Novel realistic approaches to container premarshalling

PhD thesis

Celia Jiménez Piqueras

Cover picture by Jan van der Wolf (2016). Source: pexels.com

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Novel realistic approaches to container premarshalling

September 2024

Author: Celia Jiménez Piqueras

Supervisors: Dr. Consuelo Parreño Torres
Dr. Rubén Ruiz García

Resumen

En las últimas décadas, el rápido crecimiento del transporte de mercancía contenerizada ha planteado grandes retos a la eficiencia portuaria. La optimización de operaciones se ha vuelto crucial para minimizar los costes de puertos y navieras y reducir el impacto medioambiental.

La organización y gestión del patio del puerto son esenciales para garantizar la eficiencia del puerto, ya que se utiliza para almacenar contenedores y conecta todas las zonas y actividades portuarias. Este entorno dinámico exige estrategias efectivas para colocar y retirar contenedores. Al estar apilados, retirar un contenedor puede requerir mover otros que estén encima a otras pilas, lo que ralentiza la retirada. Estos movimientos se pueden evitar ordenando los contenedores de antemano en un proceso conocido como *premarshalling*.

El problema clásico de optimización asociado al premarshalling, que trata de encontrar el mínimo número de recolocaciones necesarias para ordenar los contenedores, se basa en varias asunciones poco realistas. Esta tesis pretende facilitar la aplicación práctica del premarshalling reformulando algunas de estas asunciones.

Los contenedores del patio se organizan en grupos llamados bahías, y el problema de premarshalling se centra en organizar una bahía de contenedores cada vez. La formulación clásica solo permite recolocaciones dentro de la bahía que se está ordenando, pero esta no es una limitación real en la práctica. Proponemos una nueva versión del problema en la que los contenedores se pueden mover a una bahía adyacente para facilitar el proceso de ordenación.

Otro aspecto poco realista de la formulación clásica es su objetivo. En literatura previa se muestra que minimizar el tiempo empleado por la grúa durante el premarshalling conduce a soluciones más eficientes que minimizar el número de recolocaciones. Esta tesis va más allá de la incorporación de tiempos de grúa y asume la disponibilidad limitada de la grúa. Mientras que la formulación original no proporciona una solución cuando el tiempo necesario para completar el premarshalling excede la disponibilidad de la grúa, definimos una nueva formulación que proporciona un premarshalling completo cuando el tiempo lo permite y parcial cuando no. Definir buenas ordenaciones parciales es un reto, así que exploramos tres estrategias alternativas.

El estudio de estas variantes del problema requería un método de resolución simple que pudiera adaptarse fácilmente a diversas asunciones y proporcionar soluciones óptimas para la validación del problema. Para cumplir con tales requerimientos, proponemos un método de resolución que combina el uso de un solver de programación por restricciones con algoritmos sencillos y fáciles de implementar. La programación por restricciones ha recibido muy poca atención en la literatura de premarshalling, a pesar de haber demostrado un buen rendimiento resolviendo problemas combinatorios en diversos campos. Esta tesis revela la eficacia de esta técnica para el problema de premarshalling al proporcionar modelos de programación por restricciones que superan a los métodos de programación matemática de vanguardia.

En líneas generales, esta tesis pretende acercar la formulación matemática del problema de premarshalling a su aplicación en el mundo real. Para ello, aborda tanto aspectos teóricos como técnicos. Desde el punto de vista teórico, introduce nuevas variantes del problema que suponen un notable avance hacia una formulación más realista. Desde el punto de vista técnico, presenta modelos de programación por restricciones que demuestran la eficacia de esta técnica sobre el problema de premarshalling.

Resum

En les últimes dècades, el ràpid creixement del transport de mercaderia conteneritzada ha plantejat grans reptes a l'eficiència portuària. L'optimització d'operacions s'ha tornat crucial per a minimitzar els costos de ports i navilieres i reduir l'impacte mediambiental.

L'organització i gestió del pati del port són essencials per a garantir l'eficiència del port, ja que s'utilitza per a emmagatzemar contenidors i connecta totes les zones i activitats portuàries. Aquest entorn dinàmic exigeix estratègies efectives per a col·locar i retirar contenidors. En estar apilats, retirar un contenidor pot requerir moure uns altres que estiguen damunt a altres piles, la qual cosa alenteix la retirada. Aquests moviments es poden evitar ordenant els contenidors per endavant, un procés conegut com *premarshalling*.

El problema clàssic d'optimització associat al premarshalling, que tracta de trobar el mínim nombre de recol·locacions necessàries per a ordenar els contenidors, es basa en diverses assumpcions poc realistes. Aquesta tesi pretén facilitar l'aplicació pràctica del premarshalling reformulant algunes d'aquestes assumpcions.

Els contenidors del pati s'organitzen en grups anomenats badies, i el problema de premarshalling se centra en organitzar una badia de contenidors cada vegada. La formulació clàssica només permet recol·locacions dins de la badia que s'està ordenant, però aquesta no és una limitació real en la pràctica. Proposem una nova versió del problema en la qual els contenidors es poden moure a una badia adjacent per a facilitar el procés d'ordenació.

Un altre aspecte poc realista de la formulació clàssica és el seu objectiu. En literatura prèvia es mostra que minimitzar el temps emprat per la grua durant el premarshalling condueix a solucions més eficients que minimitzar el nombre de recol·locacions. Aquesta tesi va més enllà de la incorporació de temps de grua i assumeix la disponibilitat limitada de la grua. Mentres que la formulació original no proporciona una solució quan el temps necessari per a completar el premarshalling excedeix la disponibilitat de la grua, definim una nova formulació que proporciona un premarshalling complet quan el temps ho permet i parcial quan no. Definir bones ordenacions parcials és un repte, així que explorem tres estratègies alternatives.

L'estudi d'aquestes variants del problema requeria un mètode de resolució simple que poguera adaptar-se fàcilment a diverses assumpcions i proporcionar solucions òptimes per a la validació del problema. Per a complir amb tals requeriments, proposem un mètode de resolució que combina l'ús d'un solver de programació per restriccions amb algorismes senzills i fàcils d'implementar. La programació per restriccions ha rebut molt poca atenció en la literatura de premarshalling, malgrat haver demostrat un bon rendiment resolent problemes combinatoris en diversos camps. Aquesta tesi revela l'eficàcia d'aquesta tècnica per al problema de premarshalling en proporcionar models de programació per restriccions que superen als mètodes de programació matemàtica d'avantguarda.

En línies generals, aquesta tesi pretén acostar la formulació matemàtica del problema de premarshalling a la seua aplicació en el món real. Per a això, aborda tant aspectes teòrics com tècnics. Des del punt de vista teòric, introdueix noves variants del problema que suposen un notable avanç cap a una formulació més realista. Des del punt de vista tècnic, presenta models de programació per restriccions que demostren l'eficàcia d'aquesta tècnica sobre el problema de premarshalling.

Abstract

In recent decades, the rapid growth of containerized freight has posed significant challenges to port efficiency. Optimizing operations has become crucial for minimizing costs for ports and shipping companies and reducing environmental impact.

The organization and management of the port yard are essential to ensure port efficiency, as it is used for container storage and connects all port areas and activities. This dynamic environment demands effective strategies for placing and retrieving containers. Since containers are stacked, retrieving one may require moving containers above it to other stacks, which slows down the retrieval. These movements can be avoided by arranging the containers beforehand through a process known as container premarshalling.

The classical optimization problem associated with premarshalling, which aims to find the minimum number of relocations necessary to arrange the containers, relies on several unrealistic assumptions. This thesis aims to facilitate the practical application of premarshalling by reformulating some of these assumptions.

Containers in the yard are organized into groups called bays, and the premarshalling problem focuses on arranging one bay of containers at a time. The classical formulation only allows relocations within the bay being arranged, but this is not a real limitation in practice. We propose a novel version of the problem where containers can be moved to an adjacent bay to facilitate the arrangement process.

Another unrealistic aspect of the classical formulation is its objective. Previous literature shows that minimizing the total crane time during premarshalling leads to more efficient solutions than minimizing the number of relocations. This thesis goes beyond incorporating crane times and acknowledges limited crane availability. While the original formulation fails to provide a solution when the required time for completing the arrangement exceeds crane availability, we define a novel formulation that yields a complete premarshalling when time allows and a partial arrangement when it does not. Defining good-quality partial arrangements is challenging, and we explore three alternative strategies.

Studying these problem variants required a simple solution method that could be easily adapted to varying assumptions and provide optimal solutions for problem validation. To meet these requirements, we propose a solution method that combines the use of a constraint programming solver with simple algorithms that are easy to implement. Constraint programming has received very little attention in the premarshalling literature despite showing strong performance in solving combinatorial problems in various fields. This thesis reveals the effectiveness of this technique for the premarshalling problem by providing constraint programming models that outperform the state-of-the-art mathematical programming approaches.

Overall, this thesis aims to bridge the gap between the mathematical formulation of the premarshalling problem and real-world application. To achieve this, it addresses both theoretical and technical aspects. On the theoretical side, it introduces several novel variants of the problem that represent significant progress toward a more realistic formulation. On the technical side, it presents constraint programming solution methods that demonstrate the efficiency of this technique in addressing the premarshalling problem.

Acknowledgements

This study has been partially supported by the Spanish Ministry of Science and Innovation under predoctoral grant PRE2019-087706, the project "OPTEP-Port Terminal Operations Optimization" (RTI2018-094940-B-I00), and PID2021-124975OB-I00, financed with FEDER funds.

This thesis represents the work over several years in which I have learned a lot, growing both professionally and personally. I feel extremely fortunate to have had the opportunity to do this PhD, and I cherish every teaching and beautiful experience it has gifted me with. I believe in the power of people over that of chance to improve each other's lives. Therefore, I would like to thank everyone who has made this thesis possible and has contributed to making this period of my life an amazing journey.

Thanks to my thesis supervisors, Rubén and Consuelo, and also to Ramón, who, despite not being officially my supervisor, has played this role on numerous occasions. I have learned a lot from all three of you. Thank you for your time, even when you were extremely busy. Thank you for sharing your expertise with me and for all the advice I have received. Without all that, this thesis would not have been possible.

Thanks to the supervisors of my research stays, Kevin and Dario. It was a pleasure for me to visit your universities. Thank you for receiving me and making me feel one more in your research groups. Special thanks to Dario for welcoming me anytime I had the opportunity to come back after my stay. Thank you both for your very helpful advice and guidance; half of this thesis is the result of collaborations with you.

Thanks to the entire SOA/Resolvética group, especially to Eva, for all your kindness and help as a tutor, and in particular to Gerardo and Pedro, for your help with the cluster and integrating into ITI at the beginning of my thesis. Also, I am very grateful to Fede for inviting me to collaborate on one of your papers and convincing me to teach a course in Sevilla. And finally, many thanks to Juan Camilo for such valuable support in the final stage of my PhD.

Thanks to my fellow doctoral researchers for all the good times we have spent together: the conversations at lunchtime (and while working), the support we gave each other when staying at the university until almost closing time, and, of course, the dances at the office. I will never forget these and many other wonderful moments we shared, which have made my everyday work more enjoyable.

Thanks to all the colleagues I have had the pleasure to meet, and to all the staff of the Department of Applied Statistics and Operational Research and Quality. Special thanks to Vicent for your kindness and patience in guiding my first steps in university teaching. At another department, but still at UPV, I know I will always have a mentor; thank you, Juan Carlos, for your help in finding this PhD opportunity and for your guidance throughout all these years.

Thanks also to the colleagues I have met and the friends I have made during my research stays. Thank you for the beautiful moments we have shared and for helping me adapt to these new places and to explore and enjoy them.

Thanks to my parents, Brigi and Paco, and to my sister, Isabel. It is impossible to thank you enough for all you have done for me, but I will take this opportunity to highlight a few things. Thank you for your support in all my adventures, thank you for celebrating all my achievements with me, and thank you for always being there for me, especially in the hardest times. Your love builds my strength.

Thanks to the rest of my family for always believing in me. In particular, thanks to my aunt Lucre for hosting me and for all the support I have received from you. Also, special thanks to Pablo for helping me discover the actual requirements of port terminals, which is extremely valuable for this doctoral thesis.

Thanks to my friends in Valencia, especially to Carmen, for accompanying me through the best and the worst of the everyday life of a PhD, and also to Sorina for all the conversations and hugs that have helped me continue and pursue my dreams.

Thanks to my friends from Albacete. Even if most of us are not living there anymore, anytime we meet, you make me feel like it will always be my home. Thank you very much for all your support.

Finally, thanks to all my friends who have supported me from a distance: phone calls, little messages... All that feels like virtual hugs to me and has made my days more beautiful. Also, thanks to Francisco for encouraging me to do my PhD and always believing in my work and capabilities.

My heartfelt thanks to all of you.

Agradecimientos

Este estudio ha sido parcialmente financiado por el Ministerio de Ciencia e Innovación a través de la ayuda predoctoral PRE2019-087706, bajo el proyecto "OPTEP-Port Terminal Operations Optimization" (RTI2018-094940-B-I00), y PID2021-124975OB-I00, financiados con fondos FEDER.

Esta tesis representa el trabajo de varios años, en los que he aprendido mucho, creciendo tanto profesional como personalmente. Me siento extremadamente afortunada de haber tenido la oportunidad de realizar este doctorado, y valoro cada enseñanza y cada bonita experiencia que me ha regalado. Creo en el poder de las personas sobre el del azar para mejorarnos la vida unas a otras. Por tanto, me gustaría agradecer a cada persona que ha hecho posible esta tesis y ha contribuido a hacer de este periodo de mi vida un viaje increíble.

Gracias a mis directores de tesis, Rubén y Consuelo, y también a Ramón, que sin ser oficialmente director has desempeñado ese rol en numerosas ocasiones. He aprendido mucho de los tres. Gracias por vuestro tiempo, incluso estando extramadamente ocupados. Gracias por compartir vuestros conocimientos y experiencia conmigo, y por todos los consejos que he recibido. Sin todo ello, esta tesis no hubiera sido posible.

Gracias a los supervisores de mis estancias de investigación, Kevin y Dario. Fue un placer para mí visitar vuestras universidades. Gracias por recibirme y hacerme sentir una más en vuestros grupos de investigación. Gracias en particular a Dario por recibirme en cualquier momento que he tenido la oportunidad de volver tras mi estancia. Gracias a ambos por vuestros tan últiles consejos y guía, la mitad de esta tesis es fruto de colaboraciones con vosotros.

Gracias a todo el grupo SOA/Resolvética, especialmente a Eva, por toda tu amabilidad y tu ayuda como tutora, y en particular a Gerardo y Pedro, por vuestra ayuda con el cluster y para integrarme en el ITI al principio de mi doctorado. También, estoy muy agradecida a Fede, por invitarme a colaborar en uno de tus artículos y convencerme para dar un curso en Sevilla. Y finalmente, muchas gracias a Juan Camilo por tan valioso apoyo en la etapa final de mi doctorado.

Gracias a mis compañeros de doctorado por todos los buenos momentos que hemos pasado juntos: las conversaciones a la hora de la comida (y mientras trabajábamos), el apoyo de unos a otros cuando nos quedábamos en la universidad hasta casi la hora de cierre, y por supuesto, los bailes en el despacho. Nunca olvidaré estos y muchos otros momentos maravillosos que compartimos, que han hecho mi trabajo del día a día más ameno.

Gracias a todos los compañeros que he tenido el placer de conocer y a todo el personal del Departamento de Estadísitca e Investigación Operativa Aplicadas y Calidad. En especial, gracias a Vicent, por tu amabilidad y paciencia guiando mis primeros pasos en la enseñanza universitaria. En otro departamento, pero igualmente en la UPV, sé que siempre tendré un mentor: gracias, Juan Carlos, por tu ayuda para encontrar esta oportunidad de doctorado y tu guía a lo largo de todos estos años.

Gracias también a los compañeros que he conocido y los amigos que he hecho durante mis estancias de investigación. Gracias por los bonitos momentos que hemos compartido y por ayudarme a adaptarme a estos lugares nuevos para mí, y a explorarlos y disfrutarlos.

Gracias a mis padres, Brigi y Paco, y a mi hermana, Isabel. Es imposible agradeceros lo suficiente por todo lo que habéis hecho por mí, pero voy a aprovechar esta oportunidad para destacar algunas cosas. Gracias por vuestro apoyo en todas mis aventuras, gracias por celebrar conmigo todos mis logros, y gracias por estar siempre ahí conmigo, especialmente en los momentos más duros. Vuestro amor construye mi fuerza.

Gracias al resto de mi familia por siempre creer en mí. En particular, gracias a mi tía Lucre por acogerme y por todo el apoyo que he recibido de tu parte. También, especialmente gracias a Pablo, por ayudarme a descubrir los requerimientos reales de la terminales portuarias, algo extremadamente valioso para esta tesis doctoral.

Gracias a mis amigas y amigos en Valencia, especialmente a Carmen, por acompañarme en lo mejor y lo peor de la vida diaria de un doctorado, y a

Sorina, por todas las conversaciones y abrazos que me han ayudado a continuar y perseguir mis sueños.

Gracias a mis amigas y amigos de Albacete. Aunque la mayoría ya no vivamos allí, cada vez que nos vemos, me hacéis sentir que ese siempre será mi hogar. Muchas gracias por todo vuestro apoyo.

Finalmente, gracias a todos los amigos que me habéis apoyado en la distancia: una llamada de teléfono, unos mensajes... Todos esos gestos son como abrazos virtuales para mí y han hecho más bonitos mis días. También, gracias a Francisco por animarme a hacer el doctorado y siempre creer en mi trabajo y mis capacidades.

A todos vosotros, gracias de corazón.

Contents

List of Figures

List of Tables

Chapter 1

Motivation and scope of the research

This thesis is motivated by the efficiency challenges faced by container port terminals. Over the past few decades, containerized maritime trade has seen impressive growth, creating significant pressure on ports. In this context, optimizing container terminal operations is essential for ensuring smooth functioning and reducing costs for ports, shippers, and the environment.

Numerous terminal operations can benefit from optimization strategies. This thesis focuses on the premarshalling of containers, a process that involves arranging containers in the port yard to expedite their subsequent retrieval. While the premarshalling problem has been a subject of study in the literature, the existing formulations are not sufficiently realistic for practical implementation. This thesis presents novel formulations to incorporate requirements of port terminals neglected in previous literature, thus narrowing the gap between theoretical models and practical applications.

This chapter elaborates on the motivation behind this research and provides the outline and objectives of the thesis. Additionally, it indicates the scientific contributions associated with it.

1.1 Motivation of the research

This section expresses the motivation behind this thesis. First, it outlines the evolution and contemporary context of containerized maritime trade. Then, it explains the role and significance of optimization within this framework.

1.1.1 Containerized maritime trade

Containerization is a major milestone in freight transport. The International Maritime Organization (IMO), a United Nations agency specialized in safety, security, and environmental performance in shipping, was established in 1948. This agency promoted the standardization of shipping containers, officially stated by the International Organization for Standardization (ISO) in 1968, followed by additional specifications in 1970.

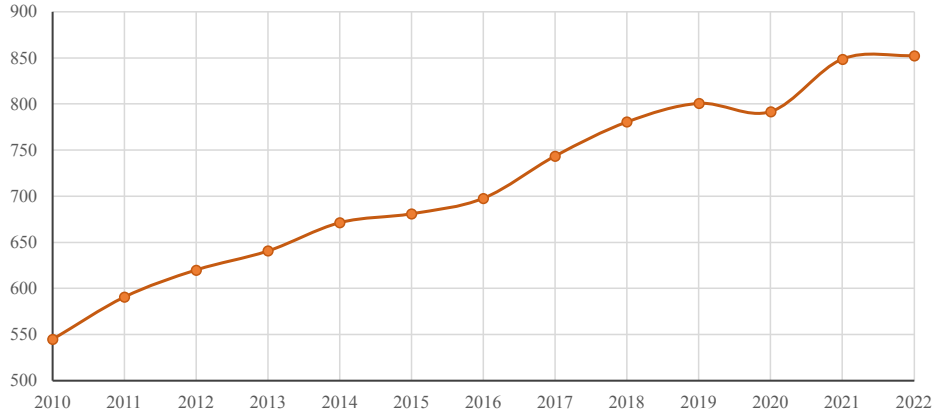
The standard container lengths are 8ft, 10ft, 20ft, 30ft, and 40ft, with the 20-foot and 40-foot containers being the most prevalent, particularly the latter. Apart from dry storage containers, specialized containers exist for cargo with specific requirements. These include refrigerated containers, insulated or thermal containers, and tanks for liquid or gaseous materials.

Containers streamline freight transfer between various modes of transport, including ships, trains, or trucks, without requiring cargo handling until reaching the final destination. This minimizes the exposure of goods, thereby reducing the risks of damage and the need for extensive packaging. Additionally, containers facilitate legality verification and tracking of cargo. Their design enables secure stacking, optimizing space utilization and simplifying transfers between transport modes. In essence, container standardization drastically reduces shipping times and costs.

Maritime transport is an essential pillar of the global economy. More than four-fifths of international trade is carried by sea. Containerization is a crucial driver of this ratio, implying an enormous increase in the maritime transportation of goods. The volume of containerized cargo transported is quantified in TEUs, with each TEU (Twenty-foot Equivalent Unit) representing the capacity of a standard 20-foot container. This unit enables the analysis of container transportation trends in the last decades. This task is undertaken annually by UNCTAD (UN Trade and Development), a permanent intergovernmental organization within the United Nations that supports developing countries in world trade.

The annual Review of Maritime Transport by UNCTAD offers valuable insights into maritime shipping, particularly containerized transport. According to UNCTAD, 1992, world container port throughput stood at just under 86 million TEUs in 1990. However, UNCTAD, 2023 reported that in 2022, throughput had soared to approximately 852 million TEUs. In 2020, the onset of the COVID-19 pandemic led to a 1.2% drop in world container port throughput, as noted in UNCTAD, 2021, but it rebounded in 2021, registering a 7% increase over 2020, as indicated in UNCTAD, 2022. These trend changes are captured by Figure 1.1, where additionally, we can observe the increment in port throughput since 2010.

Figure 1.1: Evolution in world container port throughput.



The world container port throughput is shown in million TEUs for each year from 2010 to 2022. Source: Elaborated by the author (2024) using data from the UNCTAD statistical portal.

Containerships have experienced rapid growth in size since the early 1970s, when their capacity ranged from 600 to 900 TEUs. By the 1980s, their capacity had already reached 4300 TEUs, and in 2018, the maximum capacity was above 21000 TEUs (UNCTAD, 2018). In 2023, the largest ships surpassed 24000 TEUs, and this growing trend is expected to continue in the coming years. The increase in size and better utilization of vessel capacity have led to a substantial rise in port throughput, as highlighted in the previous paragraph. Moreover, using larger vessels means handling greater amounts of containers per vessel, which increases the pressure on ports when serving a containership.

Consequently, maintaining port efficiency has become increasingly challenging due to the larger volumes of containers and sharper peaks in demand when a containership arrives at the port.

1.1.2 The role of optimization at container port terminals

The previous section illustrates the impressive scale of maritime trade. Ports handle enormous quantities of containers daily, and the rapid and sharp increment in the volume of containers transported since its standardization has placed notable pressure on the operation of port terminals.

The increase in the number of containers handled at terminals necessitates the coordination of thousands of operations, hampering port efficiency. Moreover, the vast increment in vessel size has drastically altered the container arrival and departure patterns at ports. The docking of a containership creates a substantial demand peak for cranes, operators, and other port resources. This surge may exceed the terminal's capacity, leading to prolonged waiting times.

Delays in serving vessels have serious consequences for ports, shippers, and the environment. Berthing space at ports is limited, and not operating a containership in time prevents the berthing of other vessels upon its arrival, producing notable congestion. These delays also affect the scheduled routes of vessels, increasing costs for shipping companies. Furthermore, containerships must use their engines during container loading and unloading to maintain the ship's balance. Consequently, prolonged loading/unloading times exacerbate environmental impacts. Additionally, the role of ports as exchange hubs of intermodal transport chains implies that maritime or land transport delays can escalate, affecting subsequent stages in the chain.

Port expansions are highly controversial. Despite the potential for increased operational space at the terminals, they are extremely costly and, most importantly, have devastating environmental effects on the surrounding coast. Such expansions entail an increase in emissions and the release of dangerous residues, polluting the air, water, and coastal areas. Furthermore, structures like breakwaters block the natural movement of sediments and sand carried by water flows (Esteban-Chapapría and Serra-Peris, 2021), causing severe damage to shores and ecosystems.

Optimizing port terminals maximizes the utilization of existing resources, negating the necessity for costly port expansions. The application of optimization strategies notably increases the efficiency of numerous activities, such as determining berthing positions and periods for vessels or allocating cranes for

operations in the quay and the port yard. Furthermore, various processes aimed at streamlining specific port operations can also be optimized to maximize their effectiveness. Container premarshalling is one of these processes.

Once containers arrive in the yard, they are allocated a storage position and remain stacked until they leave the port. Although assignment strategies exist to streamline the subsequent retrieval of containers from the yard, the large number of containers entering and leaving the port simultaneously, along with the dynamism of the yard, hinders the efficient placement of containers.

During inactivity periods between container retrieval or storing operations, cranes can relocate containers within the yard. This relocation of containers helps to expedite their retrieval. This thesis studies the premarshalling problem, which seeks to optimize these container relocations to perform the arrangement process in the most streamlined way and achieve the most efficient configuration of containers for retrieval from the yard.

1.2 Scope

The optimization of container terminal operations is crucial for maximizing the efficient utilization of port resources. Optimization strategies can benefit a wide range of processes and activities, notably reducing costs for ports, shippers, and the environment.

The port yard is a temporary storage area for containers and a connection between stages in the container transport chain. The arrangement of containers within this space directly impacts the loading and unloading of vessels and other means of transport. Hence, it is essential to devise strategies that streamline the organization of this area.

Idle periods of yard cranes between container retrieval or storage operations offer the opportunity to arrange containers in the port yard. However, determining an effective arrangement is highly challenging. Therefore, several optimization problems have been formulated to address this issue, with the Container Premarshalling Problem (CPMP) being one of the most extensively studied.

The premarshalling problem focuses on arranging a set of containers placed in adjacent stacks called a bay. It seeks a sequence of container relocations within these stacks to obtain an arrangement that avoids extra relocations during the

subsequent retrieval of these containers from the yard. The goal is to achieve this arrangement with the minimum number of relocations possible.

The scientific literature on the premarshalling problem offers numerous optimization methods. However, these methods often rely on unrealistic assumptions and are difficult to adapt to the actual requirements of port terminals.

The goal of this thesis is to narrow the gap between the theoretical formulation of the premarshalling problem and its practical implementation. To achieve this, we focus on the objectives described in the following section.

1.3 Objectives

This thesis is motivated by two main objectives:

- ▷ Studying the classical assumptions of the premarshalling problem and proposing new approaches that reformulate unrealistic assumptions according to the requirements of port terminals.
- ▷ Developing solution methods that are easy to implement and reproduce, as well as flexible, so that they can be adapted to different conditions and needs in practice.

1.3.1 More realistic formulations

On the one hand, the existing literature on container premarshalling assumes that the relocations for arranging a bay (set of adjacent container stacks) must be performed exclusively within these stacks. However, in practice, containers may be relocated to stacks at other bays if it is convenient for the arrangement process. To address this, we define the Premarshalling Problem with an Auxiliary Bay (CPMP-AB), where an additional bay is used as a storage buffer for containers from the bay being arranged.

On the other hand, while premarshalling is only performed during idle periods of the crane, the original formulation assumes unlimited crane availability. Consequently, if a premarshalling solution exceeds the available crane time, it cannot be implemented or must be partially executed. Unfortunately, partially applying a solution yielded by the original formulation often results in poor-quality arrangements or even worsens the initial configuration. In this thesis, we define the Premarshalling Problem Under Limited Crane Time (CPMP-LCT) to design efficient partial arrangements within the available time.

Defining partial premarshalling solutions that can be performed within the available crane time is a challenging task, and several objective functions for the CPMP-LCT are studied in this thesis. One of these objectives involves solving an additional optimization problem from the literature, the Block Relocation Problem (BRP). The BRP aims to minimize the relocations necessary for retrieving a group of containers from the yard. Thus, while premarshalling arranges the containers before their retrieval, the BRP focuses on optimization during retrieval.

The original premarshalling formulation (CPMP) does not account for the time taken by container relocations, treating each relocation as having the same cost. However, the time required for relocations varies significantly depending on the distance between the origin and destination stacks and other factors. While the original premarshalling formulation seeks a sequence with the minimum number of container relocations, a more efficient sequence would minimize the total time required for these relocations rather than the number of them. For this reason, the Container Premarshalling Problem with Crane Time Minimization Objective (CPMPCT) was previously proposed in the literature. This thesis studies the CPMPCT as a bridge between the original premarshalling formulation and the premarshalling problem under limited crane time (CPMP-LCT), which necessitates considering the time taken for relocations.

1.3.2 A versatile solution method

The optimization technique used in this thesis is Constraint Programming. We propose solution methods involving this technique for the novel premarshalling problems introduced, namely the CPMP-AB and the CPMP-LCT, including all their variants. Moreover, we present constraint programming models for the original CPMP and the additional problems studied: the CPMPCT and the BRP.

The constraint programming formulations proposed in this thesis for the original CPMP and its version minimizing crane time, the CPMPCT, demonstrate the effectiveness of this method for the premarshalling problem. These approaches are tested through computational experiments that show they enhance the performance of traditional mathematical programming models.

The proposed solution methods involve iterative algorithms tailored to each problem and the use of a constraint programming solver. The solver highly facilitates the implementation of the solution methods, which is a significant

advantage for reproducing them. Furthermore, using a solver has been particularly helpful for the development of this thesis by enabling us to test a wide range of premarshaling formulations and validate the novel approaches studied. The constraint programming model we present for the original CPMP establishes a base for designing models that solve the alternative premarshalling variants addressed.

Despite the advantages of using a solver and the flexibility of constraint programming in translating the requirements of each problem variant into constraints, designing effective solution methods is not straightforward. For this reason, this thesis includes a detailed explanation of the model-building procedure employed and how the formulations have been adapted to each premarshalling version.

1.4 Outline

The present Chapter 1 describes the objectives of this thesis and its structure. Chapter 2 provides a literature review and introduces the optimization at container port terminals and the constraint programming technique. The classical assumptions of premarshalling in the literature are analyzed in Chapter 3. Then, Chapters 4 and 5 propose novel versions of the premarshalling problem by modifying unrealistic assumptions of the original formulation. In Chapter 5, a premarshalling version from the literature that modifies one of the original assumptions is also studied as a bridge between the classical formulation and the new one proposed. Chapter 6 elaborates on the novel problem introduced in the preceding chapter and presents alternative objectives for it. Additionally, another optimization problem related to premarshalling is studied along with the definition of one of the objectives. Finally, Chapter 7 highlights the conclusions of this thesis and proposes related research lines.

The seven chapters constituting this document are outlined below:

Chapter 1: Motivation and scope of the research

This chapter contextualizes the research developed in this thesis and presents its structure and objectives. Firstly, the origins and evolution of containerized maritime trade are outlined, along with relevant figures illustrating its growth. Secondly, the role of optimization in container port terminals is explained, showing the motivation for the present research. Finally, the objectives of the thesis are specified. The structure and contents of the document are specified

in this section, and the following section indicates the scientific contributions associated with this thesis.

Chapter 2: Introduction

The first part of this chapter offers an introduction to port logistics and optimization problems in this context. Then, it provides a literature review for the premarshalling problem and other related problems, indicating the optimization methods studied prior to this thesis. After that, constraint programming, the main technique used in this study, is introduced.

Chapter 3: A constraint programming approach for the premarshalling problem

This chapter is devoted to the original version of the problem, the Container Premarshalling Problem (CPMP). We describe the problem and detail its assumptions, indicating why several of them are not realistic and in which of the subsequent chapters this issue is addressed. We present a constraint programming approach for the CPMP that serves as a foundation for developing the models for the alternative versions of premarshalling proposed in this thesis. This model is introduced along with the formulations tested in the process of designing it. This way, we illustrate the constraint programming modeling strategy used, which follows a different logic from that of generic model construction procedures for mathematical programming. The formulations are tested on well-known datasets from premarshalling literature and compared to the state-of-the-art integer programming model. The content of this chapter corresponds to the published paper Jiménez-Piqueras et al., 2023.

Chapter 4: Solving the premarshalling problem with an auxiliary bay

In this chapter, we introduce the Container Premarshalling Problem with an Auxiliary Bay (CPMP-AB). This formulation relaxes the restriction of only relocating containers within the stacks being premarshalled, as moving containers to additional stacks may facilitate the arrangement process. We describe two alternative constraint programming formulations with different allowance levels for relocating containers to auxiliary stacks. We present a computational study and discuss the performance of the proposed models, the differences between these approaches, and a comparison of the results with original premarshalling solutions. This chapter is based on the paper in preparation indicated in Section 1.5.2.

Chapter 5: Premarshalling problems considering crane times

This chapter addresses two versions of premarshalling considering crane times. First, the Container Premarshalling Problem with Crane Time Minimization Objective (CPMPCT), previously studied in the literature, is explored. This problem modifies the assumption of uniform relocation cost from the original formulation, considering the crane time instead, which is more realistic. We propose a constraint programming model and compare it through computational experiments with the state-of-the-art integer programming model for this problem. We take a further step towards a more realistic formulation, modifying the original assumption of unlimited crane availability. We define the Container Premarshalling Problem Under Limited Crane Time (CPMP-LCT), which seeks efficient partial premarshalling solutions within a given crane time limit, not yielded by previous premarshalling formulations. We introduce a constraint programming model for the novel problem and an alternative formulation tailored for a specific case. Additionally, we show the possibility of using the exact constraint programming model as a heuristic. This study corresponds to the published paper Jiménez-Piqueras et al., 2024.

Chapter 6: Alternative approaches for the premarshalling problem under limited crane time

This chapter focuses on the novel CPMP-LCT presented in the preceding chapter, considering alternative objectives. First, we present an enhanced solution method for the CPMP-LCT with its original objective. Then, we introduce constraint programming approaches for the two alternative objectives studied and a formulation for the Block Relocation Problem (BRP), which is involved in one of these objectives. The three alternative objectives are tested through computational experiments, and their suitability for different scenarios is discussed, proposing one of them as the best general approach for the CPMP-LCT. The content of this chapter belongs to the paper in preparation indicated in Section 1.5.2.

Chapter 7: Conclusions and future work

This chapter summarizes the conclusions obtained by the research developed in this thesis. It outlines the contributions of each chapter and provides a general perspective of the outcomes of this study. Additionally, potential research lines are suggested for future work.

1.5 Scientific contributions associated with this thesis

This section provides the list of contributions corresponding to this thesis.

1.5.1 Published papers in international journals

A constraint programming approach for the premarshalling problem

Jiménez-Piqueras, C., Ruiz, R., Parreño-Torres, C., Alvarez-Valdes, R.
European Journal of Operational Research, 306 (2), 668-678. (2023)
https://doi.org/10.1016/j.ejor.2022.07.042
Cited by 5 on Scopus and 8 on Google Scholar.

The contents of this paper are presented in Chapter 3.

The container premarshalling problem under limited crane time: A constraint programming approach

Jiménez-Piqueras, C., Parreño-Torres, C., Alvarez-Valdes, R., Ruiz, R.
Computers & Operations Research, 166, 106635. (2024)
https://doi.org/10.1016/j.cor.2024.106635

The contents of this paper are presented in Chapter 5.

1.5.2 Papers in preparation

At the moment of publication of this thesis, two chapters of this document are being prepared for publication as scientific papers. The contents of these chapters are the result of two research stays: the first in 2021, at Universität Bielefeld, Germany, under the supervision of Dr. Kevin Tierney, and the second in 2023, at Danmarks Tekniske Universitet, Denmark, under the supervision of Dr. Dario Pacino.

Solving the container premarshalling problem with an auxiliary bay

Jiménez-Piqueras, C., Tierney, K.

The contents of this paper in preparation are presented in Chapter 4.

Alternative objectives and constraint programming models for premarshalling under limited crane time

Jiménez-Piqueras, C., Pacino, D.

The contents of this paper in preparation are presented in Chapter 6.

1.5.3 Oral presentations at international conferences

A constraint programming approach for the premarshalling problem

Jiménez-Piqueras, C., Ruiz, R., Parreño-Torres, C., Alvarez-Valdes, R.
31st European Conference on Operational Research (EURO)
University of West Attica, Athens, Greece. (2021)

A constraint programming approach for the premarshalling problem with an auxiliary bay

Jiménez-Piqueras, C., Tierney, K., Ruiz, R.
32nd European Conference on Operational Research (EURO)
Aalto University, Espoo, Finland. (2022)

Exact methods for the premarshalling problem under limited crane time

Jiménez-Piqueras, C., Pacino, D., Parreño-Torres, C., Alvarez-Valdes, R.
10th Optimization Conference
University of Aveiro, Aveiro, Portugal. (2023)

Partial container premarshalling approaches using constraint programming

Jiménez-Piqueras, C., Pacino, D.
33nd European Conference on Operational Research (EURO)
Danmarks Tekniske Universitet, Copenhagen, Denmark. (2024)

1.5.4 Oral presentations at Spanish national conferences

Solving the premarshalling problem under limited crane time in the constraint programming paradigm

Jiménez-Piqueras, C., Ruiz, R., Parreño-Torres, C., Alvarez-Valdes, R.
3rd Spanish Young Statisticians and Operational Researchers Meeting (SYSORM)
Universidad Miguel Hernández, Elche, Spain. (2022)

Solving the container premarshalling problem considering crane availability constraints

Jiménez-Piqueras, C., Pacino, D., Parreño-Torres, C., Alvarez-Valdes, R.
XL National Congress of Statistics and Operations Research (SEIO)
Universidad Miguel Hernández, Elche, Spain. (2023)

1.5.5 Oral presentations at workshops and seminars

A constraint programming approach for the premarshalling problem

Jiménez-Piqueras, C., Ruiz, R., Parreño-Torres, C., Alvarez-Valdes, R.
Workshop on Container Stacking
Universität Bielefeld, Bielefeld, Germany. (2021)

Solving the premarshalling problem under limited crane time

Jiménez-Piqueras, C., Parreño-Torres, C., Alvarez-Valdes, R., Ruiz, R.
Seminar on Recent Research Results
DTU Management, Danmarks Tekniske Universitet, Denmark. (2023)

Considering crane availability constraints in the premarshalling problem

Jiménez-Piqueras, C., Pacino, D., Parreño-Torres, C., Alvarez-Valdes, R.
3rd EUROYoung Workshop
ESSEC Business School, Cergy, France. (2023)

Chapter 2

Introduction

This chapter provides an overview of the optimization problems that may arise in port terminals, involving berth allocation, crane assignments to operations, scheduling of tasks, or container stacking. It indicates the different parts of a terminal and describes the associated optimization problems.

Then, a literature review of the optimization methods developed for the Container Premarshalling Problem (CPMP) and variants of this problem is presented. Moreover, it describes the connection of the problems studied in this thesis with the existing literature. Additionally, recent approaches for other problems related to the CPMP, such as the Block Relocation Problem (BRP), are also indicated.

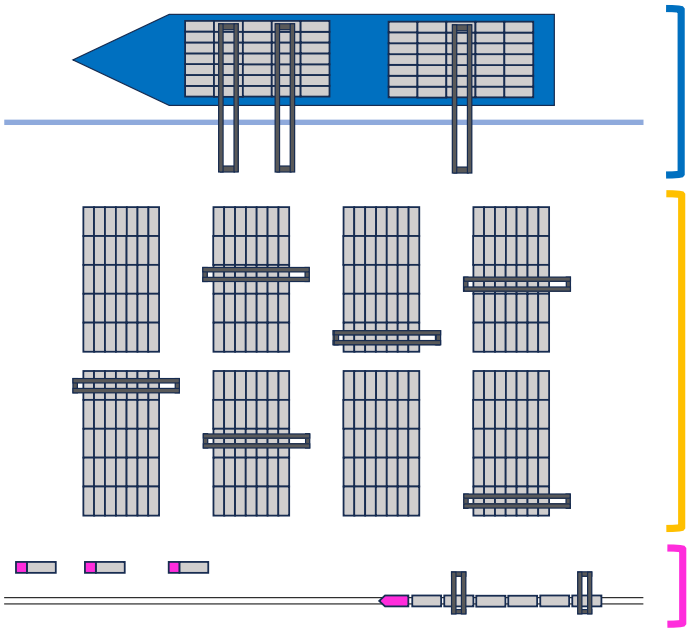
Finally, an introduction to Constraint Programming (CP) is provided. The basic concepts of this technique and the kinds of problems it is applied to are described. Also, the main modeling and search procedures are outlined.

2.1 Optimization of port terminal operations

The efficiency of a port terminal strongly depends on the optimization of operations. The activities involving container handling are classified into three main groups depending on the area of the terminal where they take place: seaside operations, yard operations, and landside operations. The three areas are depicted in Figure 2.1.

The seaside area corresponds to the quay, where containerships are berthed. In the yard area, containers are organized into stacks within groups of bays called blocks, and this storage space exchanges containers with both the seaside and the landside. The landside area is designated for trucks and trains.

