves $(3,4),(4,1)$ leads to the same layout as the single move $(3,1)$. Constraints (2.11) avoid these cases, which are known in the literature as transitive moves (Tierney et al. (2017)).

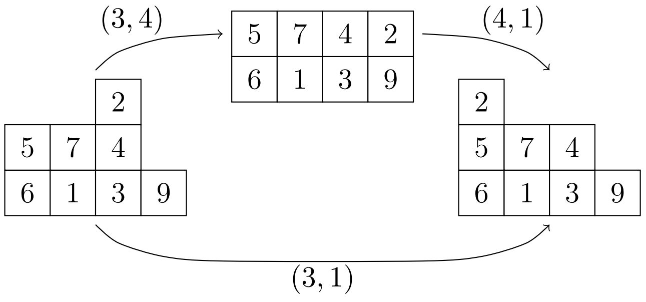


Figure 2.3: Transitive moves

Transitive moves that are not directly successive can be avoided by adding constraints (2.13). Nevertheless, these relations have more theoretical than practical interest because of the large number of constraints involved, which render them impractical and therefore are not included in the model.

$$y_{skt} + \sum_{i=1}^{t'-1} y_{uvt+i} + \sum_{r \in \mathcal{S} \setminus \{k,u,v\}} y_{krt+t'} \leq t' \quad \forall s \in \mathcal{S}, \quad \forall k \in \mathcal{S} \setminus \{s\}, \quad \forall u \in \mathcal{S} \setminus \{s,k\},$$
$$\forall v \in \mathcal{S} \setminus \{s,k,u\}, \quad \forall t \in \mathcal{T} \setminus \{T-1,T\},$$
$$\forall t' \in \mathcal{T} \setminus \{1, T-t+1, \ldots, T\} \tag{2.13}$$

Although in the definition of variables $x_{shpt}$ and $y_{skt}$ we have declared them as binary variables, only variables $x_{shpt}$ have to be binary. If these variables, defining the configuration of the bay at each period, only take $0-1$ values, variables $y_{skt}$ will also take $0-1$ values, even if they are declared as non-negative real variables. The proof is provided in Proposition 1.

**Proposition 1.** *The integrality of $y_{skt}$ variables is inherited from the structure of IP3 model.*

*Proof.* If there is no move at time point $t$, variables $x_{shpt} = x_{shpt+1}$ for all $s,h,p$. In this case, $y_{skt}$ variables will be zero for all $s,k$, as the objective function minimizes their sum and they are declared as positive real numbers.

Otherwise, the move at time point $t$ will involve a container of a given priority $p_1$, initially placed at stack $s_1$ and tier $h_1$, being moved to stack $s_2$, tier $h_2$. Considering the origin of the move: $x_{s_1h_1p_1t} = 1$ and $x_{s_1h_1p_1t+1} = 0$. Since $y_{skt} \in \mathcal{R}^+$, by constraints (2.3) and (2.8) we obtain the following inequalities:

$$1 \overset{(2.8)}{\leq} \sum_{\substack{k=1\\k\neq s_1}}^{S} y_{s_1kt} \leq \sum_{\substack{k=1\\k\neq s_1}}^{S} y_{s_1kt} + \sum_{\substack{s=1\\s\neq s_1}}^{S}\sum_{\substack{k=1\\k\neq s}}^{S} y_{skt} = \sum_{s=1}^{S}\sum_{\substack{k=1\\k\neq s}}^{S} y_{skt} \overset{(2.3)}{\leq} 1 \Longrightarrow \sum_{\substack{s=1\\s\neq s_1}}^{S}\sum_{\substack{k=1\\k\neq s}}^{S} y_{skt} = 0 \quad (2.14)$$

Considering now the destination of the move: $x_{s_2h_2p_1t} = 0$ and $x_{s_2h_2p_1t+1} = 1$. By constraints (2.3) and (2.9):

$$1 \overset{(2.3)(2.9)}{=} \sum_{\substack{s=1\\s\neq s_2}}^{S} y_{ss_2t} = \sum_{\substack{s=1\\s\neq s_1\\s\neq s_2}}^{S} y_{ss_2t} + y_{s_1s_2t} \quad (2.15)$$

We can decompose the final equation of expression (2.14) as follows:

$$0 = \sum_{\substack{s=1\\s\neq s_1}}^{S}\sum_{\substack{k=1\\k\neq s}}^{S} y_{skt} = \sum_{\substack{s=1\\s\neq s_1}}^{S}\sum_{\substack{k=1\\k\neq s\\k\neq s_2}}^{S} y_{skt} + \sum_{\substack{s=1\\s\neq s_1\\s\neq s_2}}^{S} y_{ss_2t} \Longrightarrow \sum_{\substack{s=1\\s\neq s_1\\s\neq s_2}}^{S} y_{ss_2t} = 0 \Longrightarrow y_{s_1s_2t} = 1$$

If there is a move from stack $s_1$ to stack $s_2$ and variables $x_{skpt}$ take binary variables, $y_{s_1s_2t} = 1$ and the remaining $y_{skt} = 0$. $\square$

### 2.4.2 A model with 4-indexed $y$ variables, IP4

Variables $y_{skt}$ of model IP3 do not control the priority of the container being moved at each time segment. Model IP3 is correct and produces feasible solutions, but in its linear relaxation the relationship between variables $x_{shpt}$ and $y_{skt}$ is very weak, producing bad lower bounds and long solution times. One way of strengthening the $x - y$ relationship is to define $y$ variables with a fourth index indicating the priority of the container being moved. A new model, IP4, can be defined using the variables:

$$x_{shpt} = \begin{cases} 1, & \text{if at time point } t \text{ there is a container in position } (s,h) \text{ with priority } p;\\ 0, & \text{Otherwise;}\end{cases}$$
$$\forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T}$$

$$y_{skpt} = \begin{cases} 1, & \text{if in the segment between } t \text{ and } t+1, \text{ a container with priority } p \text{ is}\\ & \text{moved from stack } s \text{ to } k;\\ 0, & \text{Otherwise;}\end{cases}$$
$$\forall s \in \mathcal{S}, \quad \forall k \in \mathcal{S}\setminus\{s\}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T}\setminus\{T\}$$

Variables $y_{sk1T-1}$ turn into parameters because containers with priority 1 will not be involved in the last move, hence $y_{sk1T-1} = 0, \forall s, k$. The objective function and constraints of model IP4 are very similar to that of model IP3, considering that each variable $y_{skt}$ is now decomposed into $P$ variables $y_{skpt}$. Constraints (2.8) and (2.9) are substituted by constraints:

$$x_{shpt} \leq x_{shpt+1} + \sum_{\substack{k=1\\k\neq s}}^{S} y_{skpt} \quad \forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T} \setminus \{T\} \tag{2.16}$$

$$x_{shpt+1} \leq x_{shpt} + \sum_{\substack{k=1\\k\neq s}}^{S} y_{kspt} \quad \forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T} \setminus \{T\} \tag{2.17}$$

in which variables $x_{shpt}$ and $y_{skpt}$ correspond now to containers with the same priority $p$, making their relationship tighter. Besides that, we can now avoid same priority symmetries by adding constraints (2.18).

$$\sum_{\substack{k=1\\k\neq s}}^{S} y_{skpt} + \sum_{\substack{k=1\\k\neq s}}^{S} y_{kspt+1} \leq 1 \quad \forall s \in \mathcal{S}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T} \setminus \{T-1, T\} \tag{2.18}$$

Rules regarding same priority symmetries are introduced in Tanaka and Tierney (2018). If in the time segment between $t$ and $t+1$, a container of priority $p$ is moved from stack $s$, in the following time segment there should not be a move of a container with priority $p$ to that stack $s$. An example can be seen in Figure 2.4, in which the pair of moves $(1,4), (2,1)$ lead to the same layout as the single move $(2,4)$.

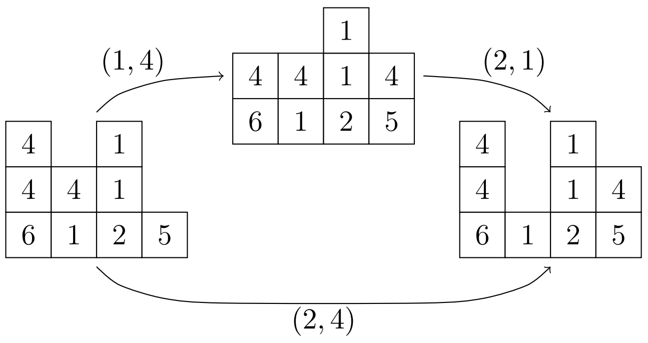


Figure 2.4: Same group symmetries

Same priority symmetries can be extended to cases in which the stacks are not used during an intermediate sequence by adding constraints (2.19). The interest of these constraints is just theoretical as in the general case of transitive moves.

$$y_{skpt} + \sum_{g=1}^{P} \sum_{i=1}^{t'-1} y_{uvgt+i} + \sum_{r \in \mathcal{S} \setminus \{s,u,v\}} y_{rspt+t'} \leq t' \quad \forall s \in \mathcal{S}, \quad \forall k \in \mathcal{S} \setminus \{s\}, \quad \forall u \in \mathcal{S} \setminus \{s,k\},$$
$$\forall v \in \mathcal{S} \setminus \{s,k,u\}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T} \setminus \{T\},$$
$$\forall t' \in \mathcal{T} \setminus \{1, T-t+1, \ldots, T\} \tag{2.19}$$

### 2.4.3 A model with 5-indexed $y$ variables, IP5

The relationship between variables $x_{shpt}$, describing the bay configuration at time $t$, and variables $y_{skt}$, describing the moves in the IP3 model, can be further strengthened if variables $y_{skt}$ include not only the origin and destination stacks of the move, but also the origin and destination tiers. A new model, IP5, can be defined using the variables:

$$x_{shpt} = \begin{cases} 1, & \text{If at time point } t \text{ there is a container in position } (s,h) \text{ with priority } p; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T}$$
$$y_{shket} = \begin{cases} 1, & \text{If in the time segment between } t \text{ and } t+1 \text{, a move from } (s,h) \text{ to } (k,e) \\ & \text{takes place;} \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}, \quad \forall k \in \mathcal{S} \setminus \{s\}, \quad \forall h, e \in \mathcal{H}, \quad \forall t \in \mathcal{T} \setminus \{T\}$$

An optimal sequence will not include moves from the lowest tier in the last time segment, hence $y_{s1keT-1} = 0, \forall s, k, e$. Taking as a reference the initial model IP3, each variable $y_{skt}$ is now decomposed into $H^2$ variables $y_{shket}$, where $h$ is the tier in which the container is before the move and $e$ the tier after the move.

Constraints (2.5) can be strengthened by adding the moves to the lowest tier (constraints 2.20). In this way, each slot can contain at most one container and if the slot is occupied in the time point $t$, it cannot receive a container.

$$\sum_{p=1}^{P} x_{s1pt} + \sum_{\substack{k=1 \\ k\neq s}}^{S} \sum_{e=1}^{H} y_{kes1t} \leq 1 \quad \forall s \in \mathcal{S}, \quad \forall t \in \mathcal{T} \setminus \{1\} \tag{2.20}$$

Constraints (2.21) ensure the following three assumptions: each slot can contain at most one container in the tiers up to tier 1; if one slot is occupied, the slots below it should also be occupied; and only topmost containers can be moved at each time segment. Therefore, IP5 includes constraints (2.21), and constraints (2.6) and (2.7) are no longer needed.

$$\sum_{p=1}^{P} x_{sh+1pt} + \sum_{\substack{k=1 \\ k\neq s}}^{S} \sum_{e=1}^{H} y_{shket} + \sum_{\substack{k=1 \\ k\neq s}}^{S} \sum_{e=1}^{H} y_{kesh+1pt} \leq \sum_{p=1}^{P} x_{shpt} \tag{2.21}$$
$$\forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H} \setminus \{H\}, \quad \forall t \in \mathcal{T} \setminus \{T\}$$

Considering the origin and destination tiers of each move, model IP5 can be strengthened by including constraints (2.22) to make the relationship between $x$ and $y$ variables stronger. If two consecutive levels of a stack are occupied at a given time, no container can be moved to any of these levels and no container can be moved from a lower position of these levels in the next time segment.

$$\sum_{p=1}^{P} x_{shpt} + \sum_{p=1}^{P} x_{sh+1pt} + \sum_{\substack{k=1\\k\neq s}}^{S}\sum_{e=1}^{H} y_{shket} + \sum_{\substack{k=1\\k\neq s}}^{S}\sum_{e=1}^{H} y_{kesh+1pt} + \sum_{\substack{k=1\\k\neq s}}^{S}\sum_{e=1}^{H} y_{keshpt} \leq 2 \qquad (2.22)$$

$$\forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H}\setminus\{H\}, \quad \forall t \in \mathcal{T}\setminus\{T\}$$

Constraints (2.2) and (2.10) involving only $x_{shpt}$ do not change. In the objective function (2.1) and in all other constraints involving variables $y_{shket}$, two more sums, $\forall h, \forall e \in \mathcal{H}$, have to be added.

### 2.4.4 A model with 6-indexed $y$ variables, IP6

An alternative model, IP6, uses a new type of variables $y$ describing completely the move of a container, including the period $t$, the origin and destination stacks, $s, k$, the origin and destination tiers, $h, e$, and the priority $p$. The variables are:

$$x_{shpt} = \begin{cases} 1, & \text{If at time point } t \text{ there is a container in position } (s,h) \text{ with priority } p;\\ 0, & \text{Otherwise;}\end{cases}$$
$$\forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T}$$

$$y_{shkept} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1, \text{ a container with priority } p\\ & \text{is moved from } (s,h) \text{ to } (k,e);\\ 0, & \text{Otherwise;}\end{cases}$$
$$\forall s \in \mathcal{S}, \quad \forall k \in \mathcal{S}\setminus\{s\}, \quad \forall h, e \in \mathcal{H}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T}\setminus\{T\}$$

We fix variables $y_{shke1T-1}$ and $y_{s1kepT-1}$ to zero because these moves are not possible in the last time segment. Taking as a reference model IP3, the two inequalities (2.8) and (2.9) controlling the topmost containers of each stack are substituted by the equality:

$$x_{shpt} + \sum_{\substack{k=1\\k\neq s}}^{S}\sum_{e=1}^{H} y_{keshpt} = x_{shpt+1} + \sum_{\substack{k=1\\k\neq s}}^{S}\sum_{e=1}^{H} y_{shkept} \ \forall s \in \mathcal{S}, \ \forall h \in \mathcal{H}, \ \forall p \in \mathcal{P}, \ \forall t \in \mathcal{T}\setminus\{T\} \qquad (2.23)$$

Constraints (2.2), keeping fixed the number of containers of each priority at each stack, are no longer necessary. Moreover, constraints (2.18) avoiding same priority symmetries are also included in this formulation.

## 2.5 Integer linear models with two variable sets to describe the container movements

The models described in the previous section were progressively strengthening the relationship between bay configurations (variables $x$) and the moves producing these configurations (variables $y$) but at the cost of significantly increasing the number of variables. Therefore, the advantage of tighter relationships, tighter linear relaxations, and tighter lower bounds can be partially offset by the computational cost of solving much larger models. In this section we keep the structure of the models described above, but change the definition of the variables in such a way that tighter relationships between configurations and moves do not require large increases in the number of variables. Each model of the previous section will have its counterpart in this new approach.

### 2.5.1 A model with split 2-indexed variables, IPS3

Variables $x_{shpt}$, describing the configuration of the bay at each period are maintained. However, former variables $y_{skt}$ describing the move are split into two variables, one variable $w_{st}$ indicating the stack origin of the move, and another variable $z_{kt}$ indicating the destination. Instead of $S \times (S-1) \times T$ variables for the moves, the new $IPS3$ model has $2 \times S \times T$. The definition of the variables is:

$$x_{shpt} = \begin{cases} 1, & \text{If at time point } t \text{ there is a container in position } (s,h) \text{ with priority } p; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T}$$

$$w_{st} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1, \text{ a container is moved from stack } s; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}, \quad \forall t \in \mathcal{T} \setminus \{T\}$$

$$z_{st} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1, \text{ a container is moved to stack } s; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}, \quad \forall t \in \mathcal{T} \setminus \{T\}$$

Using these variables, the model $IPS3$ inherits the constraints of $IP3$, replacing $\sum_{\substack{k=1 \\ k \neq s}}^{S} y_{skt}$ by $w_{st}$, and $\sum_{\substack{k=1 \\ k \neq s}}^{S} y_{kst}$ by $z_{st}$. The objective function can be expressed in terms of either variables $w_{st}$ or variables $z_{st}$, because at each time $t$ the number of containers leaving their initial stacks equals the number of containers arriving to a new stack. By including constraints (2.24), a stack will not be origin and destination of a move at one time segment.

$$z_{st} + w_{st} \leq 1 \quad \forall s \in \mathcal{S}, \quad \forall t \in \mathcal{T} \setminus \{T\}, \tag{2.24}$$

### 2.5.2 A model using priorities to define split 3-indexed variables, IPS4

The relationship between variables $x_{shpt}$ and variables $w_{st}$, $z_{st}$ is reinforced if these variables include the reference to the priority $p$ of the container being moved. An integer model IPS4, mirroring the IP4 model but splitting variables $y_{skpt}$ into $w_{spt}$ and $z_{spt}$ can be defined using variables:

$$x_{shpt} = \begin{cases} 1, & \text{If at time point } t \text{ there is a container in position } (s,h) \text{ with priority } p; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T}$$

$$w_{spt} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1 \text{, a container with priority } p \\ & \text{is moved from stack } s; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T} \setminus \{T\}$$

$$z_{spt} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1 \text{, a container with priority } p \\ & \text{is moved to stack } s; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T} \setminus \{T\}$$

The model IPS4 is very similar to IP4, considering that expressions $\sum_{\substack{k=1 \\ k\neq s}}^{S} y_{skpt}$ are now substituted by $w_{spt}$, and $\sum_{\substack{k=1 \\ k\neq s}}^{S} y_{kspt}$ by $z_{spt}$. We also strengthen the formulation by adding constraints (2.25).

$$\sum_{p=1}^{P} z_{spt} + \sum_{p=1}^{P} w_{spt} \leq 1 \quad \forall s \in \mathcal{S}, \quad \forall t \in \mathcal{T} \setminus \{T\} \tag{2.25}$$

The IP4 model had $S \times (S-1) \times P \times T$ variables for the moves, while the new *IPS4* model only needs $2 \times S \times P \times T$

### 2.5.3 A model using tiers to define split 3-indexed variables, IPS5

The IP5 model using 5-index variables $y_{shket}$ can also be substituted by a model using split 3-index variables $w_{sht}$ and $z_{ket}$. The variables of IPS5 are:

$$x_{shpt} = \begin{cases} 1, & \text{If at time point } t \text{ there is a container in position } (s,h) \text{ with priority } p; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T}$$

$$w_{sht} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1 \text{, a container is moved from } (s,h); \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H}, \quad \forall t \in \mathcal{T} \setminus \{T\}$$

$$z_{sht} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1 \text{, a container is moved to } (s,h); \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H}, \quad \forall t \in \mathcal{T} \setminus \{T\}$$

Taking as a reference the IP5 model, expressions $\sum_{\substack{k=1 \\ k\neq s}}^{S} \sum_{e=1}^{H} y_{shket}$ are now simplified to $w_{sht}$, and $\sum_{\substack{k=1 \\ k\neq s}}^{S} \sum_{e=1}^{H} y_{kesht}$ to $z_{sht}$. Constraints avoiding a stack to be origin and destination at the same time segment are no necessary because it is already considered by the equivalent of constraints (2.22) in IPS5.

### 2.5.4 A model with split 4-indexed variables, IPS6

Similarly, variables $y_{shkept}$ of the IP6 model can be decomposed into split 4-indexed variables $w_{shpt}$ and $z_{kept}$, defining a new model, IPS6. Its variables are:

$$x_{shpt} = \begin{cases} 1, & \text{If at time point } t \text{ there is a container in position } (s,h) \text{ with priority } p; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T}$$

$$w_{shpt} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1, \text{ a container with priority } p \\ & \text{is moved from } (s,h); \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T} \setminus \{T\}$$

$$z_{shpt} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1, \text{ a container with priority } p \\ & \text{is moved to } (s,h); \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T} \setminus \{T\}$$

This approach cuts the number of variables down from $S(S-1)H^2PT$ to $2SHPT$. For completeness, we present the whole IPS6 model which is our best proposal for solving the premarshalling problem.

$$\textbf{Min} \quad \sum_{s=1}^{S}\sum_{h=1}^{H}\sum_{p=1}^{P}\sum_{t=1}^{T-1} z_{shpt} \tag{2.26}$$

$$\sum_{s=1}^{S}\sum_{h=1}^{H} w_{shpt} = \sum_{s=1}^{S}\sum_{h=1}^{H} z_{shpt} \qquad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T} \setminus \{T\} \tag{2.27}$$

$$\sum_{s=1}^{S}\sum_{h=1}^{H}\sum_{p=1}^{P} w_{shpt} \leq 1, \quad \sum_{s=1}^{S}\sum_{h=1}^{H}\sum_{p=1}^{P} z_{shpt} \leq 1 \quad \forall t \in \mathcal{T} \setminus \{T\} \tag{2.28}$$

$$\sum_{s=1}^{S}\sum_{h=1}^{H}\sum_{p=1}^{P} w_{shpt+1} \leq \sum_{s=1}^{S}\sum_{h=1}^{H}\sum_{p=1}^{P} w_{shpt} \qquad \forall t \in \mathcal{T} \setminus \{T-1, T\} \tag{2.29}$$

$$\sum_{p=1}^{P} x_{s1pt} + \sum_{p=1}^{P} z_{s1pt} \leq 1 \qquad \forall s \in \mathcal{S}, \quad \forall t \in \{2, \dots, T\} \tag{2.30}$$

$$\sum_{k=p}^{P} x_{sh+1kT} \leq \sum_{k=p}^{P} x_{shkT} \qquad \forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H} \setminus \{H\}, \quad \forall p \in \mathcal{P} \tag{2.31}$$

$$x_{shpt} + z_{shpt} = x_{shpt+1} + w_{shpt} \qquad \forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T} \setminus \{T\} \tag{2.32}$$

$$\sum_{p=1}^{P} x_{sh+1pt} + \sum_{p=1}^{P} w_{shpt} + \sum_{p=1}^{P} z_{sh+1pt} \leq \sum_{p=1}^{P} x_{shpt}$$
$$\forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H} \setminus \{H\}, \quad \forall t \in \mathcal{T} \setminus \{T\} \tag{2.33}$$

$$\sum_{p=1}^{P} x_{shpt} + \sum_{p=1}^{P} x_{sh+1pt} + \sum_{p=1}^{P} w_{shpt} + \sum_{p=1}^{P} z_{sh+1pt} + \sum_{p=1}^{P} z_{shpt} \leq 2$$
$$\forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H} \setminus \{H\}, \quad \forall t \in \mathcal{T} \setminus \{T\} \tag{2.34}$$

$$\sum_{h=1}^{H}\sum_{p=1}^{P} z_{shpt} + \sum_{h=1}^{H}\sum_{p=1}^{P} w_{shpt+1} \leq 1 \qquad \forall s \in \mathcal{S}, \quad \forall t \in \mathcal{T} \setminus \{T-1, T\} \tag{2.35}$$

$$\sum_{h=1}^{H} w_{shpt} + \sum_{h=1}^{H} z_{shpt+1} \leq 1 \qquad \forall s \in \mathcal{S}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T} \setminus \{T-1, T\}, \tag{2.36}$$

$$w_{sh1(T-1)} = 0, \quad z_{sh1(T-1)} = 0, \qquad \forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H}, \tag{2.37}$$

$$w_{s1p(T-1)} = 0, \qquad \forall s \in \mathcal{S}, \quad \forall p \in \mathcal{P} \setminus \{1\} \tag{2.38}$$

$$x_{shpt} \in \{0,1\}, \quad w_{shpt}, z_{shpt} \in \mathcal{R}^{+} \qquad \forall s \in \mathcal{S}, \quad \forall h \in \mathcal{H}, \quad \forall p \in \mathcal{P}, \quad \forall t \in \mathcal{T}, \tag{2.39}$$

Taking again as a reference the IP6 model, the main difference is that constraints (2.2) are substituted by constraints (2.27). It allows a tighter relation between $w$ and $z$ variables.

## 2.6 An iterative procedure for solving the integer models

All models described in the previous sections require to determine a value of $T$, the maximum number of moves needed to rearrange the bay, plus one, because at the last period the bay must be completely ordered. While the lower bound proposed by Tanaka and Tierney (2018) is available and can be used in the models, the use of an upper bound provided by any of the heuristics developed so far is not clear. On the one hand, the number of variables and constraints, and therefore the performance of the models, depend directly on the value of $T$. Using the upper bound provided by a heuristic will make the performance of the model dependent on the performance of the heuristic. On the other hand, there is the unsolved question of deciding whether a premarshalling instance has a feasible solution or not. It is a difficult question, because it depends not only on the dimensions of the bay and the number of containers, but also on the positions of the containers. The vast majority of heuristic algorithms avoid the question by adding extra tiers to the instance, sometimes at the beginning of the procedure, sometimes when they are needed to allow further movements to be made. However, in a real situation, the maximum tier $H$ of the bay is imposed by the height of the crane, and adding additional levels to the bay may not be possible. Moreover, a value obtained by modifying the original instance cannot be considered a valid upper bound. Heuristic algorithms that do not increase the bay size, as that used by de Melo da Silva et al. (2018), do not guarantee that a feasible solution will be found in all kind of instances. In de Melo da Silva et al. (2018), they test their PMP$_{m1}$ model on the CV dataset from Caserta and Voß (2009). CV instances are filled up to the same height and have two empty tiers on top of each stack. These two empty tiers make the heuristic algorithm proposed by de Melo da Silva et al. (2018) work. Nonetheless, it fails on instances with a different structure, for example, instances with a fill percentage 75% on the EMM dataset from Expósito-Izquierdo et al. (2012), going into a loop and not providing a value for $T$.

Therefore, we have developed an iterative procedure that does not require the use of an upper bound. Initially, we set $T = LB + 1$. If the model gives a feasible solution using this value of $T$, it is optimal. Otherwise, we set $T = T + 1$ and solve the model again. The procedure is repeated for increasing values of $T$ until a feasible solution is obtained, producing an optimal solution. When we solve the models using this iterative procedure, constraints that assign the moves to the earliest possible times are no longer necessary. Although Proposition 1 proves that variables describe the movements can be defined as real variables, we keep them as binary variables because that helps to identify unfeasible instances faster.

## 2.7 Computational experiments

To test the performance of the mathematical models, exact solutions were obtained using CPLEX version 12.7 considering 2 threads and a time limit of 3600 seconds. We conducted all computational experiments on virtual machines with 4 virtual processors and 8 GBytes of RAM running Windows 10 Enterprise 64 bits. Virtual machines are run in an OpenStack virtualization platform supported by 12 blades, each with four 12-core AMD Opteron Abu Dhabi 6344 processors running at 2.6 GHz and 256 GBytes of RAM, for a total of 576 cores and 3 TBytes of RAM.

### 2.7.1 Test instances

We have used instances from four datasets of the literature:

**EMM dataset** The original instances from Expósito-Izquierdo et al. (2012) were lost, but they were generated again by Tierney et al. (2017), using the same distributions. There are several categories of instances, with 4 tiers; 4, 7 or 10 stacks; and container fill percentages of 0.50, 0.75 and 1. In this study we use the instances with fill percentages 0.50 and 0.75, without adding any additional tier. Instances with fill percentage 1 cannot be solved unless some tiers are added on top of each stack, thus changing the fill percentage, and have not been considered here.

**ZJY dataset** Zhang et al. (2015) generated 100 instances, divided into 5 categories of 20 instances each, containing 6, 7, 8 or 9 stacks and 4 tiers, and 6 stacks and 5 tiers. Although generation details are not provided, it can be observed that there may be several containers with the same priority, and the stacks are usually filled up to $|H| - 1$ tiers.

**CV dataset** In this dataset from Caserta and Voß (2009), all containers have different priorities, all stacks are filled to the same height, and there are two empty tiers on top of each stack. The instances used in this study range in size from 3 stacks and 5 tiers up to 7 stacks and 6 tiers, totalling 400 instances.

**BZ dataset** These instances from van Brink and van der Zwaan (2014) range in size from 3 stacks and 4 tiers up to 9 stacks and 6 tiers, filled 50% or 70% with containers with 2, 3, or 6 different priorities, with a total of 960 instances.

### 2.7.2 Performance of the proposed integer programming models

**Results on the EMM dataset** Table 2.1 shows the number of optimal solutions obtained by the models described in Sections 2.4 and 2.5 on the 450 instances of the *EMM* dataset. Concerning the initial models, their performance increases from IP3 to IP6. Looking at the models using split variables, their performance also increases from IPS3 to IPS6. The IPS6 model is clearly better with a moderate increase in the number of variables. All the models solve well the instances with a low fill percentage, but there are significant differences between them when solving larger ones. The first row in Table 2.2 completes the information by showing the average running time in seconds computed on the set of instances optimally solved by all the models on the EMM dataset. IPS6 is faster than the other split-variable models and they are in turn faster than the corresponding *IP3-IP6* models.

Table 2.1: Number of optimal solutions obtained by the proposed models on the EMM dataset grouped by number of stacks (S) and fill percentage (f).

| S | f | # | IP3 | IP4 | IP5 | IP6 | IPS3 | IPS4 | IPS5 | IPS6 |
|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 0.50 | 75 | 75 | 75 | 75 | 75 | 75 | 75 | 75 | 75 |
| 4 | 0.75 | 75 | 50 | 50 | 50 | 75 | 50 | 75 | 75 | 75 |
| 7 | 0.50 | 75 | 75 | 75 | 75 | 75 | 75 | 75 | 75 | 75 |
| 7 | 0.75 | 75 | 31 | 36 | 40 | 44 | 31 | 40 | 42 | 51 |
| 10 | 0.50 | 75 | 72 | 75 | 75 | 75 | 75 | 75 | 75 | 75 |
| 10 | 0.75 | 75 | 11 | 17 | 18 | 24 | 12 | 24 | 23 | 40 |
| | Total | 450 | 314 | 328 | 333 | 368 | 318 | 364 | 365 | 391 |

Table 2.2: Average running times in seconds only on the instances (O) of the studied dataset optimally solved by the eight proposed models.

| Dataset | # | O | IP3 | IP4 | IP5 | IP6 | IPS3 | IPS4 | IPS5 | IPS6 |
|---|---|---|---|---|---|---|---|---|---|---|
| EMM | 450 | 312 | 94.58 | 20.38 | 21.52 | 9.29 | 52.78 | 7.82 | 6.51 | 1.10 |
| ZJY | 100 | 15 | 575.78 | 95.32 | 75.17 | 39.89 | 243.68 | 41.29 | 28.04 | 8.10 |
| CV | 400 | 64 | 496.42 | 214.92 | 118.51 | 46.08 | 138.98 | 54.18 | 41.68 | 17.00 |
| BZ | 960 | 745 | 106.42 | 41.84 | 27.96 | 16.81 | 65.96 | 15.48 | 10.86 | 1.92 |
| | | Average | 131.34 | 46.40 | 31.92 | 16.70 | 68.80 | 15.90 | 11.63 | 2.63 |

**Results on the ZJY dataset** Table 2.3 contains the number of optimal solutions obtained by the eight models on the 100 instances of the ZJY dataset. The table exhibits two main characteristics. On the one hand, instances with 5 tiers are harder for all the models. On the other hand, the performance of the models in the other four categories follows the same pattern of the previous subsection. Again split-variable *IPS3-IPS6* models are increasingly better, outperforming *IP3-IP6* models. Concerning running times, included in the second row of Table 2.2, the differences between models are even larger than in the previous dataset. Models *IPS3-IPS6* are much faster than the other models, with IPS6 being again the fastest alternative. IPS6 cuts the CPU time down on average by 99% and 80% with respect to IP3 and IP6, respectively.

**Results on the CV dataset** Table 2.4 shows the number of optimal solutions obtained by the models on the CV instances. These instances are considered particularly difficult,

Table 2.3: Number of optimal solutions obtained by the proposed models on the ZJY dataset, grouped by number of tiers (H) and number of stacks (S).

| H | S | # | IP3 | IP4 | IP5 | IP6 | IPS3 | IPS4 | IPS5 | IPS6 |
|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 6 | 20 | 6 | 9 | 11 | 16 | 9 | 12 | 12 | 20 |
| 4 | 7 | 20 | 2 | 9 | 12 | 14 | 4 | 13 | 14 | 17 |
| 4 | 8 | 20 | 4 | 8 | 10 | 13 | 7 | 12 | 12 | 18 |
| 4 | 9 | 20 | 3 | 3 | 6 | 11 | 3 | 6 | 7 | 17 |
| 5 | 6 | 20 | 0 | 1 | 1 | 3 | 0 | 1 | 2 | 9 |
| | Total | 100 | 15 | 30 | 40 | 57 | 23 | 44 | 47 | 81 |

because containers must be handled several times (Tanaka and Tierney (2018)). They are really difficult for the integer linear models tested, especially with a higher number of tiers. It is worth noting that, unlike previous tests, IPS4 outperforms IPS5. Apart from that, a clear trend among *IP3-IP6* and *IPS3-IPS6* models is again observed looking at average running times on the third row of Table 2.2.

,

Table 2.4: Number of optimal solutions obtained by the models on the CV dataset, grouped by number of tiers (H) and number of stacks (S).

| H | S | # | IP3 | IP4 | IP5 | IP6 | IPS3 | IPS4 | IPS5 | IPS6 |
|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 3 | 40 | 33 | 34 | 38 | 40 | 35 | 39 | 39 | 40 |
| 5 | 4 | 40 | 20 | 29 | 29 | 36 | 23 | 34 | 33 | 38 |
| 5 | 5 | 40 | 7 | 15 | 18 | 25 | 12 | 26 | 24 | 32 |
| 5 | 6 | 40 | 3 | 5 | 6 | 12 | 3 | 12 | 9 | 23 |
| 5 | 7 | 40 | 0 | 2 | 1 | 5 | 1 | 2 | 3 | 11 |
| 5 | 8 | 40 | 0 | 1 | 1 | 2 | 0 | 1 | 1 | 7 |
| 6 | 4 | 40 | 1 | 2 | 3 | 5 | 2 | 5 | 5 | 9 |
| 6 | 5 | 40 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 2 |
| 6 | 6 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 6 | 7 | 40 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | Total | 400 | 64 | 88 | 96 | 126 | 76 | 120 | 114 | 163 |

**Results on the BZ dataset** Table 2.5 shows the number of optimal solutions obtained by the eight models tested on the BZ instances. As in previous tests, instances with fewer tiers are easier to solve even when the number of stacks increases. Comparing the models, we observe that *IPS4-IPS6*, the split-variable models from *IP4-IP6*, obtain the largest number of optimal solutions. Nevertheless, IP3 outperforms IPS3. For the subset of instances with 6 tiers, the number of optimal solutions ranges from 320-352 for *IP4-IP6*, and 337-413 for *IPS4-IPS6*. Table 2.2 shows the average running times in seconds for the 745 instances solved by all the models. The *IPS3-IPS6* models are the fastest and *IPS6* is the fastest of them all.

In summary, splitting the variables that describe the moves appears to be a very useful strategy. The relationship between variables describing the configurations and those describing the moves can be made tighter with a moderate increase in the number of variables, as can be seen in Table 2.6.

Table 2.5: Number of optimal solutions obtained by the proposed models on the BZ dataset, grouped by number of tiers (H), number of stacks (S), and fill percentage (f).

| H | S | f | # | IP3 | IP4 | IP5 | IP6 | IPS3 | IPS4 | IPS5 | IPS6 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 3 | 0.50 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 |
| 4 | 3 | 0.70 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 |
| 4 | 4 | 0.50 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 |
| 4 | 4 | 0.70 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 |
| 4 | 7 | 0.50 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 |
| 4 | 7 | 0.70 | 60 | 55 | 59 | 60 | 60 | 59 | 60 | 60 | 60 |
| 4 | 9 | 0.50 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 |
| 4 | 9 | 0.70 | 60 | 44 | 49 | 54 | 56 | 47 | 59 | 57 | 60 |
| 6 | 3 | 0.50 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 | 60 |
| 6 | 3 | 0.70 | 60 | 47 | 48 | 54 | 56 | 46 | 52 | 55 | 60 |
| 6 | 4 | 0.50 | 60 | 55 | 58 | 58 | 60 | 53 | 59 | 60 | 60 |
| 6 | 4 | 0.70 | 60 | 25 | 27 | 32 | 35 | 24 | 29 | 35 | 44 |
| 6 | 7 | 0.50 | 60 | 49 | 51 | 52 | 53 | 49 | 54 | 55 | 59 |
| 6 | 7 | 0.70 | 60 | 18 | 21 | 24 | 25 | 16 | 21 | 27 | 37 |
| 6 | 9 | 0.50 | 60 | 40 | 47 | 46 | 48 | 37 | 51 | 52 | 60 |
| 6 | 9 | 0.70 | 60 | 10 | 8 | 15 | 15 | 5 | 11 | 20 | 33 |
| | | Total | 960 | 763 | 788 | 815 | 828 | 756 | 816 | 841 | 893 |

Table 2.6: Number of variables describing the moves involved in the eight new models.

| Model | Variables | Model | Variables |
|---|---|---|---|
| IP3 | $S(S-1)T$ | IPS3 | $2ST$ |
| IP4 | $S(S-1)PT$ | IPS4 | $2SPT$ |
| IP5 | $S(S-1)H^2T$ | IPS5 | $2SHT$ |
| IP6 | $S(S-1)H^2PT$ | IPS6 | $2SHPT$ |

### 2.7.3 Comparison with the state-of-the-art

There are several exact methods dealing with the premarshalling problem, being the algorithm PT$^A$ described in Chapter 3 the most competitive (Tanaka et al., 2019). It solves all the instances tested here in seconds, well below the time limit of one hour per instance. Nonetheless, as discussed in Section 2.1, mathematical models are more flexible and easier to implement which motivates their use as an alternative to sophisticated exact methods. In this section, we compare the performance of the proposed model with the state-of-the-art mathematical models proposed by Lee and Hsu (2007) and more recently by de Melo da Silva et al. (2018). We re-implemented the models proposed by Lee and Hsu (2007) and by de Melo da Silva et al. (2018) and they have also been tested using the iterative procedure presented in Section 2.6.

**EMM dataset** The performance of the models proposed by Lee and Hsu (2007) and de Melo da Silva et al. (2018) and of IPS6 model on the EMM dataset is shown in Table 2.7. Our model outperforms state-of-the-art models by solving 33% and 2% more instances to optimality than Lee and Hsu (2007) model and PMP$_{m1}$, respectively. Moreover, IPS6 is much faster than the other models.

**ZJY dataset** On the ZJY dataset, while the model proposed by Lee and Hsu (2007) solves 11 instances to optimality, PMP$_{m1}$ solves 79 and IPS6 solves 81, as shown in Table 2.8. The

Table 2.7: Performance of the model proposed by Lee and Hsu, the PMP$_{m1}$ model from de Melo da Silva et al. (2018), and IPS6 on the EMM dataset, grouped by number of stacks (S) and fill percentage (f).

| | | | Lee and Hsu | | PMP$_{m1}$ | | IPS6 | |
|---|---|---|---|---|---|---|---|---|
| S | f | # | Opt. | CPU(s) | Opt. | CPU(s) | Opt. | CPU(s) |
| 4 | 0.50 | 75 | 75 | 0.03 | 75 | 0.01 | 75 | 0.01 |
| 4 | 0.75 | 75 | 50 | 15.40 | 75 | 47.15 | 75 | 27.15 |
| 7 | 0.50 | 75 | 74 | 65.90 | 75 | 0.78 | 75 | 0.60 |
| 7 | 0.75 | 75 | 26 | 693.73 | 49 | 190.05 | 51 | 99.85 |
| 10 | 0.50 | 75 | 63 | 241.20 | 75 | 3.44 | 75 | 1.87 |
| 10 | 0.75 | 75 | 7 | 696.09 | 35 | 649.84 | 40 | 230.91 |
| Total/Avg | | 450 | 295 | 148.32 | 384 | 93.52 | 391 | 42.33 |

The column “#" gives the total number of instances tested and “Opt." represent the number of optimal and feasible solutions obtained by each model. Column “CPU(s)" represents the average running time (in seconds) on the instances optimally solved by each model.

main difference between PMP$_{m1}$ and IPS6 lies again on the average running times, existing a clear superiority of model IPS6.

Table 2.8: Performance of the model proposed by Lee and Hsu, the PMP$_{m1}$ model from de Melo da Silva et al. (2018), and IPS6 on the ZJY dataset, grouped by number of tiers (H) and number of stacks (S).

| | | | Lee and Hsu | | PMP$_{m1}$ | | IPS6 | |
|---|---|---|---|---|---|---|---|---|
| H | S | # | Opt. | CPU(s) | Opt. | CPU(s) | Opt. | CPU(s) |
| 4 | 6 | 20 | 4 | 777.17 | 20 | 692.52 | 20 | 323.69 |
| 4 | 7 | 20 | 1 | 3256.73 | 17 | 273.00 | 17 | 201.91 |
| 4 | 8 | 20 | 4 | 496.86 | 17 | 380.08 | 18 | 341.26 |
| 4 | 9 | 20 | 2 | 1097.42 | 17 | 920.02 | 17 | 686.80 |
| 5 | 6 | 20 | 0 | - | 8 | 1011.21 | 9 | 694.39 |
| Total/Avg | | 100 | 11 | 958.88 | 79 | 616.24 | 81 | 419.43 |

The column “#" gives the total number of instances tested and “Opt." represent the number of optimal and feasible solutions obtained by each model. Column “CPU(s)" represents the average running time (in seconds) on the instances optimally solved by each model. The notation “-” is used when the averages cannot be calculated.

**CV dataset** IPS6 increases the number of optimal solution on average by 176% with respect to the model proposed by Lee and Hsu (2007) and by 5% with respect to PMP$_{m1}$, as shown in Table 2.9. Our model finds more optimal solutions on the categories: 5-5, 5-7, 5-8, and 6-4.

**BZ dataset** Table 2.10 shows the performance of the models on BZ dataset. The Lee and Hsu model solves 717, PMP$_{m1}$ solves 888, and IPS6 solves 893 out of the 960 BZ instances. We completely solve two more categories than de Melo da Silva et al. (2018).

We now refer to Figure 2.5 that shows the running times in seconds for finding an optimal solution required by the model PMP$_{m1}$ de Melo da Silva et al. (2018) and by the proposed model IPS6 on all datasets. Cases in which an optimal solution is not found are assigned a running time of 3600 seconds. Points below the line indicate instances in which IPS6 is faster than PMP$_{m1}$, and points above the line indicate that PMP$_{m1}$ is faster than IPS6. It can be seen a clear dominance of IPS6 model on all the studied datasets. Across instances optimally solved by both models, IPS6 cuts the running times down on average by 68% on

Table 2.9: Performance of the model proposed by Lee and Hsu, the PMP$_{m1}$ model from de Melo da Silva et al. (2018), and IPS6 on the CV dataset, grouped by number of tiers (H) and number of stacks (S).

| | | | Lee and Hsu | | PMP$_{m1}$ | | IPS6 | |
|---|---|---|---|---|---|---|---|---|
| H | S | # | Opt. | CPU(s) | Opt. | CPU(s) | Opt. | CPU(s) |
| 5 | 3 | 40 | 32 | 559.81 | 40 | 63.78 | 40 | 85.65 |
| 5 | 4 | 40 | 19 | 846.67 | 38 | 236.45 | 38 | 125.71 |
| 5 | 5 | 40 | 5 | 1997.08 | 31 | 402.37 | 32 | 452.33 |
| 5 | 6 | 40 | 2 | 278.40 | 23 | 995.09 | 23 | 817.05 |
| 5 | 7 | 40 | 0 | - | 10 | 1075.44 | 11 | 960.26 |
| 5 | 8 | 40 | 0 | - | 4 | 2034.74 | 7 | 1194.23 |
| 6 | 4 | 40 | 1 | 517.81 | 6 | 530.92 | 9 | 1425.93 |
| 6 | 5 | 40 | 0 | - | 2 | 1681.84 | 2 | 1244.29 |
| 6 | 6 | 40 | 0 | - | 1 | 1827.20 | 1 | 444.99 |
| 6 | 7 | 40 | 0 | - | 0 | - | 0 | - |
| Total/Avg | | 400 | 59 | 763.74 | 155 | 478.49 | 163 | 467.24 |

The column "#" gives the total number of instances tested and "Opt." represent the number of optimal and feasible solutions obtained by each model. Column "CPU(s)" represents the average running time (in seconds) on the instances optimally solved by each model. The notation "-" is used when the averages cannot be calculated.

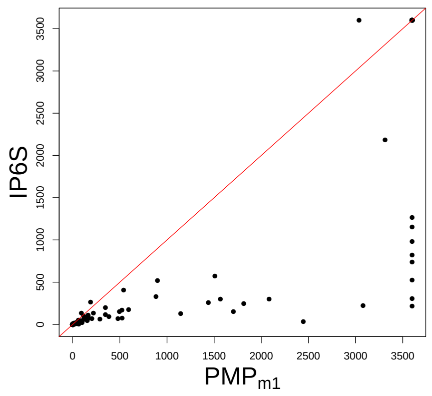
Table 2.10: Performance of the model proposed by Lee and Hsu, the PMP$_{m1}$ from de Melo da Silva et al. (2018), and IPS6 on the BZ dataset, grouped by number of tiers (H), stacks (S), and fill percentage (f).

| | | | | Lee and Hsu | | PMP$_{m1}$ | | IPS6 | |
|---|---|---|---|---|---|---|---|---|---|
| H | S | f | # | Opt. | CPU(s) | Opt. | CPU(s) | Opt. | CPU(s) |
| 4 | 3 | 0.50 | 60 | 60 | 0.12 | 60 | 0.02 | 60 | 0.02 |
| 4 | 3 | 0.70 | 60 | 60 | 8.06 | 60 | 0.88 | 60 | 0.91 |
| 4 | 5 | 0.50 | 60 | 60 | 0.82 | 60 | 0.08 | 60 | 0.07 |
| 4 | 5 | 0.70 | 60 | 58 | 103.74 | 60 | 1.97 | 60 | 1.53 |
| 4 | 7 | 0.50 | 60 | 60 | 10.92 | 60 | 0.21 | 60 | 0.17 |
| 4 | 7 | 0.70 | 60 | 53 | 288.35 | 60 | 5.91 | 60 | 3.73 |
| 4 | 9 | 0.50 | 60 | 59 | 107.54 | 60 | 1.29 | 60 | 1.07 |
| 4 | 9 | 0.70 | 60 | 40 | 108.08 | 60 | 94.92 | 60 | 22.03 |
| 6 | 3 | 0.50 | 60 | 60 | 127.78 | 60 | 2.62 | 60 | 3.00 |
| 6 | 3 | 0.70 | 60 | 47 | 301.43 | 59 | 82.33 | 60 | 102.96 |
| 6 | 5 | 0.50 | 60 | 49 | 134.79 | 60 | 11.62 | 60 | 18.59 |
| 6 | 5 | 0.70 | 60 | 22 | 640.10 | 44 | 466.39 | 44 | 264.09 |
| 6 | 7 | 0.50 | 60 | 43 | 271.25 | 59 | 123.31 | 59 | 101.49 |
| 6 | 7 | 0.70 | 60 | 11 | 182.38 | 35 | 404.02 | 37 | 234.19 |
| 6 | 9 | 0.50 | 60 | 31 | 321.56 | 59 | 156.63 | 60 | 118.95 |
| 6 | 9 | 0.70 | 60 | 4 | 336.68 | 32 | 659.36 | 33 | 385.65 |
| | | Total/Avg | 960 | 717 | 140.40 | 888 | 94.94 | 893 | 62.02 |

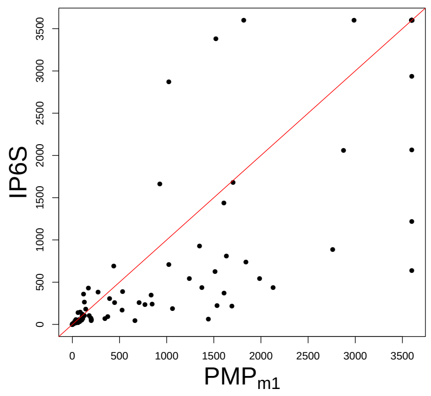
The column "#" gives the total number of instances tested and "Opt." represent the number of optimal and feasible solutions obtained by each model. Column "CPU(s)" represents the average running time (in seconds) on the instances optimally solved by each model.

EMM dataset (Figure 2.5a), by 38% on ZJY dataset (Figure 2.5b), by 26% on CV dataset (Figure 2.5c), and by 39% on BZ dataset (Figure 2.5d).

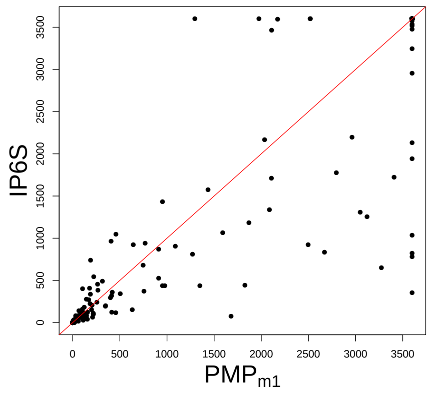
On the EMM dataset, IPS6 solves 5 more instances of the most challenging category with 10 stacks and fill percentage of 75%. Moreover, the average running times across the 34 instances optimally solved by PMP$_{m1}$ and IPS6 in this category are 580 and 154 seconds, respectively, which means an average reduction by 73%. On ZJY dataset, IPS6 also outperforms PMP$_{m1}$ in all the categories, cutting the average running time by 44% in the most difficult category of 6 stacks and 5 tiers. The trend continues on CV and BZ datasets, solving IPS6 more instances in the most challenging categories and reducing the average running time in almost all instances. There is just one category in which the average running time does not decrease with IPS6 model. This is the easiest category of the CV



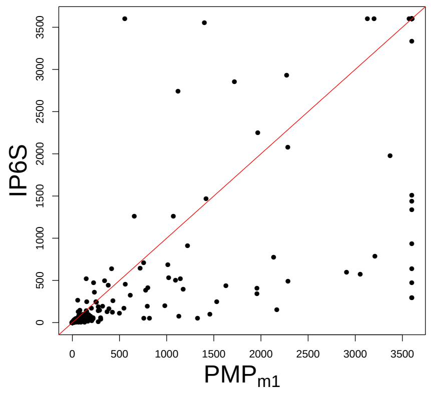
(a) EMM dataset



(b) ZJY dataset



(c) CV dataset



(d) BZ dataset

Figure 2.5: CPU times, in seconds, necessary for finding an optimal solution on all datasets with IPS6 and PMP$_{m1}$ by de Melo da Silva et al. (2018). Unsolved instances are assigned a CPU time of 3600 seconds.

dataset (5 tiers and 3 stacks) in which the average running time increases from 63 to 85 seconds. In the remainder categories, there is a high reduction of the running times.

## 2.8 Extensions of the model

The integer models developed in this study address the standard premarshalling problem. The objective is to minimize the number of moves required to order the bay and the constraints ensure that this rearrangement is made through a sequence of valid moves. However, other constraints could be included to produce solutions that are more useful in practice. This section outlines some of these possible extensions of the models.

### 2.8.1 Balancing the final configuration

The solution of the standard models could have an empty stack adjacent to a stack fully loaded up to tier $H$. This configuration is unstable and is avoided in practice. One way of preventing this type of solution could be to set a parameter $\kappa$ as the maximum difference in the number of containers of adjacent stacks. By adding constraints

$$\sum_{h=1}^{H}\sum_{p=1}^{P} x_{shpT} - \sum_{h=1}^{H}\sum_{p=1}^{P} x_{s+1hpT} \leq \kappa \qquad \forall s \in \{1,\ldots,S-1\} \tag{2.40}$$

$$\sum_{h=1}^{H}\sum_{p=1}^{P} x_{shpT} - \sum_{h=1}^{H}\sum_{p=1}^{P} x_{s+1hpT} \geq -\kappa \quad \forall s \in \{1,\ldots,S-1\} \tag{2.41}$$

we ensure that the difference between adjacent stacks does not exceed $\kappa$.

### 2.8.2 Avoiding empty or full stacks

Constraints (2.42) would prevent empty stack is in the final configuration, while constraints (2.43) would prevent full stacks.

$$\sum_{h=1}^{H}\sum_{p=1}^{P} x_{shpT} \geq 1 \quad \forall s \in \{1,\ldots,S\}; \tag{2.42}$$

$$\sum_{h=1}^{H}\sum_{p=1}^{P} x_{shpT} \leq H-1 \quad \forall s \in \{1,\ldots,S\}; \tag{2.43}$$

### 2.8.3 Favouring stable configurations

Among the solutions with the minimum number of moves, a more stable configuration would be preferred. We could define two new variables $h_{min}$ and $h_{max}$ to indicate the minimum and maximum number of containers in a stack at time $T$, adding constraints (2.44) and (2.45) to define the variables and adding the term $(h_{max} - h_{min})/(H+1)$ to the objective function. Note that $(h_{max} - h_{min})/(H+1) \leq H/(H+1) < 1$ and therefore the optimal solution of the modified model would always correspond to the minimum number of moves.

$$\sum_{h=1}^{H}\sum_{p=1}^{P} x_{shpT} \leq h_{max} \quad \forall s \in \{1,\ldots,S\} \tag{2.44}$$

$$\sum_{h=1}^{H}\sum_{p=1}^{P} x_{shpT} \geq h_{min} \quad \forall s \in \{1,\ldots,S-1\} \tag{2.45}$$

### 2.8.4 Favouring containers with the same priority in the same stack

It could be useful to have containers of the same priority one on top of the other in the same stack, making retrieving operations easier. That could be favoured by defining new variables $g_{shp}$ and subtracting the term $\sum_s \sum_h \sum_p g_{shp} / \sum_{p=1}^{P} n_p$ from the objective function.

$$g_{shp} = \begin{cases} 1, & \text{If a container of priority } p \text{ located at } (s,h) \text{ at time } T \\ & \text{is on top of other container of priority } p \\ 0, & \text{Otherwise} \end{cases}$$
$$\forall s \in \{1,\ldots,S\}; \quad \forall h \in \{2,\ldots,H\}; \quad \forall p \in \{1,\ldots,P\};$$

Since inequalities (2.46) are satisfied, the optimal solution would still produce the minimum number of moves.

$$\frac{\sum_s \sum_h \sum_p g_{shp}}{\sum_{p=1}^{P} n_p} \leq \frac{\sum_{p=1}^{P}(n_p - 1)}{\sum_{p=1}^{P} n_p} < 1 \tag{2.46}$$

In order to define adequately the new variables, we would also need to include the constraints:

$$g_{shp} \leq x_{shpT} \qquad \forall s \in \{1, \ldots, S\}; \quad \forall h \in \{2, \ldots, H\} \tag{2.47}$$

$$g_{shp} \leq 1 - (x_{shpT} - x_{sh-1pT}) \qquad \forall s \in \{1, \ldots, S\}; \quad \forall h \in \{2, \ldots, H\} \tag{2.48}$$

## 2.9 Concluding remarks

In this chapter we have studied the premarshalling problem, which plays an important role in the efficient management of storage yards, speeding up loading and unloading operations. This problem has been studied by developing integer formulations that can be easily implemented and adapted by researchers and practitioners. We have first proposed an initial model in which the moves are described using 3-indexed variables. As the linear relaxation was very weak, we developed new models using variables with 4, 5, and 6 indices. These models were conceptually better, strengthening the relationship between variables and improving the linear relaxations, but the dramatic increase in the number of variables offset their potential advantages. In a second phase, we decided to use different types of variables to describe the moves, separating the information about the origin of the move from that about its destination. In this case, models were improved with a small increase in the number of variables. We have proposed a total of eight integer linear models and we have also designed an iterative procedure to solve them, avoiding the need of an upper bound to build the model. These eight models have been extensively tested and results show the superior performance of the increasingly complex split-variable models. We have finally tested our best proposal, IPS6, and the previously published models by Lee and Hsu (2007) and de Melo da Silva et al. (2018) over an extended computational analysis that shows the better performance of our model on several well-known datasets.