Figure 2.1: An example of a port terminal configuration.



The seaside area is highlighted in blue, the yard area in yellow, and the landside area in magenta. Light gray rectangles represent containers, and cranes are indicated in dark gray. Source: Elaborated by the author (2024).

2.1.1 Seaside operations

Seaside operations take place in the quay area and are related to the loading and unloading of vessels.

When a vessel arrives at the port, it must be assigned a specific berth during a determined period of time. The Berth Allocation Problem (BAP) aims to plan this assignment for multiple vessels with the goal of minimizing the handling time for operating the vessels along with the waiting time until berthing.

The number of quay cranes that can be used to operate a berthed vessel must be determined. To this end, the Quay Crane Assignment Problem (QCAP) is defined as finding the number of cranes for each vessel so that they can leave their berthing positions as early as possible.

Once a vessel is assigned a set of cranes, the loading and unloading operations must be sequenced and distributed to the cranes. The Quay Crane Scheduling Problem (QCSP) seeks to schedule these operations to minimize the total completion time.

Before loading containers onto a vessel, they must be assigned a position aboard. The Container Stowage Problem (CSP) aims to arrange the containers in the best way subject to multiple rules involving features of containers such as weight or destination.

2.1.2 Yard operations

The yard area serves as a temporary storage area for import containers that must be transferred from vessels to land transports, and export containers that must be loaded onto a ship. It acts as the connection between seaside and landside areas.

Containers go through three phases in the port yard: storage, relocation, and retrieval. Optimizing both the storage and relocation phases aims to maximize the efficiency in subsequent container retrieval.

Upon arrival, a container must be assigned a position in the yard. The Storage Space Allocation Problem (SSAP) is formulated to find the best distribution of the containers among the blocks. Additionally, the Container Stacking Problem (CSP) determines an exact location for a container within a block, with the aim of minimizing future relocations required for retrieving the container from the yard.

The position initially assigned to a container may not be permanent. Containers may be relocated within the yard for various reasons, such as adjusting to changes in the retrieval schedules, accommodating new containers, or facilitating the future retrieval of containers currently blocked by other containers stacked above them. Containers can be relocated within a block to a set of stacks assigned to a specific vessel to avoid relocations during retrieval, which is the goal of the Container Remarshalling Problem (CRMP). Containers can also be relocated without the target of moving them to specific stacks but only to avoid blockages between them during retrieval. These relocations are performed within a specific set of stacks (a bay) and correspond to the Container Premarshalling Problem (CPMP) studied in this thesis.

The large amount of containers in the yard and the bustling environment of this area impede a perfect positioning of containers for retrieval, necessitating relocations during this process. While the containers targeted for relocation in the CRMP and the CPMP cannot be retrieved during the relocation process, the Block Relocation Problem (BRP), also known as Container Relocation Problem (CRP), aims to relocate the containers from a set of stacks in the most efficient way while retrieving them.

The optimization problems mentioned above are defined from the perspective of containers. However, it is also necessary to assign the operations to specific cranes. The Yard Crane Deployment Problem (YCDP) involves deciding the optimal number of cranes for each block and planning their movements between blocks. Once operations are assigned to cranes, the Yard Crane Scheduling Problem (YCSP) seeks the most efficient routing of cranes for executing a given sequence of container retrieval or storage operations.

Apart from crane optimization, it is also advantageous to efficiently plan the routing of yard trucks that assist in loading and unloading operations. This is the goal of the Yard Truck Scheduling Problem (YTSP).

2.1.3 Landside operations

Gate operations taking place on the landside can also benefit from optimization, and planning these operations is essential for overall port efficiency. Several tasks in this regard are determining the crane service to trains and trucks, scheduling loading and unloading operations, and organizing the arrivals of land vehicles through appointment systems and queue planning. Efficiency in these operations is crucial for reducing congestion at port gates, along with waiting times and polluting emissions.

2.2 Literature review

Extensive literature exists on the premarshalling problem. Here, we review the methods applied to the original formulation, alternative versions, and related problems.

There are several recent surveys on optimization problems at port terminals, including the CPMP and other related problems, such as the BRP. Covic, 2018, focuses on problems related to yard management. Kizilay and Eliiyi, 2021, review yard operations and quay crane scheduling, also exploring the integrations of two or three problems. Lersteau and Shen, 2022, cover literature specific to the BRP and the CPMP. Weerasinghe et al., 2024, examine container terminal operations in general, analyzing the operations research techniques developed.

2.2.1 Solving the Container Premarshalling Problem (CPMP)

This section reviews the literature specific to the Container Premarshalling Problem (CPMP), which includes numerous exact and heuristic approaches.

Constraint programming models

Rendl and Prandtstetter, 2013, provide some preliminary results using constraint programming for the CPMP and an alternative robust variant. They propose an iterative procedure to solve the problem, employing the lower bound for the number of premarshalling relocations and the heuristic presented by Bortfeldt and Forster, 2012. In Chapter 3, we present a constraint programming model for the CPMP, which is loosely based on the formulation described by Rendl and Prandtstetter, 2013.

Mathematical programming models

To the best of our knowledge, the first mathematical programming model designed for the premarshalling problem was introduced by Lee and Hsu, 2007. They developed an integer multicommodity flow model where the nodes and arcs of the embedded network represent the slots for the containers and the possible relocations between them, respectively, indexed by time points. Additionally, they presented a heuristic based on this integer model.

A unified integer programming model for both the premarshalling and block relocation problems was proposed by de Melo da Silva et al., 2018. This formulation includes three groups of decision variables; one of them describes the container layout, while the other two describe the relocations. Time is discretized, allowing only one relocation at each time step. An upper bound for the number of relocations is necessary and is determined by a heuristic.

Parreño-Torres et al., 2019, explored eight alternative integer programming formulations for the premarshalling problem, varying the groups of decision variables and their indexes. Time is divided into segments, with at most one relocation performed per segment, and an iterative solution procedure is developed to avoid the difficulty in calculating a tight upper bound for the number of relocations.

Other exact approaches

Expósito-Izquierdo et al., 2012, implemented an A* algorithm for the premarshalling problem. The optimal solutions obtained were used to evaluate the performance of the Lowest Priority First Heuristic introduced in the article. Tierney et al., 2016, proposed an iterative deepening A*, employing the lower bound for the number of relocations presented by Bortfeldt and Forster, 2012, along with multiple symmetry breaking and branching rules.

A branch and price technique was proposed by van Brink and van der Zwaan, 2014. Several branch and bound algorithms have also been developed for the premarshalling problem. Zhang et al., 2015, built a heuristic-guided branch and bound algorithm that uses a lower bound to determine which branches are explored first. Similarly, Tanaka and Tierney, 2018, proposed an iterative deepening branch and bound approach, incorporating new branching and dominance rules and a tighter lower bound that improves upon the one presented by Bortfeldt and Forster, 2012. The strategy of Tanaka and Tierney, 2018, which encountered an over-pruning issue later corrected by Jin and Yu, 2021, was further enhanced by Tanaka et al., 2019, who provided a new refinement of the lower bound.

Heuristic methods

Lee and Chao, 2009, observed that the number of mis-overlaid containers (those blocking the retrieval of others that must leave the yard earlier) serves as a lower bound for the number of premarshalling relocations required, and they presented a neighborhood search process. This approach was extended by the labeling algorithm developed by Huang and Lin, 2012, who also studied an alternative version of premarshalling where containers must be relocated to specific positions.

Caserta and Voß, 2009, described an algorithm for the CPMP based on the corridor method and generated a dataset of premarshalling instances that has been widely used in subsequent literature. Alternative datasets were built by Bortfeldt and Forster, 2012, who proposed an algorithm to calculate a lower bound for the CPMP and a tree search procedure for solving it, and by Expósito-Izquierdo et al., 2012, who presented the greedy Lowest Priority First Heuristic (LPFH). The LPFH was extended by the multi-heuristic method proposed by Jovanovic et al., 2017.

Among heuristic methods, we also find the target-guided approaches presented by Wang et al., 2015, and Wang et al., 2017. Additionally, Gheith et al., 2016, proposed a genetic approach using variable chromosome lengths, and Hottung and Tierney, 2016 developed a biased random-key genetic algorithm.

More recently, Hottung et al., 2020, introduced a deep learning tree search heuristic that uses deep neural networks for bounding and branching. The last heuristic methods include the iterated local search presented by Farrelly and Grimes, 2022, and the fill-and-reduce greedy algorithm developed by Araya and Toledo, 2023.

2.2.2 Related problems

In the literature, we find several variants of the premarshalling problem and related problems, such as the Block Relocation Problem (BRP), which is also explored in this thesis.

Variants of the premarshalling problem

The previously mentioned approaches use the classical objective for the premarshalling problem: minimizing the number of container relocations. Parreño-Torres et al., 2020, introduced the Container Premarshalling Problem with Crane Time Minimization Objective (CPMPCT). They showed that the crane time can significantly differ for solutions with the same number of relocations, thus highlighting the new objective's ability to provide more efficient solutions. In their paper, the authors presented an integer programming model and a branch and bound algorithm to solve the CPMPCT. In Chapter 5, we address this problem using constraint programming techniques. In addition to exact methods, we find the beam search algorithm developed for the CPMPCT by Parreño-Torres et al., 2022, who also proposed new dominance rules for tree search algorithms.

Wang et al., 2015, apart from proposing a heuristic method for the CPMP, defined the Container Premarshalling Problem with a Dummy Stack (CPMPDS). The CPMPDS considers the possibility of relocating containers to a space reserved for trucks during premarshalling, provided it is emptied by the end of the process. The Container Premarshalling Problem with an Auxiliary Bay (CPMP-AB), introduced in this thesis, shares the idea of using extra space for relocations. However, there are marked differences. In the CPMP-AB, containers are relocated to actual stacks instead of the truck area. Hence, this version offers more flexibility since more space is available, and there is no need to move all the containers back to the original set of stacks. More details are provided in Chapter 4.

In the literature, we also find robust premarshalling formulations that address the uncertainty in retrieval times. Rendl and Prandtstetter, 2013, presented an approach where each container has a possible range of retrieval priorities. Tierney and Voß, 2016, used a "blocking matrix" to indicate when there is a conflict for stacking a container on top of another. Boge et al., 2020, proposed a method based on the number of swaps between containers that may occur in the retrieval sequence, providing a non-strictly robust solution when a strictly robust one does not exist.

The Block Relocation Problem (BRP)

The idea of minimizing crane time instead of relocations appears in some BRP approaches, although the time computations are less realistic than those calculated by Parreño-Torres et al., 2020, for the CPMP, as they disregard crane acceleration and twistlock times. Lee and Lee, 2010, proposed a heuristic method with a time reduction phase, where a weighted sum of crane times is minimized using a mixed-integer program. Another heuristic was presented by Lin et al., 2015, to minimize both the number of relocations and the working time, providing insights into the trade-off between these objectives. An ant colony optimization algorithm for the BRP is developed by Jovanovic, Tuba, and Voß, 2019, which includes an adaptation for the crane time minimization objective. Additionally, da Silva Firmino et al., 2019, proposed an integer linear model, an A* algorithm, and a reactive GRASP with the objective of minimizing working time.

Recent literature on the BRP offers various approaches, with its original objective and close variants, aside from those involving crane time indicated in the previous paragraph. These approaches include branch and bound (Tanaka and Mizuno, 2018, Jin and Tanaka, 2023), branch and cut (Bacci et al., 2020), and iterative deepening A* algorithms (Quispe et al., 2018, Jin, 2020). Several mixed integer programming methods have been proposed, such as the model presented by Lu et al., 2020, which was improved by Liu et al., 2022, and more recent approaches are presented by Kimns and Wilschewski, 2023, and Boge and Knust, 2023). Additionally, Tanaka and Voß, 2022, proposed an algorithm based on integer programming formulations. We can also find several heuristic methods including GRASP (Jovanovic, Tanaka, et al., 2019), beam search (Ting and Wu, 2017, Bacci et al., 2019), and simulated annealing (Boge and Knust, 2023).

Zweers et al., 2020a, and Zweers et al., 2020b, studied a stochastic variant of the BRP, considering a pre-processing phase for premarshalling relocations. In this phase, the number of container relocations is limited, resulting in partial premarshalling. The Premarshalling Problem under Limited Crane Time (CPMP-LCT) introduced in this thesis and addressed in Chapters 5 and 6, also involves partial premarshalling, but from a completely different perspective: the main goal of the CPMP-LCT is to obtain efficient partial premarshalling solutions, rather than using partial premarshalling as an auxiliary pre-processing phase.

2.3 Constraint Programming

Constraint Programming (CP) is a problem-solving technique at the intersection of Artificial Intelligence, Operations Research, and Computer Science. Its origins trace back to Ivan Sutherland's PhD Thesis "Sketchpad: A man-machine graphical communication system" at Massachusetts Institute of Technology in 1963 (Sutherland, 1963). This thesis marks the starting point of the constraint satisfaction branch of Artificial Intelligence. Over time, constraint programming followed two distinct streams: the language stream and the algorithm stream.

The language stream has focused on developing constraint programming languages, which evolved from logic programming languages like Prolog. On the other hand, the algorithm stream has been dedicated to investigating constraint satisfaction and search algorithms. These streams have become closer since the 1990s. The annual International Conference on Principles and Practice of Constraint Programming and a dedicated scientific journal called Constraints were established to deal with the challenge of connecting both streams.

Constraint Programming is primarily designed for combinatorial problems formulated as Constraint Satisfaction Problems (CSPs) and their generalizations. In this section, we outline the definition of such problems and present the fundamental concepts of constraint programming modeling and search algorithms. This technique has been successfully applied in various fields, such as scheduling, assignment problems, vehicle routing, and planning.

Interested readers in a detailed description of the foundations of constraint programming, extensions, and applications can refer to Rossi et al., 2006. For more information about the history of constraint programming, see Barták, 2011.

2.3.1 Constraint Satisfaction Problems

The most basic constraint satisfaction formulation involves determining values for a group of unknowns called variables, such that they satisfy a set of constraints that limit the combinations of values the variables can take together. Formally, a standard Constraint Satisfaction Problem (CSP) is characterized by:

- ▷ A finite set of n variables: $X = \{x_1, \ldots, x_n\}$.
- ▷ A set of non-empty finite domains for the variables: $D = \{D_1, \ldots, D_n\}$, where D_i represents the set of potential values for x_i.
- ▷ A finite set of m constraints $C = \{C_1, \ldots, C_m\}$.

A constraint C_j is defined as a pair (S_j, R_j), where $S_j = \{x_{1j}, \ldots, x_{kj}\}$ is the set of variables involved in this constraint, known as the scope of C_j, and $R_j \subseteq \big(D_{1j} \times \cdots \times D_{kj}\big)$ is the set of value combinations for the variables within the scope of C_j that satisfy this constraint.

Example

A simple CSP example is:

- ▷ Three variables: x_1, x_2 and x_3.
- ▷ The corresponding domains:

$$D_1 = \{2, 3, 4\}$$

$$D_2 = \{1, 2, 3, 4, 5\}$$

$$D_3 = \{1, 2, 5, 6\}$$

- ▷ Two constraints:

$$C_1 : |x_1 - x_2| = 2$$

$$C_2 : x_1 + x_2 \leq x_3$$

Which can also be expressed as:

$$C_1 = (S_1, R_1) = (\{x_1, x_2\}, \{(2,4), (3,1), (3,5), (4,1), (4,2)\}$$

$$C_2 = (S_2, R_2) = (\{x_1, x_2, x_3\}, \{(2,1,5), (2,2,5), (2,3,5),$$
$$(3,1,5), (3,2,5), (4,1,5), (2,1,6), (2,2,6), (2,3,6),$$
$$(2,4,6), (3,1,6), (3,2,6), (3,3,6), (4,1,6), (4,2,6)\}$$

A solution to the problem is a combination of values for the variables satisfying both constraints. In this case, there are 6 solutions: $(x_1, x_2, x_3) \in \{(2,4,6), (3,1,5), (3,1,6), (4,1,5), (4,1,6), (4,2,6)\}$.

2.3.2 Types of constraints

The size of the scope of a constraint is called its arity. Constraints with arity 1, involving only one variable, are known as unary constraints and are typically embedded within the domain of the corresponding variable. A constraint is binary if its arity is 2, ternary if it is 3, and so forth. However, constraints with an arity greater than 2 are generally termed non-binary.

In constraint programming, constraints are not confined to specific conditions such as linearity, as in integer programming models. Consequently, they can often be expressed more naturally than with other optimization techniques. The fundamental idea behind constraint programming is to employ declarative programming language for modeling, focusing on what is desired to obtain rather than how to achieve it.

The example provided in the previous section involves two constraints, one with arity 2 and the other with arity 3. The binary constraint contains an absolute value expression, which is non-linear. Constraints can also incorporate boolean expressions or logical conditions. For instance, given a variable x_1 with an integer domain and a binary variable x_2, the following expressions are valid constraints:

a) $(x_1 > 0) = x_2$

b) If $(x_1 > 0)$ then $x_2 = 1$

In both examples, the term $(x_1 > 0)$ evaluates to 1 if the inequality is true and 0 if false. Both constraints indicate that variable x_2 must be equal to 1 when x_1 is positive, but only (a) imposes that x_1 must be positive when x_2 is assigned value 1.

There is a special kind of non-binary constraints called global constraints. They describe complex relationships between variables that can be typically expressed by the combination of multiple simpler constraints. Therefore, a global constraint condenses a group of constraints, capturing special patterns in combinatorial problems.

One of the most well-known global constraints is the All Different constraint. Given a set of variables, this global constraint ensures that all of them take distinct values. The expression of this condition, which could consist of a group of binary constraints, is streamlined using the All Different constraint. For example, the two following options imply that three variables x_1, x_2, and

x_3 take different values. Using global constraint (b), only that expression must be evaluated, as opposed to three in case (a):

a) $x_1 \neq x_2$; $x_1 \neq x_3$; $x_2 \neq x_3$

b) $\mathit{AllDifferent}(x_1, x_2, x_3)$

In the formulations presented in this thesis, we use a global constraint called Allowed Assignments, which specifies all possible combinations of values that a set of variables can take. Additionally, we use the Count constraint, which indicates how many variables from a given collection must take a particular value.

2.3.3 Particular cases and extensions of CSPs

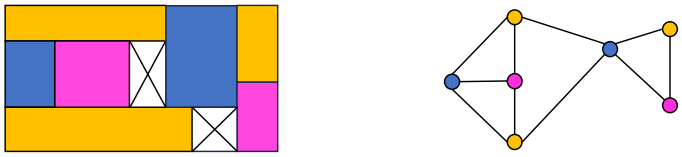
Identifying particular cases of CSPs is helpful as they can benefit from the application of particular problem-solving strategies.

The Boolean Satisfiability Problem (SAT) is a well-known particular case of CSP. It comprises a set of binary variables with domains {True, False} and a constraint that is a boolean formula. A solution to the problem is a combination of values for the variables such that the formula evaluates to true.

Another particular case is when all constraints are unary or binary. These problems can be represented as constraint graphs, where nodes represent variables and edges denote constraints. An edge connecting two nodes signifies that a constraint exists with a scope comprised of the pair of variables corresponding to these nodes.

Figure 2.2 shows an example where the plan of a building must be colored with at most three different colors ensuring that adjacent rooms have distinct colors. In the corresponding graph, each room corresponds to a node, and it is connected through arcs to all adjacent rooms. The problem of coloring a graph giving distinct colors to adjacent nodes is known as the Graph Coloring Problem. If a specific number of colors k is aimed, it is called the k-Coloring Problem.

Figure 2.2: An example of a constraint graph.



A solution to a 3-coloring problem. On the left, a building plan shows rooms colored in three colors, which are different for adjacent rooms. Outdoor areas not requiring coloring are marked with a cross. On the right, the corresponding constraint graph is depicted, with each node representing a room and each arc connecting two adjacent rooms. Source: Elaborated by the author (2024).

Constraint Programming is mainly applied to CSPs, the problems for which it is typically most successful. However, this technique is not restricted to CSPs and is also used for some extensions, such as Soft CSPs or Constraint Optimization Problems (COPs), which differentiate between feasible and optimal solutions.

In a Soft CSP, some constraints are not required to be satisfied to obtain a feasible solution, and the problem includes the objective of maximizing the number of satisfied constraints. An example is solving a 2-coloring problem on the building plan shown in Figure 2.2, minimizing the number of pairs of adjacent rooms that have the same color.

A COP is a type of CSP that includes an objective function. Thus, a solution is considered feasible if it satisfies the CSP and optimal if it additionally provides the best objective function value. The Minimum Graph Coloring Problem is an example of a COP, where a graph must be colored using the minimum number of colors and ensuring that adjacent nodes have different colors.

In this thesis, we tackle COPs, but in most cases, we opt for iterative algorithms that require solving a CSP at each iteration instead of directly solving a global COP since we observed it is a more efficient method.

2.3.4 Search strategies

Constraint programming has two main parts: modeling and search. Once a model is designed employing a constraint language, a search algorithm is required for using the model to explore the solutions space.

One of the simplest search strategies is the Generate and Test (GT) approach. This method starts by assigning to each variable a value within its domain. Then, it is checked whether these assignments satisfy all the constraints. If they do, a solution is found. Otherwise, a new combination of values is attempted. This process continues until a solution is discovered or all possible combinations have been tested.

Although the GT approach is not an efficient search procedure, it reflects the core idea of assigning values to variables and verifying the satisfaction of constraints. To enhance this strategy, potential improvements could be considered, such as generating candidate solutions in a more informed manner or merging the generation of candidate solutions with the testing process. The latter is the main idea behind the backtracking strategy.

The backtracking algorithm systematically constructs a solution by iteratively assigning values to variables and checking the partial solution for consistency with the constraints. The process begins by selecting a variable and assigning it a value. Constraints involving this variable are then checked, and if none are violated, another variable is chosen, and a value is assigned to it. Then, constraints involving the two variables already assigned in the partial solution are checked. This process continues until all variables have been assigned values that satisfy all the constraints, resulting in a valid solution. However, if a constraint is violated at any point in the algorithm, the algorithm backtracks. This means undoing the last value assignment and exploring an alternative one for the variable in question.

While the backtracking algorithm improves the GT approach, it lacks information about the unassigned variables at each step. Constraint propagation addresses this limitation by providing information about the potential values remaining for unassigned variables based on the constraints and current variable assignments. This helps identify infeasible solutions earlier in the search process.

Given an assignment of values to some variables, constraint propagation involves removing values from the current domains of unassigned variables that do not satisfy the constraints according to the current partial solution. This

way, when the domain of an unassigned variable becomes empty, we know the partial solution cannot lead to a complete solution.

Propagation can be conducted within the backtracking algorithm after each new variable assignment. If the domain of an assigned variable becomes empty after propagation, the algorithm backtracks. This enables the detection of infeasible assignments before they are made. Integrating propagation into the backtracking algorithm results in a more effective reduction of the search space, identifying conflicts earlier and avoiding part of the constraint checks.

As mentioned earlier, the search algorithm can also be enhanced by selecting assignments of values to variables according to a specific strategy rather than randomly. The best variable and value selection procedure depend on the problem being solved.

There exist multiple options for variable selection: lexicographical order, prioritizing variables with the smallest domain size or those with the smallest (or largest) value in their domains, or selecting the variable with the smallest value of domain size divided by the number of times that variable has been involved in constraints causing failure earlier in the search, among others. Most variable selection procedures adhere to the fail-first principle, attempting variables that are more likely to lead to failure first, thus reducing the search space.

The most appropriate value selection strategy highly depends on domain sizes or the kind of values they contain. In some cases, it may be advantageous to start by fixing a variable to the smallest value in its domain, the median, or simply a random value. Also, more complex strategies may be beneficial in specific cases. For example, in a problem that can be expressed as a constraint graph, as the one shown in Figure 2.2, a value from a variable's domain could be selected for discarding the fewest values from the domains of the neighboring variables in the graph. The rationale behind this strategy is to choose a value that is most likely to lead to a solution.

There are also different backtracking schemes. The basic one is Chronological Backtracking, which undoes the immediate previous variable assignment. The Backjumping strategy uses the violated constraints to identify the conflicting variable and return to that one. Backchecking and Backmarking save information about incompatible value combinations that have already occurred, ensuring they are not reconsidered.

Propagation can also be performed in various ways. The most basic method is Forward Checking, where any value in the domain of a variable that conflicts with the last assignment made is removed from its domain. Alternative meth-

ods involve checking the domains of variables that are not directly connected with the assigned variables.

Although the default search typically consists of a Depth-Firt Search with Chronological Backtracking and a propagation technique like Forward Checking, other methods, such as local search and dynamic programming, can also be used in a constraint programming solution algorithm.

Chapter 3

A constraint programming approach for the premarshalling problem

The Container Premarshalling Problem (CPMP) has received considerable attention in the scientific literature. Section 2.2 in the previous chapter provides a compilation of solution methods developed for this problem. While many of these methods fall within the field of Operations Research, there have also been recent proposals of Artificial Intelligence techniques. In this thesis, we investigate the potential of Constraint Programming in addressing the premarshalling problem.

This chapter introduces a constraint programming formulation that outperforms the current state-of-the-art integer programming model for the premarshalling problem. A significant observation from this study is the fundamentally different logic behind model construction in constraint programming compared to mixed integer linear programming. To illustrate this contrast, we detail the development of the proposed model through a series of constraint programming formulations. The study presented in this chapter corresponds to Jiménez-Piqueras et al., 2023.

3.1 CPMP: The Container Premarshalling Problem

The port yard serves as a temporary storage facility for containers awaiting loading onto vessels or other means of transport, such as trains or trucks. In this area, containers are arranged in parallel lines of stacks. These stacks of containers are grouped into bays organized within blocks, as illustrated in Figure 3.1.

Figure 3.1: An example of a port yard configuration.

The containers are organized in two blocks with four bays each. A block of containers is highlighted in blue and a bay in yellow. In this example, each bay consists of five stacks with four tiers. Source: Elaborated by the author (2024).

The essence of the CPMP lies in reorganizing containers within a bay to speed up their subsequent retrieval from the yard. When the crane must access a specific container for its retrieval, any containers placed above that one need to be relocated to other stacks. Hence, if the bay is arranged so that these relocations are not required, the retrieval process is expedited. Premarshalling seeks to optimize this arrangement by strategically relocating containers before the retrieval operation begins.

To formally define the CPMP, we distinguish between well-placed and badly placed containers. A container is considered badly placed if it is either on top of a container that needs to be retrieved before it or on top of another badly placed container. Otherwise, the container is deemed well placed. We define a bay as completely arranged if every container is well placed. In that case, all the containers can be retrieved from the bay without requiring additional relocations.

The classical container premarshalling problem aims to find a sequence of a minimum number of container relocations to obtain a bay arrangement without badly placed containers.

The assumptions for the CPMP include:

i) A single bay: The premarshalling process focuses solely on arranging one bay of containers at a time.

ii) No containers entering/leaving the bay: The CPMP literature assumes that no container can be retrieved from or introduced to the bay throughout premarshalling. All the relocations occur within the bay, and the initial set of containers remains unchanged until the bay arrangement is complete. However, in practice, moving containers to another bay during premarshalling is possible and may be advantageous, as discussed in Chapter 4.

iii) Uniform container size: All the containers in the bay have identical dimensions. In the CPMP, the sole distinguishing features between containers are their positions in the bay and their expected retrieval time.

iv) Full information on retrieval times: It is assumed that complete information regarding the order in which containers must leave the bay after premarshalling is available and remains unchanged throughout the arrangement process. However, in reality, the expected schedule may alter due to delays in other port operations, as briefly discussed in Chapter 5.

v) A single crane: There is only one crane available for rearranging the bay, capable of moving only one container at a time.

vi) Uniform movement cost: All relocations incur equivalent costs or durations. The value of a specific relocation is exclusively its potential to reduce the total number of relocations. However, this assumption overlooks the reality that crane travel distances vary among relocations, which significantly impacts time. We tackle this discrepancy in Chapter 5.

vii) Unlimited crane availability: No condition limiting the crane availability is assumed for the CPMP. However, the premarshalling is performed when the crane is not needed for other tasks, and considering infinite availability is unrealistic. We address the premarshalling assuming the existence of a limit on crane time in Chapters 5 and 6.

3.1.1 Notation

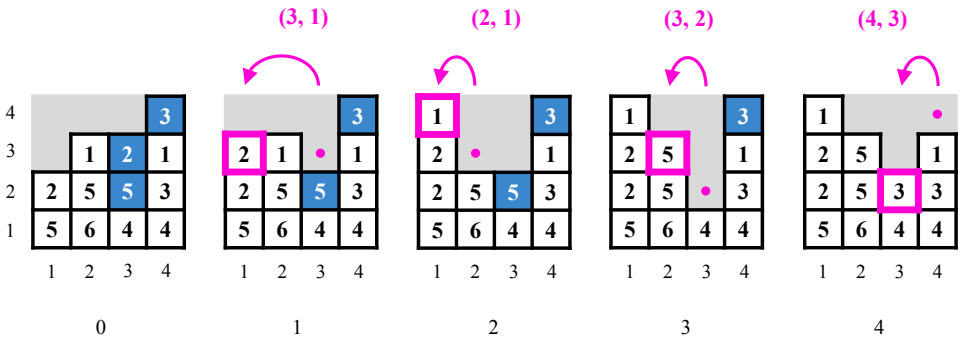
Let us introduce some formal notation. The dimensions of a bay are defined by two parameters: $\bar{s}$, representing the maximum number of stacks, and $\bar{t}$, indicating the maximum number of tiers or the maximum height of each stack. The position of a container is denoted by a pair $(s,t) \in \mathcal{S} \times \mathcal{T}$, where $\mathcal{S} := \{1,2,\ldots,\bar{s}\}$ is the set of stacks, with $\bar{s}$ being the right-most stack, and $\mathcal{T} := \{1,2,\ldots,\bar{t}\}$ the set of tiers, where $\bar{t}$ is the top tier. Let $\bar{c}$ be the total number of containers in the bay, and let $\mathcal{C} := \{1,\ldots,\bar{c}\}$ be the set of containers.

The problem's input is a bay layout determined by the locations of the containers and their retrieval priority. Priorities indicate the expected order of container retrieval from the bay after premarshalling: containers with priority 1 are retrieved first, followed by those with priority 2, and so on until priority $\bar{p}$, with $\bar{p}$ being the total number of priority groups. Let us define the set of priorities as $\mathcal{P} := \{1,2,\ldots,\bar{p}\}$ and another set including 0 for empty slots, $\mathcal{P}^0 := \{0,1,2,\ldots,\bar{p}\}$. A priority group p comprises the containers with retrieval priority p, and the total number of containers in this group is denoted by m_p.

A sequence of container movements or relocations defines a solution to the CPMP. To determine the order of these movements, we use a set of stages $\mathcal{K} := \{1,2,\ldots,\bar{k}\}$. Each relocation occurs at a different stage, and the layout at stage k results from the k^{th} movement, performed during that stage. Therefore, a solution can be represented as a sequence of pairs $\{(s,r)_1, (s,r)_2, \ldots, (s,r)_k, \ldots, (s,r)_{k^*}\}$, where $(s,r)_k$ indicates that the top container of stack s is moved to the top of stack r at stage k, and $k^* \leq \bar{k}$. It is crucial to note that the problem's feasibility requires the total number of stages $\bar{k}$ to be greater than or equal to the optimal objective value. We introduce an additional stage labeled 0 to incorporate the initial bay layout, where no relocations occur. We define an extended set that includes the initial stage: $\mathcal{K}^0 := \{0,1,2,\ldots,\bar{k}\}$.

Figure 3.2 illustrates an optimal solution example for a bay with four stacks and four tiers. Five bay configurations are shown: the initial one at stage 0, and the result of each of the four relocations defining the solution, $\{(3,1)_1, (2,1)_2, (3,2)_3, (4,3)_4\}$. Containers are depicted as boxes labeled with their priority group numbers. Badly placed containers are highlighted in blue. It is noteworthy that in the final stage, all the containers are well placed.

Figure 3.2: An example of an optimal solution to the CPMP.



The numbers at the bottom indicate the stages of the premarshalling solution. Number 0 corresponds to the initial one, 1 to 3 are intermediate stages, and 4 shows the final layout. Stack numbers are indicated below each bay layout, and tier numbers appear next to the initial one. Containers are represented as boxes with the numbers of their priority groups inside, and badly placed containers are highlighted in blue. The available space for relocations is shadowed in gray. The relocation performed at each stage is indicated in magenta by an arrow and a pair (s, r) where s and r are the origin and destination stacks, respectively. The relocated container is highlighted with a thick border, and a dot indicates its origin slot. The indexes corresponding to the tiers are displayed on the left of the figure, and those for the stacks are at the bottom of each bay layout. Source: Elaborated by the author (2024).

3.2 Constraint programming models

We have developed a constraint programming model for the CPMP. In this section, we introduce the proposed formulation and the step-by-step construction process. We consider it interesting to present the building steps for two main reasons: i) designing a constraint programming model typically demands a distinct strategy compared to a mixed integer linear programming model; ii) it facilitates understanding the final formulation and its complexity.

We describe a series of four constraint programming models, where the fourth is the proposed one and the other three are previous steps for building it. The models are named *CPX*, where *X* corresponds to the number of groups of decision variables in the formulation. The first one, *CP2*, involves two sets of variables, and *CP3*, *CP4*, and *CP5* evolve from the preceding one in the series by incorporating an additional group of decision variables and corresponding constraints.

3.2.1 CP2: Constraint programming model with 2 groups of variables

At least two groups of variables are required to formulate the problem because, for each stage, it is necessary to determine both the configuration of the bay and the movement performed. The basic model CP2 is built using only these two types of variables, $x^k_{s,t}$ and $y^k_{s,t}$:

$$x^k_{s,t} = \begin{cases} p & \text{If a container with priority } p \text{ is in slot } (s,t) \text{ at stage } k \\ 0 & \text{If slot } (s,t) \text{ is empty at stage } k \end{cases}$$

$$\forall s \in \mathcal{S},\ \forall t \in \mathcal{T},\ \forall k \in \mathcal{K}^0$$

$$y^k_{s,t} = \begin{cases} 1 & \text{If a container is moved to slot } (s,t) \text{ during stage } k \\ 0 & \text{Otherwise} \end{cases}$$

$$\forall s \in \mathcal{S},\ \forall t \in \mathcal{T},\ \forall k \in \mathcal{K}$$

These two groups of variables are the same as those described in Rendl and Prandtstetter, 2013, on which the CP2 model is loosely based. Unlike the mathematical programming formulations of de Melo da Silva et al., 2018 and Parreño-Torres et al., 2019, where the variables representing the bay configuration are binary and indexed by the containers' priorities, in CP2, these variables are integer, taking values from the set of priorities.

The objective of the premarshalling problem is to minimize the number of relocations required to arrange the bay. In terms of the decision variables described above, it can be expressed as (3.1).

$$\min \sum_{s \in \mathcal{S},\, t \in \mathcal{T},\, k \in \mathcal{K}} y^k_{s,t} \tag{3.1}$$

The CP2 model is formulated as follows:

$$x^0_{s,t} = \alpha^x_{s,t} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T} \tag{3.2}$$

$$Count\big(x^k_{s,t} : s \in \mathcal{S},\, t \in \mathcal{T},\, x^k_{s,t} = p\big) = m_p \qquad \forall p \in \mathcal{P}^0,\, k \in \mathcal{K} \tag{3.3}$$

$$x^{\bar{k}}_{s,t+1} \leq x^{\bar{k}}_{s,t} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T} \setminus \{\bar{t}\} \tag{3.4}$$

$$x^k_{s,t+1} \leq \bar{p} \cdot x^k_{s,t} \qquad s \in \mathcal{S},\, t \in \mathcal{T} \setminus \{\bar{t}\},\; k \in \mathcal{K} \setminus \{\bar{k}\} \tag{3.5}$$

$$x^{k-1}_{s,t} \leq x^k_{s,t} + \bar{p}\,(x^k_{s,t} == 0) \qquad s \in \mathcal{S},\, t \in \mathcal{T},\, k \in \mathcal{K} \tag{3.6}$$

$$x^k_{s,t} \leq x^{k-1}_{s,t} + \bar{p}\,(x^{k-1}_{s,t} == 0) \qquad s \in \mathcal{S},\, t \in \mathcal{T},\, k \in \mathcal{K} \tag{3.7}$$

$$\sum_{s \in \mathcal{S},\, t \in \mathcal{T}} y^k_{s,t} \leq 1 \qquad \forall k \in \mathcal{K} \tag{3.8}$$

$$y^k_{s,t} \leq x^k_{s,t} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T},\, k \in \mathcal{K} \tag{3.9}$$

$$x^k_{s,t} \leq \bar{p}\,(y^k_{s,t} + x^{k-1}_{s,t}) \qquad s \in \mathcal{S},\, t \in \mathcal{T},\, k \in \mathcal{K} \tag{3.10}$$

$$x^{k-1}_{s,t} \leq \bar{p}\,(1 - y^k_{s,t}) \qquad s \in \mathcal{S},\, t \in \mathcal{T},\, k \in \mathcal{K} \tag{3.11}$$

$$y^k_{s,t} \leq x^{k+1}_{s,t} \qquad s \in \mathcal{S},\, t \in \mathcal{T},\, k \in \mathcal{K} \setminus \{\bar{k}\} \tag{3.12}$$

The problem's input is the initial bay layout, determined for every slot (s,t) in $\mathcal{S}\times\mathcal{T}$ by the container priorities and value 0 for empty slots. This configuration is denoted by $\alpha^x_{s,t}$. The superscript x signifies that $\alpha^x_{s,t}$ must be assigned to variables x in the initial stage, as expressed by constraints (3.2). The bay layout during the subsequent stages is defined by constraints (3.3) to (3.7).

The number of containers in the bay with a given priority remains constant across all stages, as it is assumed the priority schedule does not change and no containers are allowed to enter or leave the bay during premarshalling. This condition is ensured by (3.3) through global constraints called Count. This kind of global constraint imposes a particular value on a specific number of variables from a given set. In this case, the number of variables $x^k_{s,t}$ assigned a value p must be m_p, the multiplicity of the priority group p.

Ensuring every container is well placed in the final configuration, i.e., no container is stacked on top of another with an earlier retrieval time, is addressed by (3.4). Constraints (3.5) avoid empty slots between containers within the same stack by imposing that a slot (s,t) must be occupied if there is a container in slot $(s,t+1)$.

When a slot is occupied during two consecutive stages, the container occupying it must be the same across both stages, meaning the assigned priority must remain unchanged. This is expressed by (3.6) and (3.7) using logical expressions of the form $(x == 0)$, that return value 1 when true (if $x = 0$), and 0 when false (if $x > 0$). Since (3.6) and (3.7) fix the priority of containers that remain stationary and constraints (3.3) ensure a constant number of containers in each priority group, it is guaranteed that the priority of the container being moved is not altered during the relocation.

Constraints (3.8) to (3.12) address the conditions regarding the relocations.

The limit of at most one movement per stage corresponds to (3.8). Constraints (3.9) ensure that when a container is moved to a slot during stage k, the slot is occupied by a container at that stage. Constraints (3.10) impose that if a container occupies a slot in stage k, it must either have been there during the previous stage or moved to that slot in stage k. Constraints (3.11) express that a container can only be moved to a slot in stage k if the slot was empty during the previous stage.

Finally, (3.12) are included to discard solutions faster in the search process. They indicate that when a container is moved to a slot, this slot must remain occupied in the next stage. While this condition is not necessary for a feasible solution, it is always satisfied by an optimal one. Ensuring that the same container is not moved in consecutive stages leads to a more efficient solution, as moving the same container in two consecutive stages can be accomplished in just one relocation. This rule expressed in (3.12) is known as "transitive move avoidance" (Tierney et al., 2016).

3.2.2 CP3: Constraint programming model with 3 groups of variables

In the CP2 model, the bay configuration at each stage is described by variables $x^k_{s,t}$, which take integer values from the set of priorities $\mathcal{P}^0$. While integer-valued variables are suitable for certain constraints, they are cumbersome for others. Additionally, variables $x^k_{s,t}$ and $y^k_{s,t}$ are weakly related in the CP2 formulation. To address these issues, model CP3 is derived from CP2, including a new set of binary variables, $\delta^k_{s,t}$. Unlike $x^k_{s,t}$, variables $\delta^k_{s,t}$ do not provide information about priorities but simply indicate whether a slot is empty or not:

$$\delta^k_{s,t} = \begin{cases} 1 & \text{If there is a container in slot } (s,t) \text{ at stage } k \\ 0 & \text{If slot } (s,t) \text{ is empty at stage } k \end{cases}$$

$$\forall s \in \mathcal{S},\, \forall t \in \mathcal{T},\, \forall k \in \mathcal{K}^0$$

In recent integer models, such as de Melo da Silva et al., 2018, and Parreño-Torres et al., 2019, the layout is represented by 4-index binary variables. In contrast, this approach uses two groups of 3-index variables: one binary, $\delta^k_{s,t}$, and the other integer, $x^k_{s,t}$. While this results in far fewer variables, it requires establishing the relationship between the two groups. Constraints (3.13) and (3.14) are defined to this end, and the rest of the constraints in CP3 source from CP2, included with exactly the same definition or rewritten using the new variables introduced. Those remaining the same are (3.2), (3.3), (3.4) and (3.8), and constraints defined by modifying CP2 constraints are (3.15) to (3.21), as indicated in Table 3.1. The objective function described for CP2 (3.1) also applies to the CP3 model.