# Chapter 3

# A branch-and-bound approach for large premarshalling problems

## 3.1 Introduction

The previous chapter addressed the CPMP by developing integer programming models that are more flexible and easier to implement than sophisticated exact methods but are not able to solve real world-sized problems. Despite recent advances in solving large-sized problems (Tierney et al., 2017; Tanaka and Tierney, 2018), several classes of premarshalling problems remain difficult for exact methods. We therefore propose a new branch-and-bound-based approach with the following novel components:

1. an extension of the IBF$^1$ bound from Tanaka and Tierney (2018),
2. two new lower bounds to complement IBF$^1$,
3. one new dominance rule and generalizations of the rules from Tanaka and Tierney (2018),
4. a memoization-based heuristic for finding feasible solutions.

We evaluate our approach on standard premarshalling benchmarks, and on a new dataset to guard against overfitting our algorithm to the available instances. An extended computational study has been carried out, and it shows that our algorithm performs better than the state-of-the-art approaches. Our algorithm dramatically reduces the search space, finding 8% more optimal solutions and 6% more feasible solutions in the tested instances. Remarkably, it is able to optimally solve the vast majority of instances tested with 7 tiers and 9-10 stacks, which are considered difficult real-world CPMP instances.

The new terminology and notation used in this chapter are introduced in Section 3.2. Section 3.3 presents the iterative deepening branch-and-bound algorithm, including new dominance rules. We formalize the new lower bounds in Section 3.4 and provide computational results in Section 3.5. Finally, some conclusions are drawn in Section 3.6.

## 3.2 Notation

In this chapter, we refer to the definitions from Bortfeldt and Forster (2012) for the different types of container moves. A *bad-good* move, BG move, is a move in which a misoverlaying container is moved to a stack where it is no longer misoverlaying. Similarly, a *bad-bad* move, BB move, involves a container being transferred from a misoverlaying position to a stack where it is still misoverlaying. We can thus define *good-bad* and *good-good* moves, i.e., GB and GG moves, in a similar way. Furthermore, BX moves refer to moves from a misoverlaid stack, with containers that are blocked by others, to any other stack and GX moves refer to moves from a non-misoverlaid stack, without containers that are blocked by others, to any other stack.

We use the notation from the previous chapter to describe the set of containers placed in the bay, $\mathcal{C}$, the set of stacks and tiers of the bay, $\mathcal{S}$ and $\mathcal{H}$, and the set of priorities, $\mathcal{P}$. The non-negative integer-valued function $group(s,h)$ gives the priority of the container in stack $s$ at tier $h$, i.e., it indicates the order in which the container in that slot must be retrieved. It equals 0 if no container is located at position $(s,h)$. Therefore, a bay has no misoverlays if $group(s,h) \geq group(s,h+1)$ for all $s \in S, h \in \mathcal{H} \setminus \{H\}$. We described a solution in Chapter 2 as a sequence of moves of the form $(s,k) : s \neq k \in \{1,\ldots,S\}$ where $s$ is the origin stack to which the topmost container to be moved belongs and $k$ is the destination stack where the container will also be placed at the top. We now describe the moves identifying also the container that is moving $c$, so we use triples $(c,s,k)$ instead of tuples $(s,k)$. Hence, a solution of $n$ moves is represented as $sol = (c_1,s_1,k_1)\ldots(c_i,s_i,k_i)\ldots(c_n,s_n,k_n)$, where container $c_i$ is the container moved from stack $s_i$ to stack $k_i$ in move $i$. Table 3.1 summarizes all the notation used throughout the chapter. It includes notation already presented and adds several new concepts.

Table 3.1: Main notation employed throughout the chapter.

| | |
|---|---|
| $\mathcal{S}$ | Set of stacks. $S := \|\mathcal{S}\|$ is the total number of stacks |
| $\mathcal{H}$ | Set of tiers. $H := \|\mathcal{H}\|$ represents the highest tier of the stacks. |
| $\mathcal{P}$ | Set of priorities or group values. |
| $\mathcal{C}$ | Set of containers. |
| $S^{\mathrm{M}}$ | Set of misoverlaid stacks. |
| $S^{\mathrm{minM}}$ | Set of misoverlaid stacks with the minimum number of misoverlaying containers. $S^{\mathrm{minM}} := \{s \in S \mid n_s^{\mathrm{M}} = f^{\mathrm{M}}\}$. |
| $S^{\mathrm{N}}$ | Set of non-misoverlaid stacks. |
| $U$ | Set of stacks where the misoverlaying containers are "upside down" sorted, i.e., $group(s,h) \leq group(s,h+1)$ for all misoverlaying tiers $h$. $U := \{s \in S^{\mathrm{M}} \mid n_s^{\mathrm{BG}} = 1\}$ |
| $U'$ | Set of stacks that would be upside down if one misoverlaying container was removed. We assume $U \subset U'$. |
| $n_s$ | Number of containers in stack $s$. |
| $n_s^{\mathrm{M}}$ | Number of misoverlaying containers in stack $s$. |
| $n_s^{\mathrm{N}}$ | Number of non-misoverlaying containers in stack $s$. |
| $n^{\mathrm{M}}$ | Total number of misoverlaying containers in the bay. |
| $f^{\mathrm{M}}$ | Minimum number of misoverlaying containers in misoverlaid stacks. ($f^{\mathrm{M}} := \min_{i \in S^{\mathrm{M}}} n_s^{\mathrm{M}}$). |
| $p_s^{\mathrm{top}}$ | Group value of the topmost container in stack $s$. If stack $s$ is empty, $p_s^{\mathrm{top}} := \|\mathcal{P}\| = P$. |
| $c_s^{\mathrm{top}}$ | Topmost container in stack $s$. If stack $s$ is empty, $c_s^{\mathrm{top}} = \varnothing$. |

| | |
|---|---|
| $m_s$ | Largest group value of the misoverlaying containers in stack $s \in S^{\mathrm{M}}$. If $s \in U$, $m_s = p_s^{\mathrm{top}}$. |
| $m_s^i$ | The group value of the $i$th largest misoverlaying group value in stack $s$, e.g., $m_s^1 = m_s$, $m_s^2$ is the second largest misoverlaying group value, etc. |
| $w_s$ | Smallest group value of non-misoverlaying containers in stack $s$. If stack $s$ is empty, $w_s := \|\mathcal{P}\| = P$. If $s \in S^{\mathrm{N}}$, $w_s = p_s^{\mathrm{top}}$. |
| $w_s^{\mathrm{sec}}$ | Second smallest group value of non-misoverlaying containers in stack $s$. If all the non-misoverlaying containers in stack $i$ have the same group, $w_s^{\mathrm{sec}} := \|\mathcal{P}\| = P$. |
| $n_s^{\mathrm{BG}}$ | Minimum number of non-misoverlaid stacks necessary for repairing stack $s$ with only BG moves. |
| $p_{si}^{\mathrm{BG}}$ | Minimum group value of the topmost container of the $i$th non-misoverlaid stack for repairing stack $s$. Sorted as $p_{s1}^{\mathrm{BG}} \geq p_{s2}^{\mathrm{BG}} \geq \cdots \geq p_{sn_s^{\mathrm{G}}}^{\mathrm{BG}}$. We assume $p_{si}^{\mathrm{BG}} = 0$ for $i > n_s^{\mathrm{BG}}$. |

Minimum and maximum values are computed over the set of stacks $U$. However, sometimes $U$ is empty and we define $\min_{i \in U}$ ($\max_{i \in U}$) to be $\infty$ ($-\infty$), respectively. To determine the highest and second highest group values of misoverlaid stacks we do the following. First, we sort the containers from highest to lowest group value and then take the first and second containers from this list to be the highest and second highest (lowest), respectively. This means $m_s$ and $m_s^2$ could contain the same group value if multiple containers have the same value.

## 3.3 Improved iterative deepening branch and bound

An iterative deepening branch and bound for the CPMP is proposed in Tanaka and Tierney (2018). We extend this approach by adding tighter lower bounds, a new branching comparison algorithm, new dominance rules, as well as an initial memoization-based heuristic. In this section, we briefly describe the iterative deepening branch and bound approach and the heuristics used during the search. We next present our branching comparison algorithm, we formalize the new dominance rules used in our approach, and, finally, we provide a memoization-based initial upper bound heuristic.

### 3.3.1 Algorithm overview

Algorithm 1 shows the pseudocode of our approach. Given an initial layout, *bay*, the algorithm calculates its lower bound, *lb*, and tries to obtain a feasible solution, $\pi$, by using the greedy heuristic presented in Tanaka and Tierney (2018). If a feasible solution is obtained, $u$ denotes its number of moves. In a nutshell, this heuristic attempts to make a solution feasible using only BG and GG moves and stops when this is no longer possible. Although it is not guaranteed to find a solution, it is extremely quick. When a solution is not found, we apply the memoization-based algorithm presented later in Section 3.3.4 before starting the search.

The iterative deepening method uses a depth-first search (DFS) in which the search depth is limited by a parameter $l$. The complete tree is searched up to level $l$, starting at $lb$ and ending at $u$. The optimality of a solution is guaranteed when $l$ equals $u$. The function BB-Recur carries out the DFS up to level $l$. At each node, all possible non-dominated moves are performed and the branches are ordered according to our branching comparison algorithm.

**Algorithm 1** Iterative deepening branch and bound algorithm pseudocode.

```
 1: function CPMP-IDBB($bay$)
 2:     $lb \leftarrow$ LB($bay$)                                  ▷ Depth lower limit
 3:     $u, \pi \leftarrow$ Greedy_Heuristic($bay$)      ▷ Feasible solution, depth upper limit
 4:     if $\pi = \varnothing$ then
 5:         $u, \pi \leftarrow$ Memoization_Heuristic($bay$)
 6:     $l \leftarrow lb$
 7:     $M \leftarrow \varnothing$                       ▷ Feasible solution to be found in the search
 8:     while $l < u$ do
 9:         $M \leftarrow$ BB-Recur($M, bay, l, u$)
10:         if $M \neq \varnothing$ then return $M$
11:         $l \leftarrow l + 1$
12:     return $\pi$
```

A move is dominated when one of the dominance rules is satisfied or the lower bound of the layout is larger than the current depth limit. We try to complete partial solutions by using the greedy heuristic algorithm. In this way, the value $u$ is updated during the search every time a better solution is found. The search ends if a solution of value $l$ is reached.

### 3.3.2 Branching comparison algorithm

The order in which the branches are explored is irrelevant for the correctness of the search. Nevertheless, a good branching strategy is critical for obtaining a solution quickly. We consider seven different tie-breaking criteria to order the branches, and they are evaluated in order until the tie is broken.

1. The branch with the lowest lower bound.
2. The largest difference $d$, given by equation (3.1):

$$d = \left( \sum_{s \in S^{\mathrm{N}}} p_s^{\mathrm{top}} \cdot (H - n_s) \right) - \left( \sum_{s \in S^{\mathrm{M}}} \sum_{1 \leq i \leq n_s^{\mathrm{M}}} m_s^i \right) \tag{3.1}$$

   On the one hand, the larger the group value of the topmost container of a non-misoverlaid stack is, the wider the range of containers that can be well-placed (containers with a group value less than or equal to that of the topmost container) above that stack. On the other hand, the lower the group value of a misoverlaying container is, the easier it will be to move it to a good position.
3. The lowest sum of the group values of the misoverlaying containers.
4. The largest number of upside down containers that can be well-placed using only BG moves.
5. The largest difference between initial and final gaps. The initial gap of a move is the difference between the group value of the topmost container of the origin stack after the move and the group value of the container being moved. The final gap is the difference between the group value of the container being moved and the group value of the top container on the destination stack before the move.
6. The smallest final gap.
7. The largest group value of the container being moved.

### 3.3.3 New dominance rules

Dominance rules for (*i*) unrelated, (*ii*) transitive, (*iii*) same group symmetry move avoidance, and (*iv*) empty stacks rules are introduced in Tanaka and Tierney (2018). We use rule (*iv*) and propose extensions of the rules (*i*) to (*iii*) by introducing the concept of an *invariant stack*. We also present a new rule that uses the current upper and lower bounds of a node. In this section, $n_s$ refers to the number of containers in stack $s$ at the end of a given move sequence $(c_1, s_1, k_1)(c_2, s_2, k_2)\dots(c_{n-1}, s_{n-1}, k_{n-1})(c_n, s_n, k_n)$ and $\delta$ is the Kronecker delta: $\delta_{x,y} = 0$ if $x \neq y$ and $\delta_{x,y} = 1$ if $x = y$.

**Definition 1.** *A stack $s$ is invariant to a sequence of moves $(c_1, s_1, k_1)\dots(c_n, s_n, k_n)$, i.e, from move 1 to n, if its layout before and after the sequence of moves is the same and the topmost container of stack s in the layout before the sequence is not moved by the sequence ($c_s^{\text{top}} \notin \{c_1, \dots, c_n\}$).*

Given stack $s$ being invariant to $(c_1, s_1, k_1)\dots(c_n, s_n, k_n)$, the maximum number of temporary containers that are simultaneously assigned to $s$ during the sequence involving from move 1 to $n$ is:

$$t_s^{1,n} := \max_{j \in \{1,\dots,n\}} \left\{ \sum_{i=1}^{j} (\delta_{s,k_i} - \delta_{s,s_i}) \right\}$$

Figure 3.1 shows a sequence of moves $(4, 3, 1)(4, 3, 1)(1, 3, 2)(4, 1, 3)(4, 1, 3)$ for which stack 1 is invariant. It can be seen that $t_1^{1,5} = 2$.

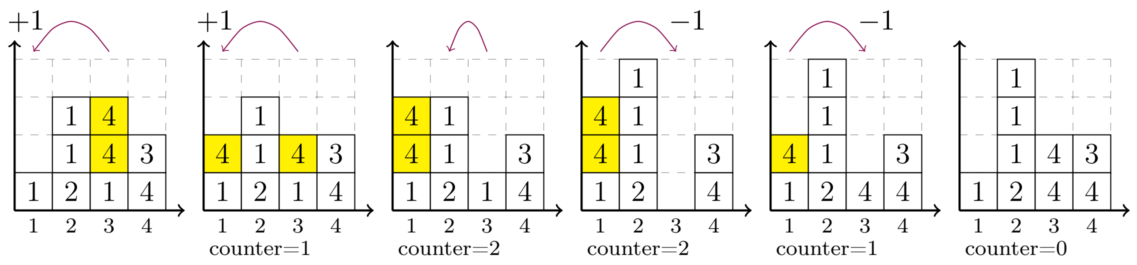


Figure 3.1: Example in which stack 1 is invariant to the sequence $(4, 3, 1)(4, 3, 1)(1, 3, 2)(4, 1, 3)(4, 1, 3)$.

#### Unrelated move symmetries

We consider the cases in which a move can be indistinctly placed before and after a sequence even though its source or destination stack has been temporarily used in that sequence. We return to the example in Figure 3.1 and suppose that a container with group value 1 is moved from stack 1 to stack 4 after the sequence. This move could be also done before the sequence, leading to the same final configuration; so one of the moves could be disallowed.

**Proposition 2.** *A sequence $(c_1, s_1, k_1), (c_2, s_2, k_2), \dots, (c_{n-1}, s_{n-1}, k_{n-1}), (c_n, s_n, k_n)$ is disallowed if all of the following conditions are satisfied:*

1. *$s_1 > s_n$*
2. *$s_n$ and $k_n$ are invariant to the sequence $(c_1, s_1, k_1), (c_2, s_2, k_2), \dots, (c_{n-1}, s_{n-1}, k_{n-1})$.*

*3.* $t^{1,n-1}_{k_n} + n_{k_n} \leq H$

*Proof.* The first condition $s_1 > s_n$ is just for tie-breaking. Since $s_n$ and $k_n$ are invariant to the sequence $(c_1, s_1, k_1), \dots, (c_{n-1}, s_{n-1}, k_{n-1})$, the same layout as the original sequence is obtained by the sequence $(c_n, s_n, k_n), (c_1, s_1, k_1), \dots, (c_{n-1}, s_{n-1}, k_{n-1})$ whenever it is feasible. It is only feasible when the third condition is satisfied, ensuring $k_n$ to have enough space to allocate the maximum number of temporary containers assigned to it during the sequence $(c_1, s_1, k_1), \dots, (c_{n-1}, s_{n-1}, k_{n-1})$. With respect to $s_n$, temporary containers always fit into it. Since $t^{1,n-1}_{s_n} + n_{s_n} + 1 \leq H$, therefore $t^{1,n-1}_{s_n} + n_{s_n} \leq H$. □

Proposition 2 is reduced to the case $s_1, k_1 \notin \{s_2, k_2, \dots, s_n, k_n\}$ in Tanaka and Tierney (2018). Note that condition $s \notin \{s_2, k_2, \dots, s_n, k_n\}$ is a particular case of $s$ invariant to $(c_2, s_2, k_2), \dots, (c_{n-1},\ s_{n-1}, k_{n-1}), (c_n, s_n, k_n)$, which satisfies $\sum_{i=2}^{n} \delta_{s_i,s} = 0$.

**Transitive move avoidance**

This kind of moves considers the relocation of a single container twice, once from one stack $a$ to another stack $x$, and other from that stack $x$ to another stack $b$. In some cases, the container can be moved directly from $a$ to $b$. In other cases, there are several options for stack $x$ and all but one of them can be disallowed. We again consider the sequence of moves in Figure 3.1 to illustrate the first case. Suppose that the container of group value 1 is moved from stack 1 to 4 before the sequence $(4,3,1)(4,3,1)(1,3,2)(4,1,3)(4,1,3)$ and it is moved from stack 4 to 3 at the end. Instead of these two moves, we could make just one move from stack 1 to 3 at the end of the sequence.

**Proposition 3.** *A sequence of moves $(c_1, s_1, k_1), (c_2, s_2, k_2), \dots, (c_{n-1}, s_{n-1}, k_{n-1}), (c_1, k_1, k_n)$ is disallowed if $c_1 \notin \{c_2, \dots c_{n-1}\}$ and one of the following conditions is satisfied:*

1. *Stack $s_1$ is invariant to the sequence $(c_2, s_2, k_2), \dots, (c_{n-1}, s_{n-1}, k_{n-1})$ and $t^{2,n-1}_{s_1} + n_{s_1} + 1 \leq H$.*
2. *Stack $k_n$ is invariant to the sequence $(c_2, s_2, k_2), \dots, (c_{n-1}, s_{n-1}, k_{n-1})$ and $t^{2,n-1}_{k_n} + n_{k_n} \leq H$.*
3. *There exists a stack $s' < k_1$ invariant to the sequence $(c_2, s_2, k_2), \dots, (c_{n-1}, s_{n-1}, k_{n-1})$ that satisfies the condition $t^{2,n-1}_{s'} + n_{s'} + 1 \leq H$.*

*Proof.* 1. Since stack $s_1$ is invariant, the number of containers in $s_1$ before the second move is the same as $n_{s_1}$. The second term ensures that $s_1$ has enough space to move the container $c_1$ after the sequence $(c_2, s_2, k_2), ..., (c_{n-1}, s_{n-1}, k_{n-1})$. Therefore, the same layout as that of the original sequence is obtained by another feasible sequence with one less move $(c_2, s_2, k_2), ..., (c_{n-1}, s_{n-1},\ k_{n-1}), (c_1, s_1, k_n)$ when $s_1 \neq k_n$ or by another feasible sequence with two less moves $(c_2, s_2, k_2), \dots,\ (c_{n-1}, s_{n-1}, k_{n-1})$ when $s_1 = k_n$.

2. Since stack $k_n$ is invariant to the sequence $(c_2, s_2, k_2), ..., (c_{n-1}, s_{n-1}, k_{n-1})$ and $n_{k_n}$ is the number of containers stored in $k_n$ after the move $(c_1, s_1, k_n)$, $k_n$ has $n_{k_n} - 1$ containers stored before and after that sequence. By the second term, $k_n$ has enough space to store the maximum number of containers that are temporarily allocated to it during the sequence $(c_2, s_2, k_2), ..., (c_{n-1}, s_{n-1}, k_{n-1})$, although it now has one less

free slot. The same layout as that of the original sequence is obtained by the feasible sequence $(c_1, s_1, k_n), (c_2, s_2, k_2), ..., (c_{n-1}, s_{n-1}, k_{n-1})$ when $s_1 \neq k_n$ or by the sequence $(c_2, s_2, k_2), ..., (c_{n-1}, s_{n-1}, k_{n-1})$ when $s_1 = k_n$.

3. The sequence of moves $(c_1, s_1, s'), (c_2, s_2, k_2), ..., (c_{n-1}, s_{n-1}, k_{n-1}), (c_1, s', k_n)$ is feasible since $c_1 \notin \{c_2, \dots c_n\}$ because $s'$ is invariant to $(c_2, s_2, k_2), ..., (c_{n-1}, s_{n-1}, k_{n-1})$, $s'$ has enough space to store the temporary moves of containers, and it provides the same layout as the original sequence. Condition $s' < k_1$ is just for tie-breaking. When $s' = k_n$, the sequence $(c_1, s_1, k_n), (c_2, s_2, k_2), ..., (c_{n-1}, s_{n-1}, k_{n-1})$ is feasible.

□

### Same group symmetries

Symmetries appear when two containers of the same group value are relocated in two different moves. We identify some cases in which an equivalent sequence is obtained by changing the source stack or the destination stack. Figure 3.2 exemplifies this kind of symmetry. The upper row shows a sequence in which two different containers with the same group value 1 are moved. The bottom row shows a sequence that leads to the same final layout but just with one move.

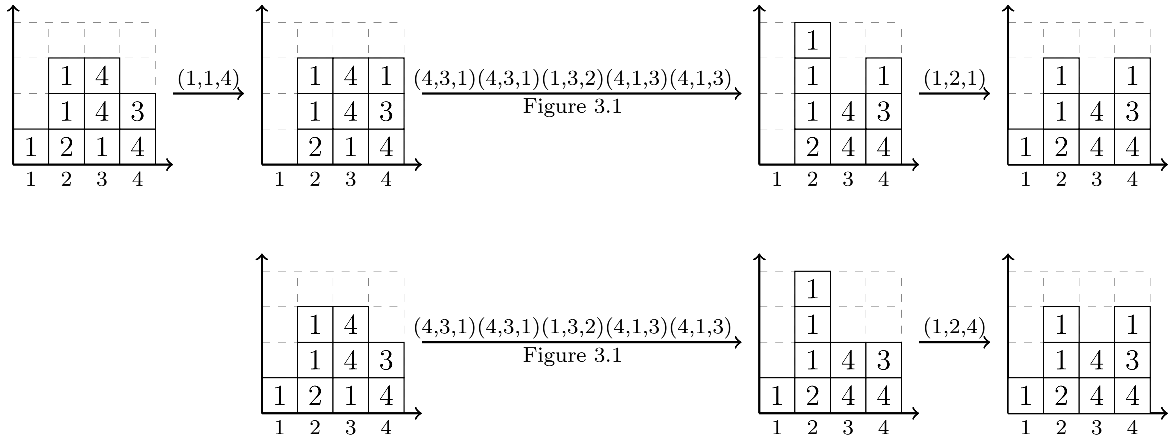


Figure 3.2: Same group symmetries avoidance example.

**Proposition 4.** *A sequence of moves* $(c_1, s_1, k_1), (c_2, s_2, k_2), \dots, (c_{n-1}, s_{n-1}, k_{n-1}), (c_n, s_n, k_n)$ *is disallowed if group*$(c_1)$ = *group*$(c_n)$ *and one of the following conditions is satisfied:*

1. $s_1 = k_n$, $t_{s_1}^{2,n-1} + n_{s_1} \leq H$, *and* $\{s_1, s_n\}$ *or* $\{s_1, k_1\}$ *are invariant from move* 2 *to* $(n-1)$.
2. $s_1 > s_n$, $t_{s_1}^{2,n-1} + n_{s_1} \leq H$, $s_1$ *and* $s_n$ *are invariant from move* 2 *to* $(n-1)$.
3. $k_1 > k_n$, $t_{k_n}^{2,n-1} + n_{k_n} \leq H$, $k_1$ *and* $k_n$ *are invariant from move* 2 *to* $(n-1)$.

*Proof.* Let $p$ = *group*$(c_1)$ = *group*$(c_n)$. It suffices to show that the same layout with respect to group values is obtained by another feasible sequence. It does not matter if the positions of containers $c_1$ and $c_n$ of the same group may be interchanged.

1. In the sequence, the container $c_1$ of group value $p$ is moved from $s_1$ first and finally $c_n$ (also of group value $p$) is moved to $k_n = s_1$. Since $s_1$ is invariant to $(c_2, s_2, k_2), \dots, (c_{n-1}, s_{n-1}, k_{n-1})$, the number of containers in $s_1$ in the initial layout is equal to that of the final layout, $n_{s_1}$. First, assume that $s_n$ is invariant and $s_n \neq k_1$, which implies $c_n \notin$

$\{c_1, \ldots, c_{n-1}\}$. Noting that $c_n$ is on top of $s_n$ in the initial layout, we consider a new sequence $(c_n, s_n, k_1), (c'_2, s_2, k_2), \ldots, (c'_{n-1}, s_{n-1}, k_{n-1})$, where $c'_k = c_n$ if $c_k = c_1$, and $c'_k = c_k$ otherwise, to move $c_n$ in place of $c_1$ by the second and later moves. In this sequence, $c_1$ is not moved from $s_1$, which may block moves from $s_1$ and/or cause lack of spare space for moves to $s_1$. However, the former is not the case because $s_1$ is invariant, and the latter never occurs from $t^{2,n-1}_{s_1} + n_{s_1} \leq H$. Therefore, this sequence is feasible and yields the same layout as the original sequence except for the interchanged positions of $c_1$ and $c_n$. Next, assume that $k_1$ is invariant and $k_1 \neq s_n$. It implies that $c_1$, which is moved to $k_1$ by $(c_1, s_1, k_1)$, is not moved by $(c_2, s_2, k_2), \ldots, (c_n, s_n, k_n)$ and thus $c_1 \notin \{c_2, \ldots, c_n\}$ holds. This condition together with the above argument on $s_1$ ensures the feasibility of a new sequence $(c_2, s_2, k_2), \ldots, (c_{n-1}, s_{n-1}, k_{n-1}), (c_n, s_n, k_1)$. It is not difficult to verify that it yields the same layout as the original sequence with respect to group values. Finally, assume that $s_n = k_1$ is invariant, therefore $c_1 = c_n$ and $c_1 \notin \{c_2, \ldots, c_{n-1}\}$ holds, thus $k_1$ remains the same at the end of the whole sequence. The same layout is obtained by the sequence $(c_2, s_2, k_2), \ldots, (c_{n-1}, s_{n-1}, k_{n-1})$ that is feasible because of the above argument on $s_1$.

2. Consider a sequence of moves $(c_n, s_n, k_1), (c'_2, s_2, k_2), \ldots, (c'_{n-1}, s_{n-1}, k_{n-1})$ where $c'_k = c_n$ if $c_k = c_1$ and $c'_k = c_k$ otherwise. As already shown in case 1, the sequence is feasible since $s_1$ and $s_n$ are invariant and $t^{2,n-1}_{s_1} + n_{s_1} \leq H$ holds. The topmost container of $s_1$ after this sequence is $c_1$ as $s_1$ is invariant, so that the sequence $(c_n, s_n, k_1), (c'_2, s_2, k_2), \ldots, (c'_{n-1}, s_{n-1}, k_{n-1}), (c_n, s_1, k_n)$ is also feasible. Obviously, it yields the same layout as the original sequence with respect to the group values.
3. We consider a sequence of moves $(c_1, s_1, k_n), (c_2, s_2, k_2), \ldots, (c_{n-1}, s_{n-1}, k_{n-1})$. Since $k_1$ is invariant, $c_1 \notin \{c_2, \ldots, c_{n-1}\}$ holds. Stack $k_n$ is also invariant and $t^{2,n-1}_{k_n} + n_{k_n} < H$ holds. These facts ensure the feasibility of the sequence. It yields the same layout as the original sequence with regard to group values since $k_1$ and $k_n$ are invariant.

□

**BB and GB move avoidance**

Let LB$^{\text{BF}}$ be the lower bound proposed by Bortfeldt and Forster (2012), $l$ be the current depth in the algorithm, and $\textsc{Moves}(x)$ be the number of moves to reach node $x$. We only permit BG and GG moves when $l = \textsc{Moves}(x) + \text{LB}^{\text{BF}}$ and at least one stack is non-misoverlaid. This rule is validated by the fact that LB$^{\text{BF}}$ assumes only BG and GG moves. It is strengthened by only allowing BG moves if $l = \textsc{Moves}(x) + n^{\text{M}}$ (or BX moves if $l = \textsc{Moves}(x) + n^{\text{M}} + f^{\text{M}}$ and all stacks in $x$ are misloverlaid).

We also implement a tie-breaking mechanism within this rule. If a container can be moved by a BG move to either one of two non-misoverlaid stacks with the same number of containers, we choose the one with the smaller topmost group value. Further ties are broken by the stack number. Since this dominance rule conflicts with (3) of Proposition 4, we modify (3) so that it is applied only when the numbers of containers in stacks $k_1$ and $k_n$ are different, at least one of the stacks $k_1$ and $k_n$ is misoverlaid, or the group value of the topmost non-misoverlaying container of stack $k_1$ is larger than or equal to that of stack $k_n$.

### 3.3.4 Memoization-based upper bound heuristic

In the definition of the CPMP, stacks are essentially interchangeable, i.e., given a layout of the bay, only the order in which containers are stacked vertically matters, and not the order of the stacks in the bay. For each arrangement of the bay, there can be up to $S!$ equivalent layouts. An example of equivalent layouts is shown in Figure 3.3. The idea of rearranging the bay to create an abstract configuration is proposed by Ku and Arthanari (2016) for the container relocation problem and also used in Quispe et al. (2018) for the same problem. We build on this idea to create an efficient and simple upper bound heuristic.

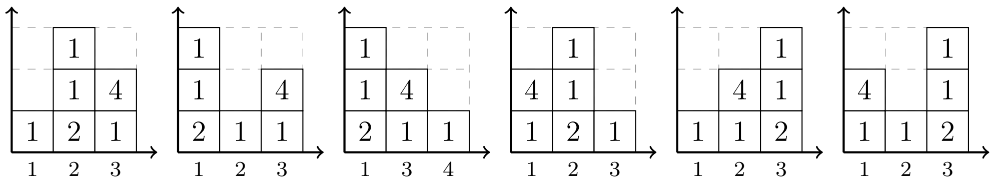


Figure 3.3: Equivalent layouts of the same bay arrangement.

We design a hash function that allows us to detect layouts corresponding to abstract configurations already visited during the search. This procedure allows to detect large number of equivalent layouts without actually having to store them all. First, nonempty stacks are ordered by non-increasing number of containers. In case of a tie, they are examined from top to bottom until a container of a different group value is found at the same tier. In this case, the one with the higher group value is ordered first. If two stacks have the same layout, it does not matter which one is ordered first. The key concatenates the groups of the containers of each stack in the ordered list from bottom to top. We again refer to Figure 3.3, this process provides the same key for the 6 layouts shown.

Since the use of the memoization strategy during the complete iterative deepening has a very high computational cost, we only apply it to the initial configuration in the cases in which the greedy heuristic does not find a feasible solution. The algorithm starts at the initial configuration, saving it in the hash table. It then creates all of the initial solution's children, pruning those already saved in the hash table and any nodes removed by our dominance rules. Just the best node according to the branching comparison algorithm is saved and the process is repeated from this node until a feasible solution is found or a maximum fixed level is reached.

## 3.4 Extended lower bounds

In this section we describe our new bounds. First, we introduce $\text{LB}^{\text{BF}}$ proposed by Bortfeldt and Forster (2012), on which all the other bounds are based, and $\text{IBF}^1$ by Tanaka and Tierney (2018), which we also extend. Finally, we present our proposed lower bounds $\text{IBF}^2$, $\text{IBF}^3$, and $\text{IBF}^4$.

### 3.4.1 The Bortfeldt and Forster lower bound

The $\mathrm{LB}^{\mathrm{BF}}$ from Bortfeldt and Forster (2012) is formalized as $\mathrm{LB}^{\mathrm{BF}} := n_{\mathrm{BX}} + n_{\mathrm{GX}}$ where

$$n_{\mathrm{BX}} := \begin{cases} n^{\mathrm{M}} + f^{\mathrm{M}} & \text{if all stacks are misoverlaid,} \\ n^{\mathrm{M}} & \text{otherwise,} \end{cases}$$

and $n_{\mathrm{GX}}$ puts a lower bound on the number of non-misoverlaying containers that must be moved. First, a slot $(s,h)$ is a *potential supply slot of group value* $p$ if $h > n_s^{\mathrm{N}} > 0$ and $w_s = p$. Let $s^{\mathrm{P}}(p)$ be the number of potential supply slots of group value $p$, and $S^{\mathrm{P}}(p) = \sum_{\substack{p' \geq p \\ p' \in \mathcal{P}}} s^{\mathrm{P}}(p') + H \cdot n_{\mathrm{E}}$ be the cumulative potential supply of group value $p$, where $n_{\mathrm{E}}$ is the number of empty stacks. The *demand* of group value $p$ is given by $d(p)$, which is the number of misoverlaying containers of group $p$. Furthermore, $D(p) = \sum_{\substack{p' \geq p \\ p' \in \mathcal{P}}} d(p')$ is the *cumulative demand* of $p$. The *cumulative demand surplus* for group value $p$ is given by $D^{\mathrm{S}}(p) = D(p) - S^{\mathrm{P}}(p)$.

The item group $p^* = \operatorname{argmax}_p D^{\mathrm{S}}(p)$ is the item group with the largest cumulative demand surplus. All stacks $s$ satisfying $w_s < p^*$ are called *potential GX* stacks, and there are $n_{\mathrm{GX}}^{\mathrm{PS}}$ of them. Assume $n_{\mathrm{GX}}^{\mathrm{PS}} \geq n_{\mathrm{GX}}^{\mathrm{S}} = \max(0, \lceil D^{\mathrm{S}}(p^*)/H \rceil$. The stacks are now sorted such that potential GX stacks come first and are then sorted according to the value $n^{p^*}(s)$ ascending, where $n^p(s)$ is the number of non-misoverlaying containers with a group $p' < p$ in stack $s$. Finally, we define $n_{\mathrm{GX}} := \sum_{s=1}^{n_{\mathrm{GX}}^{\mathrm{S}}} n^{p^*}(s)$ and get a lower bound on the number of GX moves. We refer to Bortfeldt and Forster (2012), for a proof of correctness of the bound.

### 3.4.2 $\mathrm{IBF}^1$ lower bound

Tanaka and Tierney (2018) introduce the lower bounds $\mathrm{IBF}^0$ and $\mathrm{IBF}^1$. When non-full stacks are all misoverlaid, $\mathrm{IBF}^0$ increases $\mathrm{LB}^{\mathrm{BF}}$ by the number of misoverlaying containers in the stack(s) with the smallest number of this kind of container. Moreover, $\mathrm{IBF}^1$ increases $\mathrm{IBF}^0$ by 1 in the following cases.

**Case 1:** All stacks are misoverlaid.

**Case 2:** One stack is not misoverlaid and it is not full (2a), or it is full (2b).

**Case 3:** Two stacks are not misoverlaid and at least one is not full.

**Case 4:** $n_{\mathrm{GX}} = 0$ and only one non-misoverlaid stack is not full.

### 3.4.3 Extended lower bound $\mathrm{IBF}^2$

We prove that $\mathrm{IBF}^0$ can be improved not by 1, but by 2 under some conditions in Cases 1, 2b, 3, and 4. This lower bound is denoted by $\mathrm{IBF}^2$.

#### Extension of $\mathrm{IBF}^1$, case 1: All stacks are misoverlaid

The bound $\mathrm{IBF}^1$ improves $\mathrm{IBF}^0$ by 1 through consideration of the moves required to make two stacks non-misoverlaid. Let $\mathrm{IBF}^1 := \mathrm{IBF}^0 + 1$ if it is impossible to repair a stack with the minimum number of misoverlaying containers ($f^{\mathrm{M}}$) only using BB moves and then another stack only using BG moves. We now show that in certain situations we can further improve $\text{IBF}^1$ by 1. This is done by checking whether the second stack can be repaired under the assumption that a single additional GB or BB move is permitted.

**Proposition 5.** $\text{IBF}^2 := \text{IBF}^1 + 1$ *is a valid lower bound for the CPMP if all of the following conditions are true:*

1. *All stacks are misoverlaid*
2. $\text{IBF}^1 := \text{IBF}^0 + 1$
3. $\min_{s'_1 \in U} m_{s'_1} > \max_{s_1 \in \{s \in S \mid n^{\text{M}}_s = f^{\text{M}}+1\}} w_{s_1}$
4. $\min_{s'_1 \in U} m_{s'_1} > \max_{s_2 \in S^{\text{minM}}} w^{\text{sec}}_{s_2}$
5. $\min_{s'_2 \in U'} p^{\text{X}}_{s'_2} > \max_{s_2 \in S^{\text{minM}}} w_{s_2}$

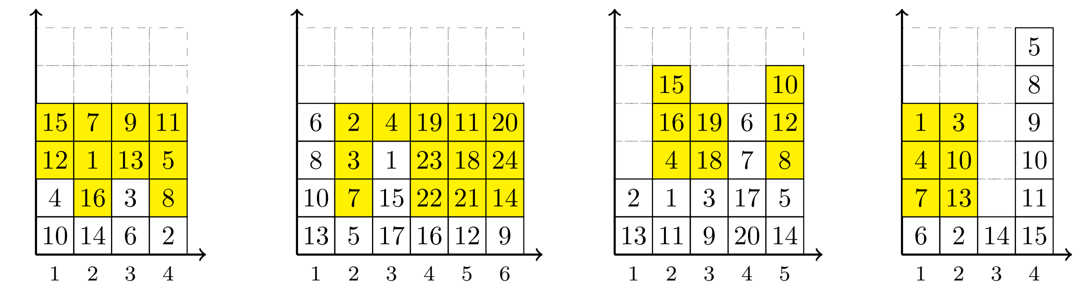
*Proof.* The following situations are the only options for solving the CPMP in $\text{IBF}^1$ moves when Conditions 1 and 2 hold:

(S1.1) A stack $s_1$ with exactly $f^{\text{M}} + 1$ misoverlaying containers is repaired first using BB moves, and then another stack $s'_1$ can be repaired using only BG moves.

(S1.2) A stack $s_2$ with exactly $f^{\text{M}}$ misoverlaying containers is repaired using BB moves, and (S1.2a) the topmost non-misoverlaying container in stack $s_2$ is moved by a GB move, and then another stack $s'_2$ is repaired only by BG moves, or (S1.2b) another stack $s'_2$ is repaired by exactly one BB move before, during or after a BG move sequence.

We now show that for each of these situations there is a condition when it does not hold, and when all of the conditions hold, there is no way to solve the CPMP in only $\text{IBF}^1$ moves. First consider Condition 3 as applied to situation (S1.1). We first incur $(f^{\text{M}} + 1)$ BB moves to repair $s_1$. All remaining moves must be BG moves. However, Condition 3 stipulates there is no upside down stack that can be placed on top of the repaired stack. Thus, situation (S1.1) is not sufficient for solving the bay in just $\text{IBF}^1$ moves. Next, Condition 4 is applied to situation (S1.2a). According to the condition, even if we perform a GB move to remove the topmost non-misoverlaying container of $s_2$, we will not be able to flip the upside down stack anywhere without causing misoverlays. This would then require additional BB moves, meaning situation (S1.2a) is also not sufficient in just $\text{IBF}^1$ moves. Finally, Condition 5 is used in situation (S1.2b). To repair stack $s'_2 \in U'$, the top group of $s_2$ should be at least $p^{\text{X}}_{s'_2}$, granted that a single BG move to another stack is permitted. Obviously, it is not possible when Condition 5 applies. Thus, when Conditions 3, 4 and 5 hold, the CPMP cannot be solved using situations (S1.1), (S1.2a) or (S1.2b), and $\text{IBF}^2 := \text{IBF}^1 + 1$ is valid. □

**Example 1.** In this example, $\text{LB}^{\text{BF}} = 13$ ($n_{\text{BX}} = 10 + 2 = 12$, $n_{\text{GX}} = 1$) and $\text{IBF}^1 = 14$ for the CV4-4-14 instance in Figure 3.4a. Consider now solving the given bay in 14 moves. First, examine the option in which stack 2 (or 4) is repaired first, and then another stack is repaired only by BG moves. The next stack to repair would be stack 1, since it is upside down. However, the topmost container with value 15 is larger than the topmost non-misoverlaying container in stack 2 with a value of 14. Thus, only BG moves will not suffice. Repairing stack 4 first leads to a similar situation. Alternatively, consider repairing stack 1 (or 3) first. Then, the topmost container of stack 1 (resp. 3) is moved by a GB move and another stack



(a) CV4-4-14, Case 1. (b) CV-4-6-20, Case 2b. (c) CV-4-5-16, Case 3 (d) Case 4

Figure 3.4: Example instances showing the lower bound improvements, $\text{IBF}^2$.

must be repaired using only BG moves. Even if we move the container with group value 4 from stack 1 with a GB move, we still cannot repair any other stack with only BG moves, as only stack 1 is upside down. Repairing stack 3 first also does not work, since even after removing container 3, the containers from stack 1 cannot be moved on top of container 6 without creating additional misoverlays. A final option to achieve a 14 move solution would be as follows. After repairing stack 1 (or 3), another stack is repaired with one BB move and BG moves. If we consider using BG moves for containers 7 and 1 from stack 2 to 1, the stack is upside down if we then use the BB move for container 16. However, $7 > 4$, thus we cause a misoverlay in stack 1. The same argument is valid for stacks 3 and 4. Similar arguments can be made when stack 3 is repaired first, thus $\text{IBF}^2 = 15$ is a valid lower bound.

### Extension of $\text{IBF}^1$, case 2b: Only one stack is non-misoverlaid and it is not full

The bound $\text{IBF}^1 = \text{IBF}^0 + 1$ is valid when exactly one stack $s$ is non-misoverlaid and not full, and $\min_{s' \in U} m_{s'} > w_s$. Note how this bound only examines the top container of the upside down stack and the non-misoverlaid stack. If we dig one container deeper, in certain situations we can improve $\text{IBF}^1$ by one move. There are two situations to consider, namely when $\text{IBF}^1 = \text{IBF}^0 + 1$ and when $\text{IBF}^1 = \text{IBF}^0$.

**Proposition 6.** $\text{IBF}^2 := \text{IBF}^1 + 1$ *is a valid lower bound for the CPMP if all of the following are true:*

1. *Only one stack $s$ is non-misoverlaid and it is not full*
2. $\text{IBF}^1 = \text{IBF}^0 + 1$
3. $\min\limits_{s' \in U} m_{s'} > w_s^{\text{sec}}$
4. $\min\limits_{s'' \in U'} p_{s''}^{\text{X}} > w_s$

*Proof.* To solve the CPMP in $\text{IBF}^1 = \text{IBF}^0 + 1$ moves when Condition 1 holds, two options for repairing another stack are (S2b.1) the topmost non-misoverlaying container in stack $s$ is moved by a GB move, and then another stack $s'$ is repaired only by BG moves, and (S2b.2) another stack $s''$ is repaired by exactly one BB move before, during or after the BG move sequence. Since these are similar to (S1.2a) and (S1.2b) in the proof of Proposition 5, respectively, it is easy to show that (S2b.1) and (S2b.2) are not possible if Conditions 4 and 5 hold, respectively. Thus, $\text{IBF}^2 = \text{IBF}^1 + 1$ must hold.

□

**Proposition 7.** $\mathrm{IBF}^2 := \mathrm{IBF}^1 + 1$ *is a valid lower bound for the CPMP if all of the following are true:*

1. *Only one stack $s$ is non-misoverlaid and it is not full*
2. *Only one stack $s'$ is upside down and $p^{\mathrm{top}}_{s'} \le w_s$.*
3. $\mathrm{IBF}^1 = \mathrm{IBF}^0$
4. $\forall s'' \in S^{\mathrm{M}} \setminus \{s'\}$, $n^{\mathrm{BG}}_{s''} > 2 \vee p^{\mathrm{BG}}_{s''1} > \max\{w'_s, w'_{s'}\} \vee p^{\mathrm{BG}}_{s''2} > \min\{w'_s, w'_{s'}\}$, *holds, where*

$$w'_{s'} := \begin{cases} w^{\mathrm{sec}}_{s'} & n_{\mathrm{GX}} > 0 \wedge n_s + n^{\mathrm{M}}_{s'} < H, \\ w_{s'} & \textit{otherwise.} \end{cases}$$

*Proof.* The only option for solving the CPMP in $\mathrm{IBF}^1 = \mathrm{IBF}^0 + 1$ moves is to repair stack $s'$ with BG moves to stack $s$ and then another stack $s'' \in S^{\mathrm{M}} \setminus \{s'\}$ with BG moves to stacks $s$ and $s'$. When $n_{\mathrm{GX}} > 0$, we may perform GG moves from $s'$ to $s$ unless $s$ becomes full. Note that GG moves from $s$ to $s'$ are not possible since $s'$ was repaired through moves to $s$. These moves can change the top group of stack $s$ to $w^{\mathrm{sec}}_{s'}$. If both the cases are taken into account, the top groups of stacks $s$ and $s'$ are given by $w'_s$ and $w'_{s'}$, respectively. To repair stack $s''$ with only BG moves to stacks $s$ and $s'$, $n^{\mathrm{BG}}_{s''} \le 2$, $p^{\mathrm{BG}}_{s''1} \le \max(w'_s, w'_{s'})$, and $p^{\mathrm{BG}}_{s''2} \le \min(w'_s, w'_{s'})$ should all be satisfied. Condition 4 ensures that no such stack $s''$ exists. Therefore, an extra move is required and $\mathrm{IBF}^2$ is valid. □

**Example 2.** Figure 3.4b shows an example instance where $\mathrm{IBF}^2$ strengthens the lower bound. Stack 1 is (clearly) the non-misoverlaid, non-full stack, $s$. Stack 3 is the upside down stack $s'$ that can be placed on $s$ with only BG moves. The lower bound is $\mathrm{IBF}^1 = \mathrm{LB}^{\mathrm{BF}} = 15$ ($n_{\mathrm{BX}} = 13, n_{\mathrm{GX}} = 2$). Since $n_{\mathrm{GX}} > 0$, we can move the container with group 1 from stack 3 to stack 1. We now attempt to repair the other stacks with BG moves. Since three misoverlaying containers in stack 2 or 5 are in nondecreasing order of group values from top to bottom, at least three non-misoverlaid stacks are required for repairing them ($n^{\mathrm{BG}}_{s''} = 3$). Stack 4 needs two stacks that can accept its containers with groups 23 and 19 ($p^{\mathrm{BG}}_{41} = 23$ and $p^{\mathrm{BG}}_{42} = 19$). Similarly, stack 6 needs two non-misoverlaying stacks with topmost groups of at least 24 and 20. Thus, we cannot repair stacks 2, 4, 5 and 6, and the bound $\mathrm{IBF}^2 := \mathrm{IBF}^1 + 1 = 16$ is obtained.

### Extension of $\mathrm{IBF}^1$, case 3: Two stacks are non-misoverlaid, at least one is not full

Let the two non-misoverlaid stacks be stacks $s_1$ and $s_2$ and assume that $w_{s_1} \le w_{s_2}$ without loss of generality. Let $\mathrm{IBF}^1 := \mathrm{IBF}^0 + 1$ if we cannot repair any stack only by GG and BG moves.

$\mathrm{IBF}^2$ tries to increase $\mathrm{IBF}^1$ by 1 if $\mathrm{IBF}^1 = \mathrm{IBF}^0$. Let $w''_{s_1}$ and $w''_{s_2}$ be the group values of the topmost non-misoverlaying containers in stacks $s_1$ and $s_2$, respectively, after GG moves between them. If $n_{\mathrm{GX}} = 0$, no GG moves are permitted, so that $w''_{s_1} := w_{s_1}$ and $w''_{s_2} := w_{s_2}$. If $n_{\mathrm{GX}} = 0$ and either stack $s_1$ or $s_2$ is full, we consider the lower bound in case 4 rather than here in case 3. When $n_{\mathrm{GX}} > 0$, the following situations set the values of $w''_{s_1}$ and $w''_{s_2}$

(S3.1) If $w_{s_1} < w_{s_2}$ and stack $s_2$ is full ($n_{s_2} = H$), then no GG moves are possible, only stack $s_1$ is available for repairing, and $w''_{s_1} = w_{s_1}, w''_{s_2} = -\infty$.
(S3.2) Otherwise.

$$w''_{s_1} := \begin{cases} w^{\text{sec}}_{s_1} & n_{s_2} < H, \\ w_{s_1} & \text{otherwise,} \end{cases} \qquad w''_{s_2} := \begin{cases} w^{\text{sec}}_{s_2} & n_{s_1} < H \wedge w_{s_1} = w_{s_2}, \\ w_{s_2} & \text{otherwise.} \end{cases} \tag{3.2}$$

**Proposition 8.** $\text{IBF}^2 := \text{IBF}^1 + 1$ *is a valid lower bound for the CPMP if all the following are true:*

1. *Two stacks are non-misoverlaid and at least one of these is not full*
2. $\text{IBF}^1 = \text{IBF}^0$
3. $\forall s' \in S^{\text{M}}$, $n^{\text{BG}}_{s'} > 2 \vee p^{\text{BG}}_{s'1} > \max\{w''_{s_1}, w''_{s_2}\} \vee p^{\text{BG}}_{s'2} > \min\{w''_{s_1}, w''_{s_2}\}$ *holds with values* $w''_{s_1}$ *and* $w''_{s_2}$ *set according to situations (S3.1) and (S3.2).*

*Proof.* First, we note that the situations (S3.1) and (S3.2) provide us with the largest possible group values on stacks $s_1$ and $s_2$ given the GG moves available. We must then repair one of the misoverlaid stacks using only BG moves to $s_1$ and $s_2$. However, when Condition 3 holds, there are no misoverlaid stacks that have containers that fit into $s_1$ and $s_2$ without requiring an additional move. □

**Example 3.** The instance in Figure 3.4c has $\text{IBF}^1 = \text{IBF}^0 = \text{LB}^{\text{BF}} = 9$ ($n_{\text{BX}} = 8, n_{\text{GX}} = 1$). According to our definition, $s_1 = 1$ and $s_2 = 4$. As $w_{s_1} < w_{s_2}$ and stack $s_2$ is not full, (S3.1) does not hold and we apply (S3.2) so the topmost groups are 13 and 6. We try to repair stacks 2, 3, or 5 with BG moves to stacks 1 or 4. Stacks 2 and 3 cannot be repaired as groups of the misoverlaying containers are larger than those of $s_1$ and $s_2$. Analyzing stack 5, despite container 12 not causing a problem ($p^{\text{BG}}_{51} = 12 \leq \max\{c_{s_1}, c_{s_2}\} = 13$), the top group of stack 3 is not large enough to support container 10 ($p^{\text{BG}}_{52} = 10 > \min\{c_{s_1}, c_{s_2}\} = 6$). Thus, no stack can be repaired and we obtain $\text{IBF}^2 := 9 + 1 = 10$.

## Extension of IBF$^1$, case 4: $n_{\text{GX}} = 0$ and only one non-misoverlaid stack is not full

When $\text{IBF}^1 = \text{IBF}^0 + 1$, we further try to increase $\text{IBF}^1$ by 1. Let $s$ be the non-full, non-misoverlaid stack, and $s'$ the stack to be repaired. Since $n_{\text{GX}} = 0$, we only need to consider one GG or BB move in addition to $n_{\text{BX}}(= \text{IBF}^0)$ BG moves. There are two situations to consider, namely (S4.1) when we perform a GG move from a full, non-misoverlaid stack $s''$ to stack $s$ and stack $s'$ is repaired with only BG moves to $s$ and $s''$, or (S4.2) when stack $s'$ is repaired by BG moves with exactly one BB move before, during or after the BG move sequence. Excluding these two situations, $\text{IBF}^2 := \text{IBF}^1 + 1$.

**Proposition 9.** $\text{IBF}^2 := \text{IBF}^1 + 1$ *is a valid lower bound for the CPMP if all the following are true:*

1. *Only one non-misoverlaid stack is not full*
2. $n_{\text{GX}} = 0$

*3.* $\mathrm{IBF}^1 = \mathrm{IBF}^0 + 1$

*4.* $\min_{s' \in U'} p^{\mathrm{X}}_{s'} > w_s$

*Proof.* First, consider situation (S4.1) in which we perform a GG move from $s''$ to $s$. Since we are only left with BG moves, we need to be able to place the misoverlaying containers from one of the misoverlaid stacks $s'$ on top of $s''$ and $s$ without any extra moves. Furthermore, stack $s''$ can accept only one container as it was full before the GG move. Thus, $s' \in U'$ and at least $p^{\mathrm{X}}_{s'} \leq w_s$ should hold to repair it. Next, for situation (S4.2), we perform a BB move when there is a stack that would be upside down if one container was removed from it. The same argument is true in this case, and $s' \in U'$ and $p^{\mathrm{X}}_{s'} \leq w_s$ are required. Thus, when $\min_{s' \in U'} p^{\mathrm{X}}_{s'} > w_s$, $\mathrm{IBF}^2 := \mathrm{IBF}^1 + 1$ is a valid lower bound.

□

**Example 4.** Figure 3.4d shows an example in which $\mathrm{IBF}^0 = \mathrm{LB}^{\mathrm{BF}} = 6$ ($n_{\mathrm{BX}} = 6$, $n_{\mathrm{GX}} = 0$) and $\mathrm{IBF}^1 = 7$. As neither stack 1 nor 2 becomes upside down by removing only one misoverlaying container, we cannot arrange the bay only with one BB move and BG moves. Therefore, $\mathrm{IBF}^2 = \mathrm{IBF}^1 + 1 = 8$.

### 3.4.4 Extended lower bound IBF$^3$

We now change our focus to dealing with cases where even $\mathrm{IBF}^2$ cannot improve the $\mathrm{LB}^{\mathrm{BF}}$, i.e., $\mathrm{IBF}^2 = \mathrm{LB}^{\mathrm{BF}}$. If at least one non-misoverlaid stack is not full, $\mathrm{IBF}^3$ tries to repair each stack in $S^{\mathrm{M}}$ with only BG moves to the stacks in $S^{\mathrm{N}}$, without considering stack height limits. If $n_{\mathrm{GX}} > 0$, at best these $GX$ moves empty $n^{\mathrm{S}}_{\mathrm{GX}}$ stacks. Considering this, we add $n^{\mathrm{S}}_{\mathrm{GX}}$ dummy empty stacks to $S^{\mathrm{N}}$. If a misoverlaid stack can be repaired, all of its misoverlaying containers are removed from the bay, and it is moved from $S^{\mathrm{M}}$ to $S^{\mathrm{N}}$ for further repairs. When this process fails to repair all the stacks, at least one extra move will be necessary and we obtain $\mathrm{IBF}^3 := \mathrm{LB}^{\mathrm{BF}} + 1$. To speed up the process, we assume that at most three non-misoverlaid stacks are necessary for repairing a stack, even when more than three are required.

**Proposition 10.** $\mathrm{IBF}^3$ *is a valid lower bound for the CPMP if* $\mathrm{IBF}^2 = \mathrm{LB}^{\mathrm{BF}}$.

*Proof.* Suppose that the procedure in the proposition fails to repair all the stacks in $S^{\mathrm{M}}$. Then, we will not be able to repair any of the remaining stacks in $S^{\mathrm{M}}$ with only BG moves even when all misoverlaying containers in the other stacks are removed from the bay. Since GG moves are taken into consideration by adding empty stacks to $S^{\mathrm{N}}$, at least an additional move is needed to repair the bay. □

### 3.4.5 Extended lower bound IBF$^4$

When $\mathrm{IBF}^3$ fails to improve the lower bound over $\mathrm{LB}^{\mathrm{BF}}$, $\mathrm{IBF}^4$ offers one last chance for improving the bound by one move. Recall that in $\mathrm{LB}^{\mathrm{BF}}$ the supply and demand of each group is computed. These values try to quantify how many places are available to accept containers of a particular group or are required for containers of a group, respectively. This bound takes advantage of the situation when at some $\hat{p}$ the cumulative demand surplus $D^{\mathrm{S}}(\hat{p}) = D(\hat{p}) - S^{\mathrm{P}}(\hat{p}) = 0$. When supply and demand are balanced at $\hat{p}$, it means that the demand with group $p < \hat{p}$ cannot be assigned to any supply of group $p \geq \hat{p}$. The groups are thus partitioned by balanced supply and demand and we examine if containers providing the demand can be moved to stacks providing supply using only BG moves. In certain situations in which this is not possible, $\mathrm{IBF}^4 = \mathrm{IBF}^3 + 1$.

We introduce some setup information and several terms to help us describe the $\mathrm{IBF}^4$ bound. First, when $n_{\mathrm{GX}} > 0$, assume each $n^{\mathrm{S}}_{\mathrm{GX}}$ stack provides $H$ supply for the largest group. Let $[\check{p}^k, \hat{p}^k]$ be the $k$th subset of groups partitioned in the above manner, so that $D^{\mathrm{S}}(\hat{p}^k + 1) = 0$ holds. For each $\bar{p}$ satisfying $\check{p}^k \leq \bar{p} \leq \hat{p}^k$, we only consider from now on the demand and supply of groups $p$ such that $\bar{p} \leq p \leq \hat{p}^k$. We refer to a stack having one or more containers with demand as a demand stack. Formally, a stack $s$ is a demand stack if $\bar{p} \leq m^i_s \leq \hat{p}^k$ holds for some $i$. A stack that provides supply is referred to as a supply stack: A supply stack $s'$ satisfies $\bar{p} \leq w_{s'} \leq \hat{p}^k$. We call a stack "*dirty*" when it is a demand and supply stack, and the demand and supply from a dirty stack *dirty demand* and *dirty supply*, respectively. The total number of dirty demands in a dirty stack with groups $p$ satisfying $\bar{p} \leq p \leq \hat{p}^k$ is referred to as the *dirty height.* We refer to a demand stack and a supply stack as being *clean* when they are not dirty, and their demand and supply as *clean demand* and *clean supply.* If one of the following conditions is satisfied, $\mathrm{IBF}^3 = \mathrm{LB}^{\mathrm{BF}}$ can be improved by 1:

1. The minimum dirty height is larger than the clean supply.
2. There is only one *clean supply stack*, there are no *dirty stacks*, and there exists a *clean demand stack* such that the group of the topmost demand container is smaller than the maximum group of the demand containers in that stack.
3. There is only one *clean supply stack* and the maximum group of the dirty demand containers is larger than the topmost dirty demand containers and the topmost non-misoverlaying containers in *dirty stacks.*

**Proposition 11.** *IBF⁴ is a valid lower bound for the CPMP if* $\mathrm{IBF}^3 = \mathrm{LB}^{\mathrm{BF}}$.

*Proof.* 1. If the minimum dirty height is larger than clean supply, none of dirty stacks can be repaired only by BG moves, so that $\mathrm{IBF}^3 = \mathrm{LB}^{\mathrm{BF}}$ is improved by 1.

2. If there is only one clean supply stack and no dirty stack, clean demand containers are moved to the unique clean supply stack only by BG moves. It is not possible when there is a clean demand stack where demand containers are not upside down, therefore $\mathrm{IBF}^3 = \mathrm{LB}^{\mathrm{BF}}$ can be increased by 1. To speed up the check, we only search for such a clean demand stack that the priority of the topmost demand container is smaller than the maximum group value of demand containers in that stack.
3. If there is only one clean stack, at least one of the topmost dirty demand containers is moved first to that stack. Let $p^{\mathrm{D}}_{\max}$ be the maximum group of dirty demand containers. A container with group $p^{\mathrm{D}}_{\max}$ should also be moved to the clean supply stack if $p^{\mathrm{D}}_{\max}$ is larger than topmost non-misoverlaying containers in dirty stacks. However, a BG move is not possible if $p^{\mathrm{D}}_{\max}$ is larger than the topmost dirty demand containers, thus we can improve $\mathrm{IBF}^3$ by 1.

□

## 3.5 Computational experiments

We carry out an extensive computational analysis to assess the performance of the algorithm proposed for the CPMP, which we refer to as $PT^A$. We test our approach and several algorithm variants that arise from different combinations of the contributions presented in this study, see Table 3.2. The letters U, T, G, and XB, refer to unrelated move, transitive move, same group, and BB/GB move avoidance dominance rules, respectively. "Gen." refers to the generalized rules and $BCA^1$ and $BCA^2$ to the branching comparison algorithm presented in Tanaka and Tierney (2018) and our modification in this work, respectively. These novel approaches are evaluated against the state-of-art exact algorithm from Tanaka and Tierney (2018), and obtain a significantly larger number of optimal solutions as well as feasible solutions of unsolved instances. We focus on answering the following research questions:

1. What effect do the improved lower bounds have on the performance of $PT^A$?
2. What effect do the dominance rules have on the performance of $PT^A$?
3. What is the effect of the new branching strategy?
4. Do the benefits of using the memoization-based upper bound heuristic outweigh the time invested?
5. Does $PT^A$ improve the state-of-the-art exact algorithms?
6. Are we overfitting the well-known datasets?

We conduct all computational experiments on Intel Xeon X5650 CPUs at 2.67 GHz running CentOS 6.6, with each execution of the algorithms being given a maximum of 3 GB of RAM and an hour of CPU time.

Table 3.2: An overview of the algorithms directly compared in this section.

| Algorithm | Lower bound | Branching | Dominance rules | Hash table |
|---|---|---|---|---|
| $PT^A$ | $IBF^2$, $IBF^3$, $IBF^4$ | $BCA^2$ | Gen.(T, U, G), XB | Initial UB |
| $PT^D$ | $IBF^2$, $IBF^3$, $IBF^4$ | $BCA^2$ | Gen.(T, U, G), XB | - |
| $PT^{4+}$ | $IBF^2$, $IBF^3$, $IBF^4$ | $BCA^2$ | T, U, G | - |
| $PT^4$ | $IBF^2$, $IBF^3$, $IBF^4$ | $BCA^1$ | T, U, G | - |
| $PT^3$ | $IBF^2$, $IBF^3$ | $BCA^1$ | T, U, G | - |
| $PT^2$ | $IBF^2$ | $BCA^1$ | T, U, G | - |
| $TT^A$ (Tanaka and Tierney, 2018) | $IBF^1$ | $BCA^1$ | T, U, G | - |
| IDA* (Tierney et al., 2017) | $LB^{BF}$ | Left | T, U | - |

### 3.5.1 Test instances

We focus on instance sets from Bortfeldt and Forster (2012) (BF dataset) and Caserta and Voß (2009) (CV dataset) as they contain the most difficult instances in the literature. In Tanaka and Tierney (2018) three other datasets were used from van Brink and van der Zwaan (2014) (BZ dataset), from Expósito-Izquierdo et al. (2012) (EMM dataset), and from Zhang et al. (2015) (ZJY dataset); however, these instances are all solved to optimality in little runtime by $TT^A$, so we do not consider them again here. Finally, the CPMP has been thoroughly studied, and most of the state-of-art exact algorithms use the previous datasets to test their performance. Avoiding overfitting of the studied methods on these instances is a crucial concern. To ensure that we are not overfitting, we also consider a fresh dataset, CVX, from Hottung et al. (2020). Experimental results on this dataset are conducted at the end of the section, showing that our approach also performs well on new data and definitely does not overfit. We now briefly describe the datasets.

**CV dataset** In these instances, all containers have different group values and stacks are filled to the same height. The instances range in size from 3 tiers by 3 stacks up to 6 tiers by 10 stacks full of containers. Two empty tiers are added above the top of each stack.

**BF dataset** This dataset is made up of two group values (BF and LC). BF contains 32 categories, having either 16 or 20 stacks with a height of either 8 or 5 tiers. LC is divided into 5 categories whose instances are based on the instances by Lee and Chao (2009). They have either 10 or 12 stacks and either 5 or 6 tiers. Since the load capacity of these instances is under 100%, it is not necessary to add empty tiers as in the CV dataset. Moreover, unlike the CV dataset, there can be multiple containers of the same group.

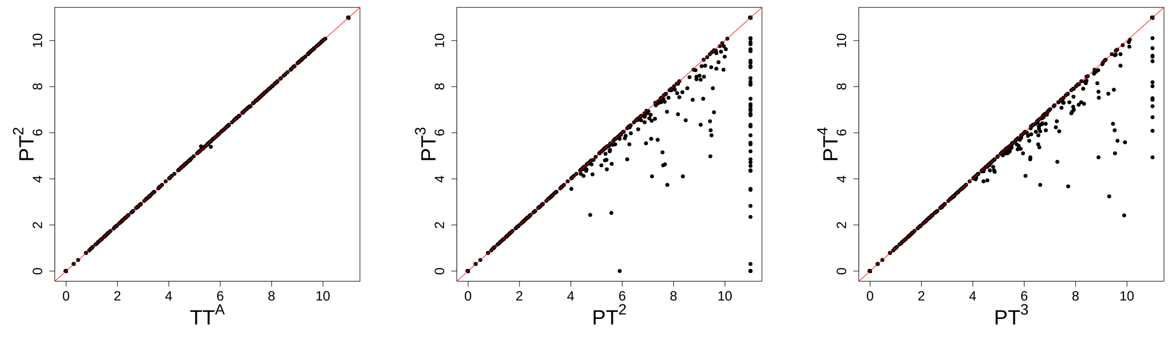
**CVX dataset** There are three groups of instances in the CVX dataset: CVX1, CVX2, and CVX3. Instances in CVX1 follow the structure of CV (all containers have different group values and stacks are filled to the same height). Stacks are also filled to the same height in CVX2 and CVX3, but in these cases, there are two and three containers of each group value, respectively. The instances in CVX1-CVX3 range in size from 5 tiers by 7 stacks up to 5 tiers by 10 stacks full of containers, with two extra empty tiers are added as in the CV dataset.

### 3.5.2 Improved lower bounds IBF$^2$, IBF$^3$, IBF$^4$

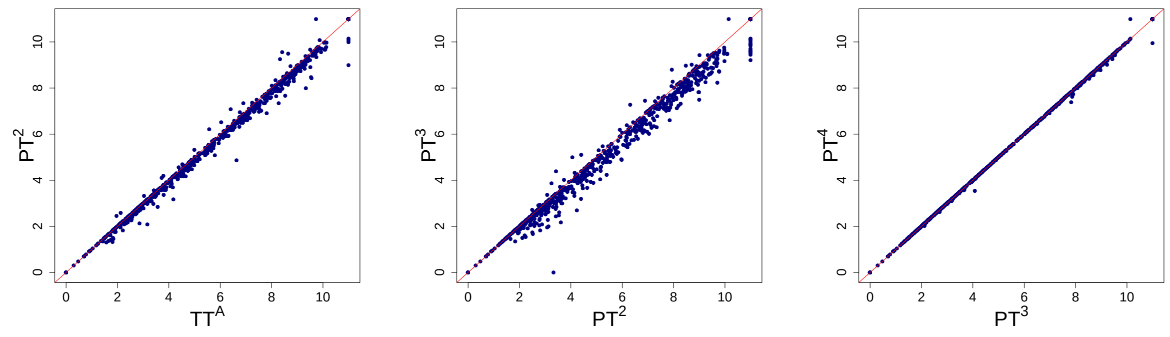
We evaluate the effectiveness of the improved lower bounds, IBF$^2$, IBF$^3$, and IBF$^4$, presented in this work. Since IBF$^3$ tries to improve the bound only if IBF$^2$ fails, and IBF$^4$, in turn, only tries to improve the bound only if IBF$^3$ fails, we assess their incremental effectiveness.

**Root node analysis** Given the strength of IBF$^1$, it is not surprising that the benefits of the improved bounds are limited in the root node, except for IBF$^2$ on the CV dataset that increases the lower bound by 1 on 200 of 760 instances. Nevertheless, we now show that they are useful during the search.

**BF Dataset** Figure 3.5a shows the logarithmic number of nodes searched for solving BF instances with and without the improved lower bounds. Points below the line indicate improved solving performance from the new lower bound, whereas points above the line indicate degraded performance. The most effective bounds on the BF dataset are IBF$^3$ and IBF$^4$. We highlight that PT$^3$ solves 44 more instances than PT$^2$ (which solves the same as TT$^A$), as well as dramatically reduces the number of nodes in all the instances. PT$^4$ adds 13 more optimal solutions to that achieved by PT$^3$, finding 499 optimal solutions instead of the 442 of $TT^A$. The number of nodes decreases again significantly, especially on the most challenging instances. Considering just the 442 cases in which $PT^4$ and $TT^A$ achieve optimal solutions, there are 8 categories in which the reduction percentage of the number of nodes exceeds 90%, and the CPU time is reduced on average by 76%.



(a) BF instances



(b) CV instances

Figure 3.5: Number of nodes necessary for finding an optimal solution on all BF and CV instances with the lower bound improvements ($PT^2$, $PT^3$, $PT^4$) and without ($TT^A$). The graphics uses a log-10 scale and unsolved instances are assigned a node count of $10^{11}$.

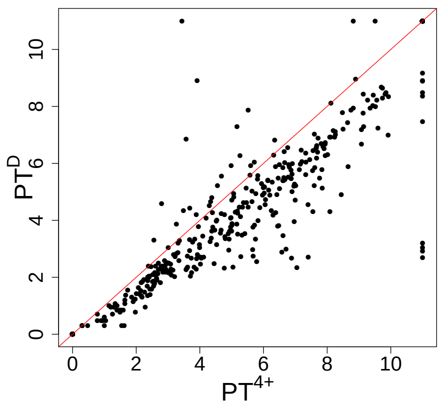
**CV Dataset** In general, $PT^2$ finds solutions in slightly fewer nodes than $TT^A$, see Figure 3.5b. $PT^3$ obtains a greater reduction in the number of nodes over $PT^2$ whereas the number of nodes explored in $PT^3$ and $PT^4$ is quite similar, with slight gains for $PT^4$. In regards to the number of optimal solutions, $PT^2$ solves 3 more instances to optimality than $TT^A$, while $PT^3$ and $PT^4$ solve 16 more than $PT^2$. $PT^4$ reduces the number of nodes on average by 65% and the CPU time by 71% across instances optimally solved by both $PT^4$ and $TT^A$.

### 3.5.3 Branching strategy

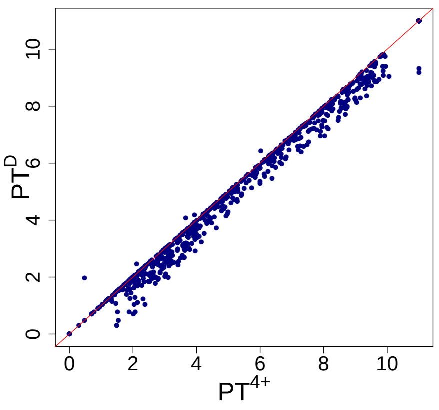
We next examine the performance of the branching comparison algorithm $BCA^2$ proposed in Section 3.3.2 against that used by $TT^A$, changing the branching strategy of $PT^4$ and calling this variant $PT^{4+}$. $PT^{4+}$ solves 502 instances to optimality instead of 499 on the BF instances, and 681 instead of 679 cases for the CV instances. The number of feasible solutions goes from 592 to 599 on the BF dataset, and from 710 to 713 on the CV dataset.

### 3.5.4 New dominance rules

Figure 3.6 shows the logarithmic number of nodes explored for solving CV and BF instances with $PT^{4+}$, and with $PT^{D}$ (including all the new dominance rules), clearly showing that the new dominance rules introduced in this chapter reduce the number of nodes searched.



(a) BF instances



(b) CV instances

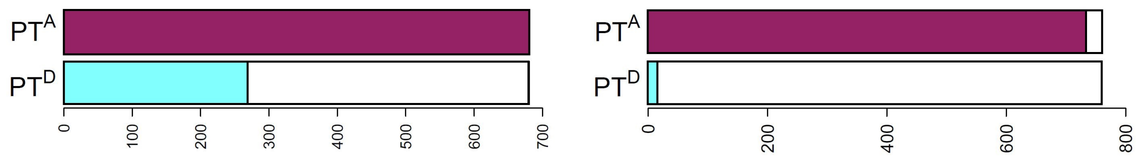
Figure 3.6: Number of nodes necessary for finding an optimal solution on all BF and CV instances with $PT^{4+}$ and $PT^{D}$. The graphics use a log-10 scale and unsolved instances are assigned a node count of $10^{11}$.

On the BF dataset, $PT^{D}$ solves 7 more instances to optimality and cuts the number of nodes down on average by 93% and CPU time on average by 47% in those instances optimally solved by both versions. We obtain new optimal solutions in some of the most challenging categories such as BF13, BF15, and BF31. The average node count is reduced in almost all categories and the reduction percentage exceeds 90% in 19 of them. On the CV dataset, $PT^{D}$ optimally solves 3 more instances and cuts the number of examined nodes down by 49%. Nevertheless, CPU time increases by 23%.

### 3.5.5 Memoization-based upper bound heuristic

The algorithm $PT^{D}$ finds a feasible solution on the root node in 268 instances of 681 on the BF dataset and in 16 instances of 760 on the CV dataset. Using the memoization strategy, we obtain 411 and 719 more feasible solutions on the root node on BF and CV instances, respectively, as shown in Figure 3.7. This strategy spends an average CPU time higher than 1 second only in three categories: BF30 (1.03 seconds), BF31 (1.18 seconds), and BF32 (3.58 seconds). Our heuristic finds a feasible solution for all the instances of these categories while $TT^{A}$ does not provide a solution in any of them. The average CPU time is still small on the CV dataset, only 0.02 seconds on average.

$PT^{A}$ solves the same number of instances as $PT^{D}$ but finds at least one feasible solution on 679 instances of BF instead of 611, and on 751 instances of CV instead of 712. The feasible solutions achieved during the iterative deepening are progressively closer to the lower bound than those obtained initially by the memoization strategy, but $PT^{A}$ allows having feasible solutions in a more significant set of instances in an integrated algorithm.



(a) BF instances (b) CV instances

Figure 3.7: The colored bars represent feasible solutions found on the root node with PT$^A$ and PT$^D$ on BF and CV datasets.

## 3.5.6 Comparison with the state-of-the-art

Table 3.3: Algorithm performance of the IDA* algorithm from Tierney et al. (2017), the TT$^A$ algorithm from Tanaka and Tierney (2018) and PT$^D$ and PT$^A$ on the BF dataset from Bortfeldt and Forster (2012) grouped by categories (Cat).

| | | # | | Avg. Time | | | Avg. Log Nodes | | | # Optimal | | | # Feasible | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Cat. | $C$ | Inst. | IDA* | TT$^A$ | PT$^A$ | IDA* | TT$^A$ | PT$^A$ | IDA* | TT$^A$ | PT$^A$ | IDA* | TT$^A$ | PT$^A$ |
| BF1 | 48 | 20 | 121.77 | 0.00 | 0.11 | 11.05 | 0.00 | 0.00 | 18 | 20 | 20 | 18 | 20 | 20 |
| BF2 | 48 | 20 | 175.81 | 0.00 | 0.04 | 8.73 | 0.00 | 0.00 | 15 | 20 | 20 | 15 | 20 | 20 |
| BF3 | 48 | 20 | 271.40 | 0.01 | 0.03 | 11.10 | 0.48 | 0.39 | 11 | 20 | 20 | 11 | 20 | 20 |
| BF4 | 48 | 20 | 11.81 | 0.00 | 0.05 | 9.91 | 0.12 | 0.12 | 18 | 20 | 20 | 18 | 20 | 20 |
| BF5 | 64 | 20 | 567.62 | 209.51 | 19.83 | 16.57 | 12.02 | 7.77 | 8 | 18 | 20 | 8 | 20 | 20 |
| BF6 | 64 | 20 | 504.27 | 165.47 | 0.52 | 20.16 | 14.39 | 9.18 | 1 | 18 | 20 | 1 | 20 | 20 |
| BF7 | 64 | 20 | 237.97 | 8.18 | 2.74 | 17.74 | 13.24 | 10.49 | 10 | 20 | 20 | 10 | 20 | 20 |
| BF8 | 64 | 20 | 845.54 | 138.56 | 53.25 | 19.28 | 15.49 | 11.86 | 3 | 20 | 20 | 3 | 20 | 20 |
| BF9 | 77 | 20 | 2084.96 | 150.59 | 123.45 | 21.13 | 13.98 | 9.05 | 5 | 16 | 19 | 5 | 20 | 20 |
| BF10 | 77 | 20 | 257.22 | 0.00 | 6.14 | 15.02 | 1.63 | 3.37 | 3 | 11 | 18 | 3 | 20 | 20 |
| BF11 | 77 | 20 | 338.50 | 293.10 | 16.10 | 18.59 | 15.51 | 10.88 | 3 | 9 | 18 | 3 | 20 | 20 |
| BF12 | 77 | 20 | - | 104.23 | 70.12 | - | 6.14 | 5.92 | 0 | 12 | 20 | 0 | 20 | 20 |
| BF13 | 103 | 20 | - | 1013.68 | 546.06 | - | 21.80 | 18.21 | 0 | 3 | 5 | 0 | 13 | 20 |
| BF14 | 103 | 20 | - | - | - | - | - | - | 0 | 0 | 0 | 0 | 7 | 20 |
| BF15 | 103 | 20 | - | - | 1556.41 | - | - | 19.55 | 0 | 0 | 2 | 0 | 13 | 20 |
| BF16 | 103 | 20 | - | 165.25 | 1067.33 | - | 20.21 | 19.43 | 0 | 1 | 1 | 0 | 6 | 18 |
| BF17 | 60 | 20 | 0.00 | 0.01 | 0.09 | 6.85 | 0.66 | 0.49 | 6 | 20 | 20 | 6 | 20 | 20 |
| BF18 | 60 | 20 | 12.83 | 0.01 | 0.19 | 8.19 | 0.00 | 0.00 | 16 | 20 | 20 | 16 | 20 | 20 |
| BF19 | 60 | 20 | 2.55 | 0.01 | 0.10 | 9.87 | 0.60 | 0.29 | 11 | 20 | 20 | 11 | 20 | 20 |
| BF20 | 60 | 20 | 24.57 | 0.01 | 0.09 | 9.00 | 0.00 | 0.00 | 14 | 20 | 20 | 14 | 20 | 20 |
| BF21 | 80 | 20 | 727.45 | 145.63 | 11.54 | 18.11 | 14.00 | 8.25 | 6 | 18 | 20 | 6 | 20 | 20 |
| BF22 | 80 | 20 | - | 212.79 | 0.87 | - | 11.87 | 8.59 | 0 | 16 | 19 | 0 | 20 | 20 |
| BF23 | 80 | 20 | 37.60 | 186.99 | 87.24 | 15.06 | 12.14 | 8.82 | 3 | 17 | 20 | 3 | 20 | 20 |
| BF24 | 80 | 20 | - | 566.22 | 9.44 | - | 17.34 | 11.47 | 0 | 16 | 18 | 0 | 20 | 20 |
| BF25 | 96 | 20 | - | 378.14 | 258.79 | - | 8.64 | 7.28 | 0 | 8 | 12 | 0 | 20 | 20 |
| BF26 | 96 | 20 | - | 0.01 | 22.85 | - | 1.82 | 2.58 | 0 | 16 | 19 | 0 | 20 | 20 |
| BF27 | 96 | 20 | 1203.96 | 96.61 | 28.66 | 20.32 | 12.75 | 10.14 | 2 | 11 | 17 | 2 | 20 | 20 |
| BF28 | 96 | 20 | 238.33 | 0.01 | 237.67 | 18.90 | 2.75 | 4.12 | 1 | 11 | 15 | 1 | 20 | 20 |
| BF29 | 128 | 20 | - | - | 512.44 | - | - | 18.94 | 0 | 0 | 1 | 0 | 11 | 20 |
| BF30 | 128 | 20 | - | - | 1538.85 | - | - | 19.77 | 0 | 0 | 2 | 0 | 7 | 20 |
| BF31 | 128 | 20 | - | - | 1788.39 | - | - | 19.77 | 0 | 0 | 2 | 0 | 9 | 20 |
| BF32 | 128 | 20 | - | - | - | - | - | - | 0 | 0 | 0 | 0 | 4 | 20 |
| LC1 | 35 | 1 | 0.00 | 0.00 | 0.01 | 5.98 | 4.48 | 3.53 | 1 | 1 | 1 | 1 | 1 | 1 |
| LC2a | 50 | 10 | 208.33 | 2.67 | 0.86 | 13.50 | 10.32 | 7.82 | 9 | 10 | 10 | 9 | 10 | 10 |
| LC2b | 50 | 10 | 312.96 | 1.91 | 0.10 | 17.00 | 12.71 | 6.16 | 6 | 10 | 10 | 6 | 10 | 10 |
| LC3a | 54 | 10 | 331.53 | 1.27 | 2.95 | 18.15 | 13.24 | 9.90 | 6 | 10 | 10 | 6 | 10 | 10 |
| LC3b | 54 | 10 | 403.67 | 14.22 | 3.83 | 18.25 | 15.50 | 11.58 | 7 | 10 | 10 | 7 | 10 | 10 |
| Total: | | 681 | - | - | - | - | - | - | 183 | 442 | 509 | 183 | 591 | 679 |

The column #Inst. contains the number of instances tested in each group. The average time and average log nodes include only instances for which an optimal solution was found. The notation "-" is used when the averages cannot be calculated.

Table 3.4: Algorithm performance of the IDA* algorithm from Tierney et al. (2017), the TT$^{A}$ algorithm from Tanaka and Tierney (2018) and PT$^{A}$ on the CV dataset from Caserta and Voß (2009).

| | | | # | Avg. Time | | | Avg. Log Nodes | | | # Optimal | | | # Feasible | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *H* | *S* | *C* | Inst. | IDA* | TT$^{A}$ | PT$^{A}$ | IDA* | TT$^{A}$ | PT$^{A}$ | IDA* | TT$^{A}$ | PT$^{A}$ | IDA* | TT$^{A}$ | PT$^{A}$ |
| 5 | 3 | 9 | 40 | 0.00 | 0.00 | 0.00 | 6.08 | 5.02 | 4.29 | 40 | 40 | 40 | 40 | 40 | 40 |
| 5 | 4 | 12 | 40 | 0.00 | 0.00 | 0.00 | 6.68 | 5.11 | 4.05 | 40 | 40 | 40 | 40 | 40 | 40 |
| 5 | 5 | 15 | 40 | 0.01 | 0.00 | 0.00 | 7.76 | 6.15 | 4.48 | 40 | 40 | 40 | 40 | 40 | 40 |
| 5 | 6 | 18 | 40 | 0.01 | 0.00 | 0.00 | 8.03 | 6.06 | 4.26 | 40 | 40 | 40 | 40 | 40 | 40 |
| 5 | 7 | 21 | 40 | 0.03 | 0.00 | 0.01 | 9.24 | 6.95 | 4.93 | 40 | 40 | 40 | 40 | 40 | 40 |
| 5 | 8 | 24 | 40 | 0.07 | 0.00 | 0.01 | 9.27 | 6.75 | 4.80 | 40 | 40 | 40 | 40 | 40 | 40 |
| 6 | 4 | 16 | 40 | 0.66 | 0.04 | 0.04 | 12.32 | 10.15 | 9.40 | 40 | 40 | 40 | 40 | 40 | 40 |
| 6 | 5 | 20 | 40 | 2.20 | 0.04 | 0.03 | 12.92 | 10.13 | 9.08 | 40 | 40 | 40 | 40 | 40 | 40 |
| 6 | 6 | 24 | 40 | 15.12 | 0.22 | 0.09 | 14.52 | 11.66 | 9.93 | 40 | 40 | 40 | 40 | 40 | 40 |
| 6 | 7 | 28 | 40 | 29.91 | 0.75 | 0.37 | 15.76 | 12.92 | 10.68 | 40 | 40 | 40 | 40 | 40 | 40 |
| 7 | 4 | 20 | 40 | 299.50 | 36.97 | 32.10 | 17.91 | 16.21 | 15.24 | 34 | 40 | 40 | 34 | 40 | 40 |
| 7 | 5 | 25 | 40 | 396.38 | 106.89 | 96.68 | 19.11 | 16.75 | 15.54 | 35 | 40 | 40 | 35 | 40 | 40 |
| 7 | 6 | 30 | 40 | 883.10 | 231.66 | 169.60 | 20.21 | 18.91 | 17.29 | 17 | 39 | 40 | 17 | 40 | 40 |
| 7 | 7 | 35 | 40 | 732.43 | 507.22 | 312.16 | 19.63 | 18.66 | 17.02 | 18 | 37 | 40 | 18 | 40 | 40 |
| 7 | 8 | 40 | 40 | 1332.14 | 376.94 | 222.61 | 20.91 | 19.83 | 17.77 | 10 | 35 | 38 | 10 | 40 | 40 |
| 7 | 9 | 45 | 40 | 2016.98 | 466.57 | 470.61 | 21.46 | 20.33 | 18.34 | 6 | 30 | 38 | 6 | 40 | 40 |
| 7 | 10 | 50 | 40 | 1901.81 | 360.27 | 327.51 | 21.46 | 19.99 | 17.85 | 3 | 32 | 36 | 3 | 39 | 40 |
| 8 | 6 | 36 | 40 | - | 1318.63 | 1796.14 | - | 22.21 | 21.72 | 0 | 5 | 8 | 0 | 13 | 34 |
| 8 | 10 | 60 | 40 | - | 1980.76 | 1928.22 | - | 22.64 | 20.77 | 0 | 2 | 4 | 0 | 16 | 37 |
| | | Total: | 760 | - | - | - | - | - | - | 523 | 660 | 684 | 523 | 708 | 751 |

The column #Inst. contains the number of instances tested in each group. The average time and average log nodes include only instances for which an optimal solution was found. The notation "-" is used when the averages cannot be calculated.

**BF Instances** Table 3.3 shows the performance of the state-of-the-art approaches Tierney et al. (2017) and Tanaka and Tierney (2018), together with the proposed approach PT$^{A}$. PT$^{A}$ solves 15% more instances to optimality, 509 instead of 442. PT$^{A}$ fails to provide a feasible solution in only two instances of the category BF16, in contrast with the 90 instances in which TT$^{A}$ does not provide any solution. Furthermore, PT$^{A}$ solves 61 more instances of the hardest categories, those with 8 tiers (BF9 through BF16 and BF25 through BF32). Using PT$^{A}$, 15 categories through BF1-BF32 are completely solved to optimality, 5 more categories than TT$^{A}$. We note the large reduction in the number of examined nodes provided by the new contributions on categories where TT$^{A}$ and PT$^{A}$ achieve the same number of optimal solutions.

**CV Instances** We refer now to Table 3.4. TT$^{A}$ solves 660 of 760 instances, while PT$^{A}$ solves 684. All of the instances ranging in size from 3 tiers by 3 stacks up to 5 tiers by 7 are solved by PT$^{A}$, which is two more categories than TT$^{A}$. We note again that the number of examined nodes and the CPU time dramatically decrease on the most challenging categories.

**Running PT$^{A}$ for a day of CPU time** We also tried solving the BF and CV instances using a day of CPU time to gauge whether some solutions were barely out of reach of the one hour timeout. On the BF dataset, our approach solves 28 more instances to optimality, 2 of them in BF14 and also 2 in BF32. Remarkably, on the CV dataset, PT$^{A}$ is able to solve all the instances in the categories 5-8, 5-9, and 5-10 in 576.53, 1315.64, and 3076.00 seconds on average, respectively. Categories 5-9 and 5-10 represent the size of the largest RMGC systems we are aware of, meaning our algorithm is able to solve difficult real-world CPMP instances to optimality. Furthermore, it also solves 33 out of 80 instances of categories 6-6 and 6-10, which pose difficulty even to metaheuristic approaches.

**CVX Instances** We now evaluate both algorithms on a "fresh" dataset to ensure we are not overfitting our approach to the existing instances. The results are shown in Table 3.5. On this dataset, our algorithm still outperforms $\text{TT}^{\text{A}}$, solving 441 instances to optimality instead of 397 in the category in which all containers have different group values (CVX1), 468 instead of 442 in the category with at least two containers of each group (CVX2), and 486 instead of 468 in the category with at least three containers of each group (CVX3). A feasible solution is found for all the instances by $\text{PT}^{\text{A}}$ while $\text{TT}^{\text{A}}$ does not find any solution on 4 instances. $\text{PT}^{\text{A}}$ cuts the number of nodes down by 91% and the CPU time by 65%. These data show the excellent performance of our approach on a set of instances different to that used for its development.

Table 3.5: Algorithm performance of the $\text{TT}^{\text{A}}$ algorithm from Tanaka and Tierney (2018) and $\text{PT}^{\text{A}}$ on the CVX dataset from Hottung et al. (2020).

| | | # | | | Avg. Time | | Avg. Log Nodes | | # Optimal | | # Feasible | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $H$ | $S$ | $C$ | Inst. | $\text{TT}^{\text{A}}$ | $\text{PP}^{\text{A}}$ | $\text{TT}^{\text{A}}$ | $\text{PP}^{\text{A}}$ | $\text{TT}^{\text{A}}$ | $\text{PP}^{\text{A}}$ | $\text{TT}^{\text{A}}$ | $\text{PP}^{\text{A}}$ |
| CVX1 | 7 | 7 | 35 | 250 | 293.34 | 216.85 | 18.96 | 17.21 | 223 | 241 | 250 | 250 |
| | 7 | 10 | 50 | 250 | 750.33 | 414.07 | 20.58 | 18.15 | 174 | 200 | 247 | 250 |
| | | | Total: | 500 | - | - | - | - | 397 | 441 | 497 | 500 |
| CVX2 | 7 | 7 | 35 | 250 | 198.24 | 107.78 | 18.22 | 16.28 | 242 | 248 | 250 | 250 |
| | 7 | 10 | 50 | 250 | 452.63 | 314.86 | 20.01 | 17.52 | 200 | 220 | 249 | 250 |
| | | | Total: | 500 | - | - | - | - | 442 | 468 | 499 | 500 |
| CVX3 | 7 | 7 | 35 | 250 | 80.88 | 43.30 | 17.06 | 15.14 | 248 | 250 | 250 | 250 |
| | 7 | 10 | 50 | 250 | 318.79 | 212.28 | 19.22 | 16.67 | 220 | 236 | 250 | 250 |
| | | | Total: | 500 | - | - | - | - | 468 | 486 | 500 | 500 |

The column #Inst. contains the number of instances tested in each group. The average time and average log nodes include only instances for which an optimal solution was found. The notation "-" is used when the averages cannot be calculated.

## 3.6 Concluding remarks

We presented novel lower bounds and dominance rules for solving real-world sized premarshalling problems. Our approach extends the IBF[1] lower bound from the literature, as well as introduces two new lower bounds, extended and new dominance rules, and a feasible solution heuristic. Our lower bounds and dominance rules are computationally inexpensive, allowing them to be implemented as part of an iterative deepening branch-and-bound procedure. Our approach finds feasible solutions for nearly every CPMP instance it encounters, and optimal solutions on between 80.0% and 99.2% of the instances in industrially relevant instance categories.

# Chapter 4

# Minimizing the crane time in container premarshalling problems

## 4.1 Introduction

The objective of the container premarshalling problem (CPMP), studied in Chapters 2 and 3, is to minimize the number of container moves required to transform the initial layout of a bay into a final layout without blocking containers. The number of moves has traditionally been used as an indicator of the time employed by the crane to rearrange the bay. However, in many cases, the number of moves is not entirely representative of crane times (Lin et al. (2015); da Silva Firmino et al. (2019); Jovanovic et al. (2019b)). For example, it takes significantly more time to move a container between the farthest slots of a bay than to do so from the top level of a stack to the top level of an adjacent stack. Furthermore, such moves differ not only in the amount of time required by the crane, but also in terms of the energy consumption. According to Wilmsmeier and Spengler (2016), horizontal activities, such as those used by yard cranes, account for a total of 45% of the energy consumed in the terminal. Their findings regarding the current energy consumption of container terminals worldwide highlight the need for action to address competitiveness, energy security, and climate change. With so much of the terminal's energy being consumed by yard cranes, it becomes clear how important it is to reduce crane times in order to support green port policies.

We study a variant of the CPMP in which the goal is to minimize the time spent by the crane in transforming the bay into one without blocking containers. We refer to this problem as the premarshalling problem with crane time minimization objective, CPMPCT. We compute the crane time by considering whether the movement made is in the vertical or the horizontal axis and whether it is a loading or an unloading movement, since the speed and acceleration of the crane depend on these factors. We also consider twistlock[1] times, which are proportional to the tier where the container to be moved is located. With a precise estimation of the crane time, we carry out a computational study that motivates the importance of the problem and shows the non-direct relationship between the number of moves

[1]A twistlock and corner casting together form a standardized rotating connector for securing shipping containers that is used by the cranes for lifting the containers.

and the crane time. To solve the problem optimally, we propose two different approaches: a mathematical formulation, and a branch-and-bound-based approach. Moreover, we propose new upper and lower bounds for the number of moves and for the time spent by the crane in solving the CPMPCT, and we deal with dominance rules proposed in previous academic works, discussing whether or not they can be adapted to this new objective function. Our contributions are tested in an extensive computational analysis, using datasets from the literature, and the results show that instances of practical size can be solved to optimality.

We first review the relevant literature in Section 4.2. In Section 4.3, we formally describe the new problem of minimizing the crane times, showing that there is no direct relationship with the classic objective of minimizing the number of movements and explaining how time cranes can be calculated. Section 4.4 introduces new upper and lower bounds and dominance criteria that would be used in the exact approaches. The integer linear model is presented in Section 4.5, while the components of the branch-and-bound algorithm are described in Section 4.6. The computational results are provided in Section 4.7 and conclusions are discussed in Section 4.8.

## 4.2 Literature review

Literature addressing the container premarshalling problem is described in Section 2.2. All mentioned models and algorithms use the minimization of the number of moves required to rearrange the bay as the objective function. It is a simple and intuitive measure of the effort required, and allows for the development of high-quality bounds and dominance criteria. However, as will be shown in this chapter, already published in Parreño-Torres et al. (2020), the number of moves does not accurately represent the time or the cost needed for the premarshalling process. We therefore study the premarshalling with a crane time minimization objective and develop two exact approaches to solve it: an integer linear model, and a branch and bound algorithm, with new upper and lower bounds, dominance criteria, and a heuristic procedure to provide feasible solutions.

Even though the time required by the crane to perform moves had not been considered in the premarshalling problem prior to this thesis, several authors working on the closely related block relocation problem have considered the time required by the crane as the objective function. Lee and Lee (2010) use the weighted sum of the number of moves and the crane working time as their objective function. Lin et al. (2015) develop a heuristic algorithm to minimize the number of moves. They show that solutions with the minimum number of moves do not produce minimum working times, and the trade-off between these objectives can be controlled by adjusting the penalty values. Jovanovic et al. (2019b) develop an ant colony optimization algorithm. Their approach initially considers the standard minimization of the number of moves, but then adapts the same algorithm to the objective of minimizing the total crane working time. da Silva Firmino et al. (2019) only consider the crane working time as the objective of their reactive GRASP algorithm, integer linear model, and A* search algorithm. It should be noted that in all these studies crane times are calculated considering constant speeds for the crane trolley and spreader, distinguishing if the crane is loaded or unloaded. By contrast, in this chapter, we consider a more complex crane time computation that also considers crane acceleration and twistlock times. The crane time we model closely resembles real observed crane working times.

## 4.3 Premarshalling problem with crane time minimization objective

In this section we propose to minimize the time required by the crane to sort the bay, calculating the crane time precisely. In addition, we motivate the use of this new objective function by giving examples and computational results that show that decreasing the number of crane moves is not always linked to a decrease in crane time. Since the CPMP is a well known NP-Hard problem, the CPMPCT is also NP-Hard as it can be reduced to the CPMP considering unitary times for the crane moves.

### 4.3.1 Problem description

The problem of premarshalling with a crane time minimization objective CPMPCT seeks to obtain a sequence of moves that minimizes the time spent by the crane to transform the initial configuration of a bay into a final one in which no containers are blocking the retrieval of others. The assumptions of the problem are the same as those of the CPMP: single bay assumption, uniform move cost assumption, single crane assumption, and full information assumption.

We use again the notation of previous chapters to describe the set of stacks of the bay $\mathcal{S}$, where $S = |\mathcal{S}|$ is the total number of stacks, the set of tiers $\mathcal{H}$, where $H = |\mathcal{H}|$ represents the highest tier of the stacks, and the set of priorities $\mathcal{P}$, where 1 is the highest priority and $P = |\mathcal{P}|$ the lowest. Moves are defined not only by indicating the origin and destination stacks but also the tiers involved. Therefore, a solution is a sequence of moves of the form $(s, h, k, e) : s \neq k \in \mathcal{S}$ and $h, e \in \mathcal{H}$ where $(s, h)$ is the origin position of the container being moved and $(k, e)$ its destination position. Given a solution $sol = (s_1, h_1, k_1, e_1) \dots (s_i, h_i, k_i, e_i) \dots (s_n, h_n, k_n, e_n)$, the time spent by the crane is denoted as $time(sol)$. We include the time spent by the crane to position the spreader from its initial position outside of the bay to the position of the first container to be moved. We also distinguish the movements carried out by the crane with and without a loaded container. The crane moves unloaded from the destination of the last container moved to the position of the next container to be moved. The crane then picks up the next container to be moved, and moves loaded to the container's destination. The function $p_i(s, h)$ gives the priority of the container stored at position $(s, h)$ after the $i$-th move. The bay is considered ordered after move $n$ if $p_n(s, h) \geq p_n(s, h + 1)$ for all $s \in \mathcal{S}$ and $h \in \mathcal{H} \setminus \{H\}$.

### 4.3.2 Parameters used to compute the time required by the crane

There are several types of cranes used to handle containers in port yards, each with different technical specifications. In this work we consider the rubber tired gantry (RTG) cranes used in the Noatum terminal of Valencia's port in Spain. The company Noatum facilitated this information during their collaboration in the EU Project Transforming Transport (TT, Grant agreement No. 731932). The information corresponds to an RTG Transtainer 79 from Konecranes. To move a container, the crane is first positioned above it, moving horizontally along the top travel lane to the corresponding stack and then moving down to reach the requested position. Then the container is twistlocked, hoisted up, moved horizontally to the destination stack and hoisted down to its new position. This is a common operation that can be observed in most gantry cranes.

Figure 4.1 shows the structure of the RTG crane. The values of the parameters represented in the figure are shown in Table 4.1. Note that in this work we assume (w.l.o.g.) that all containers in the bay have the same dimensions.

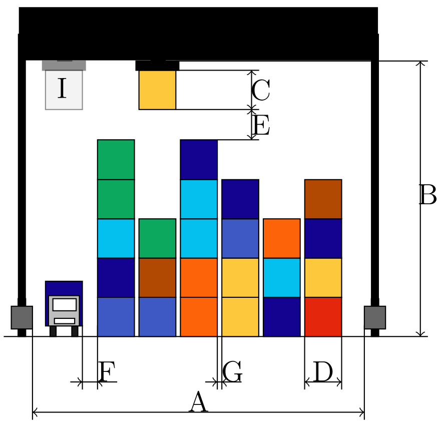


Figure 4.1: Front view of an RTG crane.

Table 4.1: Parameters referring to storage distances represented in Figure 4.1.

| Letter | Notation | Description | Value |
|---|---|---|---|
| A | - | Useful distance between legs [m] | 21.925 |
| B | - | Lifting height under spreader [m] | 18.200 |
| C | *ch* | Container height [m] | 2.591 |
| D | *cw* | Container width [m] | 2.438 |
| E | *vsep* | Distance between the top level and the container at the topmost tier [m] | 2.000 |
| F | *hsep* | Distance between first stack and the truck (or the initial point axis x) [m] | 1.000 |
| G | *msep* | Distance between containers [m] | 0.300 |

The data provided by the company leads to a differentiation between the time spent by the crane according to whether it performs a loaded vertical move ($vmax^{vl}$), an unloaded vertical move ($vmax^{vu}$), a loaded horizontal move ($vmax^{rl}$) or an unloaded horizontal move ($vmax^{ru}$). The maximum speeds of the crane for each of these cases and the horizontal and vertical distance the crane has to travel to reach them can be seen in Table 4.2. Furthermore, this table shows the twistlock time, which is directly proportional to the level at which the container to move is placed. This is due to the oscillation of the crane spreader, which depends on the cable's length, due to the sway motion of the suspended load. Crane operators have to wait until the oscillation stops to twistlock the container. Some cranes employ anti-sway systems that arrest the pendulums and the rotational sway motion, resulting in reduced twistlock times, but this is not the case for the studied cranes. We suppose that when the crane moves horizontally or vertically it requires $d^{\alpha}$ meters to reach its maximum speed. Up to that distance, the acceleration is constant and therefore depicts a uniformly accelerated rectilinear motion (u.a.r.m). Once it reaches the maximum speed, the crane follows a uniformly rectilinear motion (u.r.m) and starts to decelerate when there are $d^{\alpha}$ meters left until its destination point. If the distance travelled by the crane is less than $2d^{\alpha}$, half the distance is accelerating and the other half is decelerating.

Table 4.2: Parameters employed to compute the crane costs provided by the company.

| Notation | Description | Value |
|---|---|---|
| $vmax^{vl}$ | Hoisting speed loaded [m/s] | 0.5 |
| $vmax^{vu}$ | Hoisting speed unloaded [m/s] | 1.00 |
| $vmax^{rl}$ | Travelling speed loaded [m/s] | $1.1\bar{6}$ |
| $vmax^{ru}$ | Travelling speed unloaded [m/s] | $2.1\bar{6}$ |
| $d^{v}$ | Distance to max. hoisting speed [m] | 2.65 |
| $d^{r}$ | Distance to max. travelling speed [m] | 2.50 |
| $b_h$ | Twistlock time at level $h$ [s] | 5·(H-h+1) |

Once the maximum speeds of the crane and the distances required to reach them have been set, equation (4.1) calculates the acceleration $[m/s^2]$ for the moves considered. It is obtained by using the equations of the u.a.r.m.

$$a^{\alpha\beta} = \frac{(vmax^{\alpha\beta})^2}{2d^{\alpha}} \qquad \forall \alpha \in \{v, r\}; \quad \forall \beta \in \{l, u\} \tag{4.1}$$

Let $x$ be the distance travelled by the crane (vertically or horizontally) in meters. The time spent by the crane (loaded or unloaded) in seconds to carry out the move can be calculated according to equation (4.2). If $x < 2d^{\alpha}$, the time spent by the crane to reach $x/2$ meters equals $\sqrt{x/a^{\alpha\beta}}$ according to the second equation of the u.a.r.m. Since we consider deceleration and acceleration to be opposite terms, the time spent by the crane to travel $x$ is $t^{\alpha\beta}(x) = 2\sqrt{x/a^{\alpha\beta}} = (2\sqrt{a^{\alpha\beta}x})/a^{\alpha\beta}$. If $x \geq 2d^{\alpha}$, the first $d^{\alpha}$ meters in u.a.r.m require $\sqrt{2d^{\alpha}/a^{\alpha\beta}}$ seconds (second equation of u.a.r.m). The crane then follows u.r.m during $x - 2d^{\alpha}$ meters taking $(x - 2d^{\alpha})/vmax^{\alpha\beta}$ seconds. The last $d^{\alpha}$ meters it is decelerating, which is again u.a.r.m with an associated time equal to $\sqrt{2d^{\alpha}/a^{\alpha\beta}}$. Therefore, the total time spent by the crane to perform the move is $t^{\alpha\beta}(x) = 2\sqrt{2d^{\alpha}/a^{\alpha\beta}} + (x - 2d^{\alpha})/vmax^{\alpha\beta} = (2vmax^{\alpha\beta})/a^{\alpha\beta} + (x - 2d^{\alpha})/vmax^{\alpha\beta}$.

$$t^{\alpha\beta}(x) = \begin{cases} \frac{2\sqrt{a^{\alpha\beta}x}}{a^{\alpha\beta}}, & \text{If } x < 2d^{\alpha} \\ \frac{2vmax^{\alpha\beta}}{a^{\alpha\beta}} + \frac{x-2d^{\alpha}}{vmax^{\alpha\beta}}, & \text{If } x \geq 2d^{\alpha} \end{cases} \qquad \forall \alpha \in \{v, r\}; \quad \forall \beta \in \{l, u\} \tag{4.2}$$

With reference to the distances described in Table 4.1, the distance in meters between the top travel lane and level $h$, $distance^{v}(h)$, follows equation (4.3). Furthermore, the distance in meters between stacks $s$ and $k$, $distance^{r}(s, k)$, follows equation (4.4). If $k = 0$, the crane is moved from (or to) the initial point, represented as I in Figure 4.1, to (or from) stack $s$.

$$distance^v(h) = vsep + (H - h + 1) \cdot ch \tag{4.3}$$

$$distance^r(s,k) = \begin{cases} |k - s| \cdot (msep + cw), & \text{If } k \neq 0 \\ (s-1) \cdot msep + s \cdot cw + hsep, & \text{If } k = 0 \end{cases} \tag{4.4}$$

Note that if $s < S$, $distance^r(s, s+p) = distance^r(1, p+1)$ for all $p \in \mathbb{N}$ such that $p < S - s$. Otherwise, $distance^r(s, s-p) = distance^r(1, p+1)$ for all $p \in \mathbb{N}$ such that $p \leq S$. With all parameters adjusted, we then calculate the total time spent in the rearrangement of the bay following the sequence $sol = (s_1, h_1, k_1, e_1) \dots (s_i, h_i, k_i, e_i) \dots (s_n, h_n, k_n, e_n)$.

$$time(sol) = t(s_1, h_1, k_1, e_1) + \cdots + t(s_i, h_i, k_i, e_i) + \cdots + t(s_n, h_n, k_n, e_n) \tag{4.5}$$

where:

$$t(s_i, h_i, k_i, e_i) = \begin{cases} c^0_{k_{i-1}s_i} + v^0_{h_i} + b_{h_i} + v^1_{h_i} + c^1_{s_ik_i} + v^1_{e_i} + v^0_{e_i}, & \text{If } i \neq 0 \\ c^0_{s_i0} + v^0_{h_i} + b_{h_i} + v^1_{h_i} + c^1_{s_ik_i} + v^1_{e_i} + v^0_{e_i} & \text{If } i = 0 \end{cases} \tag{4.6}$$

$$\begin{aligned} v^0_h &= t^{vu}\Big(distance^v(h)\Big); \quad v^1_h = t^{vl}\Big(distance^v(h)\Big); \\ c^0_{sk} &= t^{ru}\Big(distance^r(s,k)\Big); \quad c^1_{sk} = t^{rl}\Big(distance^r(s,k)\Big); \end{aligned} \tag{4.7}$$

For each move $(s_i, h_i, k_i, e_i)$ we consider the time spent to move the crane (unloaded) along the upper travel line from its current position to the origin stack of the next container to be moved $c^0_{k_{i-1}s_i}$ (or $c^0_{s_i0}$ if the crane is originally at the initial point), the time spent to move the crane down to reach the container $v^0_{h_i}$, the twistlock time $b_{h_i}$, the time spent to hoist the container up $v^1_{h_i}$, the time spent to move the crane (loaded) along the upper travel line to the destination stack $c^1_{s_ik_i}$, the time spent to move the crane down to release the container $v^1_{e_i}$, and the time spent to position the crane on the upper travel line $v^0_{e_i}$.

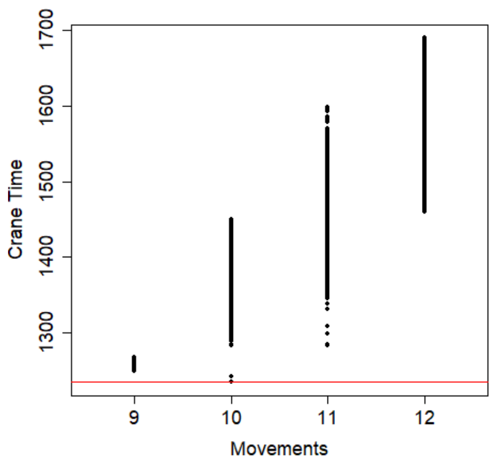
Hence, a lower bound for the time spent to perform a move $t_{min}$ is:

$$t_{min} = c^0_{12} + v^0_H + b_H + v^1_H + c^1_{12} + v^1_H + v^0_H = (2v^0_H + c^0_{12}) + (2v^1_H + b_H + c^1_{12}) \tag{4.8}$$

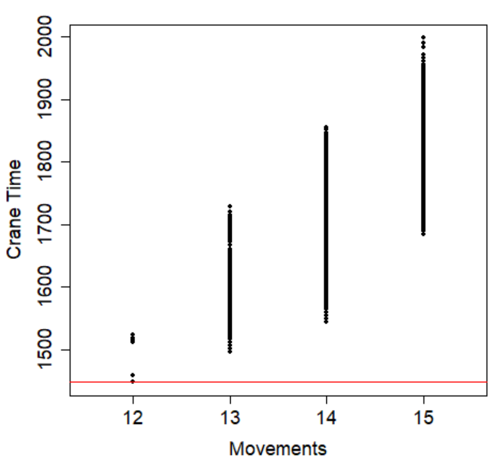
in which the crane moves to an adjacent stack, picks up a container at the topmost tier, and moves it to the topmost tier of an adjacent stack.

### 4.3.3 Crane time against number of movements

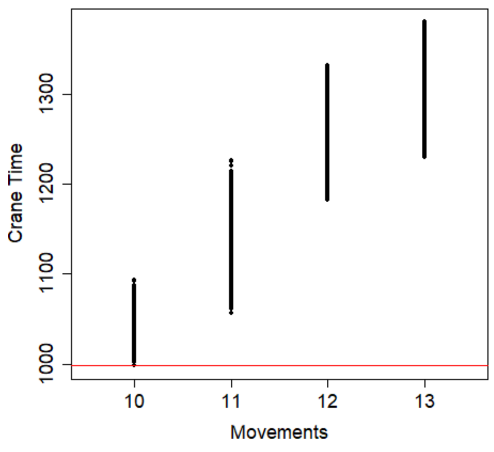
We now study the relation between the number of moves required to rearrange the bay and the time taken by the crane to do it. Figure 4.2 shows the information of a large set of solutions of four instances belonging to datasets frequently studied in the literature. The $x$-axis describes the number of movements to solve the CPMP, with the minimum value being the optimal solution. The $y$-axis represents the total crane time measured in seconds. All the alternate optimal solutions for the CPMP are represented, whereas for a greater number of movements only a set of solutions is represented. It can be seen that the crane time can increase up to 24% over two feasible solutions with the same number of moves. It can also be seen that an optimal solution for the CPMP can have a crane time higher than a non-optimal solution. Even though the examples represented are small, notice that the variation in the crane time between solutions with the same number of moves increases with the number of moves. The best solution found for the CPMPCT, not necessarily the optimal one for this problem, is the optimal one for the CPMP in the cases shown in Figure 4.2b and 4.2c. However, in Figures 4.2a and 4.2d the best solution found for the CPMPCT is not the optimal one for the CPMP.



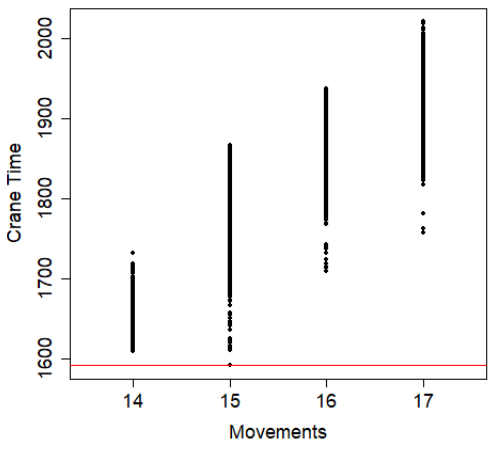
(a) BZ: data_p6_s5_h6_f50_2.txt



(b) CV: data4-5-36.txt



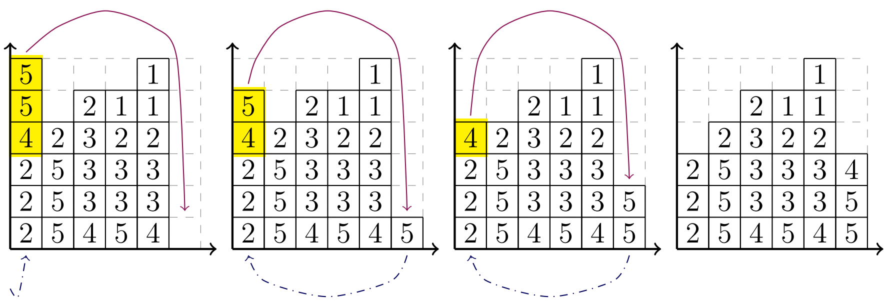
(c) EMM: emm_s7_t4_p0.75_c1_21.txt



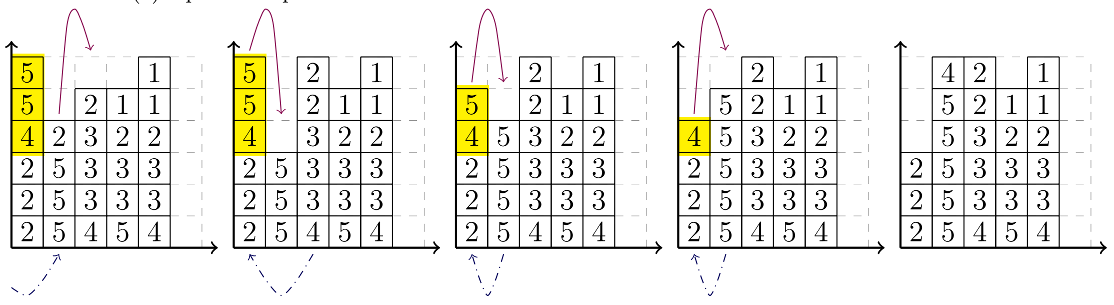
(d) ZJY: 6_5_7.txt

Figure 4.2: Comparison of crane times and number of movements on instances from datasets studied in the state-of-the-art. The horizontal line in red indicates the shortest crane time shown in the figure.