Table 3.1: Constraints from model CP2 substituted in model CP3.

CP2	(3.6)	(3.7)	(3.9)	(3.10)	(3.11)	(3.12)
CP3	(3.15)	(3.16)	(3.17)	(3.18)	(3.19) (3.20)	(3.21)

The first row of the table shows the constraints of model CP2 that have been substituted in model CP3 with the corresponding constraints in the second row. Source: Jiménez-Piqueras et al., 2023.

$$x^k_{s,t} \le \bar{p} \cdot \delta^k_{s,t} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T},\, k \in \mathcal{K}^0 \qquad (3.13)$$

$$\delta^k_{s,t} \le x^k_{s,t} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T},\, k \in \mathcal{K}^0 \qquad (3.14)$$

$$x^{k-1}_{s,t} \le x^k_{s,t} + \bar{p}\left(1 - \delta^k_{s,t}\right) \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T},\, k \in \mathcal{K} \qquad (3.15)$$

$$x^k_{s,t} \le x^{k-1}_{s,t} + \bar{p}\left(1 - \delta^{k-1}_{s,t}\right) \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T},\, k \in \mathcal{K} \qquad (3.16)$$

$$y^k_{s,t} \le \delta^k_{s,t} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T},\, k \in \mathcal{K} \qquad (3.17)$$

$$\delta^k_{s,t} \le y^k_{s,t} + \delta^{k-1}_{s,t} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T},\, k \in \mathcal{K} \qquad (3.18)$$

$$y^k_{s,t+1} + \delta^{k-1}_{s,t+1} \le \delta^{k-1}_{s,t} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T} \setminus \{\bar{t}\},\, k \in \mathcal{K} \qquad (3.19)$$

$$y^k_{s,1} + \delta^{k-1}_{s,1} \le 1 \qquad \forall s \in \mathcal{S},\, k \in \mathcal{K} \qquad (3.20)$$

$$y^k_{s,t} \le \delta^{k+1}_{s,t} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T},\, k \in \mathcal{K} \setminus \{\bar{k}\} \qquad (3.21)$$

Constraints (3.13) and (3.14) are added to relate the integer variables x with the new group of binary variables δ. These constraints ensure that $\delta^k_{s,t}$ takes the value 0 when $x^k_{s,t}$ is 0, and 1 when $x^k_{s,t}$ is positive.

The constraints that use the information about whether a slot is occupied or not change from CP2 to CP3 because it is easier and more precise to write them in terms of variables δ instead of x. Following this idea, the constraints listed in Table 3.1 are substituted accordingly.

Constraints (3.15) and (3.16) are derived from (3.6) and (3.7) of CP2 by replacing the logical expression involving variables x by a linear expression in terms of δ. They express the same condition in both models: the priority associated with a slot must be the same in consecutive stages if no container is placed or removed.

To ensure that the destination slot of the container relocated at a stage becomes occupied in that stage, CP3 includes constraints (3.17) instead of (3.9), as the information provided by binary variables δ is sufficient, eliminating the need for integer variables x. Similarly, constraints (3.10) are substituted with (3.18), which express that an occupied slot at stage k must either have been occupied in $k-1$ or that a container was moved into it at k.

Constraints (3.11) from CP2 are substituted with (3.19) and (3.20) in CP3. In constraints (3.19), not only variables $x^k_{s,t}$ are replaced with $\delta^k_{s,t}$, but also the term $\delta^{k-1}_{s,t}$ is incorporated to enhance the constraints: (3.19) impose that a container can only be moved to slot (s,t) if it was empty during the previous stage, as in (3.11), while also requiring that slot $(s,t-1)$ was occupied. The additional term cannot apply to constraints for the first tier, and thus (3.20) are introduced for that case.

Constraints (3.12) are replaced by (3.21), which express the transitive move avoidance using variables δ instead of x.

Constraints (3.5) were included in CP2 to prevent empty slots between containers within the same stack. However, this condition is covered by constraints (3.19) in CP3, and (3.5) are not needed.

3.2.3 CP4: Constraint programming model with 4 groups of variables

In the previous models, we introduced variables $y^k_{s,t}$ that identify the slot to which a container is moved. In CP4, the definition of the movements is more precise due to a new group of variables that indicate the origin slot of a relocation:

$$z^k_{s,t} = \begin{cases} 1 & \text{If a container is removed from slot } (s,t) \text{ during stage } k \\ 0 & \text{Otherwise} \end{cases}$$

$$\forall s \in \mathcal{S},\ \forall t \in \mathcal{T},\ \forall k \in \mathcal{K}$$

Adding variables $z^k_{s,t}$ allows a stronger connection between the variables that describe the bay configuration and those determining the movements. Variables that identify the origin slot of each relocation are also used in the integer programming models proposed by de Melo da Silva et al., 2018, and Parreño-Torres et al., 2019.

The objective function (3.1) remains suitable for CP4, although an equivalent one using variables $z^k_{s,t}$ could also be employed. The constraints in model CP4 include (3.2) to (3.4), (3.8), (3.13) to (3.16), (3.20) and (3.21) from the previous models. Also, some constraints from CP3 are extended in CP4 to the new variables, namely (3.22), (3.25) and (3.26), or are reformulated by using these variables, (3.23) and (3.24). The connection between these constraints and those from CP3 is specified in Table 3.2. Additionally, the CP4 formulation incorporates (3.27) and (3.28) to include extra conditions to expedite the search for solutions.

Table 3.2: Constraints from model CP3 substituted in model CP4.

CP2	(3.8)	(3.10)	(3.11)	
CP3	(3.8)	(3.18)	(3.19)	(3.20)
CP4	(3.8) (3.22)	(3.23)	(3.24)	(3.20) (3.25) (3.26)

The first rows show the constraints of models CP2 and CP3 that have been substituted in model CP4 with the corresponding constraints in the bottom row, or that remain but are extended with additional constraints using the new variables introduced in CP4. Source: Jiménez-Piqueras et al., 2023.

$$\sum_{s\in\mathcal{S},\, t\in\mathcal{T}} z^k_{s,t} \leq 1 \qquad \forall k \in \mathcal{K} \tag{3.22}$$

$$\delta^k_{s,t} + z^k_{s,t} = y^k_{s,t} + \delta^{k-1}_{s,t} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T},\, k \in \mathcal{K} \tag{3.23}$$

$$y^k_{s,t+1} + \delta^{k-1}_{s,t+1} + z^k_{s,t} \leq \delta^{k-1}_{s,t} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T} \setminus \{\bar{t}\},\, k \in \mathcal{K} \tag{3.24}$$

$$z^{k+1}_{s,\bar{t}} + y^k_{s,\bar{t}} \leq \delta^k_{s,\bar{t}} \qquad \forall s \in \mathcal{S},\, k \in \mathcal{K} \setminus \{\bar{k}\} \tag{3.25}$$

$$z^1_{s,\bar{t}} \leq \delta^0_{s,\bar{t}} \qquad \forall s \in \mathcal{S} \tag{3.26}$$

$$\sum_{t\in\mathcal{T}} y^k_{s,t} + \sum_{t\in\mathcal{T}} z^{k+1}_{s,t} \leq 1 \qquad \forall s \in \mathcal{S},\, k \in \mathcal{K} \setminus \{\bar{k}\} \tag{3.27}$$

$$\sum_{t\in\mathcal{T}} \left(z^k_{s,t} + y^k_{v,t} + z^{k+1}_{u,t} + y^{k+1}_{r,t} \right) \leq 3 \qquad \forall s \in \mathcal{S},\, v \in \mathcal{S} \setminus \{s\},\; u \in \mathcal{S} : u < s \wedge u \neq v,\; r \in \mathcal{S} \setminus \{s, v, u\},\, k \in \mathcal{K} \setminus \{\bar{k}\} \tag{3.28}$$

A new group of constraints analogous to (3.8) is introduced for variables z, (3.22). It expresses that, at most, one container can be removed from its slot at each stage.

Incorporating variables z into the model enables the strengthening of certain constraints, such as (3.18) and (3.19), which are replaced by (3.23) and (3.24), respectively. Specifically, (3.23) are obtained by including the term $z^k_{s,t}$ in (3.18), which allows to impose the equality between the two sides of the constraints. Constraints (3.23) and (3.24) regulate the correspondence between empty and occupied slots and placing or retrieving containers, while (3.18) and (3.19) only address the placing of containers.

In particular, constraints (3.24) impose the condition that a container can only be retrieved from a slot that was occupied at the previous stage. Constraints (3.20), which complements (3.19) for the first tier in CP3, are maintained, and analogous constraints for variables z are added to the formulation: (3.25) and (3.26). In (3.25), the term $y^k_{s,\bar{t}}$ is not necessary, but it narrows the range of possible values for $z^{k+1}_{s,\bar{t}}$. While these constraints without $y^k_{s,\bar{t}}$ only indicate that a container cannot be removed from an empty slot, including $y^k_{s,\bar{t}}$ also express that a container moved to a slot in stage k cannot be removed from that slot during stage $k+1$. However, the term $y^k_{s,\bar{t}}$ cannot be included in the constraints for the initial stage, and we define (3.26) for this case.

Constraints (3.17) are omitted in CP4 because the condition that the destination slot of a relocation must become occupied is already imposed by (3.23), (3.24), (3.20), (3.25) and (3.26) combined.

Two new groups of constraints are introduced to discard symmetries and infeasible solutions earlier. (3.27) impose that when a container is moved to a stack, no container can be removed from that stack in the next stage. Constraints (3.28) serve to break symmetries. Specifically, if r, s, u, v are four different stacks, the sequences of relocations $\{(r,s)_k, (u,v)_{k+1}\}$ and $\{(u,v)_k, (r,s)_{k+1}\}$ produce an identical result. Constraints (3.28) allow only the first sequence, thus addressing this symmetry.

3.2.4 CP5: Constraint programming model with 5 groups of variables

In the final formulation CP5, the group of variables $w^k_{s,t}$ is incorporated to identify the badly placed containers at each stage. More precisely, these variables indicate whether there is a badly placed container in a slot or not.

$$w^k_{s,t} = \begin{cases} 1 & \text{If there is a badly placed container in } (s,t) \text{ during stage } k \\ 0 & \text{Otherwise} \end{cases}$$

$$\forall s \in \mathcal{S},\ \forall t \in \mathcal{T},\ \forall k \in \mathcal{K}$$

The target of premarshalling is to relocate the badly placed containers so they become well placed. Therefore, information about where and how many badly placed containers are is extremely valuable. The number of badly placed containers in the initial bay layout is a lower bound for the number of relocations of any premarshalling solution. Moreover, this principle applies to each premarshalling stage, where the number of badly placed containers is a lower bound for the minimum number of movements necessary to complete the bay's arrangement from that layout. Consequently, using variables $w^k_{s,t}$ allows us to detect solutions where the lower bound at a particular stage exceeds the number of movements still to be conducted, which clearly indicates infeasibility. Hence, these variables facilitate the early elimination of infeasible solutions during the search process. To the best of our knowledge, variables of this type have not been used in previous models in the literature. They are a salient feature of CP5 since they significantly improve the formulation.

The CP5 formulation comprises the objective function (3.1), all constraints from model CP4 except for (3.4), and constraints (3.29) to (3.34), which use variables w.

$$x^k_{s,t+1} \leq x^k_{s,t} + \bar{p} \cdot w^k_{s,t+1} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T} \setminus \{\bar{t}\},\, k \in \mathcal{K} \quad (3.29)$$

$$w^k_{s,t} + \delta^k_{s,t+1} \leq w^k_{s,t+1} + 1 \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T} \setminus \{\bar{t}\},\, k \in \mathcal{K} \quad (3.30)$$

$$w^k_{s,t} \leq \delta^k_{s,t} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T},\, k \in \mathcal{K} \quad (3.31)$$

$$w^k_{s,1} = 0 \qquad \forall s \in \mathcal{S},\, k \in \mathcal{K} \quad (3.32)$$

$$x^k_{s,t} + 1 \leq x^k_{s,t+1} + (\bar{p}+1) \cdot (1 - w^k_{s,t+1} + w^k_{s,t}) \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T} \setminus \{\bar{t}\},\, k \in \mathcal{K} \quad (3.33)$$

$$\sum_{s \in \mathcal{S}, t \in \mathcal{T}} w^k_{s,t} + k \leq \bar{k} \qquad \forall k \in \mathcal{K} \quad (3.34)$$

Constraints (3.29) to (3.33) specify which slots are occupied or not by badly placed containers. Constraints (3.29) express that if a container is on top of another container with an earlier retrieval time, it is badly placed. Constraints (3.30) impose that if there is a badly placed container in slot (s,t) and slot $(s,t+1)$ is occupied, then the container in slot $(s,t+1)$ is also badly placed. There cannot be badly placed containers in empty slots or the first tier, as indicated by (3.31) and (3.32), respectively. Constraints (3.33) ensure that if there are no badly placed containers in slot (s,t) and no container with a higher priority in $(s,t+1)$, then there is no badly placed container in slot $(s,t+1)$.

Introducing variables w serves to enforce that the number of badly placed containers at each stage does not exceed the number of remaining stages, which is the maximum number of movements that can still be performed. This is a necessary condition for every feasible solution and enhances the formulation reducing the range of possibilities to explore to build a solution. It is included in CP5 through constraints (3.34), which impose that the number of badly placed containers at each stage must be less than or equal to the number of subsequent stages.

Constraints (3.4), which specify that there must be no badly placed container in the last stage, are not included in the CP5 model. This condition is already implied by the combination of constraints (3.29) and (3.30), which identify badly placed containers, and (3.34), which limit the number of them at each stage.

3.2.5 Solution method

A straightforward method for solving the proposed constraint programming models could be the solution procedure proposed by de Melo da Silva et al., 2018. In this approach, the number of stages is fixed to an upper bound for the number of relocations needed to arrange the bay. A solver is then employed to find a solution that minimizes the objective function value. However, the optimization model may become significantly large in the absence of a tight upper bound for use as a fixed number of stages. Obtaining tight upper bounds for the premarshalling problem is tough and requires complex metaheuristic algorithms. Hence, a method that does not rely on an upper bound would be desirable.

An alternative approach in the CPMP literature that does not need an upper bound is the iterative algorithm used by Parreño-Torres et al., 2019, and Rendl and Prandtstetter, 2013. Each iteration involves solving the problem given a specific number of stages $\bar{k}$. In the first iteration, $\bar{k}$ is set to a lower bound for the number of relocations needed to arrange the bay entirely. If an infeasibility is encountered, the algorithm iterates, incrementing $\bar{k}$ by one and solving the problem with $\bar{k}$ stages. Essentially, the algorithm begins with a lower bound for the minimization objective function and updates it until a feasible solution is found. Consequently, once a feasible solution is obtained, the algorithm terminates, as this solution is optimal.

The iterative algorithm offers a notable advantage over the first approach discussed in this section by relying on a lower bound rather than an upper bound for the number of container movements since high-quality lower bounds are available for the CPMP. To our knowledge, the most refined lower bound in the literature is the one introduced by Tanaka et al., 2019, which has been demonstrated to be very tight. This lower bound is an enhancement of the one developed by Tanaka and Tierney, 2018, which builds upon the lower bound proposed by Bortfeldt and Forster, 2012.

We choose the iterative algorithm approach for solving the constraint programming models presented in the previous sections for two main reasons: the availability of a tight lower bound (together with the absence of a tight upper bound) and the generally better performance of constraint programming in solving satisfaction problems compared to optimization problems. At each iteration of the algorithm, setting the number of stages to the best lower bound for the number of relocations restricts the range of possible values for the objective function to a singleton. In other words, a satisfaction problem must be solved at each iteration, given a fixed value for the objective.

The proposed solution method for models CP2 to CP5 employs a constraint programming solver to find a feasible solution given a fixed objective value. However, instead of the solver's optimization algorithm, we use the iterative algorithm described above. For this purpose, we omit the objective function (3.1) and provide a satisfaction problem for the solver at each iteration. Additionally, since the number of stages corresponds to the number of relocations for the problem to solve at any iteration, we replace constraints (3.8) by (3.35) in CP2 to CP5, and substitute (3.22) with (3.36) in CP4 and CP5. This way, more information is available for the solver to make the value selection for decision variables and the propagation.

$$\sum_{s \in \mathcal{S},\, t \in \mathcal{T}} y^k_{s,t} = 1 \qquad \forall k \in \mathcal{K} \tag{3.35}$$

$$\sum_{s \in \mathcal{S},\, t \in \mathcal{T}} z^k_{s,t} = 1 \qquad \forall k \in \mathcal{K} \tag{3.36}$$

Several approaches were tested during the design of the solution method before choosing the final one described above. Further details on these unsuccessful alternatives are provided in Section 3.3.4.

3.3 Computational experiments

The models introduced in the previous sections have been analyzed through an extensive computational study on four datasets described in Section 3.3.1: BZ, ZJY, EMM, and CV.

These experiments serve multiple purposes: to validate the formulations presented in this chapter, to assess the performance differences from CP2 to CP5, and to compare the final model, CP5, with the integer programming model proposed by Parreño-Torres et al., 2019, referred to as IPS6.

Furthermore, we contrast CP5 with an equivalent mathematical programming model formulation CP5^{IP}. Analogously, we compare IPS6 with its constraint programming version IPS6^{CP}. This analysis allows us to investigate how solving similar formulations with the two different techniques, constraint programming and mathematical programming, affects the results.

In addition to the experiments, we also compare the sizes of the four constraint programming models to observe the impact of incorporating each new group of variables.

3.3.1 Technical details of the experiments

All the solution methods presented in this thesis have been tested through computational experiments to evaluate their performance.

The experiments are executed with a time limit of 3600 seconds. We use virtual machines equipped with four processors and 16 GBytes of RAM, operating within an OpenStack virtualization cluster supported by several servers in 1U blade configuration. Each server contains two Intel Xeon Gold 5220 processors, with 18 cores each, running at 2.2 GHz and 384 GBytes of RAM.

The models and algorithms are implemented using IBM ILOG CPLEX Optimization Studio. We use the IBM solver CP Optimizer for the constraint programming models and CPLEX for the mathematical programming models, both at version 20.1.0, the latest version available during the development of this thesis.

Four well-known datasets from the literature on the premarshalling problem are used in the experiments and are described below. Each instance consists of a bay of containers with information about retrieval priorities.

BZ dataset

The BZ dataset from van Brink and van der Zwaan, 2014, contains 960 instances. There are sixteen categories with varying bay dimensions and fill percentages (the rate of occupied slots). The number of stacks is 3, 5, 7, or 9, the number of tiers is 4 or 6, and the fill percentages are 50% or 70%. Each category includes sixty instances, where the containers are classified into 2, 3, or 6 priority levels (twenty instances of each type). The distribution of containers in the bay varies among the instances.

ZJY dataset

The ZJY dataset, generated by Zhang et al., 2015, consists of 100 instances. There are four categories with 4 tiers and 6, 7, 8, or 9 stacks, and one category with 5 tiers and 6 stacks. The container fill percentage is 70% for all categories. Zhang et al., 2015, do not provide the details of how these instances were generated. We observe that multiple containers in the bay may have the same priority. Also, we can see that not all stacks are filled with the same number of containers, and there may be one or two empty slots per stack.

EMM dataset

The EMM dataset was initially generated by Expósito-Izquierdo et al., 2012, and later by Tierney et al., 2016, after the original set was lost. We considered a set of 450 instances from this group and selected only those instances for which the initial bay layout was not already arranged, resulting in 417 instances. All the bays have 4 tiers, and 4, 7, or 10 stacks. For the same bay dimensions, there are instances with fill percentages of 50% and others with 75%, making a total of six instance categories.

CV dataset

The CV dataset was generated by Caserta and Voß, 2009. We consider 320 instances from this dataset. They are divided into categories comprising forty instances, with bay dimensions of 3 to 8 stacks with 5 tiers, and 4 and 5 stacks with 6 tiers. Additionally, for the experiments in this chapter, we consider two categories with 6 and 7 stacks and 6 tiers. Unlike the other datasets in this study, in the CV dataset, all containers in the bay have distinct retrieval priorities. The stacks are filled with the same number of containers, and two additional empty tiers are included in the dimensions described above. The container fill percentage is 60% for instances with 5 tiers and near 70% for categories with 6 tiers.

3.3.2 Size of the CP models

The number of variables and constraints for each CP model presented in this chapter depends on several factors: the number of stacks and tiers of the bay, the number of stages, and the number of priority groups. Table 3.3 shows these figures for the instances with the largest dimensions in each dataset.

Regarding the number of stages $\bar{k}$, we have considered the lower bound for the number of movements, i.e., the number of stages used in the first iteration of the algorithm described in Section 3.2.5. Since there are several instances with the same values of the parameters $\bar{s}$, $\bar{t}$, and $\bar{p}$, but varying in lower bound, we have chosen the largest lower bound value to obtain the figures in Table 3.3.

We can observe a gradual increase in the number of variables from one model to the next in the series. However, the number of constraints experiences a sharp increase from model CP3 to CP4 due to the addition of constraints (3.28). It should be noted that, despite the augmentation in model size, (3.28) serve to break symmetries and accelerate the search process.

Table 3.3: Size of the CP models.

					#Variables				#Constraints			
Dataset	$\bar{s}$	$\bar{t}$	$\bar{k}$	$\bar{p}$	CP2	CP3	CP4	CP5	CP2	CP3	CP4	CP5
BZ	9	6	27	6	2,970	4,482	5,940	7,398	10,179	12,033	50,391	55,719
ZJY	9	4	17	10	1,260	1,908	2,520	3,132	4,335	5,199	29,093	31,225
EMM	10	4	28	8	2,280	3,440	4,560	5,680	7,840	9,350	76,848	80,766
CV	7	6	27	28	2,310	3,486	4,620	5,754	8,559	10,001	20,185	24,335

Number of variables and constraints for the instances with the largest dimensions in each dataset, namely the number of stacks, $\bar{s}$, tiers, $\bar{t}$, priority groups, $\bar{p}$, and stages, $\bar{k}$. Source: Jiménez-Piqueras et al., 2023.

3.3.3 Performance of models CP2, CP3, CP4 and CP5

The results obtained from the four datasets reveal that each of the constraint programming models developed in this study outperforms its predecessor in the modeling process. In other words, with the addition of each new group of decision variables and the corresponding constraints, the formulation improves. Tables 3.4 to 3.7 present these results for each dataset. The results progression from CP2 to CP5 is evident: the total number of solved instances significantly increases, and the average running time on the instances solved by the four models experiences a dramatic decrease.

The increase in solved instances from the most basic model to the most complex one is clear in all four tables. Particularly notable is the performance on the CV dataset, which contains the most difficult instances: CP2 solves fewer than 25% of the instances, whereas CP5 solves more than 70% of them within an hour, as shown in Table 3.7. Similarly, there is a significant performance difference for the ZJY instances, as depicted in Table 3.5, where CP2 solves 26% of the instances compared to CP5, which solves 93%.

Although the total number of instances solved increases from CP2 to CP5, the dominance of the models is not strict. There are a few instances that are solved by a given model but not by every following model in the series. For example, in Table 3.4 for the BZ dataset, the number of instances solved by all the models is lower than the number of instances solved by CP2 (779 vs. 790). Similarly, for the EMM dataset in Table 3.6, we observe there are 308 instances solved by all the models compared to 313 instances solved by CP2.

The running time for the instances solved by all four models experiences an enormous decrease from the initial formulation to the final one. For the ZJY and CV datasets, the time is more than 75 times greater for CP2 than for CP5, and more than 20 times greater regarding the BZ and EMM instances. It is important to note that the BZ and EMM datasets contain some groups of instances that are easily solved by the first model in the series, and hence, there is little room for improvement in the corresponding running times.

Table 3.4: Performance of models CP2, CP3, CP4, and CP5 on the BZ dataset.

				#Optimal					Average time (s)			
$\bar{s}$	$\bar{t}$	%	#Inst.	CP2	CP3	CP4	CP5	All	CP2	CP3	CP4	CP5
3	4	50	60	**60**	**60**	**60**	**60**	60	**0.3**	**0.3**	**0.3**	**0.3**
3	4	70	60	**60**	**60**	**60**	**60**	60	1.5	1.0	**0.6**	0.7
3	6	50	60	**60**	**60**	**60**	**60**	60	7.2	2.1	0.8	**0.7**
3	6	70	60	**60**	**60**	**60**	**60**	60	248.4	28.0	**9.0**	9.9
5	4	50	60	**60**	**60**	**60**	**60**	60	0.5	0.4	**0.3**	**0.3**
5	4	70	60	**60**	**60**	**60**	**60**	60	8.4	3.9	1.3	**0.9**
5	6	50	60	56	57	**60**	**60**	56	77.1	20.9	2.0	**1.0**
5	6	70	60	32	34	46	**51**	30	454.2	118.2	8.9	**6.0**
7	4	50	60	**60**	**60**	**60**	**60**	60	2.0	1.5	**0.6**	**0.6**
7	4	70	60	59	59	**60**	**60**	59	64.2	17.2	4.3	**2.3**
7	6	50	60	50	50	56	**60**	48	196.2	63.6	4.1	**2.1**
7	6	70	60	19	25	33	**42**	18	420.3	109.7	14.5	**8.7**
9	4	50	60	**60**	**60**	**60**	**60**	60	17.6	9.4	1.9	**1.8**
9	4	70	60	50	50	**60**	**60**	47	153.6	109.3	28.3	**6.0**
9	6	50	60	39	43	58	**60**	36	107.9	87.9	9.3	**8.1**
9	6	70	60	5	10	25	**33**	5	164.0	178.9	13.3	**9.3**
Total			960	790	808	878	**906**	779	87.0	29.2	4.8	**2.8**

Results on the instances from the BZ dataset, classified by the number of stacks, $\bar{s}$, the number of tiers, $\bar{t}$, and the fill percentage (%) of the bay. #Inst. refers to the number of instances in each group. Columns #Optimal show the number of instances optimally solved by each model. In the columns on the right, the average running time is displayed for the instances solved by all the models, i.e., those indicated in column All. The best values are shown in bold. Source: Jiménez-Piqueras et al., 2023.

Table 3.5: Performance of models CP2, CP3, CP4, and CP5 on the ZJY dataset.

			#Optimal					Average time (s)			
$\bar{s}$	$\bar{t}$	#Inst.	CP2	CP3	CP4	CP5	All	CP2	CP3	CP4	CP5
6	4	20	9	14	19	**20**	9	452.1	78.1	7.1	**4.0**
6	5	20	1	2	9	**15**	1	2201.2	343.2	**17.0**	22.3
7	4	20	6	10	17	**19**	6	1622.6	426.5	12.4	**5.5**
8	4	20	8	10	14	**20**	8	295.0	324.4	10.1	**5.2**
9	4	20	2	6	9	**19**	2	197.8	323.5	8.1	**6.1**
Total		100	26	42	68	**93**	26	721.6	263.4	9.7	**5.6**

Results on the instances from the ZJY dataset, classified by the number of stacks of the bay, $\bar{s}$, and the number of tiers, $\bar{t}$. #Inst. refers to the number of instances in each group. Columns #Optimal show the number of instances optimally solved by each model. In the columns on the right, the average running time is displayed for the instances solved by all the models, i.e., those indicated in column All. The best values are shown in bold. Source: Jiménez-Piqueras et al., 2023.

Table 3.6: Performance of models CP2, CP3, CP4, and CP5 on the EMM dataset.

			#Optimal					Average time (s)			
$\bar{s}$	%	#Inst.	CP2	CP3	CP4	CP5	All	CP2	CP3	CP4	CP5
4	50	58	**58**	**58**	**58**	**58**	58	**0.3**	**0.3**	**0.3**	**0.3**
4	75	65	**65**	**65**	**65**	**65**	65	147.2	31.7	**12.5**	15.0
7	50	69	**69**	**69**	**69**	**69**	69	7.9	4.0	1.4	**1.0**
7	75	75	33	39	50	**55**	32	265.4	68.4	13.7	**9.6**
10	50	75	73	70	**75**	**75**	70	168.4	87.5	6.8	**4.0**
10	75	75	15	19	27	**40**	14	391.9	97.9	15.2	**10.3**
Total		417	313	320	344	**362**	308	116.6	39.1	6.7	**5.8**

Results on the instances from the EMM dataset, classified by the number of stacks, $\bar{s}$, the number of tiers, $\bar{t}$, and the fill percentage (%) of the bay. #Inst. refers to the number of instances in each group. Columns #Optimal show the number of instances optimally solved by each model. In the columns on the right, the average running time is displayed for the instances solved by all the models, i.e., those indicated in column All. The best values are shown in bold. Source: Jiménez-Piqueras et al., 2023.

Table 3.7: Performance of models CP2, CP3, CP4, and CP5 on the CV dataset.

			#Optimal					Average time (s)			
$\bar{s}$	$\bar{t}$	#Inst.	CP2	CP3	CP4	CP5	All	CP2	CP3	CP4	CP5
3	5	39	**39**	**39**	**39**	**39**	39	76.0	13.6	5.4	**3.8**
4	5	40	35	**40**	**40**	**40**	35	316.8	54.3	7.3	**4.1**
5	5	40	17	31	37	**39**	17	731.4	122.5	10.0	**4.7**
6	5	40	4	11	28	**39**	4	519.3	137.0	15.5	**4.4**
7	5	40	2	5	12	**36**	2	1852.3	136.4	11.5	**5.8**
8	5	40	0	2	9	**30**	0	-	-	-	-
4	6	40	5	16	**32**	31	5	284.7	45.6	8.6	**6.7**
5	6	40	0	1	7	**17**	0	-	-	-	-
6	6	40	0	0	1	**10**	0	-	-	-	-
7	6	40	0	0	0	0	0	-	-	-	-
Total		399	102	145	205	281	102	330.3	54.5	7.5	**4.3**

Results on 400 instances from the CV dataset, classified by the number of stacks of the bay, $\bar{s}$, and the number of tiers, $\bar{t}$. #Inst. refers to the number of instances in each group, where one instance is omitted for being the initial bay layout already arranged. Columns #Optimal show the number of instances optimally solved by each model. In the columns on the right, the average running time is displayed for the instances solved by all the models, i.e., those indicated in column All. The best values are in bold. Source: Jiménez-Piqueras et al., 2023.

The cumulative effect of the enhanced models, progressing from CP2 to CP5, on attaining optimal solutions is summarized in Table 3.8. This table presents the percentage of optimal solutions achieved by each model across the four datasets. The results vary depending on the datasets' difficulty, with BZ being the easiest to solve and CV being the most challenging. Let us analyze how the differences between the models explain the data in Table 3.8.

The values in the CP2 column reflect that, although the formulation is correct, the relationship between the integer variables describing the bay configuration, $x^k_{s,t}$, and the binary variables indicating the destination slot of a relocation, $y^k_{s,t}$, is rather loose and the model cannot yield a solution in numerous instances.

By introducing a new group of binary variables for identifying the occupied slots, $\delta^k_{s,t}$, in the CP3 model, the relationship between variables is strengthened, resulting in improved outcomes across all datasets, with an average increase of 7.5% in optimal solutions.

In the CP4 model, the introduction of variables $z^k_{s,t}$, which denote the origin of each movement, reinforces the connection between container positions and

relocations in the formulation. This enhancement yields an average increase of 13.2% in optimal solutions.

The CP5 model includes an additional set of variables, $w^k_{s,t}$, to identify the badly placed containers, enabling earlier detection of infeasible solutions. Consequently, the performance of the constraint programming solver improves, leading to an average increase of 13% more optimal solutions and consolidating this formulation as the best proposal.

Table 3.8: Percentage of optimal solutions achieved by models CP2, CP3, CP4, and CP5.

	Percentage of optimal solutions			
Dataset	CP2	CP3	CP4	CP5
BZ	82	84	91	94
ZJY	26	42	68	93
EMM	75	77	82	87
CV	26	36	51	70

Percentage of optimal solutions achieved by models CP2, CP3, CP4, and CP5 across the four datasets. Source: Jiménez-Piqueras et al., 2023.

3.3.4 Performance of the algorithm

The iterative algorithm employed for the CP models is detailed in Section 3.2.5. At each iteration, a satisfaction problem is solved for a fixed number of premarshalling relocations. The algorithm iterates until a solution is found or the 3600s running time limit is reached.

The average number of iterations the algorithm performs varies depending on the dataset, ranging from 1.2 to 2.1. It conducts a maximum of 6 iterations on the CV instances and 9 on the BZ dataset. These figures are derived from the results obtained using the best model we propose, CP5.

As mentioned in section 3.2.5, we conducted several tests before settling on the proposed solution method.

One test we performed involved solving the CP5 model by providing the constraint programming solver with an optimization problem instead of a satisfaction problem, as in the iterative algorithm. The formulation used is the one described in Section 3.2.4, including its objective function. This approach

resulted in a dramatic decrease in optimal solutions, with most feasible solutions being of poor quality. These results underscore the effectiveness of the iterative algorithm compared to the solver's optimization procedure for the proposed CP models.

We also explored an alternative version of the iterative algorithm using a binary search instead of incrementing the number of stages $\bar{k}$ by one at each iteration. However, while this approach could be interesting when having similar tightness of upper and lower bounds, it proved ineffective for addressing the premarshalling problem, for which only tight lower bounds are available. In contrast, the iterative algorithm we selected leverages the tightness of the lower bound, using it as the starting point for the solution search.

3.3.5 Comparison with the state-of-the-art integer programming model and between constraint programming and mathematical programming approaches

We have implemented the state-of-the-art integer programming model IPS6 from Parreño-Torres et al., 2019, and compared the results with those of the constraint programming model we propose, CP5.

Additionally, to explore the differences between constraint programming and mathematical programming, we have built an integer programming version of CP5, denoted as CP5^{IP}, and adapted IPS6 to constraint programming, IPS6^{CP}.

To ensure a fair comparison, all the models have been solved under the same conditions using the iterative algorithm described in Section 3.2.5.

The formulation of IPS6^{CP} is the same as IPS6, with the only difference being that when solved using constraint programming, the decision variables describing the relocations are specified as integer variables instead of being relaxed as for IPS6.

The formulation of model CP5^{IP} is the same as CP5, except for the set of non-linear constraints (3.3), which are substituted with (3.37). Also, like for CP5, the objective function is not included in CP5^{IP} when using the algorithm. Constraints (3.37) indicate that the sum of the priorities assigned to the slots in the bay must be the same for any pair of consecutive stages. This condition, combined with the analogous constraints for containers that remain stationary, namely (3.15) and (3.16), ensures that the priority of the container that is relocated does not change. Hence, including constraints (3.37) in CP5^{IP} implies

that the multiplicities of the priority groups remain unchanged throughout the premarshalling process, as constraints (3.3) do in CP5.

$$\sum_{s\in\mathcal{S},t\in\mathcal{T}} x_{s,t}^{k} = \sum_{s\in\mathcal{S},t\in\mathcal{T}} x_{s,t}^{k-1} \qquad \forall k \in \mathcal{K} \tag{3.37}$$

Tables 3.9 to 3.12 compare the number of instances optimally solved by the four models, CP5, IPS6, and their integer and constraint programming versions, respectively. Also, the average running time is displayed for the instances solved by all four models.

Table 3.9: Comparison between constraint programming and mathematical programming models on the BZ dataset.

				#Optimal					Average time (s)			
$\bar{s}$	$\bar{t}$	%	#Inst.	*CP5*	*IPS6*	$CP5^{IP}$	$IPS6^{CP}$	All	*CP5*	*IPS6*	$CP5^{IP}$	$IPS6^{CP}$
3	4	50	60	**60**	**60**	**60**	**60**	60	0.3	0.3	**0.2**	0.6
3	4	70	60	**60**	**60**	**60**	**60**	60	**0.7**	1.7	1.0	25.7
3	6	50	60	**60**	**60**	**60**	**60**	60	**0.7**	4.0	1.6	68.4
3	6	70	60	**60**	**60**	**60**	47	47	**2.2**	10.3	6.6	573.5
5	4	50	60	**60**	**60**	**60**	**60**	60	**0.3**	0.5	0.4	3.6
5	4	70	60	**60**	**60**	**60**	**60**	60	**0.9**	2.0	2.9	112.9
5	6	50	60	**60**	**60**	**60**	**60**	60	**1.7**	11.3	35.3	516.8
5	6	70	60	**51**	45	38	30	30	**6.0**	13.3	102.8	1915.0
7	4	50	60	**60**	**60**	**60**	**60**	60	**0.6**	0.8	1.1	12.7
7	4	70	60	**60**	**60**	59	**60**	59	**2.3**	4.7	68.2	847.3
7	6	50	60	**60**	58	54	56	53	**3.3**	10.2	209.3	1121
7	6	70	60	**42**	39	23	30	23	**14.2**	96.4	570.5	2457.8
9	4	50	60	**60**	**60**	59	**60**	59	**1.6**	1.0	17.2	143.9
9	4	70	60	**60**	**60**	48	**60**	48	**5.6**	16.4	414.2	1509.3
9	6	50	60	**60**	**60**	41	58	41	**7.6**	29.7	355.1	2264.6
9	6	70	60	33	**38**	3	21	3	6.6	**4.4**	52.0	3600.2
Total			960	**906**	900	805	842	783	**2.5**	9.2	89.1	612.2

Results on the instances from the BZ dataset, classified by the number of stacks, $\bar{s}$, the number of tiers, $\bar{t}$, and the fill percentage (%) of the bay. #Inst. refers to the number of instances in each group. Columns #Optimal show the number of instances optimally solved by each model. In the columns on the right, the average running time is displayed for the instances solved by all the models, i.e., those indicated in column All. The best values are shown in bold. Source: Jiménez-Piqueras et al., 2023.

Table 3.10: Comparison between constraint programming and mathematical programming models on the ZJY dataset.

			#Optimal					Average time (s)			
$\bar{s}$	$\bar{t}$	#Inst.	*CP5*	*IPS6*	$CP5^{IP}$	$IPS6^{CP}$	All	*CP5*	*IPS6*	$CP5^{IP}$	$IPS6^{CP}$
6	4	20	**20**	**20**	15	10	10	**4.4**	24.6	291.8	1642.0
6	5	20	**15**	10	1	2	0	-	-	-	-
7	4	20	**19**	18	13	13	13	**10.6**	89.8	725.4	3148.3
8	4	20	**20**	17	9	14	9	**6.7**	18.1	617.7	1658.0
9	4	20	**19**	17	3	10	3	**7.0**	12.2	1352.0	1003.6
Total		100	**93**	82	41	49	35	**7.5**	46.1	627.5	2150.9

Results on the instances from the ZJY dataset, classified by the number of stacks of the bay, $\bar{s}$, and the number of tiers, $\bar{t}$. #Inst. refers to the number of instances in each group. Columns #Optimal show the number of instances optimally solved by each model. In the columns on the right, the average running time is displayed for the instances solved by all the models, i.e., those indicated in column All. The best values are shown in bold. Source: Jiménez-Piqueras et al., 2023.

Table 3.11: Comparison between constraint programming and mathematical programming models on the EMM dataset.

			#Optimal					Average time (s)			
$\bar{s}$	%	#Inst.	*CP5*	*IPS6*	$CP5^{IP}$	$IPS6^{CP}$	All	*CP5*	*IPS6*	$CP5^{IP}$	$IPS6^{CP}$
4	50	58	**58**	**58**	**58**	**58**	58	0.3	0.3	**0.2**	0.4
4	75	65	**65**	**65**	**65**	40	40	**0.7**	1.7	1.6	23.2
7	50	69	**69**	**69**	**69**	**69**	69	**1.0**	1.2	7.8	199.4
7	75	75	**55**	50	38	40	38	**7.1**	240.8	364.3	1084.4
10	50	75	**75**	**75**	72	**75**	72	4.3	**2.2**	178.9	1176.8
10	75	75	40	**43**	13	29	13	**9.4**	28.0	616.4	1492.4
Total		417	**362**	360	315	311	290	**2.8**	33.9	121.9	551.9

Results on the instances from the EMM dataset, classified by the number of stacks, $\bar{s}$, the number of tiers, $\bar{t}$, and the fill percentage (%) of the bay. #Inst. refers to the number of instances in each group. Columns #Optimal show the number of instances optimally solved by each model. In the columns on the right, the average running time is displayed for the instances solved by all the models, i.e., those indicated in column All. The best values are shown in bold. Source: Jiménez-Piqueras et al., 2023.

Table 3.12: Comparison between constraint programming and mathematical programming models on the CV dataset.