Figure 4.3 illustrates an example in which the minimum number of moves to solve the CPMP is 3, with a crane time of 393.7 seconds (Figure 4.3a). However, the number of moves to optimally solve the CPMPCT is 4, with a crane time of 386.5 seconds (Figure 4.3b).



(a) Optimal sequence of moves to solve the CPMP. The crane time is 393.696 seconds.



(b) Optimal sequence of moves to solve the CPMPCT.The crane time is 386.534 seconds.

Figure 4.3: An example layout in which the optimal sequence of moves to solve the CPMPCT has a larger number of moves than the optimal sequence to solve the CPMP. Blocking containers are shown in yellow and arcs represent the movements of the crane loaded (continuous lines) and unloaded (dashed lines).

## 4.4 Bounds and dominance criteria

This section proposes upper bounds for the number of moves, lower bounds for the time spent by the crane, as well as dominance rules to be used in the models and algorithms developed in this chapter for the CPMPCT.

In the following proofs, let *sol* be a feasible solution for the CPMP and $time(sol)$ be the time spent by the crane to carry out the solution. We denote as $\gamma$ the number of blocking containers in the initial layout of the bay and the $\eta_1, \dots, \eta_\gamma$ tiers where they are initially stored.

### 4.4.1 Upper bound for the number of moves to solve the CPMPCT

Establishing an upper bound on the number of moves to rearrange the bay is not easy in the classic CPMP, but it can be done using an iterative procedure, progressively increasing the number of moves until a feasible solution is found (Parreño-Torres et al., 2019b). When we solve the CPMPCT, we cannot determine this upper bound by a similar iterative procedure because it is possible to get better solutions by increasing the number of moves. Nor can we set the bound as the number of moves of any solution to the CPMP, as the optimal solution may have a greater number of moves. However, when using mathematical models for solving the problem, it is necessary to have an upper bound on the number of movements to ensure that we are solving the problem optimally.

**Proposition 12.** *The optimal number of moves to solve the CPMPCT is bounded above by* $\lfloor \frac{time(sol)}{t_{min}} \rfloor$.

*Proof.* Let $sol^* = (s_1, h_1, k_1, e_1) \dots (s_i, h_i, k_i, e_i) \dots (s_n, h_n, k_n, e_n)$ be an optimal solution for the CPMPCT. We compute the total time spent by the crane as follows:

$$\begin{aligned} time(sol^*) &= t(s_1, h_1, k_1, e_1) + \dots + t(s_i, h_i, k_i, e_i) + \dots + t(s_n, h_n, k_n, e_n) \\ &\geq t_{min} + \dots + t_{min} + \dots + t_{min} \geq n \cdot t_{min} \end{aligned}$$

As $sol^*$ is the optimal solution to solve the CPMPCT and $t_{min} > 0$,

$$time(sol) \geq time(sol^*) \geq n \cdot t_{min} \iff \frac{time(sol)}{t_{min}} \geq n$$

Since $n \in \mathbb{N}$, $n \leq \lfloor \frac{time(sol)}{t_{min}} \rfloor$.

□

**Proposition 13.** *The optimal number of moves to solve the CPMPCT is bounded above by:*

$$\left\lfloor \frac{time(sol) - \sum_{i=1}^{\gamma} \left( c_{12}^0 + v_{\eta_i}^0 + b_{\eta_i} + v_{\eta_i}^1 + c_{12}^1 + v_H^1 + v_H^0 \right) + (\gamma \cdot t_{min})}{t_{min}} \right\rfloor$$

*Proof.* In order to arrange the bay, all the blocking containers in the initial layout must be moved at least once. Let $sol^* = (s_1, h_1, k_1, e_1) \dots (s_i, h_i, k_i, e_i) \dots (s_n, h_n, k_n, e_n)$ be an optimal solution of the CPMPCT. The blocking containers should be moved from their positions so the spreader has to go down to their initial tier and the following inequality is satisfied:

$$time(sol) \geq time(sol^*) \geq (n - \gamma) \cdot t_{min} + \sum_{i=1}^{\gamma} \left( c_{12}^0 + v_{\eta_i}^0 + b_{\eta_i} + v_{\eta_i}^1 + c_{12}^1 + v_H^1 + v_H^0 \right)$$

Since $n \in \mathbb{N}$, we reformulate the inequality above to get:

$$\left\lfloor \frac{time(sol) - \sum_{i=1}^{\gamma} \left( c_{12}^0 + v_{\eta_i}^0 + b_{\eta_i} + v_{\eta_i}^1 + c_{12}^1 + v_H^1 + v_H^0 \right) + (\gamma \cdot t_{min})}{t_{min}} \right\rfloor \geq n$$

□

With these proofs, we can take advantage of the many heuristic and exact procedures developed for the CPMP to obtain feasible solutions in very short times.

### 4.4.2 Lower bound for the time spent by the crane

Let $LB := IBF^4$ be the lower bound for the number of moves described in Tanaka et al. (2019). We use this bound to propose valid lower bounds for the total time spent by the crane.

**Proposition 14.** $LB_{ct}^0 := LB \cdot t_{min}$ *is a valid lower bound for the CPMPCT.*

*Proof.* If $LB$ is the smallest number of moves to sort the bay, the minimum time spent by the crane is that number multiplied by the minimum time spent to perform a single move. □

**Proposition 15.** *A valid lower bound for the CPMPCT is:*

$$LB^1_{ct} := (LB - \gamma) \cdot t_{min} + \sum_{i=1}^{\gamma} \left( c^0_{12} + v^0_{\eta_i} + b_{\eta_i} + v^1_{\eta_i} + c^1_{12} + v^1_H + v^0_H \right) \tag{4.9}$$

*Proof.* To sort the bay in the required order, all the blocking containers must be moved at least once, so the crane has to go down to tier $\eta_i$ to move blocking container $i$. Therefore, the minimum time to move container $i$ is $\left( c^0_{12} + v^0_{\eta_i} + b_{\eta_i} + v^1_{\eta_i} + c^1_{12} + v^1_H + v^0_H \right)$, which is greater than (or equal to) $t_{min}$. □

Let $\tau_1, \ldots, \tau_\gamma$ be the $\gamma$ highest tiers to which the $\gamma$ blocking containers in the initial layout of the bay could be moved on top of non blocking containers without considering blocking containers. See Figure 4.4 for further details.

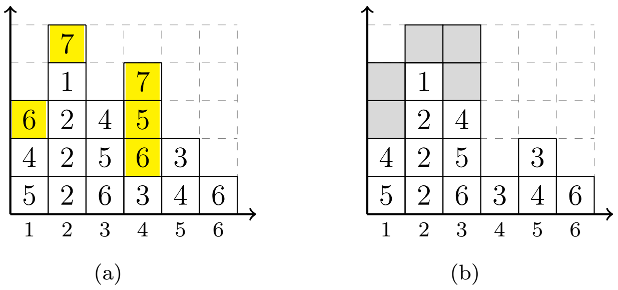


Figure 4.4: An example showing a layout with 5 blocking containers and a selection of the 5 highest tiers to which the blocking containers could be moved. Subfigure (a) shows the initial layout in which the blocking containers are marked in yellow. In Subfigure (b), only the non blocking containers are represented and a selection of the highest tiers to which they can be moved are highlighted in gray.

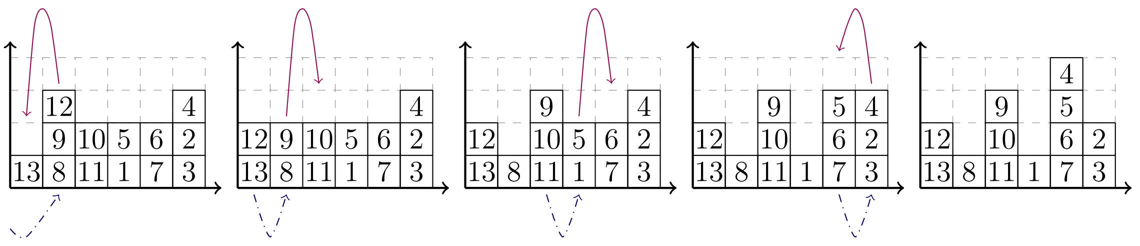
**Proposition 16.** *A valid lower bound for the CPMPCT is:*

$$LB^2_{ct} := (LB - \gamma) \cdot t_{min} + \sum_{i=1}^{\gamma} \left( c^0_{12} + v^0_{\eta_i} + b_{\eta_i} + v^1_{\eta_i} + c^1_{12} + v^1_{\tau_i} + v^0_{\tau_i} \right) \tag{4.10}$$

*Proof.* Starting from the initial layout of the bay, all blocking containers must be moved and the highest tiers to which they can be moved are $\tau_1, \ldots, \tau_\gamma$. Suppose that we can move some blocking container to a higher position, this happen in the case that a non blocking container has been placed in one of the levels $\tau_1, \ldots, \tau_\gamma$ and therefore this vertical distance has already been accounted for, so the bound holds. □

### 4.4.3 Dominance criteria

Dominance rules for the CPMP have been highly studied in the literature (Tanaka et al. (2019); Tierney et al. (2017); Tanaka and Tierney (2018)) and have shown their effectiveness in reducing the number of nodes to be explored in branch and bound algorithms. Nevertheless, when we study the CPMPCT we cannot consider the vast majority of them. For example, empty stack rules stating that if there is more than one empty stack only the move to the left-most stack is considered are not satisfied here because all empty stacks are not equivalent in this problem. Nor could we consider unrelated move symmetries because even in the simplest case in which two consecutive moves have different origin and destination stack the order of the movements is relevant as can be seen in Figure 4.5. Two sequences of 4 moves, leading to the same configuration are not equivalent in terms of crane times.


(a) The time spent by the crane to perform the sequence (2,1)(2,3)(4,5)(6,5) is 405.813 seconds.

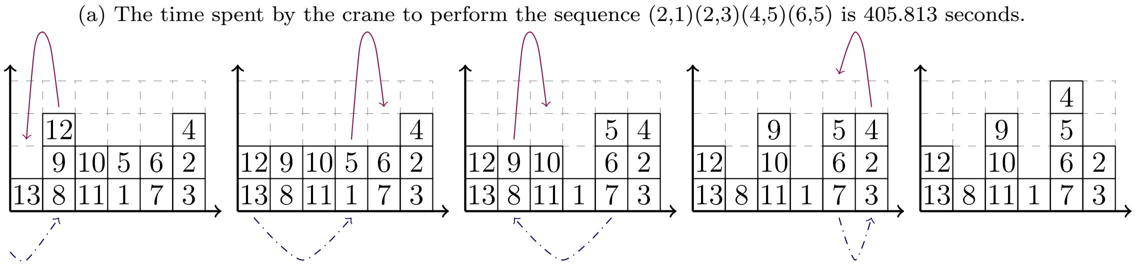

(b) The time spent by the crane to perform the sequence (2,1)(4,5)(2,3)(6,5) is 413.864 seconds.

Figure 4.5: An example showing two sequences considered as unrelated move symmetries in the CPMP whose crane time is not the same.

Eliminating same group symmetries, which involve the relocation of two containers with the same priority, is highly relevant in the context of the CPMP. However, in the CPMPCT, two movements do not take the same crane time if the origin or/and destination stack is modified, so most of these rules are not satisfied. In Figure 4.6 two sequences of 3 moves, involving two containers of priority 5 lead to the same configuration but with different times. The only same group rule that we can apply to the CPMPCT states that if a container of priority $p$ is moved from one stack $s$, no container of priority $p$ will be moved to that stack in the next movement.

**Proposition 17.** *Let* $t1 = c^1_{s_ik_i} + c^0_{k_is_{i+1}} + c^1_{s_{i+1}s_i}$ *and* $t2 = c^0_{s_is_{i+1}} + c^1_{s_{i+1}k_i} + c^0_{k_is_i}$*, then* $t1 > t2$.

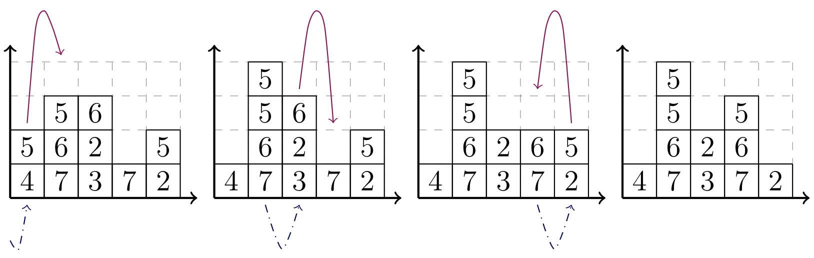
*Proof.* Let $t1$ and $t2$ be:

$$t1 = c^1_{s_ik_i} + c^0_{k_is_{i+1}} + c^1_{s_{i+1}s_i} = t^{rl}\Big(distance^r(s_i,k_i)\Big) + t^{ru}\Big(distance^r(k_i,s_{i+1})\Big) + t^{rl}\Big(distance^r(s_{i+1},s_i)\Big)$$
$$t2 = c^0_{s_is_{i+1}} + c^1_{s_{i+1}k_i} + c^0_{k_is_i} = t^{rl}\Big(distance^r(s_i,s_{i+1})\Big) + t^{ru}\Big(distance^r(s_{i+1},k_i)\Big) + t^{rl}\Big(distance^r(k_i,s_i)\Big)$$

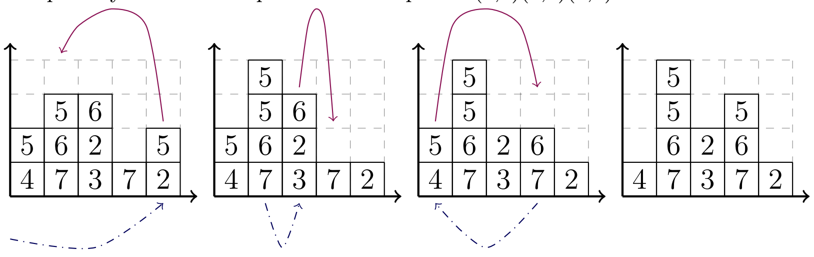
To address this proof, we rewrite equation (4.2) as a function of the distance travelled $x$, the maximum speed reached $vmax^{r\beta}$, and the distance to achieve the maximum travelling speed $d^r$. Note that acceleration relies on the last two factors above.

$$t^{r\beta}(x) = \begin{cases} \frac{2\sqrt{2d^rx}}{vmax^{r\beta}}, & \text{If } x < 2d^\alpha \\ \frac{x+2d^r}{vmax^{r\beta}}, & \text{If } x \geq 2d^\alpha \end{cases} \qquad \forall\beta \in \{l,u\}$$

Therefore, $t1$ and $t2$ depend on the distances among $s_i$, $s_{i+1}$, and $k_i$, which depend on their relative position in the bay. If we sort 3 stacks in non-decreasing order, we get six cases. Nevertheless we just have to consider three of them because $distance^r(s,k) = distance^r(k,s)$:



(a) The time spent by the crane to perform the sequence (1,2)(3,4)(5,4) is 296.9831 seconds.



(b) The time spent by the crane to perform the sequence (5,2)(3,4)(1,4) is 314.755 seconds.

Figure 4.6: An example showing two sequences considered as same group symmetries in the CPMP and whose crane time is not the same.

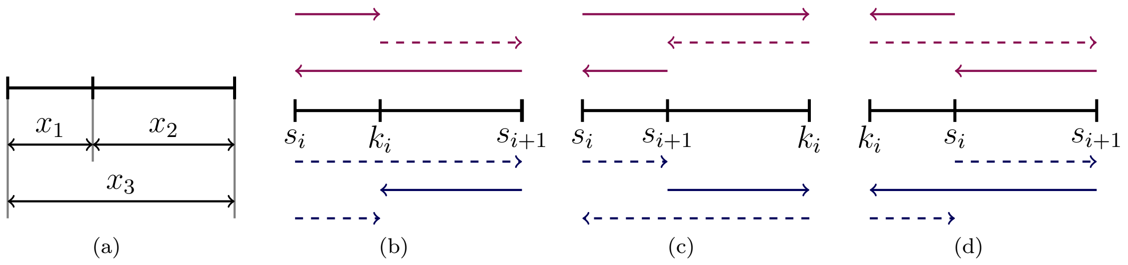


Figure 4.7: Given three stacks at distances $x_1$, $x_2$, and $x_3$ (Figure 4.7a), Figures 4.7b to 4.7d show the different relative positions that must be addressed.

**Case 1.** $s_i < k_i < s_{i+1}$, Figure 4.7b.

**Case 2.** $s_i < s_{i+1} < k_i$, Figure 4.7c.

**Case 3.** $k_i < s_i < s_{i+1}$, Figure 4.7d.

In Subfigures 4.7b to 4.7d, the lines above and below the segment describe the relative position between the stacks. Solid lines represent the movements of the crane loaded and dashed lines show the crane unloaded. The time spent to perform the movements represented above the segment is $t1$ and below the segment is $t2$.

The difference $t1 - t2$ can be described as a function of three terms $t1 - t2 = \frac{term_1 term_3}{term_2}$, where $term_1 = (vmax^{ru} - vmax^{rl})$, $term_2 = vmax^{ru} vmax^{rl}$, and $term_3$ changes according to the case discussed. We study the different cases, proving that $t1 - t2 > 0 \leftrightarrow t1 > t2$. Since $vmax^{ru} > vmax^{rl}$ and the speeds are positive values, $term_1$ and $term_2$ are greater than 0, thus we just need to prove that $term_3$ is greater than (or equal to) 0.

**Case 1.** $s_i < k_i < s_{i+1}$, Subfigure 4.7b.

We consider $x_1 = distance^r(s_i, k_i)$, $x_2 = distance^r(k_i, s_{i+1})$, and $x_3 = distance^r(s_i, s_{i+1})$, so $t1 = t^{rl}(x_1) + t^{ru}(x_2) + t^{rl}(x_3)$ and $t2 = t^{ru}(x_3) + t^{rl}(x_2) + t^{ru}(x_1)$. Five scenarios should be discussed:

S1: $x_1, x_2, x_3 < 2d^r$

$$term_3 = 2\sqrt{2d^r x_1} + 2\sqrt{2d^r x_3} - 2\sqrt{2d^r x_2} > 2\sqrt{2d^r x_1} > 0$$

The first inequality is satisfied because $x_3 > x_2$, so $2\sqrt{2d^r x_3} > 2\sqrt{2d^r x_2}$ and the second one because the distance $x_1$ is positive ($s_i \neq k_i$).

S2: $x_1, x_2 < 2d^r$ and $x_3 \geq 2d^r$

$$term_3 = 2\sqrt{2d^r x_1} + x_3 + 2d^r - 2\sqrt{2d^r x_2}) > 2\sqrt{2d^r x_1} + x_3 - 2d^r > 2\sqrt{2d^r x_1} > 0$$

Since $x_2 < 2d^r$, then $2d^r x_2 < 4(d^r)^2 \rightarrow 2\sqrt{2d^r x_2} < 4d^r$ and the first inequality is satisfied. The second and third one follows from the fact that $x_3 \geq 2d^r$ and that $x_1 > 0$.

S3: $x_1 < 2d^r$ and $x_2, x_3 \geq 2d^r$

Distance $x_3$ is greater than $x_2$ and $x_1 > 0$, so $term_3 = (2\sqrt{2d^r x_1} + x_3 - x_2) > 2\sqrt{2d^r x_1} > 0$

S4: $x_2 < 2d^r$ and $x_1, x_3 \geq 2d^r$

As $x_2 < 2d^r$, then $2\sqrt{2d^r x_2} < 4d^r$. Moreover the distances are positive because $s_i \neq k_i \neq s_{i+1}$, thus $term_3 = (x_1 + x_3 + 4d^r - 2\sqrt{2d^r x_2}) > x_1 + x_3 > 0$.

S5: $x_1, x_2, x_3 \geq 2d^r$

The distance $x_3$ is greater than $x_2$ and all distances are greater than 0, so the inequality $term_3 = (x_1 + x_3 + 2d^r - x_2) > 0$ is fulfilled.

Therefore, given three stacks $s_i, k_i, s_{i+1}$ such as $s_i < k_i < s_{i+1}$, the inequality $t1 > t2$ is always satisfied.

**Case 2.** $s_i < s_{i+1} < k_i$, Subfigure 4.7c. Analogous proof to that in the previous case.

**Case 3.** $k_i < s_i < s_{i+1}$, Subfigure 4.7d.

In this case, $x_1 = distance^r(s_i, k_i)$, $x_2 = distance^r(s_i, s_{i+1})$, and $x_3 = distance^r(k_i, s_{i+1})$, therefore $t1 = t^{rl}(x_1) + t^{ru}(x_3) + t^{rl}(x_2)$ and $t2 = t^{ru}(x_2) + t^{rl}(x_3) + t^{ru}(x_1)$.

S1: $x_1, x_2, x_3 < 2d^r$

Since $x_3 = x_1 + x_2$,

$$\begin{aligned} term_3 &= 2\sqrt{2d^r x_1} + 2\sqrt{2d^r x_2} - 2\sqrt{2d^r x_3} = 2\sqrt{2d^r}\left(\sqrt{x_1} + \sqrt{x_2} - \sqrt{x_3}\right) \\ &= 2\sqrt{2d^r}\left(\sqrt{x_1} + \sqrt{x_2} - \sqrt{x_1 + x_2}\right) \end{aligned}$$

The term $2\sqrt{2d^r}$ is greater than 0 and $\sqrt{x_1} + \sqrt{x_2} \geq \sqrt{x_1 + x_2}$. Therefore $term_3 > 0$.

S2: $x_1, x_2 < 2d^r$ and $x_3 \geq 2d^r$

Since $2d^r > x_1 \rightarrow \sqrt{2d^r x_1} > x_1$, $2d^r > x_2 \rightarrow \sqrt{2d^r x_2} > x_2$, $x_3 = x_1 + x_2$, and $x_3 \geq 2d^r$, the following inequalites are satisfied:

$$term_3 = 2\sqrt{2d^r x_1} + 2\sqrt{2d^r x_2} - x_3 - 2d^r > 2(x_1 + x_2) - x_3 - 2d^r = x_3 - 2d^r \geq 0$$

S3: $x_1 < 2d^r$ and $x_2, x_3 \geq 2d^r$

Since $x_3 = x_1 + x_2$, and $2d^r > x_1 \rightarrow \sqrt{2d^r x_1} > x_1$,

$$term_3 = (2\sqrt{2d^r x_1} + x_2 - x_3) = (2\sqrt{2d^r x_1} - x_1) > 0.$$

S4: $x_2 < 2d^r$ and $x_1, x_3 \geq 2d^r$. Proof analogous to the previous scenario.

S5: $x_1, x_2, x_3 \geq 2d^r$

The last scenario follows from $x_3 = x_1 + x_2$ and $2d^r > 0$, then $term3 = x_1 + x_2 + 2d^r - x_3 = 2d^r > 0$.

Therefore, given three stacks $s_i, k_i, s_{i+1}$ such as $k_i < s_i < s_{i+1}$, the inequality $t1 > t2$ is always satisfied.

$\square$

**Proposition 18.** *A sequence* $sol = (s_1, h_1, k_1, e_1) \dots (s_i, h_i, k_i, e_i)(s_{i+1}, h_{i+1}, s_i, h_i) \dots (s_n, h_n, k_n, e_n)$ *is disallowed for the CPMPCT if* $p_{i-1}(s_i, h_i) = p_{i+1}(s_i, h_i)$.

*Proof.* The same layout as in $sol$ is obtained by the sequence $sol_1 = (s_1, h_1, k_1, e_1) \dots (s_{i+1}, h_{i+1}, k_i, e_i) \dots (s_n, h_n, k_n, e_n)$. Moreover the time spent by the crane to carry out *sol₁* is shorter than that spent to carry out $sol$:

$$\begin{aligned}
time(sol) = {} & t(s_1, h_1, k_1, e_1) + \dots + \left(c^0_{k_{i-1}s_i} + c^1_{s_ik_i} + c^0_{k_is_{i+1}} + c^1_{s_{i+1}s_i}\right) + \Big(v^0_{h_i} + b_{h_i} + v^1_{h_i} \\
& + v^1_{e_i} + v^0_{e_i} + v^0_{h_{i+1}} + b_{h_{i+1}} + v^1_{h_{i+1}} + v^1_{h_i} + v^0_{h_i}\Big) + \dots + t(s_n, h_n, k_n, e_n) \\
> {} & t(s_1, h_1, k_1, e_1) + \dots + \left(c^0_{k_{i-1}s_i} + c^0_{s_is_{i+1}} + c^1_{s_{i+1}k_i} + c^0_{k_is_i}\right) + \Big(v^1_{e_i} + v^0_{e_i} \\
& + v^0_{h_{i+1}} + b_{h_{i+1}} + v^1_{h_{i+1}}\Big) + \dots + t(s_n, h_n, k_n, e_n) \\
> {} & t(s_1, h_1, k_1, e_1) + \dots + \left(c^0_{k_{i-1}s_{i+1}} + c^1_{s_{i+1}k_i}\right) + \left(v^1_{e_i} + v^0_{e_i} + v^0_{h_{i+1}} + b_{h_{i+1}} + v^1_{h_{i+1}}\right) \\
& + \dots + t(s_n, h_n, k_n, e_n) = time(sol_1)
\end{aligned}$$

The first inequality is satisfied by Proposition 17 and the fact that the times are always positive. The second inequality is satisfied because $(c^0_{k_{i-1}s_i} + c^0_{s_is_{i+1}}) \leq c^0_{k_{i-1}s_{i+1}}$ and $(c^0_{k_is_i} + c^0_{s_is_{i+2}}) \leq c^0_{k_is_{i+2}}$.

$\square$

Another dominance rule from the CPMP that applies is the direct transitive rule. This rule refers to the movement of a container from position $(s_i, h_i)$ to $(k_i, e_i)$ and then from $(k_i, e_i)$ to another position $(k_{i+1}, e_{i+1})$ in two moves rather than in a single move from $(s_i, h_i)$ to $(k_{i+1}, e_{i+1})$.

**Proposition 19.** *A sequence* $sol = (s_1, h_1, k_1, e_1) \dots (s_i, h_i, k_i, e_i)(k_i, e_i, k_{i+1}, e_{i+1}) \dots (s_n, h_n, k_n, e_n)$ *is disallowed for the CPMPCT.*

*Proof.* Since the sequence $(s_i, h_i, k_i, e_i)(k_i, e_i, k_{i+1}, e_{i+1})$ and $(s_i, h_i, k_{i+1}, e_{i+1})$ start and end in the same stacks and slot $(k_i, e_i)$ is only used for temporary storage of the container, the same layout as in $sol$ is obtained by the sequence $sol_1 = (s_1, h_1, k_1, e_1) \dots (s_i, h_i, k_{i+1}, e_{i+1}) \dots$

$(s_n, h_n, k_n, e_n)$ with a shorter crane time:

$$\begin{aligned}
\mathit{time}(sol) &= t(s_1,h_1,k_1,e_1) + \dots + \Big(c^0_{k_{i-1}s_i} + c^1_{s_ik_i} + c^0_{k_ik_i} + c^1_{k_ik_{i+1}}\Big) + \Big(v^0_{h_i} + b_{h_i} \\
&\quad + v^1_{h_i} + v^1_{e_i} + v^0_{e_i} + v^0_{e_i} + b_{e_i} + v^1_{e_i} + v^1_{e_{i+1}} + v^0_{e_{i+1}}\Big) + \dots + t(s_n,h_n,k_n,e_n) \\
&\quad > t(s_1,h_1,k_1,e_1) + \dots + \Big(c^0_{k_{i-1}s_i} + c^1_{s_ik_{i+1}} + v^0_{h_i} + b_{h_i} + v^1_{h_i} + v^1_{e_{i+1}} + v^0_{e_{i+1}}\Big) + \dots \\
&\quad + t(s_n,h_n,k_n,e_n) = \mathit{time}(sol_1)
\end{aligned}$$

The inequality follows from the fact that times are always positive and from the fact that we consider crane acceleration $\Big((c^1_{s_ik_i} + c^1_{k_1k_{i+1}}) > c^1_{s_ik_{i+1}}\Big)$. □

## 4.5 An integer programming model, IPCT

We model the CPMPTC by considering the *segments* in which a single move takes place, separated by points in which we have the layout of the bay after the move. If there is a move between points $t$ and $t+1$, the layout of the bays just differ in the position of the container being moved. The first point corresponds to the initial layout of the bay and the last point $T$ to the arranged configuration after a solution of at most $T-1$ moves. Note that $T$ should be strictly greater than the upper bound for the number of moves required to solve the problem. Otherwise, the problem could be infeasible or return a solution for which we cannot guarantee its optimality.

Several mathematical models for the CPMP are studied in Parreño-Torres et al. (2019b). We take as starting point the description of the variables of their IPS6 model since it has the best performance. This model uses two sets of variables, one describing the layout of the bay at each period, the $x$ variables, and another describing the movement of the container being moved at each segment. The second set can be divided into two different groups of variables: $w$ variables that describe the initial position of the container being moved; and $z$ variables that describe the destination position where it will be placed. Our model, IPCT, uses two new groups of variables to link the origin and destination stack of each move, and the destination stack of a move and the origin stack of the next one.

$$x_{shpt} = \begin{cases} 1, & \text{If at point } t \text{ there is a container in stack } s \text{, tier } h \text{, whose priority is } p; \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T}$$

$$w_{shpt} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1 \text{, a container with priority } p \\ & \text{is moved from } (s,h); \\ 0, & \text{Otherwise;} \end{cases}$$
$$\forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T} \setminus \{T\};$$

$$z_{shpt} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1 \text{, a container with priority } p \\ & \text{is moved to } (s,h); \\ 0, & \text{Otherwise;} \end{cases}$$

$$\forall s \in \mathcal{S}; \quad \forall h \in \mathcal{H}; \quad \forall p \in \mathcal{P}; \quad \forall t \in \mathcal{T} \setminus \{T\};$$

$$l_{skt} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1 \text{ there is a move from stack } s \text{ to stack } k; \\ 0, & \text{Otherwise;} \end{cases}$$

$$\forall s, k \in \mathcal{S} : s \neq k; \quad \forall t \in \mathcal{T} \setminus \{T\};$$

$$u_{skt} = \begin{cases} 1, & \text{If in the segment between } t \text{ and } t+1 \text{ there is a move from stack } k \text{ and the} \\ & \text{destination stack of the previous move was stack } s; \\ 0, & \text{Otherwise;} \end{cases}$$

$$\forall s, k \in \mathcal{S} : s \neq k; \quad \forall t \in \mathcal{T} \setminus \{T\};$$

When a container is moved there are two vertical moves in both the origin and destination positions: one with the crane loaded and another with the crane unloaded. We therefore consider $v_h = v_h^0 + v_h^1$ to simplify notation.

The objective function of our model, IPCT, can be formulated as follows:

$$\text{Min} \sum_{s=1}^{S} \Bigg( \sum_{\substack{k=1 \\ k \neq s}}^{S} \Big( (c_{0s}^0 \cdot l_{sk1}) + \sum_{t=1}^{T-1} \Big( c_{sk}^1 \cdot l_{skt} \Big) + \sum_{t=2}^{T-1} \Big( c_{sk}^0 \cdot u_{skt} \Big) \Big) \\ + \sum_{h=1}^{H} \sum_{p=1}^{P} \sum_{t=1}^{T-1} \Big( (v_h + b_h) \cdot w_{shpt} + v_h \cdot z_{shpt} \Big) \Bigg) \tag{4.11}$$

The upper line of equation (4.11) considers the time spent by the crane to perform all the horizontal moves and the bottom line the time spent by the crane to perform all the vertical moves. The first term of the upper line corresponds to the movement of the spreader from its initial position (out of the bay) to the position of the first container to be moved. The second term corresponds to the movements of the loaded crane along the upper travel line from the stacks where the containers are initially located to their destination stacks. The last term of the first line corresponds to the horizontal moves performed to position the crane in the stacks where the next container to be moved is placed. With respect to the bottom line, the first term considers the vertical moves performed to reach the container and to hoist it up (in this case the container is also twistlocked). Finally, the vertical moves of the crane to release the container and position the spreader again on the upper travel line are considered in the second term of the bottom line.

The constraints of IPCT model are as follows.

$$\sum_{s=1}^{S} \sum_{h=1}^{H} w_{shpt} = \sum_{s=1}^{S} \sum_{h=1}^{H} z_{shpt} \qquad \forall p \in \mathcal{P}, \forall t \in \mathcal{T} \setminus \{T\} \tag{4.12}$$

$$\sum_{s=1}^{S} \sum_{h=1}^{H} \sum_{p=1}^{P} w_{shpt} \leq 1; \quad \sum_{s=1}^{S} \sum_{h=1}^{H} \sum_{p=1}^{P} z_{shpt} \leq 1 \qquad \forall t \in \mathcal{T} \setminus \{T\} \tag{4.13}$$

$$\sum_{s=1}^{S} \sum_{h=1}^{H} \sum_{p=1}^{P} w_{shpt+1} \leq \sum_{s=1}^{S} \sum_{h=1}^{H} \sum_{p=1}^{P} w_{shpt} \qquad \forall t \in \mathcal{T} \setminus \{T-1, T\} \tag{4.14}$$

$$\sum_{k=p}^{P} x_{sh+1kT} \leq \sum_{k=p}^{P} x_{shkT} \qquad \forall s \in \mathcal{S}, \forall h \in \mathcal{H} \setminus \{H\}, \forall p \in \mathcal{P} \tag{4.15}$$

$$\sum_{p=1}^{P} x_{s1pt} + \sum_{p=1}^{P} z_{s1pt} \leq 1 \qquad \forall s \in \mathcal{S}, \forall t \in \{2, \dots, T\} \tag{4.16}$$

$$\sum_{p=1}^{P} x_{sh+1pt} + \sum_{p=1}^{P} w_{shpt} + \sum_{p=1}^{P} z_{sh+1pt} \leq \sum_{p=1}^{P} x_{shpt} \qquad \forall s \in \mathcal{S}, \forall h \in \mathcal{H} \setminus \{H\}, \forall t \in \mathcal{T} \setminus \{T\} \tag{4.17}$$

$$x_{shpt} + z_{shpt} = x_{shpt+1} + w_{shpt} \qquad \forall s \in \mathcal{S}, \forall h \in \mathcal{H}, \forall p \in \mathcal{P}, \forall t \in \mathcal{T} \setminus \{T\} \tag{4.18}$$

$$\sum_{p=1}^{P} x_{shpt} + \sum_{p=1}^{P} x_{sh+1pt} + \sum_{p=1}^{P} w_{shpt} + \sum_{p=1}^{P} z_{sh+1pt} + \sum_{p=1}^{P} z_{shpt} \leq 2$$
$$\forall s \in \mathcal{S}, \forall h \in \mathcal{H} \setminus \{H\}, \forall t \in \mathcal{T} \setminus \{T\} \tag{4.19}$$

$$\sum_{h=1}^{H}\sum_{p=1}^{P} z_{shpt} + \sum_{h=1}^{H}\sum_{p=1}^{P} w_{shpt+1} \leq 1 \qquad \forall s \in \mathcal{S}, \forall t \in \mathcal{T} \setminus \{T-1, T\} \tag{4.20}$$

$$\sum_{h=1}^{H} w_{shpt} + \sum_{h=1}^{H} z_{shpt+1} \leq 1 \qquad \forall s \in \mathcal{S}, \forall p \in \mathcal{P}, \forall t \in \mathcal{T} \setminus \{T-1, T\} \tag{4.21}$$

$$\sum_{h=1}^{H}\sum_{p=1}^{P} \left(w_{shpt} + z_{khpt}\right) \leq 1 + l_{skt} \qquad \forall s, k \in \mathcal{S} : s \neq k, \forall t \in \mathcal{T} \setminus \{T\} \tag{4.22}$$

$$\sum_{h=1}^{H}\sum_{p=1}^{P} \left(z_{shpt-1} + w_{khpt}\right) \leq 1 + u_{skt} \qquad \forall s, k \in \mathcal{S} : s \neq k, \forall t \in \mathcal{T} \setminus \{1, T\} \tag{4.23}$$

$$w_{sh1(T-1)} = 0 \quad z_{sh1(T-1)} = 0 \qquad \forall s \in \mathcal{S}, \forall h \in \mathcal{H} \tag{4.24}$$

$$w_{s1p(T-1)} = 0 \qquad \forall s \in \mathcal{S}, \forall p \in \mathcal{P} \setminus \{1\} \tag{4.25}$$

Constraints (4.12) force the number of containers of each priority remains constant throughout the moves. At most one move can be made in each segment by constraints (4.13). Moves are placed in the earliest time segments by constraints (4.14), i.e., if there has been a move in the segment between $t$ and $t+1$ points, another move must have been made between $t-1$ and $t$, otherwise the former move could have been made earlier. In this way, we force the bay to be ordered at the last point $T$, constraints (4.15). Constraints (4.16) and (4.17) ensure that each slot can hold at most one container. Moreover, constraints (4.16) are strengthened so that if one slot of the bottom tier is occupied, it cannot receive another container. Constraints (4.17) also have two other effects: when a slot is occupied, the slots below it must also be occupied; and only the highest containers can move. Constraints (4.18) say that if a container does not move, it is still present in the layout of the bay of the next point. If a container moves in the segment between $t$ and $t+1$, it will be in its initial position in the layout of point $t$, but not in the layout of point $t+1$. Analogously, the position to which a container moves cannot be occupied in the layout of point $t$ since it will be occupied in $t+1$. Constraints (4.19) strengthen our formulation by limiting the moves that can be made when we consider two consecutive tiers of the same stack. If positions $(s,h)$ and $(s,h+1)$ are occupied at point $t$, then no containers can be moved to those positions nor can be moved from position $(s,h)$ between $t$ and $t+1$ points. Similarly, if a container is moved from position $(s,h)$ between $t$ and $t+1$ points, then that position is occupied at point $t$, there are no containers on its top, and no containers can be loaded to that stack between $t$ and $t+1$. Constraints (4.20) avoid direct transitive moves and (4.21) the movement of the same container in two successive moves. Constraints (4.20) and (4.21) are not used on instances

**Algorithm 2** Branch and bound algorithm pseudocode.

```
 1: function CTA($bay$)
 2:     $lb \leftarrow$ LowerBound($bay$)                          ▷ Lower bound for the number of moves
 3:     $u \leftarrow \infty$                                        ▷ Upper bound for the number of moves
 4:     $\pi \leftarrow$ Heuristic($bay, \varnothing, \varnothing, u$)                    ▷ Feasible solution
 5:     if $\pi \neq \varnothing$ then $u \leftarrow$ Upper($bay, \pi$)
 6:     $l \leftarrow lb$
 7:     while $l \leq u$ do
 8:         DFS($bay, \pi, l, u$)
 9:         if $l = moves(\pi) + 1$ and $l \neq u$ then $l \leftarrow u$
10:         else
11:             $l \leftarrow l + 1$
12:     return $\pi$
```

where all containers have different priority. Constraints (4.22) and (4.23) establish between which stacks the movements are made. Constraints (4.22) connect the stacks between which a move of a container occurs and (4.23) connect the destination stack of a move with the origin stack of the next container to be moved. Finally, constraints (4.24) and (4.25) are valid constraints ensuring that neither a container of the lowest priority nor a container placed on the bottom tier of a stack will be moved on the last movement.

## 4.6 Branch and Bound algorithm

### 4.6.1 Structure of the search tree

We propose a branch and bound algorithm in which the search strategy consists of repeatedly executing a limited depth version of a depth first search with increasing depth limits. It is known in computer science as an *iterative deepening search* and allows us to conduct a breadth-first search (BFS) in a depth-first way so as not to run out of memory. Algorithm 2 shows the pseudocode of our approach. Given the initial configuration of the bay, on line 2 the function LowerBound() computes the lower bound for the number of moves $lb$. On line 3, the upper bound on the number of moves to solve the CPMPCT, $u$, is initialized to infinity. The procedure then tries on line 4 to obtain a feasible solution, that is, a solution without blocking containers, using the heuristic algorithm Heuristic() described in Section 4.6.2. If a feasible solution $\pi$ is obtained, the upper bound for the number of moves, $u$, is updated by function Upper(), according to Proposition 13. The function DFS() runs a DFS up to a depth $l$. The value $l$ is initialized at $lb$ on line 6. Unlike in the CPMP, even if the approach finds a feasible solution with $l$ moves, the algorithm increases the depth by one (line 11) and runs the DFS again, as it is possible to obtain solutions with shorter crane times and more moves. Every time a better solution $\pi$ is found, the upper bound $u$ is updated. The algorithm stops once $l = u$, ensuring that the solution obtained is optimal. If a solution with $moves(\pi) = l$ is reached and the DFS with depth limit $l + 1$ does not improve it, the depth limit is directly updated to $u$ (line 9).

All possible moves are explored at each level of the DFS, pruning branches that satisfy at least one of these criteria: (*i*) Proposition 19 is satisfied, (*ii*) a lower bound for the number of moves is greater than the current depth limit, (*iii*) the lower bound for the crane time *lbc*, using $LB^1_{ct}$, exceeds the crane time of the best solution obtained so far. If $LB^1_{ct}$ does not exceed it, we use $LB^2_{ct}$, but as its computational cost is higher, we only compute it if considering an approximation, in which all blocking containers could be moved to the bottom tier, the branch could be pruned.

The DFS first explores branches with the lowest lower bound for the number of moves, using as a tie breaking criterion the lowest lower bound for the crane time $LB^1_{ct}$. During the DFS, the heuristic algorithm HEURISTIC() is run at every node to try to obtain a feasible solution.

### 4.6.2 A heuristic algorithm for repairing non-completely arranged solutions

Given a non-completely arranged solution $\pi'$, the best solution achieved so far $\pi$, and the upper bound for the number of moves $u$, HEURISTIC() tries to obtain a solution better than the current one (see Algorithm 3 for the pseudocode). The heuristic makes BG and GG moves such as those proposed in Tanaka and Tierney (2018), but also GB moves, stopping when the number of moves performed is greater than the upper bound for the number of moves or when the time spent to perform those moves plus a the lower bound for the time spent to perform a move $t_{min}$ exceeds the best current solution (line 4).

The heuristic first selects the best BG move by function BESTBG() on line 6 according to the following order of preference:

1. The move with the smallest difference between the container at the top of the destination stack and the container moved. The smaller the difference, the smaller the number of containers that can be placed between the two containers.
2. The container with the highest priority. Therefore, the placement of this container does not affect other BG movements of the blocking containers of the current disposition of the bay.

When no further BG moves are possible, the heuristic selects the best GG move by BESTGG() on line 7 according to the following list:

1. The move with the largest difference between the new top of the origin stack and the top of the destination stack before the movement. We only consider the moves in which the difference is at least one. The larger the difference, the larger the number of containers that can be placed on the top of both stacks.
2. The move with the highest priority of the new top of the origin stack. The larger it is, the more containers can be placed on top of it.
3. The move with the fewest number of containers in the origin stack. This allows the heuristic to empty a complete stack in later moves.

If a BG or GG move has been selected, it is applied and added to the partial solution on line 11 by APPLY(). As soon as the bay is ordered, the heuristic ends and updates the best solution obtained so far, $\pi$, with the new solution reached if the crane time is lower on line 13. Otherwise, as long as the number of moves that have been made is lower than the upper bound for the number of moves, $u$, and the partial solution does not exceed the crane time of the best solution, GB moves are carried out by function TRY_GB() on line 9.

**Algorithm 3** Heuristic pseudocode.

```
1: function HEURISTIC($bay$, $\pi'$, $\pi$, $u$)
2:     $M \leftarrow \pi'$                                        ▷ Partial solution
3:     while $moves(M) < u$ and $time(M) + t_{min} < time(\pi)$ do
4:             ▷ $u$: Upper bound for the number of moves; $t_{min}$: Minimum time spent to perform a move
5:         $m \leftarrow \varnothing$
6:         $m \leftarrow$ BESTBG($bay$)
7:         if $m = \varnothing$ then $m \leftarrow$ BESTGG($bay$)
8:         if $m = \varnothing$ then
9:             $flag \leftarrow$ TRY_GB($M$, $bay$)
10:        else
11:            $M \leftarrow$ APPLY($bay, M, m$)                      ▷ Make the move
12:            if BLOCKS($M$) $= 0$ and $time(M) < time(\pi)$ then
13:                $\pi \leftarrow M$                      ▷ Update the best solution so far
14:                return $\pi$
15:        if $m = \varnothing$ and $flag =$ FALSE then
16:            return $\varnothing$
17:    return $\pi$
```

The idea of the TRY_GB() function is to empty a stack that has no blocking containers, placing its containers on stacks where there are also no blocking containers. The stack with the smallest number of containers is always selected to be emptied and its containers are moved to the second stack with the smallest number until it is filled, then to the third stack and so on. Since the containers moved through GB moves will be placed upside down, in the following iterations the empty column will be filled up employing BG moves. If a feasible solution is not obtained, the function returns FALSE. This function is just used if there is at most one blocking container per stack. Refer to Figure 4.8 for a example in which a feasible solution is obtained by performing GB moves.

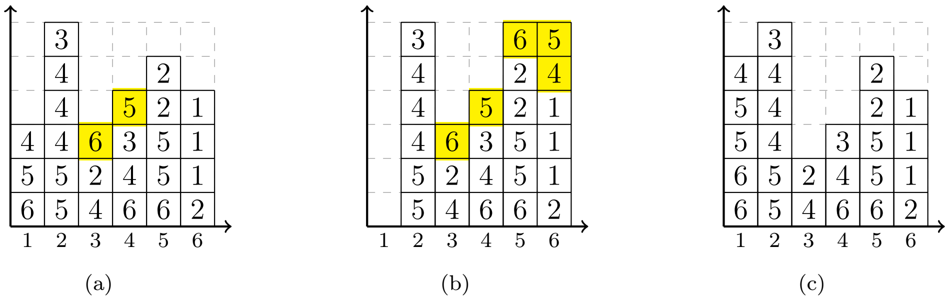


Figure 4.8: Example in which a feasible solution is obtained by performing GB moves. Subfigure (a) shows a layout in which neither BG nor GG moves satisfying the heuristic criteria can be performed. Subfigure (b) shows the layout after emptying one stack. Finally, the bay is arranged by performing only BG moves as shown in Subfigure (c).

## 4.7 Computational experiments

We conduct an extensive computational analysis to test the performance of the mathematical model and the branch and bound algorithm proposed. We refer them as IPCT and CTA, respectively. Both were coded in C/C++ and executed on virtual machines with 4 virtual processors and 16 GBytes of RAM running Windows 10 Enterprise 64 bits. The virtual machines are run in an OpenStack virtualization platform supported by 12 blades, each with four 12-core AMD Opteron Abu Dhabi 6344 processors running at 2.6 GHz and 256 GBytes of RAM, for a total of 576 cores and 3 TBytes of RAM. We fix the time limit to be 3600 seconds and use CPLEX 12.7 with 4 threads as a solver.

### 4.7.1 Test instances

We assess our approaches using these four well-known datasets from the scientific literature:

**EMM dataset** The instances from Expósito-Izquierdo et al. (2012). In total, 450 instances with 4 tiers and with the number of stacks varying among 4, 7 or 10 stacks.

**ZJY dataset** Instances from Zhang et al. (2016) that are grouped into 5 categories of 20 instances each, with 6, 7, 8 or 9 stacks and 4 tiers, and 6 stacks and 5 tiers.

**BZ dataset** This dataset from van Brink and van der Zwaan (2014) contains instances with 4 and 5 tiers and a number of stacks ranging in size from 3 to 9, with a total of 960 instances.

**CV dataset** Instances from Caserta and Voß (2009) range in size from 3 stacks and 5 tiers up to 10 stacks and 10 tiers, totalling 760 instances. These instances are among the most difficult in the premarshalling literature because all containers have different priorities and all stacks are filled to the same tier with two empty tiers on top of each stack.

### 4.7.2 Performance of the mathematical model

Since IPCT requires an initial feasible solution to set the $T$ value, we asses its performance using the instances from EMM, ZJY, BZ, and CV datasets solved by IPS6 in Parreño-Torres et al. (2019b). Thus, the $T$ used to solve IPCT is obtained from the solution provided by IPS6 according to Proposition 13. Mathematical models for the CPMP are flexible and easier to implement, but fail to solve the largest CV instances sizes. Therefore, only the CV instances ranging in size from 3 stacks and 5 tiers up to 7 stacks and 6 tiers, are considered.

We focus here on evaluating the performance of our model and on seeing if the crane time of the solutions obtained really differ from that of the solutions obtained with IPS6 (which solves the classic CPMP). Table 4.3 shows the results obtained for the four datasets grouping the instances by number of tiers (H) and number of stacks (S).

In the light of the results, IPCT solves to optimality 87.98%, 85.33%, 43.21%, and 67.48% of the instances belonging to EMM, ZJY, BZ, and CV datasets. These percentages increase significantly to 97.95%, 87.65%, 97.76%, and 83.43% if we consider the instances in which the model obtains a feasible solution. The average crane time of the solutions reached with IPS6 do not correspond in any case to that of the optimal solution obtained by IPCT for the CPMPCT and furthermore the relative deviation can be up to 24%.

We focus now on the BZ dataset, groups with 7 and 9 stacks, and 4 and 6 tiers. The crane time decreases on average between 4.7% and 7.7% and between 13.83% and 24% in instances

Table 4.3: Performance of IPCT model on the instances optimally solved by IPS6 on the EMM, ZJY, BZ, and CV datasets grouped by number of tiers (H) and number of stacks (S).

| | | | | IPCT | | | | | IPS6 | | IPCT vs. IPS6 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | Avg. | | | Avg. | | | |
| Dataset | H | S | #Inst. | #Opt. | #Feas. | Moves | CTime | CPU(s) | Moves | CTime | AVRPD | MAXRPD |
| EMM | 4 | 4 | 150 | 150 | 150 | 4.76 | 472.93 | 251.90 | 4.76 | 477.46 | 1.13 | 21.29 |
| | 4 | 7 | 126 | 115 | 124 | 5.77 | 605.08 | 185.89 | 5.77 | 630.02 | 3.75 | 14.66 |
| | 4 | 10 | 115 | 79 | 109 | 6.29 | 670.03 | 332.02 | 6.29 | 715.00 | 6.28 | 15.30 |
| | Total | | 391 | 344 | 383 | 5.45 | 562.37 | 248.23 | 5.45 | 583.01 | 3.19 | 21.29 |
| ZJY | 4 | 6 | 20 | 10 | 17 | 8.40 | 878.60 | 740.87 | 8.40 | 912.82 | 3.48 | 9.41 |
| | 4 | 7 | 17 | 11 | 16 | 9.36 | 987.85 | 533.23 | 9.36 | 1012.90 | 2.62 | 8.91 |
| | 4 | 8 | 18 | 9 | 17 | 7.33 | 794.97 | 839.89 | 7.33 | 813.80 | 2.61 | 5.54 |
| | 4 | 9 | 17 | 4 | 13 | 8.00 | 846.07 | 265.92 | 8.00 | 880.24 | 3.90 | 4.45 |
| | 5 | 6 | 9 | 1 | 8 | 11.00 | 1261.70 | 2017.25 | 11.00 | 1263.11 | 0.11 | 0.11 |
| | Total | | 81 | 35 | 71 | 8.46 | 898.66 | 683.26 | 8.46 | 925.10 | 2.94 | 9.41 |
| BZ | 4 | 3 | 120 | 120 | 120 | 3.69 | 374.34 | 0.62 | 3.69 | 376.93 | 0.54 | 7.68 |
| | 4 | 5 | 120 | 120 | 120 | 7.58 | 444.76 | 5.00 | 7.58 | 457.37 | 2.61 | 14.15 |
| | 4 | 7 | 120 | 117 | 120 | 4.38 | 535.70 | 100.70 | 4.38 | 562.20 | 4.74 | 18.51 |
| | 4 | 9 | 120 | 98 | 119 | 8.23 | 600.71 | 200.94 | 8.23 | 644.29 | 6.96 | 24.03 |
| | 6 | 3 | 120 | 116 | 119 | 5.26 | 917.30 | 43.50 | 5.26 | 924.27 | 0.79 | 9.09 |
| | 6 | 5 | 120 | 95 | 100 | 8.52 | 1023.02 | 237.54 | 8.52 | 1054.03 | 2.87 | 18.69 |
| | 6 | 7 | 104 | 64 | 92 | 5.92 | 1050.33 | 391.55 | 5.92 | 1111.04 | 5.42 | 13.83 |
| | 6 | 9 | 93 | 32 | 83 | 8.31 | 1018.88 | 439.58 | 8.31 | 1106.15 | 7.69 | 15.10 |
| | Total | | 893 | 762 | 873 | 6.09 | 686.69 | 129.77 | 6.09 | 712.45 | 3.38 | 24.03 |
| CV | 5 | 3 | 40 | 40 | 40 | 8.79 | 984.27 | 201.16 | 8.79 | 994.09 | 0.91 | 4.24 |
| | 5 | 4 | 38 | 34 | 37 | 8.41 | 973.75 | 342.69 | 8.41 | 992.59 | 2.12 | 12.66 |
| | 5 | 5 | 32 | 21 | 29 | 8.38 | 987.73 | 817.42 | 8.38 | 1018.80 | 3.15 | 7.91 |
| | 5 | 6 | 23 | 8 | 15 | 8.00 | 956.86 | 1057.91 | 8.00 | 985.91 | 2.76 | 5.34 |
| | 5 | 7 | 11 | 1 | 5 | 8.00 | 940.13 | 868.36 | 8.00 | 1010.72 | 6.98 | 6.98 |
| | 5 | 8 | 7 | 0 | 2 | - | - | - | - | - | - | - |
| | 6 | 4 | 9 | 5 | 7 | 9.60 | 1170.06 | 861.35 | 9.60 | 1179.32 | 0.79 | 1.74 |
| | 6 | 5 | 2 | 1 | 1 | 11.00 | 1360.83 | 3032.89 | 11.00 | 1391.82 | 2.23 | 2.23 |
| | 6 | 6 | 1 | 0 | 0 | - | - | - | - | - | - | - |
| | Total | | 163 | 110 | 136 | 8.58 | 991.15 | 486.68 | 8.58 | 1009.93 | 1.91 | 12.66 |

The column “#Inst." represents the total number of instances and columns “#Opt." and “#Feas." show the number of optimal and feasible solutions obtained. The “Moves" and “CTime" columns represent the average number of moves and average time spent by the crane on the instances solved to optimality and the “CPU(s)" columns the average running time (in seconds) spent by each model. The columns “AVRPD" and “MAXRPD" show the average of the relative percentage deviation and the maximum relative percentage deviation in the optimally solved instances. The notation “-” is used when the averages cannot be calculated.

that present the best improvement. These cases represent fairly common bay sizes in terminals around the world. The practical implications are huge. The presented results indicate that minimizing crane time results in much shorter premarshalling operations, effectively saving on expensive hourly wages of stevedores, crane operations, etc.

With respect to the running times, the highest are obtained in the datasets in which the model solves a lower percentage of instances, CV and ZJY datasets. In these datasets the average times are 486.68 and 683.26 seconds. On the other hand, BZ dataset has the shortest average running time, 129.77 seconds and is followed by EMM dataset with an average of 248.23 seconds.

In the instances studied in Table 4.3, the number of moves to solve the CPMP equals the number of moves to solve the CPMPCT. This happens because we only consider the instances solved optimally by *IPS6*, and the average number of moves is relatively small (about 8 moves). However, Figure 3 in Section 4.3.3 shows four cases that belong to data sets from the literature, and in two of them the number of movements in the optimal solution for the CPMP is less than that of the CPMPCT. This situation is more frequent as the size of the bay grows and the number of movements needed to reorganize it increases.

### 4.7.3 Comparing the model with the branch and bound algorithm

We compare the performance of IPCT and that of CTA over the set of instances tested in the previous section. Table 4.4 shows the results obtained. It provides the number of optimal and feasible solutions reached by IPCT and CTA, as well as the average running times for the instances in which both of them reached the optimal solution.

Table 4.4: Comparing CTA vs. IPCT in the EMM, ZJY, BZ and CV instances to evaluate the performance of the mathematical model IPCT in the previous section grouped by number of tiers (H) and number of stacks (S).

| | | | | #Opt. | | #Feas. | | | Avg. | | Avg. CPU(s) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | H | S | #Inst. | IPCT | CTA | IPCT | CTA | #Both | Moves | CTime | IPCT | CTA |
| EMM | 4 | 4 | 150 | 150 | 150 | 150 | 150 | 150 | 4.76 | 472.93 | 251.90 | 0.27 |
| | 4 | 7 | 126 | 115 | 125 | 124 | 126 | 115 | 5.77 | 605.08 | 185.89 | 0.79 |
| | 4 | 10 | 115 | 79 | 83 | 109 | 115 | 76 | 6.15 | 654.80 | 294.54 | 107.00 |
| | | Total | 391 | 344 | 358 | 383 | 391 | 341 | 5.41 | 558.03 | 239.14 | 24.23 |
| ZJY | 4 | 6 | 20 | 10 | 20 | 17 | 20 | 10 | 8.40 | 878.60 | 740.87 | 0.41 |
| | 4 | 7 | 17 | 11 | 17 | 16 | 17 | 11 | 9.36 | 987.85 | 533.23 | 1.32 |
| | 4 | 8 | 18 | 9 | 17 | 17 | 18 | 9 | 7.33 | 794.97 | 839.89 | 16.90 |
| | 4 | 9 | 17 | 4 | 9 | 13 | 17 | 4 | 8.00 | 846.07 | 265.92 | 18.27 |
| | 5 | 6 | 9 | 1 | 9 | 8 | 9 | 1 | 11.00 | 1261.70 | 2017.25 | 6.26 |
| | | Total | 81 | 35 | 72 | 71 | 81 | 35 | 8.46 | 898.66 | 683.26 | 7.14 |
| BZ | 4 | 3 | 120 | 120 | 120 | 120 | 120 | 120 | 3.69 | 374.34 | 0.62 | 0.01 |
| | 4 | 5 | 120 | 120 | 120 | 120 | 120 | 120 | 4.38 | 444.76 | 5.00 | 0.02 |
| | 4 | 7 | 120 | 117 | 119 | 120 | 120 | 117 | 5.26 | 535.70 | 100.70 | 4.85 |
| | 4 | 9 | 120 | 98 | 98 | 119 | 120 | 96 | 5.81 | 590.17 | 135.02 | 65.75 |
| | 6 | 3 | 120 | 116 | 120 | 119 | 120 | 116 | 7.58 | 917.30 | 43.50 | 0.02 |
| | 6 | 5 | 104 | 95 | 102 | 100 | 104 | 94 | 8.16 | 1014.21 | 211.95 | 19.80 |
| | 6 | 7 | 96 | 64 | 56 | 92 | 96 | 53 | 7.76 | 962.17 | 125.93 | 201.11 |
| | 6 | 9 | 93 | 32 | 21 | 83 | 93 | 21 | 7.29 | 918.22 | 267.80 | 502.67 |
| | | Total | 893 | 762 | 756 | 873 | 893 | 737 | 5.90 | 664.38 | 85.06 | 40.65 |
| CV | 5 | 3 | 40 | 40 | 40 | 40 | 40 | 40 | 8.78 | 984.27 | 201.16 | 0.01 |
| | 5 | 4 | 38 | 34 | 38 | 37 | 38 | 34 | 8.41 | 973.75 | 342.69 | 0.03 |
| | 5 | 5 | 32 | 21 | 32 | 29 | 32 | 21 | 8.38 | 987.73 | 817.42 | 0.15 |
| | 5 | 6 | 23 | 8 | 23 | 15 | 23 | 8 | 8.00 | 956.86 | 1057.91 | 0.54 |
| | 5 | 7 | 11 | 1 | 11 | 5 | 11 | 1 | 8.00 | 940.13 | 868.36 | 0.52 |
| | 5 | 8 | 7 | 0 | 6 | 2 | 7 | 0 | - | - | - | - |
| | 6 | 4 | 9 | 5 | 9 | 7 | 9 | 5 | 9.60 | 1170.06 | 861.35 | 0.07 |
| | 6 | 5 | 2 | 1 | 2 | 1 | 2 | 1 | 11.00 | 1360.83 | 3032.89 | 1.13 |
| | 6 | 6 | 1 | 0 | 1 | 0 | 1 | 0 | - | - | - | - |
| | | Total | 163 | 110 | 162 | 136 | 163 | 110 | 8.58 | 991.15 | 486.68 | 0.10 |

The column “#Inst." gives the total number of instances tested at each group, the columns “#Opt." and “#Feas." represent the number of optimal and feasible solutions obtained by each approach, and the column “#Both” provides the number of instances in which both of them reached the optimal solution. The “Moves", “CTime" and “Avg. CPU(s)" columns represent the average number of moves, average time spent by the crane, and the average running time (in seconds). The notation “-” is used when the averages cannot be calculated.

The algorithm CTA obtains a feasible solution in all of the instances whereas IPCT did not obtain any in a few of the cases. Regarding to the number of optimal solutions, CTA obtains 96 more optimal solutions than IPCT over the 1528 instances tested, or 6.3% more. Although CTA solves 3.6%, 45.7%, and 31.9% more than IPCT on EMM, ZJY, and CV datasets, on BZ it solves 0.7% less than IPCT due to the effect of the largest instances in this dataset.

Considering the instances optimally solved by both of them (column "#Both"), it is remarkable that the average running times are cut down by CTA between 52.20% on the BZ dataset and 99.98% on the CV dataset. We further note that in some groups such as those with 5 tiers and 4 or 5 stacks, the average running times greatly decrease from 342.69 seconds to 0.03 seconds and from 817.42 seconds to 0.15 seconds, though in the largest groups of BZ, with 6 tiers and 7 or 9 stacks, the situation reverses.

### 4.7.4 Performance of the branch and bound algorithm

We evaluate our algorithm on the instances of EMM, ZJY and BZ datasets, and also on all the instances of CV dataset. The results are shown in Table 4.5 and 4.6. For the instances optimally solved by CTA, they show the average crane times and the average running times. For the instances in which a feasible solution is found, the average crane times and the average running times required to get the best solution.

In Table 4.5, a solution is found for all the instances and in a running time that on average does not exceed 180 seconds for any of the datasets. In addition, an optimal solution is found for around 80% of the instances in the three datasets.

Table 4.5: Performance of the algorithm CTA on EMM, ZJY, and BZ datasets grouped by number of tiers (H) and stacks (S).

| | | | | | Avg. Opt. | | | | Avg. Feas. | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Dataset | H | S | #Inst. | #Opt. | Moves | CTime | CPU(s) | #Feas. | Moves | CTime | CPU(s) |
| EMM | 4 | 4 | 225 | 150 | 4.76 | 472.93 | 0.27 | 150 | 4.76 | 472.93 | 0.27 |
| | 4 | 7 | 225 | 125 | 6.32 | 657.64 | 27.13 | 150 | 8.64 | 905.64 | 87.25 |
| | 4 | 10 | 225 | 83 | 6.52 | 692.36 | 197.32 | 150 | 11.51 | 1222.22 | 395.29 |
| | | Total | 675 | 358 | 5.71 | 588.30 | 55.33 | 450 | 8.30 | 866.93 | 160.94 |
| ZJY | 4 | 6 | 20 | 20 | 10.15 | 1053.42 | 16.92 | 20 | 10.15 | 1053.42 | 16.92 |
| | 4 | 7 | 20 | 18 | 10.22 | 1071.74 | 23.39 | 20 | 10.70 | 1122.24 | 21.24 |
| | 4 | 8 | 20 | 17 | 9.18 | 976.58 | 238.74 | 20 | 9.85 | 1053.99 | 202.94 |
| | 4 | 9 | 20 | 10 | 10.20 | 1078.95 | 585.43 | 20 | 11.80 | 1266.02 | 295.57 |
| | 5 | 6 | 20 | 15 | 13.00 | 1448.46 | 488.19 | 20 | 14.05 | 1582.43 | 366.21 |
| | | Total | 100 | 80 | 10.50 | 1118.48 | 224.94 | 100 | 11.31 | 1215.62 | 180.58 |
| BZ | 4 | 3 | 120 | 120 | 3.69 | 374.34 | 0.01 | 120 | 3.69 | 374.34 | 0.01 |
| | 4 | 5 | 120 | 120 | 4.38 | 444.76 | 0.02 | 120 | 4.38 | 444.76 | 0.02 |
| | 4 | 7 | 120 | 119 | 5.37 | 546.53 | 12.55 | 120 | 5.43 | 553.47 | 12.50 |
| | 4 | 9 | 120 | 98 | 5.92 | 600.48 | 94.52 | 120 | 7.13 | 724.87 | 128.94 |
| | 5 | 3 | 120 | 120 | 7.88 | 949.12 | 0.03 | 120 | 7.88 | 949.12 | 0.03 |
| | 5 | 5 | 120 | 105 | 8.96 | 1102.35 | 47.29 | 120 | 10.06 | 1233.84 | 41.40 |
| | 5 | 7 | 120 | 56 | 8.02 | 996.30 | 265.96 | 120 | 12.38 | 1528.31 | 265.03 |
| | 5 | 9 | 120 | 21 | 7.29 | 918.22 | 502.67 | 120 | 15.40 | 1932.41 | 814.76 |
| | | Total | 960 | 759 | 6.16 | 694.19 | 54.25 | 960 | 8.29 | 967.64 | 157.84 |

The column "#Inst." gives the total number of instances tested in each group and the columns "#Opt." and "#Feas." represent the number of optimal and feasible solutions obtained. The "Moves", "CTime" and "CPU(s)" columns represent the average number of moves, average time spent by the crane and the average running time (in seconds), respectively.

We now focus on Table 4.6, which contains the results for the CV dataset. The algorithm solves 654 out of the 760 instances, but just 245 to optimality. All the instances with 5 and 6 tiers are solved, and 249 of the 280 instances with 7 tiers. The worst performance is presented in the instance with 8 tiers.

Table 4.6: Performance of CTA on CV dataset grouped by number of tiers (H) and number of stacks (S).

| | | | | Avg. Opt. | | | | Avg. Feas. | | |
|---|---|---|---|---|---|---|---|---|---|---|
| H | S | #Inst. | #Opt. | Moves | CTime | CPU(s) | #Feas. | Moves | CTime | CPU(s) |
| 5 | 3 | 40 | 40 | 8.78 | 984.27 | 0.01 | 40 | 8.78 | 984.27 | 0.01 |
| 5 | 4 | 40 | 40 | 9.03 | 1043.19 | 0.11 | 40 | 9.03 | 1043.19 | 0.11 |
| 5 | 5 | 40 | 40 | 10.15 | 1187.69 | 37.69 | 40 | 10.15 | 1187.69 | 37.69 |
| 5 | 6 | 40 | 35 | 10.77 | 1270.00 | 204.94 | 40 | 11.33 | 1338.96 | 179.33 |
| 5 | 7 | 40 | 25 | 11.48 | 1374.62 | 610.68 | 40 | 12.85 | 1531.04 | 384.34 |
| 5 | 8 | 40 | 11 | 11.00 | 1326.97 | 985.54 | 40 | 13.53 | 1633.31 | 273.04 |
| 6 | 4 | 40 | 35 | 15.26 | 1830.44 | 61.12 | 40 | 15.83 | 1897.71 | 53.76 |
| 6 | 5 | 40 | 9 | 14.56 | 1789.99 | 78.93 | 40 | 17.95 | 2218.00 | 18.54 |
| 6 | 6 | 40 | 3 | 12.67 | 1600.40 | 921.00 | 40 | 19.33 | 2405.58 | 76.17 |
| 6 | 7 | 40 | 0 | - | - | - | 40 | 21.90 | 2765.90 | 181.16 |
| 7 | 4 | 40 | 6 | 18.17 | 2221.35 | 178.50 | 40 | 23.55 | 2933.43 | 597.13 |
| 7 | 5 | 40 | 1 | 15.00 | 1876.20 | 434.54 | 38 | 24.92 | 3208.36 | 373.53 |
| 7 | 6 | 40 | 0 | - | - | - | 37 | 29.49 | 3925.59 | 1288.49 |
| 7 | 7 | 40 | 0 | - | - | - | 36 | 31.56 | 4282.63 | 1327.28 |
| 7 | 8 | 40 | 0 | - | - | - | 36 | 35.94 | 4973.35 | 1780.95 |
| 7 | 9 | 40 | 0 | - | - | - | 32 | 39.44 | 5588.67 | 1596.88 |
| 7 | 10 | 40 | 0 | - | - | - | 30 | 41.10 | 5864.61 | 1758.06 |
| 8 | 6 | 40 | 0 | - | - | - | 2 | 36.50 | 5252.07 | 1346.71 |
| 8 | 10 | 40 | 0 | - | - | - | 3 | 59.33 | 9134.43 | 1632.72 |
| | Total | 760 | 245 | 11.14 | 1315.10 | 171.07 | 654 | 21.07 | 2737.96 | 546.25 |

The column "#Inst." gives the total number of instances tested at each group and the columns "#Opt." and "#Feas." represent the number of optimal and feasible solutions obtained. The "Moves", "CTime" and "CPU(s)" columns represent the average number of moves, average time spent by the crane and the average running time (in seconds). The notation "-" is used when the averages cannot be calculated.

## 4.8 Concluding remarks

We propose the premarshalling problem considering crane time minimization objective (CPMPCT). Even though the premarshalling problem has been widely studied in the literature in the last years due to its relationship with the increase in productivity of container terminals in an extremely competitive environment, all previous papers minimize the number of moves as their objective function to rearrange the bay. In this chapter, crane times have been calculated in a precise way, considering different speeds and accelerations depending on whether the crane is moving with load or not, as well as twistlock times. We show that the number of moves is not a suitable indicator for measuring crane time.

We present a novel dominance criteria to solve the CPMPCT, upper bounds for the number of movements required to solve the CPMPCT, and lower bounds for the problem. Two exact approaches are proposed: a mathematical model and a branch and bound algorithm. An extended computational analysis highlight the need to consider the new objective function as opposed to the one used so far, cutting the crane time down by 24% in some cases. The branch and bound algorithm outperforms the mathematical models, obtaining a solution in 95% of the instances tested.

# Chapter 5

# Solution strategies for a multi-port container stowage planning problem

## 5.1 Introduction

In order to minimize the number of loading/unloading operations at ports, it is essential to have an efficient stowage plan indicating the exact position of each container in the ship. Consider a ship that has to visit a set of ports in a given order to pick up and deliver containers. It starts its route by loading an initial set of containers at the first port, performs requested loading and unloading operations at the intermediary ports, and finally unloads all the remaining containers at the final port. The following data are given: (i) the number of containers to be loaded at each port, (ii) the pickup and delivery port for each container, and thus (iii) the number of containers to be unloaded at each port.

Due to stacking, containers are accessible only if there are no other containers on top of them. If a container is not accessible, the containers above it must be unloaded first and reloaded if they have not yet reached their delivery port. This type of additional movement of a container that has not reached its port of discharge is referred to as a *relocation* or *reshuffle*. Reshuffles are undesirable, as they cause additional operational costs for the ports and increase the service time. We consider a centralized system that aims at minimizing the total number of relocations throughout the route of a ship, so we allow reshuffles that are not strictly necessary in a port if they reduce the number of relocations at later ports.

This chapter deals with the basic problem, where all the containers are of identical size and no additional constraints are present such as stability, maximum weight, etc. It is referred to as the *Container Stowage Planning Problem* (CSPP) and it has been proven to be NP-Hard (Avriel et al., 2000). We basically provide two types of contributions. Firstly, we have developed a new integer linear model, as an alternative to the existing models by Avriel et al. (1998) and by Ding and Chou (2015), requiring fewer variables and obtaining better results in terms of the number of optimal solutions obtained and the quality of feasible solutions when the time limit is reached without proving optimality. However, as the computational study shows the limits of mathematical models for solving large-size instances, in the second part of the study we have developed a metaheuristic GRASP algorithm, going one step further than the constructive algorithms proposed so far and including randomization strategies and a local search tailored to the problem. An extensive computational study shows that GRASP obtains much better results than previous procedures, generating stowage plans with a minimal number of reshuffles in a high percentage of medium- and large-size instances.

The structure of the chapter is as follows. The relevant literature on stowage problems is reviewed in Section 5.2. Section 5.3 presents a detailed definition of the problem. In Section 5.4 we present a new mathematical formulation and propose several families of valid inequalities. The GRASP metaheuristic is presented in Section 5.5, describing its randomized constructive algorithm and the local search. The computational experiments carried out are summarized in Section 5.6. Finally, conclusions and future work are highlighted and discussed in Section 5.7.

## 5.2 Previous work

Academic work on stowage planning problems has increased notably in recent years. However, contributions have been restricted by the inaccessibility of the container business what has led to question the representative power of the models developed. The studies can be classified into two main groups: single-port and multi-port problems.

The single-port problem consists in determining the arrangement of containers in the ship at a given port without considering the loading of containers in subsequent ports. The vast majority of papers in the literature related to stowage planning belong to this group and it is usual to refer to this problem as the Master Bay Planning Problem (MBPP). For a detailed description of the MBPP and its constraints, see for instance Ambrosino et al. (2004). Even though this problem is difficult, it is not as hard as the multi-port problem, so there is a trend of progressively including more realistic constraints. Papers related to the MBPP can be divided into two approaches: those that consider the problem as a whole and those that decompose it into several optimization subproblems. In the first approach, exact methods such as mathematical models (Ambrosino et al., 2004) or branch and bound algorithms (Sciomachen and Tanfani, 2003) can be found, as well as heuristic algorithms, such as Sciomachen and Tanfani (2007), based on an exact method for the 3D-BPP. In the second approach, the problem is divided into two phases. First, the containers are distributed in clusters along the vessel. Then, for each cluster a specific position for each container must be found solving the *Slot Planning Problem* (SPP). A decomposition of this kind makes it possible to solve the SPP independently for each section but does not guarantee an optimal solution for the complete problem. Examples of this approach are Ambrosino et al. (2010), Pacino et al. (2011), Delgado et al. (2012), and Parreño et al. (2016). An updated review of the single-port problems can be found in Larsen and Pacino (2020).

In the multi-port problem, the stowage problem is considered as a dynamic loading and unloading problem. According to the classification by Lehnfeld and Knust (2014), it is a combined problem, since it is possible to perform two types of moves. Two main approaches have been followed in recent years. On the one hand, some authors use a multi-phase approach, proposing a decomposition of the problem into several optimization models that can be solved individually. A common practice is again to first solve a multi-port MBPP (MP-MBPP) variant to allocate containers to different sections of the ship and then to solve a SPP to determine the exact position (slot) of the containers in each section. This decomposition makes it possible to include realistic characteristics of the problem, in the objective function, considering costs related to non-loaded containers, reshuffles, and quay cranes, and also to include constraints concerning the stability of the ship. Among the studies utilizing this decomposition structure for multi-port stowage planning problems, Wilson and Roach (2000) derive a method based on Tabu Search (TS). Kang and Kim (2002) iteratively solve the MP-MBPP via a greedy heuristic and the SPP via a tree search method, Pacino et al. (2011) combine an IP formulation for the MP-MBPP and a constraint programming approach for the SPP, and Ambrosino et al. (2015) and Ambrosino et al. (2017) provide an MIP formulation for the MP-MBPP and a heuristic for the SPP.

On the other hand, there are several studies that follow a single-phase approach. The problem is solved as a whole, looking for loading strategies that minimize the number of reshuffles at each port on the route, thus providing a complementary view of the stowage problem, usually considering routes with many ports and ships with large numbers of containers, but disregarding stability conditions and considering just one type of container. The first mathematical formulation of the CSPP was provided by Avriel and Penn (1993) and revisited by Avriel et al. (1998). In addition to this formulation, Avriel and Penn (1993) introduce a *Whole Columns Heuristic Procedure* to solve the CSPP. Given a transportation matrix representing the demand between each pair of ports, the authors decompose it into two sub-matrices. The containers for the first sub-matrix are allocated through a simple procedure, whereas binary linear programming is used for the second sub-matrix. Avriel et al. (1998) extend this idea by using a dynamic slot-assignment scheme leading to a more efficient heuristic, which is referred to as the *Suspensory Heuristic Procedure*. The same problem was studied by Dubrovsky et al. (2002), who presented a metaheuristic approach that provides a genetic algorithm with an efficient solution-encoding scheme. This algorithm shows a similar performance to that of Avriel et al. (1998) on the CSPP instances tested and also introduces a ship stability constraint through a penalty function. Ding and Chou (2015) indicate that it is always possible, without loss of generality, to assume that the ship is fully loaded when traveling between ports. The transportation matrices satisfying this property are referred to as *full* matrices and Ding and Chou (2015) present a method for generating such matrices. They further provide a new IP formulation and a heuristic method to solve the CSPP for full transportation matrices. This heuristic generates a stowage plan for each port without considering the transportation information for subsequent ports (hence no look-ahead strategy). The computational study by Ding and Chou (2015) on large-scale problem instances indicates that their algorithm performs better than the Suspensory Heuristic of Avriel et al. (1998) when the number of ports visited is large. A mixed integer programming (MIP) formulation of the CSPP with an exponential number of variables is proposed by Roberti and Pacino (2018). The authors develop a column generation-based lower bounding procedure and combine it with a compact MIP model to solve instances with up to 10 ports and 5000 containers to optimality. Problem instances of this size remain unsolvable by the compact IP formulations available in the literature.

The state-of-the-art compact IP formulation and heuristic method for solving the CSPP have recently been provided by Parreño-Torres et al. (2019a), fully described in the current chapter. This new IP formulation has fewer variables than the formulations by Avriel et al. (1998) and Ding and Chou (2015). We propose several valid inequalities to strengthen the linear programming (LP) relaxation. The experimental study comparing the three formulations for the CSPP indicates that the new IP model outperforms the previous models by Avriel and Penn (1993) and Ding and Chou (2015) in terms of both solution quality and computing time. Moreover, we develop a Greedy Randomized Adaptive Search Procedure (GRASP) and compare it with the heuristic algorithms by Avriel et al. (1998) and Ding and Chou (2015) on medium and large-scale instances that they generate. The GRASP provides very good solutions, of higher quality than those obtained by the previous algorithms, in less than five minutes for every instance tested.

## 5.3 Problem description

Container stowage plans use a bay-row-tier coordinate system to identify the position of a container aboard the ship, as shown in Figure 5.1. The bays represent the transverse sections of the ship and are numbered from bow (or forward) to stern (or aft). The rows run the length of the ship, with the starboard rows having odd numbers and the port rows having even numbers. The last coordinate is the tier or layer of the containers. Each combination of bay-row-tier defines a cell.

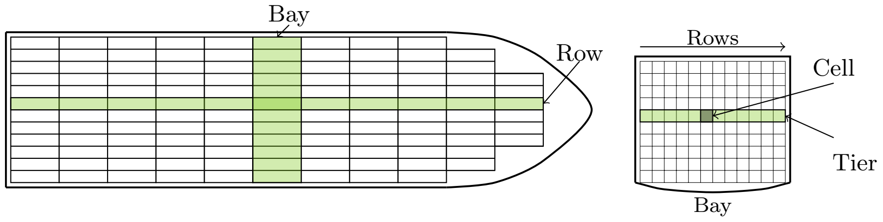


Figure 5.1: Container ship scheme, showing the bay-row-tier coordinate system.

Let us consider a ship with capacity $\Omega$ containers traveling along a trade route consisting of $N$ ports. The ship has $B$ bays, where each bay $b$ contains $C^b$ rows, and each row $c$ of bay $b$ has $R$ cells. All containers are the same size and no stability conditions are considered, so the ship can be seen as having a single bay with $R$ tiers and $\sum_{b\in\mathcal{B}}\sum_{c\in\mathcal{C}^b}(b \times c)$ stacks. Let $S$ equal to $\sum_{b\in\mathcal{B}}\sum_{c\in\mathcal{C}^b}(b \times c)$, the total capacity of the ship can be calculated as $\Omega = R \cdot S$. The position of each container is described by the pair $(r, s)$ where $r \in \{1, \dots, R\} = \mathcal{R}$, $r = 1$ is is the bottom tier, and $s \in \{1, \dots, S\} = \mathcal{S}$, $s = 1$ is the first stack on the left. In other words, tuple $(r, s)$ denotes the location in tier $r$ and stack $s$.

The loading/unloading operations along the route are defined by an $N \times N$ transportation matrix $T$ whose input $(i, j)$ is the number of containers originating at port $i$ with destination port $j$. It is a non-negative upper triangular matrix, so $T_{ij} = 0$ for all $i \geq j$. We assume, as has been done in previous studies (Avriel et al., 1998; Ding and Chou, 2015), that all transportation matrices are feasible, so all the containers to be shipped can be stowed at every port according to equation (5.1).

$$\sum_{k=1}^{i} \sum_{j=i+1}^{N} T_{kj} \leq \Omega \quad \forall i \in \mathcal{N} : i < N; \tag{5.1}$$

The vessel starts its route at port 1, and sequentially visits ports $2, 3, \ldots, N$. At each port $i = 1, \ldots, N-1$, containers with destination $i$ are unloaded and containers bound for destinations $j = i+1, \ldots, N$ are loaded. A container whose destination is port $j$ is called a $j$-container. At the last port $N$, the vessel is emptied. Therefore, it can be seen as a circular route in which port 1 is the same as port $N$. If there is a $j$-container in position $(r, s)$ and there is a $k$-container, $k < j$, in some position $(r', s)$ with $r' < r$, the first one is said to be a blocking container and will produce a reshuffle in some port $l$, $l \leq k$. The objective of the problem is to minimize the number of reshuffles along the route. Note that a non-full loading transportation matrix $T$ can always be transformed into a full-loading matrix $T'$, adding as many dummy containers from $i$ to $(i+1)$ as necessary. We can consider, without loss of generality, the inequality of equation (5.1) as an equality.

## 5.4 Mathematical formulation

In this section we describe a new integer linear model and several classes of valid inequalities to enhance it.

### 5.4.1 An integer programming model

The model involves two types of binary variables describing the configuration of the ship and the containers that are moved unproductively at each port.

$$x_{ijrs} = \begin{cases} 1, & \text{if a } j\text{-container is stowed in slot } (r,s) \text{ after loading operations at port } i. \\ 0, & \text{otherwise} \end{cases}$$
$$\forall i, j \in \mathcal{N} \text{ s.t. } i \leq N-2 \text{ and } i < j, \quad \forall r \in \mathcal{R}, \quad \forall s \in \mathcal{S}$$

$$y_{ijrs} = \begin{cases} 1, & \text{if a } j\text{-container stowed in slot } (r,s) \text{ at port } i-1 \text{ is unproductively moved} \\ & \text{at port } i. \\ 0, & \text{otherwise} \end{cases}$$
$$\forall i, j \in \mathcal{N} \text{ s.t. } 1 < i < N \text{ and } i < j, \quad \forall r \in \mathcal{R}, \quad \forall s \in \mathcal{S}$$

According to the problem description, the ship is emptied after the unloading operations at port $N$. Therefore from port $N-1$ to port $N$ there are only $N$-containers in the ship, and there will be no reshuffles at port $N$. Moreover, since we assume that the number of containers to be loaded on board is not greater than the number of available locations, the exact position of each container when the ship leaves port $N-1$ is irrelevant. Under these assumptions, neither variables $x_{(N-1)Nrs}$ nor variables $y_{Nrs}$ need to be considered.

The multi-port container ship stowage problem can be formulated as follows:

$$\textbf{Min} \quad \sum_{i=2}^{N-1} \sum_{j=i+1}^{N} \sum_{r=1}^{R} \sum_{s=1}^{S} y_{ijrs} \tag{5.2}$$

$$x_{i-1jrs} - x_{ijrs} \leq y_{ijrs} \qquad \forall i,j \in \mathcal{N} : 1 < i \leq N-2, i < j \quad \forall r \in \mathcal{R}, \quad \forall s \in \mathcal{S} \tag{5.3}$$

$$x_{i-1ir-1s} + \sum_{j=i+1}^{N} y_{ijr-1s} \leq x_{i-1irs} + \sum_{j=i+1}^{N} y_{ijrs} \qquad \forall i \in \mathcal{N} : 1 < i < N \quad \forall r \in \mathcal{R} : r > 1, \quad \forall s \in \mathcal{S} \tag{5.4}$$

$$\sum_{r=1}^{R} \sum_{s=1}^{S} x_{1jrs} = T_{1j} \qquad \forall j \in \mathcal{N} : j > 1 \tag{5.5}$$

$$\sum_{r=1}^{R} \sum_{s=1}^{S} (x_{ijrs} - x_{i-1jrs}) = T_{ij} \qquad \forall i,j \in \mathcal{N} : 1 < i \leq N-2, i < j \tag{5.6}$$

$$\sum_{j=i+1}^{N} x_{ijrs} \leq 1 \qquad \forall i \in \mathcal{N} : i \leq N-2, \quad \forall r \in \mathcal{R}, \quad \forall s \in \mathcal{S} \tag{5.7}$$

The objective function (5.2) seeks to minimize the total number of reshuffles. Reshuffles are identified by constraints (5.3) and (5.4). Constraints (5.3) identify changes of destination in a given position. Constraints (5.4) identify unproductive moves required to unload a container, serving two purposes. On the one hand, to take into account the obvious case in which if $x_{i-1ir-1s} = 1$, meaning that there is an $i$-container in slot $(r-1, s)$ at port $i-1$, the container above it in slot $(r, s)$ must also be unloaded at port $i$ (making $x_{i-1irs} = 1$) or there is a reshuffle and one of the variables $y_{ijrs}$ must be 1 for some $j > i$. On the other hand, if there has been a reshuffle in the stack at any tier $r-1$ (and then the variable $y_{ijr-1s}$ is equal to 1 for some $j > i$) the container above it in slot $(r, s)$ must be unloaded at port $i$ or it will produce a new reshuffle (and some variable $y_{ijrs}$ must be 1 for some $j > i$). In this second case, all blocking containers in the stack will be counted as reshuffles. Finally, constraints (5.5) and (5.6) fix the number of containers to be loaded at each port and constraints (5.7) force each location to have at most one container at each port. At the first port, the ship is empty before loading operations, and after these operations there are $T_{1j}$ $j$-containers on board (constraints (5.5)). At other ports, the number of $j$-containers loaded at each port $i$, $T_{ij}$, is equal to the number of $j$-containers on board after loading operations at this port $i$ minus the number of $j$-containers on board before loading operations at that port, that is, minus the total number of containers whose destination is port $j$ and were loaded at a port before $i$ (constraints (5.6)).

We refer to the example in Figure 5.2 to better understand the model and detect the different types of reshuffles that can appear. It shows the configurations of a solution on ports 2 and 3. Each square represents a container and the number indicates the destination port. The gray-marked containers are those that are relocated on port 3. As there is a change of destination in position $(3, 1)$ between ports 2 and 3, there has been a reshuffle at port 3 of that container and $y_{3531} = 1$ by constraints (5.3). We now focus on the 6-container located in slot $(4, 1)$ at port 2. It must have been unloaded at port 3 to unload the 5-container but there is a container with the same destination in slot $(4, 1)$ at port 3, so constraints (5.3) do not consider this case. It is identified by constraints (5.4). Since there has been a reshuffle in the stack 1 at tier 3 at port 3, the container above it in slot $(4, 1)$ must have destination 3 or it will produce a new reshuffle as is the case. Constraints (5.4) also identify unproductive moves required to unload a container at its destination port. Consider now the container located in slot $(4, 3)$ at port 2. There is a 3-container at port 2 in slot $(3, 3)$, so the container on top of it has to be unloaded unproductively at port 3 because its destination is not port 3.

Port 2

| | | | | |
|---|---|---|---|---|
| 5 | 3 | 3 | 3 | 4 |
| 4 | 6 | 3 | 4 | 4 |
| 3 | 5 | 5 | 3 | 5 |
| 2 | 6 | 5 | 3 | 5 |
| 1 | 6 | 6 | 3 | 6 |
| | 1 | 2 | 3 | 4 |

Port 3

| | | | | |
|---|---|---|---|---|
| 5 | 6 | 4 | 4 | 4 |
| 4 | 6 | 5 | 4 | 4 |
| 3 | 6 | 5 | 4 | 5 |
| 2 | 6 | 5 | 4 | 5 |
| 1 | 6 | 6 | 4 | 6 |
| | 1 | 2 | 3 | 4 |

Figure 5.2: Example of configurations when leaving ports 2 and 3 to detect the different types of reshuffles that can arise.

## 5.4.2 New valid inequalities

The integer linear formulation described above produces valid solutions for the multi-port stowage problem, but its linear relaxation is quite loose, producing very low lower bounds. In most cases, the lower bound at the root node is precisely zero. In order to enhance the integer formulation, we studied the solutions provided by its linear relaxation and identified some valid constraints satisfied by all integer solutions but not by the linear relaxation solutions. Adding these inequalities to the initial model would produce a tighter linear relaxation, improving the quality of the bounds and therefore reducing the search tree required to obtain an optimal solution.

We consider the upper tier $R$ of the bay at port $i-1$ and the number of containers going to the next port, $i$. If the number of slots in the upper tier occupied by $i$-containers at port $i-1$ is less than the minimum number required to ensure direct access to these containers, there will be at least as many reshuffles as necessary at port $i$ to reach that number. It is satisfied by adding constraints (5.8).

$$\sum_{s=1}^{S} x_{i-1iRs} + \sum_{j=i+1}^{N} \sum_{s=1}^{S} y_{ijRs} \geq \left\lceil \frac{\sum_{p=1}^{i-1} T_{pi}}{R} \right\rceil \quad \forall i \in \mathcal{N} : 1 < i < N \tag{5.8}$$

Constraints (5.9) extend (5.8) to consider not only tier $R$, but all tiers from $r'$ to $R$.

$$\sum_{s=r'}^{R}\sum_{s=1}^{S} x_{i-1isc} + \sum_{j=i+1}^{N}\sum_{s=r'}^{R}\sum_{s=1}^{S} y_{ijsc} \geq (R-r'+1)\cdot\left\lfloor\frac{\sum_{p=1}^{i-1}T_{pi}}{R}\right\rfloor + \min\Big\{\sum_{p=1}^{i-1}T_{pi}(\text{mod}R), (R-r'+1)\Big\} \tag{5.9}$$

$$\forall i \in \mathcal{N} : 1 < i < N, \quad \forall r' \in \mathcal{R} : r' > 1$$

We now consider the bottom tier. If there are containers on board going to the last port, $N$, either they are placed on top of other $N$-containers or in the bottom tier. Otherwise, they will block other containers and that will produce some reshuffles at subsequent ports.

$$R\cdot\sum_{s=1}^{S} x_{iN1s} + \sum_{i'=i+1}^{N-1}\sum_{j=i'+1}^{N}\sum_{r=2}^{R}\sum_{s=1}^{S} y_{i'jrs} \geq \sum_{p=1}^{i} T_{pN} \quad \forall i \in \mathcal{N} : i \leq N-2 \tag{5.10}$$

Instead of considering only the containers going to the last port, $N$, we can consider the containers going to a port $k$ and subsequent ports. The argument is similar to the previous constraints, but we now have constraints (5.11).

$$R\cdot\sum_{j=k}^{N}\sum_{s=1}^{S} x_{ij1s} + \sum_{i'=i+1}^{N-1}\sum_{j=i'+1}^{N}\sum_{r=2}^{R}\sum_{s=1}^{S} y_{i'jrs} \geq \sum_{p=1}^{i}\sum_{j=k}^{N} T_{pj} \quad \forall i,k \in \mathcal{N} : i \leq N-2, i < k \tag{5.11}$$

If after loading operations at port $i-1$ there is a container in position $(r,s)$ whose destination is port $k$ or a port subsequent to $k$ on top of a container whose destination is previous to $k$, the first container will be relocated at some port prior to $k$. Then, constraints (5.12) hold.

$$\sum_{j=k}^{N} x_{i-1jr+1s} \leq \sum_{j=k}^{N} x_{i-1jrs} + \sum_{p=i}^{k-1}\sum_{j=p+1}^{N} y_{pjr+1s} \quad \forall i,k \in \mathcal{N} : 1 < i \leq N-2, i < k, \quad \forall r \in \mathcal{R} : r < R, \quad \forall s \in \mathcal{S} \tag{5.12}$$

When we introduce these constraints, the linear programming relaxation provides a solution which lies in general not too far away from the integer optimum. More importantly, the lower bound is increased.

We illustrate their effect by an example, in which we consider a container ship with 6 tiers and 4 stacks and whose route is composed of 8 ports. Although its optimal value is 3 reshuffles, the objective value of the LP-relaxation (without valid constraints), provided by CPLEX at the root node of its internal Branch & Bound process, is zero, which is indeed a loose lower bound. Figure 5.3 shows the configuration provided by the LP-relaxation at port 2. On the left hand side, the bay is represented by the large square, divided into smaller squares that represent each slot of the bay, and each one of these squares is split up (using dotted lines) into smaller squares in order to assign the value taken by the variable that represents each destination port.

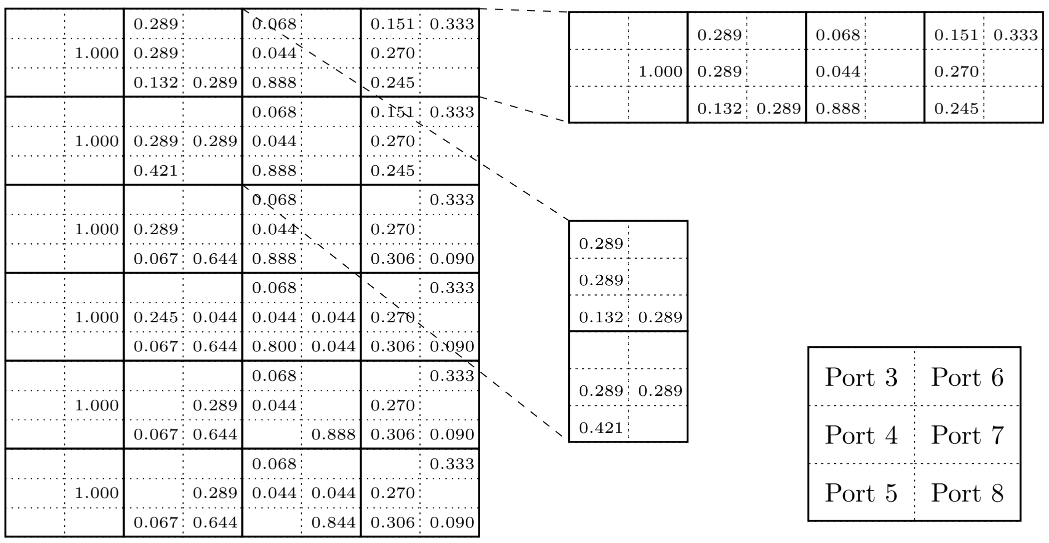


Figure 5.3: Example of configuration at port 2 given by the LP-relaxation at the root node.

Looking at the two enlarged slots that represent slots $(6, 2)$ and $(5, 2)$, and considering port $k = 8$ as a reference, the variable whose destination is port 8 takes a positive value in the upper slot. Nevertheless, in the lower slot the variable whose destination is port 8 takes value zero, so constraints (5.12) are not satisfied. Moreover, according to the transportation matrix, there is one container going to the next port, port 3, so in order to avoid reshuffles, it should be stowed in the top tier, tier 6. However, looking at the top tier, at the top right corner of Figure 5.3, the sum of the values of variables whose destination is port 3 is less than one. This solution could be removed by including constraints (5.8).

If the valid constraints are included in the initial model and the LP-relaxation of this extended model is solved, its objective value takes value two, so the lower bound is increased and the solution obtained is very near to an integer solution.

## 5.5 A GRASP algorithm

GRASP (Greedy Randomized Adaptive Search Procedure) is a multi-start metaheuristic in which each iteration consists basically of two phases: a randomized construction phase and an improvement phase. The first phase builds a feasible solution, and the improvement phase seeks to improve the outcome. Once the two phases have been executed, the solution is stored and the method starts another iteration, keeping the best solution found so far. An extended description of the algorithm can be found in Resende and Ribeiro (2016). GRASP algorithms have been successfully developed for many hard combinatorial problems, in container loading (Correcher et al., 2017), location (Diaz et al., 2017), and routing (Ho and Szeto, 2016), among other areas. Concerning stowage problems closely related to that studied here, Parreño et al. (2016) also used GRASP for the slot planning problem, but in recent papers other approaches have been successfully applied: particle swarm optimization (Matsaini and Santosa, 2018), genetic algorithms (Azevedo et al., 2014, 2018; Yurtseven et al., 2018), simulated annealing (Yurtseven et al., 2018), or hill climbing (Yurtseven et al., 2018).

The following subsections present the specific implementation of the metaheuristic GRASP that we have developed for this stowage problem: the randomizing strategies, and the local search procedure.

### 5.5.1 Randomizing the constructive algorithms

To the best of our knowledge, there are two constructive algorithms in the scientific literature that deal with the problem studied in this chapter: that of Avriel et al. (1998) and that of Ding and Chou (2015). The suspensory algorithm proposed by Avriel et al. (1998) is basically composed of a hanging step and a filling step. In the hanging step the containers are assigned to stacks by increasing order of destination, occupying the top positions (*hanging*), and in the filling step the stacks are completed and containers are put in the usual positions one on top of the other. On the other hand, the algorithm by Ding and Chou (2015) is composed of rules. Containers by decreasing order of destination are assigned one by one following the corresponding rule in each case. The computational results provided by Ding and Chou (2015) show that their algorithm outperforms that of Avriel et al. (1998) and obtains good solutions for the large-size instances tested.

Unlike other implementations of GRASP, the randomization procedure used here does not randomize the selection of the next element to be included in a partial solution, but the constructive process itself. The two constructive algorithms can be randomized, taking into account the way in which they build a feasible solution.

In Avriel et al. (1998) the positions for the containers are assigned stack by stack. We randomize the next stack to be assigned at each step and the order in which containers are loaded at port 1. Since the algorithm places the containers by increasing destination, the randomization consists of using a geometric distribution of parameter 1/2 to choose the sequence. We also randomize the number of containers to be loaded in the Hanging Stage (H), and the containers not assigned in this phase are considered leftover containers to be loaded in the Filling Stage (F).

In Ding and Chou (2015) the assignment is made according to the destination of the containers, group by group, following a set of rules. We randomize the algorithm using two strategies. On the one hand, we modify Rule 4. In the original algorithm, if the destination of the container to be loaded is the last port ($j = N$), and the set of empty stacks and stacks partially filled only with $N$-containers, denoted by $E(4)$, is not empty, Rule 4 assigns $N$-containers to the stack with the largest number of empty slots in set $E(4)$ until no such loading moves can be carried out. We modify this rule, assigning the $N$-containers to stacks selected at random from $E(4)$. On the other hand, we randomize the order of some rules followed in Step 4. If $j < N$, there are three rules, Rules 5, 6, and 7, that assign containers to stacks in such a way that there will not be any blocking containers. These rules are used sequentially. In the randomized algorithm the rules are taken in random order.

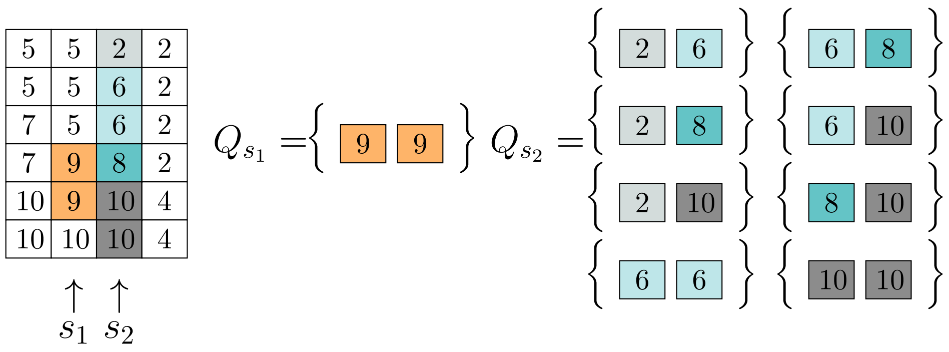
### 5.5.2 Improvement phase

In the improvement phase, the solutions of the constructive phase are modified to find better solutions. When defining a move, we have to take into account the specific structure of the multi-port problem, in which decisions at one port directly determine the layout at later ports. Looking for a move that was simple to compute and, at the same time, able to modify the solution substantially, we realized that loading operations at port 1 play an important role in the final configuration of the stowage plan. The initial configuration largely determines the position of the empty slots after the unloading operations at the subsequent ports. Therefore, we have developed a local search algorithm based on this idea in which only the configuration at port 1 is subject to changes. At this port, we select two stacks $s_1$ and $s_2$, with $s_2 > s_1$. Containers belonging to $s_1$ are grouped according to their destination port and we define $G_j$ as the set formed by all the $j$-containers in stack $s_1$, $j \in \{2, \dots, N\}$. Let $O$ and $D$ be the sets of containers placed in stacks $s_1$ and $s_2$ respectively.

A neighboring solution can be obtained by exchanging a homogeneous subset of containers belonging to set $O$ with another subset of the same cardinality, but not necessarily homogenous, from set $D$. A subset is said to be homogeneous if it is composed of containers with the same destination. As containers with the same destination tend to appear together in the stowage plans, it seems natural to select homogeneous subsets from stack $s_1$ to be moved to other stack $s_2$. However, it would not be easy to find homogeneous subsets of the same cardinality in $s_2$ and therefore we allow any subset in this stack to be included in the move. The exchange is carried out according to the following steps:

1. Take $n$ containers from any group $G_j$ in stack $s_1$ with $j : |G_j| > 0$, $n \in \{1, \dots, |G_j|\}$. This set is denoted as $Q^1_{s_1}$.
2. Select another set of $n$ containers in stack $s_2$ with destinations different from $j$. This set does not have to be homogeneous and it is denoted as $Q^1_{s_2}$.
3. Let $O' = (O \setminus Q^1_{s_1}) \cup Q^1_{s_2}$ be the new set of containers to be loaded in stack $s_1$.
4. Let $D' = (O \setminus Q^1_{s_2}) \cup Q^1_{s_1}$ be the new set of containers to be loaded in stack $s_2$.
5. Place containers of $O'$ in stack $s_1$ and of $D'$ in stack $s_2$ in decreasing order of destination. Note that the remaining stacks $s \in \{1, \dots, S\} \setminus \{s_1, s_2\}$ maintain their initial configuration.
6. After the exchange between the two stacks, complete the solution with one of the constructive algorithms described above.

Figure 5.4 shows on the left-hand side the configuration at port 1 of one solution for an instance in which $R = 6$, $S = 4$, and $N = 10$. Each square represents a container and the number indicates the port of destination. The figure also shows the possible exchanges to be considered if $s_1 = 2$, $s_2 = 3$, and two containers from the group of 9-containers of $s_1$ ($G_9$) are chosen.



5 5 2 2
5 5 6 2
7 5 6 2
7 9 8 2
10 9 10 4
10 10 10 4
$s_1$ $s_2$
$Q_{s_1} = \{ 9\ 9 \}$
$Q_{s_2} = \{2\ 6\}\ \{6\ 8\}\ \{2\ 8\}\ \{6\ 10\}\ \{2\ 10\}\ \{8\ 10\}\ \{6\ 6\}\ \{10\ 10\}$

Figure 5.4: Example of neighboring solutions.

The local search algorithm starts from a candidate solution $x$ and then iteratively moves to a neighboring solution $x' \in N(x)$ following a first-improving strategy. The search starts with $s_1 = 1$ and all possible swaps with $s_2 \in \{2, \dots, S\}$ are checked until an improved neighboring solution is found. If an improvement is not obtained, it proceeds with $s_1 = 2$ and $s_2 \in \{3, \dots, S\}$, and continues until all pairs of stacks have been considered.

Algorithm 4 shows the pseudo-code of the procedure for exchanging containers between stacks $s_1$ and $s_2$ of a solution $x$. The objective function $f(x)$ is the total number of reshuffles. If Algorithm 4 returns $x$, it has not been able to obtain a better solution. If it finds a better solution $x'$, the search starts again with the new solution as the initial solution and with stacks $s_1 = 1$ and $s_2 = 2$. In line 6 Comb$(n; j; s_2)$ determines all the ways of selecting $n$ containers belonging to row $s_2$ with destinations different from $j$. For each selection, $Q^1_{s_2}$, a new neighboring solution is built.

**Algorithm 4** Exchanging containers between $s_1$ and $s_2$.

```
1: function Exchanges($x$, $s_1$, $s_2$)
2:     for $j = 2$ to $N$ do
3:         if $|G_j| > 0$ then
4:             for $n = 1$ to $|G_j|$ do
5:                 $Q^1_{s_1} = \cup^n_{i=1}\{j\}$;
6:                 Comb$(n, j, s_2)$;
7:                 for all $Q^1_{s_2}$ in Comb$(n, j, s_2)$ do
8:                     $x' \leftarrow$ Swap$(Q^1_{s_1}, s_1, Q^1_{s_2}, s_2)$;
9:                     if $f(x') < f(x)$ then
10:                        Return $x'$;
11:    Return $x$
```

## 5.6 Computational experiments

To test the performance of the mathematical model and the heuristic and metaheuristic algorithms, exact solutions were obtained by CPLEX 12.7 considering a time limit of 3600 seconds and heuristic algorithms were run with a time limit of 300 seconds or 1000 iterations. We conducted all computational experiments on virtual machines with 4 virtual processors and 16 GBytes of RAM running Windows 10 Enterprise 64 bits. Virtual machines are run in an OpenStack virtualization platform supported by 12 blades, each with four 12-core AMD

Opteron Abu Dhabi 6344 processors running at 2.6 GHz and 256 GB of RAM, for a total of 576 cores and 3 TBytes of RAM.

### 5.6.1 Test instances

We used two sets of instances in which full transportation matrices were randomly generated using the Authentic Matrices Generation Algorithm proposed by Ding and Chou (2015). For each set of parameters, $N$ (ports), $R$ (tiers), and $S$ (stacks), five different instances were generated. On the one hand, as the instances generated by Ding and Chou (2015) were not available, we generated Set I looking for large-size instances. On the other hand, Set II was generated looking for smaller but more computationally difficult instances. Studying Set I we observed that when the number of stacks was increased, the problems became easier, with an optimal solution of zero reshuffles in most cases, making it difficult to assess the relative efficiency of the algorithms. Therefore, we decided to generate Set II with fewer stacks but more tiers and ports so as to create instances in which the optimal solutions have many reshuffles, in order to obtain more information about the performance of the algorithms.

Set I: 625 instances. $N \in \{4, 6, \dots, 22\}$; $R = 10$; $S \in \{200, 300, \dots, 2000\}$;

Set II: 840 instances. $N \in \{4, 6, \dots, 16\}$; $R \in \{6, 8, \dots, 12\}$; $S \in \{2, 4, \dots, 12\}$

### 5.6.2 Performance of the integer programming model, ILR

All combinations of valid inequalities were tested in order to decide the best configuration of the integer linear model proposed. In the light of the results, we decided to use the basic model (expressions (5.2)-(5.7)), as well as valid inequalities (5.9) with $r' = R - 5$. We refer to this model as ILR.

We compared the performance of our mathematical formulation with the two existing models described in Avriel et al. (1998) and Ding and Chou (2015) by implementing and running all of them with the default CPLEX, with a time limit of 3600 CPU seconds and using 4 threads. Table 5.1 shows the results obtained by the three models on the second set of instances, Set II, whose main characteristic is that instances are smaller and have a greater number of reshuffles. The instances in Set I are too large for mathematical formulations. Although all models optimally solve instances of four ports, with zero reshuffles, they are not able to obtain an optimal solution when the number of ports increases.

In Table 5.1 all instances of 4 and 6 ports in Set II are optimally solved by the three models with averages of 0.1 and 0.6 reshuffles. For 8 ports, Avriel's and Ding and Chou's models provide the same number of optimal solutions and our model reaches 3 more optimal solutions. For larger numbers of ports, ILR outperforms the other models in terms of both the number of optimal solutions and the average number of reshuffles, with differences increasing as the number of ports increases. However, although the average number of reshuffles in our model is reduced by 74.8% compared to that of Avriel's model, it is still too high, because when the optimal solution is not found, the average number of reshuffles of the best solution found is sometimes very high. Over the 473 instances optimally solved by the three models,

Table 5.1: Comparing ILR with the state-of-the-art models from Avriel et al. (1998) and from Ding and Chou (2015) on Set II grouped by number of ports (N).

| | | Avriel et al. | | | Ding and Chou | | | ILR | | |
|---|---|---|---|---|---|---|---|---|---|---|
| N | # | #Opt | Res. | CPU | #Opt | Res. | CPU | #Opt | Res. | CPU |
| 4 | 120 | 120 | 0.1 | 0.2 | 120 | 0.1 | 0.2 | 120 | 0.1 | 0.1 |
| 6 | 120 | 120 | 0.6 | 5.7 | 120 | 0.6 | 22.7 | 120 | 0.6 | 1.1 |
| 8 | 120 | 115 | 1.2 | 241.8 | 115 | 1.2 | 311.8 | 118 | 1.2 | 116.7 |
| 10 | 120 | 90 | 2.5 | 1342.2 | 66 | 3.1 | 1927.6 | 99 | 2.5 | 1038.4 |
| 12 | 120 | 48 | 6.3 | 2516.1 | 35 | 26.8 | 2873.1 | 59 | 7.4 | 2172.6 |
| 14 | 120 | 28 | 64.1 | 3001.7 | 17 | 116.7 | 3178.4 | 36 | 19.1 | 2743.8 |
| 16 | 120 | 9 | 269.7 | 3359.8 | 7 | 276.5 | 3422.7 | 15 | 68.0 | 3266.6 |
| Total | 840 | 530 | 49.2 | 1495.3 | 480 | 60.7 | 1676.6 | 567 | 14.1 | 1334.2 |

The column # contains the number of instances tested in each group and row # Opt. the number of optimal solutions reached. The average number of reshuffles and average time in seconds are expressed in columns Resh. and CPU.

our model achieves a reduction in CPU time of 50.42% compared to Avriel's model and of 75.9% compared to Ding's model.

### 5.6.3 Selecting the best configuration for the GRASP algorithm

We first test the constructive algorithms developed for the multi-port container ship stowage problem and their randomized counterparts developed in this chapter. The GRASP presented here applies the best randomized version in the construction phase and the best constructive approach to complete the solutions in the improvement phase. We then study the effect of the local search by itself on the sets studied and we finally establish the stopping criteria used.

#### Performance of the constructive and randomized algorithms

We reimplemented the two constructive algorithms proposed by Avriel et al. (1998), SH, and Ding and Chou (2015), DCA, and their randomized versions, in C++. In the SH algorithm, there are two alternative stack selection rules, the function rule and the necessary shift rule. The authors tested these two rules and established that the function rule is better than the necessary shift rule if $R \leq 10$ and $N \geq 10$, and is worse in other cases. Therefore, we apply these two alternatives following that criterion.

As can be seen in Table 5.2, the average numbers of reshuffles obtained by the deterministic algorithms are very small in all cases in Set I and increase in Set II. The greater the number of ports and the smaller the number of stacks, the more difficult the problem. The results shown in the second part of Table 5.2 underline this statement. It can be seen that the average numbers of reshuffles for groups of 4 to 16 ports in Set II are higher than those for groups with a similar number of ports in Set I. Although the SH algorithm finds more solutions with 0 reshuffles than the DCA algorithm, solving 465 instances compared to 264 in Set I and 177 compared to 146 in Set II, DCA reduces the number of reshuffles by 56.3% and 11.1% in Sets I and II, respectively. Therefore, the constructive algorithm by Ding and Chou (2015) outperforms that by Avriel et al. (1998) for both sets.