			#Optimal					Average time (s)			
$\bar{s}$	$\bar{t}$	#Inst.	*CP5*	*IPS6*	$CP5^{IP}$	$IPS6^{CP}$	All	*CP5*	*IPS6*	$CP5^{IP}$	$IPS6^{CP}$
3	5	39	**39**	38	**39**	35	34	**2.4**	33.5	8.0	953.2
4	5	40	**40**	38	39	33	33	**3.0**	65.1	30.7	1383.4
5	5	40	**39**	32	32	23	23	**6.1**	122.0	79.2	2314.2
6	5	40	**39**	26	17	13	12	**9.7**	291.9	495.0	2716.5
7	5	40	**36**	12	6	6	5	**12.6**	360.1	1693.6	3600.6
8	5	40	**30**	5	1	3	1	**17.3**	220.8	1489.5	3600.3
4	6	40	**31**	7	5	4	4	**5.5**	95.7	63.4	2731.1
5	6	40	**17**	3	3	0	0	-	-	-	-
6	6	40	**10**	1	0	0	0	-	-	-	-
7	6	40	0	0	0	0	0	-	-	-	-
Total		399	**281**	162	142	117	112	**4.8**	107.2	171.9	1753.7

Results on 400 instances from the CV dataset, classified by the number of stacks of the bay, $\bar{s}$, and the number of tiers, $\bar{t}$. #Inst. refers to the number of instances in each group, where one instance is omitted for being the initial bay layout already arranged. Columns #Optimal show the number of instances optimally solved by each model. In the columns on the right, the average running time is displayed for the instances solved by all the models, i.e., those indicated in column All. The best values are in bold. Source: Jiménez-Piqueras et al., 2023.

CP5 solves more instances than IPS6 across all datasets, especially for the most difficult one, CV, where CP5 solves 30% more instances. Regarding running times, there is a great contrast between the two models. In Tables 3.9 and 3.10, the average running time for IPS6 is over 4 and 6 times longer than CP5 for BZ and ZJY, respectively. The difference is even more pronounced for the other datasets: CP5 is over 12 and 22 times faster on average than IPS6 for EMM and CV datasets, as shown in Tables 3.11 and 3.12. These results reveal the superior performance of CP5 over the integer programming model IPS6.

In Tables 3.9 to 3.12, we observe that $CP5^{IP}$ solves significantly fewer instances than CP5: 52% fewer for ZJY and approximately 35% for CV. Also, the average running times for the instances solved by all the models are considerably higher for $CP5^{IP}$ compared to CP5. This demonstrates that the procedure followed to develop the constraint programming model is ineffective for an integer linear programming model.

While the differences in number of solved instances and running times between IPS6 and $IPS6^{CP}$ are smaller than between CP5 and $CP5^{IP}$, they are still

significant. IPS6^{CP} performs noticeably worse than IPS6. The difference in the number of solved instances ranges from 6% for the BZ dataset to 33% for the ZJY dataset. Moreover, IPS6^{CP} has the longest average time of the four models for each dataset, as we can observe in Tables 3.9 to 3.12. Compared to IPS6, IPS6^{CP} is more than 16 times slower for the EMM and CV datasets, almost 39 times slower for the BZ dataset, and over 46 times slower for the ZJY dataset. These figures reveal that the formulation designed for mathematical programming does not translate effectively to constraint programming.

3.4 Concluding remarks

The premarshalling problem has been extensively studied in the literature, yet a competitive constraint programming model had not been introduced until proposed in this chapter. This constraint programming formulation, CP5, outperforms the state-of-the-art integer programming model for this problem IPS6, Parreño-Torres et al., 2019.

The CP5 model has been built in four steps. Initially, we developed CP2, which has two groups of decision variables. Subsequently, the models CP3, CP4, and CP5 were derived by incorporating an additional set of variables and the corresponding constraints into the preceding formulation. Each group of variables added provides valuable information to the solver, and the new constraints enhance the interrelation between variables. This progressive refinement is reflected in the computational results as a marked improvement from one model to the next in the series.

The methodology employed in developing CP5 has proven to be successful in the constraint programming domain. However, this strategy seems ineffective for mathematical programming. We adapted the constraint programming formulation, CP5, into an integer linear programming model, CP5^{IP}, and obtained significantly worse results in the number of solutions found and running times. Similarly, we conducted an analogous experiment in the opposite direction: adapting IPS6 into a constraint programming model, IPS6^{CP}. The constraint programming version solves fewer instances, and the running times highly increase, thus leading to the same conclusion in both experiments. In summary, this investigation underscores that constraint programming and mathematical programming generally do not benefit from the same modeling strategies.

Chapter 4

Solving the premarshalling problem with an auxiliary bay

The Premarshalling Problem with an Auxiliary Bay (CPMP-AB) presents a novel perspective by incorporating an adjacent bay as a storage buffer for containers. This extension relaxes the overly restrictive assumption in the classical CPMP that confines the premarshalling movements solely to the bay being arranged. In practice, considering an auxiliary bay serves to expedite the premarshalling process, especially for arranging bays with a high filling level. Moreover, it enables the arrangement of bays that otherwise could not be premarshalled due to limited space for relocations within the bay.

To address the CPMP-AB, we adapted the constraint programming formulation proposed in the previous chapter for the CPMP. This chapter presents two alternative approaches to accommodate different allowance levels for relocations between bays. Our computational study demonstrates the potential of using an auxiliary bay to reduce the number of premarshalling relocations and solve instances that otherwise remain unsolved. Furthermore, we analyze the differences in the solutions provided by the alternative approaches and discuss the scenarios where each one may be more suitable.

4.1 CPMP-AB: The Container Premarshalling Problem with an Auxiliary Bay

In the port yard, container bays are organized into blocks, with each bay adjacent to one or two other bays. In the original premarshalling problem (CPMP), detailed in Section 3.1, it is assumed that the crane remains static throughout the premarshalling process, and only the crane's trolley moves along the bay to relocate containers. However, this assumption is unrealistic because the crane can travel along the block and relocate containers to nearby bays. Relocating a container within a bay is preferable to moving it to other bays because moving the entire crane, not just the trolley, is more costly. Nevertheless, some relocations between bays during premarshalling may expedite the process and allow for rearranging bays that would otherwise be infeasible.

We will denote the bay to be arranged as the main bay and another bay serving as additional space for placing containers from the main bay as the auxiliary bay. An example is shown in Figure 4.1.

Figure 4.1: An example of a main bay and an auxiliary bay.

An example of a port yard configuration where the main bay of a premarshallig process is highlighted in orange and the auxiliary bay in green. A container from the main bay moved to the auxiliary bay is highlighted in orange. Source: Elaborated by the author (2024).

The premarshalling problem with an auxiliary bay extends the original premarshalling problem by allowing containers from the main bay to be placed in the auxiliary bay and brought back to the main bay when it is convenient.

A premarshalling version in the literature where additional space is used during the arrangement process is the Container Premarshalling Problem with a Dummy Stack (CPMPDS) defined by Wang et al., 2015. Instead of considering an auxiliary bay as the CPMP-AB, the CPMPDS allows the placement of containers in the space reserved for trucks, serving as an additional stack that must be empty by the end of premarshalling. However, using an auxiliary bay offers more advantages than the dummy stack. There is more space available for placing containers and also the possibility of bringing them back to the original stacks in a different order than when they were moved to the auxiliary space. Moreover, there is no obligation for the containers to return to the original stacks. In the CPMP-AB, only if a container from the main bay is badly placed in the auxiliary bay must it be moved back to the main bay, as expressed by assumption (xii) described below.

The idea behind CPMP-AB and CPMP is the same: finding a sequence with the minimum number of relocations to entirely arrange the main bay. However, this definition is vague for the CPMP-AB because it treats relocations within the main bay and between bays equally, despite the latter being more costly. To address this issue, we propose two alternative approaches to the CPMP-AB, each offering a different allowance level for inter-bay movements. We will outline the general assumptions for the CPMP-AB and then delve into the details of each particular version.

The CPMP-AB inherits assumptions (i), and (iii) to (vii) from the CPMP, described in Section 3.1. Note that assumption (i) refers to the main bay in the CPMP-AB: only the main bay is arranged during the premarshalling process. Assumption (ii), only allowing relocations within the bay being arranged in the CPMP, is substituted with (xi) for the CPMP-AB, which considers inter-bay relocations. Additionally, specific assumptions for the CPMP-AB include (viii) to (x), and (xii).

viii) A single auxiliary bay: Only one bay serves as an auxiliary bay to the main bay.

ix) Adjacent bays: Only a bay that is adjacent to the main bay within the block can serve as an auxiliary bay.

x) No relocations within the auxiliary bay: Containers in the auxiliary bay remain stationary during the premarshalling process, and those moved to the auxiliary bay from the main bay cannot be relocated within the auxiliary bay.

xi) No container entering/leaving the ensemble of two bays: Containers from the main bay can be relocated between bays or within the main bay but never leave the ensemble of the main and auxiliary bays, and no external container is aggregated.

xii) Completely arranged auxiliary bay: It is assumed that the bay used as an auxiliary bay is entirely arranged at the beginning of the premarshalling process and must also be arranged at the end. Any container moved to the auxiliary bay that is badly placed there must return to the main bay by the end of premarshalling to maintain the arrangement of the auxiliary bay.

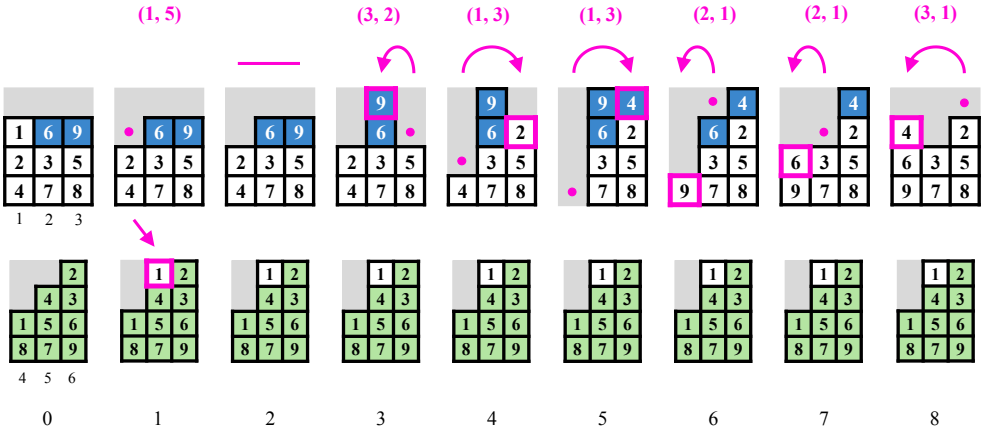
We propose two alternatives to accommodate the preference for relocations within the bay rather than relocations between bays. One approach introduces a penalty for inter-bay movements, while the other is less restrictive regarding such movements but still prioritizes those within the bay. The reason behind introducing a penalty is that some port terminals may be more conservative than others regarding inter-bay movements. The suitability of each approach in practical scenarios is discussed in Section 4.3.2, given the solutions obtained through the computational study.

In the CPMP-AB without a penalty, two objective functions are considered in lexicographic order. The primary objective is to minimize the total number of relocations, followed by the secondary objective of minimizing the relocations between bays. Therefore, the priority goal is to achieve the minimum number of relocations for arranging the bay, and within this frame, a solution with the minimum number of inter-bay relocations is chosen. With this approach, while movements within the bay are preferred over relocations between bays, any inter-bay movement that reduces the total number of relocations is performed.

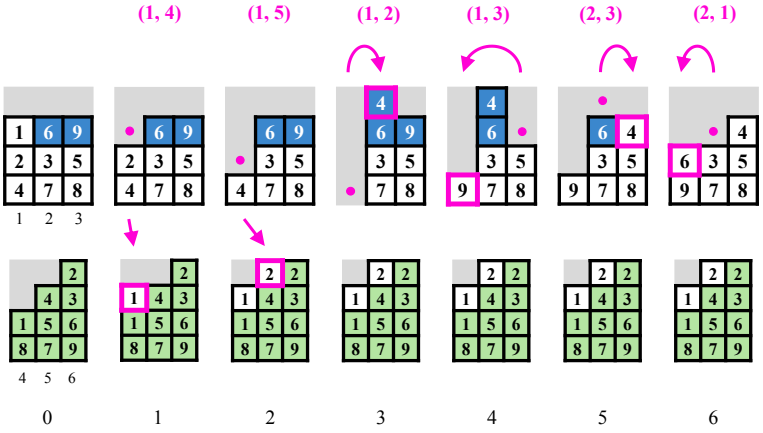
In the CPMP-AB with a penalty for inter-bay movements, each inter-bay movement is equated with two relocations within the bay. Essentially, this imposes a penalty equivalent to one relocation for each movement between bays, associating a double cost to inter-bay movements. Therefore, in this approach, the general objective function of minimizing the total number of relocations includes the penalty, resulting in the sum of the relocations within the bay and twice the number of inter-bay relocations.

Figure 4.2 illustrates optimal solutions for the CPMP-AB using the approach with a penalty (Figure 4.2a) and without a penalty (Figure 4.2b).

Figure 4.2: Examples of optimal premarshalling solutions using an auxiliary bay.



(a) An example of an optimal solution is considering a penalty for inter-bay movements.



(b) An example of an optimal solution not considering a penalty for inter-bay movements.

In each example, the main bay is shown in the first row and the auxiliary bay in the second. Each stage is identified by a number under the bay layout, and stack numbers are indicated in the initial stage. Containers are labeled with their retrieval priority, and the available space for containers is shadowed in gray. Containers originally in the auxiliary bay are highlighted in green, and badly placed containers are highlighted in blue. Relocations are highlighted in magenta. An arrow represents each relocation; a think border indicates the relocated container and a dot marks its previous location. Instead of an arrow, a red segment represents a penalty associated with an inter-bay movement. Also, the origin and destination stacks of each relocation are indicated above the bay layouts. Source: Elaborated by the author (2024).

The initial bay layouts in Figure 4.2 are extracted from the CV dataset, for which details can be found in Section 3.3.1. The initial configuration of the main bay (top left layout of each example) is the original layout from the dataset. The initial auxiliary bay configuration (bottom left corner of each example) results from premarshalling the corresponding original bay.

In Figure 4.2a, note that there is a stage with no relocations after the inter-bay movement, which reflects that the relocation between bays is counted as twice a relocation within the bay. Further details about this penalty are provided in Section 4.2.2. In contrast, in Figure 4.2b, where no penalty is imposed, the problem can be solved with a sequence of six relocations instead of seven.

Without an auxiliary bay, the instance depicted in the example in Figure 4.2 would be infeasible. If only relocations within the bay are permitted, a complete arrangement could only be achieved if an additional empty tier on top of the bay were allowed. Placing containers on an extra empty tier is feasible only if it complies with safety regulations and the crane's height permits it. In that scenario, premarshalling could be executed for this example using seven relocations within the bay and no movements between bays. This solution would also be optimal for the CPMP-AB approach with a penalty, while if no penalty were considered, the optimal solution would involve using the auxiliary bay to arrange the bay in a total of five relocations.

4.1.1 Notation

The notation for the CPMP detailed in Section 3.1.1, applies to the CPMP-AB unless otherwise specified in this section.

The set of stacks, $S := \mathcal{S}_1 \cup \mathcal{S}_2$, is extended to include the stacks corresponding to the main bay, $\mathcal{S}_1 := \{1, \ldots, \bar{s}\}$, and those of the auxiliary bay, $\mathcal{S}_2 := \{\bar{s} + 1, \ldots, 2\bar{s}\}$. The set of tiers $\mathcal{T} = \{1, \ldots, \bar{t}\}$ remains unchanged, thus $\mathcal{S} \times \mathcal{T}$ are all the slots considered in the CPMP-AB.

Given that containers initially placed in the auxiliary bay cannot be relocated during premarshalling, we distinguish between slots where no containers can be removed or placed, denoted as $\mathcal{I} \subset (\mathcal{S}_2 \times \mathcal{T})$, and those where it is possible, denoted as $\mathcal{J} := \mathcal{J}_1 \cup \mathcal{J}_2$, with $J_1 := \mathcal{S}_1 \times \mathcal{T}$ and $J_2 := (\mathcal{S}_2 \times \mathcal{T}) \setminus \mathcal{I}$. In the formulations presented in Section 4.2, we use subsets of $\mathcal{I}$ and $\mathcal{J}$ containing the top slots in these sets: $\mathcal{J}^* := \mathcal{S}_1 \times \{\bar{t}\}$ and $\mathcal{I}^* := \{(s,t) \in \mathcal{I} : (s,t+1) \in \mathcal{J}\}$.

The priorities of the containers are defined similarly to the CPMP, with an additional parameter $m_p^{\mathcal{J}}$ introduced to determine the multiplicity of each pri-

ority group $p \in \mathcal{P}^0$ in the slots where relocation is possible, $\mathcal{J}$. The sets of stages $\mathcal{K}$ and $\mathcal{K}^0$ retain the same definitions as in the CPMP.

4.2 Constraint programming models for the CPMP-AB

We introduce two constraint programming formulations, ABp and AB, to address the CPMP-AB. Model ABp integrates a penalty for relocations between bays, whereas the AB model does not. These formulations are adapted from the CP5 model for the CPMP, presented in Section 3.2.4.

The decision variables used in AB and ABp are derived from CP5 but extended to cover both the main and auxiliary bays. Constraints are also extended to accommodate the two-bay structure and to incorporate additional conditions specific to the CPMP-AB. Furthermore, constraints are reformulated using more suitable expressions for constraint programming, such as global constraints and logical functions, as in the formulations for premarshalling considering crane times detailed in section Section 5.3. The constraints (3.27) and (3.28) from CP5 are omitted in the AB and ABp formulations. These are additional constraints defined in the step of building the CP4 model for discarding infeasible solutions and removing symmetries, but they have not proven to enhance the performance of the models proposed for the CPMP-AB.

4.2.1 AB: Model for premarshalling with an auxiliary bay

In the AB formulation, the variables describing the layout of containers, namely, $x^k_{s,t}$, $\delta^k_{s,t}$, and $w^k_{s,t}$, are defined similarly to CP5 but for the extended set of stacks $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$. Likewise, the variables for identifying the relocations, $y^k_{s,t}$ and $z^k_{s,t}$, are included for the slots in the main bay and those in the auxiliary bay where relocation is possible, i.e., in every slot $(s,t) \in \mathcal{J}$. Additionally, variables $w^k_{s,t}$ are also defined for the initial stage in the AB formulation since it is required for two groups of table constraints included in this model but not in CP5.

$$x^k_{s,t} = \begin{cases} p & \text{If a container with priority } p \text{ is in slot } (s,t) \text{ at stage } k \\ 0 & \text{If slot } (s,t) \text{ is empty at stage } k \end{cases}$$

$$\forall s \in \mathcal{S},\ \forall t \in \mathcal{T},\ \forall k \in \mathcal{K}^0$$

$$\delta_{s,t}^k = \begin{cases} 1 & \text{If there is a container in slot } (s,t) \text{ at stage } k \\ 0 & \text{If slot } (s,t) \text{ is empty at stage } k \end{cases}$$

$$\forall s \in \mathcal{S},\, \forall t \in \mathcal{T},\, \forall k \in \mathcal{K}^0$$

$$w_{s,t}^k = \begin{cases} 1 & \text{If a badly placed container is in } (s,t) \text{ during stage } k \\ 0 & \text{Otherwise} \end{cases}$$

$$\forall s \in \mathcal{S},\, t \in \mathcal{T},\, k \in \mathcal{K}^0$$

$$y_{s,t}^k = \begin{cases} 1 & \text{If a container is moved to slot } (s,t) \text{ during stage } k \\ 0 & \text{Otherwise} \end{cases}$$

$$\forall (s,t) \in \mathcal{J},\, \forall k \in \mathcal{K}$$

$$z_{s,t}^k = \begin{cases} 1 & \text{If a container is removed from slot } (s,t) \text{ during stage } k \\ 0 & \text{Otherwise} \end{cases}$$

$$\forall (s,t) \in \mathcal{J},\, \forall k \in \mathcal{K}$$

The objective of the CPMP-AB is to minimize the number of relocations required to arrange the main bay, which can be expressed as (4.1). Additionally, the AB approach considers a second objective: minimizing the number of relocations between bays, corresponding to (4.2). The AB objectives are minimized in lexicographical order, with (4.1) being the primary objective and (4.2) the secondary one.

$$\min \sum_{(s,t)\in\mathcal{J},\, k\in\mathcal{K}} y_{s,t}^k \tag{4.1}$$

$$\min \sum_{(s,t)\in\mathcal{J}_2,\, k\in\mathcal{K}} \left(y_{s,t}^k + z_{s,t}^k\right) \tag{4.2}$$

The initial layouts of the main and auxiliary bays are determined by variables $x_{s,t}^0$, $\delta_{s,t}^0$, and $w_{s,t}^0$, specified by parameters $\alpha_{s,t}^x$, $\alpha_{s,t}^\delta$, and $\alpha_{s,t}^w$, through constraints (4.3), (4.4), and (4.5), respectively. Containers initially placed in the auxiliary bay cannot be moved during premarshalling. Therefore, the bay layout variables corresponding to these slots are fixed to their initial values for all premarshalling stages by (4.6) to (4.8). Furthermore, there are never badly placed containers in the first tier because they cannot be stacked on top of

another badly placed container or a container with an earlier retrieval time. This fact is indicated by constraints (4.9). These constraints are analogous to those used in CP5 for determining the initial layout and assigning the value 0 to $w^k_{s,1}$, but are rewritten here for clarity.

$$x^0_{s,t} = \alpha^x_{s,t} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T} \tag{4.3}$$
$$\delta^0_{s,t} = \alpha^\delta_{s,t} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T} \tag{4.4}$$
$$w^0_{s,t} = \alpha^w_{s,t} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T} \tag{4.5}$$
$$x^k_{s,t} = \alpha^x_{s,t} \qquad \forall (s,t) \in \mathcal{I},\, k \in \mathcal{K} \tag{4.6}$$
$$\delta^k_{s,t} = \alpha^\delta_{s,t} \qquad \forall (s,t) \in \mathcal{I},\, k \in \mathcal{K} \tag{4.7}$$
$$w^k_{s,t} = \alpha^w_{s,t} \qquad \forall (s,t) \in \mathcal{I},\, k \in \mathcal{K} \tag{4.8}$$
$$w^k_{s,1} = 0 \qquad \forall s \in \mathcal{S},\, k \in \mathcal{K} \tag{4.9}$$

The values of the variables representing the bay layout in the initially occupied slots of the auxiliary bay, i.e., for the slots in the set $\mathcal{I}$, are fixed by (4.6) to (4.8). The rest of the constraints and variables specifically apply to the remaining slots, those in set $\mathcal{J}$, where containers can be placed and removed and are presented below. Table 4.1 indicates the equivalence between these constraints in the AB model and those in CP5 from which they are derived.

Table 4.1: Constraints from model CP5 substituted in model AB.

AB	CP5
(4.10)	(3.3)
(4.11)	(3.13), (3.14)
(4.12)	(3.15), (3.16)
(4.13)	(3.29)
(4.14)	(3.33)
(4.15)	(3.34)
(4.16)	(3.8)
(4.17)	(3.22)
(4.19)	(3.21)
(4.20), (4.21)	(3.23), (3.24), (3.20), (3.25), (3.26), (3.30), (3.31)

The left column shows the constraints of model AB replacing the corresponding constraints from CP5 indicated in the right column. Source: Elaborated by the author (2024).

$$Count\big(x^k_{s,t} : (s,t) \in \mathcal{J},\, x^k_{s,t} = p\big) = m^{\mathcal{J}}_p \qquad \forall p \in \mathcal{P}^0,\, k \in \mathcal{K} \tag{4.10}$$

$$\big(x^k_{s,t} > 0\big) = \delta^k_{s,t} \qquad \forall (s,t) \in \mathcal{J},\, k \in \mathcal{K} \tag{4.11}$$

$$\big(x^k_{s,t} == x^{k-1}_{s,t}\big) = \big(\delta^k_{s,t} == \delta^{k-1}_{s,t}\big) \qquad \forall (s,t) \in \mathcal{J},\, k \in \mathcal{K} \tag{4.12}$$

$$\big(x^k_{s,t+1} > x^k_{s,t}\big) \;\leq\; w^k_{s,t+1} \qquad \forall (s,t) \in (\mathcal{J} \setminus \mathcal{J}^*) \cup \mathcal{I}^*,\; k \in \mathcal{K} \tag{4.13}$$

$$w^k_{s,t+1} \;\leq\; w^k_{s,t} + \big(x^k_{s,t} < x^k_{s,t+1}\big) \qquad \forall (s,t) \in (\mathcal{J} \setminus \mathcal{J}^*) \cup \mathcal{I}^*,\; k \in \mathcal{K} \tag{4.14}$$

$$\sum_{(s,t)\in\mathcal{J}} w^k_{s,t} + k \leq \bar{k} \qquad \forall k \in \mathcal{K} \tag{4.15}$$

$$\sum_{(s,t)\in\mathcal{J}} y^k_{s,t} \;\leq\; 1 \qquad \forall k \in \mathcal{K} \tag{4.16}$$

$$\sum_{(s,t)\in\mathcal{J}} z^k_{s,t} \;\leq\; 1 \qquad \forall k \in \mathcal{K} \tag{4.17}$$

$$\sum_{(s,t)\in\mathcal{J}_2} \big(z^k_{s,t} + y^k_{s,t}\big) \;\leq\; 1 \qquad \forall k \in \mathcal{K} \setminus \{\bar{k}\} \tag{4.18}$$

$$y^k_{s,t} \;\leq\; \delta^{k+1}_{s,t} \qquad \forall (s,t) \in \mathcal{J},\; k \in \mathcal{K} \setminus \{\bar{k}\} \tag{4.19}$$

$$\begin{aligned} &AllowedAssignments\big(A,\\ &\delta^{k-1}_{s,t}, \delta^{k-1}_{s,t+1}, \delta^k_{s,t}, \delta^k_{s,t+1}, z^k_{s,t}, z^k_{s,t+1},\\ &y^k_{s,t}, y^k_{s,t+1}, w^{k-1}_{s,t}, w^{k-1}_{s,t+1}, w^k_{s,t}, w^k_{s,t+1}\big) \end{aligned} \qquad \forall (s,t) \in \mathcal{J} \setminus \mathcal{J}^*,\, k \in \mathcal{K} \tag{4.20}$$

$$\begin{aligned} &AllowedAssignments\big(B,\\ &\delta^{k-1}_{s,t}, \delta^{k-1}_{s,t+1}, \delta^k_{s,t}, \delta^k_{s,t+1}, z^k_{s,t+1},\\ &y^k_{s,t+1}, w^{k-1}_{s,t}, w^{k-1}_{s,t+1}, w^k_{s,t}, w^k_{s,t+1}\big) \end{aligned} \qquad \forall (s,t) \in \mathcal{I}^*,\, k \in \mathcal{K} \tag{4.21}$$

The condition that the number of containers in each priority group must remain constant throughout the premarshalling process is ensured in AB by the Count constraints (4.10). The difference between these constraints and (3.3) from CP5 is that in AB, the count of the containers within each priority group is conducted over the set of slots $\mathcal{J}$.

The constraints linking variables x and δ correspond to (4.11) and (4.12) in AB. Constraints (4.11) express that positive values of variables x correspond to occupied slots, i.e., to value 1 in the case of variables δ, and both are assigned a value of 0 if the slot is empty. Constraints (4.12) ensure the priority associated with a slot occupied by a container that is not relocated remains the same, or it continues to be 0 if a slot remains empty.

If a container placed in slot (s, t) must be retrieved before the container in $(s, t + 1)$, the container in $(s, t + 1)$ is marked as badly placed, as expressed by constraints (4.13). Also, a container on top of a badly placed container is considered badly placed, a condition ensured by constraints (4.20) and (4.21) in AB and detailed in subsequent paragraphs. The condition that a badly placed container must be on top of a container with an earlier retrieval time or on top of a badly placed container is expressed through (4.14). Constraints (4.15) impose an upper bound on the number of badly placed containers using variables w, similarly to (3.34) in CP5.

The limit of at most one relocation per stage is imposed by (4.16) and (4.17) in AB for the extended set of slots. The group of constraints (4.19) defines the transitive move avoidance rule, ensuring that the same container is not moved in consecutive stages. An additional set of constraints, (4.18), is included in AB to prevent containers from the main bay placed in the auxiliary bay from being relocated within the auxiliary bay.

The table constraints (4.20) and (4.21) include in the AB formulation the relationship between the bay configuration variables and movement variables, and some conditions regarding badly placed containers. These constraints replace all the groups of constraints from CP5 indicated in Table 4.1. The global constraint Allowed Assignments defines the feasible combinations of values for the decision variables specified as arguments and imposes this ensemble of variables to take one of these combinations. Matrices A and B define the allowed combinations in (4.20) and (4.21). Table constraints (4.20) apply to all slots where containers can be placed and removed, excluding those on top of containers that are not subject to relocation during premarshalling. Such slots require separate table constraints (4.21) because variables y and z are not defined for slots occupied by containers that cannot be moved.

The general idea behind the table constraints (4.20) and (4.21) is that the bay layout is only altered when a movement is performed, and it must change accordingly. Specifically, only a container on the top of a stack can be relocated, and it must be placed on top of a container or the ground without empty slots in between. Moreover, a container's transition between badly placed and well-

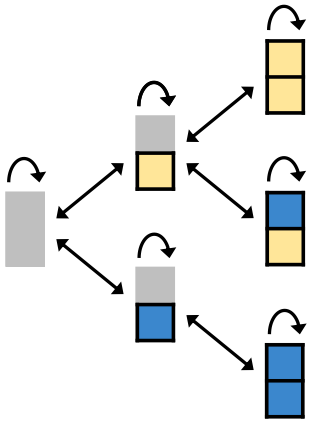
placed states occurs exclusively during the relocation of this container and depends on the container below in the destination stack.

Only specific configurations or states are viable for a given pair of consecutive slots within the same stack. These states, along with the permissible transitions between them, are represented in Figure 4.3 and elaborated upon below for a generic pair of slots (s,t) and $(s,t+1)$.

- Both slots are empty. In this scenario, there are three possible transitions for the next stage: both slots remain empty, slot (s,t) becomes occupied by a well-placed container, or it becomes occupied by a badly placed container.
- Slot (s,t) is occupied, and $(s,t+1)$ is empty. This case has two states depending on the well or badly placed condition of the container in (s,t). The possible transitions are: the same state is maintained, slot (s,t) becomes empty, or slot $(s,t+1)$ becomes occupied. If $(s,t+1)$ becomes occupied, the container placed there would be badly placed if the one at (s,t) is badly placed; otherwise, the container at $(s,t+1)$ may be either well placed or badly placed depending on container priorities.
- Both slots are occupied. There are three possible configurations: both containers are well placed, both are badly placed, or the one below is well placed and the other is badly placed. This state may remain the same in the next stage, or $(s,t+1)$ may become empty.

The arrows in Figure 4.3 indicate a total of sixteen allowed transitions between six possible states for a pair of containers in consecutive slots (s,t) and $(s,t+1)$. Accordingly, matrix A in (4.20) contains sixteen allowed combinations for the decision variables involved in these constraints. In contrast, matrix B associated with (4.21) comprises only eleven combinations. This is because, in this case, the bottom slot (s,t) belongs to the set $\mathcal{I}^*$, which includes slots in the auxiliary bay initially occupied by containers that cannot be relocated. Consequently, the combinations in B always assign the value 1 to variables $\delta_{s,t}^{k-1}$ and $\delta_{s,t}^{k}$, since (s,t) is always occupied in this case. These combinations correspond to the eleven transitions between states that do not involve the state where both slots are empty.

Figure 4.3: Possible states for two consecutive slots and transitions between them.



The six possible states for two consecutive slots (s, t) and $(s, t+1)$. The sixteen feasible transitions between states from one stage k to the next $k+1$ are indicated by arrows, with curved arrows representing the option of maintaining the same configuration from $k-1$ to k. Well-placed containers are highlighted in light yellow, badly placed containers are highlighted in blue, and empty slots are in gray. Source: Elaborated by the author (2024) based on a figure from Jiménez-Piqueras et al., 2024.

4.2.2 ABp: Alternative model with a penalty for inter-bay relocations

We introduce the ABp model, an alternative approach considering a penalty for relocations between bays. This penalty aims to reduce the frequency of inter-bay movements compared to the AB model. The formulation ABp contains the same variables and constraints as AB, except for one modified group of constraints and two additional sets that are defined to incorporate the penalty. The solution method described in Section 4.2.3 requires the penalty to be defined through constraints in ABp, as this method uses the ABp model to address satisfaction problems and omits its objective function.

The objective function of ABp, (4.22), minimizes the number of relocations required to arrange the bay, with inter-bay movements incurring a double cost compared to relocations within the main bay. This objective function uses variables $y^k_{s,t}$ across the set of slots $\mathcal{J}$ to count all relocations and includes an additional term using variables $z^k_{s,t}$ and $y^k_{s,t}$ across the set $\mathcal{J}_2$ to ensure relocations originating or terminating in the auxiliary bay are counted twice.

$$\min \sum_{k \in \mathcal{K}} \left(\sum_{(s,t) \in \mathcal{J}} y^k_{s,t} + \sum_{(s,t) \in \mathcal{J}_2} \left(y^k_{s,t} + z^k_{s,t} \right) \right) \tag{4.22}$$

The constraints introduced in ABp to include the penalty for inter-bay movements are (4.23) to (4.25).

$$\sum_{(s,t) \in \mathcal{J}} w^k_{s,t} + \sum_{(s,t) \in \mathcal{J}_2} w^k_{s,t} + k \leq \bar{k} \qquad \forall k \in \mathcal{K} \tag{4.23}$$

$$\sum_{(s,t) \in \mathcal{J}} \left(y^{k+1}_{s,t} + z^{k+1}_{s,t} \right) \leq 2 \cdot \Big(\sum_{(s,t) \in \mathcal{J}_2} \delta^{k-1}_{s,t} == \sum_{(s,t) \in \mathcal{J}_2} \delta^k_{s,t} \Big) \qquad \forall k \in \mathcal{K} \setminus \{\bar{k}\} \tag{4.24}$$

$$\sum_{(s,t) \in \mathcal{J}_2} \left(z^{\bar{k}}_{s,t} + y^{\bar{k}}_{s,t} \right) = 0 \tag{4.25}$$

The penalty for inter-bay movements is incorporated into the formulation by enforcing a mandatory stage without relocations immediately after a movement between bays. This structure ensures that an inter-bay relocation is counted as two relocations within the bay. Constraints (4.23) replace (4.15) from AB. (4.23) contain an additional term to count each movement between bays twice and thus include the penalties in the count of movements performed until a certain stage. Consequently, the number of stages available for relocations after a given stage is calculated as the total number of stages minus the sum of relocations already performed and the corresponding penalties. The penalty is the sole condition imposing a stage without relocations, and it is enforced by constraints (4.24). These constraints express that if the number of containers in the auxiliary bay changes in stage k with respect to $k-1$, no relocation is allowed in stage $k+1$. In the final stage, constraint (4.25) prohibits inter-bay relocations to prevent circumventing the penalty.

4.2.3 Solution method

The discussion provided in Section 3.2.5 concerning the best solving method for the CPMP using the constraint programming models proposed is equally applicable to the CPMP-AB. A tight upper bound is unavailable, and the iterative algorithm has proven to be more effective than providing the solver with a fixed number of stages and an objective function to minimize. For these

reasons, we employ an iterative algorithm to search for optimal solutions to the CPMP-AB.

The algorithm described for the CPMP could be applied to the constraint programming models proposed for the CPMP-AB. However, a lower bound for the optimal objective value of the CPMP-AB would be required, and deriving such a lower bound for the new problem is complex. Nonetheless, the existing lower bound for the CPMP can be used with a slight modification in the algorithm, as detailed below.

The algorithm proposed for AB and ABp begins with an initial iteration where the number of stages $\bar{k}$ is set to the lower bound for the CPMP. The purpose of this iteration is to determine whether the lower bound for the CPMP is also a lower bound for the CPMP-AB or it is actually an upper bound. A satisfaction problem is solved, providing the solver with the formulation without including an objective function. Additionally, when using AB, constraints (4.16) and (4.17) are substituted with (4.26) and (4.27), imposing exactly one relocation per stage, as done for models CP2 to CP5. However, the same adjustment is not applied to ABp because it is necessary to allow stages without relocations to account for the penalty for inter-bay movements.

$$\sum_{(s,t)\in\mathcal{J}} y^k_{s,t} = 1 \qquad \forall k \in \mathcal{K} \tag{4.26}$$

$$\sum_{(s,t)\in\mathcal{J}} z^k_{s,t} = 1 \qquad \forall k \in \mathcal{K} \tag{4.27}$$

If the problem proposed in the first iteration is infeasible, a new iteration is performed using the same formulation but increasing the number of stages by one. The algorithm continues iterating in this manner as long as the problem remains infeasible. If a solution is eventually found, it is optimal for the total number of relocations. Therefore, it would be an optimal solution if using the ABp model. However, if using AB, the second objective should be minimized in a subsequent phase of the algorithm.

If, on the contrary, a solution is already found in the first iteration, the value of $\bar{k}$ is an upper bound for the CPMP-AB, and the existence of better solutions must be checked. Hence, a new iteration is performed, decreasing the number of stages by one. If a solution is found, the search for better solutions must continue, and the process is repeated. If an infeasible problem is encountered, the last solution found is proven optimal for the total number of relocations.

Consequently, the solution is optimal if using ABp, or it is optimal for the first objective of AB, and the second objective must be addressed next.

If solving the CPMP-AB with the AB model, an additional phase is conducted to minimize the second objective. The optimal value for the first objective, which is the total number of relocations, is set as the number of stages $\bar{k}$. A satisfaction problem is solved, including in the formulation constraint (4.28), which imposes an upper limit $\bar{r}$ to the second objective. $\bar{r}$ is first set to the number of inter-bay relocations performed in the optimal solution for the first objective minus one. If a solution is found where some inter-bay relocations are conducted, $\bar{r}$ is decremented by one, and a new iteration is performed. If a solution without inter-bay relocations is obtained, it is optimal. Otherwise, encountering an infeasible problem confirms that the best solution yielded thus far is indeed optimal.

$$\sum_{(s,t)\in\mathcal{J}_2,\, k\in\mathcal{K}} \left(y^k_{s,t} + z^k_{s,t}\right) \leq \bar{r} \tag{4.28}$$

4.3 Computational experiments

In this section, we examine the results derived from solving the CPMP-AB and compare them to those of the CPMP. Additionally, we delve into the difference between the solutions generated by the proposed model incorporating a penalty for movements between bays, ABp, and the model not considering penalties, AB.

The instances used in the experiments are sourced from the CV dataset. We chose this dataset because it is the most complex among those considered in Section 3.3.1. In these instances, all the stacks in a bay are filled with the same number of containers, allowing us to test two occupancy levels by considering either two top empty tiers, as typically done in the literature, and only one empty tier. Specifically, with two empty tiers, the instances used in these experiments have fill percentages of 60% and 67%. In contrast, with one empty tier, they have fill percentages of 75% and 80%, which are more challenging to solve. We consider 320 instances of varying dimensions: some instances have 3 tiers occupied and 3 to 8 stacks, while others have 4 tiers occupied and 4 or 5 stacks.

The instances in the literature consist of a single bay, but an additional bay is required to solve the CPMP-AB. We designate the bay in the original CV

instances as the main bay and assign an auxiliary bay to each one. The auxiliary bay is assumed to be completely arranged, as indicated in Section 4.1. In these experiments, the arrangement of the auxiliary bay is obtained by solving the CPMP on the original instances, and the resulting arranged configurations are randomly paired with the main bays.

The algorithms employed in these experiments require an initial value for the number of premarshalling stages $\bar{k}$, as specified in Section 4.2.3. We adopt the lower bound for the CPMP provided by Tanaka et al., 2019.

Further details on the CV dataset and technical specifications regarding the experiments are provided in Section 3.3.1.

4.3.1 Performance of the proposed models

Table 4.2 presents the total number of feasible and optimal solutions obtained by the approaches with and without a penalty for movements between bays, considering instances from the CV dataset with one or two top empty tiers, as specified in the table. Additionally, the average running times for the instances solved in all cases are displayed.

Table 4.2: Performance of models AB and ABp.

$\bar{c}$	$\bar{s}$	$\bar{t}$		#Optimal				#Feasible					Average CPU (s)			
		2e	1e	2e		1e		2e		1e		All	2e		1e	
				ABp	AB	ABp	AB	ABp	AB	ABp	AB		ABp	AB	ABp	AB
9	3	5	4	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	40	6	**5**	4	**3**
12	4	5	4	**40**	**40**	**40**	**40**	**40**	**40**	**40**	**40**	40	34	**27**	42	**16**
15	5	5	4	**37**	**37**	**38**	37	38	**39**	38	**39**	36	**224**	285	**87**	237
16	4	6	5	17	**21**	20	**25**	17	**36**	20	**35**	12	624	**596**	153	**48**
18	6	5	4	**32**	30	**35**	31	34	**38**	35	**36**	31	**272**	547	**313**	466
20	5	6	5	6	**8**	7	**11**	6	**36**	7	**30**	5	529	**340**	629	**267**
21	7	5	4	**24**	23	**26**	18	26	**36**	27	**32**	18	652	**572**	**513**	1067
24	8	5	4	**18**	16	**14**	8	21	**37**	14	**29**	10	**251**	885	**506**	1130
Total				214	**215**	**220**	210	222	**302**	221	**281**	192	**221**	294	**177**	292

Results on 320 instances from the CV dataset, grouped into eight categories of 40 instances, ordered by the number of containers in the bay, $\bar{c}$. The bays in each category have $\bar{s}$ stacks and $\bar{t}$ tiers, including 2 or 1 extra empty tiers denoted by 2e and 1e. The number of optimal and feasible solutions obtained are displayed in columns #Optimal and #Feasible, respectively. Columns Average CPU (s) present the average running time for the instances solved by all the models, i.e., the instances indicated in column All. The best values are shown in bold. Source: Elaborated by the author (2024).

Although the number of optimal solutions is similar across all the cases studied, a notable difference exists in the total number of feasible solutions found. AB yields significantly more feasible solutions than ABp, which could be attributed to the less restrictive nature of the model without penalties, which facilitates finding solutions. However, for the same reason, proving optimality may be more challenging in the AB approach. This challenge, coupled with the additional phase in the algorithm for minimizing the relocations between bays, could account for the longer total average running times of AB compared to ABp. Nonetheless, the difference in average time varies considerably across the categories of instances, with AB obtaining shorter average times than ABp in numerous categories.

4.3.2 Differences in the solutions when considering or not an auxiliary bay, and when including or not a penalty for inter-bay relocations

The results presented in Table 4.3 enable a comparison between the solutions for the CPMP-AB and those of the original premarshalling without an auxiliary bay, as well as between the approaches proposed for the CPMP-AB considering or not a penalty for inter-bay movements.

As expected, the model without penalties, AB, produces more inter-bay relocations than ABp. Nevertheless, the average percentage of relocations between bays relative to the total number of relocations in a solution does not exceed 23%, which is still relatively low.

The percentage of solved instances experiencing a reduction in the number of relocations when considering an auxiliary bay compared to an optimal CPMP solution is significantly higher using AB than ABp: 92% versus 40% for the instances considering two extra empty tiers, and 79% versus 56% for those with one extra empty tier. Furthermore, the reduction in relocations is more pronounced for AB: an average reduction of 26% for AB versus 19% for ABp on instances with two additional empty tiers, and 58% versus 56% on instances with only one empty tier. The latter comparison shows that the reduction percentage is similar for ABp and AB. This similarity arises because most instances in the categories with 9 and 16 containers have a 100% reduction percentage. This percentage is assigned to instances that are infeasible for the original premarshalling problem when only one extra tier is considered, but they can be solved using an auxiliary bay.

The results demonstrate that using an auxiliary bay can significantly reduce the number of premarshalling relocations, especially for bays with high fill percentages. Moreover, infeasible instances for the CPMP can be solved using an auxiliary bay.

The analysis of the differences in the solutions yielded by the proposed approaches for the CPMP-AB with and without a penalty for movements between bays allows us to specify which one is more advantageous depending on the requirements in practice. If achieving solutions with an average percentage of inter-bay movements of less than 10% is desired, even if it entails reducing the number of premarshalling movements in only around half of the instances solved, then the ABp model is appropriate. Conversely, suppose one can accept an average percentage of inter-bay movements around 20%. In that case, the AB model is preferred because using an auxiliary bay without penalty reduces the number of relocations in most instances, and the reduction is more significant than with ABp.

Table 4.3: Comparison of the solutions yielded by models AB and ABp.

$\bar{c}$	$\bar{s}$	$\bar{t}$		Inter-bay relocations (%)				Inst. with reduction (%)				#I	Reloc. reduction (%)			
		2e	1e	2e		1e		2e		1e			2e		1e	
				ABp	AB	ABp	AB	ABp	AB	ABp	AB	All	ABp	AB	ABp	AB
9	3	5	4	16	**35**	21	**31**	80	**93**	93	**95**	30	24	**31**	90	**90**
12	4	5	4	8	**28**	12	**21**	48	**95**	83	**93**	17	18	**25**	21	**26**
15	5	5	4	4	**20**	5	**16**	32	**87**	42	**82**	10	12	**19**	15	**18**
16	4	6	5	9	**24**	13	**17**	76	**97**	**100**	**100**	8	16	**24**	85	**87**
18	6	5	4	1	**16**	1	**13**	12	**87**	17	**75**	2	9	**13**	12	**15**
20	5	6	5	12	**16**	11	**19**	67	**100**	**100**	60	4	14	**16**	14	**21**
21	7	5	4	0	**15**	0	**17**	4	**92**	11	**69**	0	-	-	-	-
24	8	5	4	2	**13**	1	**13**	14	**89**	7	**41**	0	-	-	-	-
Total				7	**23**	9	**20**	40	**92**	56	**79**	71	19	**26**	56	**58**

Results on 320 instances from the CV dataset, grouped into eight categories of 40 instances, ordered by the number of containers in the bay, $\bar{c}$. The bays in each category have $\bar{s}$ stacks and $\bar{t}$ tiers, including 2 or 1 extra empty tiers denoted by 2e and 1e. The average percentage of inter-bay relocations relative to the total number of relocations in a solution is displayed in the Inter-bay relocations (%) columns. The percentage of solved instances with a reduction in relocations compared to an optimal CPMP solution is shown in columns Inst. with reduction (%). The average percentage reduction is provided in the Reloc. reduction (%) columns, for the instances solved in all cases, i.e., the instances indicated in column #I. The best values are shown in bold. Source: Elaborated by the author (2024).

4.4 Concluding remarks

The original formulation of the premarshalling problem assumes all relocations for arranging the bay must be performed within it. However, this assumption is not realistic because during premarshalling at port terminals, some containers are relocated to another bay if it facilitates the arrangement operations. The consideration of an auxiliary bay may reduce the number of premarshalling relocations and allow for arranging bays with little space for relocations that otherwise could not be arranged.

We introduce the Premarshalling Problem with an Auxiliary Bay (CPMP-AB) to accommodate the use of an auxiliary bay for placing containers from the bay being arranged. The objective of this novel problem is to minimize the number of premarshalling relocations, as in the CPMP, but additionally, how inter-bay relocations are addressed must be specified. We propose two different approaches for regulating the movements between bays, providing flexibility to adapt to particular regulations of port terminals. A constraint programming model and the corresponding solving method are presented for each approach.

Our proposed constraint programming model AB minimizes two objectives in lexicographic order. Firstly, it seeks to minimize the number of relocations required to arrange the bay. Secondly, it minimizes inter-bay movements, keeping the total number of relocations fixed. This approach ensures that relocations within the main bay are favored over movements between bays yielding the same total number of relocations and, at the same time, it allows every relocation between bays capable of reducing the number of total relocations. This model maximizes efficiency by prioritizing movements within the bay, which are less costly, and leveraging the advantages of an auxiliary bay.

The alternative constraint programming model ABp, imposes a stronger limitation on inter-bay movements by assigning a double cost to relocations between bays compared to movements within the bay. Hence, the objective is to minimize the total number of relocations, considering the penalty for each inter-bay movement as if it were an additional relocation. In essence, a relocation is favored in a solution only if, even when counted as two movements within the bay, it reduces the total number of relocations. This approach is designed for specific scenarios where the amount of inter-bay relocations is required to be lower than provided by the AB model.

The computational study demonstrates the advantages of incorporating an auxiliary bay. It significantly reduces the number of relocations compared to CPMP solutions, especially for bays where three-quarters of the slots are

occupied. Moreover, instances that are infeasible for the CPMP due to limited space for relocations become solvable by introducing an auxiliary bay.

The experiments also illustrate the versatility of the CPMP-AB formulation thanks to the alternative approaches proposed, which vary in the allowance level for inter-bay relocations. The ABp model is suitable when only around 10% of relocations are permitted between bays. Conversely, the AB model is preferred if a higher percentage is accepted, specifically around 20%. This preference arises because the AB model offers a reduction in the number of relocations in more cases than the ABp model, and this reduction is also more significant.

Chapter 5

Premarshalling problems considering crane times

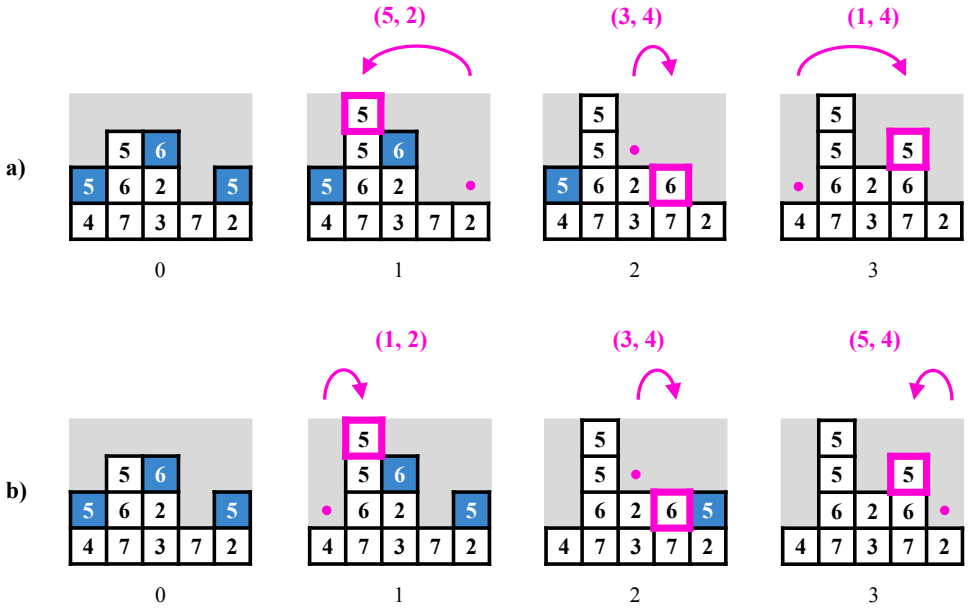
At port terminals, cranes are available for premarshalling for a limited duration. However, existing literature does not offer a solution when this time is insufficient to complete the bay arrangement. To address this issue, we define the Container Premarshalling Problem under Limited Crane Time (CPMP-LCT), seeking the best solution within the available time. This novel problem involves measuring the time taken by relocations, which is a much more realistic efficiency indicator for a solution than the number of relocations considered in the original approach. For this reason, the Container Premarshalling Problem with Crane Time Minimization Objective (CPMPCT) was proposed in recent literature and is addressed in this chapter as a foundation for the CPMP-LCT.

We present a constraint programming formulation for the CPMPCT that improves the performance of the state-of-the-art integer programming model for this problem and serves as a sound basis for developing our model for the CPMP-LCT. Computational experiments show the efficiency of CPMP-LCT solutions and the possibility of the proposed exact model functioning as a heuristic, yielding good-quality solutions within short running times. Furthermore, we propose an alternative formulation tailored for scenarios where all container priorities in the bay are distinct.

5.1 CPMPCT: The Container Premarshalling Problem with Crane Time Minimization Objective

The objective function of the classical CPMP aims to minimize the number of container relocations required to arrange the bay. However, this objective may not yield the most efficient solutions in practice. For instance, while one would choose to relocate a container to an adjacent stack over a distant one if the resulting configurations are similar, the CPMP neglects this preference since the number of relocations does not differentiate between these movements. By contrast, the time taken by relocations does catch this difference, as illustrated in Figure 5.1.

Figure 5.1: An example of two CPMP optimal solutions with different total crane times.



The solutions incur a total crane time of a) 315 and b) 297 seconds, using the same number of relocations and yielding the same final layout. The numbers below the layouts indicate the stages of the premarshalling solution. The relocation performed at each stage is indicated in magenta by an arrow and a pair (s, r) where s and r are the origin and destination stacks, respectively. The relocated container is highlighted with a thick border, and a dot indicates its origin slot. Badly placed containers are highlighted in blue. Source: Elaborated by the author (2024) based on a figure from Parreño-Torres et al., 2020.

Relocating between close stacks requires less time than moving a container across the bay, as illustrated in Figure 5.1. Although the two premarshalling solutions shown in this figure result in the same final layout, these sequences of movements take different total crane times. The relocations in Figure 5.1b involve shorter distances between stacks for the first and third relocations compared to those in Figure 5.1a, and thus take shorter times. Consequently, while the two solutions depicted in Figure 5.1 are optimal for the CPMP, Figure 5.1a would not be an optimal solution if the objective were to minimize the total crane time instead of the number of relocations.

Figure 5.1 exemplifies that the total time required by a premarshalling sequence depends on the distance between the stacks involved in a relocation. In addition, the tier where a container is placed also influences relocation time. Furthermore, fewer relocations do not always guarantee shorter total crane time. All these facts are a significant motivation for considering crane times instead of the number of relocations as an efficiency measure for premarshalling solutions and led Parreño-Torres et al., 2020, to define the Container Premarshalling Problem with Crane Time Minimization Objective (CPMPCT). This problem inherits the assumptions from the CPMP described in Section 3.1, replacing (vi), which assumes uniform movement costs, by (xiii).

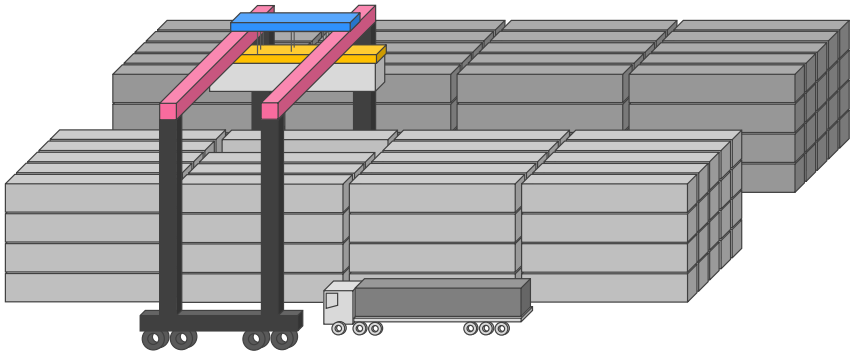
xiii) Crane time costs: Each relocation incurs a cost that is equivalent to the crane's time to perform it. This time depends on the origin and destination slots of the relocation.

The distinction between the CPMP and CPMPCT lies in the objective function. While the CPMP aims to minimize the number of relocations to obtain a bay layout without badly placed containers, the CPMPCT seeks the same kind of arrangement but minimizes the crane time instead of the number of relocations.

5.1.1 Crane time specifications and notation

This study uses the data and calculations for a rubber tire gantry crane described by Parreño-Torres et al., 2020. Container relocations require horizontal movements of the crane's trolley along the bridge girder and vertical movements of the crane's spreader originating and ending at the bridge girder. These crane elements are illustrated in Figure 5.2.

Figure 5.2: An example of a port yard configuration with a rubber tire gantry crane.



The crane's trolley is highlighted in blue, the spreader in yellow, and the bridge girder in pink. Source: Elaborated by the author (2024).

A container relocation involves the following crane movements. Suppose the crane's trolley is positioned above stack s_1, and a container needs to be relocated from slot (s_2, t_2) to slot (s_3, t_3). The sequence of movements for this relocation is as follows:

1. The trolley travels horizontally from stack s_1 to s_2.
2. At stack s_2, the spreader descends to just above slot (s_2, t_2).
3. The container in slot (s_2, t_2) is twistlocked and hoisted to the bridge girder.
4. The loaded trolley moves horizontally to stack s_3.
5. The container is lowered into slot (s_3, t_3), and the spreader returns unloaded to the bridge girder.

The total time taken for the relocation is the sum of the following times using the notation described below:

$$h^0_{s_1,s_2} + v_{t_2} + l_{t_2} + h^1_{s_2,s_3} + v_{t_3}$$

$h^0_{s,r}$: Horizontal travel time of the trolley from stack s to r with no load.

$h^1_{s,r}$: Horizontal travel time of the trolley from stack s to r with load.

v_t: Vertical travel time of the spreader from the bridge girder to tier t and coming back to the girder, either to place a container at tier t or to hoist it.

l_t: Twistlock time to attach a container placed in a slot at tier t.

Before the premarshalling process begins, the trolley is at a safety position outside the bay, next to the leftmost stack. We denote this position as stack 0. The premarshalling process starts with the trolley traveling without load from stack 0 to a particular stack $s \in \mathcal{S}$. The time taken for this shift is represented by $h^0_{0,s}$.

5.2 CPMP-LCT: The Container Premarshalling Problem under Limited Crane Time

Container premarshalling is aimed at increasing the operational efficiency of port terminals. Unlike mandatory operations, such as retrieving containers from the yard for loading onto a vessel, premarshalling is a complementary activity that enhances the development of other operations. Consequently, premarshalling is only conducted during idle periods of the crane, in between other terminal operations.

It is essential to note that premarshalling is subject to the availability of the crane because this limitation can prevent the completion of the arrangement process. Unfortunately, the existing literature does not offer any premarshalling solutions if this occurs. However, it is often beneficial to perform partial premarshalling when an entire arrangement is not possible.

A partial premarshalling solution could be derived from a complete arrangement, stopping at an intermediate step of a solution for the CPMP or the CPMPCT. However, this method often yields poor-quality solutions, as illustrated later in Section 5.2.1. To seek good-quality partial premarshalling solutions, we define the Container Premarshalling Problem under Limited Crane Time (CPMP-LCT).

The CPMP-LCT aims to find a sequence of relocations that produce the most efficient bay arrangement within a limited crane time. If sufficient time is available, an optimal solution to the problem is an entire bay arrangement;

otherwise, an optimal solution is the most efficient partial arrangement. The objective of the CPMP-LCT, which quantifies the efficiency of a partial arrangement, is presented in the following paragraphs after describing the general assumptions of the problem.

The assumptions for the CPMP-LCT include (i) to (v) of the assumptions provided for the CPMP in Section 3.1. However, assumption (vi), which considers a uniform movement cost for relocations in the CPMP, is replaced by (xiii) from the CPMPCT, where the cost of each relocation is its associated crane time. Moreover, in the CPMP-LCT, assumption (vii) from the CPMP is not applicable: instead of unlimited crane availability, assumption (xiv) is defined for the CPMP-LCT.

xiv) Limited crane time: The crane's time available for premarshalling is subject to a limit τ. The total time taken by the premarshalling relocations cannot exceed this limit.

Although assumption (iv) of the CPMP, which considers full information about retrieval times, also applies to CPMP-LCT, the objective function described in the subsequent paragraphs aims to reduce the risk associated with changes in retrieval information.

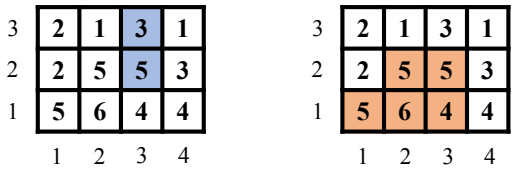
Defining an objective to measure the efficiency of partial solutions is a challenging task, and it is discussed in greater detail in Chapter 6. Nevertheless, the ultimate goal of the CPMP-LCT is clear. Like the CPMP, the CPMP-LCT aims to prepare the bay of containers in the best way to expedite the retrieval of containers. Specifically, the CPMP-LCT seeks the best arrangement of the bay for a retrieval process subsequent to premarshalling, where all the containers are retrieved from the bay following the priority schedule.

The objective defined for the CPMP-LCT in this chapter requires distinguishing between accessible and inaccessible containers. An accessible container is one that can be retrieved from the bay before any relocation is needed in the retrieval process. Conversely, a container is inaccessible if it or another container with an earlier retrieval time requires a relocation to be retrieved. In other words, a container is inaccessible if a container above it with a later retrieval time blocks its retrieval or if such blocking affects another container scheduled earlier in the retrieval sequence.

Figure 5.3 shows an example of a bay layout where the inaccessible containers are indicated on the right and the badly placed containers on the left. The container with priority 3 in the third stack is badly placed because it is necessary to relocate it to achieve a complete arrangement of the bay, even though it

is not blocking the retrieval of another container. The badly placed container with priority 5 is blocking the retrieval of the container with priority 4 in that stack. Since this is the only blockage in the bay, all the containers preceding the container with priority 4 in stack 3 can be retrieved without additional relocations. Hence, as observed on the right, all the containers with priorities earlier than 4 and the container with priority 4 in the fourth stack are accessible. Conversely, the container with priority 4 in the third stack needs the relocation of the container with priority 5 to be retrieved, and thus, it is inaccessible. Consequently, the containers with later priorities than 4 are also inaccessible, as relocations are needed in the retrieval process before they can be retrieved.

Figure 5.3: Examples of badly placed and inaccessible containers.



An example of a bay layout where the inaccessible containers are highlighted in orange on the right, and the badly placed containers are highlighted in blue on the left. The indexes corresponding to the tiers are displayed vertically on the left of each bay layout, and those for the stacks are indicated horizontally at the bottom. Elaborated by the author (2024).

We define the objective for the CPMP-LCT as minimizing the number of inaccessible containers. This approach seeks a sequence of relocations that can be performed within the crane time limit and results in a bay layout with the fewest inaccessible containers. Consequently, if the premarshalling is complete, all the containers in the bay will be accessible in the final layout. On the contrary, if the time is insufficient for a complete arrangement, this objective prioritizes unblocking those containers that must be retrieved first. In essence, it aims to unblock as many containers as possible in the order of retrieval priorities.

The objective of minimizing the inaccessible containers offers several advantages. Apart from the potential reduction in relocations during the retrieval phase resulting from unblocking containers, this approach minimizes the risk of the solution being affected by changes in the retrieval priorities. As information about retrieval priorities can change over time due to variations in route or berth assignments, containers with later retrieval times are more susceptible

to this uncertainty. Therefore, this objective prioritizes containers for which the information is the most reliable. Additionally, focusing on containers with the earliest retrieval times is advantageous if information about the priorities of the last containers to be retrieved is missing or if the retrieval process is expected to be interrupted, potentially allowing for later premarshalling of the remaining containers.

5.2.1 Partial premarshalling solutions

In this section, we illustrate a partial premarshalling solution and elaborate on the characteristics of partial solutions that underscore the importance of defining the CPMP-LCT.

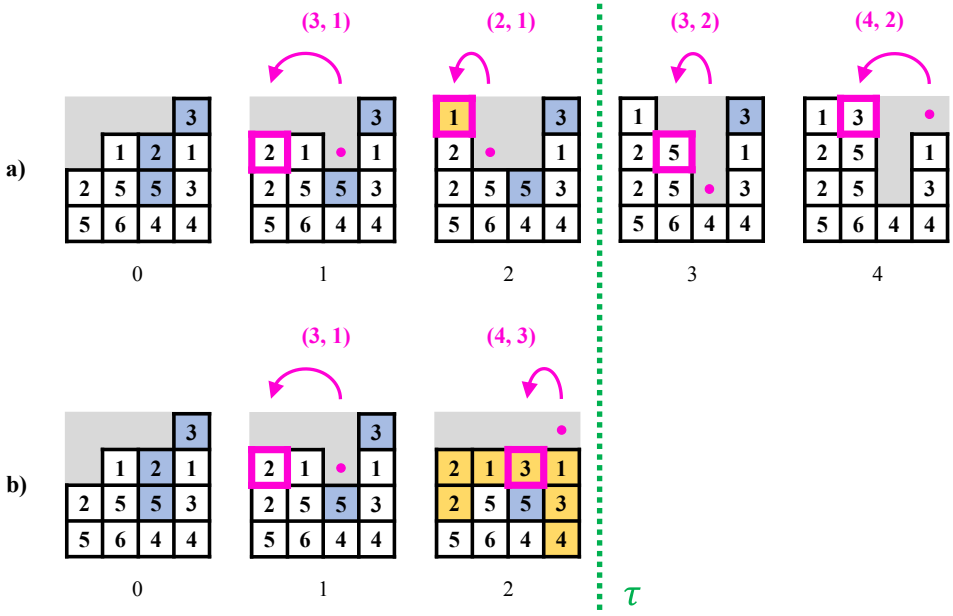
Given a crane time limit and a sequence of relocations intended to fully arrange a bay that exceeds the available time, a partial premarshalling solution can be derived by following this sequence and stopping before the limit is exceeded. However, this method often produces inefficient partial premarshalling solutions.

Figure 5.4a shows an optimal solution for completely arranging the bay. Assuming a time limit τ that permits only the first two movements, as indicated in the figure, the configuration in the second stage is the final layout of a solution within the time limit. In this bay configuration, only one container, highlighted in yellow, can be retrieved without additional relocations.

Figure 5.4b represents a partial premarshalling sequence within the time limit. A different movement is performed in the second stage compared to Figure 5.4a. This relocation results in a configuration where seven containers are accessible, a significant improvement over the single accessible container in Figure 5.4a. Thus, the example depicted in Figure 5.4b corresponds to a more efficient partial premarshalling solution.

Figure 5.4 indicates the badly placed containers in light blue. In Figure 5.4a, we observe no badly placed containers in the final layout of the complete premarshalling, adhering to the definition of a solution to the CPMP or the CPMPCT. Conversely, in the final configuration of Figure 5.4b, there are two badly placed containers (the container with priority 3 relocated in stage 2 is badly placed as it is on top of a badly placed container). This is the same number as in the second and final stage of the partial premarshalling solution depicted in Figure 5.4a. However, there is a notable difference in the number of inaccessible containers between the two partial premarshalling solutions.

Figure 5.4: Examples of complete and partial premarshalling solutions.



An example of an optimal solution for a) complete premarshalling and b) partial premarshalling under crane time limit τ. The numbers below the bay layouts indicate the premarshalling stages. The relocation performed at each stage is indicated in magenta by an arrow and a pair (s, r), where s and r are the origin and destination stacks. The relocated container is highlighted with a thick border, and a dot indicates its origin slot. The badly placed containers are shaded in light blue, and the accessible containers in the second stage are highlighted in yellow. A green dotted line represents the crane time limit τ. Elaborated by the author (2024) based on a figure from Jiménez-Piqueras et al., 2024.

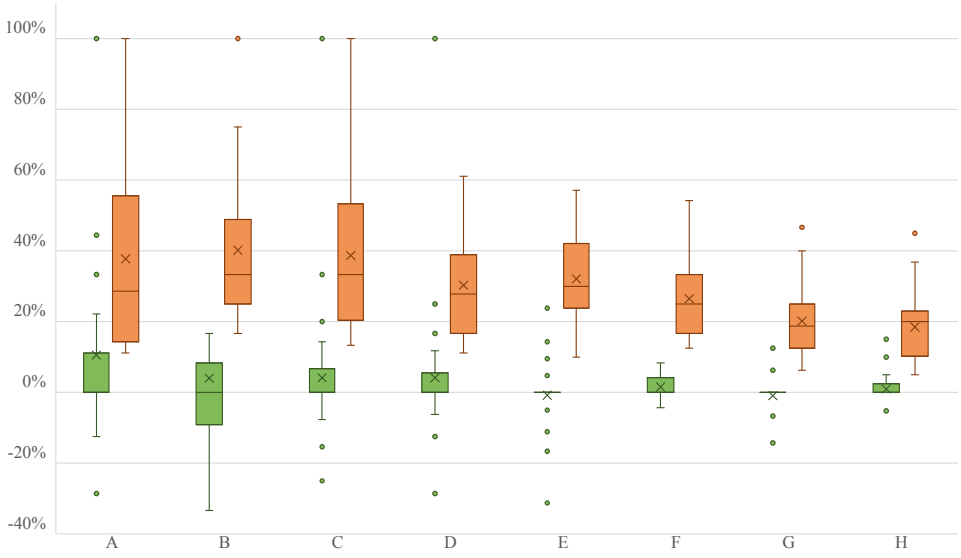
Seeking a solution without badly placed containers is an efficient approach for defining complete premarshalling solutions, but partial premarshalling solutions require different strategies. The solution depicted in Figure 5.4a is provided by the MCT model proposed for solving the CPMPCT in Section 5.3.1. In contrast, the solution shown in Figure 5.4b is yielded by LCT1, the formulation for the CPMP-LCT introduced in Section 5.3.2.

Figure 5.5 allows for a comparison of the quality of solutions provided by LCT1 and MCT for premarshalling under a limited crane time of ten minutes over a large set of instances specified in Table 5.1. The complete premarshalling solutions yielded by MCT are processed to obtain partial premarshalling ones

when the time limit is exceeded, using the same method illustrated in Figure 5.4: selecting the longest sequence of relocations within the time limit, starting at the initial relocation of the complete solution.

We observe that the solutions provided by LCT1 produce a significantly greater reduction in the number of inaccessible containers between the initial and final bay configurations compared to those given by MCT. Moreover, in most of the solutions provided by MCT, the reduction in the number of inaccessible containers is close to 0% or even negative. A negative reduction indicates that the final configuration contains more inaccessible containers than the initial one. In contrast, solving the CPMP-LCT through LCT1 guarantees solutions that do not worsen the initial bay configuration.

Figure 5.5: Percentage reduction in inaccessible containers.



Percentage reduction in inaccessible containers between the initial and final bay configurations for premarshalling solutions within a crane time limit of ten minutes over the instances indicated in Table 5.1. In green are the results obtained by the MCT model, and in orange are those given by LCT. Source: Elaborated by the author (2024) based on a figure from Jiménez-Piqueras et al., 2024.

Table 5.1: Details of the instances considered in Figure 5.5.

Category	A	B	C	D	E	F	G	H	Total
$\bar{s}$	3	4	5	6	7	8	4	5	
$\bar{t}$	5	5	5	5	5	5	6	6	
#Instances	40	40	40	39	35	27	40	25	286

The instances are grouped by number of stacks $\bar{s}$ and number of tiers $\bar{t}$, and are sourced from the CV dataset (information about this dataset can be found in Section 3.3.1). Only the results of the instances solved by both models, MCT and LCT1, are used in the plot. The number of these instances per group appears in row #Instances. Source: Jiménez-Piqueras et al., 2024.

5.3 Constraint programming models for the CPMPCT and the CPMP-LCT

This section presents a constraint programming formulation to solve the CPMPCT, which we call MCT. This model is based on the constraint programming model for the classical premarshalling problem CP5, introduced in Section 3.2.4.

The MCT model is a stepping-stone for modeling the CPMP-LCT. Using MCT as a basis, we have developed two constraint programming models to solve the CPMP-LCT with the objective of minimizing the number of inaccessible containers: a generic formulation and an alternative formulation for specific bays where all containers have distinct retrieval priorities.

5.3.1 MCT: A model for premarshalling minimizing crane time

The MCT formulation involves seven groups of decision variables: the five groups of variables from the CP5 model and two additional sets. The new groups of decision variables, $g^k_{s,r}$ and $u^k_{s,r}$, allow us to incorporate the time corresponding to the horizontal movements of the crane's trolley into the objective function. All the variables in the MCT model are described below. Variables $w^k_{s,t}$, which are not defined for the initial stage in CP5, need to be extended to this stage in MCT due to a group of table constraints included in the formulation.

$$x^k_{s,t} = \begin{cases} p & \text{If a container with priority } p \text{ is in slot } (s,t) \text{ at stage } k \\ 0 & \text{If slot } (s,t) \text{ is empty at stage } k \end{cases}$$

$\forall s \in \mathcal{S}, \forall t \in \mathcal{T}, \forall k \in \mathcal{K}^0$

$$\delta^k_{s,t} = \begin{cases} 1 & \text{If there is a container in slot } (s,t) \text{ at stage } k \\ 0 & \text{If slot } (s,t) \text{ is empty at stage } k \end{cases}$$

$\forall s \in \mathcal{S}, \forall t \in \mathcal{T}, \forall k \in \mathcal{K}^0$

$$w^k_{s,t} = \begin{cases} 1 & \text{If there is a badly placed container in } (s,t) \text{ during stage } k \\ 0 & \text{Otherwise} \end{cases}$$

$\forall s \in \mathcal{S}, t \in \mathcal{T}, k \in \mathcal{K}^0$

$$y^k_{s,t} = \begin{cases} 1 & \text{If a container is moved to slot } (s,t) \text{ during stage } k \\ 0 & \text{Otherwise} \end{cases}$$

$\forall s \in \mathcal{S}, t \in \mathcal{T}, k \in \mathcal{K}$

$$z^k_{s,t} = \begin{cases} 1 & \text{If a container is removed from slot } (s,t) \text{ during stage } k \\ 0 & \text{Otherwise} \end{cases}$$

$\forall s \in \mathcal{S}, t \in \mathcal{T}, k \in \mathcal{K}$

$$g^k_{s,r} = \begin{cases} 1 & \text{If the crane's trolley moves loaded with a container from} \\ & \text{stack } s \text{ to stack } r \text{ during stage } k \\ 0 & \text{Otherwise} \end{cases}$$

$\forall s \in \mathcal{S}, \forall r \in \mathcal{S} \setminus \{s\}, \forall k \in \mathcal{K}$

$$u^k_{s,r} = \begin{cases} 1 & \text{If the crane's trolley moves unloaded from stack } s \text{ to } r, \\ & \text{being } r \text{ the origin stack of a relocation during stage } k \\ 0 & \text{Otherwise} \end{cases}$$

$\forall s \in \mathcal{S}, \forall r \in \mathcal{S} \setminus \{s\}, \forall k \in \mathcal{K} \setminus \{1\}$

The objective of the CPMPCT is to minimize the crane time spent in arranging the bay. According to the notation for crane time presented in Section 5.1.1, we define the objective function (5.1). The first term accounts for the horizontal movements of the trolley, and the second term accounts for the vertical movements of the spreader. Variables $g^1_{s,r}$ indicate the first movement, where the trolley travels unloaded from the safety position to the origin stack of the first relocation. In the remaining stages, variables $u^k_{s,r}$ identify the trolley movements without load toward the origin stack of each relocation. Movements where the trolley is loaded with a container are described by variables $g^k_{s,r}$. The vertical travel times of the spreader are included using variables $y^k_{s,t}$ and $z^k_{s,t}$. Variables $z^k_{s,t}$ are also used to incorporate the twistlock times in the objective.

$$\min \sum_{s\in\mathcal{S}} \left(\sum_{\substack{r\in\mathcal{S}\\ r\neq s}} \left(h^0_{0,s}\cdot g^1_{s,r} + \sum_{k\in\mathcal{K}\setminus\{1\}} \left(h^0_{r,s}\cdot u^k_{r,s}\right) + \sum_{k\in\mathcal{K}} \left(h^1_{s,r}\cdot g^k_{s,r}\right) \right) + \right.$$
$$\left. \sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}} \left(\left(v_t + l_t\right)\cdot z^k_{s,t} + v_t \cdot y^k_{s,t} \right) \right) \tag{5.1}$$

Similar to CP5, the initial bay configuration in MCT is set through variables $x^0_{s,t}$, $\delta^0_{s,t}$, and additionally $w^0_{s,t}$, using the parameters $\alpha^x_{s,t}$, $\alpha^\delta_{s,t}$, and $\alpha^w_{s,t}$ in constraints (5.2) to (5.4). Also, the values of variables $w^k_{s,1}$ are set to 0 by (5.5) because the containers placed in the first tier never block other containers and thus are never badly placed.

$$x^0_{s,t} = \alpha^x_{s,t} \qquad \forall s\in\mathcal{S},\, t\in\mathcal{T} \tag{5.2}$$
$$\delta^0_{s,t} = \alpha^\delta_{s,t} \qquad \forall s\in\mathcal{S},\, t\in\mathcal{T} \tag{5.3}$$
$$w^0_{s,t} = \alpha^w_{s,t} \qquad \forall s\in\mathcal{S},\, t\in\mathcal{T} \tag{5.4}$$
$$w^k_{s,1} = 0 \qquad \forall s\in\mathcal{S},\, k\in\mathcal{K} \tag{5.5}$$

The MCT model incorporates most of the constraints from CP5, with some reformulated using boolean expressions and global constraints, similar to the formulations for the CPMP-AB presented in Section 4.2.

The entire formulation of MCT is shown below. Some constraints are the same as in CP5 but are included here for clarity. These constraints are indicated with the equality symbol (=) in Table 5.2. The remaining constraints in Table 5.2

express equivalent conditions in both models but are formulated differently. Additionally, MCT introduces two groups of constraints, (5.16) and (5.17), to incorporate the new variables that describe the movements of the crane's trolley.

$$Count\big(x^k_{s,t} : s \in \mathcal{S},\, t \in \mathcal{T},\, x^k_{s,t} = p\big) = m_p \qquad \forall p \in \mathcal{P}^0,\, k \in \mathcal{K} \tag{5.6}$$

$$\big(x^k_{s,t} > 0\big) = \delta^k_{s,t} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T},\, k \in \mathcal{K} \tag{5.7}$$

$$\big(x^k_{s,t} == x^{k-1}_{s,t}\big) = \big(\delta^k_{s,t} == \delta^{k-1}_{s,t}\big) \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T},\, k \in \mathcal{K} \tag{5.8}$$

$$\big(x^k_{s,t+1} > x^k_{s,t}\big) \;\leq\; w^k_{s,t+1} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T} \setminus \{\bar{t}\},\; k \in \mathcal{K} \tag{5.9}$$

$$w^k_{s,t+1} \;\leq\; w^k_{s,t} + \big(x^k_{s,t} < x^k_{s,t+1}\big) \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T} \setminus \{\bar{t}\},\; k \in \mathcal{K} \tag{5.10}$$

$$\sum_{s \in \mathcal{S}, t \in \mathcal{T}} w^k_{s,t} \;+\; k \;\leq\; \bar{k} \qquad \forall k \in \mathcal{K} \tag{5.11}$$

$$\sum_{s \in \mathcal{S},\, t \in \mathcal{T}} y^k_{s,t} \;\leq\; 1 \qquad \forall k \in \mathcal{K} \tag{5.12}$$

$$\sum_{s \in \mathcal{S},\, t \in \mathcal{T}} z^k_{s,t} \;\leq\; 1 \qquad \forall k \in \mathcal{K} \tag{5.13}$$

$$y^k_{s,t} \;\leq\; \delta^{k+1}_{s,t} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T},\; k \in \mathcal{K} \setminus \{\bar{k}\} \tag{5.14}$$

$$AllowedAssignments\big(A,\; \delta^{k-1}_{s,t}, \delta^{k-1}_{s,t+1}, \delta^k_{s,t}, \delta^k_{s,t+1}, z^k_{s,t}, z^k_{s,t+1}, y^k_{s,t}, y^k_{s,t+1}, w^{k-1}_{s,t}, w^{k-1}_{s,t+1}, w^k_{s,t}, w^k_{s,t+1}\big) \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T} \setminus \{\bar{t}\},\; k \in \mathcal{K} \tag{5.15}$$

$$g^k_{s,r} = \left(\sum_{t \in \mathcal{T}} \big(z^k_{s,t} + y^k_{r,t}\big) == 2\right) \qquad \forall s \in \mathcal{S},\, r \in \mathcal{S} \setminus \{s\},\; k \in \mathcal{K} \tag{5.16}$$

$$u^k_{s,r} = \left(\sum_{t \in \mathcal{T}} \big(y^{k-1}_{s,t} + z^k_{r,t}\big) == 2\right) \qquad \forall s \in \mathcal{S},\, r \in \mathcal{S} \setminus \{s\},\; k \in \mathcal{K} \setminus \{1\} \tag{5.17}$$

Table 5.2: Correspondence between constraints from models MCT and CP5.

MCT		CP5
(5.6)	=	(3.3)
(5.7)		(3.13), (3.14)
(5.8)		(3.15), (3.16)
(5.9)		(3.29)
(5.10)		(3.33)
(5.11)	=	(3.34)
(5.12)	=	(3.8)
(5.13)	=	(3.22)
(5.14)	=	(3.21)
(5.15)		(3.20), (3.23), (3.24), (3.25), (3.26), (3.30), (3.31)

The left column of the table shows the constraints of model MCT replacing the corresponding constraints from CP5 indicated in the right column. Symbol = indicates that the constraints in both columns are the same. Source: Elaborated by the author (2024).

The conditions required to regulate the bay configuration in MCT are the same as in CP5 and are described by (5.6) to (5.11), using variables x, δ and w. The Count constraints (5.6) ensure the number of containers with a given priority remains the same across stages. Variables δ represent the projection of the integer variables x onto the binary set, and this connection is established by (5.7). Constraints (5.8) prevent a container's priority from changing between consecutive stages if the slot it occupies in the first stage remains occupied in the subsequent one.

Variables w, identifying badly placed containers, are defined in terms of variables x through constraints (5.9) and (5.10). Constraints (5.9) classify a container as badly placed if it is on top of a container with an earlier retrieval time. Also, a container is deemed badly placed if it is above another badly placed container. This condition is covered by the table constraints (5.15) in MCT. (5.10) indicate that if a container in slot $(s, t+1)$ is badly placed, then at least one of the following conditions must hold: it must be retrieved later than the container located in slot (s, t), or the container in (s, t) is also badly placed. Constraints imposing an upper bound on the number of badly placed containers at each stage using variables w are the same for CP5 and MCT, (3.34) and (5.11), respectively.

MCT requires the conditions referring to container relocations used in CP5 and are expressed by (5.12) to (5.15). Additional constraints, (5.16) and (5.17), are incorporated into MCT to define the variables that identify the crane movements, which are used to measure the crane time.