With respect to the randomization strategies, Table 5.2 also shows the values of the solutions obtained by each randomized algorithm for the instances in both sets. It can

Table 5.2: Algorithm performance of the constructive algorithms from Avriel et al. (1998) and from Ding and Chou (2015) and their randomized versions on both sets grouped by number of ports (N).

| | | Avriel et al. | | | | | | Ding and Chou | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Deterministic, SH | | | Randomized | | | Deterministic | | | Randomized | | |
| N | # | # Z | Res. | CPU | # Z | Res. | CPU | # Z | Res. | CPU | # Z | Res. | CPU |
| Set I | | | | | | | | | | | | | |
| 4 | 95 | 95 | 0.0 | 0.0 | 95 | 0.0 | 0.0 | 86 | 0.2 | 0.1 | 95 | 0.0 | 0.8 |
| 6 | 95 | 94 | 0.0 | 0.0 | 94 | 0.0 | 0.1 | 69 | 0.6 | 0.1 | 93 | 0.1 | 5.0 |
| 8 | 95 | 86 | 0.3 | 0.0 | 87 | 0.3 | 0.5 | 63 | 0.7 | 0.1 | 91 | 0.1 | 5.1 |
| 10 | 95 | 73 | 1.5 | 0.0 | 74 | 1.3 | 4.4 | 16 | 2.1 | 0.1 | 82 | 0.2 | 33.2 |
| 12 | 95 | 55 | 2.6 | 0.0 | 58 | 2.3 | 8.7 | 22 | 3.0 | 0.1 | 71 | 0.5 | 48.7 |
| 14 | 95 | 35 | 5.8 | 0.0 | 39 | 4.0 | 18.7 | 4 | 5.2 | 0.1 | 58 | 1.0 | 89.8 |
| 16 | 95 | 13 | 10.1 | 0.0 | 19 | 6.6 | 39.8 | 2 | 5.9 | 0.1 | 41 | 1.7 | 123.0 |
| 18 | 95 | 9 | 16.5 | 0.0 | 14 | 11.1 | 74.2 | 1 | 7.7 | 0.1 | 35 | 2.1 | 136.9 |
| 20 | 95 | 5 | 29.4 | 0.0 | 8 | 18.0 | 115.5 | 0 | 11.2 | 0.1 | 8 | 4.9 | 190.5 |
| 22 | 95 | 0 | 45.5 | 0.1 | 2 | 30.9 | 208.5 | 1 | 12.7 | 0.1 | 16 | 6.1 | 172.1 |
| Total | 950 | 465 | 11.2 | 0.0 | 490 | 7.4 | 47.0 | 264 | 4.9 | 0.1 | 590 | 1.7 | 80.5 |
| Set II | | | | | | | | | | | | | |
| 4 | 120 | 116 | 0.1 | 0.0 | 116 | 0.1 | 0.0 | 104 | 0.2 | 0.0 | 110 | 0.1 | 0.0 |
| 6 | 120 | 47 | 2.4 | 0.0 | 54 | 1.7 | 0.0 | 33 | 2.3 | 0.0 | 58 | 1.4 | 0.0 |
| 8 | 120 | 13 | 5.6 | 0.0 | 23 | 3.1 | 0.1 | 9 | 5.9 | 0.0 | 22 | 3.5 | 0.1 |
| 10 | 120 | 1 | 11.1 | 0.0 | 9 | 6.4 | 0.1 | 0 | 11.1 | 0.0 | 1 | 7.7 | 0.2 |
| 12 | 120 | 0 | 18.3 | 0.0 | 0 | 10.4 | 0.2 | 0 | 15.7 | 0.0 | 0 | 10.6 | 0.2 |
| 14 | 120 | 0 | 25.3 | 0.0 | 1 | 15.9 | 0.3 | 0 | 21.1 | 0.0 | 0 | 15.2 | 0.3 |
| 16 | 120 | 0 | 32.1 | 0.0 | 0 | 20.0 | 0.4 | 0 | 27.8 | 0.0 | 0 | 20.0 | 0.4 |
| Total | 840 | 177 | 13.5 | 0.0 | 203 | 8.2 | 0.2 | 146 | 12.0 | 0.0 | 191 | 8.3 | 0.2 |

The column # shows the number of instances tested in each group, and column # Z the total number of solutions found with 0 reshuffles. The average number of reshuffles and average time in seconds are expressed in stacks Res. and CPU.

be observed that in all cases the average number of reshuffles decreases compared to their deterministic counterparts. The randomized version of the SH algorithm reduces the average number of reshuffles given by its deterministic version by 33.9% and 39.3% on Sets I and II, respectively. On the other hand, randomizing the order of the steps followed in the DCA algorithm reduces the average number of reshuffles by 65.3% on Set I and by 30.83% on Set II. According to these results, we select the algorithm by Ding and Chou (2015) and its randomization to be part of the GRASP.

## Performance of the local search

In order to study the effect of the local search described in Section 5.5.2 by itself, before including it in the GRASP algorithm, we applied it to the solutions obtained using the DCA algorithm. Once the new configuration for port 1 has been obtained, the solution is completed using the same algorithm. Table 5.3 shows the results obtained on both sets. Besides the original local search, denoted in the table as LS I, and in order to reduce its running time, we also include a modified version, LS II, in which, given a group $G_j : |G_j| > 0$ in stack $s_1$, instead of considering the exchanges of any number of containers $n \in \{1, \dots, |G_j|\}$ we only consider the exchange of $n = 1$ or $n = |G_j|$ $j$-containers.

Two points are noticeable in this table. First, the local search is able to find solutions for all the groups with a significantly lower average number of reshuffles, from 4.9 to 3.31 on Set I, and from 12 to 4.83 on Set II. Second, with the modified version of the local search, the average number of reshuffles is reduced by 24.8% on Set I and slightly increased by 2.1%

Table 5.3: Comparing local search procedures on both sets grouped by number of ports.

| | | LSI | | | LSII | | |
|---|---|---|---|---|---|---|---|
| N | # | # Z | Resh. | CPU | # Z | Resh. | CPU |
| Set I | | | | | | | |
| 4 | 95 | 94 | 0.01 | 8.68 | 91 | 0.04 | 13.68 |
| 6 | 95 | 83 | 0.20 | 51.25 | 86 | 0.13 | 37.82 |
| 8 | 95 | 89 | 0.11 | 44.60 | 93 | 0.03 | 17.85 |
| 10 | 95 | 55 | 0.76 | 169.43 | 74 | 0.40 | 93.59 |
| 12 | 95 | 40 | 1.61 | 193.09 | 50 | 1.20 | 161.20 |
| 14 | 95 | 23 | 3.18 | 252.07 | 28 | 2.32 | 235.39 |
| 16 | 95 | 11 | 4.15 | 274.81 | 20 | 2.97 | 252.85 |
| 18 | 95 | 4 | 5.73 | 290.97 | 11 | 4.14 | 279.48 |
| 20 | 95 | 1 | 8.19 | 299.35 | 2 | 6.28 | 296.89 |
| 22 | 95 | 1 | 9.21 | 298.85 | 3 | 7.39 | 295.80 |
| Total | 950 | 401 | 3.31 | 188.31 | 458 | 2.49 | 168.46 |
| Set II | | | | | | | |
| 4 | 120 | 115 | 0.07 | 0.00 | 115 | 0.07 | 0.00 |
| 6 | 120 | 78 | 0.81 | 0.08 | 81 | 0.79 | 0.01 |
| 8 | 120 | 53 | 1.69 | 0.60 | 54 | 1.79 | 0.12 |
| 10 | 120 | 16 | 3.81 | 3.10 | 14 | 4.01 | 0.79 |
| 12 | 120 | 6 | 6.00 | 5.90 | 7 | 6.02 | 1.66 |
| 14 | 120 | 4 | 8.76 | 7.77 | 4 | 8.74 | 2.65 |
| 16 | 120 | 0 | 12.71 | 15.22 | 1 | 13.08 | 4.00 |
| Total | 840 | 272 | 4.83 | 4.67 | 276 | 4.93 | 1.32 |

The column # shows the number of instances tested in each group, and column # Z the total number of solutions found with 0 reshuffles. The average number of reshuffles and average time in seconds are expressed in columns Res. and CPU.

on Set II. Nevertheless, the running times are reduced by 10.5% and 71.7% on Sets I and II, respectively. We therefore select this LS II version for the final configuration of the GRASP algorithm.

## 5.6.4 Results of the complete GRASP algorithm

Table 5.4 presents the aggregated results of GRASP for each group in Set I and Set II and summarizes the number of times that GRASP achieves a solution of zero reshuffles and the number of times that it achieves optimal solutions, either because they match an optimal solution provided by one or more of the IP models or because the solution has zero reshuffles. Differences between columns #Z and #Opt. only appear in Set II, on which the integer models have been tested.

With respect to Set I, GRASP is able to solve the majority of these instances, 603 out of 950, to proven optimality. For the remaining instances, the number of reshuffles in the solutions is greater than zero. Although their optimality cannot be proven, it is possible that some of them are in fact optimal, further increasing the total number of optimal solutions found by GRASP.

The instances from 4 to 14 ports are solved with an average number of reshuffles lower than or equal to 1 in an average time of 49.47 seconds. The instances with 16 and 18 ports are solved with an average lower than 2 and those of 20 and 22 ports with averages of 4.1 and 4.3 reshuffles. The average number of reshuffles in Set I obtained with GRASP is just 1.3, while that obtained with the DCA algorithm, the best according to Section 5.6.3, is 4.9. The results obtained with GRASP are also better than that obtained with the randomized version of DCA, 1.3 compared to 1.7 reshuffles, and with the local search technique *LSII*,

Table 5.4: Results obtained with the GRASP algorithm on both sets grouped by number of ports (N).

| | | GRASP | | | |
|---|---|---|---|---|---|
| N | # | #Z | #Opt. | Resh. | CPU |
| Set I | | | | | |
| 4 | 95 | 95 | 95 | 0.0 | 1.1 |
| 6 | 95 | 93 | 93 | 0.1 | 10.7 |
| 8 | 95 | 93 | 93 | 0.0 | 8.5 |
| 10 | 95 | 84 | 84 | 0.1 | 48.0 |
| 12 | 95 | 72 | 72 | 0.4 | 87.7 |
| 14 | 95 | 54 | 54 | 1.0 | 140.8 |
| 16 | 95 | 41 | 41 | 1.6 | 183.1 |
| 18 | 95 | 39 | 39 | 1.8 | 194.8 |
| 20 | 95 | 12 | 12 | 4.1 | 271.9 |
| 22 | 95 | 20 | 20 | 4.3 | 253.7 |
| Total | 950 | 603 | 603 | 1.3 | 120.0 |
| Set II | | | | | |
| 4 | 120 | 116 | 117 | 0.1 | 0.0 |
| 6 | 120 | 88 | 89 | 0.7 | 4.5 |
| 8 | 120 | 70 | 74 | 1.3 | 32.3 |
| 10 | 120 | 31 | 33 | 3.1 | 115.7 |
| 12 | 120 | 14 | 14 | 5.0 | 168.1 |
| 14 | 120 | 14 | 15 | 7.6 | 193.1 |
| 16 | 120 | 6 | 9 | 11.6 | 217.1 |
| Total | 840 | 339 | 351 | 4.2 | 104.4 |

Column # shows the number of instances tested in each group, column #Z the total number of solutions found with 0 reshuffles, and Opt. the total number of optimal solutions. The average number of reshuffles and average time in seconds are expressed in columns Res. and CPU.

1.3 compared to 2.49. As expected, the stopping criterion that prevails when the number of stacks and ports is so high is the time limit of 300 seconds.

The situation with Set II is somewhat different, because this set is composed of instances with larger numbers of reshuffles. The average number of reshuffles is more than twice that of Set I and the percentage of solutions with zero reshuffles decreases from 63.47% to 40.35%. While the average number of reshuffles for this set with the DCA algorithm is 12, and 8.3 with its randomized version, the average with GRASP is 4.2 reshuffles. With up to 10 ports, the new model, ILR, provides slightly better solutions than GRASP, but for instances with 12 ports, GRASP outperforms the model with an average of 5 reshuffles instead of 7.4. The difference increases as the number of ports increases; in instances with 16 ports GRASP obtains an average of 11.6 reshuffles while for the model it is 68. Although the time limit was set at 300 seconds, the average time for all groups of instances in Set II is much shorter. In fact, GRASP finishes after 300 seconds only in 6 instances of 8 ports, 45 of 10 ports, 51 of 12 ports, 55 of 14 ports, and 83 of 16 ports. In most cases, the algorithm stops after 1000 iterations.

The results show the effectiveness of the GRASP we propose in this study. It obtains extremely good solutions in very short computing times, less than 5 minutes for the larger instances. These solutions are clearly better than those provided by other heuristic methods, which could be used to obtain acceptable solutions if extremely fast answers were needed. Integer models can only be used for small instances, and even in the cases in which the models can prove optimality, the solutions obtained by GRASP match or are very close to the optimal solutions. For medium and large-size instances, the integer models fail to obtain feasible solutions in many cases, and when they do obtain feasible solutions these are much worse than those provided by GRASP.

## 5.7 Concluding remarks

Efficient stowage plans for increasingly larger container ships have a critical impact on the performance of operations at container terminals. The problem is being studied from various different perspectives, using a range of decomposition strategies and algorithmic approaches. In this article we have chosen to study the complete multi-port problem, considering loading and unloading operations along the route, and have studied it from two complementary points of view. First, we have developed a new integer model and compared it with existing proposals in an extensive computational study. The conclusion is that, though our model performs consistently better than previous models, none of them is able to produce acceptable solutions for large-size instances. Therefore, in a second part we have developed a GRASP algorithm with new features, such as a randomizing strategy that randomizes the steps of the existing constructive algorithms. A new local search, tailored to the problem, improves the solutions of the randomized constructive phase and allows the GRASP to obtain many optimal solutions and extremely good solutions for instances for which optimality cannot be proven, clearly outperforming previous heuristic algorithms. Our conclusion is that the GRASP framework is very well suited to this problem because using a constructive algorithm is a good way of studying the sequence of operations in which the empty spaces left when unloading containers are then used in the loading operations. By adding randomization and a local search phase we achieve a very efficient algorithm for this complex problem.

# Chapter 6

# Mathematical models and matheuristics for a generalized multi-port CSPP

## 6.1 Introduction

This chapter aims to fill the gap in the literature between decomposed problems with realistic constraints and simplified integrated problems by providing methods to solve the combined CSPP with container weight- and size-related constraints. To the best of our knowledge, the CSPP with these constraints has never been addressed with integrated methods in the literature. Our contribution is threefold: (1) We introduce three generalizations of the CSPP, which come with several additional complexities: (i) the CSPP with 20-foot and 40-foot containers where specific constraints are present, (ii) the CSPP with identical container size but different weights and stability constraints, and (iii) the CSPP with 20' and 40' containers, container weights, and stability constraints. (2) We provide IP formulations for the CSPP and the newly introduced generalizations. (3) Using the decomposable structure of these mathematical models, we develop three matheuristic approaches, namely, relax-and-fix, insert-and-fix, and fractional-relax-and-fix heuristics, to solve the most general problem for instances with up to 20000 containers, which is approximately the capacity of the largest ships available nowadays.

The remainder of this chapter is organized as follows: Section 6.2 describes the problems we study and defines the notation we use throughout the chapter. Section 6.3 presents the mathematical models that we introduce for the aforementioned variants. Section 6.4 provides the details of the three matheuristic algorithms that we have developed. In this section, we also introduce a constructive procedure integrated into the fractional-relax-and-fix algorithm, which guarantees obtaining a feasible solution even when the solvers cannot provide one within the time limit. Section 6.5 details the computational experiments we conducted on our mathematical models and matheuristic algorithms. Finally, we present our conclusions in Section 6.6.

## 6.2 Problem description

Container stowage plans use a bay-row-tier coordinate system to identify the position of a container aboard the ship that was described in Section 5.3. Each position defined by these three coordinates is called a *cell* and it can contain two 20' or one 40' container as shown in Figure 6.1. We therefore can use a fourth coordinate to identify each *slot* with capacity for one 20' container.

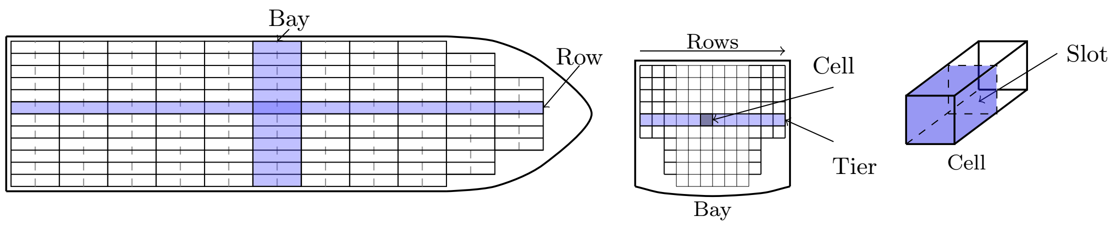


Figure 6.1: Container ship scheme, showing the bay-row-tier-slot coordinate system.

We consider a ship that visits several ports where container loading and unloading operations are carried out. The route is circular, starting and ending at a given port. The boat is empty before the route starts and also when the route is completed. The capacity and structure of the ship are known and we are also provided with the details of the containers that are loaded at each port, with their weight and size and their port of discharge. We thus have to decide the stowage plan of the ship along the whole route in order to minimize the total number of reshuffles. Reshuffles are required at a port for unloading containers placed below other containers that do not have to be unloaded at this port and for loading containers in slots below other existing containers. A feasible solution must satisfy the following rules:

a) No container can hang in the air. If a slot has a container, the slots below it must also have containers.

b) 20' containers cannot be loaded on top of 40' containers, but one 40' container can be stacked on top of two 20' containers ("Russian stacks" or "mixed stacks").

c) A cell must be either empty or full (both slots occupied).

d) Stability constraints, which are expressed in terms of maximum tolerance in the difference of weights between horizontal and vertical cross sections, must be satisfied.

Let us consider a ship with capacity $\Omega$ TEUs traveling along a trade route consisting of $N$ ports. The ship has $B$ bays, where each bay $b$ contains $C^b$ rows, and each row $c$ of bay $b$ has $R^{bc}$ cells with 2 slots each. Let $\mathcal{B}$ be the index set of the bays, $\mathcal{C}^b$ the index set of the rows in bay $b$, $\mathcal{R}^{bc}$ the index set of the tiers in row $c$ of bay $b$, and $\mathcal{X}$ the set of the slots at each cell. The total capacity of the ship can be calculated as $\Omega = \sum_{b\in\mathcal{B}}\sum_{c\in\mathcal{C}^b}\sum_{r\in\mathcal{R}^{bc}}\sum_{x\in\mathcal{X}}(b \times c \times r \times x)$. We consider two sizes of containers $\mathcal{S}$, 20' and 40', and three different weight classes $\mathcal{M}$, depending on whether the containers are light, medium, or heavy. Table 6.1 shows the notation used in the following sections.

Table 6.1: Notation employed throughout the chapter.

| | |
|---|---|
| $\Omega$ | Container ship capacity, which is expressed in terms of the total number of slots. |

| | |
|---|---|
| $\mathcal{N}$ | Index set of ports. $\mathcal{N} = \{1, \ldots, N\}$ |
| $\mathcal{O}$ | Ports where loading operations can occur. $\mathcal{O} = \{1, \ldots, N-1\} \subseteq \mathcal{N}$ |
| $\mathcal{O}^r$ | Ports where relocations can occur. $\mathcal{O}^r = \{2, \ldots, N-1\} \subseteq \mathcal{O} \subseteq \mathcal{N}$ |
| $\mathcal{D}^i$ | Set of possible destination ports for containers loaded at port $i$. $\mathcal{D}^i = \{i+1, \ldots, N\}$ |
| $\mathcal{B}$ | Index set of bays. $\mathcal{B} = \{1, \ldots, B\}$ |
| $\mathcal{C}^b$ | Index set of rows of bay $b$. $\mathcal{C}^b = \{1, \ldots, C^b\}$ |
| $\mathcal{R}^{bc}$ | Index set of tiers of bay $b$, row $c$. $\mathcal{R}^{bc} = \{1, \ldots, R^{bc}\}$ |
| $\mathcal{P}$ | Set of cells. $\mathcal{P} = \{p = (b, c, r) : b \in \mathcal{B}, c \in \mathcal{C}^b, r \in \mathcal{R}^{bc}\}$ |
| $\mathcal{X}$ | Index set of slots. $\mathcal{X} = \{1, 2\}$ |
| $\mathcal{HP}$ | Set of slots belonging to the bow end. |
| $\mathcal{LP}$ | Set of slots belonging to the stern end. |
| $\mathcal{WP}$ | Set of slots belonging to the starboard side. |
| $\mathcal{YP}$ | Set of slots belonging to the port side. |
| $\mathcal{A}$ | Areas of the ship, depending on their longitudinal and lateral position: port quarter (back left), starboard quarter (back right), port bow (front left), and starboard bow (front right). $\mathcal{A} = \{\{\mathcal{LP}\} \cap \{\mathcal{YP}\}, \{\mathcal{LP}\} \cap \{\mathcal{WP}\}, \{\mathcal{HP}\} \cap \{\mathcal{YP}\}, \{\mathcal{HP}\} \cap \{\mathcal{WP}\}\}, \quad \lvert\mathcal{A}\rvert = 4$ |
| $\mathcal{M}$ | Set of weight classes (light, medium and heavy). |
| $\mathcal{S}$ | Set of container sizes (20' and 40'). |
| $\mathcal{T}$ | Set of 20' containers. |
| $\mathcal{F}$ | Set of 40' containers. |
| $\mathcal{T}_i$ | Set of 20' containers loaded at port i. |
| $\mathcal{F}_i$ | Set of 40' containers loaded at port i. |
| $T_{ij}$ | Number of 20' containers going from port $i$ to port $j$. |
| $F_{ij}$ | Number of 40' containers going from port $i$ to port $j$. |
| $T_{ij}^m$ | Number of 20' containers of weight class $m$ going from port $i$ to port $j$. |
| $F_{ij}^m$ | Number of 40' containers of weight class $m$ going from port $i$ to port $j$. |
| $E_i$ | Number of empty slots after loading operations at port $i$. |
| $W_i$ | Total weight on board after loading operations at port $i$. |
| $w_m^T$ | Weight of a 20' container of class $m$. |
| $w_m^F$ | Weight of a 40' container of class $m$. |
| $W^a$ | Total weight on board in area $a$. |
| $Q_1$ | Maximum percentage of weight deviation allowed between bow and stern ends. |
| $Q_2$ | Maximum percentage of weight deviation allowed between starboard and port sides. |

Since the slots of a cell must be both occupied or both empty, the number of empty slots after loading operations at each port has to be even. Note that a cell can be represented by index $p \in \mathcal{P}$ or by triple $(b, c, r)$, such that $b \in \mathcal{B}, c \in \mathcal{C}^b, r \in \mathcal{R}^{bc}$. We use either one representation or the other, depending on the set in which the cells vary.

## 6.3 Integer programming models

To the best of our knowledge, the CSPP with different container sizes and stability constraints has never been addressed with integrated methods in the literature. In this section we present three models that solve the problem by relaxing some of its assumptions and another model, referred to as BSW, that considers all of them.

### 6.3.1 A base model with 20' containers and no stability constraints: Base model

We first present the Base model, which addresses the problem by considering a single container size (20') and no stability constraints, so no weight classes are involved. The three types of binary variables involved in the model are:

$$t_{ij}(p,x) = \begin{cases} 1, & \text{if one 20' container with destination port } j \text{ is stowed in slot } (p,x) \text{ after} \\ & \text{the loading operations at port } i \\ 0, & \text{otherwise} \end{cases} \tag{6.1}$$

$\forall i \in \mathcal{O}; \quad \forall j \in \mathcal{D}^i; \quad \forall p \in \mathcal{P}; \quad \forall x \in \mathcal{X};$

$$rt_i(p,x) = \begin{cases} 1, & \text{if one 20' container stowed in slot } (p,x) \text{ at port } i-1 \text{ is} \\ & \text{relocated at port } i \\ 0, & \text{otherwise} \end{cases} \tag{6.2}$$

$\forall i \in \mathcal{O}^r; \quad \forall p \in \mathcal{P}; \quad \forall x \in \mathcal{X};$

$$e_i(p,x) = \begin{cases} 1, & \text{if slot } (p,x) \text{ is empty after the loading operations at port } i \\ 0, & \text{otherwise} \end{cases} \tag{6.3}$$

$\forall i \in \mathcal{O}; \quad \forall p \in \mathcal{P}; \quad \forall x \in \mathcal{X};$

With these variables and the parameters described in the previous section, we can formulate the problem as follows:

$$\text{Min} \sum_{i\in\mathcal{O}^r}\sum_{p\in\mathcal{P}}\sum_{x\in\mathcal{X}} rt_i(p,x) \tag{6.4}$$

$$t_{i-1j}(p,x) - t_{ij}(p,x) \le rt_i(p,x) \quad \forall i \in \mathcal{O}^r,\ \forall j \in \mathcal{D}^i,\ \forall p \in \mathcal{P},\ \forall x \in \mathcal{X} \tag{6.5}$$

$$t_{i-1i}(b,c,r,x) + rt_i(b,c,r,x) \le t_{i-1i}(b,c,r+1,x) + rt_i(b,c,r+1,x) + e_{i-1}(b,c,r+1,x)$$
$$\forall i \in \mathcal{O}^r,\ \forall b \in \mathcal{B},\ \forall c \in \mathcal{C}^b,\ \forall r \in \mathcal{R}^{bc} \setminus \{R^{bc}\},\ \forall x \in \mathcal{X} \tag{6.6}$$

$$\sum_{p\in\mathcal{P}}\sum_{x\in\mathcal{X}} t_{i(i+1)}(p,x) = \sum_{k=1}^{i} T_{k(i+1)} \forall i \in \mathcal{O} \tag{6.7}$$

$$\sum_{p\in\mathcal{P}}\sum_{x\in\mathcal{X}} t_{ij}(p,x) = \sum_{p\in\mathcal{P}}\sum_{x\in\mathcal{X}} t_{(i+1)j}(p,x) - T_{(i+1)j} \forall i \in \mathcal{O}\setminus\{N-1\},\ \forall j \in \mathcal{D}^i \setminus \{i+1\} \tag{6.8}$$

$$\sum_{p\in\mathcal{P}}\sum_{x\in\mathcal{X}} e_i(p,x) = E_i \qquad \forall i \in \mathcal{O} \tag{6.9}$$

$$\sum_{j\in\mathcal{D}^i} t_{ij}(p,x) + e_i(p,x) = 1 \qquad \forall i \in \mathcal{O},\ \forall p \in \mathcal{P},\ \forall x \in \mathcal{X} \tag{6.10}$$

$$\sum_{j\in\mathcal{D}^i} t_{ij}(p,1) = \sum_{j\in\mathcal{D}^i} t_{ij}(p,2) \qquad \forall i \in \mathcal{O},\ \forall p \in \mathcal{P} \tag{6.11}$$

$$e_i(b,c,r,1) \le e_i(b,c,r+1,1) \qquad \forall i \in \mathcal{O},\ \forall b \in \mathcal{B},\ \forall c \in \mathcal{C}^b,\ \forall r \in \mathcal{R}^{bc} \setminus \{R^{bc}\} \tag{6.12}$$

The objective function (6.4) minimizes the number of relocations along the ship's route. Constraints (6.5) and (6.6) identify these unproductive moves. If, prior to unloading operations at port $i$, one slot is occupied by one container whose port of discharge is not $i$, and it is no longer there after the loading operations, this container has been relocated at port $i$ (constraints (6.5)). When visiting a port $i$, if there is a container to be discharged in this port from a certain cell, the cells above it must be occupied by containers going to this port or must be empty; otherwise, the containers above them will have to be relocated (constraints (6.6)). Similarly, if a container is relocated at port $i$ and there is another container above it, the latter is also relocated, unless it is going to port $i$. Constraints (6.7)-(6.9) set the number of containers to be loaded in each port. Upon leaving port $i$, the number of containers to be unloaded at the next port $i+1$ equals the total number of containers going to port $i+1$ loaded since the beginning of the route, by constraints (6.7). By constraints (6.8), the number of containers to be unloaded at any port after $i+1$, upon leaving port $i+1$, equals the number of containers on board the ship when it leaves port $i$ plus those that will be loaded at $i+1$. Finally, when leaving a port $i$, the number of empty slots is $E_i$, by constraints (6.9). Constraints (6.10) ensure that a slot is either empty or occupied by one container. Constraints (6.11) ensure that a cell is either empty or both of its slots are occupied, and together with (6.12) ensure that no containers can be left hanging.

Since the number of empty slots plus the number of containers on board when leaving a given port $i$ equals the container ship capacity $\Omega$ and each slot is either empty or occupied, we can also consider constraints (6.7) to (6.10) as inequalities, as follows:

$$\sum_{p\in\mathcal{P}}\sum_{x\in\mathcal{X}} t_{i(i+1)}(p,x) \geq \sum_{k=1}^{i} T_{k(i+1)} \qquad \forall i \in \mathcal{O} \tag{6.13}$$

$$\sum_{p\in\mathcal{P}}\sum_{x\in\mathcal{X}} t_{ij}(p,x) \geq \sum_{p\in\mathcal{P}}\sum_{x\in\mathcal{X}} t_{(i+1)j}(p,x) - T_{(i+1)j} \qquad \forall i \in \mathcal{O}\setminus\{N-1\},\ \forall j \in \mathcal{D}^i\setminus\{i+1\} \tag{6.14}$$

$$\sum_{p\in\mathcal{P}}\sum_{x\in\mathcal{X}} e_i(p,x) \geq E_i \qquad \forall i \in \mathcal{O} \tag{6.15}$$

$$\sum_{j\in\mathcal{D}^i} t_{ij}(p,x) + e_i(p,x) \leq 1 \qquad \forall i \in \mathcal{O},\ \forall p \in \mathcal{P},\ \forall x \in \mathcal{X} \tag{6.16}$$

### 6.3.2 A new model considering container weights and stability constraints: BW model

To consider the stability constraints, but only one container size, we present the BW model. It uses the same definition for $rt_i(p,x)$ and $e_i(p,x)$ variables as before, but introduces an additional sub-index to the $t$ variables to identify the weight class of the container as follows:

$$t_{ijm}(p,x) = \begin{cases} 1, & \text{if one 20' container of weight class } m \text{ with destination port } j \text{ is stowed} \\ & \text{in slot } (p,x) \text{ after the loading operations at port } i \\ 0, & \text{otherwise} \end{cases} \tag{6.17}$$

$\forall i \in \mathcal{O};\quad \forall j \in \mathcal{D}^i;\quad \forall m \in \mathcal{M};\quad \forall p \in \mathcal{P};\quad \forall x \in \mathcal{X}.$

Using these variables, the BW model is:

$$\text{Min} \sum_{i\in\mathcal{O}^r}\sum_{p\in\mathcal{P}}\sum_{x\in\mathcal{X}} rt_i(p,x) \tag{6.18}$$

$$t_{i-1jm}(p,x) - t_{ijm}(p,x) \leq rt_i(p,x) \qquad \forall i \in \mathcal{O}^r,\ \forall j \in \mathcal{D}^i,\ \forall m \in \mathcal{M},\ \forall p \in \mathcal{P},\ \forall x \in \mathcal{X} \tag{6.19}$$

$$\sum_{m \in \mathcal{M}} t_{i-1im}(b,c,r,x) + rt_i(b,c,r,x) \leq \sum_{m \in \mathcal{M}} t_{i-1im}(b,c,r+1,x) + rt_i(b,c,r+1,x) + e_i(b,c,r+1,x)$$
$$\forall i \in \mathcal{O}^r,\ \forall b \in \mathcal{B},\ \forall c \in \mathcal{C}^b,\ \forall r \in \mathcal{R}^{bc} \setminus \{R^{bc}\},\ \forall x \in \mathcal{X} \tag{6.20}$$

$$\sum_{p \in \mathcal{P}} \sum_{x \in \mathcal{X}} t_{i(i+1)m}(p,x) \geq \sum_{k=1}^{i} T^m_{k(i+1)} \qquad \forall i \in \mathcal{O},\ \forall m \in \mathcal{M} \tag{6.21}$$

$$\sum_{p \in \mathcal{P}} \sum_{x \in \mathcal{X}} t_{ijm}(p,x) \geq \sum_{p \in \mathcal{P}} \sum_{x \in \mathcal{X}} t_{(i+1)jm}(p,x) - T^m_{(i+1)j}$$
$$\forall i \in \mathcal{O},\ \forall m \in \mathcal{M} \setminus \{N-1\},\ \forall j \in \mathcal{D}^i \setminus \{i+1\} \tag{6.22}$$

$$\sum_{p \in \mathcal{P}} \sum_{x \in \mathcal{X}} e_i(p,x) \geq E_i \qquad \forall i \in \mathcal{O} \tag{6.23}$$

$$\sum_{j \in \mathcal{D}^i} \sum_{m \in \mathcal{M}} t_{ijm}(p,x) + e_i(p,x) \leq 1 \qquad \forall i \in \mathcal{O},\ \forall p \in \mathcal{P},\ \forall x \in \mathcal{X} \tag{6.24}$$

$$\sum_{j \in \mathcal{D}^i} \sum_{m \in \mathcal{M}} t_{ijm}(p,1) = \sum_{j \in \mathcal{D}^i} \sum_{m \in \mathcal{M}} t_{ijm}(p,2) \quad \forall i \in \mathcal{O},\ \forall p \in \mathcal{P} \tag{6.25}$$

$$e_i(b,c,r,1) \leq e_i(b,c,r+1,1) \qquad \forall i \in \mathcal{O},\ \forall b \in \mathcal{B},\ \forall c \in \mathcal{C}^b,\ \forall r \in \mathcal{R}^{bc} \setminus \{R^{bc}\} \tag{6.26}$$

$$-Q_1 W_i \leq \sum_{j \in \mathcal{D}^i} \sum_{m \in \mathcal{M}} \Big( \sum_{p,x \in \mathcal{LP}} w^T_m t_{ijm}(p,x) - \sum_{p,x \in \mathcal{HP}} w^T_m t_{ijm}(p,x) \Big) \leq Q_1 W_i \qquad \forall i \in \mathcal{O} \tag{6.27}$$

$$-Q_2 W_i \leq \sum_{j \in \mathcal{D}^i} \sum_{m \in \mathcal{M}} \Big( \sum_{p,x \in \mathcal{WP}} w^T_m t_{ijm}(p,x) - \sum_{p,x \in \mathcal{YP}} w^T_m t_{ijm}(p,x) \Big) \leq Q_2 W_i \qquad \forall i \in \mathcal{O} \tag{6.28}$$

The formulation is similar to that of the previous model, adding in each case the weight class of each container. The objective function (6.18) minimizes the number of relocations. Constraints (6.19)-(6.20) identify them and constraints (6.21)-(6.23) identify the number of containers for each destination and in each weight class that must be on board when leaving each port as well as the number of empty slots. Constraints (6.24)-(6.26) enforce the rules regarding stowage: each slot holds at most one container, a cell must be either empty or full, and no containers can hang in the air. Finally, constraints (6.27)-(6.28) handle the stability of the ship: the difference between the total weight in the bow and stern ends (constraints (6.27)) and between the starboard and port sides (constraints (6.28)) when leaving each port must be below a given percentage of the total weight on board the ship.

### 6.3.3 A new model with 20' and 40' containers and no stability constraints: BS model

We now present the BS model, which considers two container sizes (20' and 40') but no stability constraints, so the sub-index for the weight class is no longer needed. It uses the definition of the $t_{ij}(p,x)$ and $rt_i(p,x)$ variables in the Base model, and new variables for the positions of 40' containers and their relocations.

$$f_{ij}(p) = \begin{cases} 1, & \text{if one 40' container with destination port } j \text{ is stowed in cell } p \\ & \text{after the loading operations at port } i \\ 0, & \text{otherwise} \end{cases} \tag{6.29}$$

$\forall i \in \mathcal{O};\quad \forall j \in \mathcal{D}^i;\quad \forall p \in \mathcal{P};$

$$rf_i(p) = \begin{cases} 1, & \text{if one 40' container stowed in cell } p \text{ at port } i-1 \text{ is relocated} \\ & \text{at port } i \\ 0, & \text{otherwise} \end{cases} \tag{6.30}$$
$$\forall i \in \mathcal{O}^r; \quad \forall p \in \mathcal{P};$$

We consider here, without loss of generality, that the ship is full after loading operations at each port $i$ by adding $E_i/2$ 40' containers from each port $i$ to $i+1$. These containers will not produce reshuffles and will be assigned to the upper positions. So, in this model we do not need to declare $e$ variables and we are considering $F_{i,i+1}$ as $F_{i,i+1} + E_i/2 \quad \forall i \in \mathcal{O}$.

$$\text{Min} \sum_{i \in \mathcal{O}^r} \sum_{p \in \mathcal{P}} \Big( \sum_{x \in \mathcal{X}} rt_i(p,x) + rf_i(p) \Big) \tag{6.31}$$

$$f_{i-1j}(p) - f_{ij}(p) \le rf_i(p) \qquad \forall i \in \mathcal{O}^r,\ \forall j \in \mathcal{D}^i,\ \forall p \in \mathcal{P} \tag{6.32}$$

$$t_{i-1j}(p,x) - t_{ij}(p,x) \le rt_i(p,x) \qquad \forall i \in \mathcal{O}^r,\ \forall j \in \mathcal{D}^i,\ \forall p \in \mathcal{P},\ \forall x \in \mathcal{X} \tag{6.33}$$

$$\begin{aligned} & t_{i-1i}(b,c,r,x) + f_{i-1i}(b,c,r) + rt_i(b,c,r,x) + rf_i(b,c,r) \le \\ & \qquad t_{i-1i}(b,c,r+1,x) + f_{i-1i}(b,c,r+1) + rt_i(b,c,r+1,x) + rf_i(b,c,r+1) \\ & \qquad\qquad \forall i \in \mathcal{O}^r,\ \forall b \in \mathcal{B},\ \forall c \in \mathcal{C}^b,\ \forall r \in \mathcal{R}^{bc} \setminus \{R^{bc}\},\ \forall x \in \mathcal{X} \end{aligned} \tag{6.34}$$

$$rf_i(p) + \sum_{j \in \mathcal{D}^{i-1}} t_{i-1j}(p,1) + f_{i-1i}(p) \le 1 \qquad \forall i \in \mathcal{O} \setminus \{1\},\ \forall p \in \mathcal{P} \tag{6.35}$$

$$\sum_{p \in \mathcal{P}} \sum_{x \in \mathcal{X}} t_{i(i+1)}(p,x) \ge \sum_{k=1}^{i} T_{k(i+1)} \qquad \forall i \in \mathcal{O} \tag{6.36}$$

$$\sum_{p \in \mathcal{P}} f_{i(i+1)}(p) \ge \sum_{k=1}^{i} F_{k(i+1)} \qquad \forall i \in \mathcal{O} \tag{6.37}$$

$$\sum_{p \in \mathcal{P}} \sum_{x \in \mathcal{X}} t_{ij}(p,x) \ge \sum_{p \in \mathcal{P}} \sum_{x \in \mathcal{X}} t_{(i+1)j}(p,x) - T_{(i+1)j} \qquad \forall i \in \mathcal{O} \setminus \{N-1\},\ \forall j \in \mathcal{D}^i \setminus \{i+1\} \tag{6.38}$$

$$\sum_{p \in \mathcal{P}} f_{ij}(p) \ge \sum_{p \in \mathcal{P}} f_{(i+1)j}(p) - F_{(i+1)j} \qquad \forall i \in \mathcal{O} \setminus \{N-1\},\ \forall j \in \mathcal{D}^i \setminus \{i+1\} \tag{6.39}$$

$$\sum_{j \in \mathcal{D}^i} t_{ij}(p,1) + \sum_{j \in \mathcal{D}^i} f_{ij}(p) \le 1 \qquad \forall i \in \mathcal{O},\ \forall p \in \mathcal{P} \tag{6.40}$$

$$\sum_{j \in \mathcal{D}^i} t_{ij}(p,2) \le \sum_{j \in \mathcal{D}^i} t_{ij}(p,1) \qquad \forall i \in \mathcal{O},\ \forall p \in \mathcal{P} \tag{6.41}$$

$$\sum_{j \in \mathcal{D}^i} t_{ij}(b,c,r+1,2) \le \sum_{j \in \mathcal{D}^i} t_{ij}(b,c,r,2) \qquad \forall i \in \mathcal{O},\ \forall b \in \mathcal{B},\ \forall c \in \mathcal{C}^b,\ \forall r \in \mathcal{R}^{bc} \setminus \{R^{bc}\} \tag{6.42}$$

The objective function (6.31) minimizes the number of relocations of 20' and 40' containers. When a cell is occupied, before unloading operations at port $i$, by a 40' container whose destination is not that port and it is no longer there after the loading operations, there has been an unproductive relocation in that cell by constraints (6.32). Likewise for 20' containers and slots by constraints (6.33). If a container going to port $i$ occupies a given cell at that port, then either the cells above it are occupied by containers going to that port or the containers in those positions will be relocated, by constraints (6.34). Similarly, if a container is relocated at port $i$, the upper slot is either occupied by a container going to that port or this container is also relocated. Since constraints (6.34) mix relocations of 20' and 40' containers, it could happen that one relocation of a 40' container is incorrectly used, without any 40' container being involved, when two relocations of 20' containers are needed. Therefore, constraints (6.35) ensure that reshuffles of 40' containers in cell $p$ at port $i$ can only occur if there is one 40' container not going to this port stowed in that cell before unloading operations at port $i$ and the cell is not occupied by 20' containers or by a 40' container with destination port $i$. Constraints (6.36)-(6.39) set the number of containers to be loaded in each port as in previous models. Constraints (6.40) ensure that a cell stows at most one 40' or one 20' container in its first slot. This is also satisfied for the second slot by constraints (6.41), which also ensure that a cell stows either two 20' containers or none. The last constraints (6.42), together with constraints (6.41), prevent the stacking of 20' containers above 40' ones.

### 6.3.4 A new model with 20' and 40' containers, considering container weights and stability constraints: BSW model

We finally present the BSW model to solve the problem considering all the assumptions presented in Section 6.2. It uses variables $t_{ijm}(p,x)$ defined by equation (6.17); variables $rt_i(p,x)$ and $rf_i(p)$ defined by equations (6.2) and (6.30); variables $f_{ijm}(p)$, which indicate whether there is a 40' container with destination $j$ and weight class $m$ in cell $p$, and variables $e_i(p)$, indicating whether cell $p$ is empty when leaving port $i$:

$$f_{ijm}(p) = \begin{cases} 1, & \text{if one 40' container of class } m \text{ with destination port } j \text{ is stowed in cell } p \\ & \text{after the loading operations at port } i \\ 0, & \text{otherwise} \end{cases} \tag{6.43}$$
$$\forall i \in \mathcal{O}; \quad \forall j \in \mathcal{D}^i; \quad \forall m \in \mathcal{M}; \quad \forall p \in \mathcal{P};$$

$$e_i(p) = \begin{cases} 1, & \text{if cell } p \text{ is empty after the loading operations at port } i \\ 0, & \text{otherwise} \end{cases} \tag{6.44}$$
$$\forall i \in \mathcal{O}; \quad \forall p \in \mathcal{P};$$

With these variables, the objective function of the BSW model (6.45) seeks to minimize the total number of reshuffles. We weight the 20' and 40' relocations alike, but a penalty could be applied to the 40' ones by adding a coefficient greater than one to the $rf$ variables.

$$\text{Min} \sum_{i\in\mathcal{O}^r} \sum_{p\in\mathcal{P}} \Big( \sum_{x\in\mathcal{X}} rt_i(p,x) + rf_i(p) \Big) \tag{6.45}$$

For the sake of clarity, constraints are divided into two groups: Reshuffles and Storage.

The *Reshuffles at port* $i$, $\forall i \in \mathcal{O}^r$, group of constraints identifies the unproductive moves at each port $i$. Constraints (6.46) identify relocations of containers in a given cell and constraints (6.47) in a given slot. Constraints (6.48) identify unproductive moves required to unload a container and constraints (6.49) restrict the cases in which reshuffles of 40' containers can occur.

*Reshuffles (i)* :

$$f_{i-1jm}(p) - f_{ijm}(p) \leq rf_i(p) \qquad \forall i \in \mathcal{O}^r,\ \forall j \in \mathcal{D}^i,\ \forall m \in \mathcal{M},\ \forall p \in \mathcal{P} \tag{6.46}$$

$$t_{i-1jm}(p,x) - t_{ijm}(p,x) \leq rt_i(p,x) \quad \forall i \in \mathcal{O}^r,\ \forall j \in \mathcal{D}^i,\ \forall m \in \mathcal{M},\ \forall p \in \mathcal{P},\ \forall x \in \mathcal{X} \tag{6.47}$$

$$\begin{aligned}\sum_{m\in\mathcal{M}} \Big(t_{i-1im}(b,c,r,x) + f_{i-1im}(b,c,r)\Big) + rt_i(b,c,r,x) + rf_i(b,c,r) \leq e_{i-1}(b,c,r+1)\\ + \sum_{m\in\mathcal{M}} \Big(t_{i-1im}(b,c,r+1,x) + f_{i-1im}(b,c,r+1)\Big) + rt_i(b,c,r+1,x) + rf_i(b,c,r+1)\\ \forall i \in \mathcal{O}^r,\ \forall b \in \mathcal{B},\ \forall c \in \mathcal{C}^b,\ \forall r \in \mathcal{R}^{bc} \setminus \{R^{bc}\},\ \forall x \in \mathcal{X}\end{aligned} \tag{6.48}$$

$$rf_i(p) + \sum_{j\in\mathcal{D}^{i-1}} \sum_{m\in\mathcal{M}} t_{i-1jm}(p,1) + \sum_{m\in\mathcal{M}} f_{i-1im}(p) \leq 1 \qquad \forall i \in \mathcal{O} \setminus \{1\},\ \forall p \in \mathcal{P} \tag{6.49}$$

The constraints involved in the *Storage at port i* group ensure a proper stowage $\forall i \in \mathcal{O}$. Constraints (6.50)-(6.53) specify the number of containers of each size and type on board the ship upon leaving each port $i$. Constraints (6.50) and (6.52) do so for the 20' containers and (6.51) and (6.53) for the 40' containers. Constraints (6.54) set the number of empty cells (i.e., half the number of empty slots). Constraints (6.55) and (6.56) guarantee that a slot will hold a maximum of one 40' or one 20' container. By constraints (6.56), there are two 20' containers or none in a cell. Constraints (6.56) and (6.57) prevent 20' containers being stacked on top of 40' containers, and no containers can hang because of constraints (6.58). The stability of the ship is fulfilled due to constraints (6.59) and (6.60). The difference in weight between the containers stowed in the bow and stern ends of the ship is less than $Q_1$ percent of the total weight at that port by constraints (6.59). The difference in weight between the containers stowed in the starboard and port sides of the ship is less than $Q_2$ percent of the total weight at that port by constraints (6.60).

Storage (i) :

$$\sum_{p\in\mathcal{P}} \sum_{x\in\mathcal{X}} t_{i(i+1)m}(p,x) \geq \sum_{k=1}^{i} T^m_{k(i+1)} \qquad \forall i \in \mathcal{O},\ \forall m \in \mathcal{M} \tag{6.50}$$

$$\sum_{p\in\mathcal{P}} f_{i(i+1)m}(p) \geq \sum_{k=1}^{i} F^m_{k(i+1)} \qquad \forall i \in \mathcal{O},\ \forall m \in \mathcal{M} \tag{6.51}$$

$$\sum_{p\in\mathcal{P}} \sum_{x\in\mathcal{X}} t_{ijm}(p,x) \geq \sum_{p\in\mathcal{P}} \sum_{x\in\mathcal{X}} t_{(i+1)jm}(p,x) - T^m_{(i+1)j} \quad \forall i \in \mathcal{O} \setminus \{N-1\},\ \forall j \in \mathcal{D}^i \setminus \{i+1\},\ \forall m \in \mathcal{M} \tag{6.52}$$

$$\sum_{p\in\mathcal{P}} f_{ijm}(p) \geq \sum_{p\in\mathcal{P}} f_{(i+1)jm}(p) - F^m_{(i+1)j} \qquad \forall i \in \mathcal{O} \setminus \{N-1\},\ \forall j \in \mathcal{D}^i \setminus \{i+1\},\ \forall m \in \mathcal{M} \tag{6.53}$$

$$\sum_{p\in\mathcal{P}} e_i(p) \geq E_i/2 \qquad \forall i \in \mathcal{O} \tag{6.54}$$

$$\sum_{j\in\mathcal{D}^i} \sum_{m\in\mathcal{M}} t_{ijm}(p,1) + \sum_{j\in\mathcal{D}^i} \sum_{m\in\mathcal{M}} f_{ijm}(p) + e_i(p) \leq 1 \quad \forall i \in \mathcal{O}, p \in \mathcal{P} \tag{6.55}$$

$$\sum_{j\in\mathcal{D}^i} \sum_{m\in\mathcal{M}} t_{ijm}(p,2) \leq \sum_{j\in\mathcal{D}^i} \sum_{m\in\mathcal{M}} t_{ijm}(p,1) \qquad \forall i \in \mathcal{O}, p \in \mathcal{P} \tag{6.56}$$

$$\sum_{j\in\mathcal{D}^i} \sum_{m\in\mathcal{M}} t_{ij}(b,c,r+1,2) \leq \sum_{j\in\mathcal{D}^i} \sum_{m\in\mathcal{M}} t_{ij}(b,c,r,2) \quad \forall i \in \mathcal{O}, b \in \mathcal{B}, c \in \mathcal{C}^b, r \in \mathcal{R}^{bc} \setminus \{R^{bc}\} \tag{6.57}$$

$$e_i(b,c,r) \leq e_i(b,c,r+1) \qquad \forall i \in \mathcal{O}, b \in \mathcal{B},\ \forall c \in \mathcal{C}^b, r \in \mathcal{R}^{bc} \setminus \{R^{bc}\} \quad (6.58)$$

$$\Big|\sum_{j\in\mathcal{D}^i}\sum_{m\in\mathcal{M}}\Big(\sum_{p,x\in\mathcal{LP}}\big(w_m^T t_{ijm}(p,x)+w_m^F f_{ijm}(p)\big)-\sum_{p,x\in\mathcal{RP}}\big(w_m^T t_{ijm}(p,x)+w_m^F f_{ijm}(p)\big)\Big)\Big| \leq Q_1 W_i \qquad \forall i \in \mathcal{O} \quad (6.59)$$

$$\Big|\sum_{j\in\mathcal{D}^i}\sum_{m\in\mathcal{M}}\Big(\sum_{p,x\in\mathcal{WP}}\big(w_m^T t_{ijm}(p,x)+w_m^F f_{ijm}(p)\big)-\sum_{p,x\in\mathcal{YP}}\big(w_m^T t_{ijm}(p,x)+w_m^F f_{ijm}(p)\big)\Big)\Big| \leq Q_2 W_i \qquad \forall i \in \mathcal{O} \quad (6.60)$$

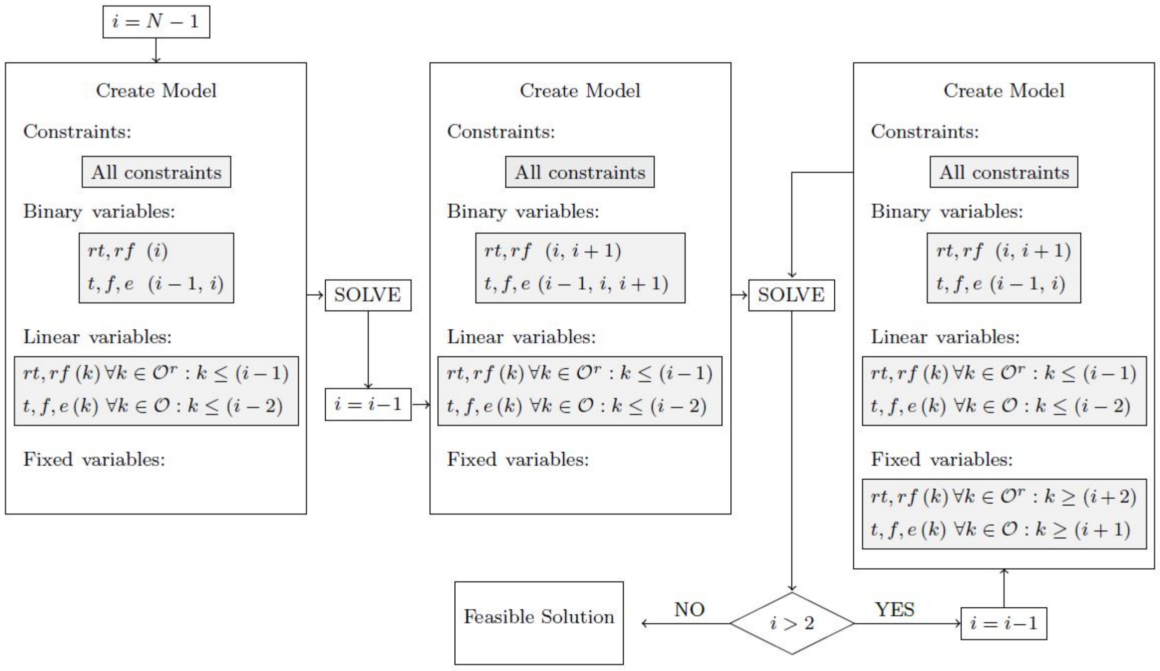
## 6.4 Matheuristic algorithms

The integer programming model described in Section 6.3.4 requires a large number of variables and constraints and solvers such as CPLEX do not provide satisfactory results for real-size instances. The structure of the BSW model, in which the variables describe the layout of the ship at each port of its route, suggests the use of matheuristic algorithms such as Relax-and-Fix, Insert-and-Fix, and Fractional Relax-and-Fix (Wolsey, 1998). Matheuristics of this kind have been widely used in lot-sizing and scheduling problems arising in foundries (de Araujo et al., 2008), animal feed plants (Toso et al., 2009), or flat-panel display industries (Lee and Lee, 2020), as well as in other areas such as bin packing problems (Paquay et al., 2018), production-distribution planning (Wei et al., 2017), or production planning and warehouse layout problems (Zhang et al., 2017). One main advantage of these matheuristics is that they are based on the decomposition of mathematical models, so they inherit their main advantages: they are easy for practitioners to implement; they are flexible, allowing constraints to be added or removed to fulfill the requirements of the specific problem being solved, and for their resolution they use commercial solvers, which have shown a dramatic increase in performance in recent years, often by five or even more orders of magnitude (Bixby, 2002), and are constantly improving. This section presents these three matheuristics based on the decomposition of the BSW model into simpler models that are easier to solve.

### 6.4.1 The Relax-and-Fix algorithm: RF

Relax-and-Fix is an iterative algorithm that solves, at each iteration, a mixed integer linear problem (MILP), which is a relaxation of the original problem. All the constraints of the original problem are included in each iteration, the variables that had been declared as binary in the previous iteration are set to the value obtained, a new subset of variables is declared as binary, and the integrality of the remaining variables is relaxed. Figure 6.2 shows a schematic diagram of the RF algorithm we have developed.