Constraints (5.12) and (5.13) impose the limit of one relocation per stage. The variables identifying the origin and destination slots of relocations, z and y, are linked to the variables describing the bay configuration through constraints (5.14) and (5.15). Constraints (5.14) relate y and δ, discarding solutions where the same container is moved during two consecutive stages (transitive move avoidance).

Table constraints (5.15) impose the conditions that the following combination of constraints do in CP5: (3.20), (3.23) to (3.26), (3.30) and (3.31). The global constraint Allowed Assignments restricts the possible combinations of values for the set of variables indicated as arguments to those included in matrix A, containing all permitted combinations. Matrix A is identical to that used in constraints (4.20) in the AB model for the CPMP-AB, presented in the preceding chapter. It enumerates sixteen possible combinations of values detailed at the end of Section 4.2.1.

Constraints (5.15) specify how different groups of variables must reflect the same relocation, ensuring the bay configuration changes according to the relocation performed. Such conditions include the slot where the container is relocated being empty and becoming occupied in the stage where the relocation is performed. Analogously, the slot from which this container is retrieved must be occupied and become empty with the relocation. Additionally, these constraints prevent empty slots between containers. Badly placed containers are also considered in these table constraints, ensuring a container moved on top of a badly placed container must be deemed as badly placed.

The newly defined variables for MCT, g and u, describing crane movements, are connected through constraints (5.16) and (5.17) with the variables indicating container relocations, y and z. If the crane's trolley travels loaded with a container from stack s to stack r at stage k, the logical expression on the right-hand side of (5.16) returns a value of 1, and it is assigned to variable $g^k_{s,r}$. Similarly, in (5.17), if the trolley travels unloaded from s (the destination stack of the relocation performed at stage $k-1$) to stack r (the origin stack of the relocation conducted at stage k), variable $u^k_{s,r}$ is set to 1.

The only conditions from CP5 not included in MCT are those expressed by constraints (3.27) and (3.28). The purpose of constraints (3.27) in CP5 is

to discard infeasibilities faster, but they have not shown an improvement in the performance of the MCT model. Constraints (3.28) are meant to remove symmetries in the CPMP but do not apply to the CPMPCT, as two solutions equivalent for minimizing the number of relocations may not be equivalent for minimizing the crane time.

5.3.2 LCT1: A model for premarshalling under limited crane time

We have developed a constraint programming formulation, LCT1, to solve the CPMP-LCT. This model is adapted from MCT, described in previous Section 5.3.1.

LCT1 contains all the variables in the MCT formulation except for the set w. In MCT, variables w are beneficial because they allow for imposing an upper bound on the number of badly placed containers at each stage. However, this upper bound is only valid when the premarshalling is to be completed and cannot be defined when solving the CPMP-LCT.

The objective of the LCT1 formulation is to minimize the number of inaccessible containers. A new group of decision variables, $q_{s,t}$, is defined to identify such containers and formulate the objective function (5.18).

$$q_{s,t} = \begin{cases} 1 & \text{If an inaccessible container occupies } (s,t) \text{ in the final layout} \\ 0 & \text{Otherwise} \end{cases}$$

$$\forall s \in \mathcal{S},\, \forall t \in \mathcal{T}$$

$$\min \sum_{s \in \mathcal{S}} \sum_{t \in \mathcal{T}} q_{s,t} \tag{5.18}$$

The LCT1 formulation contains all the constraints of MCT except for those involving variables w, namely (5.4), (5.5), (5.9) to (5.11) and (5.15). Additionally, LCT1 includes constraints (5.19) to (5.22).

$$\begin{aligned} &AllowedAssignments(C, \\ &\delta^{k-1}_{s,t}, \delta^{k-1}_{s,t+1}, \delta^{k}_{s,t}, \delta^{k}_{s,t+1}, \\ &z^{k}_{s,t}, z^{k}_{s,t+1}, y^{k}_{s,t}, y^{k}_{s,t+1}) \end{aligned} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T} \setminus \{\bar{t}\},\, k \in \mathcal{K} \tag{5.19}$$

$$\sum_{s\in\mathcal{S},t\in\mathcal{T}} z_{s,t}^{k+1} \leq \sum_{s\in\mathcal{S},t\in\mathcal{T}} z_{s,t}^{k} \qquad \forall k \in \mathcal{K}\setminus\{\bar{k}\} \tag{5.20}$$

$$\sum_{j\in\mathcal{T}:j>t} (x_{s,j}^{\bar{k}} > x_{s,t}^{\bar{k}}) \leq \bar{t}\cdot q_{s,t} \qquad \forall s\in\mathcal{S}, t\in\mathcal{T}\setminus\{\bar{t}\} \tag{5.21}$$

$$\sum_{i\in\mathcal{S}}\sum_{j\in\mathcal{T}} (x_{s,t}^{\bar{k}} > x_{ij}^{\bar{k}})\cdot q_{i,j} \leq \bar{c}\cdot q_{s,t} \qquad \forall s\in\mathcal{S},\ t\in\mathcal{T} \tag{5.22}$$

$$\sum_{s\in\mathcal{S}}\left(\sum_{\substack{r\in\mathcal{S}\\ r\neq s}}\left(h_{0,s}^{0}\cdot g_{s,r}^{1} + \sum_{k\in\mathcal{K}\setminus\{1\}}(h_{r,s}^{0}\cdot u_{r,s}^{k}) + \sum_{k\in\mathcal{K}}(h_{s,r}^{1}\cdot g_{s,r}^{k})\right)+\right.$$

$$\left.\sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}}\left((v_t+l_t)\cdot z_{s,t}^{k} + v_t\cdot y_{s,t}^{k}\right)\right) \leq \tau \tag{5.23}$$

The table constraints (5.19) replace (5.15) from MCT. These constraints must be rewritten for LCT1 to remove variables w. Matrix A in the arguments of (5.15) contains sixteen possible combinations for the variables, while matrix C associated with (5.19) includes only seven combinations. There are fewer combinations in C because it is not necessary to distinguish between badly placed and well-placed containers in this context. In Figure 4.3, we can see that there are seven allowed transitions between the possible states for a given pair of slots (s,t) and $(s,t+1)$ without badly placed containers. These transitions correspond to the combinations included in C.

In the CPMP-LCT, no condition is imposed on the final layout, unlike in the CPMP or the CPMPCT, where the final configuration must contain no badly placed containers. Therefore, any sequence of relocations within the time limit τ is a feasible solution for the CPMP-LCT, including the trivial solution of not performing any relocation. This characteristic of the problem means a wide range of solutions to evaluate, including numerous solutions with many stages where no relocations are performed. To significantly narrow this range of potential solutions, we impose that all relocations occur consecutively during the first stages. Constraints (5.20) are included in LCT1 to enforce this, thereby avoiding the consideration of equivalent solutions.

LCT1 must include some constraints to identify inaccessible containers. A container is inaccessible if: a) there is a container at a higher tier of the same stack with a later retrieval time, or b) there exists an inaccessible container with an earlier retrieval time. Condition (a) is expressed by (5.21), and condition (b) by (5.22).

The mathematical expression that computes the crane time and defines the objective function of MCT is included in LCT1 as a constraint, (5.23), which ensures that the total crane time used in the relocations does not exceed the limit τ.

5.3.3 LCT2: An alternative model for bays where all container priorities differ

When all the containers in a bay have distinct priorities, each priority group uniquely defines a container. Therefore, minimizing the number of inaccessible containers in the final layout is equivalent to minimizing the number of priority groups with an inaccessible container in the final configuration. Based on this idea, we can model the problem for this specific scenario with a formulation that requires fewer decision variables referring to the inaccessible containers. We define the model LCT2 by replacing variables $q_{s,t}$ from LCT1 with q^p, which are defined as follows:

$$q^p = \begin{cases} 1 & \text{If there exists a container with priority } p \text{ that is inaccessible} \\ & \text{in the final configuration} \\ 0 & \text{Otherwise} \end{cases}$$

$$\forall p \in \mathcal{P}$$

The objective function is modified accordingly, replacing (5.18) from LCT1 by (5.24).

$$\min \sum_{p \in \mathcal{P}} q^p \tag{5.24}$$

Constraints (5.21) and (5.22) from LCT1 are substituted with (5.25) and (5.26) in LCT2. The rest of the constraints are the same in both formulations.

$$\sum_{s \in \mathcal{S}} \sum_{t \in \mathcal{T}} (x^{\bar{k}}_{s,t} == p) \cdot \sum_{j \in \mathcal{T}: j > t} (x^{\bar{k}}_{s,j} > x^{\bar{k}}_{s,t}) \leq \bar{c} \cdot q^p \qquad \forall p \in \mathcal{P} \tag{5.25}$$

$$q^p \leq q^{p+1} \qquad \forall p \in \mathcal{P} \setminus \{\bar{p}\} \tag{5.26}$$

Variables q^p refer to priority groups rather than single containers. When all the containers have distinct priorities, variables q^p identify the inaccessible containers in the same way as variables $q_{s,t}$. Therefore, solving the CPMP-LCT

via LCT1 and LCT2 yields the same optimal solution. However, if multiple containers have the same priority, while an optimal solution for the LCT1 formulation is also optimal for LCT2, the reverse may not be true. LCT2 only distinguishes between solutions that have no inaccessible containers from a particular priority group and solutions that do have one or more inaccessible containers with that priority, while LCT1 is also sensitive to the specific number of inaccessible containers within a priority group.

5.3.4 Solution method

The MCT model has a continuous objective, unlike the CP5 model proposed in Chapter 3 for the CPMP. While the minimization of the number of relocations can be successfully guided by an iterative algorithm, as explained in Section 3.2.5, the same cannot be applied to minimizing the crane time.

The MCT formulation serves as an intermediate step between the formulation for the original premarshalling, CP5, and the proposed models for the novel CPMP-LCT, namely LCT1 and LCT2. Therefore, we have chosen a solution method that allows for easy comparison with the state-of-the-art integer programming model for the CPMPCT, IPCT, proposed by Parreño-Torres et al., 2020, thus proving the adequacy of MCT as a basis for the LCT1 and LCT2 formulations.

The method used in Section 5.4 for the MCT model involves replacing constraints (5.12) and (5.13) with (3.35) and (3.36), used in the solving procedure explained in Section 3.2.5 for the proposed models for the CPMP, to enhance the performance of the constraint programming solver. Moreover, it requires a fixed number of stages $\bar{k}$. We define this value using the solutions provided by the branch and bound algorithm for the CPMPCT presented by Parreño-Torres et al., 2020. For the instances where the branch and bound obtains an optimal solution, we use the number of relocations as the number of stages. For the rest of the instances, the number of stages is calculated using Proposition 2 of Parreño-Torres et al., 2020, which provides an upper bound for the number of relocations of an optimal solution using the crane time of a feasible solution. Unlike the procedure proposed for CP5, in this case, the objective function is provided to the constraint programming solver, which solves an optimization problem instead of several satisfaction problems.

The LCT1 and LCT2 models are provided to the constraint programming solver including their objective function, similar to MCT. However, in this case, the number of stages does not necessarily correspond to the number of

relocations performed. Unlike the CPMPCT, we do not have a previous reference for the number of stages of the CPMP-LCT. Nevertheless, the limited crane time allows for setting a simplified upper bound on the number of relocations: we define the number of stages, $\bar{k}$, as the ceiling of the quotient between the crane time limit, τ, and the minimum crane time required to perform a single relocation, t_{min}:

$$\bar{k} = \left\lceil \frac{\tau}{t_{min}} \right\rceil \tag{5.27}$$

While the crane time is measured in both the CPMPCT and the CPMP-LCT, it is not involved in the objective function of the CPMP-LCT. The objective of minimizing the inaccessible containers is discrete, as in the original premarshalling. Hence, as for CP5, an iterative algorithm is suitable as a solution method for the models proposed for the CPMP-LCT. This is addressed in Chapter 6.

5.4 Computational experiments

All the formulations proposed in this chapter have been tested through computational experiments to evaluate their performance. First, we contrast the results obtained by the MCT model, described in Section 5.3.1, with those yielded by the state-of-the-art integer programming model for the CPMPCT, namely IPCT, developed by Parreño-Torres et al., 2020. Then, we test the formulations for the CPMP-LCT, LCT1 and LCT2, described in Section 5.3.2 and Section 5.3.3, respectively.

To ensure a fair comparison, MCT and IPCT have been solved using the same fixed number of stages $\bar{k}$ and the corresponding constraint programming and mathematical programming solvers to search for solutions that minimize the total crane time. Further details about the solution method used for MCT, LCT1, and LCT2 can be found in Section 5.3.4.

A total of 420 instances are used in the computational study. These instances are sourced from the CV and ZJY datasets described in Section 3.3.1. The rationale for selecting these specific datasets is that the instances are sufficiently complex, with crane times required for premarshalling that are long enough to be meaningful for defining the CPMP-LCT.

5.4.1 Minimizing crane time: MCT vs IPCT

Table 5.3 shows the results obtained by the proposed constraint programming model MCT and the state-of-the-art integer programming model IPCT. This comparison demonstrates the superior performance of MCT over IPCT.

In the ZJY dataset, MCT solves all but two instances, which are 10 instances more than using IPCT. Moreover, the constraint programming formulation obtains shorter average crane times than IPCT. Despite the integer linear model having a slightly shorter average computational time, MCT provides better solutions overall.

The advantage of MCT over IPCT is even more significant across the CV dataset. The MCT model solves 89% of the instances, whereas IPCT only solves 58%. Additionally, MCT yields more optimal solutions, is computationally faster than IPCT, and generally achieves smaller objective function values.

Table 5.3: Performance of models MCT and IPCT for the CPMPCT.

Dataset	$\bar{s}$	$\bar{t}$	#I	#Optimal		#Feasible			Av. CTime (s)		Av. CPU (s)	
				MCT	*IPCT*	*MCT*	*IPCT*	*Both*	*MCT*	*IPCT*	*MCT*	*IPCT*
ZJY	6	4	20	**14**	**14**	**20**	18	18	**1038**	1040	**1253**	1326
	7	4	20	13	**15**	**20**	17	17	**1173**	1177	1456	**1425**
	8	4	20	**11**	**11**	**20**	18	18	1042	**1038**	2060	**2035**
	9	4	20	**5**	**5**	**19**	18	17	**1351**	1354	3075	**2865**
	6	5	20	5	**6**	**19**	17	17	**1590**	1620	3043	**3014**
	Total		100	48	**51**	**98**	88	87	**1234**	1241	2165	**2122**
CV	3	5	40	**40**	**40**	**40**	**40**	40	**1010**	**1010**	**7**	94
	4	5	40	**40**	38	**40**	36	36	**1002**	**1002**	**36**	278
	5	5	40	**33**	28	**40**	37	37	**1199**	1206	**814**	1430
	6	5	40	**21**	16	**39**	29	29	**1243**	1264	**1867**	2271
	7	5	40	**11**	6	**35**	20	20	**1417**	1457	**2425**	2912
	8	5	40	**5**	2	**27**	11	11	**1381**	1417	**2527**	3247
	4	6	40	**15**	7	**40**	10	10	1640	**1638**	**1217**	2004
	5	6	40	**3**	1	**25**	2	2	**1652**	1693	**1859**	1917
	Total		320	**167**	137	**286**	185	185	**1191**	1202	**967**	1360

Results obtained with MCT and IPCT for the instances of the ZJY and CV datasets with $\bar{s}$ stacks and $\bar{t}$ tiers. The number of instances per category is shown in the #I column. Column #Optimal contains the number of instances proven optimal by each model, and column #Feasible provides the total of instances for which a solution is found. The column labeled Both shows the number of instances for which both models find a solution. The average crane time and running time for the instances in the column Both are displayed in the right-most columns. The best values are shown in bold. Source: Jiménez-Piqueras et al., 2024.

5.4.2 Performance of the proposed models for the CPMP-LCT: LCT1 and LCT2

In this section, we analyze the performance of the proposed model for the CPMP-LCT, LCT1, across the ZJY and CV datasets, with crane time limits set at 20 and 15 minutes. Additionally, we compare LCT1 with LCT2, the alternative formulation proposed for the CPMP-LCT when there are no repeated priorities. This comparison is conducted using the CV dataset, where all the containers have distinct priorities.

The results obtained from the ZJY dataset are presented in Table 5.4, and those from the CV dataset are displayed in Table 5.5. Across both datasets, we observe an average reduction in inaccessible containers between the initial and final configurations of 30% or more. This reduction increases with the available crane time, τ. Bays with the smallest dimensions and a crane time limit of 15 minutes yield an average reduction of 60% or more in both datasets.

Table 5.4: Performance of model LCT1 for the CPMP-LCT on the ZJY dataset.

τ (min)	$\bar{s}$	$\bar{t}$	#Instances	#Optimal	#Feasible	Reduction (%)	Average CPU (s)
10	6	4	20	4	20	37	3314
	7	4	20	0	20	31	3600
	8	4	20	3	20	44	3072
	9	4	20	0	20	38	3600
	6	5	20	0	20	18	3600
	Total		100	7	100	34	3437
15	6	4	20	4	20	60	2918
	7	4	20	0	20	43	3600
	8	4	20	4	20	55	2902
	9	4	20	2	20	46	3469
	6	5	20	0	20	27	3600
	Total		100	10	100	46	3298

Results obtained by LCT1 over the ZJY dataset given a crane time limit τ. The column #Instances contains the number of instances for each category defined by the number of stacks $\bar{s}$ and tiers $\bar{t}$. LCT1 solves the instances indicated in column #Feasible, and the number of instances proven optimal is displayed in column #Optimal. The average percentage reduction in the number of inaccessible containers and the average running time are calculated for the feasible solutions and shown in columns Reduction (%) and Average CPU (s), respectively. Source: Jiménez-Piqueras et al., 2024.

The results presented in Table 5.5 reveal that when there are no repeated priorities in the bay, the LCT2 model is a better approach than LCT1. The number of optimal solutions increases by 79% and 112% for τ values of 10

and 15 minutes, respectively, when solving the problem with LCT2 instead of LCT1. This higher number of optimal solutions implies a decrease in running time, as shown in the table. Despite this, the reduction in inaccessible containers between the initial and the final layout is almost the same for both models. This consistency suggests that LCT1 provides good-quality solutions even if they are not proven optimal.

In contrast to MCT and IPCT, the LCT1 and LCT2 models provide feasible solutions for all instances. The requirement of a completely arranged bay in the final stage imposes a strong condition that hinders the attainment of feasible solutions. However, when partial arrangement is sought, this condition is not applicable. Solving the CPMP-LCT using LCT1 or LCT2 makes it possible to obtain solutions for instances that the models designed for complete premarshalling do not solve.

Table 5.5: Performance of models LCT1 and LCT2 for the CPMP-LCT on the CV dataset.

τ (min)	$\bar{s}$	$\bar{t}$	#Instances	#Optimal		#Feasible	Reduction (%)		Average CPU (s)	
				LCT1	LCT2	Both	LCT1	LCT2	LCT1	LCT2
10	3	5	40	**40**	**40**	40	**38**	**38**	7	**5**
	4	5	40	**40**	**40**	40	**40**	**40**	116	**18**
	5	5	40	27	**40**	40	**39**	**39**	2259	**106**
	6	5	40	2	**39**	40	**30**	**30**	3503	**492**
	7	5	40	0	**35**	40	**30**	**30**	3600	**1345**
	8	5	40	0	**32**	40	**26**	**26**	3600	**1985**
	4	6	40	**40**	**40**	40	**20**	**20**	176	**13**
	5	6	40	22	**40**	40	**18**	**18**	2761	**59**
	Total		320	171	**306**	320	**30**	**30**	2009	**504**
15	3	5	40	**40**	**40**	40	**63**	**63**	46	**18**
	4	5	40	12	**28**	40	**62**	**62**	2609	**1737**
	5	5	40	4	**5**	40	**54**	53	3308	**3268**
	6	5	40	**2**	**2**	40	39	**43**	3496	**3426**
	7	5	40	0	0	40	36	**38**	**3600**	**3600**
	8	5	40	0	0	40	29	**33**	**3600**	**3600**
	4	6	40	1	**35**	40	**29**	**29**	3518	**924**
	5	6	40	0	**15**	40	24	**25**	3600	**2678**
	Total		320	59	**125**	320	42	**43**	2981	**2414**

Results obtained by LCT1 and LCT2 over the CV dataset given a crane time limit τ. The column #Instances contains the number of instances for each category defined by the number of stacks $\bar{s}$ and tiers $\bar{t}$. Both LCT1 and LCT2 obtain solutions for the instances indicated in column #Feasible. Those instances proven optimal by each model are displayed in columns #Optimal. The average percentage reduction in the number of inaccessible containers and the average running time are calculated for the feasible solutions and shown in columns Reduction (%) and Average CPU (s), respectively. The best values are shown in bold. Source: Jiménez-Piqueras et al., 2024.

5.4.3 Heuristic solutions in short times for the CPMP-LCT

Since models LCT1 and LCT2 always provide feasible solutions, they can be used as heuristic approaches. We have analyzed the results obtained with both models, imposing short running time limits of 60, 120, and 300 seconds, and compared them with the solutions presented in the previous section, with a limit of 3600 seconds. This evaluation allows us to assess the potential of these formulations as heuristic approaches. The results for the CV dataset are displayed in Table 5.6 and Table 5.7, for LCT1 and LCT2, respectively, and Table 5.8 shows the results yielded by LCT1 on the ZJY dataset.

Table 5.6: Performance of model LCT1 on the CV dataset in short running times.

τ	$\bar{s}$	$\bar{t}$	#Instances	#Optimal				Reduction (%)			
(min)				*60 s*	*120 s*	*300 s*	*3600 s*	*60 s*	*120 s*	*300 s*	*3600 s*
10	3	5	40	40	40	40	40	37	37	37	37
	4	5	40	11	19	40	40	40	40	40	40
	5	5	40	0	0	1	27	34	35	37	39
	6	5	40	1	1	1	2	27	28	29	30
	7	5	40	0	0	0	0	26	28	30	30
	8	5	40	0	0	0	0	21	23	25	26
	4	6	40	7	16	36	40	20	20	20	20
	5	6	40	0	0	0	22	17	18	18	18
	Total		320	59	76	118	171	28	29	29	30
15	3	5	40	25	40	40	40	62	62	62	62
	4	5	40	5	6	9	12	50	53	56	62
	5	5	40	0	1	2	4	35	42	47	54
	6	5	40	0	1	1	2	27	30	32	39
	7	5	40	0	0	0	0	21	26	30	36
	8	5	40	0	0	0	0	14	16	23	29
	4	6	40	0	0	0	1	22	24	27	29
	5	6	40	0	0	0	0	18	19	21	24
	Total		320	30	48	52	59	31	34	37	42
	Total		320	30	48	52	59	31	34	37	42

Results obtained with LCT1 for the instances of the CV dataset with $\bar{s}$ stacks and $\bar{t}$ tiers, and crane time limit τ. Column #Instances shows the number of instances per category. Columns #Optimal contains the number of instances proven optimal by the model using different running time limits. A feasible solution is obtained for every instance and running time limit. The average percentage reduction in the number of inaccessible containers in the feasible solutions obtained for each running time limit is displayed in the Reduction (%) columns. Source: Jiménez-Piqueras et al., 2024.

Table 5.7: Performance of model LCT2 on the CV dataset in short running times.

τ (min)	$\bar{s}$	$\bar{t}$	#Instances	#Optimal				Reduction (%)			
				60 s	*120 s*	*300 s*	*3600 s*	*60 s*	*120 s*	*300 s*	*3600 s*
10	3	5	40	40	40	40	40	37	37	37	37
	4	5	40	39	40	40	40	40	40	40	40
	5	5	40	16	29	35	40	37	39	39	39
	6	5	40	5	10	21	39	30	30	30	30
	7	5	40	0	0	6	35	28	29	30	30
	8	5	40	0	0	1	32	24	25	26	26
	4	6	40	40	40	40	40	20	20	20	20
	5	6	40	27	36	39	40	18	18	18	18
	Total		320	167	195	222	306	29	30	30	30
15	3	5	40	40	40	40	40	62	62	62	62
	4	5	40	7	8	11	28	53	56	59	62
	5	5	40	1	3	3	5	46	51	52	53
	6	5	40	1	1	2	2	31	34	37	43
	7	5	40	0	0	0	0	26	30	34	38
	8	5	40	0	0	0	0	20	24	26	33
	4	6	40	4	6	16	35	25	28	29	29
	5	6	40	0	0	2	15	21	23	23	25
	Total		320	53	58	74	125	36	38	40	43

Results obtained with LCT2 for the instances of the CV dataset with $\bar{s}$ stacks and $\bar{t}$ tiers, and crane time limit τ. Column #Instances shows the number of instances per category. Columns #Optimal contains the number of instances proven optimal by the model using different running time limits. A feasible solution is obtained for every instance and running time limit. The average percentage reduction in the number of inaccessible containers in the feasible solutions obtained for each running time limit is displayed in the Reduction (%) columns. Source: Jiménez-Piqueras et al., 2024.

As expected, the number of optimal solutions increases significantly with the running time limit. However, the increase in the reduction of inaccessible containers is much less pronounced. This is because feasible solutions obtained within short running time limits are already of good quality.

For example, with a 10-minute crane time limit on the CV dataset, we observe a difference in the reduction of inaccessible containers of only 2% and 1%, respectively, for LCT1 and LCT2, when comparing the results obtained at 60 and 3600 seconds displayed in Tables 5.6 and 5.7. For the ZJY dataset, this difference is also low, at 7%, as can be deduced from Table 5.8. With a higher crane time limit of 15 minutes, the differences are slightly larger but

still small: 11% for LCT1 and 7% for LCT2 on the CV dataset, and 18% on the ZJY dataset.

Overall, there is an average reduction in the number of inaccessible containers of at least 28% for all the models, datasets, and crane time limits. The figures analyzed in this section demonstrate that using LCT1 and LCT2 provides good heuristic solutions.

Table 5.8: Performance of model LCT1 on the ZJY dataset in short running times.

τ	$\bar{s}$	$\bar{t}$	#Instances	#Optimal				Reduction (%)			
(min)				*60 s*	*120 s*	*300 s*	*3600 s*	*60 s*	*120 s*	*300 s*	*3600 s*
10	6	4	20	0	0	0	4	31	31	34	37
	7	4	20	0	0	0	0	27	29	31	31
	8	4	20	1	2	3	3	30	38	43	44
	9	4	20	0	0	0	0	28	33	36	38
	6	5	20	0	0	0	0	16	17	18	18
	Total		100	1	2	3	7	27	30	32	34
15	6	4	20	0	1	4	4	35	43	51	60
	7	4	20	0	0	0	0	27	30	32	43
	8	4	20	2	3	4	4	33	42	48	55
	9	4	20	0	0	0	2	24	30	33	46
	6	5	20	0	0	0	0	18	21	23	27
	Total		100	2	4	8	10	28	33	37	46

Results obtained with LCT1 for the instances of the ZJY dataset with $\bar{s}$ stacks and $\bar{t}$ tiers, and crane time limit τ. Column #Instances shows the number of instances per category. Columns #Optimal contains the number of instances proven optimal by the model using different running time limits. A feasible solution is obtained for every instance and any running time limit. The average percentage reduction in the number of inaccessible containers in the feasible solutions obtained for each running time limit is displayed in the Reduction (%) columns. Source: Jiménez-Piqueras et al., 2024.

5.5 Concluding remarks

The classical premarshalling formulations provide a sequence of relocations to fully arrange a bay. However, obtaining a partial bay arrangement when the crane availability is insufficient to complete the premarshalling process remained an open problem. This study introduces the Container Premarshalling Problem under Limited Crane Time (CPMP-LCT).

As a previous step to modeling the novel problem, we have addressed the Container Premarshalling Problem with Crane Time Minimization Objective (CPMPCT). We have introduced a constraint programming formulation for this problem, which outperforms the state-of-the-art mathematical programming model.

Given a crane time limit, a basic partial premarshalling can be obtained by solving the CPMP or its crane time variant, the CPMPCT, considering only the initial relocations performed within the time limit. However, we have shown that the bay configurations achieved this way are of low quality and may even worsen the initial bay configuration. In contrast, the solutions obtained through the CPMP-LCT have proven to offer significantly better partial arrangements. Moreover, the models proposed in this study to solve the CPMP-LCT provide feasible solutions for instances unsolved by the state-of-the-art models for complete premarshalling.

We have explored alternative constraint programming models for the CPMP-LCT. This chapter presents the details and computational experiments for the most relevant formulation and an alternative one tailored for scenarios where all containers in the bay have distinct priorities. Additionally, the constraint programming models proposed serve as effective heuristics for the CPMP-LCT, offering good-quality feasible solutions in short running times.

Incorporating crane times is a significant advancement toward addressing real-world scenarios in the premarshalling problem, and our constraint programming model for the CPMPCT contributes to it by yielding more efficient solutions than previous proposals. However, the essential part of this study goes beyond measuring crane times: it considers a crane time limit in the arrangement process. This enhancement allows the formulation to adapt to multiple real-life scenarios where there may not be enough time to arrange the bay entirely.

Chapter 6

Alternative approaches for the premarshalling problem under limited crane time

The introduction of the Container Premarshalling Problem under Limited Crane Time (CPMP-LCT) means a significant improvement in adapting the original premarshalling formulation to the practical requirements of port terminals. This novel version defines partial premarshalling solutions when the available time is insufficient for a complete arrangement. The goal of partial premarshalling is the same as for complete premarshalling: to best prepare the bay for the subsequent retrieval of containers. However, defining an objective that ensures this condition is not straightforward.

The objective of minimizing the number of inaccessible containers presented in Chapter 5 may be advantageous in particular scenarios, but it fails to provide sufficiently efficient partial premarshalling solutions from a general perspective. In this chapter, we explore alternative objectives for the CPMP-LCT and provide constraint programming models and solution methods tailored to each one. A computational study allows us to evaluate the solutions yielded by each approach and propose a general objective for the CPMP-LCT, highlighting the advantages of the alternatives in specific scenarios.

6.1 Alternative objectives for the CPMP-LCT

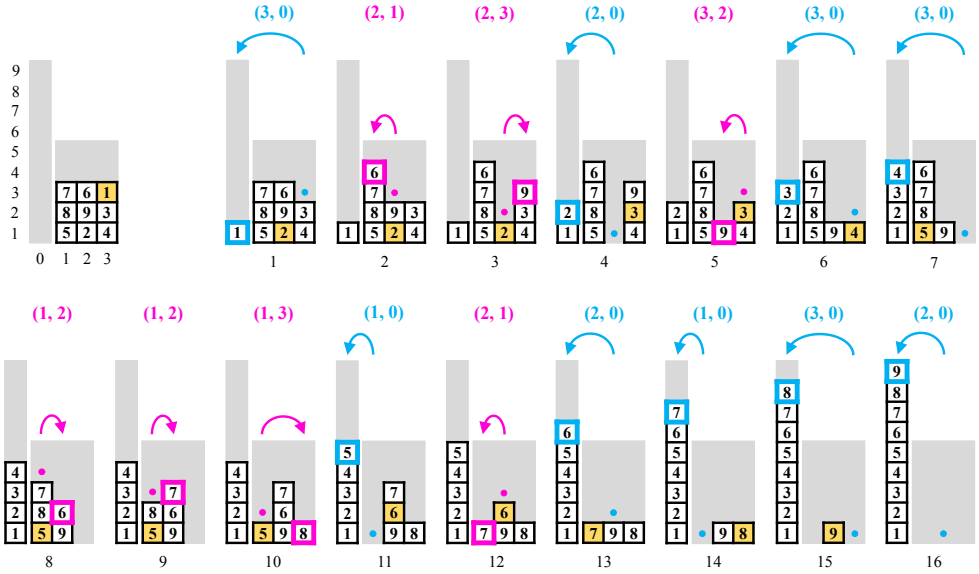
The Container Premarshalling Problem under Limited Crane Time (CPMP-LCT) aims to find a sequence of container relocations that arranges a set of containers most efficiently without exceeding a given available crane time. When a complete arrangement of the containers can be performed within the time limit, this is considered the most efficient solution if no other conditions are imposed. However, when only a partial premarshalling is possible due to crane time limitation, defining the objective to pursue for obtaining efficient solutions may be unclear.

The approach proposed in Chapter 5 for the CPMP-LCT seeks the minimization of the number of inaccessible containers in the bay. A container is considered inaccessible if one or more relocations are needed before its retrieval from the beginning of the retrieval process. Conversely, a container is accessible if this one and all containers with earlier retrieval priorities can be retrieved without additional relocations.

When minimizing the inaccessible containers, a relocation is considered more inconvenient the earlier in the retrieval process it must be performed. This objective focuses on maximizing accessibility for containers with the earliest retrieval priorities. Hence, this approach may be advantageous when possible retrieval interruptions allow for additional premarshalling before retrieving containers from later priority groups. Also, these solutions mitigate the risk of being affected by changes in the retrieval priorities, which typically have a greater impact on containers scheduled for the latest retrievals. However, in different scenarios, an alternative objective might be more beneficial or improve other aspects of the solutions.

Solving the CPMP-LCT by minimizing the inaccessible containers overlooks those that remain inaccessible after the partial premarshalling. If the subsequent container retrieval is expected to be performed uninterruptedly, a more advantageous approach might be to minimize the total number of relocations needed to retrieve the whole set of containers, instead of only focusing on those that must be retrieved first. This objective corresponds to that of the Block Relocation Problem (BRP). The BRP aims to sequence container retrievals and relocations within the bay to ensure all containers are retrieved according to the priority schedule while minimizing the overall number of relocations. An example of a BRP solution is depicted in Figure 6.1, where the nine containers in the bay are retrieved using seven relocations within the bay.

Figure 6.1: An example of a solution to the BRP.



An example of a BRP solution for an instance sourced from the CV dataset (details about this dataset can be found in Section 3.3.1). The initial bay layout is shown in the top left corner. The subsequent layouts represent the sequence of retrievals, highlighted in blue, and relocations within the bay, highlighted in magenta. On the left of the bay layouts, there is a dummy stack for placing the containers that are retrieved from the bay. This stack is identified by 0, as indicated in the initial layout, where the indexes for the real stacks and the tiers are also depicted. At each stage, the next container to be retrieved from the bay is highlighted in yellow. The relocation or retrieval performed at each stage is indicated by an arrow and a pair (s, r) where s and r are the origin and destination stacks. The container is highlighted with a thick border, and a dot indicates its origin slot. Source: Elaborated by the author (2024).

Finding a solution for the CPMP-LCT that minimizes the number of relocations in the retrieval phase entails solving the BRP to evaluate the objective function. This makes the problem highly complex and challenging to solve. Therefore, besides studying the objective of minimizing the BRP relocations, which ideally describes the efficiency of a partial premarshalling solution, we also propose an intermediate approach between minimizing inaccessible containers and minimizing relocations in the retrieval phase, aiming to combine the advantages of both objectives.

The intermediate approach between the two objectives described so far is based on two main ideas:

- ▷ Using a more straightforward objective function than that of the BRP, which still yields solutions with a similar number of relocations required during the retrieval phase.
- ▷ Reducing the number of inaccessible containers while also addressing those that remain inaccessible.

The containers that remain inaccessible after partial premarshalling require relocations to be retrieved because some of them block the retrieval of others. We consider a container is a blocking container if it is placed at a higher tier than another container in the same stack with an earlier retrieval time. While minimizing the number of blocking containers does not guarantee reducing the number of relocations in the retrieval phase to a minimum, a decrease in blocking containers often implies a reduction in the number of BRP relocations, as shown in Section 6.4. Therefore, we introduce a new term for blocking containers into the objective function of minimizing the inaccessible containers. Specifically, we define the objective of minimizing the sum of inaccessible and blocking containers.

In summary, the objective functions for the CPMP-LCT studied in this chapter are the following:

a) Minimizing the inaccessible containers: This objective focuses on unblocking the maximum number of containers in the order of retrieval priorities. It is described in Chapter 5, and we propose an enhanced solution method for it in Section 6.2.1 of this chapter.

b) Minimizing the sum of inaccessible and blocking containers: This is an intermediate approach between the other two objectives. It aims to reduce the number of inaccessible containers as in objective (a) but also addresses the containers that remain inaccessible after partial premarshalling, aiming to reduce the number of blocking containers among them, which is correlated with objective (c). We propose a constraint programming formulation and a solution method for this approach in Section 6.2.2.

c) Minimizing the BRP relocations: This is the objective of the Block Relocation Problem (BRP) and aims to minimize the number of relocations required to retrieve all the containers from the bay after premarshalling. This approach simultaneously solves the CPMP-LCT over the initial bay

layout and the BRP over the configuration resulting from premarshalling. A constraint programming model and its corresponding solutions method are presented in Section 6.2.3.

Figure 6.2 illustrates optimal final layouts resulting from solving the CPMP-LCT with the three alternative objectives on the same instance for a crane time limit of ten minutes. Some figures associated with this example are displayed in Table 6.1.

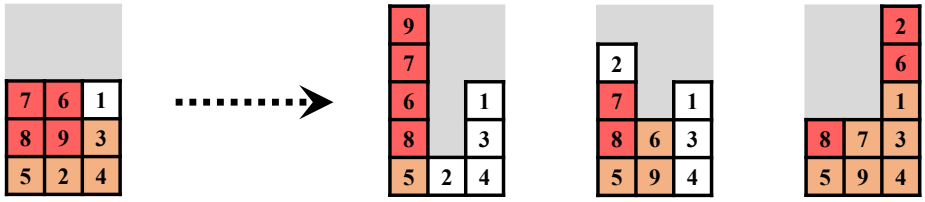
We observe that the number of inaccessible containers is reduced from 8 in the initial layout to 5 in the final layout provided by the objectives of minimizing the inaccessible containers and the sum of inaccessible and blocking containers. However, the number of inaccessible containers increases when using the objective of minimizing BRP relocations, resulting in a final layout where all the containers are inaccessible.

The objective that most reduces the number of blocking containers is (b), which includes a specific term for blocking containers. It is followed by objective (c), which provides 3 blocking containers, and then objective (a), with the same number of blocking containers as in the initial layout, 4.

The initial bay layout requires 7 relocations to retrieve all the containers from the bay. This number is reduced by all the approaches, with objectives (b) and (c) achieving the most significant reductions. By minimizing the sum of inaccessible and blocking containers or the BRP relocations, the containers can be retrieved from the layout after premarshalling using 3 relocations. However, an additional relocation is needed for the layout resulting from minimizing the inaccessible containers, making a total of 4 relocations.

This example illustrates the nature of objective (b) as an intermediate approach between objectives (a) and (c). Minimizing the sum of inaccessible and blocking containers provides the same number of inaccessible containers as only minimizing inaccessible containers but also reduces the number of blocking containers, producing a bay configuration that requires the same number of relocations to retrieve the containers as the approach of minimizing BRP relocations.

Figure 6.2: Examples of optimal final layouts for the alternative objectives.



An example of optimal final layouts for an instance sourced from the CV dataset (details about this dataset can be found in Section 3.3.1) solving the CPMP-LCT with a crane time limit of ten minutes. The initial bay layout is depicted on the left, and on the right, the figure shows the final layouts of the solutions yielded by the objectives of a) minimizing the number of inaccessible containers, b) minimizing the sum of inaccessible and blocking containers, and c) minimizing the relocations required for container retrieval. Inaccessible containers that are not blocking containers are highlighted in orange and those that are inaccessible and also blocking containers are highlighted in red. Some figures corresponding to this example are displayed in Table 6.1. Source: Elaborated by the author (2024).

Table 6.1: Features of optimal solutions for the alternative objectives.

	Initial bay layout	a) Minimize #Inaccessible	b) Minimize #Inaccessible + #Blocking	c) Minimize #BRP relocations
#Inaccessible	8	**5**	**5**	9
#Blocking	4	4	**2**	3
#BRP relocations	7	4	**3**	**3**

The table displays the number of inaccessible containers, blocking containers, and BRP relocations for the initial bay layout and the final bay layouts depicted in Figure 6.2, resulting from the objectives of a) minimizing the number of inaccessible containers, b) minimizing the sum of inaccessible and blocking containers, and c) minimizing the relocations required for container retrieval. The best values are shown in bold. Elaborated by the author (2024).

6.2 Constraint programming models and solution methods for the CPMP-LCT

This section introduces constraint programming models for the two novel objectives for the CPMP-LCT proposed in this chapter: minimizing the sum of inaccessible and blocking containers and minimizing the BRP relocations. Moreover, we have designed solution methods for each of these approaches. Additionally, we present an enhanced procedure for addressing the original objective of minimizing the inaccessible containers in the CPMP-LCT.

The solution method proposed for minimizing the inaccessible containers involves using LCT1, described in the preceding chapter in Section 5.3.2, along with an additional formulation introduced in this section, referred to as LCT-I. In the name of model LCT-I, "I" signifies Inaccessible, indicating the objective of minimizing inaccessible containers, while "LCT" denotes Limited Crane Time, as in LCT1. Similarly, we denote the constraint programming model developed for minimizing the sum of inaccessible and blocking containers as LCT-IB, with "B" representing Blocking. Finally, the model designed to minimize BRP relocations is labeled LCT-BRP.

All the models presented in this section require a fixed number of premarshalling stages, $\bar{k}$. We use the same value as specified for LCT1 in Section 5.3.4, (5.27), which is the ceiling of the quotient between the crane time limit, τ, and the minimum crane time required for a relocation, t_{min}.

6.2.1 LCT-I: A model and a solution method for minimizing the inaccessible containers

In Chapter 5, the CPMP-LCT is addressed using LCT1 to define an optimization problem and passing this formulation to a constraint programming solver. However, constraint programming solvers are generally more adept at solving satisfaction problems than optimization ones. Therefore, it may be advantageous to guide the search for an optimal solution using an iterative algorithm, employing the solver at each iteration to solve a satisfaction problem. This strategy has been successfully employed for the constraint programming models presented in Chapters 3 and 4.

In this section, we describe an algorithm designed to enhance the efficiency of the LCT1 model introduced in Section 5.3.2 for solving the CPMP-LCT with the original objective of minimizing inaccessible containers. The algorithm

consists of two phases: the first phase uses the LCT1 model, while the second phase employs a newly proposed formulation, LCT-I.

First phase: Model LCT1

In the first phase of the proposed algorithm for solving the CPMP-LCT with the objective of minimizing inaccessible containers, we use the LCT1 formulation, adapting it to define a satisfaction problem. To achieve this, we transform the objective function (5.18) into constraint (6.1), which imposes an upper limit f on the objective.

$$\sum_{s \in \mathcal{S}} \sum_{t \in \mathcal{T}} q_{s,t} \;\leq\; f \tag{6.1}$$

We denote the best objective value achieved as f^*, which is updated at each iteration of the algorithm whenever a solution is found. This parameter is initialized with the number of inaccessible containers in the initial layout, corresponding to the objective value of the trivial solution where no relocations are performed. At each iteration, the value of parameter f is set to $f = f^* - 1$, intending to either find a better solution than the current best or identify infeasibility.

Specifically, the first iteration of the algorithm aims to solve the problem with a maximum number of inaccessible containers f or demonstrate the optimality of the trivial solution. If a solution is discovered, the parameters f and f^* are updated, and a new iteration is performed. This procedure is repeated whenever a new solution is found. However, if an infeasible problem is encountered during any iteration, the algorithm terminates, as the best solution has been proven optimal.

Additionally, the first phase of the algorithm incorporates a fail limit per iteration. This prevents the algorithm from becoming trapped in an iteration where the solver cannot either find a feasible solution to the problem or prove its infeasibility. If the fail limit is reached, the algorithm proceeds to the second phase.

Second phase: Model LCT-I

The purpose of the second phase is to enhance, or ideally, prove the optimality of the best solution identified thus far when the first phase seems to not progress in this regard. The idea behind the second phase is to address the CPMP-LCT from a different perspective, transforming the partial premarshalling problem to be solved at each iteration into a complete premarshalling problem. Based on this concept, we formulate the LCT-I model.

The LCT-I model is designed to address satisfaction problems. Hence, no objective function is defined for it. It is derived from the formulation for complete premarshalling MCT, including all the constraints from this model, (5.2) to (5.17), except for constraints (5.9) that identify badly placed containers. Instead, constraints (5.9) are replaced by (6.2), along with (6.3) if there are repeated priorities in the bay, as detailed in subsequent paragraphs.

Additionally, LCT-I incorporates constraints (5.20) and (5.23) from LCT1. Constraints (5.20) are included in LCT-I for the same purpose as in LCT1: discarding equivalent solutions to narrow down the search space. Constraints (5.23) impose the crane time limit τ. However, in contrast to LCT1, a model for partial premarshalling, LCT-I is a formulation for complete premarshalling. Consequently, in LCT-I, the limit τ does not restrict the number of relocations to perform but solely determines the feasibility of achieving a complete arrangement: if the time needed for complete premarshalling exceeds the limit, the problem is deemed infeasible.

Constraints (6.2), and optionally (6.3), are the key for translating the partial premarshalling targeted in each iteration into a complete premarshalling problem. These constraints ensure that the limit on inaccessible containers imposed by (6.1) in the first phase is also incorporated in the second phase.

$$\left(x^k_{s,t+1} > x^k_{s,t}\right) \cdot \left(x^k_{s,t} < \widehat{p}\right) \leq w^k_{s,t+1} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T} \setminus \{\bar{t}\},\, k \in \mathcal{K} \tag{6.2}$$

$$\sum_{s \in \mathcal{S},\, t \in \mathcal{T} \setminus \{\bar{t}\}} \left(\left(x^k_{s,t+1} > x^k_{s,t}\right) - w^k_{s,t+1}\right) \cdot \left(x^k_{s,t} == \widehat{p}\right) \leq \gamma \qquad \forall k \in \mathcal{K} \tag{6.3}$$

In LCT-I, there are no variables for identifying the inaccessible containers as in LCT1. Instead, the LCT-I model focuses on badly placed containers, similar to other formulations for complete premarshalling. However, constraints (6.2) and (6.2) allow the connection between these two types of containers using parameter $\hat{p}$.

The original constraints (5.9) from MCT indicate that if a container in slot (s, t) has an earlier retrieval time than a container in slot $(s, t+1)$, then the container in $(s, t+1)$ is badly placed. Conversely, the new constraints (6.2), replacing (5.9), express that if a container in slot (s, t) has a retrieval time $p < \hat{p}$ and p is an earlier retrieval time than that of a container occupying $(s, t+1)$, then the container in $(s, t+1)$ is badly placed since it blocks the one in (s, t).

In the first phase, constraints (6.1) permit f containers to remain inaccessible. In the second phase, constraints (6.2) allow the containers with priorities $\hat{p}$ to $\bar{p}$ to remain inaccessible by neglecting the blockages between these containers. This can be interpreted as if the last f containers to be retrieved were regrouped into a single priority group, thus not blocking each other.

If all the containers in the bay have distinct priorities, the f last containers to be retrieved correspond to the f last priority groups, i.e., priority groups $\hat{p} = \bar{p} - f + 1$ to $\bar{p}$. However, if there are repeated priorities, $\hat{p}$ would take the priority of the $\bar{p} - f + 1$ container to be retrieved, which may not be the value $\bar{p} - f + 1$. In this case, if the priority group assigned to $\hat{p}$ has multiple containers and the f last containers to be retrieved do not include all of them, (6.3) must be incorporated into the formulation. Let γ be the number of containers in priority group $\hat{p}$ that are part of the f containers that are permitted to remain inaccessible, then constraints (6.3) impose that the blockages on at most γ containers with priority $\hat{p}$ can be neglected.

The second phase of the algorithm begins by assigning $\hat{p}$ the priority of the $\bar{p} - f + 1$ container to be retrieved, where $f = f^* - 1$, and f^* corresponds to the number of inaccessible containers in the best solution found during the first phase. Using this value for $\hat{p}$, the LCT-I model is employed to solve the problem of obtaining a partial premarshalling with at most f inaccessible containers. If a solution is found, the parameters f^*, f, and $\hat{p}$ are updated, and a new iteration is performed. If an infeasible problem is encountered at some point, the last solution obtained is proven optimal.

6.2.2 LCT-IB: A model and a solution method for minimizing the sum of inaccessible and blocking containers

We propose the LCT-IB model to solve the CPMP-LCT with the objective of minimizing the sum of inaccessible and blocking containers. The LCT-IB formulation results from modifying the LCT1 formulation, presented in Section 5.3.2, to include the blocking containers in the objective function.

LCT-IB contains all the groups of decision variables from LCT1, but with $q_{s,t}$ turning into integer variables to accommodate the blocking containers.

$$q_{s,t} = \begin{cases} 2 & \text{If there is an inaccessible and blocking container in slot } (s,t) \text{ in the final stage} \\ 1 & \text{If there is an inaccessible but not blocking container in slot } (s,t) \text{ in the final stage} \\ 0 & \text{If slot } (s,t) \text{ is empty or it is occupied by an accessible container in the final stage} \end{cases}$$

$$\forall s \in \mathcal{S},\ \forall t \in \mathcal{T}$$

The objective function of LCT-IB is analogous to (5.18) from LCT1, but uses the integer variables $q_{s,t}$. All the constraints in LCT1 are included in LCT-IB, with one group of constraints slightly modified and an additional group of constraints added.

Constraints (5.21) and (5.22) from LCT1 ensure that variables $q_{s,t}$ are positive when there is an inaccessible container in slot (s,t). This condition must also be imposed in LCT-IB, but the parameter $\bar{c}$ in (5.22) has to be multiplied by 2 in LCT-IB. This adjustment prevents the variables $q_{s,t}$ on the right-hand side of the inequality from being incorrectly forced to be equal to 2. Consequently, constraints (5.22) from LCT1, are substituted with (6.4) in LCT-IB.

A new group of constraints, (6.5), is included in LCT-IB to ensure a variable $q_{s,t}$ takes a value of 2 when, in any tier below t within stack s, there is a container with an earlier retrieval time than the container in slot (s,t). This condition indicates the presence of a blocking container in slot (s,t).

$$\sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{T}} (x^{\bar{k}}_{s,t} > x^{\bar{k}}_{ij}) \cdot q_{i,j} \leq 2\,\bar{c} \cdot q_{s,t} \qquad \forall (s,t) \in \mathcal{L} \tag{6.4}$$

$$\sum_{j \in \mathcal{T}: j < t} (x^{\bar{k}}_{s,t} > x^{\bar{k}}_{s,j}) \leq \bar{t} \cdot \big(q_{s,t} == 2\big) \qquad \forall (s,t) \in \mathcal{L} : t > 1 \tag{6.5}$$

Solving method

We propose using LCT-IB along with an iterative algorithm equivalent to the first phase of the algorithm described in Section 6.2.1. At each stage, we solve a satisfaction problem by replacing the objective function with constraint (6.1) that imposes an upper limit f on it, as described for LCT1 in the previous section. The only difference is that in LCT-IB, the limit f is applied to the sum of inaccessible and blocking containers instead of only the number of inaccessible containers.

The algorithm used for LCT-IB comprises a single phase, and the fail limit applied in the first phase of the algorithm described in Section 6.2.1 is not used here.

The parameter f^*, representing the best objective value found, is initialized with the value of the objective function in the initial bay layout, corresponding to the trivial solution of no relocations. The algorithm then performs a first iteration with an upper limit on the objective function set to $f = f^* - 1$. Whenever a solution is found, f^* and f are updated, and a new iteration is conducted. When an infeasible problem is encountered, the last solution found is proven optimal. Otherwise, if the algorithm stops due to the running time limit, it returns the best feasible solution obtained up to that point.

6.2.3 LCT-BRP: A model and a solution method for minimizing the relocations in the retrieval phase

We have developed the model LCT-BRP to solve the CPMP-LCT using the objective of the BRP. This approach aims to find a partial premarshalling solution within a limited crane time that minimizes the number of relocations needed during the subsequent retrieval of all containers in the bay.

The LCT-BRP model solves the CPMP-LCT and the BRP altogether. Thus, a solution comprises a sequence of relocations performed during the premarshalling phase (corresponding to CPMP-LCT) and the subsequent sequence of retrievals and relocations conducted during the retrieval phase (corresponding to BRP). The set of stages is extended according to this requirement, using the notation described in the following subsection, which also explains how the bay configuration is adapted to describe the sequence of movements in the retrieval phase.

Notation

A solution to the CPMP-LCT with the objective of minimizing the BRP relocations consists of two phases: the premarshalling phase and a subsequent retrieval phase. The set of stages used in all models presented in this thesis, $\mathcal{K}$, includes stages corresponding to these two phases in the LCT-BRP formulation. As in previous models, $\bar{k}$ indicates the total number of stages, and additionally, $\bar{k}_1$ denotes the number of stages in the first phase, corresponding to premarshalling. We use the following sets of stages:

- General sets, including or not the initial stage: $\mathcal{K}^0 := \{0, 1, 2, \ldots, \bar{k}\}$ and $\mathcal{K} := \{1, 2, \ldots, \bar{k}\}$.
- Sets of stages corresponding to the premarshalling phase, including or not the initial stage: $\mathcal{K}_1^0 = \{0, 1, \ldots, \bar{k}_1\}$ and $\mathcal{K}_1 = \{1, \ldots, \bar{k}_1\}$.
- A set of stages corresponding to the retrieval phase: $\mathcal{K}_2 = \{\bar{k}_1 + 1, \ldots, \bar{k}\}$.

The bay configuration is extended during the retrieval phase to account for container retrievals. Specifically, a dummy stack denoted by 0 is considered to place the containers retrieved from the bay. The number of tiers in stack 0 is equal to the number of containers in the bay, $\bar{c}$, as all of them must be retrieved. We define the set of all stacks, including the retrieval one, as $\mathcal{L}^0 := (\{0\} \times \mathcal{C}) \cup \mathcal{L}$, where $\mathcal{C} = \{1, \ldots, \bar{c}\}$ is the set of containers and $\mathcal{L} := \mathcal{S} \times \mathcal{T}$ corresponds to the set of bay slots.

The parameter m_p indicates the number of containers in priority group p in this model, similar to previous ones. However, for the particular case of $p = 0$, while m_0 counts the empty slots in the premarshalling phase, it is replaced by $m_0^0 := m_0 + \bar{c}$ in the second stage to incorporate the empty slots of stack 0.

Formulation

The formulation LCT-BRP is derived from the LCT1 model presented in Section 5.3.2. The sets of variables from LCT1 are extended in LCT-BRP to include the stack for the retrieval of containers and the stages corresponding to the retrieval phase. Variables $q_{s,t}$ that identify the inaccessible containers in LCT1 are omitted in LCT-BRP, but an additional group of variables is defined, $b_{s,t}$, to identify the blocking containers at the end of the premarshalling phase:

$$b_{s,t} = \begin{cases} 1 & \text{If there is a blocking container in slot } (s,t) \text{ in stage } \bar{k}_1 \\ 0 & \text{Otherwise} \end{cases}$$

$$\forall (s,t) \in \mathcal{L}$$

In LCT-BRP, the variables that describe the bay configuration, $\delta^k_{s,t}$ and $x^k_{s,t}$, are defined for all $(s,t,k) \in \left(\mathcal{L} \times \mathcal{K}^0_1\right) \cup \left(\mathcal{L}^0 \times \mathcal{K}_2\right)$.

Variables $y^k_{s,t}$, which indicate the slot (s,t) where the container relocated at stage k is placed, are extended to stack 0 in the retrieval phase, so they are defined for all $(s,t,k) \in \left(\mathcal{L}\times\mathcal{K}_1\right)\cup\left(\mathcal{L}^0\times\mathcal{K}_2\right)$. No container can be retrieved from stack 0, so variables $z^k_{s,t}$ are not defined in that stack, but for all $(s,t,k) \in \mathcal{L}\times\mathcal{K}$.

The sets of variables that describe the crane movements, $g^k_{s,r}$ and $u^k_{s,r}$, are not extended because they are only needed in the premarshalling phase.

The objective function of minimizing the number of relocations in the retrieval phase can be expressed as (6.6), using variables y over the set of bay slots during the retrieval phase. The destination slots of the retrievals performed in this phase are in stack 0, so they are not counted in the objective function.

$$\min \sum_{(s,t,k)\in\mathcal{L}\times\mathcal{K}_2} y^k_{s,t} \tag{6.6}$$

The constraints for initializing variables x and δ, namely (6.7) and (6.8), remain unchanged from LCT1.

$$x^0_{s,t} = \alpha^x_{s,t} \qquad \forall (s,t) \in \mathcal{L} \tag{6.7}$$

$$\delta^0_{s,t} = \alpha^\delta_{s,t} \qquad \forall (s,t) \in \mathcal{L} \tag{6.8}$$

The LCT-BRP is designed by adapting most of the constraints of LCT1 to the new sets of stages and stack, replacing some of them with new ones, and omitting those related to variables $q_{s,t}$, namely (5.21) and (5.22). The entire set of constraints for LCT-BRP is displayed below, along with Table 6.2, which indicates the correspondence between the constraints from LCT-BRP and LCT1.

Table 6.2: Correspondence between constraints from models LCT-BRP and LCT1.

LCT-BRP	LCT1
(6.9), (6.10), (6.11)	(5.6)
(6.12)	(5.7)
(6.13)	(5.8)
(6.18), (6.19)	(5.12)
(6.20)	(5.13)
(6.21)	(5.20)
(6.22)	(5.16)
(6.23)	(5.17)
(6.24), (6.25)	(5.14)
(6.26), (6.27), (6.28), (6.29)	(5.19)
(6.30)	(5.23)

The left column shows the constraints of LCT-BRP, replacing the corresponding constraints from LCT1 indicated in the right column. Source: Elaborated by the author (2024).

$$Count\big(x^k_{s,t} : (s,t) \in \mathcal{L},\, x^k_{s,t} = p\big) = m_p \qquad \forall p \in \mathcal{P}^0,\, k \in \mathcal{K}_1 \tag{6.9}$$

$$Count\big(x^k_{s,t} : (s,t) \in \mathcal{L}^0,\, x^k_{s,t} = p\big) = m_p \qquad \forall p \in \mathcal{P},\, k \in \mathcal{K}_2 \tag{6.10}$$

$$Count\big(x^k_{s,t} : (s,t) \in \mathcal{L}^0,\, x^k_{s,t} = 0\big) = m^0_0 \qquad \forall p \in \mathcal{P}^0,\, k \in \mathcal{K}_2 \tag{6.11}$$

$$\big(x^k_{s,t} > 0\big) = \delta^k_{s,t} \qquad \forall (s,t,k) \in \big(\mathcal{L} \times \mathcal{K}_1\big) \cup \big(\mathcal{L}^0 \times \mathcal{K}_2\big) \tag{6.12}$$

$$\big(x^k_{s,t} == x^{k-1}_{s,t}\big) = \big(\delta^k_{s,t} == \delta^{k-1}_{s,t}\big) \qquad \forall (s,t,k) \in \big(\mathcal{L} \times (\mathcal{K}_1 \cup \{\bar{k}_1 + 1\})\big) \cup \big(\mathcal{L}^0 \times (\mathcal{K}_2 \setminus \{\bar{k}_1 + 1\})\big) \tag{6.13}$$

$$\big(x^k_{0,t} > 0\big) \;\leq\; \big(x^k_{0,t-1} \leq x^k_{0,t}\big) \qquad \forall t \in \mathcal{C} \setminus \{1\},\; k \in \mathcal{K}_2 \setminus \{\bar{k}_1 + 1\} \tag{6.14}$$

$$\sum_{s \in \mathcal{S}, t \in \mathcal{T}} \delta^k_{s,t} \;+\; k \;\leq\; \bar{k} \qquad \forall k \in \mathcal{K}_2 \tag{6.15}$$

$$\sum_{s \in \mathcal{S},\, t \in \mathcal{T}} b_{s,t} \;\leq\; \bar{k} - \bar{k}_1 - \bar{c} \tag{6.16}$$

$$b_{s,t} = \left(\Big(\sum_{j \in \mathcal{T}: j<t} (x^{\bar{k}_1}_{s,t} > x^{\bar{k}_1}_{s,j}) \Big) > 0 \right) \quad \forall s \in \mathcal{S},\, t \in \mathcal{T} \setminus \{1\} \tag{6.17}$$

$$\sum_{(s,t) \in \mathcal{L}} y^k_{s,t} \leq 1 \quad \forall k \in \mathcal{K}_1 \tag{6.18}$$

$$\sum_{(s,t) \in \mathcal{L}^0} y^k_{s,t} \leq 1 \quad \forall k \in \mathcal{K}_2 \tag{6.19}$$

$$\sum_{(s,t) \in \mathcal{L}} z^k_{s,t} \leq 1 \quad \forall k \in \mathcal{K} \tag{6.20}$$

$$\sum_{(s,t) \in \mathcal{L}} z^{k+1}_{s,t} \leq \sum_{(s,t) \in \mathcal{L}} z^k_{s,t} \quad \forall k \in \big(\mathcal{K}_1 \setminus \{\bar{k}_1\}\big) \cup \big(\mathcal{K}_2 \setminus \{\bar{k}\}\big) \tag{6.21}$$

$$g^k_{s,r} = \left(\sum_{t \in \mathcal{T}} \big(z^k_{s,t} + y^k_{r,t}\big) == 2 \right) \quad \forall s \in \mathcal{S},\, r \in \mathcal{S} \setminus \{s\},\, k \in \mathcal{K} \tag{6.22}$$

$$u^k_{s,r} = \left(\sum_{t \in \mathcal{T}} \big(y^{k-1}_{s,t} + z^k_{r,t}\big) == 2 \right) \quad \forall s \in \mathcal{S},\, r \in \mathcal{S} \setminus \{s\},\, k \in \mathcal{K} \setminus \{1\} \tag{6.23}$$

$$y^k_{s,t} \leq \delta^{k+1}_{s,t} \quad \forall (s,t) \in \mathcal{L},\, k \in \mathcal{K} \setminus \{\bar{k}\} \tag{6.24}$$

$$\sum_{i \in \mathcal{K}_2: i<k} y^i_{0,t} \leq \delta^k_{0,t} \quad \forall t \in \mathcal{C},\, k \in \mathcal{K}_2 \setminus \{\bar{k}_1 + 1\} \tag{6.25}$$

$$\begin{aligned} &AllowedAssignments\big(C, \\ &\delta^{k-1}_{s,t}, \delta^{k-1}_{s,t+1}, \delta^k_{s,t}, \delta^k_{s,t+1}, \\ &z^k_{s,t}, z^k_{s,t+1}, y^k_{s,t}, y^k_{s,t+1}\big) \end{aligned} \quad \forall s \in \mathcal{S},\, t \in \mathcal{T} \setminus \{\bar{t}\},\, k \in \mathcal{K} \tag{6.26}$$

$$\delta^{\bar{k}_1+1}_{0,t} = 0 \quad \forall t \in \mathcal{C} \setminus \{1\} \tag{6.27}$$

$$\delta^{\bar{k}_1+1}_{0,t} = y^{\bar{k}_1+1}_{0,t} \quad \forall t \in \mathcal{C} \tag{6.28}$$

$$\begin{aligned} &AllowedAssignments\big(D, \\ &\delta^{k-1}_{0,t}, \delta^{k-1}_{0,t+1}, \delta^k_{0,t}, \delta^k_{0,t+1}, y^k_{0,t}, y^k_{0,t+1}\big) \end{aligned} \quad \forall t \in \mathcal{C} \setminus \{\bar{c}\},\, k \in \mathcal{K}_2 \setminus \{\bar{k}_1 + 1\} \tag{6.29}$$

$$\sum_{s\in\mathcal{S}}\left(\sum_{\substack{r\in\mathcal{S}\\ r\neq s}}\left(h^0_{0,s}\cdot g^1_{s,r}+\sum_{k\in\mathcal{K}\setminus\{1\}}\left(h^0_{r,s}\cdot u^k_{r,s}\right)+\sum_{k\in\mathcal{K}}\left(h^1_{s,r}\cdot g^k_{s,r}\right)\right)+\right.$$
$$\left.\sum_{k\in\mathcal{K}}\sum_{t\in\mathcal{T}}\left(\left(v_t+l_t\right)\cdot z^k_{s,t}+v_t\cdot y^k_{s,t}\right)\right)\leq\tau \tag{6.30}$$

LCT-BRP contains three groups of Count constraints to ensure the number of containers in each priority group and the number of empty slots remain constant. Constraints (6.9) impose this condition during the premarshalling phase. Constraints (6.10) and (6.11) correspond to the BRP phase, with (6.10) ensuring the correct number of containers with a given priority and (6.10) maintaining constant the number of empty slots, including the dummy stack for retrievals.

Positive values of variables x must correspond to the value 1 for variables δ, as expressed by (6.12). Constraints (6.13) ensure that the priority associated with a slot does not change between stages if no container is placed or removed.

The formulation includes constraints (6.14) to ensure that retrievals are performed according to the priority schedule. These constraints indicate that if a slot in stack 0 is occupied, the container below it must belong to the same priority group or be assigned an earlier retrieval.

An additional group of constraints, (6.15), is introduced to detect infeasibilities faster. These constraints express that the number of containers in the bay at stage k during the retrieval phase must be less or equal to the number of remaining stages. A solution violating this constraint is infeasible because, at most, one container can be retrieved per stage.

In the retrieval phase, some relocations may be necessary, and the number of blocking containers in the bay layout after premarshalling is a lower bound to this number of relocations. This lower bound is imposed by (6.16), with variables b defined in terms of variables x through constraints (6.17). These constraints ensure a variable $b_{s,t}$ takes the value 1 when there is a blocking container in slot (s,t), i.e., when a container at a lower tier of the same stack has an earlier retrieval priority.

The condition allowing at most one movement per stage is defined similarly to LCT1 for the premarshalling phase by (6.18) and (6.20). However, in the retrieval phase, it must be extended to account for the retrievals from the bay. To

this end, (6.19) is incorporated using variables y to identify the retrievals, i.e., the container movements with destination stack 0. Additionally, constraints (6.21) indicate that a container movement cannot be performed at a stage unless another movement is conducted in the previous one, except for stage $\bar{k}_1$, because no movement may be performed during the last premarshalling stages.

The constraints (6.22) and (6.23) define the crane movements for the set of stages $\mathcal{K}_1$, as the crane time must be measured in the premarshalling phase to impose on it the upper limit defined by (6.30). Variables $g^k_{s,r}$ indicate when the crane's trolley moves loaded with a container from stack s to stack r, i.e., $g^k_{s,r}$ must be equal to 1 when a container is retrieved from stack s and placed in stack r, as expressed by (6.22). Variables $u^k_{s,r}$ must be equal to 1 when the trolley moves unloaded from stack s to stack r, which occurs when a container was placed in stack s in $k-1$, and a container must be retrieved from stack r at k, as indicated by (6.23).

The conditions relating bay configuration variables with variables identifying the container movements are expressed by (6.24) to (6.29) in LCT-BRP.

Constraints (6.24) ensure that the slot where a container is placed remains occupied in the subsequent stage. This condition can be strengthened for stack 0 because when a container is retrieved from the bay and placed in stack 0, that movement is permanent. Hence, (6.25) is added to express that a slot in stack 0 where a container is placed must remain occupied during all the following stages.

The table constraints (6.26) are the same as (5.19) from LCT1 but for the extended set of stages $\mathcal{K} = \mathcal{K}_1 \cup \mathcal{K}_2$, and constraints (6.27) to (6.29) are incorporated for stack 0. The Allowed Assignments constraint (6.26) determines the connection between the relocations of containers with the definition of slots as empty or occupied and ensures containers are relocated to allowed slots, avoiding empty spaces between containers in a stack.

All the slots in stack 0 must be empty in the first stage of the retrieval phase, except maybe the one at the first tier. Variables δ are initialized with this information by (6.27). The slot at the first tier must be occupied in the first stage if a container is moved there and empty otherwise, as indicated by (6.28). Table constraints (6.29) correspond to (6.26) for stack 0. Variables z are not included in (6.29) because this group is not defined for stack 0, as no container can be retrieved from that stack.

There are two main versions of the Block Relocation Problem: the restricted and the unrestricted. In the restricted version, an additional condition is imposed on the containers that can be elected to be relocated at each stage. Only those that block the retrieval of the following container needed to be retrieved can be relocated. This condition is not considered in the unrestricted version, where any container at the top of each stack can be relocated. The condition for the restricted version in LCT-BRP is imposed by (6.31).

$$\sum_{t \in \mathcal{T}} z_{s,t}^{k} \;\leq\; \sum_{t \in \mathcal{T}} z_{s,t}^{k+1} + \sum_{t \in \mathcal{C}} y_{0,t}^{k} \qquad \forall s \in \mathcal{S},\, k \in \mathcal{K}_2 \setminus \{\bar{k}\} \tag{6.31}$$

Solution method

Similarly to the algorithms previously described for the other approaches, the search for a solution using the LCT-BRP model is also guided by an interactive algorithm, solving a satisfaction problem with a different upper bound on the objective value at each iteration.

However, instead of including a constraint limiting the objective value, in this case, the number of BRP relocations is limited by the number of stages defined for the relocation phase. A total of $\bar{c}$ retrieval movements are needed in the BRP phase, so the rest of the stages provided for this phase can be used for relocations inside the bay. Hence, by fixing the number of stages in the retrieval phase, we impose an upper limit on the BRP relocations, which is the objective value.

When using the algorithm, constraints (6.19) are substituted with (6.32), and (6.20) are replaced with (6.33), but only for the stages in $\mathcal{K}_2$, ensuring exactly one movement per stage in the retrieval phase. Also, the objective function (6.6) is omitted.

$$\sum_{(s,t) \in \mathcal{L}^0} y_{s,t}^{k} \;=\; 1 \qquad \forall k \in \mathcal{K}_2 \tag{6.32}$$

$$\sum_{(s,t) \in \mathcal{L}} z_{s,t}^{k} \;=\; 1 \qquad \forall k \in \mathcal{K}_2 \tag{6.33}$$

The algorithm begins by solving the problem with a maximum number of relocations in the retrieval phase equal to the number of blocking containers in the initial bay layout. If a solution is found, a binary search is conducted

to find a solution with a smaller objective value or to prove the optimality of the best solution found. In case the problem solved in the first iteration is infeasible, the number of stages for the retrieval phase is increased by one, and a new iteration is performed. This process is repeated until an optimal solution is discovered or the running time limit is reached.

6.3 A constraint programming model for the Block Relocation Problem

One of the objectives proposed for the CPMP-LCT in this chapter is to minimize the number of relocations in the retrieval phase after premarshalling, which corresponds to the objective of the Block Relocation Problem (BRP).

To delve into the differences between solutions obtained using the alternative objectives studied, we compare them in terms of inaccessible containers, blocking containers and BRP relocations in Section 6.4. The number of inaccessible containers and blocking containers can be directly calculated for every solution. However, obtaining the number of relocations needed to retrieve the containers from the bay requires solving the BRP. For this purpose, we present a constraint programming model for solving the BRP, denoted as CP-BRP.

CP-BRP is equivalent to the part of LCT-BRP related to BRP. Hence, the set of stages $\mathcal{K} = \{1, \ldots, \bar{k}\}$ used in CP-BRP corresponds to the stages $\mathcal{K}_2 = \{\bar{k}_1 + 1, \ldots, \bar{k}\}$ in LCT-BRP.

The variables of CP-BRP are those from LCT-BRP except for $g^k_{s,r}$ and $u^k_{s,r}$, which are only used in the premarshalling part of LCT-BRP. Additionally, variables $b_{s,t}$ from LCT-BRP incorporate a new index in CP-BRP, $b^k_{s,t}$, being defined for every stage.

All the variables involved in the CP-BRP model are displayed on the next page, followed by the constraints of the formulation and Table 6.3, which indicates the correspondence between constraints from CP-BRP and LCT-BRP. All the constraints from LCT-BRP have been considered for building the CP-BRP, except for those involving variables $g^k_{s,r}$ and $u^k_{s,r}$, namely (6.22), (6.23) and (6.30). The objective function of the CP-BRP model, (6.34), is equivalent to that of the LCT-BRP.

$$\min \sum_{(s,t,k) \in \mathcal{L} \times \mathcal{K}} y^k_{s,t} \tag{6.34}$$

$$x_{s,t}^{k} = \begin{cases} p & \text{If a container with priority } p \text{ is in slot } (s,t) \text{ at stage } k \\ 0 & \text{If slot } (s,t) \text{ is empty at stage } k \end{cases}$$

$$\forall (s,t) \in \mathcal{L}^0,\ k \in \mathcal{K}^0$$

$$\delta_{s,t}^{k} = \begin{cases} 1 & \text{If there is a container in slot } (s,t) \text{ at stage } k \\ 0 & \text{If slot } (s,t) \text{ is empty at stage } k \end{cases}$$

$$\forall (s,t) \in \mathcal{L}^0,\ k \in \mathcal{K}^0$$

$$b_{s,t}^{k} = \begin{cases} 1 & \text{If there is a blocking container in slot } (s,t) \text{ at stage } k \\ 0 & \text{Otherwise} \end{cases}$$

$$\forall (s,t) \in \mathcal{L} : t > 1\, k \in \mathcal{K} \setminus \{\bar{k}\}$$

$$y_{s,t}^{k} = \begin{cases} 1 & \text{If a container is moved to slot } (s,t) \text{ during stage } k \\ 0 & \text{Otherwise} \end{cases}$$

$$\forall (s,t) \in \mathcal{L}^0,\ k \in \mathcal{K}$$

$$z_{s,t}^{k} = \begin{cases} 1 & \text{If a container is removed from slot } (s,t) \text{ during stage } k \\ 0 & \text{Otherwise} \end{cases}$$

$$\forall (s,t) \in \mathcal{L},\ k \in \mathcal{K}$$

Table 6.3: Correspondence between constraints from models CP-BRP and LCT-BRP.

CP-BRP	LCT-BRP	CP-BRP	LCT-BRP
(6.35)	(6.7)	(6.45)	(6.20)
(6.36)	(6.8)	(6.46)	(6.21)
(6.37)	(6.9), (6.10), (6.11)	(6.47)	(6.24)
(6.38)	(6.12)	(6.48)	(6.25)
(6.39)	(6.13)	(6.49)	(6.26)
(6.40)	(6.14)	(6.50)	(6.27)
(6.41)	(6.15)	(6.51)	(6.28)
(6.42)	(6.16)	(6.52)	(6.29)
(6.43)	(6.17)	(6.53)	(6.31)
(6.44)	(6.18), (6.19)		

The columns labeled CP-BRP show the constraints of this model replacing the corresponding constraints from LCT-BRP. Source: Elaborated by the author (2024).

$$x^0_{s,t} \;=\; \alpha^x_{s,t} \qquad \forall (s,t) \in \mathcal{L}^0 \tag{6.35}$$

$$\delta^0_{s,t} \;=\; \alpha^\delta_{s,t} \qquad \forall (s,t) \in \mathcal{L}^0 \tag{6.36}$$

$$Count\big(x^k_{s,t} : (s,t) \in \mathcal{L}^0,\, x^k_{s,t} = p\big) = m_p \qquad \forall p \in \mathcal{P}^0,\, k \in \mathcal{K} \tag{6.37}$$

$$\big(x^k_{s,t} > 0\big) = \delta^k_{s,t} \qquad \forall (s,t) \in \mathcal{L}^0,\, k \in \mathcal{K} \tag{6.38}$$

$$\big(x^k_{s,t} == x^{k-1}_{s,t}\big) = \big(\delta^k_{s,t} == \delta^{k-1}_{s,t}\big) \qquad \forall (s,t) \in \mathcal{L}^0,\, k \in \mathcal{K} \tag{6.39}$$

$$\big(x^k_{0,t} > 0\big) \;\leq\; \big(x^k_{0,t-1} \leq x^k_{0,t}\big) \qquad \forall t \in \mathcal{C} \setminus \{1\},\, k \in \mathcal{K} \setminus \{1\} \tag{6.40}$$

$$\sum_{s \in \mathcal{S}, t \in \mathcal{T}} \delta^k_{s,t} \;+\; k \;\leq\; \bar{k} \qquad \forall k \in \mathcal{K} \tag{6.41}$$

$$\sum_{s \in \mathcal{S},\, t \in \mathcal{T} \setminus \{1\}} b^k_{s,t} + k + \sum_{(s,t) \in \mathcal{L}} \delta^k_{s,t} \;\leq\; \bar{k} \qquad \forall k \in \mathcal{K} \setminus \{\bar{k}\} \tag{6.42}$$

$$b^k_{s,t} \;=\; \left(\Big(\sum_{j \in \mathcal{T} : j < t} (x^k_{s,t} > x^k_{s,j}) \Big) > 0 \right) \qquad \forall (s,t) \in \mathcal{L} : t \;>\; 1,\; k \in \mathcal{K} \setminus \{\bar{k}\} \tag{6.43}$$

$$\sum_{(s,t) \in \mathcal{L}^0} y^k_{s,t} \;\leq\; 1 \qquad \forall k \in \mathcal{K} \tag{6.44}$$

$$\sum_{(s,t) \in \mathcal{L}} z^k_{s,t} \;\leq\; 1 \qquad \forall k \in \mathcal{K} \tag{6.45}$$

$$\sum_{(s,t) \in \mathcal{L}} z^{k+1}_{s,t} \;\leq\; \sum_{(s,t) \in \mathcal{L}} z^k_{s,t} \qquad \forall k \in \mathcal{K} \setminus \{\bar{k}\} \tag{6.46}$$

$$y^k_{s,t} \;\leq\; \delta^{k+1}_{s,t} \qquad \forall (s,t) \in \mathcal{L},\, k \in \mathcal{K} \setminus \{\bar{k}\} \tag{6.47}$$

$$\sum_{i \in \mathcal{K} : i < k} y^i_{0,t} \;\leq\; \delta^k_{0,t} \qquad \forall t \in \mathcal{C},\, k \in \mathcal{K} \setminus \{1\} \tag{6.48}$$

$$\begin{aligned} &AllowedAssignments\big(C,\\ &\delta^{k-1}_{s,t}, \delta^{k-1}_{s,t+1}, \delta^k_{s,t}, \delta^k_{s,t+1},\\ &z^k_{s,t}, z^k_{s,t+1}, y^k_{s,t}, y^k_{s,t+1}\big) \end{aligned} \qquad \forall s \in \mathcal{S},\, t \in \mathcal{T} \setminus \{\bar{t}\},\, k \in \mathcal{K} \tag{6.49}$$

$$\delta^1_{0,t} = 0 \qquad \forall t \in \mathcal{C} \setminus \{1\} \tag{6.50}$$

$$\delta^1_{0,t} = y^1_{0,t} \qquad \forall t \in \mathcal{C} \tag{6.51}$$

$$\begin{aligned} &AllowedAssignments\big(D,\\ &\delta^{k-1}_{0,t}, \delta^{k-1}_{0,t+1}, \delta^k_{0,t}, \delta^k_{0,t+1}, y^k_{0,t}, y^k_{0,t+1}\big) \end{aligned} \qquad \forall t \in \mathcal{C} \setminus \{\bar{c}\},\, k \in \mathcal{K} \setminus \{1\} \tag{6.52}$$

Compared to the LCT-BRP, the only difference in the constraints describing the bay layout through variables x and δ, namely (6.35) to (6.41), is the set of bay slots and the stages where they are defined. Constraints (6.35) and (6.36) assign the initial bay layout to the variables. The Count constraints (6.37) ensure the number of containers with a specific retrieval priority and the number of empty slots remain constant. The connection between variables x and δ is established through (6.38) and (6.39). Constraints (6.38) indicate that variables x and δ are both equal to 0 (for empty slots) or both positive (for occupied slots). If the value of a variable x or δ remains unchanged between stages, (6.39) ensure it also remains unchanged for the corresponding variable in the other set. Constraints (6.40) impose that the retrievals from the bay are performed according to the priority schedule.

In CP-BRP, the definition of variables $b^k_{s,t}$ has been extended to every stage, compared to LCT-BRP, and the constraints have been modified accordingly. The condition in LCT-BRP, which states that the number of stages in the BRP phase must allow at least as many relocations as there are blocking containers in the layout after premarshalling, is extended in CP-BRP to every stage, as expressed in (6.42). These constraints indicate that the remaining stages must be enough to retrieve the containers still in the bay and relocate the blocking containers at least once. Constraints (6.43) ensure a variable $b^k_{s,t}$ takes the value 1 if there is a blocking container in slot (s,t) at stage k (i.e., a container with a later priority than a container below) and takes the value 0 otherwise.

The constraints describing the movements and establishing a connection between the variables referring to the bay layout and those identifying the movements are (6.44) to (6.52).

At most one movement is allowed per stage, as indicated by (6.44) and (6.45), and they must be performed consecutively in the first stages as imposed by (6.46). The destination slot of a relocation must remain occupied in the subsequent stage, as determined by (6.47), and the destination slot in stack 0 of a retrieval must remain occupied for the rest of the stages, as expressed by (6.48).

The Allowed Assignments constraints (6.49) ensure the bay layout changes according to the relocations performed, only placing and removing containers in allowed slots. The table constraints (6.52) impose these conditions for stack 0, together with (6.50) and (6.51) for the initial stage.

In addition to the constraints expressing the condition of the restricted version of the BRP, (6.53), an extra group of constraints is incorporated in the CP-

BRP, (6.54). These constraints indicate that only a blocking container can be relocated. This condition does not apply to the unrestricted version, where any container can be moved.

$$\sum_{t \in \mathcal{T}} z_{s,t}^{k} \;\leq\; \sum_{t \in \mathcal{T}} z_{s,t}^{k+1} + \sum_{t \in \mathcal{C}} y_{0,t}^{k} \qquad \forall s \in \mathcal{S},\, k \in \mathcal{K} \setminus \{\bar{k}\} \tag{6.53}$$

$$z_{s,t}^{k+1} \;\leq\; 1 - \left(b_{s,t}^{k} == \sum_{c \in \mathcal{C}} y_{0,c}^{k+1} \right) \qquad \forall (s,t) \in \mathcal{L} : t > 1,\, k \in \mathcal{K} \setminus \{\bar{k}\} \tag{6.54}$$

Solution method

We solve the BRP using CP-BRP along with an iterative algorithm, omitting the objective function (6.34) in the formulation. We use the number of stages $\bar{k}$ to guide the search for an optimal solution.