This algorithm starts by declaring the variables corresponding to the latest ports as binary, and at each iteration it declares the variables corresponding to an earlier port as binary, until it reaches the first port on the route. The first iteration ($i = N - 1$) solves a sub-problem that includes all the Reshuffles and Storage constraints of the BSW model and



$i = N - 1$
Create Model
Constraints:
All constraints
Binary variables:
$rt, rf \ (i)$
$t, f, e \ (i-1, i)$
Linear variables:
$rt, rf\,(k)\,\forall k \in \mathcal{O}^r : k \leq (i-1)$
$t, f, e\,(k)\,\forall k \in \mathcal{O} : k \leq (i-2)$
Fixed variables:
SOLVE
$i = i-1$
Create Model
Constraints:
All constraints
Binary variables:
$rt, rf \ (i, i+1)$
$t, f, e \ (i-1, i, i+1)$
Linear variables:
$rt, rf\,(k)\,\forall k \in \mathcal{O}^r : k \leq (i-1)$
$t, f, e\,(k)\,\forall k \in \mathcal{O} : k \leq (i-2)$
Fixed variables:
SOLVE
Create Model
Constraints:
All constraints
Binary variables:
$rt, rf \ (i, i+1)$
$t, f, e \ (i-1, i)$
Linear variables:
$rt, rf\,(k)\,\forall k \in \mathcal{O}^r : k \leq (i-1)$
$t, f, e\,(k)\,\forall k \in \mathcal{O} : k \leq (i-2)$
Fixed variables:
$rt, rf\,(k)\,\forall k \in \mathcal{O}^r : k \geq (i+2)$
$t, f, e\,(k)\,\forall k \in \mathcal{O} : k \geq (i+1)$
Feasible Solution
NO
$i > 2$
YES
$i = i-1$

Figure 6.2: Schematic figure of the Relax-and-Fix algorithm.

consider the relocation variables, $rt$ and $rf$, at port $N-1$ and the layout variables, $t$, $f$, $e$, at ports $N-2$ and $N-1$ as binary, while all other variables are relaxed to be linear. The second iteration ($i = N-2$) solves the sub-problem considering the relocation variables at ports $N-2$ and $N-1$ and the layout variables at ports $N-3$, $N-2$ and $N-1$ as binary. All successive iterations $i$, such that $i \in \mathcal{O}^r$ and $i \leq N-3$, solve the sub-problem declaring the relocation variables at ports $i$ and $i+1$ and the layout variables at ports $i-1$ and $i$ as binary, and fixing all the relocation variables for ports $k \geq i+2$ and the layout variables for ports $k \geq i+1$ to the values obtained in previous iterations.

### 6.4.2 The Insert-and-Fix algorithm: IF

Insert-and-Fix is an iterative algorithm that solves, at each iteration, an integer linear problem (ILP), which is a subset of the original. Unlike Relax-and-Fix, the constraints are not all considered at every iteration but are added step by step throughout the algorithm. In each iteration, the variables that had been declared as binary in the previous iteration are set to the values obtained and new constraints are added together with the binary variables inherent in them. Figure 6.3 shows a schematic diagram of the IF algorithm we have developed.

The first iteration of the IF algorithm ($i = N-1$) solves a sub-problem that includes the Reshuffles constraints of the BSW model for port $N-1$ and the Storage constraints for ports $N-2$ and $N-1$, together with the relocation and layout binary variables involved in those constraints, $rt, rf$ at port $N-1$ and $t, f, e$ at ports $N-2$ and $N-1$. At the second iteration ($i = N-2$), the Reshuffles constraints for port $N-2$ and the Storage constraints for $N-3$ are included. The addition of these constraints entails the addition of the binary relocation variables at ports $N-2$ and layout variables at port $N-3$. All successive iterations $i$, such

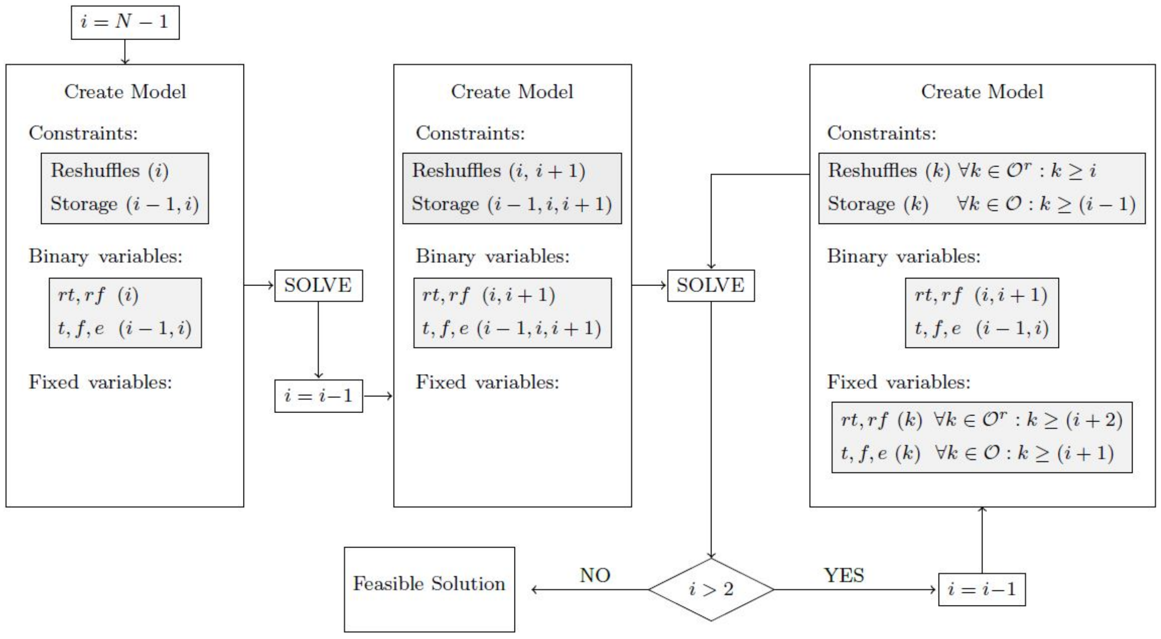


Figure 6.3: Schematic figure of the Insert-and-Fix algorithm.

that $i \in \mathcal{O}^r$ and $i \leq N - 3$, solve the sub-problem with the Reshuffles constraints for ports $k \geq i$ and the Storage constraints for ports $k \geq i - 1$, in addition to the binary relocation variables at ports $i$ and $i+1$ and layout variables at ports $i-1$ and $i$, fixing all the relocation variables for ports $k \geq i+2$ and the layout variables for ports $k \geq i+1$ to the values obtained in previous iterations.

### 6.4.3 The Fractional Relax-and-Fix algorithm: FRF

The Fractional Relax-and-Fix is an iterative procedure that can be seen as a combination of the Relax-and-Fix and Insert-and-Fix algorithms. Similarly to Relax-and-Fix, it solves a MILP at each iteration. However, the MILP is not a relaxation of the original problem, but a relaxation of one subset of the original problem, as in Insert-and-Fix. In each iteration, the variables that had been declared as binary in the previous iteration are set to the values obtained and new constraints are added together with the variables inherent in them. Only the integrality of some of the variables added at each iteration is considered. Figure 6.4 shows a schematic diagram of the FRF algorithm we have developed.

The FRF algorithm solves, at the first iteration, a sub-problem that includes the Reshuffles constraints of the BSW model for port $N-1$ and the Storage constraints for ports $N-2$ and $N-1$, together with the relocation and layout binary variables inherent in them. At the second iteration ($i = N-2$), the Reshuffles constraints for port $N-2$ and the Storage constraints for $N-3$ are added, but the variables involved, the relocation variables at ports $N-2$ and the layout variables at port $N-3$ are included as linear variables. All successive iterations $i$, such that $i \in \mathcal{O}^r$ and $i \leq N-3$, solve the sub-problem with the binary relocation variables at ports $i+1$ and layout variables at ports $i$, adding the Reshuffles constraints for port $i$ and the Storage constraints for $i-1$, as well as the variables inherent in them with

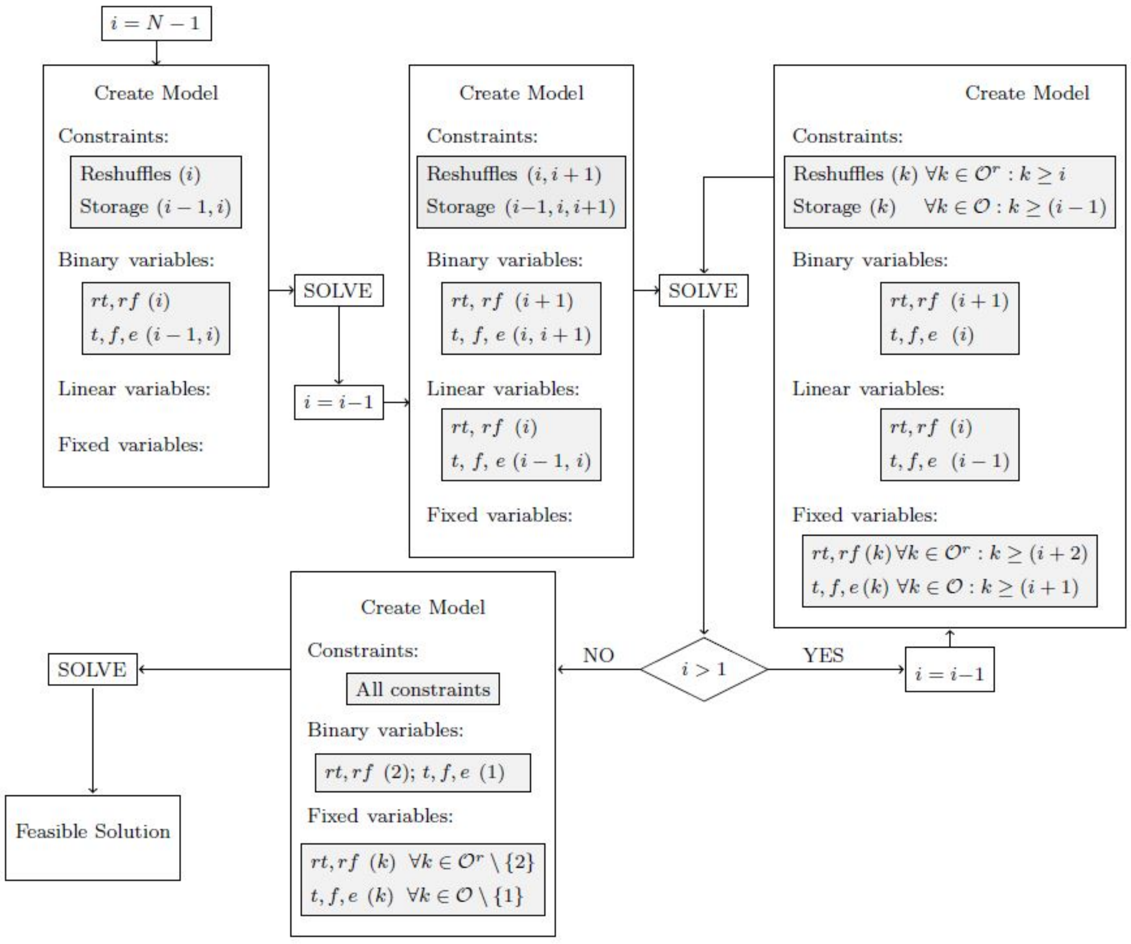


Figure 6.4: Schematic figure of the Fractional Relax-and-Fix algorithm.

their integrality relaxed, and fixing all the relocation variables for ports $k \geq i + 2$ and the layout variables for ports $k \geq i + 1$ to the values obtained in previous iterations.

### 6.4.4 A heuristic algorithm to provide initial solutions for the solvers

We have developed a heuristic algorithm, INIT_LAYOUT, to provide the solver with an initial solution at each iteration of the RF, IF, and FRF algorithms. When these algorithms are going to solve the integer model including port $k - 1$, INIT_LAYOUT provides an initial layout of the containers aboard when leaving port $k - 1$, knowing the layout of the containers aboard the ship when leaving the next port $k$, given by the solution of the previous model, and the set of containers to be loaded/unloaded at each port, given by the transportation matrices. For the first model to be solved at port $N - 1$, a specific heuristic, FIRST_LAYOUT, provides the layout at this port taking into account that only containers going to the last port $N$ will be aboard.

The RF and IF algorithms call the heuristic once per iteration. At iteration $i$, the layout of the ship at port $i$ is known from the previous iteration, so the heuristic obtains the layout of the ship when it leaves port $i - 1$. The FRF algorithm calls the heuristic twice per iteration. A solution for the layout of the ship when leaving port $i + 1$ is known at iteration

$i$, given by the solution of the previous model. The algorithm uses the heuristic to obtain the layout at port $i$, and once it is obtained, is used again to obtain the layout at port $i-1$.

INIT_LAYOUT can also be used to complete solutions if the matheuristic algorithms, RF, IF, and FRF, fail to provide a solution for all the ports in the route within the allowed running time. The heuristic starts from the solution of the last integer model solved and builds the layout for the remaining ports back to the first port on the route. We refer to the algorithms that include the heuristic to complete solutions, if needed, as $\mathrm{RF_H}$, $\mathrm{IF_H}$, and $\mathrm{FRF_H}$.

**FIRST_LAYOUT: Obtaining the layout of the ship when leaving port $N-1$**

Algorithm 5 provides the pseudocode for the procedure used to generate the layout of the ship when leaving port $N-1$. It first assigns 20' containers from port $N-1$ to port $N$ by calling the function ASSIGN_TWENTY() in line 3, and then 40' containers from port $N-1$ to port $N$ by function ASSIGN_FORTY() in line 4. These functions work as follows: the area of the ship with the least weight is selected and in that area the column with the least occupancy in which 20' or 40' containers can be placed, depending of the function used, is considered. If there is no row $c$ in that area with these characteristics, the process is repeated in the other areas until a row $c$ is found. Then, function LOADT() assigns the heaviest and the lightest 20' containers remaining to be loaded to the two slots of each empty cell in the column, and then LOADF() assigns the lightest 40' container remaining until the row $c$ of the bay selected is filled up. The process is repeated, updating the weight in the different areas of the ship, and it ends as soon as there are no more containers available or no more cells in which to load them. After the assignments, if the stability conditions of the ship are not met, a repair procedure, REPAIR(), is used in line 5.

**Algorithm 5** A heuristic algorithm to provide the layout of the ship at port $N-1$.

```
1: function FIRST_LAYOUT($\mathcal{T}_{N-1N}$, $\mathcal{F}_{N-1N}$)
2:     $\pi^{N-1} \leftarrow \varnothing$                ▷ Let $\pi^{N-1}$ be the ship's layout when leaving port $N-1$
3:     ASSIGN_TWENTY($\pi^{N-1}$, $\mathcal{T}_{N-1N}$)
4:     ASSIGN_FORTY($\pi^{N-1}$, $\mathcal{F}_{N-1N}$)
5:     if STABILITY($\pi^{N-1}$) = FALSE then REPAIR($\pi^{N-1}$)
6:     return $\pi^{N-1}$
```

The REPAIR() function carries out an iterative procedure until the stability conditions are fulfilled or no more moves can be made. It first checks whether the stability between bow and stern ends is satisfied. If it is not, it removes a container from the area with the highest weight and tries to put it in the area with the lowest weight. Let us assume that stability is not satisfied because the weight in the bow end is much greater than in the stern end and let us also assume that of the two areas in the bow end, the starboard bow area carries greater weight than the port bow area, while in the stern end the starboard stern area carries greater weight than the port stern area. In this case, the procedure removes a container from the starboard bow area. It selects 40' containers first, and when there are no 40' containers available to remove, it starts removing 20' containers. It tries to place the removed container in the area with the lowest weight, in this case in the port stern area. If there are no slots available in that area, it tries the next areas in increasing order of weight, in this case the starboard stern area and, if necessary, the port bow area. If the stability between bow and stern is satisfied or no further movement from bow to stern can be made, the procedure checks the stability between starboard and port and, if not satisfied, repeats the process for these sides.

### INIT_LAYOUT, an algorithm to provide the solver with initial solutions

Algorithm 6 contains the pseudocode of the heuristic that provides the solver with initial solutions. Given the layout of the ship at port $k$, $\pi^k$, and the sets of containers to be loaded/unloaded at each port, it provides the layout of the ship at port $k-1$ as well as a list of the relocations that this layout will produce at port $k$. If $k = N-1$ and there is no known layout at port $N-1$, HEURISTIC_FIRST() is applied to obtain a layout at port $N-1$ in line 2. The layout of the ship at port $k-1$ is initially copied from that at port $k$ in line 3 and the list of relocations at port $k$ initialized in line 4. Lines 5 and 6 initialize two lists: *uneven*, in which identifiers of cells occupied by a single TEU are stored, and *hang*, with identifiers of rows in which there are hanging containers after the removal of other containers. The layout copied from port $k$ contains $T_{kd}$ containers, for all $d > k$, that will be loaded in port $k$ and therefore cannot be part of the layout at port $k-1$. From all positions occupied by containers going to each port $d > k$, $T_{kd}$ positions have to be determined and their containers removed, before completing the layout with containers going from port $k-1$ to port $k$.

Function REMOVE_CONTAINERS() on line 7 removes the containers that will be loaded at port $k$ from the layout of the ship at port $k-1$, as follows:

- Step 1. Consider a tier $t$, initially the highest tier, $t = hl$, and the latest destination $d = ld$ of containers loaded at $k$.
- Step 2. Create a list *rem* with information on the cells at tier $t$ occupied by containers going to port $d$ that could have been loaded at port $k$. It is ordered according to the following criteria: first cells that leave the least number of containers hanging if the container(s) stored are removed, then cells of rows where the most containers to be loaded at port $k$ can be removed, then cells in which two 20' containers can be removed, then cells in which one 40' container can be removed, and finally those in which just one 20' container can be removed. Ties are broken by increasing bay, row, and tier.
- Step 3. If list *rem* is empty, set $t = t-1$ and go back to step 2. If not, check whether the removal of container(s) in the cell being considered means leaving some containers hanging, and if so, also go back to step 2 to add more cells to the list. If there are no containers left hanging or $t$ is equal to the lowest tier in the row, the containers from the first cell on the list are removed, as well as those from the lower tiers of the same row if they could also be loaded at port $k$. Sets $\mathcal{T}_k$ and $\mathcal{F}_k$ are updated and if $\mathcal{T}_k = \varnothing$ and $\mathcal{F}_k = \varnothing$, the process ends.
- Step 4. If $T_{kd} = F_{kd} = 0$ for the current $d$, set $d$ to the latest destination of containers loaded at $k$, $t = hl$ and go to step 2. Otherwise, if list *rem* is empty, set $t = t-1$ and go to step 2. If it is not, update *rem* considering that $\mathcal{T}_k$ and $\mathcal{F}_k$ have been updated and go to step 3.

**Algorithm 6** A heuristic algorithm to provide initial solutions.

```
1: function INIT_LAYOUT($\pi^k, k, \mathcal{T}, \mathcal{F}$)    ▷ Let $\pi^k$ be the ship's layout when leaving port $k$
2:     if $k = N - 1$ and $\pi^k = \varnothing$ then $\pi^k \leftarrow$ HEURISTIC_FIRST($\mathcal{T}, \mathcal{F}$)
3:     $\pi^{k-1} \leftarrow \pi^k$    ▷ Copy the layout of port $k$ at port $k - 1$
4:     $r^k \leftarrow \varnothing$    ▷ $r^k$: ship's relocations at port $k$
5:     $uneven \leftarrow \varnothing$    ▷ uneven: row-bay pairs with cells occupied by only one TEU
6:     $hang \leftarrow \varnothing$    ▷ dirty: rows with hanging containers
7:     REMOVE_CONTAINERS($\pi^{k-1}, \mathcal{T}_k, \mathcal{F}_k, uneven, hang$) ▷ Remove containers loaded at $k$ from the $k - 1$ layout
8:     $\mathcal{U} \leftarrow \varnothing$;    ▷ $\mathcal{U}$: set of 20' containers being relocated at port $k$
9:     if $uneven <> \varnothing$ then
10:        FILL_UNEVEN($\pi^{k-1}, uneven, hang, \mathcal{T}_{k-1k}$)    ▷ Complete cells storing one TEU with $\mathcal{T}_{k-1k}$
11:        REMOVE_UNEVEN($\pi^{k-1}, uneven, hang, \mathcal{U}$) ▷ Remove and add 20' containers in cells storing one TEU
12:    RUN_HANG($\pi^{k-1}, r^k, hang, \mathcal{T}_{k-1k}$)    ▷ Fill with 20' containers or unhook 40' containers
13:    ASSIGN_TWENTY($\pi^{k-1}, \mathcal{T}_{k-1k}$)
14:    ASSIGN_FORTY($\pi^{k-1}, \mathcal{F}_{k-1k}$)
15:    if $\mathcal{T}_{k-1k} <> \varnothing$ then INSERT_TWENTY($\pi^{k-1}, r^k, \mathcal{T}_{k-1k}, \mathcal{U}$)
16:    if STABILITY($\pi^{k-1}$) = FALSE then REPAIR($\pi^{k-1}$)
17:    return $\pi^{k-1}, r^k, \pi^k$
```

Once the containers to be loaded at port $k$ have been removed, the containers of sets $\mathcal{T}_{k-1k}$ and $\mathcal{F}_{k-1k}$ are added in such a way as to produce a feasible complete layout at port $k - 1$. If there are cells occupied by only one 20' container, function FILL_UNEVEN() in line 10 tries to fill the empty slots with containers from set $\mathcal{T}_{k-1k}$. If all the containers in set $\mathcal{T}_{k-1k}$ have been assigned but there are still cells storing only one 20' container, the function REMOVE_UNEVEN() in line 11 runs through them, removing 20' containers and adding them at the next cell with just one 20' container stored. These moves can produce some relocations at port $k$. Function RUN_DIRTY() on line 12 goes through the rows with hanging containers, either filling the lower tiers of the row with 20' containers or unhooking the containers. Again, relocations at port $k$ may be produced in this step. Then, functions ASSIGN_TWENTY() and ASSIGN_FORTY() are executed if there are more containers to assign. When not all the 20' containers from port $k - 1$ to port $k$ have been assigned in the previous steps of the heuristic, and there are 40' containers in all the rows of the ship that are not completely filled, function INSERT_TWENTY() in line 15 goes through the rows inserting the 20' containers. Finally, if the stability conditions of the ship are not met, the REPAIR() procedure described in Section 6.4.4 is applied.

## 6.5 Computational experiments

We conducted an extensive computational analysis to test the performance of the integer programming models and matheuristics proposed in this chapter. We focused on answering the following research questions:

1. What is the effect of including container sizes in the models?
2. What is the effect of including container weights in the models?
3. How does the simultaneous inclusion of container sizes and weights affect the problem?
4. What is the maximum instance size that the mathematical models can solve?
5. What is the behavior of the model-based matheuristics for this problem?

6. Which algorithm performs best: RF, IF, or FRF?
7. Up to what size of instances can the matheuristics solve?
8. How does the best proposal, $\text{FRF}_\text{H}$, perform in a real ship instance?

Mathematical models and matheuristics were coded in C++ and executed on virtual machines with 8 virtual processors and 16 GBytes of RAM running Windows 10 Enterprise 64 bits. The virtual machines were run in an OpenStack virtualization platform supported by 12 blades, each with four 12-core AMD Opteron Abu Dhabi 6344 processors running at 2.6 GHz and 256 GBytes of RAM, for a total of 576 cores and 3 TBytes of RAM. We fixed the time limit at 3600 seconds and used solver IBM ILOG CPLEX 12.9 with 8 threads.

### 6.5.1 Test instances

To obtain the test instances, we generated full transportation matrices using the Authentic Matrices Generation Algorithm proposed by Ding and Chou (2015) for 6 ports ($N = 6$). Given a capacity (in TEUs), these matrices provide the number of containers to be transported from each port to each port later in the route so that the number of TEUs on board equals the total capacity when leaving each port. To achieve a load percentage of $x\%$, we can generate these matrices by setting the capacity to $x\%$ of the container ship's capacity. In this study we consider an 85% load capacity. Note that a higher load capacity would generate endless reshuffles at each port. A lower load capacity is not economically feasible, so 85% is a sensible figure. Instances generated in this way are used to evaluate the Base model. To evaluate the BW model we split those instances, considering that 1/3 of the containers on board are light, 1/3 medium, and 1/3 heavy. We followed a similar procedure to obtain the instances for evaluating the BS model. In this case, the percentage of loading of 20' and 40' containers varies. In one third of the instances, 25% of the TEUs are occupied by 20' containers and 75% by 40' containers. In another third, 75% are occupied by 20' containers and 25% by 40' containers. The remaining instances have a 50%-50% occupancy rate. In the instances generated to evaluate the BSW, as well as the matheuristics presented in this chapter, the rates of light, medium and heavy containers are also 1/3-1/3-1/3, and the percentages of 20' and 40' containers are 25%-75%, 75%-25%, and 50%-50%. The weights of the containers are set according to Table 6.2.

Table 6.2: Weight of the containers in tons according to their size $\mathcal{S}$ and weight class $\mathcal{M}$.

| | $\mathcal{M}$ | | |
|---|---|---|---|
| $\mathcal{S}$ | Light | Medium | Heavy |
| 20' | 7 | 14 | 21 |
| 40' | 10 | 20 | 30 |

We generated instances for two groups of ships, TI1 and TI2, each group containing ships of 14 different capacities. There are 15 instances per ship type, so 210 per group, giving a total of 420 instances. Group TI1 includes small ships with capacities of up to 3200 TEUs, and group TI2 large ships with capacities of up to 20000 TEUs (approximately corresponding to the largest ships available today). We consider that the number of rows in the ships is constant in each bay and the number of tiers in each bay and row is also constant. This is by no means a strong simplification as more specific bay configurations would be straightforward to consider. The specific features of TI1 and TI2 are described below:

TI1 consists of ships with capacities ranging from 400 up to 3200 TEUs. According to the capacity, we consider ships with 5 bays, 4 to 10 rows, and 10 tiers, and also ships with 10 bays, 6 to 16 rows, and 10 tiers.

TI2 consists of ships with capacities ranging from 2000 up to 20000 TEUs. All these ships have 25 bays, 6 to 16 rows, and 10 tiers.

We also consider the structure of a real ship, for which we have access to real data (Larsen and Pacino, 2020). This ship has a capacity of 7032 TEUs, i.e., 3516 cells, with 21 bays, up to 16 rows in each bay, and up to 15 tiers in each bay and row. The number of rows per bay and tiers per bay-row are not constant here, so the models are adapted to this case. We generated 30 instances for this ship.

### 6.5.2 Performance of the integer programming models

We aim to evaluate how the simplified problem, which only considers 20' containers and no stability constraints, becomes more and more complex as we add different container sizes as well as container weights and stability constraints. Table 6.3 shows the results obtained by the integer programming models, *Base*, *BW*, *BS*, and *BSW*, on the instances of the TI1 group with up to 3200 containers. The test instances are grouped by container ship capacity ($\Omega$) with 85% of load capacity and considering $Q_1 = Q_2 = 0.01$.

Table 6.3: Performance of the Base, BW, BS, and BSW models on instances grouped by container ship capacity ($\Omega$) with 85% of load capacity.

| | | Base | | | BW ($Q_1 = Q_2 = 0.01$) | | | | BS | | | | BSW ($Q_1 = Q_2 = 0.01$) | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\Omega$ | # | O | R | CPU | O | F | R | CPU | O | F | R | CPU | O | F | R | CPU |
| 400 | 15 | 15 | 0.0 | 35.9 | 15 | 15 | 0.0 | 375.9 | 15 | 15 | 0.0 | 279.5 | 11 | 12 | 0.4 | 2420.2 |
| 600 | 15 | 15 | 0.0 | 76.0 | 15 | 15 | 0.0 | 681.8 | 15 | 15 | 0.0 | 311.3 | 3 | 3 | 0.0 | 2113.5 |
| 800 | 15 | 15 | 0.0 | 148.5 | 15 | 15 | 0.0 | 1212.2 | 15 | 15 | 0.0 | 660.9 | 0 | 1 | 1.0 | 3600.4 |
| 1000 | 15 | 15 | 0.0 | 172.7 | 15 | 15 | 0.0 | 1465.2 | 15 | 15 | 0.0 | 1296.3 | 0 | 0 | - | - |
| 1200 | 15 | 15 | 0.0 | 197.8 | 13 | 13 | 0.0 | 2081.0 | 14 | 14 | 0.0 | 1451.5 | 0 | 0 | - | - |
| 1400 | 15 | 15 | 0.0 | 317.8 | 13 | 13 | 0.0 | 2263.6 | 15 | 15 | 0.0 | 1137.2 | 0 | 0 | - | - |
| 1600 | 15 | 15 | 0.0 | 367.2 | 10 | 11 | 300.5 | 2701.6 | 14 | 15 | 0.9 | 1288.7 | 0 | 0 | - | - |
| 2000 | 15 | 15 | 0.0 | 623.1 | 5 | 9 | 1562.2 | 3496.6 | 15 | 15 | 0.0 | 1877.0 | 0 | 0 | - | - |
| 2200 | 15 | 15 | 0.0 | 687.4 | 2 | 5 | 3264.6 | 3549.1 | 11 | 15 | 277.1 | 2473.1 | 0 | 0 | - | - |
| 2400 | 15 | 15 | 0.0 | 805.0 | 0 | 2 | 5927.5 | 3600.8 | 12 | 15 | 725.6 | 2719.9 | 0 | 0 | - | - |
| 2600 | 15 | 15 | 0.0 | 1037.5 | 1 | 2 | 3081.0 | 3304.4 | 6 | 13 | 1219.1 | 3309.4 | 0 | 0 | - | - |
| 2800 | 15 | 15 | 0.0 | 1165.9 | 1 | 3 | 4157.7 | 3470.5 | 10 | 12 | 805.8 | 2489.0 | 0 | 0 | - | - |
| 3000 | 15 | 15 | 0.0 | 1520.0 | 0 | 3 | 7029.3 | 3601.4 | 8 | 11 | 1316.6 | 2856.4 | 0 | 0 | - | - |
| 3200 | 15 | 15 | 0.0 | 1553.9 | 0 | 1 | 7907.0 | 3601.3 | 4 | 5 | 1099.0 | 2908.8 | 0 | 0 | - | - |
| Tot/Avg. | 210 | 210 | 0.0 | 622.0 | 105 | 122 | 763.7 | 1885.8 | 169 | 190 | 318.7 | 1683.3 | 14 | 16 | 0.4 | 2436.5 |

Column "#" indicates the total number of instances tested in each group and columns "O" and "F" represent the number of optimal and feasible solutions obtained, respectively. The "R" and "CPU" columns represent, respectively, the average number of reshuffles and the average running time (in seconds) on instances in which a feasible solution is obtained. The 'Tot/Avg." row indicates the totals of the columns showing absolute frequencies and the averages of the columns showing average values. The notation "-" is used when the averages cannot be calculated.

With respect to the simplified problem, the Base model optimally solves every single instance tested with zero relocations and an average running time lower than 630 seconds. If we study the problem considering container weight and stability constraints, the BW model solves all instances up to 1000 TEUs and 87% of instances with 1200 and 1400 TEUs, with zero relocations. However, when the container capacity of ships is greater than or equal to 2000 TEUs, the model reaches optimal and feasible solutions in only a few instances, 9 and 25 out of 115, respectively. Moreover, the feasible solutions obtained are very bad, producing solutions with up to more than 7900 relocations. The model that considers different container sizes and no stability constraints, BS, performs better than the BW model, providing good results for instances up to 2000 TEUs. As in the previous case, from this capacity upwards the average number of relocations in the feasible solutions is extremely high. The problem considering both different container sizes and stability constraints is much more difficult. The BSW model only solves 11 instances of 400 TEUs to optimality and just 3 instances of 600 TEUs. One more non-optimal solution is obtained in the group of 800 TEUs. From Table 6.3 it can be concluded that the inclusion of container sizes has a stronger effect than the container weights and stability constraints, and that the combination of both characteristics makes the resulting model very difficult to solve.

### 6.5.3 Comparing the matheuristic proposals.

We evaluate the matheuristic algorithms, RF, IF, and FRF, first on the TI1 instances used to test the BSW model, showing the results in Table 6.4, and then on the larger TI2 instances with up to 20000 containers, whose results are shown in Table 6.5. Looking first at Table 6.4, it can be seen that FRF solves all instances, with an average of just 4.4 relocations. The same average number of relocations is obtained with IF, although for one instance this algorithm does not provide a solution. The worst performance is that of RF, which solves only 37 instances out of 210. Unlike the models, an increase in the ship capacity seems to improve the solutions obtained by the FRF algorithm. It solves the 60 instances for ships from 2600 to 3200 TEUs, obtaining in 51 cases zero resuffles, with an average running time of less than 780 seconds and an average of 1.3 relocations. However, the solutions for the smaller ships, with groups of 400 and 600 TEUs, are worse, with only 3 zero relocation solutions obtained out of 30, an average running time of around 2500 seconds, and an average of 14 relocations. The performance of IF is similar to that of FRF, although this trend is not as noticeable.

Table 6.5 shows the results obtained by the three matheuristics on the TI2 instances. The RF algorithm solves only 5 instances in the smaller group. On the other hand, the IF algorithm solves 209 out of 210 and the RF algorithm 205 out of 210. The IF algorithm finds a solution with zero relocations in 80.4% of the instances while FRF obtains zero relocation solutions in 96.1%. The average number of relocations in the solutions obtained by IF is 1.3, while the average of those obtained by FRF is 0.3. With respect to the running time, the average time spent by the IF is almost twice that of FRF. FRF solves 97.6% of stowage plans for ships from 2000 to 20000 TEUs with an average number of relocations of just 0.3 and an average running time of 730 seconds (about 12 minutes).

Table 6.4: Performance of the matheuristic algorithms on instances grouped by container ship capacity ($\Omega$) with 85% of load capacity. Considering $Q_1 = Q_2 = 0.01$.

| | | RF | | | | IF | | | | FRF | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\Omega$ | # | F | Z | R | CPU | F | Z | R | CPU | F | Z | R | CPU |
| 400 | 15 | 9 | 5 | 1.2 | 2623.0 | 14 | 3 | 4.2 | 2954.9 | 15 | 0 | 10.1 | 2029.0 |
| 600 | 15 | 0 | - | - | - | 15 | 3 | 13.5 | 2996.0 | 15 | 3 | 18.4 | 2921.9 |
| 800 | 15 | 2 | 2 | 0.0 | 2414.0 | 15 | 10 | 3.1 | 1103.6 | 15 | 8 | 6.3 | 1744.4 |
| 1000 | 15 | 1 | 1 | 0.0 | 2343.6 | 15 | 13 | 0.5 | 706.3 | 15 | 7 | 2.3 | 1570.2 |
| 1200 | 15 | 1 | 1 | 0.0 | 2721.1 | 15 | 9 | 3.7 | 1885.6 | 15 | 4 | 8.6 | 2027.0 |
| 1400 | 15 | 2 | 0 | 0.0 | 2560.9 | 15 | 13 | 0.2 | 1028.3 | 15 | 12 | 0.6 | 1167.1 |
| 1600 | 15 | 6 | 5 | 1.8 | 2593.8 | 15 | 12 | 14.2 | 877.0 | 15 | 11 | 8.1 | 1055.6 |
| 2000 | 15 | 5 | 4 | 0.6 | 2925.2 | 15 | 14 | 0.1 | 732.1 | 15 | 12 | 0.2 | 737.3 |
| 2200 | 15 | 4 | 4 | 0.0 | 2439.1 | 15 | 12 | 1.5 | 1917.7 | 15 | 13 | 0.4 | 860.0 |
| 2400 | 15 | 1 | 1 | 0.0 | 3042.9 | 15 | 9 | 3.2 | 2173.5 | 15 | 9 | 3.1 | 1504.0 |
| 2600 | 15 | 2 | 1 | 0.5 | 2928.2 | 15 | 11 | 4.9 | 1730.3 | 15 | 9 | 1.9 | 1361.2 |
| 2800 | 15 | 2 | 2 | 0.0 | 3049.6 | 15 | 14 | 0.6 | 651.1 | 15 | 15 | 0.0 | 301.0 |
| 3000 | 15 | 0 | - | - | - | 15 | 11 | 3.1 | 1175.0 | 15 | 14 | 0.2 | 727.9 |
| 3200 | 15 | 2 | 2 | 0.0 | 3413.8 | 15 | 9 | 8.4 | 1651.1 | 15 | 13 | 1.9 | 707.8 |
| Total/Avg | 210 | 37 | 30 | 0.7 | 2713.3 | 209 | 143 | 4.4 | 1534.8 | 210 | 130 | 4.4 | 1336.8 |

Column "#" indicates the total number of instances tested in each group and columns "O" and "F" represent the number of optimal and feasible solutions obtained, respectively. The "R" and "CPU" columns represent, respectively, the average number of reshuffles and the average running time (in seconds) on instances in which a feasible solution is obtained. The "Tot/Avg." row indicates the totals of the columns showing absolute frequencies and the averages of the columns showing average values. The notation "-" is used when the averages cannot be calculated.

Table 6.5: Performance of the matheuristic algorithms grouped by container ship capacity ($\Omega$) with 85% of load capacity. Large instances considering $Q_1 = Q_2 = 0.01$.

| | | RF | | | | IF | | | | FRF | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\Omega$ | # | F | Z | R | CPU | F | Z | R | CPU | F | Z | R | CPU |
| 2000 | 15 | 5 | 4 | 0.6 | 2660.3 | 15 | 15 | 0.0 | 559.9 | 15 | 14 | 0.1 | 515.0 |
| 3000 | 15 | 0 | - | - | - | 15 | 12 | 2.7 | 787.8 | 15 | 13 | 0.5 | 230.5 |
| 4000 | 15 | 0 | - | - | - | 15 | 12 | 0.5 | 1057.0 | 15 | 15 | 0.0 | 147.0 |
| 5000 | 15 | 0 | - | - | - | 15 | 10 | 4.9 | 1418.9 | 14 | 14 | 0.0 | 309.2 |
| 6000 | 15 | 0 | - | - | - | 15 | 11 | 0.5 | 1149.5 | 15 | 15 | 0.0 | 262.8 |
| 7000 | 15 | 0 | - | - | - | 15 | 12 | 3.1 | 946.2 | 15 | 12 | 0.8 | 994.3 |
| 8000 | 15 | 0 | - | - | - | 15 | 7 | 3.1 | 2347.6 | 15 | 15 | 0.0 | 370.5 |
| 9000 | 15 | 0 | - | - | - | 15 | 14 | 0.7 | 774.6 | 15 | 15 | 0.0 | 441.5 |
| 10000 | 15 | 0 | - | - | - | 15 | 15 | 0.0 | 564.5 | 14 | 14 | 0.0 | 485.7 |
| 12000 | 15 | 0 | - | - | - | 15 | 13 | 0.3 | 1542.9 | 15 | 14 | 1.3 | 985.4 |
| 14000 | 15 | 0 | - | - | - | 15 | 12 | 0.5 | 1530.0 | 15 | 15 | 0.0 | 827.6 |
| 16000 | 15 | 0 | - | - | - | 15 | 10 | 1.5 | 2394.8 | 15 | 14 | 1.0 | 1450.5 |
| 18000 | 15 | 0 | - | - | - | 14 | 12 | 0.1 | 2278.7 | 13 | 13 | 0.0 | 1527.2 |
| 20000 | 15 | 0 | - | - | - | 15 | 13 | 0.4 | 2445.6 | 14 | 14 | 0.0 | 1827.9 |
| Total/Avg | 210 | 5 | 4 | 0.6 | 2660.3 | 209 | 168 | 1.3 | 1410.0 | 205 | 197 | 0.3 | 731.5 |

Column "#" indicates the total number of instances tested in each group and columns "O" and "F" represent the number of optimal and feasible solutions obtained, respectively. The "R" and "CPU" columns represent, respectively, the average number of reshuffles and the average running time (in seconds) on instances in which a feasible solution is obtained. The "Tot/Avg." row indicates the totals of the columns showing absolute frequencies and the averages of the columns showing average values. The notation "-" is used when the averages cannot be calculated.

### 6.5.4 Performance of our best proposal: FRF$_\mathrm{H}$

Since we aim to obtain stowage plans in every case, we propose FRF$_\mathrm{H}$, consisting of the FRF algorithm with the solutions completed when necessary with the heuristic described in Section 6.4.4. Table 6.6 shows the results obtained by this algorithm on the instances used in the previous section. The time limit for FRF is fixed at 3600 seconds, and if no feasible solution has been obtained in that time, the layout in the ports already fixed is maintained and the layout of containers in the remaining ports is obtained by the heuristic procedure.

The solutions for $\text{FRF}_\text{H}$ match the solutions for FRF when the latter obtains a solution within the time limit and vary in the instances where it does not. We see that by increasing the running time by 500 seconds in the worst case, we obtain solutions for all instances. Furthermore, the quality of the solutions is good: four solutions with zero relocations are obtained and one with 13 relocations.

In addition, it can be seen in Table 6.6 that increasing the percentage of slots occupied by 40' containers increases the difficulty of the problem. This is due to the fact that 40' containers cannot be stacked on top of 20' containers. The instances with 25% of the slots occupied by 20' containers and 75% by 40' containers are solved in an average running time of 975 seconds and with an average of 0.8 relocations. The instances with 50% of the slots occupied by 20' and 50% by 40' containers are solved in an average running time of 755 seconds and with an average of 0.1 relocations. Finally, those with 75% of the slots occupied by 20' and 25% by 40' are solved in an average running time of 686 seconds, providing a solution with zero relocations in all cases.

### 6.5.5 A real ship case study

In the previous sections, we have considered a ship with a given number of bays and set the number of rows in each bay and the number of tiers per bay and row, since we do not have data on the exact distribution of ships of all those capacities. This section evaluates our algorithm on a real ship with a capacity of 7032 TEUs. The results obtained are shown in Table 6.7.

The proposed algorithm, $\text{FRF}_\text{H}$, obtains solutions in all the instances tested with an average of 0.2 relocations and in an average running time of less than 920 seconds. The instances in which the percentage of slots occupied by 20' containers is 25% and the percentage of slots occupied by 40' containers is 75% are all solved with zero relocations. The instances in which the occupancy rate is 50%-50% are solved with an average of 0.2 relocations and a maximum of 2 relocations. Finally, the instances in which the occupancy rate is 75%-25% are solved with an average of 0.4 relocations and a maximum of 4 relocations.

## 6.6 Concluding remarks

We studied a generalized container stowage planning problem with different container sizes, weights and stability constraints which, to the best of our knowledge, has never been addressed with integrated methods before. We provided integer programming formulations for this general problem and for three special cases: (i) identical container sizes and different container weights, (ii) identical container weights and different container sizes, and (iii) different container weights and sizes. Through the computational study conducted on these formulations, we observed that the difficulty of solving the problems increased considerably as containers of different sizes and stability constraints were included. In fact, CPLEX could not obtain any feasible solution for the medium-scale instances of the most general problem within the time limit of one hour.

Table 6.6: Performance of algorithm $\mathrm{FRF_H}$ on instances grouped by container ship capacity ($\Omega$) with 85% of load capacity. Large instances considering $Q_1 = Q_2 = 0.01$.

| | # | | | FRF | | | | | $FRF_H$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\Omega$ | 20' | 40' | # | F | Z | R | Max R | CPU | # | R | Ad. CPU |
| 2000 | 0.25 | 0.75 | 5 | 5 | 5 | 0.0 | 0 | 513.0 | - | - | - |
| 3000 | 0.25 | 0.75 | 5 | 5 | 4 | 0.6 | 3 | 183.4 | - | - | - |
| 4000 | 0.25 | 0.75 | 5 | 5 | 5 | 0.0 | 0 | 240.5 | - | - | - |
| 5000 | 0.25 | 0.75 | 5 | 4 | 4 | 0.0 | 0 | 565.2 | 1 | 13.0 | 41.4 |
| 6000 | 0.25 | 0.75 | 5 | 5 | 5 | 0.0 | 0 | 197.8 | - | - | - |
| 7000 | 0.25 | 0.75 | 5 | 5 | 3 | 1.6 | 6 | 1622.2 | - | - | - |
| 8000 | 0.25 | 0.75 | 5 | 5 | 5 | 0.0 | 0 | 366.9 | - | - | - |
| 9000 | 0.25 | 0.75 | 5 | 5 | 5 | 0.0 | 0 | 410.5 | - | - | - |
| 10000 | 0.25 | 0.75 | 5 | 4 | 4 | 0.0 | 0 | 415.4 | 1 | 0.0 | 142.8 |
| 12000 | 0.25 | 0.75 | 5 | 5 | 4 | 4.0 | 20 | 1372.3 | - | - | - |
| 14000 | 0.25 | 0.75 | 5 | 5 | 5 | 0.0 | 0 | 802.7 | - | - | - |
| 16000 | 0.25 | 0.75 | 5 | 5 | 4 | 3.0 | 15 | 1778.6 | - | - | - |
| 18000 | 0.25 | 0.75 | 5 | 4 | 4 | 0.0 | 0 | 1693.3 | 1 | 0.0 | 293.8 |
| 20000 | 0.25 | 0.75 | 5 | 5 | 5 | 0.0 | 0 | 1772.1 | - | - | - |
| | Total/Avg. | | 70 | 67 | 62 | 0.7 | 20 | 850.7 | 3 | 4.3 | 159.4 |
| 2000 | 0.50 | 0.50 | 5 | 5 | 4 | 0.4 | 2 | 551.8 | - | - | - |
| 3000 | 0.50 | 0.50 | 5 | 5 | 4 | 0.8 | 4 | 422.0 | - | - | - |
| 4000 | 0.50 | 0.50 | 5 | 5 | 5 | 0.0 | 0 | 96.0 | - | - | - |
| 5000 | 0.50 | 0.50 | 5 | 5 | 5 | 0.0 | 0 | 257.4 | - | - | - |
| 6000 | 0.50 | 0.50 | 5 | 5 | 5 | 0.0 | 0 | 398.4 | - | - | - |
| 7000 | 0.50 | 0.50 | 5 | 5 | 4 | 0.8 | 4 | 903.9 | - | - | - |
| 8000 | 0.50 | 0.50 | 5 | 5 | 5 | 0.0 | 0 | 368.1 | - | - | - |
| 9000 | 0.50 | 0.50 | 5 | 5 | 5 | 0.0 | 0 | 467.0 | - | - | - |
| 10000 | 0.50 | 0.50 | 5 | 5 | 5 | 0.0 | 0 | 500.8 | - | - | - |
| 12000 | 0.50 | 0.50 | 5 | 5 | 5 | 0.0 | 0 | 740.9 | - | - | - |
| 14000 | 0.50 | 0.50 | 5 | 5 | 5 | 0.0 | 0 | 900.4 | - | - | - |
| 16000 | 0.50 | 0.50 | 5 | 5 | 5 | 0.0 | 0 | 1278.4 | - | - | - |
| 18000 | 0.50 | 0.50 | 5 | 5 | 5 | 0.0 | 0 | 1392.7 | - | - | - |
| 20000 | 0.50 | 0.50 | 5 | 4 | 4 | 0.0 | 0 | 1903.0 | 1 | 0.0 | 232.4 |
| | Total/Avg. | | 70 | 69 | 66 | 0.1 | 4 | 710.2 | 1 | 0.0 | 232.4 |
| 2000 | 0.75 | 0.25 | 5 | 5 | 5 | 0.0 | 0 | 480.4 | - | - | - |
| 3000 | 0.75 | 0.25 | 5 | 5 | 5 | 0.0 | 0 | 86.0 | - | - | - |
| 4000 | 0.75 | 0.25 | 5 | 5 | 5 | 0.0 | 0 | 104.5 | - | - | - |
| 5000 | 0.75 | 0.25 | 5 | 5 | 5 | 0.0 | 0 | 156.1 | - | - | - |
| 6000 | 0.75 | 0.25 | 5 | 5 | 5 | 0.0 | 0 | 192.3 | - | - | - |
| 7000 | 0.75 | 0.25 | 5 | 5 | 5 | 0.0 | 0 | 456.8 | - | - | - |
| 8000 | 0.75 | 0.25 | 5 | 5 | 5 | 0.0 | 0 | 376.4 | - | - | - |
| 9000 | 0.75 | 0.25 | 5 | 5 | 5 | 0.0 | 0 | 447.1 | - | - | - |
| 10000 | 0.75 | 0.25 | 5 | 5 | 5 | 0.0 | 0 | 526.8 | - | - | - |
| 12000 | 0.75 | 0.25 | 5 | 5 | 5 | 0.0 | 0 | 843.1 | - | - | - |
| 14000 | 0.75 | 0.25 | 5 | 5 | 5 | 0.0 | 0 | 779.7 | - | - | - |
| 16000 | 0.75 | 0.25 | 5 | 5 | 5 | 0.0 | 0 | 1294.4 | - | - | - |
| 18000 | 0.75 | 0.25 | 5 | 4 | 4 | 0.0 | 0 | 2041.4 | 1 | 0.0 | 489.4 |
| 20000 | 0.75 | 0.25 | 5 | 5 | 5 | 0.0 | 0 | 1823.7 | - | - | - |
| | Total/Avg. | | 70 | 69 | 69 | 0.0 | 0 | 637.0 | 1 | 0.0 | 489.4 |
| | Total/Avg. | | 210 | 205 | 197 | 0.3 | 20 | 731.5 | 5 | 2.6 | 240.0 |

Column "$\Omega$" indicates the container ship's capacity, columns "L", "20' " and "40' ", the percentages of load capacity, and that of slots occupied by 20' and 40' containers, respectively. Columns "$Q_1$ and $Q_2$ indicate the maximum percentage of deviation between bow and stern ends and between starboard and port sides. Column "#" shows the total number of instances tested in each group and columns "F" and "Z" represent, respectively, the number of solutions obtained and the number of instances in which a solution with zero reshuffles is obtained. "R" and "Max R" columns represent the average and maximum number of reshuffles achieved, respectively. "CPU" column represents the average running time (in seconds). Column "#" of $\mathrm{FRF_H}$ represents the number of instances repaired with the simple heuristic,"R" its average number of reshuffles and "Ad. CPU" the average extra CPU time to obtain those feasible solutions. Row "Total/Avg." indicates the totals of the columns showing absolute frequencies and the averages of the columns showing average values. The notation "-" is used when the averages cannot be calculated.

Although the formulation did not perform well as a whole, exploiting its decomposable structure led to three successful matheuristics, namely, relax-and-fix (RF), insert-and-fix (IF), and fractional-relax-and-fix (FRF), which is a combination of the first two heuristics.

Table 6.7: Results obtained with the $\text{FRF}_{\text{H}}$ algorithm in a real case study with container ship capacity $\Omega = 7032$ TEUs considering 85% of load capacity and $Q_1 = Q_2 = 0.01$.

| | % | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| $\Omega$ | 20’ | 40’ | # | F | Z | R | Max R | CPU |
| 7032 | 0.25 | 0.75 | 10 | 10 | 10 | 0.00 | 0 | 782.65 |
| 7032 | 0.50 | 0.50 | 10 | 10 | 9 | 0.20 | 2 | 672.04 |
| 7032 | 0.75 | 0.25 | 10 | 10 | 9 | 0.40 | 4 | 1293.78 |
| | Total/Avg | | 30 | 30 | 28 | 0.20 | 4 | 916.16 |

Columns “20’ ” and “40’ ” indicate the percentages of slots occupied by 20’ and 40’ containers. The column “#” gives the total number of instances tested and “F" and “Z” represent the number of solutions obtained and the number of instances in which a solution with zero reshuffles is obtained. The “R” and “Max R" columns represent the average number and the maximum number of reshuffles achieved. The “CPU" column represents the average running time (in seconds). The “Total/Avg” row indicates the totals of the columns showing absolute frequencies and the averages of the columns showing average values. The notation “-” is used when the averages cannot be calculated.

On the small and medium-scale instances, the IF and FRF algorithms showed similar performance and clearly outperformed the RF heuristic. When compared on the large-scale instances, the FRF heuristic performed better than IF in average solution quality as well as in average computing time. In order to guarantee a feasible solution for any instance tested within the given time limit, we integrated FRF with a constructive heuristic procedure. The integrated procedure was able to solve all problem instances with up to 20,000 containers, which is equivalent to the capacity of the largest container ships available today. In addition to the randomly generated instances, we tested our best-performing algorithm on instances using data from a real ship, also obtaining high-quality solutions in all cases.

We observe from the experimental study that the algorithms have more difficulty in solving the instances with very small or very large ship capacities compared to the ones with medium level capacities. This could be explained by the fact that having larger capacities makes it possible to reach a high-quality feasible solution more easily up to a certain threshold, and after that the models become too large to be handled by the solvers. Moreover, the problems seem to be more challenging to solve as the percentage of 40’ containers becomes larger. This is indeed expected, as having more 40’ containers leads to less flexibility in the container layouts.

# Chapter 7

# Conclusions and future work

In this thesis we have studied two NP-hard combinatorial optimization problems that arise in the context of maritime container terminals. The first is the premarshalling problem, which arises in the yard area, and the second is the stowage problem, which arises in the seaside area. The primary aim of both problems is to reduce the berthing time of ships, which is currently the main indicator of the efficiency of container terminals. To solve both problems, exact and heuristic methods have been developed, addressing not only the standard problems previously defined in the scientific literature but also new problems with more realistic objectives and/or constraints, bringing theoretical results closer to the day-to-day operational reality of container terminals. This chapter summarizes the main contributions of the thesis in Section 7.1 and the future lines of work in Section 7.2.

## 7.1 Contributions

Chapters 2, 3, and 4 deal with the premarshalling problem, which consists in sorting the containers placed in the storage area of a container terminal in such a way that they can be retrieved afterwards without any additional reshuffling. The premarshalling process can be done before the arrival of a ship, when the workload at the terminal is at a minimum, so that no reshuffling needs to be carried out when ships are being loaded/unloaded, thus increasing the performance of the terminal when it is most needed.

In Chapter 2, several mathematical models were developed to solve the classical container premarshalling problem, CPMP, in which the objective is to minimize the number of moves in the reshuffling process, leading to a final arrangement in which the containers are directly available according to the loading sequence. Two kinds of models are proposed: integer linear models with one variable set to describe the container movements (IP3, IP4, IP5, and IP6), and integer linear models with two variable sets (IPS3, IPS4, IPS5, and IPS6). The proposed models require determining a value of $T$, an upper bound for the number of movements needed to rearrange the bay, plus one, because at the last period the bay must be completely ordered. Since the vast majority of heuristic algorithms developed so far add extra tiers to the instance, sometimes at the beginning of the procedure, sometimes when they are needed to allow further movements to be made, and heuristic algorithms that do not increase the bay size do not guarantee finding a feasible solution, the use of an upper bound provided by any of these heuristics is not clear. We therefore propose an iterative procedure to solve the models, avoiding the need for an upper bound to build them. We have tested our best proposal, IPS6, on several well-known datasets, and it clearly outperforms the state-of-the-art models by Lee and Hsu (2007) and de Melo da Silva et al. (2018), on several well-known datasets. The flexibility of the mathematical models is also highlighted in this chapter, and several extensions are proposed to satisfy possible additional practical constraints.

Chapter 3 also tackles the classical container premarshalling problem, CPMP, but in this case by providing a sophisticated exact method, a branch-and-bound algorithm, with novel components such as a new branching comparison algorithm, a memoization heuristic for finding feasible solutions, and new lower bounds and dominance rules. Overall, the approach proposed solves many more instances optimally than previous works, and finds feasible solutions for nearly every problem it encounters in limited CPU times, and optimal solutions for between 80.0% and 99.2% of the instances in industrially relevant instance categories, making heuristic methods for premarshalling unnecessary in practice or at most only necessary for solving very large instances faster.

A new variant of the premarshalling problem is presented in Chapter 4. This variant takes into account the objective of minimizing the crane time (CPMPCT), instead of minimizing the number of moves. A precise calculation of the crane time is made, considering the speed of the crane with and without load, as well as the acceleration and deceleration times. The data used correspond to the rubber-tired gantry (RTG) cranes used at the Noatum terminal in the port of Valencia. An extended computational analysis of various datasets shows that the number of movements is not a suitable indicator for measuring the time spent by the crane, since using the new objective function time reduces the time by 24% in some cases. To solve the CPMPCT, we propose two exact methods: a mathematical model, which optimally solves small instances; and a branch-and-bound algorithm, which provides a solution in 95% of the instances tested.

Chapters 5 and 6 deal with the container stowage planning problem, which consists in determining the position of the containers on board a ship along its route with the objective of minimizing the number of unproductive moves required in the loading and unloading operations at each port. These moves, also called reshuffles, arise at a port when some containers going to later ports have to be unloaded to gain access to containers that must be unloaded at that port and are situated below them. Once all the containers going to this port have been unloaded, the reshuffled containers are loaded again into the ship, together with all the containers that had to be loaded at this port. These movements are time and money consuming, so by reducing them the loading berthing times of the ships can be reduced, thus saving money.

Chapter 5 addresses the multi-port container stowage planning problem, CSPP. It considers that all the bays and rows of the ship have the same number of tiers and all the containers are the same size and it ignores the stability of the ship. Thus, container weight- and size-related constraints are not considered. Two types of contributions are provided:

a new linear integer model together with a set of valid constraints that enhance the linear relaxation of the model, and a GRASP metaheuristic algorithm with various randomization strategies and a local search tailored to the problem. An extended computational analysis was performed in which, to the best of our knowledge, the efficiency of integer programming models for the problem was tested for the first time. The results show that this model outperforms the state-of-the-art models. With respect to GRASP, the computational results show that it performs well on different-sized data sets, generating stowage plans with a minimum number of relocations in a high percentage of medium- and large-sized instances.