The number of blocking containers in the initial layout is a lower bound for the relocations needed to retrieve all the containers from the bay. We assign $\bar{k}$ the sum of this lower bound and the number of containers in the bay, so it is a lower bound for the total number of movements, counting both relocations and retrievals. A first iteration is performed using this value for $\bar{k}$. If a solution is found, it is optimal. Otherwise, if the problem is infeasible, the number of stages is incremented by one, and a new iteration is performed. Essentially, this is the same algorithm used for the constraint programming models for the CPMP described in Chapter 3, but defining the number of stages according to a lower bound for the BRP.

6.4 Computational experiments

In this section, we describe the computational study and analyze the results obtained. We compare the solution method presented for the original model for the CPMP-LCT, LCT1, in Section 5.3.2, with the solution method proposed in this chapter for minimizing the inaccessible containers, which uses the LCT1 model and a novel formulation LCT-I. Then, we evaluate the LCT-I approach together with the two other alternative approaches proposed using other objectives, LCT-IB and LCT-BRP, to delve into the differences in their solutions.

The experiments are conducted over 320 instances from the CV dataset, which is described in Section 3.3.1, where also technical details of the experiments are

provided. The crane times data is sourced from Parreño-Torres et al., 2020, as in the experiments of the previous chapter. Additionally, the algorithm described in Section 6.2.1 for solving the CPMP-LCT with the objective of minimizing the inaccessible containers incorporates a fail limit of 500 000 in the first phase.

6.4.1 Performance of the proposed models and solution methods for the CPMP-LCT

The premarshalling under limited crane time always has a trivial feasible solution since conducting no movement satisfies all the constraints. However, finding an optimal solution or a good feasible solution is not easy, and this difficulty strongly depends on the chosen objective. Table 6.4 shows the number of optimal solutions and the average running time for the different strategies presented in this chapter and the model LCT1 using the solution method described in Section 5.3.4.

The algorithm described in Section 6.2.1 of this chapter for solving the CPMP-LCT with the original objective of minimizing the number of inaccessible containers is denoted in Table 6.4 by LCT-I, the name of the model used in the second phase of this algorithm. The method employed to solve the same problem, but only using model LCT1, as described in the previous chapter, is denoted in the table by LCT1.

The algorithm approach involving the LCT-I model solves almost all the instances used in the computational study with optimality proof. This means a significant increase in optimal solutions compared to the method solely using LCT1 and the solver. In addition, we observe a marked decrease in running time when employing the iterative algorithm. Therefore, the solution method proposed in this chapter for minimizing inaccessible containers significantly improves the performance of the original method.

The complexity of the objective function considered for the CPMP-LCT clearly affects the number of optimal solutions and the running time, as reflected in Table 6.4. Minimizing the number of relocations in the retrieval phase after premarshalling is the most complex objective. It provides the lowest number of optimal solutions and the highest running times. Conversely, minimizing the number of inaccessible containers provides the shortest running times and the most significant number of optimal solutions. These figures indicate the intermediate complexity of the objective of minimizing the sum of inaccessible and blocking containers compared to the other two objectives.

Table 6.4: Performance of LCT1, LCT-I, LCT-IB and LCT-BRP for the CPMP-LCT.

$\bar{c}$	$\bar{s}$	$\bar{t}$	#Optimal				Average CPU (s)			
			LCT1	LCT-I	LCT-IB	LCT-BRP	LCT1	LCT-I	LCT-IB	LCT-BRP
9	3	5	**40**	**40**	**40**	**40**	7	**2**	3	8
12	4	5	**40**	**40**	**40**	**40**	116	**31**	48	306
15	5	5	27	**40**	**40**	22	2259	**70**	1285	2133
16	4	6	**40**	**40**	**40**	20	176	**36**	62	2232
18	6	5	2	**40**	4	3	3503	**190**	3336	3428
20	5	6	22	**40**	39	2	2761	**59**	1775	3379
21	7	5	0	**40**	0	0	3600	**564**	3600	3600
24	8	5	0	**34**	0	0	3600	**1251**	3600	3600
Total			171	**314**	203	127	2003	**275**	1713	2044

Results on 320 instances from the CV dataset, grouped into eight categories of 40 instances, ordered by the number of containers in the bay, $\bar{c}$. The bays in each category have $\bar{s}$ stacks and $\bar{t}$ tiers. The number of optimal solutions obtained is displayed in the #Optimal columns, and the average running time for all the instances is presented in the Average CPU (s) column. The best values are shown in bold. Source: Elaborated by the author (2024).

6.4.2 Differences in the solutions yielded by the three different objectives

Tables 6.5 and 6.6 show the average percentage reduction in inaccessible containers, blocking containers, and BRP relocations, for optimal and feasible solutions obtained through the three approaches discussed in this study. These reductions represent the difference between the final layout provided by the best solution found for each instance and its initial layout. The BRP relocations are given as part of the solution when using the objective of minimizing these relocations. For the other approaches and for the initial bay layouts, the number of relocations necessary for retrieving the containers from the bay has been calculated using the BRP model described in Section 6.3.

The data displayed in Table 6.5 for the solutions proven optimal by all the approaches offers valuable insights into how each objective influences the characteristics of the solutions. The LCT-I approach achieves the most significant reduction in the number of inaccessible containers, closely followed by the approach minimizing the sum of inaccessible and blocking containers. The objective of minimizing BRP relocations yields the largest reduction in blocking containers, which is very similar to the reduction achieved by minimizing the sum of inaccessible and blocking containers. This similarity suggests a cor-

relation between reducing the number of BRP relocations and the number of blocking containers, aligning with the underlying principle of the LCT-IB approach. As expected, the reduction in relocations during the retrieval process is most significant in the solutions provided by LCT-BRP. We also observe that this reduction is notably greater for LCT-IB than LCT-I, indicating the effectiveness of the former as an approximation for LCT-BRP. These observations demonstrate the capacity of the LCT-IB approach to deliver good-quality solutions for the other two objectives.

Table 6.5: Comparison of optimal solutions yielded by LCT-I, LCT-IB and LCT-BRP.

$\bar{c}$	$\bar{s}$	$\bar{t}$	#Inaccessible reduction (%)			#Blocking reduction (%)			#BRP relocations red. (%)		
			LCT-I	LCT-IB	LCT-BRP	LCT-I	LCT-IB	LCT-BRP	LCT-I	LCT-IB	LCT-BRP
9	3	5	**37**	36	27	32	**51**	49	41	53	**59**
12	4	5	**40**	39	26	30	47	**50**	42	53	**60**
15	5	5	**47**	46	32	42	52	**60**	49	59	**65**
16	4	6	**20**	**20**	12	14	25	**27**	26	33	**41**
18	6	5	**67**	**67**	46	**79**	**79**	**79**	**79**	**79**	**79**
20	5	6	**18**	**18**	**18**	11	**47**	**47**	11	**50**	**50**
21	7	5	0	0	0	0	0	0	0	0	0
24	8	5	0	0	0	0	0	0	0	0	0
Total			**37**	36	26	31	46	**48**	41	51	**58**

Results on 320 instances from the CV dataset, grouped into eight categories of 40 instances, ordered by the number of containers in the bay, $\bar{c}$. The bays in each category have $\bar{s}$ stacks and $\bar{t}$ tiers. The table shows the average percentage reduction in inaccessible containers, blocking containers, and BRP relocations between the initial and final bay layouts of the instances optimally solved by all three approaches, LCT-I, LCT-IB and, LCT-BRP. The best values are shown in bold. Source: Elaborated by the author (2024).

Since an optimal solution is not always found, it is interesting to evaluate the feasible solutions. Table 6.6 is built analogously to Table 6.5, but with the results for all feasible solutions. Minimizing the sum of inaccessible and blocking containers is the most attractive objective, providing more significant reductions in the number of BRP relocations and blocking containers than the other two approaches. Moreover, the reduction in inaccessible containers is extremely close to that achieved by minimizing the number of inaccessible containers. Consequently, the LCT-IB approach provides good solutions in terms of the original CPMP-LCT objective and, at the same time, greatly enhances the solutions concerning the reduction in blocking containers and relocations in the retrieval phase.

Table 6.6: Comparison of feasible solutions yielded by LCT-I, LCT-IB and LCT-BRP.

$\bar{c}$	$\bar{s}$	$\bar{t}$	#Inaccessible reduction (%)			#Blocking reduction (%)			#BRP relocations red. (%)		
			LCT-I	LCT-IB	LCT-BRP	LCT-I	LCT-IB	LCT-BRP	LCT-I	LCT-IB	LCT-BRP
9	3	5	**37**	36	27	32	**51**	49	41	53	**59**
12	4	5	**40**	39	26	30	47	**50**	42	53	**60**
15	5	5	**39**	38	22	30	**44**	40	41	**51**	49
16	4	6	**20**	**20**	10	12	**21**	18	24	28	**30**
18	6	5	**30**	29	13	24	**39**	25	34	**44**	29
20	5	6	**18**	**18**	6	11	**21**	9	20	**28**	14
21	7	5	**30**	29	7	23	**35**	16	32	**42**	16
24	8	5	**26**	25	2	15	**28**	5	25	**35**	3
Total			**30**	29	14	22	**36**	26	32	**42**	33

Results on 320 instances from the CV dataset, grouped into eight categories of 40 instances, ordered by the number of containers in the bay, $\bar{c}$. The bays in each category have $\bar{s}$ stacks and $\bar{t}$ tiers. The table shows the average percentage reduction in inaccessible containers, blocking containers, and BRP relocations between the initial and final bay layouts of the instances solved by all three approaches, LCT-I, LCT-IB, and LCT-BRP. The best values are shown in bold. Source: Elaborated by the author (2024).

In conclusion, if the conditions in practice define a specific interest in minimizing the number of inaccessible containers over other objectives, as indicated in Section 5.2, the LCT-I approach is recommended due to its high rate of optimal solutions and shorter running times, compared to the other alternatives. However, in scenarios where such conditions are not prevalent, minimizing the sum of inaccessible and blocking containers is the most suitable objective. This approach offers versatile solutions, effectively reducing both inaccessible and blocking containers, as well as BRP relocations, making it well-suited for a broad range of cases.

6.5 Concluding remarks

The Premarshalling Problem under Limited Crane Time (CPMP-LCT) is an extension of the classical Premarshalling Problem (CPMP). It aims to provide a complete premarshalling solution when the availability permits while also defining a partial premarshalling solution within the given crane time limit if a complete arrangement is not possible. Therefore, the CPMP-LCT is more versatile and realistic than the CPMP. However, it also necessitates more com-

plex formulations and involves the challenging task of evaluating the efficiency of partial premarshalling solutions.

The ultimate goal of the CPMP-LCT is the same as that of the CPMP: to best prepare the bay for container retrieval after premarshalling. A layout without blockages between containers is the ideal configuration for the retrieval phase since it requires no additional relocations. While such configuration is ensured by complete premarshalling, it is not achievable with partial premarshalling, so criteria for defining efficient partial premarshalling solutions must be determined.

The original version of the CPMP-LCT presented in Chapter 5 aims to minimize the number of inaccessible containers in the bay. This objective focuses on unblocking the maximum number of containers in retrieval order, i.e., prioritizing the containers that must be retrieved first. Consequently, this approach is advantageous when the information about the last retrievals is missing or likely to change. It is also beneficial if the retrieval of containers from the bay may be interrupted, allowing for additional premarshalling of containers with the latest priorities. In this chapter, we propose an enhanced solution method for this approach, using an additional formulation and an iterative algorithm, which has demonstrated a significant improvement over the original method.

While the original objective is advantageous for specific scenarios, an alternative objective may be more appropriate in a general case. A beneficial approach for preparing the bay for the retrieval phase is minimizing the relocations needed to retrieve all the containers from the bay. This means finding partial premarshalling solutions that minimize the objective function of the Block Relocation Problem (BRP). We propose a constraint programming model and a solution method for solving the CPMP-LCT with this objective. Additionally, we provide a constraint programming model for the BRP to assess the relation with this objective of the solutions provided by other approaches.

Unfortunately, solving the CPMP-LCT with the objective of minimizing the relocations in the retrieval phase is highly complex, as it requires solving the CPMP-LCT and the BRP simultaneously. Therefore, we explore another objective that combines the advantages of the two other objectives studied. This intermediate objective is to minimize the sum of inaccessible and blocking containers. We have developed a constraint programming model and a solution method for solving the CPMP-LCT with this objective. The computational experiments reveal that this approach offers high-quality partial premarshalling solutions in terms of the three objectives studied, demonstrating its adequacy for solving the CPMP-LCT in a general case.

Chapter 7

Conclusions and future work

The purpose of this thesis is to provide more realistic formulations of the container premarshalling problem and develop an effective, flexible solution method that can address the original problem and these alternative variants. We have examined the assumptions of the original formulation and defined novel versions of the problem by modifying several assumptions to align with the resource availability and the operational requirements of port terminals. Subsequently, we have developed constraint programming models and tailored algorithms specific to each formulation. The constraint programming approaches presented in this thesis improve the performance of state-of-the-art mathematical programming models for premarshalling variants already studied in the literature and enable us to validate the novel problems introduced.

This chapter summarizes the main conclusions of this thesis. We review the progress made toward developing a more realistic formulation of the premarshalling problem. Additionally, we discuss the significance of the proposed solution methods and how constraint programming enabled us to accomplish the objective of developing effective solution methods for various formulations and alternative approaches. Finally, we suggest directions for future research that build on the work presented in this thesis.

7.1 Reformulating unrealistic assumptions for the premarshalling problem

While numerous optimization methods have been proposed for premarshalling, the classical formulation of the problem is not realistic enough for successful application at port terminals. This thesis addresses this issue by introducing novel formulations that contribute to bridging the gap between theoretical methods and practical applications.

First, we examine the classical Container Premarshalling Problem (CPMP). Chapter 3 provides a detailed description of the original formulation and its assumptions. The CPMP seeks to find a sequence of relocations within a bay of containers that results in an arrangement where no container blocks the retrieval of another, with this sequence comprising the minimum number of relocations possible.

Two key assumptions in the CPMP are that relocations can only occur within the bay being arranged, and that the crane is fully available for premarshalling. These assumptions are unrealistic in practice and are reformulated in this thesis, defining two novel problem versions: the Container Premarshalling Problem with an Auxiliary Bay (CPMP-AB) and the Container Premarshalling Problem under Limited Crane Time (CPMP-LCT). The CPMP-AB is presented in Chapter 4, and the CPMP-LCT is studied in Chapters 5 and 6.

Another assumption of the CPMP is that all relocations incur the same cost. An alternative approach, known as the Container Premarshalling Problem with Crane Time Minimization Objective (CPMPCT), has been studied in the literature. This version allows for more efficient solutions by considering relocation costs based on crane movement duration. This thesis addresses the CPMPCT as a preliminary step for defining the CPMP-LCT in Chapter 5.

7.1.1 Using an auxiliary bay for premarshalling

The Container Premarshalling Problem with an Auxiliary Bay (CPMP-AB) introduces the use of an auxiliary bay as additional space for relocating containers during the arrangement process. Although this is a common practice at port terminals, the classical formulation overlooks this option.

The objective of the CPMP-AB is to arrange a bay with the minimum number of relocations, as in the original CPMP. However, this version permits relocations between the bay being arranged (main bay) and another bay (auxiliary bay). We propose two approaches for defining the cost of these inter-bay reloca-

tions. One approach minimizes two objectives in lexicographic order: initially minimizing the total number of relocations to arrange the main bay and then, with that total fixed, minimizing the number of relocations between bays. The other approach minimizes the total cost of relocations, assigning double the cost to those between bays compared to relocations within the main bay.

The solutions obtained through the computational study show that using an auxiliary bay can significantly reduce the relocations required for premarshalling, particularly in bays with high occupancy rates and limited space for relocation. Furthermore, the auxiliary bay enables solving instances that would be infeasible if only relocations within the bay to be arranged were allowed.

The two proposed allowance levels for inter-bay relocations extend the applicability of the solutions to port terminals with varying operational needs. Allowing inter-bay relocation without a penalty leads to more instances obtaining a reduction in the total number of relocations and a more substantial reduction than when this kind of relocation incurs a higher cost. Port terminals permitting around 20% of relocations in a premarshalling solution to occur between bays could implement the first approach, which fully exploits the potential of using an auxiliary bay. Alternatively, the approach penalizing inter-bay relocations is more appropriate when only about 10% of relocations are allowed between bays. Both approaches improve on the solutions provided by the original CPMP by enhancing the utilization of port resources.

7.1.2 Considering crane times and limited availability of the crane

The study of the Container Premarshalling Problem with Crane Time Minimization Objective (CPMPCT) and the introduction of the Container Premarshalling Problem under Limited Crane Time (CPMP-LCT) in this thesis contribute to obtaining more realistic premarshalling solutions. These problems arise from modifying two classical assumptions: uniform relocation cost and unlimited crane availability. On the one hand, considering varying relocation costs based on different durations leads to more efficient solutions, as previously demonstrated in the literature. On the other hand, recognizing that premarshalling is only performed during idle periods of the crane conflicts with the classical assumption of unlimited crane availability. To address this, we propose the novel CPMP-LCT.

The original CPMP formulation assumes a uniform cost for all relocations, aiming to minimize their total number. However, the time required for reloca-

tions can vary significantly. As a result, solutions that minimize the total crane time are more efficient than those that focus on minimizing the number of relocations. This is the rationale behind the CPMPCT, a version of the premarshalling problem already defined in the literature. We address the CPMPCT as a preliminary step toward a more realistic formulation, the CPMP-LCT, that introduces the constraint of limited crane availability in addition to considering the time taken by relocations. The CPMP-LCT defines a complete arrangement of the bay when it can be performed within the available time and a partial arrangement otherwise, while the original formulation does not cover the latter case.

A partial arrangement may be advantageous over not conducting premarshalling due to limited time. However, defining good-quality partial premarshalling solutions is a challenging task. A simple procedure might be to partially apply a sequence of relocations intended to fully arrange a bay, stopping at an intermediate step before the time limit is exceeded. Nevertheless, this procedure typically yields poor-quality partial premarshalling solutions. In contrast, the CPMP-LCT significantly enhances the quality of partial solutions, as demonstrated through the computational study in this thesis.

The quality of a partial premarshalling solution is determined by its potential to increase the efficiency of the subsequent retrieval process, which is the same goal as in designing complete premarshalling solutions. Maximum efficiency during the retrieval phase is achieved when no container blocks the retrieval of another. However, after partial premarshalling, there are still blockages between containers, necessitating additional relocations during retrieval. Therefore, defining efficient partial premarshalling solutions is notably more challenging than defining complete ones, which simply require a layout without blocking containers. For this reason, we have analyzed three alternative objectives for the CPMP-LCT.

The initial objective we studied for the CPMP-LCT prioritizes the accessibility to containers that must be retrieved first according to the given retrieval sequence. Specifically, it minimizes the number of inaccessible containers, with a container considered inaccessible if this or any container with an earlier retrieval time is blocked by another container. This objective is particularly advantageous when there is inaccurate information about later retrievals or when additional premarshalling is possible after some containers have been retrieved. However, there may be more suitable approaches in general situations, as this overlooks containers that remain blocked after partial premarshalling, potentially leading to additional relocations during retrieval.

We have studied an alternative objective that addresses the containers involved in blockages after partial premarshalling: minimizing the total number of relocations required during the retrieval phase. It corresponds to the objective of the Block Relocation Problem (BRP), which aims to retrieve all containers from the bay according to the retrieval schedule with the minimum number of relocations. This objective is naturally derived from the original definition of premarshalling: a complete premarshalling solution ensures no relocations are needed during retrieval, and this approach for the CPMP-LCT seeks to minimize relocations when achieving zero relocations is not feasible. However, employing this objective entails solving the CPMP-LCT and the BRP simultaneously, with a strong interdependence between them, which is notably challenging.

Therefore, we have additionally explored a third objective that balances the complexity of the other two and yields good values for both criteria: minimizing the sum of inaccessible and blocking containers. The computational analysis demonstrates that this intermediate objective effectively yields efficient partial premarshalling solutions in general cases. It prioritizes the containers that must be retrieved first, similar to the initial objective studied, while also increasing the reduction in relocations needed during the retrieval phase, approaching the BRP objective. Moreover, this approach significantly reduces the complexity of the problem compared to minimizing relocations in the retrieval phase.

7.1.3 Future research lines toward a more realistic premarshalling formulation

With the reformulation of the assumptions indicated in the previous sections, this thesis represents a significant advancement toward developing a realistic formulation of the premarshalling problem, facilitating its implementation at port terminals. Nevertheless, there is still potential for further improvement in this direction.

The CPMP assumes complete knowledge of the retrieval order for containers after premarshalling. However, this is not the case in practice. In this thesis, the formulation of the CPMP-LCT acknowledges possible changes in the expected retrieval times of containers. Nevertheless, a more specific consideration of this uncertainty could be integrated into the formulation by employing strategies similar to those described for the robust CPMP in the literature.

Furthermore, additional conditions regarding the final layout should be incorporated. The current formulations in the literature only consider retrieval priorities to differentiate between containers. However, containers have other relevant distinguishing features, such as their weight, which may require specific bay configurations to ensure the stability of the stacks. Moreover, for safety reasons, marked differences in stack height and empty stacks next to much higher stacks must be avoided. These requirements become even more critical during adverse weather conditions, such as strong winds.

Additionally, to achieve a premarshalling formulation suitable for practical implementation, it would be necessary to integrate all the enhanced assumptions. This formulation would consider an auxiliary bay together with limited crane time, incorporate a strategy for dealing with the uncertainty in retrieval times, and ensure compliance with safety regulations.

7.2 Constraint Programming: An effective and versatile solution method for premarshalling

Constraint programming has demonstrated its effectiveness in solving combinatorial problems across various fields. Despite this, it has received very little attention in the literature on premarshalling. This thesis explores its application, offering constraint programming solution methods for several versions of the premarshalling problem. Those addressing premarshalling approaches already studied in the literature allow us to assess the efficiency of constraint programming for the premarshalling problem and are a base for developing solution methods for the novel formulations proposed in this thesis.

The extensive literature on premarshalling and the flexibility that constraint programming offers in defining constraints facilitate modeling the problem using this technique. Nevertheless, finding a truly effective formulation is a challenging task. This difficulty is addressed throughout this thesis, particularly in Chapter 3, where the original version of the problem is studied.

7.2.1 Designing constraint programming models

Computational experiments in this thesis reveal that the mathematical programming models for the CPMP from the literature do not perform well when applied within the constraint programming framework. Indeed, constraint programming requires modeling strategies different from classical operations research techniques, such as integer programming. In Chapter 3, we describe

the modeling process followed to build a constraint programming model for the CPMP and present three intermediate formulations that lead to the final formulation we propose.

Essentially, the modeling strategy employed begins with a basic formulation containing the minimum decision variables necessary to describe the problem. It then systematically incorporates more variables and constraints. Additional variables are valuable only if they identify specific information that aids in the search process. Simply adding more variables does not necessarily improve performance. In fact, it can reduce the model's efficiency if not carefully selected.

The effectiveness of the additional variables in the proposed models is explained by the table constraints included in the formulations presented in Chapters 4 to 6, which address the alternative versions of the CPMP explored in this thesis. These formulations are derived from the model designed for the CPMP, with table constraints replacing a large set of constraints from this initial model. These global constraints are a simplified way of imposing the same conditions as the original set of constraints and capture the core structure of the problem. They involve all the additional variables incorporated during the construction of the model for the CPMP and ensure a strong connection between the bay layout and container relocations.

The last group of decision variables introduced during the CPMP modeling process deserves particular attention. These binary variables indicate whether or not there is a badly placed container in a slot. During the search process, constraints involving these variables help detect partial solutions that will not lead to feasible solutions, allowing for the earlier elimination of the corresponding branches of the search tree.

7.2.2 A versatile and effective solution method

The constraint programming approaches proposed in this thesis are designed to be easy to implement, reproduce, and adapt to various problem variants. These features were essential for this thesis since the goal of defining more realistic formulations for the premarshalling problem involves developing and testing a significant number of formulations. Additionally, validating the novel problems proposed required analyzing optimal solutions. Hence, developing an exact solution method was a requisite for this study.

The need for a highly flexible and exact solution method led us to use a solver. It allows for developing models that can be easily adapted to varying require-

ments, unlike other exact methods such as complex ad-hoc algorithms. While the literature on premarshalling includes significant mathematical programming contributions that benefit from the advantages of using a solver, constraint programming was almost unexplored for this problem.

The development of constraint programming models for the original CPMP and its variant with crane time minimization objective, CPMPCT, revealed the potential of this technique for addressing the premarshalling problem. In both cases, the constraint programming models proposed in this thesis outperform the state-of-the-art mathematical programming models. This achievement established a solid base for developing constraint programming models for the novel premarshalling variants defined in this study, namely the CPMP-AB and the CPMP-LCT, yielding satisfactory results that validate the new formulations.

In the proposed methods, we mainly employ the solver to tackle satisfaction problems, guiding the search for optimal solutions through iterative algorithms. Constraint programming typically excels in satisfaction problems rather than optimization problems. Accordingly, the algorithms devised for the proposed models significantly enhance the solution methods. These methods generally provide a significant number of optimal solutions and can also generate good-quality feasible solutions in the absence of an optimality proof. Particularly, results presented in this thesis for the CPMP-LCT show that the models can deliver high-quality feasible solutions within short running times, suggesting the method's suitability as a heuristic approach.

7.2.3 Future research lines from a solution method perspective

As detailed in the previous sections, the constraint programming approaches developed in this thesis offer significant advantages. They are versatile, easy to implement, and capable of providing high-quality solutions. These features have been crucial for the development of this study and are also beneficial for users seeking a simple solution approach that is easy to use and adapt to changing requirements.

Even then, the methods developed in this thesis could be improved, or new techniques could be explored to address the newly proposed problems. For instance, the constraint programming models could be tested with alternative solvers to investigate the possibility of better performance. Additionally, further research could focus on developing more complex ad-hoc algorithms to ensure optimality over a broader range of solutions and reduce running times.

Bibliography

Araya, I., & Toledo, M. (2023). A fill-and-reduce greedy algorithm for the container pre-marshalling problem. *Operational Research*, *23*(51). https://doi.org/10.1016/j.cor.2019.104781 (cit. on p. 21).

Bacci, T., Mattia, S., & Ventura, P. (2019). The bounded beam search algorithm for the block relocation problem. *Computers & Operations Research*, *103*, 252–264. https://doi.org/10.1016/j.cor.2018.11.008 (cit. on p. 23).

Bacci, T., Mattia, S., & Ventura, P. (2020). A branch-and-cut algorithm for the restricted block relocation problem. *European Journal of Operational Research*, *287*, 452–459. https://doi.org/10.1016/j.ejor.2020.05.029 (cit. on p. 23).

Barták, R. (2011). History of constraint programming. In *Wiley encyclopedia of operations research and management science*. John Wiley & Sons. https://doi.org/10.1002/9780470400531.eorms0382 (cit. on p. 24).

Boge, S., Goerigk, M., & Knust, S. (2020). Robust optimization for premarshalling with uncertain priority classes. *European Journal of Operational Research*, *287*, 191–210. https://doi.org/10.1016/j.ejor.2020.04.049 (cit. on p. 22).

Boge, S., & Knust, S. (2023). The blocks relocation problem with item families minimizing the number of reshuffles. *OR Spectrum*, *45*, 395–435. https://doi.org/10.1007/s00291-022-00703-x (cit. on p. 23).

Bortfeldt, A., & Forster, F. (2012). A tree search procedure for the container pre-marshalling problem. *European Journal of Operational Research*, *217*, 531–540. https://doi.org/10.1016/j.ejor.2011.10.005 (cit. on pp. 19–21, 47).

Caserta, M., & Voß, S. (2009). A corridor method-based algorithm for the pre-marshalling problem. In M. Giacobini, A. Brabazon, S. Cagnoni, G. A. D. Caro, A. Ekárt, A. I. Esparcia-Alcázar, M. Farooq, A. Fink, & P. Machado (Eds.), *Applications of evolutionary computing* (pp. 788–797). Springer, Berlin, Heidelberg. https://doi.org/https://doi.org/10.1007/978-3-642-01129-0_89 (cit. on pp. 21, 50).

Covic, F. (2018). A literature review on container handling in yard blocks. In R. Cerulli, A. Raiconi, & S. Voß (Eds.), *Computational logistics* (pp. 139–167). Springer International Publishing. https://doi.org/10.1007/978-3-030-00898-7_9 (cit. on p. 19).

da Silva Firmino, A., de Abreu Silva, R. M., & Times, V. C. (2019). A reactive GRASP metaheuristic for the container retrieval problem to reduce crane's working time. *Journal of Heuristics*, *25*, 141–173. https://doi.org/10.1007/s10732-018-9390-0 (cit. on p. 23).

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*, *271*, 40–56. https://doi.org/10.1016/j.ejor.2018.05.004 (cit. on pp. 20, 38, 41, 43, 47).

Esteban-Chapapría, V., & Serra-Peris, J. (2021). Vulnerability of coastal areas due to infrastructure: The case of Valencia port (Spain). *Land*, *10*(12), 1344. https://doi.org/10.3390/land10121344 (cit. on p. 4).

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. https://doi.org/10.1016/j.eswa.2012.01.187 (cit. on pp. 20, 21, 50).

Farrelly, J., & Grimes, D. (2022). An iterated local search approach to the container pre-marshalling problem. *33rd Irish Signals and Systems Conference*, 1–6. https://doi.org/10.1109/ISSC55427.2022.9826140 (cit. on p. 21).

Gheith, M., Eltawil, A. B., & Harraz, N. A. (2016). Solving the container premarshalling problem using variable length genetic algorithms. *Engineering Optimization*, *48*(4), 687–705. https://doi.org/10.1080/0305215X.2015.1031661 (cit. on p. 21).

Hottung, A., Tanaka, S., & Tierney, K. (2020). Deep learning assisted heuristic tree search for the container pre-marshalling problem. *Computers & Operations Research*, *113*, 104781. https://doi.org/10.1016/j.cor.2019.104781 (cit. on p. 21).

Hottung, A., & Tierney, K. (2016). A biased random-key genetic algorithm for the container pre-marshalling problem. *Computers & Operations Research*, *75*, 83–102. https://doi.org/10.1016/j.cor.2016.05.011 (cit. on p. 21).

Huang, S.-H., & Lin, T.-H. (2012). Heuristic algorithms for container premarshalling problems. *Computers & Industrial Engineering*, *62*, 13–20. https://doi.org/10.1016/j.cie.2011.08.010 (cit. on p. 21).

Jiménez-Piqueras, C., Parreño-Torres, C., Alvarez-Valdes, R., & Ruiz, R. (2024). The container premarshalling problem under limited crane time: A constraint programming approach. *Computers & Operations Research*, *166*. https://doi.org/https://doi.org/10.1016/j.cor.2024.106635 (cit. on pp. 10, 73, 91–93, 104–109).

Jiménez-Piqueras, C., Ruiz, R., Parreño-Torres, C., & Alvarez-Valdes, R. (2023). A constraint programming approach for the premarshalling problem. *European Journal of Operational Research*. https://doi.org/https://doi.org/10.1016/j.ejor.2022.07.042 (cit. on pp. 9, 33, 41, 43, 51–55, 57–59).

Jin, B. (2020). A note on "An exact algorithm for the blocks relocation problem with new lower bounds". *Computers & Operations Research*, *124*, 105082. https://doi.org/10.1016/j.cor.2020.105082 (cit. on p. 23).

Jin, B., & Tanaka, S. (2023). An exact algorithm for the unrestricted container relocation problem with new lower bounds and dominance rules. *European*

Journal of Operational Research, *304*(2), 494–514. https://doi.org/10.1016/j.ejor.2022.04.006 (cit. on p. 23).

Jin, B., & Yu, M. (2021). Note on the dominance rules in the exact algorithm for the container pre-marshalling problem by Tanaka & Tierney (2018). *European Journal of Operational Research*, *293*(2), 802–807. https://doi.org/10.1016/j.ejor.2020.12.041 (cit. on p. 20).

Jovanovic, R., Tanaka, S., Nishi, T., & Voß, S. (2019). A grasp approach for solving the blocks relocation problem with stowage plan. *Flexible Services and Manufacturing Journal*, *31*, 702–729. https://doi.org/10.1007/s10696-018-9320-3 (cit. on p. 23).

Jovanovic, R., Tuba, M., & Voß, S. (2017). A multi-heuristic approach for solving the pre-marshalling problem. *Central European Journal of Operations Research*, *25*, 1–28. https://doi.org/10.1007/s10100-015-0410-y (cit. on p. 21).

Jovanovic, R., Tuba, M., & Voß, S. (2019). An efficient ant colony optimization algorithm for the blocks relocation problem. *European Journal of Operational Research*, *274*, 78–90. https://doi.org/10.1016/j.ejor.2018.09.038 (cit. on p. 23).

Kimns, A., & Wilschewski, F. (2023). A new modeling approach for the unrestricted block relocation problem. *OR Spectrum*, *45*, 1071–1111. https://doi.org/10.1007/s00291-023-00728-w (cit. on p. 23).

Kizilay, D., & Eliiyi, D. T. (2021). A comprehensive review of quay crane scheduling, yard operations and integrations thereof in container terminals. *Flexible Services and Manufacturing Journal*, *33*, 1–42. https://doi.org/10.1007/s10696-020-09385-5 (cit. on p. 19).

Lee, Y., & Chao, S.-L. (2009). A neighborhood search heuristic for pre-marshalling export containers. *European Journal of Operational Research*, *196*, 468–475. https://doi.org/10.1016/j.ejor.2008.03.011 (cit. on p. 21).

Lee, Y., & Hsu, N.-Y. (2007). An optimization model for the container pre-marshalling problem. *Computers & Operations Research*, *34*, 3295–3313. https://doi.org/10.1016/j.cor.2005.12.006 (cit. on p. 19).

Lee, Y., & Lee, Y.-J. (2010). A heuristic for retrieving containers from a yard. *Computers & Operations Research*, *37*(6), 1139–1147. https://doi.org/10.1016/j.cor.2009.10.005 (cit. on p. 23).

Lersteau, C., & Shen, W. (2022). A survey of optimization methods for block relocation and premarshalling problems. *Computers & Industrial Engineering*, *172*, 108529. https://doi.org/10.1016/j.cie.2022.108529 (cit. on p. 19).

Lin, D.-Y., Lee, Y.-J., & Lee, Y. (2015). The container retrieval problem with respect to relocation. *Transportation Research Part C: Emerging Technologies*, *52*, 132–143. https://doi.org/10.1016/j.trc.2015.01.024 (cit. on p. 23).

Liu, S., Liu, S., Lu, C., Zhou, M., & Abusorrah, A. (2022). Valid inequality and variable fixation for unrestricted block relocation problems. *IEEE Transactions on Intelligent Transportation Systems*, *23*(10), 18822–18834. https://doi.org/10.1109/TITS.2022.3151069 (cit. on p. 23).

Lu, C., Zeng, B., & Liu, S. (2020). A study on the block relocation problem: Lower bound derivations and strong formulations. *IEEE Transactions on Automation Science and Engineering*, *17*, 1829–1853. https://doi.org/10.1109/TASE.2020.2979868 (cit. on p. 23).

Parreño-Torres, C., Alvarez-Valdes, R., & Parreño, F. (2022). A beam search algorithm for minimizing crane times in premarshalling problems. *European Journal of Operational Research*, *302*(3), 1063–1078. https://doi.org/10.1016/j.ejor.2022.01.038 (cit. on p. 22).

Parreño-Torres, C., Alvarez-Valdes, R., & Ruiz, R. (2019). Integer programming models for the pre-marshalling problem. *European Journal of Operational Research*, *274*, 142–154. https://doi.org/10.1016/j.ejor.2018.09.048 (cit. on pp. 20, 38, 41, 43, 47, 48, 56, 60).

Parreño-Torres, C., Alvarez-Valdes, R., Ruiz, R., & Tierney, K. (2020). Minimizing crane times in pre-marshalling problems. *Transportation Research Part E: Logistics and Transportation Review*, *137*, 101917. https://doi.org/10.1016/j.tre.2020.101917 (cit. on pp. 22, 23, 84, 85, 102, 103, 135).

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. https://doi.org/10.1016/j.cor.2018.06.021 (cit. on p. 23).

Rendl, A., & Prandtstetter, M. (2013). Constraint models for the container pre-marshaling problem. In G. Katsirelos & C.-G. Quimper (Eds.), *The twelfth international workshop on constraint modelling and reformulation (modref 2013)* (pp. 44–56). (Cit. on pp. 19, 22, 38, 47).

Rossi, F., van Beek, P., & Walsh, T. (Eds.). (2006). *Handbook of constraint programming*. Elsevier. (Cit. on p. 24).

Sutherland, I. (1963). *Sketchpad: A man-machine graphical communication system* [Doctoral dissertation, Massachusetts Institute of Technology]. (Cit. on p. 24).

Tanaka, S., & Mizuno, F. (2018). An exact algorithm for the unrestricted block relocation problem. *Computers & Operations Research*, *95*, 12–31. https://doi.org/10.1016/j.cor.2018.02.019 (cit. on p. 23).

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. https://doi.org/10.1016/j.ejor.2017.05.046 (cit. on pp. 20, 47).

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. https://doi.org/10.1016/j.ejor.2019.04.005 (cit. on pp. 20, 47, 77).

Tanaka, S., & Voß, S. (2022). An exact approach to the restricted block relocation problem based on a new integer programming formulation. *European Journal of Operational Research*, *296*(2), 485–503. https://doi.org/10.1016/j.ejor.2021.03.062 (cit. on p. 23).

Tierney, K., Pacino, D., & Voß, S. (2016). Solving the pre-marshalling problem to optimality with A* and IDA*. *Flexible Services and Manufacturing Journal*, *29*, 223–259. https://doi.org/10.1007/s10696-016-9246-6 (cit. on pp. 20, 40, 50).

Tierney, K., & Voß, S. (2016). Solving the robust container pre-marshalling problem. In A. Paias, M. Ruthmair, & S. Voß (Eds.), *Lecture notes in computer science* (pp. 131–145, Vol. 9855). Springer, Cham. (Cit. on p. 22).

Ting, C.-J., & Wu, K.-C. (2017). Optimizing container relocation operations at container yards with beam search. *Transportation Research Part E*, *103*, 17–31. https://doi.org/10.1016/j.tre.2017.04.010 (cit. on p. 23).

UNCTAD. (1992). *Review of maritime transport 1992* (tech. rep.). United Nations Conference on Trade and Development. (Cit. on p. 3).

UNCTAD. (2018). *50 years of review of maritime transport, 1968–2018: Reflecting on the past, exploring the future* (tech. rep.). Transport, Trade Facilitation Newsletter, Series No. 10, United Nations Conference on Trade, and Development. (Cit. on p. 3).

UNCTAD. (2021). *Review of maritime transport 2021* (tech. rep.). United Nations Conference on Trade and Development. (Cit. on p. 3).

UNCTAD. (2022). *Review of maritime transport 2022* (tech. rep.). United Nations Conference on Trade and Development. (Cit. on p. 3).

UNCTAD. (2023). *Review of maritime transport 2023* (tech. rep.). United Nations Conference on Trade and Development. (Cit. on p. 3).

van Brink, M., & van der Zwaan, R. (2014). A branch and price procedure for the container premarshalling problem. *arXiv*, *1406.7107v1*. https://doi.org/10.1007/978-3-662-44777-2_66 (cit. on pp. 20, 49).

Wang, N., Jin, B., & Lim, A. (2015). Target-guided algorithms for the container pre-marshalling problem. *Omega*, *53*, 67–77. https://doi.org/10.1016/j.omega.2014.12.002 (cit. on pp. 21, 22, 63).

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. https://doi.org/10.1016/j.ejor.2016.05.061 (cit. on p. 21).

Weerasinghe, B. A., Perera, H. N., & Bai, X. (2024). Optimizing container terminal operations: A systematic review of operations research applications. *Maritime Economics & Logistics*, *26*, 307–341. https://doi.org/10.1057/s41278-023-00254-0 (cit. on p. 19).

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 : Theory Applications and Practice*, *22*, 509–523. https://doi.org/10.23055/ijietap.2015.22.5.1514 (cit. on pp. 20, 49).

Zweers, B. G., Bhulai, S., & van der Mei, R. D. (2020a). Optimizing pre-processing and relocation moves in the stochastic container relocation problem. *European Journal of Operational Research*, *283*(3), 954–971. https://doi.org/10.1016/j.ejor.2019.11.067 (cit. on p. 23).

Zweers, B. G., Bhulai, S., & van der Mei, R. D. (2020b). Pre-processing a container yard under limited available time. *Computers & Operations Research*, *123*, 105045. https://doi.org/10.1016/j.cor.2020.105045 (cit. on p. 23).

This thesis aims to bridge the gap between the mathematical formulation of the container premarshalling problem and its practical application. It addresses both theoretical and technical aspects of this optimization problem. On the theoretical side, this thesis contributes novel problem variants to the scientific literature. It studies and reformulates unrealistic assumptions made in the original formulation. On the technical side, it presents constraint programming approaches for both the newly proposed variants and previous formulations. These methods demonstrate the efficiency of constraint programming in solving the container premarshalling problem.