Chapter 6 tackles the multi-port container stowage planning problem, CSPP, considering container weight- and size-related constraints. In this chapter the ship is no longer seen as a rectangular matrix, since each bay and row can have a variable number of tiers to which containers of different weight and size can be allocated. A mathematical model is presented for this problem, as well as for three adaptations of the problem: the basic problem in which all containers are identical; the problem that considers a single container size, different container weights and ship stability; and the problem with two types of containers and no stability constraints. Through an extended study, we observed that the difficulty of solving the problems increases considerably as containers of different sizes and stability constraints are included. To solve large-scale instances, three matheuristics are presented, the Relax-and-Fix, Insert-and-Fix, and Fractional Relax-and-Fix algorithms, that exploit the structure of the mathematical model. The fractional-relax-and-fix method, in combination with a constructive procedure that gives the solver an initial solution in each iteration, provides very high-quality solutions in a reasonable time for instances with up to 20,000 TEUs, which is equivalent to the capacity of the largest container ships available today. In addition to the randomly generated instances, we tested our best performance algorithm on a real-case ship, also obtaining high-quality solutions in all instances.

## 7.2 Future lines of research

This thesis has developed an extensive study of mathematical models for the classical pre-marshalling problem, CPMP. Mathematical models can be a useful tool for professionals, as they can be easily implemented and extended by including new and interesting practical constraints in a simple way. Moreover, the ad-hoc algorithm presented for the CPMP provides optimal solutions in nearly all practical size instances, so we do not consider necessary to research new methodologies as metaheuristic algorithms for this problem. However, further research could integrate the problem into a robust framework (as in Tierney and Voß, 2016) to prevent the need for several iterations of the CPMP due to last-minute changes in container retrieval orders resulting from delays. Integration of the problem in a robust framework could also be considered for the CPMPCT, although for this problem there is still a wide range of possibilities. For example, in Chapter 4 we consider crane time, but the approaches developed could be used for other purposes such as minimizing energy consumption during the arrangement process. Moreover, further lines of research could develop new lower bounds for the problem and consider more complex heuristic procedures involving other types of movements such as BB moves, although that would require a careful new study, given that the runtime would certainly increase.

With respect to the multi-port container planning problem, the generalized problem considered in Chapter 6 can be extended in several ways, progressively adding characteristics of the real problem. Reefer containers have only a limited set of available slots, those with electric sockets, high-cube containers reduce the number of tiers in a stack, and containers with dangerous goods involve new strict stowage conditions. In real problems these container types have to be considered and their characteristics taken into account by proposed solution procedures. A more precise ship structure, with the inclusion of hatch covers and the constraints this would entail, should also be addressed, as well as a study of the stability of the ship with physical equations that consider the many forces acting on the ship. Although these features can be included in the integer linear models, the results obtained clearly indicate that the extended model will not be able to solve real-world instances. Developing new metaheuristic approaches that do not rely on the performance of integer programming solvers seems a more promising approach, because their flexibility and modest computing time requirements make them able to cope with more complex versions of the problem.

# Appendix A

# Resumen extendido

El desarrollo del contenedor ha revolucionado el comercio marítimo de mercancías, permitiendo la manipulación de carga de diversos tipos y dimensiones con un costo reducido y disminuyendo el costo de importación de muchos productos, hasta el punto de que puede resultar más barato transportarlos al otro lado del mundo que producirlos localmente. Hoy en día, aproximadamente el 90% de la carga no a granel en todo el mundo se transporta en buques portacontenedores, cuyas capacidades pueden sobrepasar los 20000 TEUs (*Twenty-foot Equivalent Unit*, unidad de medida correspondiente a un contenedor normalizado de 20 pies). El creciente aumento en el volumen de carga transportada y en el aumento del tamaño de los barcos, junto a la creación de mega alianzas que operan la mayor parte del transporte marítimo de contenedores, han desencadenado en una nueva dinámica en la que las compañías navieras tienen mayor poder de negociación e influencia sobre las terminales de contenedores, las cuales deben competir por menos servicios de barcos cada vez más grandes. La necesidad de manipular más contenedores en el mismo periodo de tiempo ejerce presión tanto en las operaciones de atraque como en las de patio. Las terminales de contenedores no pueden aumentar indefinidamente el número de grúas disponibles, por lo que su atención se centra en la optimización de los recursos existentes para reducir los tiempos de atraque.

Una forma eficaz de acelerar las operaciones de carga y descarga de los buques en las terminales de contenedores es utilizar el tiempo de inactividad de las grúas de patio, antes de la llegada de un buque, para organizar los contenedores. Un buen arreglo del patio contribuye significativamente a suavizar los picos de carga de trabajo, evitando la sobreutilización de la capacidad portuaria en algunos días y la subutilización en otros (Drewry Maritime Research, 2016), y también contribuye a aumentar la productividad de la terminal. El patio de contenedores es un gran espacio en el que se almacenan los contenedores de importación, antes de ser transferidos al interior en trenes o camiones, y los contenedores de exportación o transbordo, antes de ser cargados en los barcos. Se divide en varios bloques compuestos por bahías paralelas. Cada bahía tiene el mismo número de pilas de contenedores y cada pila tiene un número máximo de niveles que se fija dependiendo de la altura de las grúas de patio. Como consecuencia del apilamiento, para alcanzar un contenedor específico puede ser necesario tener que manipular los contenedores situados encima de él. En este contexto, el problema de premarshalling consiste en reordenar los contenedores en cada bahía del patio, con el fin de evitar un mayor número de recolocaciones a la llegada del buque y, de este modo, acelerar los tiempos de servicio.

Otra forma eficaz de acelerar la carga y descarga de los buques es dotarlos con planes de estiba eficientes que describan en qué posición debe cargarse cada contenedor a bordo. Estos planes deben satisfacer restricciones de alto nivel, relacionadas con la estabilidad de la nave; así como restricciones de bajo nivel, relacionadas con la forma en la que los contenedores deben ser apilados, la distribución del peso y otras condiciones específicas como las que regulan el transporte de mercancías peligrosas o mercancías IMO. Los buques portacontenedores navegan de puerto a puerto cargando y descargando miles de contenedores. Teniendo en cuenta los grandes volúmenes transportados, maximizar la eficiencia en la manipulación de los contenedores es crucial para la economía mundial y para el medio ambiente, ya que la reducción de operaciones portuarias innecesarias resulta en una gran reducción de la huella de carbono. En los planes de estiba, la posición de un contenedor se identifica mediante un sistema de coordenadas bahía-fila-nivel. Las bahías representan las secciones transversales del buque y están numeradas de proa a popa; las filas recorren la sección longitudinal o eslora y se representan con números impares desde el eje central a estribor y con números pares desde el eje central a babor. La última coordenada es el nivel o capa de los contenedores. Cada posición definida por estas tres coordenadas se denomina celda y puede albergar dos contenedores de 20 pies o uno de 40 pies. Debido al apilamiento, sólo es posible acceder a un contenedor si no hay otros contenedores por encima de él. De lo contrario, los contenedores que están encima deben ser descargados en puerto y cargados de nuevo si aún no se ha llegado a su puerto de entrega. Este tipo de movimientos improductivos de contenedores en puertos distintos a su puerto de descarga se denominan reubicaciones o recolocaciones y conllevan costes operativos a la terminal y un aumento del tiempo de atraque. En este contexto, el problema de la estiba multi-puerto busca obtener un plan de estiba del barco de modo que se reduzca al mínimo el número total de movimientos improductivos en las operaciones de carga y descarga del barco a lo largo de la ruta a la que presta servicio.

Esta tesis doctoral se centra principalmente en el problema de premarshalling y en el problema de la estiba, problemas de optimización combinatoria NP-difíciles que surgen en el patio y en el muelle de las terminales marítimas de contenedores, antes y durante las operaciones de carga y descarga de los buques, y cuya resolución deriva en una disminución del tiempo de atraque y, por lo tanto, en un aumento de la eficiencia de las terminales. Para resolver ambos problemas, se han desarrollado métodos exactos y heurísticos, que no sólo abordan los problemas previamente definidos en la literatura científica, sino también nuevos problemas con distintas funciones objetivo y nuevas generalizaciones y/o restricciones que acercan los resultados teóricos a la realidad del día a día de las terminales de contenedores. La finalidad de la tesis es doble. Por un lado, contribuir a mejorar la eficiencia de las terminales de contenedores marítimos. Por otro lado, aportar nuevas ideas y conocimientos al campo de la Investigación Operativa que puedan ser útiles para abordar otros problemas combinatorios de naturaleza similar. Tras un primer capítulo en el que se introduce al lector en el marco del comercio marítimo de contenedores, contextualizando y motivando el desarrollo de la tesis, los capítulos 2, 3 y 4 tratan el problema de premarshalling, los capítulos 5 y 6 el problema de la estiba, y el capítulo 7 presenta las contribuciones de la tesis y esboza las líneas de investigación y el trabajo futuro que podría derivarse de ella.

El problema clásico de premarshalling (CPMP) consiste en obtener la secuencia mínima de movimientos de contenedores necesaria para transformar la disposición inicial de una bahía en una disposición final sin que ningún contenedor bloquee la retirada de otros. Las simplificaciones, o supuestos básicos, con las que se ha estudiado el problema en la literatura científica son las siguientes:

- *Única bahía.* Se considera que el movimiento de la grúa entre bahías no sólo consume tiempo sino que también viola las restricciones de seguridad debido a las interacciones grúa-grúa o humano-grúa.
- *Coste de movimiento uniforme.* Como solo se tiene en cuenta el número de movimientos, todos los movimientos de las grúas tienen un coste o duración equivalente. Esta condición no es del todo cierta en la práctica, ya que algunos movimientos pueden realizarse entre pilas que están muy separadas entre sí.
- *Única grúa.* Se considera una única grúa que sólo puede mover un contenedor a la vez siendo todos los contenedores del mismo tipo.
- *Información completa.* Se supone que todos los grupos de contenedores son conocidos de antemano. De este modo, se puede asignar un número a cada contenedor con el orden en el que debe ser descargado. Como los retrasos son comunes en las operaciones portuarias, en realidad los grupos de algunos contenedores pueden cambiar, lo que podría causar nuevos desajustes incluso en patios previamente ordenados.

En el capítulo 2, *Integer programming models for the container premarshalling problem*, se aborda el problema clásico de premarshalling mediante el desarrollo de diferentes modelos de programación lineal entera que pueden ser declarados como modelos de programación lineal entera mixta debido a las relaciones existentes entre sus variables. Aparte de la ventaja conceptual de proporcionar una comprensión más profunda del problema, los modelos de programación lineal son útiles por tres razones: i) son fáciles de aplicar por los profesionales de los distintos sectores; ii) se pueden resolver con solucionadores comerciales, que han mostrado un aumento drástico de su desempeño en los últimos años, a menudo en cinco o incluso más órdenes de magnitud, y están en constante mejora; iii) son muy flexibles, lo que permite añadir o quitar restricciones para satisfacer requisitos específicos del problema que se está resolviendo. Las contribuciones del capítulo se encuentran publicadas en el siguiente artículo científico:

Parreño-Torres, C., Alvarez-Valdes, R., Ruiz, R., 2019. Integer programming models for the premarshalling problem. European Journal of Operational Research. 274, 142 - 154. doi: 10.1016/j.ejor.2018.09.048.

El estudio de los modelos enteros para el problema se compone principalmente de dos fases. En una primera fase, desarrollamos una serie de modelos en los que, además de las variables que describen la configuración de la bahía, se describe otro conjunto de variables que describen los movimientos necesarios para reordenar la bahía. Diferentes formas de definir el conjunto de variables que describen los movimientos definen diferentes modelos de programación lineal entera para el problema. Se presentan cuatro modelos -IP3, IP4, IP5 y

IP6- cuya complejidad va en aumento, mejorando la relación entre las variables y la relajación lineal de los modelos a expensas de ampliar el número total de variables. En una segunda fase, el conjunto de variables que describe los movimientos se divide en dos subconjuntos, uno que indica el origen del movimiento y otro que indica el destino del contenedor movido. Cada uno de los cuatro modelos presentados en la primera fase tiene su equivalente en esta nueva fase -IPS3, IPS4, IPS5 y IPS6-. Esta estrategia alternativa ha demostrado ser mucho más eficiente, en cuanto a número de soluciones óptimas obtenidas y también en cuanto a tiempos de ejecución.

Los modelos propuestos requieren la determinación de un valor de $T$, un límite superior para el número de movimientos necesarios para reordenar la bahía más uno, ya que en el último período la bahía debe estar completamente ordenada. Una cota o límite superior para este valor $T$ podría ser proporcionada por un algoritmo heurístico. Sin embargo, su validez no es clara. Por una parte, el número de variables y, por lo tanto, el rendimiento de los modelos dependen directamente del valor de esa cota. Por ello, si se usa un límite superior proporcionado por una heurística, el rendimiento del modelo dependerá del rendimiento de la heurística usada. Por otra parte, está la cuestión no resuelta de decidir si una instancia de premarshalling tiene o no solución posible. Esta cuestión depende no sólo de las dimensiones de la bahía y del número de contenedores, sino también de las posiciones de los contenedores. La gran mayoría de los algoritmos heurísticos evitan la cuestión añadiendo niveles adicionales a las instancias, a veces al principio del procedimiento, a veces cuando se necesitan, para permitir nuevos movimientos y los algoritmos heurísticos que no añaden niveles adicionales no garantizan que se encuentre una solución viable en todos los tipos de instancias. En una situación real, el número máximo de niveles de la bahía está impuesto por la altura de la grúa y un valor obtenido mediante la modificación de la instancia original no puede considerarse un límite superior válido. Para evitar la necesidad de obtener un límite superior para construir los modelos, proponemos un procedimiento iterativo que comienza fijando el valor de $T$ como una cota inferior para el número de movimientos más uno y va aumentando ese valor hasta obtener una solución factible para el problema. El primer valor para el que el procedimiento da una solución factible se corresponde con el número mínimo de movimientos para ordenar la bahía y la solución proporcionada es óptima. Haciendo uso de este procedimiento de resolución, un amplio estudio computacional muestra que nuestra mejor propuesta, IPS6, supera claramente los modelos previamente publicados por Lee and Hsu (2007) y de Melo da Silva et al. (2018) sobre varios conjuntos de datos conocidos. También se destaca en este capítulo la flexibilidad de los modelos matemáticos mediante la propuesta de varias extensiones, adaptándolos a características o restricciones particulares que pueden surgir en la práctica.

Pese a las claras ventajas que presentan los modelos de programación lineal por su flexibilidad y su fácil implementación, no son capaces de resolver problemas de tamaño real para el problema de premarshalling. Además, aunque en Tierney et al. (2017) y Tanaka and Tierney (2018) se presentan avances significativos en la resolución de problemas de gran tamaño mediante métodos exactos sofisticados, hay varias clases de premarshalling que siguen resultando difíciles para los algoritmos propuestos por estos autores. En el capítulo 3, *A branch-and-bound approach for large premarshalling problems*, proponemos un nuevo algoritmo Branch and Bound adaptado a la estructura del problema, capaz de resolver las clases de instancias más difíciles. Este algoritmo obtiene soluciones factibles para casi todas las instancias probadas y soluciones óptimas entre el 80% y el 99,2% de las instancias perteneciente a categorías relevantes para la industria. El capítulo está basado en el siguiente artículo científico:

Tanaka, S., Tierney, K., Parreño-Torres, C., Alvarez-Valdes, R., Ruiz, R., 2019. A branch and bound approach for large premarshalling problems. European Journal of Operational Research 278, 211 - 225. doi: 10.1016/j.ejor.2019.04.005

Proponemos un algoritmo Branch and Bound en el que la estrategia de búsqueda consiste en ejecutar repetidamente una búsqueda en profundidad limitada incrementando el límite de profundidad en cada iteración. Se comienza con un límite de profundidad igual a una cota inferior para el número de movimientos y se incrementa en uno en cada iteración hasta que se alcanza una solución factible con un número de movimientos igual al valor actual del límite. En ese caso, la optimalidad de la solución está garantizada. Este tipo de búsqueda se conoce en ciencias de la computación como búsqueda en profundidad iterativa y permite combinar la eficiencia de la búsqueda en profundidad y la completitud de la búsqueda en anchura. Entre las nuevas componentes que presenta el algoritmo propuesto, $\mathrm{PT}^{\mathrm{A}}$, se encuentra la extensión de la cota inferior $\mathrm{IBF}^1$ propuesta por Tanaka and Tierney (2018), probando que puede incrementarse en uno en algunos casos, $\mathrm{IBF}^2$, y se presentan dos nuevas cotas inferiores, $\mathrm{IBF}^3$ y $\mathrm{IBF}^4$, que se basan en la relajación lineal del problema. También se presenta una nueva regla de dominancia que hace uso de las cotas superiores e inferiores de un nodo para disminuir el espacio de soluciones, así como una generalización de las reglas de dominancia presentadas en Tanaka and Tierney (2018), presentando para ello el concepto de pila invariante a una secuencia de movimientos. Aunque el orden de ramificación es irrelevante para la validez del procedimiento, una buena estrategia es indispensable para acelerar la obtención de soluciones. Por este motivo, el algoritmo hace uso de hasta siete nuevos criterios de ramificación para ordenar las ramas. Por último, se presenta e integra una heurística basada en memoria capaz de obtener soluciones factibles del problema.

Un extenso estudio computacional, en el que se consideran los conjuntos de instancias de referencia en la literatura junto a un nuevo conjunto de instancias que evita el sobreajuste u *overfitting* muestra que nuestro algoritmo funciona mejor que el propuesto por Tanaka and Tierney (2018). Se reduce drásticamente el espacio de búsqueda, encontrando un 8% más de soluciones óptimas y un 6% más de soluciones factibles en las instancias probadas. Además, es capaz de resolver de forma óptima la gran mayoría de las instancias con 7 niveles y 9-10 pilas, consideradas como instancias difíciles en la práctica. El buen desempeño del algoritmo desarrollado hace que los métodos heurísticos para el problema clásico de premarshalling sean innecesarios o, a lo sumo, solo necesarios para resolver en cuestión de segundos instancias de tamaño muy grande. De este modo, no consideramos como línea futura el desarrollo de otras metodologías como algoritmos metaheurísticos. El trabajo futuro en torno al problema clásico debería ir más en la línea de Tierney and Voß, 2016 e integrarlo en un marco robusto para evitar la necesidad de múltiples iteraciones debido a cambios de última hora en el orden de recuperación de los contenedores.

El coste de movimiento uniforme es una de las simplificaciones con las que se ha estudiado el problema en la literatura científica. El número de movimientos se ha utilizado tradicionalmente como indicador del tiempo empleado por la grúa para reorganizar la bahía. Sin embargo, en muchos casos, el número de movimientos no es totalmente representativo de los tiempos de la grúa (Lin et al. (2015); da Silva Firmino et al. (2019); Jovanovic et al. (2019b)). Por ejemplo, lleva mucho más tiempo mover un contenedor entre las posiciones más lejanas de una bahía que hacerlo desde el nivel superior de una pila al nivel superior de una pila adyacente. Además, esos movimientos difieren no sólo en la cantidad de tiempo que requiere la grúa, sino también en cuanto al consumo de energía. Según Wilmsmeier and Spengler (2016), las actividades horizontales como las realizadas por las grúas de patio representan un 45% del total energía consumida por una terminal de contenedores. Las conclusiones de ese estudio ponen de manifiesto la necesidad de adoptar medidas para abordar la competitividad, la seguridad energética y el cambio climático en las terminales de contenedores de todo el mundo. Dado que gran parte de la energía de la terminal es consumida por las grúas de patio, se hace evidente la importancia de reducir el tiempo real que emplean las grúas en los movimientos para apoyar las nuevas y necesarias políticas portuarias ecológicas.

En el capítulo 4, *Minimizing the crane time in premarshalling problems*, se presenta una nueva variante del problema de premarshalling cuyo objetivo consiste en minimizar el tiempo empleado por la grúa para transformar la bahía inicial en una sin contenedores de bloqueo (CPMPCT), en lugar de minimizar el número de movimientos requeridos (CPMP). Dado que el problema clásico es un problema NP-difícil, esta variante también lo es ya que puede reducirse al primero considerando tiempos unitarios en los movimientos de la grúa. Este capítulo está basado en la siguiente publicación científica:

Parreño-Torres, C., Alvarez-Valdes, R., Ruiz, R., Tierney, K., 2020. Minimizing the crane time in premarshalling problems. Transportation Research Part E 137, Art. ID 101917. doi: 10.1016/j.tre.2020.101917.

Se realiza un cálculo preciso del tiempo de la grúa, haciendo uso de las características técnicas de las grúas pórtico sobre neumáticos RTG utilizadas en la terminal de Noatum del puerto de Valencia. Para realizar el cálculo se tiene en cuenta la velocidad de la grúa en el eje vertical y horizontal, con y sin carga, así como los tiempos de aceleración y desaceleración. Además, se considera el tiempo de enganche del contenedor, tiempo de *twistlock*, que es proporcional al nivel en el que se encuentra el contenedor que se desea mover. Esto se debe a la oscilación del spreader de la grúa por el movimiendo de balanceo de la carga suspendida. Aunque algunas grúas emplean sistemas anti-balanceo que detienen los péndulos y el movimiento de oscilación rotacional, este sistema no se da en las grúas pórtico estudiadas por lo que los operadores tienen que esperar hasta que la oscilación se detenga para poder enganchar el contenedor. Esta oscilación depende de la longitud total del cable.

Para resolver el problema de manera óptima proponemos dos enfoques diferentes: un modelo de programación lineal entera y un algoritmo Branch and Bound. La modelización matemática propuesta, IPCT, parte de la descripción de las variables propuestas para el modelo IPS6, que resuelve el problema clásico, e incluye dos nuevos conjuntos. Un conjunto de variables para vincular la pila de origen y destino del contenedor movido y otro conjunto de variables para vincular la pila de destino de un contenedor con la pila de origen del siguiente contenedor a mover. De nuevo, el modelo propuesto depende de la determinación de un valor de $T$, un límite superior para el número de movimientos necesarios para reordenar la bahía más uno. En este caso, no es posible evitar el cálculo de $T$ mediante el uso de un procedimiento iterativo como el presentado en el capítulo 2 ni fijar ese valor como el número de movimientos de cualquier solución factible para el problema clásico, ya que no nos permitiría garantizar la optimalidad de la solución encontrada. Esto es debido a que una solución óptima para el problema de premarshalling considerando el coste de grúa puede conllevar un mayor número de movimientos que la solución óptima para el problema clásico de premarshalling. Para abordar este asunto, desarrollamos dos cotas superiores para el número de movimientos necesario para resolver el problema de premarshalling con tiempos de grúa, CPMPCT. Además, se proponen límites inferiores para el problema y se discuten las reglas de dominancia propuestas en trabajos académicos anteriores y su adaptabilidad o no a esta nueva función objetivo. Las cotas superiores e inferiores y las reglas de dominancia propuestas para el problema, junto a un algoritmo heurístico que trata de ordenar las disposiciones intermedias de la bahía realizando distintos tipos de movimientos, se integran en un algoritmo Branch and Bound, CTA. Durante la búsqueda de soluciones, el algoritmo no puede garantizar la optimalidad en el momento que se alcanza una solución con un número de movimientos igual al valor actual del límite de profundidad, sino que debe continuar iterando hasta que el límite iguale a la cota superior del número de movimientos obtenida para resolver el problema con tiempos de grúa.

Un extenso análisis computacional sobre varios conjuntos de instancias muestra que el número de movimientos no es un indicador adecuado para medir el tiempo empleado por la grúa, ya que mediante la nueva función objetivo el tiempo de grúa se reduce hasta un 24% en algunos casos. Además, el algoritmo Branch and Bound supera en desempeño a la modelización matemática y obtiene solución en el 95% de los casos probados y solución óptima en las instancias de tamaño relevante en la práctica. Con respecto a este problema, sí que consideramos que existe un gran abanico de posibilidades para líneas de trabajo futuras. Desde el estudio de nuevos límites y heurísticas rápidas más completas que se puedan integrar en métodos exactos sofisticados y permitan resolver las instancias más grandes de manera óptima, al desarrollo de otras metodologías como algoritmos mateheurísticos o a la integración del problema en un marco robusto como se ha propuesto para el problema clásico. Asimismo, en este capítulo consideramos el tiempo de grúa, pero los enfoques desarrollados podrían utilizarse para otros fines, como la reducción del consumo de energía durante el proceso de recolocación de los contenedores.

Los capítulos 5 y 6 tratan el problema de planificación de la estiba en buques portacontenedores, que consiste en determinar la posición de los contenedores a bordo de un barco a lo largo de su ruta. Consideramos un barco que tiene que visitar un conjunto de puertos en un orden determinado para recoger y entregar contenedores. El barco comienza su ruta en el primer puerto, donde se debe cargar un conjunto inicial de contenedores. Tras este, recorre una serie de puertos en los que se realiza operaciones de carga y descarga. Finalmente, en el último puerto, se produce la descarga de todos los contenedores restantes. Al conocerse el puerto de recogida y entrega de cada contenedor, se considera como datos de entrada el número total de contenedores que se cargarán y descargarán en cada puerto. Consideramos un sistema centralizado que tiene como objetivo minimizar el número total de movimientos improductivos a lo largo de la ruta del barco, permitiendo reubicaciones no estrictamente necesarias en un puerto si su realización conlleva una reducción del total de movimientos.

El capítulo 5, *Solution strategies for a multi-port container stowage planning problem*, aborda el problema básico de planificación de la estiba, donde todos los contenedores son de idéntico tamaño y no existen limitaciones adicionales como estabilidad, peso, etc. Las contribuciones del capítulo han sido publicadas en el siguiente documento:

Parreño-Torres, C., Alvarez-Valdes, R., Parreño, F, 2019. Solution strategies for a multi-port container ship stowage problem. Mathematical Problems in Engineering. Art. ID 9029267. doi: 10.1155/2019/9029267.

En este trabajo consideramos que todas las bahías y filas del barco tienen el mismo número de niveles. De este modo, podemos representar el barco como una matriz rectangular cuyo número de filas es igual al número de niveles y cuyo número de columnas es igual al producto del número total de bahías por el número total de filas del barco. Para resolver el problema, presentamos básicamente dos tipos de contribuciones. En primer lugar, desarrollamos un nuevo modelo de programación lineal entera junto con un conjunto de restricciones válidas que mejoran su relajación lineal. El modelo propuesto, ILR, requiere un menor número de variables que los modelos propuestos por Avriel et al. (1998) y por Ding and Chou (2015) y proporciona mejores resultados en cuanto al número de soluciones óptimas obtenidas y en cuanto a la calidad de las soluciones factibles. Sin embargo, y aunque supera claramente los modelos existentes, nuestra modelización no es capaz de producir soluciones para instancias de gran tamaño en un tiempo máximo de computación de una hora. Por lo tanto, en una segunda parte del estudio, desarrollamos un algoritmo metaheurístico GRASP, que va un paso más allá de los algoritmos constructivos propuestos hasta el momento para el problema e incluye estrategias de aleatorización y una búsqueda local adaptada a sus características. Un extenso análisis computacional muestra que el algoritmo propuesto genera planes de estiba con un mínimo número de reubicaciones en un alto porcentaje de instancias de tamaño medio y grande y obtiene soluciones expremedamenente buenas en tiempos de computación muy cortos, menos de 5 minutos para las instancias más grandes con 22 puertos y hasta 20000 contenedores.

El problema de la estiba ha sido estudiado a lo largo de las años mediante métodos de Investigación Operativa siguiendo dos enfoques claramente diferenciados. Por un lado, se encuentran los trabajos que descomponen el problema en subproblemas de optimización que pueden resolverse individualmente. Una práctica común en este enfoque es resolver primero un problema de asignación de contenedores a diferentes secciones del barco y después resolver otro problema para determinar la posición exacta que ocupa cada contenedores dentro de esas secciones. Al descomponer el problema en subproblemas, se permite incluir características más realistas: en la función objetivo, considerando por ejemplo costes relacionados con los contenedores no cargados, recolocaciones o el uso de grúas de muelle; en las restricciones, considerando por ejemplo la estabilidad del barco. Por otro lado, los trabajos que consideran el problema íntegro, considerando generalmente rutas con muchos puertos y buques de gran capacidad, pero con importantes simplificaciones como considerar un único tipo de contenedor y no considerar las condiciones de estabilidad del barco. En este úlimo grupo se encuentra el trabajo del capítulo descrito anteriormente. En el capítulo 6, *Mathematical models and matheuristics for a generalized multi-port container stowage planning problem*, se estudia una generalización del problema de la estiba, considerando distintos tamaños de contenedores, pesos y restricciones de estabilidad del barco. Al incluir distintos tamaños de contenedores, debemos considerar restricciones adicionales de apilado, impuestas por seguridad, como las que regulan la existencia de pilas mixtas o rusas en las que dos contenedores de 20 pies no pueden estibarse sobre uno de 40 pies pero sí a la inversa. En este capítulo, se pretende llenar el vacío existente en la literatura científica entre los problemas descompuestos con limitaciones realistas y los problemas integrados simplificados. Las contribuciones alcanzadas se han recogido en un artículo que actualmente se encuentra en proceso de revisión:

Parreño-Torres, C., Çalik, H., Alvarez-Valdes, R., Ruiz, R., 2020. Mathematical models and matheuristics for a generalized multi-port container stowage planning problem. Submitted to Computers & Operations Research.

El buque ya no se considera como una matriz rectangular, ya que cada bahía y fila puede tener un número variable de niveles a los que pueden asignarse contenedores de diferente peso y tamaño. Se estudia cómo cambia la complejidad del problema de la estiba con y sin la consideración de las limitaciones de tamaño y peso de los contenedores. Para ello, se proponen formulaciones de programación entera para el problema simplificado, en el que sólo se considera un tipo de contenedor y no se tienen en cuenta restricciones relacionadas con el peso como la estabilidad del barco, así como para tres adaptaciones del problema: el problema simplificado en el que todos los contenedores tienen el mismo tamaño y se consideran distintos pesos y restricciones de estabilidad; el problema en el que se consideran dos tamaños diferentes de contenedores, de 20 y 40 pies, pero no se consideran restricciones de peso; y el problema generalizado que considera dos tipos de contenedores, de 20 y 40 pies, y las restricciones de estabilidad que van ligadas a diferentes pesos de contenedores. Mediante un estudio computacional en el que se ha evaluado el desempeño de estas formulaciones, observamos que la dificultad de resolver los problemas aumenta considerablemente al incluir contenedores de diferentes tamaños y limitaciones de estabilidad. De hecho, CPLEX no obtiene ninguna solución factible para los casos de mediana escala del problema más general en el plazo de una hora. Para resolver instancias de gran escala, se presentan tres algoritmos mateheurísticos que explotan la estructura de la relajación lineal del problema: el Relax-and-Fix (RF), el Insert-and-Fix (IF) y el Fractional Relax-and-Fix (FRF). Además, proponemos un procedimiento constructivo que permite dar al solucionador comercial una solución inicial en cada iteración.

El estudio computacional llevado a cabo revela que los algoritmos IF y FRF muestran un rendimiento similar en las instancias de pequeña y mediana escala y superan claramente al algoritmo RF. Cuando se compararon en las instancias de gran escala, el FRF tuvo un mejor rendimiento que IF tanto en calidad media de la soluciones obtenidas como en tiempo medio de computación. Con el objetivo de garantizar la obtención de una solución factible en las instancias probadas, combinamos el FRF con el procedimiento heurístico constructivo. De este modo, el procedimiento combinado fue capaz de resolver todas las instancias con un tamaño equivalente a la capacidad de los mayores buques portacontenedores disponibles hoy en día. Además de las instancias generadas aleatoriamente, probamos nuestro mejor algoritmo utilizando los datos de un barco real, obteniendo también soluciones de alta calidad en todos los casos. En base al estudio computacional, podemos observar que los algoritmos propuestos presentan mayor dificultad en la resolución de instancias para barcos de tamaño muy pequeño o muy grande, con respecto a las soluciones obtenidas para los barcos de capacidad media. Esto podría explicarse por el hecho de que al aumentar la capacidad del barco hay más posibilidades para obtener soluciones factibles de gran calidad. No obstante, al llegar a un determinado umbral, los modelos se vuelven demasiado grandes para ser manejados por los solucionadores comerciales. Además, la dificultad de los problemas aumenta a medida que aumenta el porcentaje de contenedores de 40 pies y se reduce el de 20 pies. Esto se explica por el hecho de que un mayor número de contenedores de 40 pies conlleva una menor flexibilidad en las configuraciones del barco. Con respecto al trabajo futuro relacionado con este problema, consideramos que pueden adoptarse diferentes líneas. Por ejemplo, añadir progresivamente características prácticas como la carga de contenedores frigoríficos, que solo pueden ser ubicados en un conjunto limitado de posiciones (aquellos con enchufes eléctricos); la carga de contenedores más altos de lo habitual, que reducen el número de niveles en una pila; o la carga de contenedores de mercancías peligrosas, que implican estrictas condiciones de estiba. Otra línea podría ser considerar un estructura más precisa del barco, con la inclusión de las escotillas y las limitaciones que ello supondría, o mejorar la restricciones de estabilidad del barco, considerando ecuaciones físicas que tengan en cuenta las distintas fuerzas que actúan sobre él. En cuanto al desarrollo de diferentes metodologías, los resultados que hemos obtenido indican que aunque los modelos puedan soportar la inclusión de nuevas restricciones, claramente no serán capaces de resolver instancias de tamaño real. Por ello, parece más prometedor el desarrollo de nuevos enfoques metaheurísticas que no dependan en su rendimiento de modelos matemáticos.

# Bibliography

Ambrosino, D., Anghinolfi, D., Paolucci, M., Sciomachen, A., 2010. An experimental comparison of different heuristics for the master bay plan problem. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 6049 LNCS, 314 – 325. doi:10.1007/978-3-642-13193-6_27.

Ambrosino, D., Paolucci, M., Sciomachen, A., 2015. Experimental evaluation of mixed integer programming models for the multi-port master bay plan problem. Flexible Services and Manufacturing Journal 27, 263 – 284. doi:10.1007/s10696-013-9185-4.

Ambrosino, D., Paolucci, M., Sciomachen, A., 2017. Computational evaluation of a MIP model for multi-port stowage planning problems. Soft Computing 21, 1753 – 1763. doi:10.1007/s00500-015-1879-y.

Ambrosino, D., Sciomachen, A., Tanfani, E., 2004. Stowing a containership: the master bay plan problem. Transportation Research Part A 38, 81–99. doi:10.1016/j.tra.2003.09.002.

de Araujo, S., Arenales, M., Clark, A., 2008. Lot sizing and furnace scheduling in small foundries. Computers and Operations Research 35, 916 – 932. doi:10.1016/j.cor.2006.05.010.

Avriel, M., Penn, M., 1993. Exact and approximate solutions of the container ship stowage problem. Computers & industrial engineering 25, 271 – 274. doi:10.1016/0360-8352(93)90273-Z.

Avriel, M., Penn, M., Shpirer, N., 2000. Container ship stowage problem: complexity and connection to the coloring of circle graphs. Discrete Applied Mathematics 103, 271 – 279. doi:10.1016/S0166-218X(99)00245-0.

Avriel, M., Penn, M., Shpirer, N., Witteboon, S., 1998. Stowage planning for container ships to reduce the number of shifts. Annals of Operations Research 76, 55–71. doi:10.1023/A:1018956823693.

Azevedo, A., Neto, L., Chaves, A., Moretti, A., 2018. Solving the 3D stowage planning problem integrated with the quay crane scheduling problem by representation by rules and genetic algorithm. Applied Soft Computing Journal 65, 495–516. doi:10.1016/j.asoc.2018.01.006.

Azevedo, A., Ribeiro, C., De Sena, G., Chaves, A., Neto, L., Moretti, A., 2014. Solving the 3D container ship loading planning problem by representation by rules and meta-heuristics. International Journal of Data Analysis Techniques and Strategies 6, 228–260. doi:10.1504/IJDATS.2014.063060.

Bacci, T., Mattia, S., Ventura, P., 2019. The bounded beam search algorithm for the block relocation problem. Computers and Operations Research 103, 252 – 264. doi:10.1016/j.cor.2018.11.008.

Bierwirth, C., Meisel, F., 2010. A survey of berth allocation and quay crane scheduling problems in container terminals. European Journal of Operational Research 202, 615 – 627. doi:10.1016/j.ejor.2009.05.031.

Bixby, R.E., 2002. Solving real-world linear programs: A decade and more of progress. Operations Research 50, 3 – 15. doi:10.1287/opre.50.1.3.17780.

Bortfeldt, A., Forster, F., 2012. A tree search procedure for the container pre-marshalling problem. European Journal of Operational Research 217, 531 – 540. doi:10.1016/j.ejor.2011.10.005.

Böse, J.W., 2011. Handbook of terminal planning. volume 49. Springer Science & Business Media.

Boysen, N., Emde, S., 2016. The parallel stack loading problem to minimize blockages. European Journal of Operational Research 249, 618 – 627. doi:10.1016/j.ejor.2015.09.033.

van Brink, M., van der Zwaan, R., 2014. A branch and price procedure for the container premarshalling problem, in: Schulz, A., Wagner, D. (Eds.), Algorithms - ESA 2014. Springer Berlin Heidelberg. volume 8737 of *Lecture Notes in Computer Science*, pp. 798–809. doi:10.1007/978-3-662-44777-2_66.

Carlo, H.J., Vis, I.F., Roodbergen, K.J., 2014. Storage yard operations in container terminals: Literature overview, trends, and research directions. European journal of operational research 235, 412 – 430. doi:10.1016/j.ejor.2013.10.054.

Carlo, H.J., Vis, I.F., Roodbergen, K.J., 2015. Seaside operations in container terminals: literature overview, trends, and research directions. Flexible Services and Manufacturing Journal 27, 224 – 262. doi:10.1007/s10696-013-9178-3.

Cartenì, A., De Luca, S., 2012. Tactical and strategic planning for a container terminal: Modelling issues within a discrete event simulation approach. Simulation Modelling Practice and Theory 21, 123 – 145. doi:10.1016/j.simpat.2011.10.005.

Caserta, M., Schwarze, S., Voß, S., 2011. Container rehandling at maritime container terminals, in: Böse, J. (Ed.), Handbook of Terminal Planning. Springer, New York. volume 49 of *Operations Research/Computer Science Interfaces Series*, pp. 247–269.

Caserta, M., Voß, S., 2009. A corridor method-based algorithm for the pre-marshalling problem, in: Giacobini, M., Brabazon, A., Cagnoni, S., Di Caro, G.A., Ekárt, A., Esparcia-Alcázar, A.I., Farooq, M., Fink, A., Machado, P. (Eds.), Applications of Evolutionary Computing. Springer Berlin Heidelberg, pp. 788 – 797. doi:10.1007/978-3-642-01129-0_89.

Choe, R., Park, T., Oh, M.S., Kang, J., Ryu, K.R., 2011. Generating a rehandling-free intra-block remarshaling plan for an automated container yard. Journal of Intelligent Manufacturing 22, 201 – 217. doi:10.1007/s10845-009-0273-y.

Clarkson, 2019. The Supply Vessel Register 2019. Clarkson Research Services.

Correcher, J.F., Alonso, M.T., Alvarez-Valdes, R., Parreño, F., 2017. Solving a large multi-container loading problem in the car manufacturing industry. Computers and Operations Research 82, 139 – 152. doi:10.1016/j.cor.2017.01.012.

Delgado, A., Jensen, R.M., Janstrup, K., Rose, T.H., Andersen, K.H., 2012. A constraint programming model for fast optimal stowage of container vessel bays. European Journal of Operational Research 220, 251 – 261. doi:10.1016/j.ejor.2012.01.028.

Diaz, J.A., Luna, D.E., Camacho-Vallejo, J.F., Casas-Ramirez, M.S., 2017. GRASP and hybrid GRASP-Tabu heuristics to solve a maximal covering location problem with customer preference ordering. Expert Systems with Applications 82, 67 – 76. doi:10.1016/j.eswa.2017.04.002.

Ding, D., Chou, M.C., 2015. Stowage planning for container ships: A heuristic algorithm to reduce the number of shifts. European Journal of Operational Research 246, 242 – 249. doi:10.1016/j.ejor.2015.03.044.

Dragović, B., Tzannatos, E., Park, N.K., 2017. Simulation modelling in ports and container terminals: literature overview and analysis by research field, application area and tool. Flexible Services and Manufacturing Journal 29, 4 – 34. doi:10.1007/s10696-016-9239-5.

Drewry Maritime Research, 2016. Global Container Terminal Operators Annual Review and Forecast 2016. Annual Review and Forecast.

Dubrovsky, O., Levitin, G., Penn, M., 2002. A genetic algorithm with a compact solution encoding for the container ship stowage problem. Journal of Heuristics 8, 585 – 599. doi:10.1023/A:1020373709350.

Expósito-Izquierdo, C., Melián-Batista, B., Moreno-Vega, M., 2012. Pre-Marshalling Problem: Heuristic solution method and instances generator. Expert Systems with Applications 39, 8337 – 8349. doi:10.1016/j.eswa.2012.01.187.

Galle, V., Barnhart, C., Jaillet, P., 2018. Yard crane scheduling for container storage, retrieval, and relocation. European Journal of Operational Research 271, 288 – 316. doi:10.1016/j.ejor.2018.05.007.

Gharehgozli, A., Laporte, G., Yu, Y., De Koster, R., 2014. Scheduling twin yard cranes in a container block. Transportation Science 49, 141223041352002. doi:10.1287/trsc.2014.0533.

Gharehgozli, A., Zaerpour, N., de Koster, R., 2019. Container terminal layout design: transition and future. Maritime Economics & Logistics , 1 – 30doi:10.1057/s41278-019-00131-9.

Gheith, M., Eltawil, A.B., Harraz, N.A., 2016. Solving the container pre-marshalling problem using variable length genetic algorithms. Engineering Optimization 48, 687 – 705. doi:10.1080/0305215X.2015.1031661.

Günther, H.O., Kim, K.H., 2005. Container Terminals and Automated Transport Systems. Springer. doi:10.1007/978-3-540-49550-5_1.

Gupta, N., Nau, D., 1992. On the complexity of blocks-world planning. Artificial Intelligence 56, 223 – 254. doi:10.1016/0004-3702(92)90028-V.

Hansen, J.R., Fagerholt, K., Stålhane, M., 2017. A shortest path heuristic for evaluating the quality of stowage plans in roll-on roll-off liner shipping, in: International Conference on Computational Logistics, Springer. pp. 351 – 365.

Ho, S.C., Szeto, W.Y., 2016. GRASP with path relinking for the selective pickup and delivery problem. Expert Systems with Applications 51, 14 – 25. doi:10.1016/j.eswa.2015.12.015.

Hornell, J., 1946. Water transport: origins and early evolution. Cambridge.

Hottung, A., Tanaka, S., Tierney, K., 2020. Deep learning assisted heuristic tree search for the container pre-marshalling problem. Computers & Operations Research 113, 104781. doi:10.1016/j.cor.2019.104781.

Hottung, A., Tierney, K., 2016. A biased random-key genetic algorithm for the container pre-marshalling problem. Computers & Operations Research 75, 83 – 102. doi:10.1016/j.cor.2016.05.011.

Huang, S.H., Lin, T.H., 2012. Heuristic algorithms for container pre-marshalling problems. Computers & Industrial Engineering 62, 13 – 20. doi:10.1016/j.cie.2011.08.010.

Huang, X., Dai, X., Luo, Y., Wang, Y., 2019. Design of container terminal handling system based on index forecast and economic evaluation. Journal of Coastal Research 94, 377 – 384. doi:10.2112/SI94-077.1.

Jovanovic, R., Tanaka, S., Nishi, T., Voß, S., 2019a. A grasp approach for solving the blocks relocation problem with stowage plan. Flexible Services and Manufacturing Journal 31, 702 – 729. doi:10.1007/s10696-018-9320-3.

Jovanovic, R., Tuba, M., Voß, S., 2017. A multi-heuristic approach for solving the premarshalling problem. Central European Journal of Operations Research 25, 1 – 28. doi:10.1007/s10100-015-0410-y.

Jovanovic, R., Tuba, M., Voß, S., 2019b. An efficient ant colony optimization algorithm for the blocks relocation problem. European Journal of Operational Research 274, 78 – 90. doi:10.1016/j.ejor.2018.09.038.

Kang, J., Oh, M.S., Ahn, E.Y., Ryu, K.R., Kim, K.H., 2006. Planning for intra-block remarshalling in a container terminal, in: Ali, M., Dapoigny, R. (Eds.), Advances in Applied Artificial Intelligence. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1211 – 1220. doi:10.1007/11779568_128.

Kang, J., Oh, M.S., Ryu, K.R., Kim, K.H., 2005. Sequencing container moves for intra-block remarshalling in a container terminal yard. Journal of Navigation and Port Research 29, 83 – 90. doi:10.5394/KINPR.2005.29.1.083.

Kang, J.G., Kim, Y.D., 2002. Stowage planning in maritime container transportation. Journal of the Operational Research Society 53, 415 – 426. doi:10.1057/palgrave.jors.2601322.

Ku, D., Arthanari, T.S., 2016. Container relocation problem with time windows for container departure. European Journal of Operational Research 252, 1031 – 1039. doi:10.1016/j.ejor.2016.01.055.

Larsen, R., Pacino, D., 2020. A heuristic and a benchmark for the stowage planning problem. Maritime Economics and Logistics. forthcoming.

Lee, Y., Chao, S.L., 2009. A neighborhood search heuristic for pre-marshalling export containers. European Journal of Operational Research 196, 468 – 475. doi:10.1016/j.ejor.2008.03.011.

Lee, Y., Hsu, N.Y., 2007. An optimization model for the container pre-marshalling problem. Computers & Operations Research 34, 3295 – 3313. doi:10.1016/j.cor.2005.12.006.

Lee, Y., Lee, K., 2020. Lot-sizing and scheduling in flat-panel display manufacturing process. Omega 93, 102036. doi:10.1016/j.omega.2019.02.005.

Lee, Y., Lee, Y., 2010. A heuristic for retrieving containers from a yard. Computers & Operations Research 37, 1139 – 1147. doi:10.1016/j.cor.2009.10.005.

Lehnfeld, J., Knust, S., 2014. Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. European Journal of Operational Research 239, 297 – 312. doi:10.1016/j.ejor.2014.03.011.

Lin, D.Y., Lee, Y.J., Lee, Y., 2015. The container retrieval problem with respect to relocation. Transportation Research Part C 52, 132 – 143. doi:10.1016/j.trc.2015.01.024.

Matsaini, Santosa, B., 2018. Solving the container stowage problem (CSP) using particle swarm optimization (PSO). IOP Conference Series: Materials Science and Engineering 337, Article number 012002. doi:10.1088/1757-899x/337/1/012002.

Meisel, F., 2009. Seaside Operations Planning in Container Terminals, Contributions to Management Science. Physica-Verlag (A Springer Company), Berlin. doi:10.1007/978-3-7908-2191-8.

Pacino, D., Delgado, A., Jensen, R.M., Bebbington, T., 2011. Fast generation of near-optimal plans for eco-efficient stowage of large container vessels. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 6971 LNCS, 286 – 301. doi:10.1007/978-3-642-24264-9_22.

Paquay, C., Limbourg, S., Schyns, M., Oliveira, J., 2018. Mip-based constructive heuristics for the three-dimensional bin packing problem with transportation constraints. International Journal of Production Research 56, 1581–1592. doi:10.1080/00207543.2017.1355577.

Parreño, F., Pacino, D., Alvarez-Valdes, R., 2016. A GRASP algorithm for the container stowage slot planning problem. Transportation Research Part E: Logistics and Transportation Review 94, 141 – 157. doi:10.1016/j.tre.2016.07.011.

Parreño-Torres, C., Alvarez-Valdes, R., Parreño, F., 2019a. Solution strategies for a multiport container ship stowage problem. Mathematical Problems in Engineering , 9029267.doi:10.1155/2019/9029267.

Parreño-Torres, C., Alvarez-Valdes, R., Ruiz, R., 2019b. Integer programming models for the pre-marshalling problem. European Journal of Operational Research 274, 142 – 154. doi:10.1016/j.ejor.2018.09.048.

Parreño-Torres, C., Alvarez-Valdes, R., Ruiz, R., Tierney, K., 2020. Minimizing the crane time in premarshalling problems. Transportation Research Part E 137, 101917. doi:10.1016/j.tre.2020.101917.

Petering, M.E., 2011. Decision support for yard capacity, fleet composition, truck substitutability, and scalability issues at seaport container terminals. Transportation Research Part E: Logistics and Transportation Review 47, 85 – 103. doi:10.1016/j.tre.2010.07.007.

Prayogo, D., Hidayatno, A., et al., 2017. Development of integrated tactical level planning in container terminal, in: 2017 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM), IEEE. pp. 1013 – 1017. doi:10.1109/IEEM.2017.8290045.

Quispe, K.E.Y., Lintzmayer, C.N., Xavier, E.C., 2018. An exact algorithm for the blocks relocation problem with new lower bounds. Computers & Operations Research 99, 206 – 217. doi:10.1016/j.cor.2018.06.021.

Resende, M.G., Ribeiro, C.C., 2016. Optimization by GRASP. Springer. doi:10.1007/978-1-4939-6530-4.

Roberti, R., Pacino, D., 2018. A decomposition method for finding optimal container stowage plans. Transportation Science 52, 1444 – 1462. doi:10.1287/trsc.2017.0795.

Roy, D., de Koster, M., 2014. Modeling and design of container terminal operations. ERIM report series reference no. ERS-2014-008-LIS doi:10.2139/ssrn.2458048.

Scarre, C., 1995. Chronicle of the Roman Emperors: The Reign-by-Reign Record of the Rulers of Imperial Rome. Thames & Hudson.

Sciomachen, A., Tanfani, E., 2003. The master bay plan problem: a solution method based on its connection to the three-dimensional bin packing problem. IMA Journal of Management Mathematics 14, 251 – 269. doi:10.1093/imaman/14.3.251.

Sciomachen, A., Tanfani, E., 2007. A 3D-BPP approach for optimising stowage plans and terminal productivity. European Journal of Operational Research 183, 1433 – 1446. doi:10.1016/j.ejor.2005.11.067.

de Melo da Silva, M., Toulouse, S., Calvo, R.W., 2018. A new effective unified model for solving the pre-marshalling and block relocation problems. European Journal of Operational Research 276, 40 – 56. doi:10.1016/j.ejor.2018.05.004.

da Silva Firmino, A., de Abreu Silva, R., Times, V., 2019. A reactive GRASP metaheuristic for the container retrieval problem to reduce crane's working time. Journal of Heuristics 25, 141 – 173. doi:10.1007/s10732-018-9390-0.

Slaney, J., Thiébaux, S., 2001. Blocks world revisited. Artificial Intelligence 125, 119 – 153. doi:10.1016/S0004-3702(00)00079-5.

Sniedovich, M., Voß, S., 2006. The corridor method: a dynamic programming inspired metaheuristic. Control and Cybernetics 35, 551 – 578.

Steenken, D., Voß, S., Stahlbock, R., 2004. Container terminal operation and operations research-a classification and literature review. OR spectrum 26, 3 – 49. doi:10.1007/s00291-003-0157-z.

Stein, S.K. (Ed.), 2017. The Sea in world History. Exploration, travel and trade. ABC-CLIO.

Tanaka, S., Mizuno, F., 2018. An exact algorithm for the unrestricted block relocation problem. Computers & Operations Research 95, 12 – 31. doi:10.1016/j.cor.2018.02.019.

Tanaka, S., Tierney, K., 2018. Solving real-world sized container pre-marshalling problems with an iterative deepening branch-and-bound algorithm. European Journal of Operational Research 264, 165 – 180. doi:https://doi.org/10.1016/j.ejor.2017.05.046.

Tanaka, S., Tierney, K., Parreño-Torres, C., Alvarez-Valdes, R., Ruiz, R., 2019. A branch and bound approach for large pre-marshalling problems. European Journal of Operational Research 278, 211 – 225. doi:10.1016/j.ejor.2019.04.005.

Tanaka, S., Voß, S., 2019. An exact algorithm for the block relocation problem with a stowage plan. European Journal of Operational Research 279, 767 – 781. doi:10.1016/j.ejor.2019.06.014.

Tang, L., Liu, J., Yang, F., Li, F., Li, K., 2015. Modeling and solution for the ship stowage planning problem of coils in the steel industry. Naval Research Logistics 62, 564 – 581. doi:10.1002/nav.21664.

Tang, L., Zhao, R., Liu, J., 2012. Models and algorithms for shuffling problems in steel plants. Naval Research Logistics 59, 502 – 524. doi:10.1002/nav.21503.

Tierney, K., Pacino, D., Voß, S., 2017. Solving the pre-marshalling problem to optimality with A* and IDA*. Flexible Services and Manufacturing Journal 29, 223 – 259. doi:10.1007/s10696-016-9246-6.

Tierney, K., Voß, S., 2016. Solving the robust container pre-marshalling problem, in: International Conference on Computational Logistics, Springer. pp. 131 – 145. doi:10.1007/978-3-319-44896-1_9.

Toso, E., Morabito, R., Clark, A., 2009. Lot sizing and sequencing optimisation at an animal-feed plant. Computers and Industrial Engineering 57, 813 – 821. doi:10.1016/j.cie.2009.02.011.

UNCTAD, 2018. United Nations Conference on Trade and Development (UNCTAD) Review of Maritime Transport. United Nations. URL: https://unctad.org/en/PublicationsLibrary/rmt2018_en.pdf.

UNCTAD, 2019. United Nations Conference on Trade and Development (UNCTAD) Review of Maritime Transport. United Nations. URL: https://unctad.org/en/PublicationsLibrary/rmt2019_en.pdf.

Wang, N., Jin, B., Lim, A., 2015. Target-guided algorithms for the container pre-marshalling problem. Omega 53, 67 – 77. doi:10.1016/j.omega.2014.12.002.

Wang, N., Jin, B., Zhang, Z., Lim, A., 2017. A feasibility-based heuristic for the container pre-marshalling problem. European Journal of Operational Research 256, 90 – 101. doi:10.1016/j.ejor.2016.05.061.

Wei, W., Guimaraes, L., Amorim, P., Almada-Lobo, B., 2017. Tactical production and distribution planning with dependency issues on the production process. Omega 67, 99–114. doi:10.1016/j.omega.2016.04.004.

Wilmsmeier, G., Spengler, T., 2016. Energy consumption and container terminal efficiency. Bulletin FAL 6, 1–10. URL: https://repositorio.cepal.org/handle/11362/40928.

Wilson, I., Roach, P., 2000. Container stowage planning: a methodology for generating computerised solutions. Journal of the Operational Research Society 51, 1248 – 1255. doi:10.1057/palgrave.jors.2601022.

Wolsey, L., 1998. Integer programming. Wiley.

Yurtseven, M., Boulougouris, E., Turan, O., 2018. Container ship stowage plan using steepest ascent hill climbing, genetic, and simulated annealing algorithms. Marine Design XIII 1, 617–623.

Zhang, G., Nishi, T., Turner, S., Oga, K., Li, X., 2017. An integrated strategy for a production planning and warehouse layout problem: Modeling and solution approaches. Omega 68, 85–94. doi:10.1016/j.omega.2016.06.005.

Zhang, R., Jiang, Z.Z., Yun, W.Y., 2015. Stack pre-marshalling problem: A heuristic-guided branch-and-bound algorithm. International Journal of Industrial Engineering 22, 509 – 523.

Zhang, R., Liu, S., Kopfer, H., 2016. Tree search procedures for the blocks relocation problem with batch moves. Flexible Services and Manufacturing Journal 28, 397 – 424. doi:10.1007/s10696-015-9229-z.

Zhao, N., Liu, Y., Mi, W., Shen, Y., Xia, M., 2020. Operation Management in the Container Terminal. Springer Singapore, Singapore. pp. 47 – 73. doi:10.1007/978-981-15-2937-5_2.

Zhen, L., Jiang, X., Lee, L.H., Chew, E.P., 2013. A review on yard management in container terminals. Industrial Engineering and Management Systems 12, 289 – 304.

Zhu, H., Ji, M., Guo, W., Wang, Q., Yang, Y., 2019. Mathematical formulation and heuristic algorithm for the block relocation and loading problem. Naval Research Logistics 66, 333 – 351. doi:10.1002/nav.21843.