Optimization and Robustness in Planning and Scheduling Problems. Application to Container Terminals

Mario Rodríguez Molins

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy in the Departament de Sistemes Informàtics i Computació at Universitat Politècnica de València

Supervisors: Dr. Miguel A. Salido Gregorio
Prof. Federico Barber Sanchís

February 2015

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

A mi familia.

KARL POPPER

All life is problem solving.

EARL NIGHTINGALE

Never give up on a dream just because of the time it will take to accomplish it. The time will pass anyway.

Acknowledgements

First of all, I would like to thank my advisors: Miguel Ángel Salido and Federico Barber. I would like to thank Miguel for his constant dedication and his assistance all these years. I would also like to thank Federico for introducing me to his research group as well as for his guidance in my work. I would like to thank you both for encouraging my research and for allowing me to grow as a research scientist.

Other special words of acknowledgements are directed to my colleagues at *Departamento de Sistemas Informáticos y Computación* (DSIC). I have plenty of wonderful memories with them. Thank you all for these moments and for creating such a nice working environment. Moreover, your advice and help has been extremely valuable to me. I would like to thank in particular my fellows who once worked with me: Laura, Marlene, Valentina, Joan E., Flabio, César, Ximo, Joan P., Marcos, Enrique and Miquel.

Last but not least, a special thanks to my family and friends, from whom I have received endless love and support. The completion of this work has been possible thanks to them.

Abstract

Despite the continuous evolution in computers and information technology, real-world combinatorial optimization problems are NP-problems, in particular in the domain of planning and scheduling. Thus, although exact techniques from the Operations Research (OR) field, such as Linear Programming, could be applied to solve optimization problems, they are difficult to apply in real-world scenarios since they usually require too much computational time, i.e: an optimized solution is required at an affordable computational time. Furthermore, decision makers often face different and typically opposing goals, then resulting multi-objective optimization problems. Therefore, approximate techniques from the Artificial Intelligence (AI) field are commonly used to solve the real world problems. The AI techniques provide richer and more flexible representations of real-world (Gomes 2000), and they are widely used to solve these type of problems. AI heuristic techniques do not guarantee the optimal solution, but they provide near-optimal solutions in a reasonable time. These techniques are divided into two broad classes of algorithms: constructive and local search methods (Aarts and Lenstra 2003). They can guide their search processes by means of heuristics or metaheuristics depending on how they escape from local optima (Blum and Roli 2003). Regarding multi-objective optimization problems, the use of AI techniques becomes paramount due to their complexity (Coello Coello 2006).

Nowadays, the point of view for planning and scheduling tasks has changed. Due to the fact that real world is uncertain, imprecise and non-deterministic, there might be unknown information, breakdowns, incidences or changes, which become the initial plans or schedules invalid. Thus, there is a new trend to cope these aspects in the optimization techniques, and to seek robust solutions (schedules) (Lambrechts, Demeulemeester, and Herroelen 2008).

In this way, these optimization problems become harder since a new objective function (robustness measure) must be taken into account during the solution search. Therefore, the robustness concept is being studied and a general robustness measure has been developed for any scheduling problem (such as Job Shop Problem, Open Shop Problem, Railway Scheduling or Vehicle Routing Problem). To this end, in this thesis, some techniques have been developed to improve the search of optimized and robust solutions in

planning and scheduling problems. These techniques offer assistance to decision makers to help in planning and scheduling tasks, determine the consequences of changes, provide support in the resolution of incidents, provide alternative plans, etc.

As a case study to evaluate the behaviour of the techniques developed, this thesis focuses on problems related to container terminals. Container terminals generally serve as a transshipment zone between ships and land vehicles (trains or trucks). In (Henesey 2006a), it is shown how this transshipment market has grown rapidly. Container terminals are open systems with three distinguishable areas: the berth area, the storage yard, and the terminal receipt and delivery gate area. Each one presents different planning and scheduling problems to be optimized (Stahlbock and Voß 2008). For example, berth allocation, quay crane assignment, stowage planning, and quay crane scheduling must be managed in the berthing area; the container stacking problem, yard crane scheduling, and horizontal transport operations must be carried out in the yard area; and the hinterland operations must be solved in the landside area.

Furthermore, dynamism is also present in container terminals. The tasks of the container terminals take place in an environment susceptible of breakdowns or incidences. For instance, a Quay Crane engine stopped working and needs to be revised, delaying this task one or two hours. Thereby, the robustness concept can be included in the scheduling techniques to take into consideration some incidences and return a set of robust schedules.

In this thesis, we have developed a new domain-dependent planner to obtain more efficient solutions in the generic problem of reshuffles of containers. Planning heuristics and optimization criteria developed have been evaluated on realistic problems and they are applicable to the general problem of reshuffling in blocks world scenarios.

Additionally, we have developed a scheduling model, using constructive metaheuristic techniques on a complex problem that combines sequences of scenarios with different types of resources (Berth Allocation, Quay Crane Assignment, and Container Stacking problems). These problems are usually solved separately and their integration allows more optimized solutions.

Moreover, in order to address the impact and changes that arise in dynamic real-world environments, a robustness model has been developed for scheduling tasks. This model has been applied to metaheuristic schemes, which are based on genetic algorithms. The extension of such schemes, incorporating the robustness model developed, allows us to evaluate and obtain more robust solutions. This approach, combined with the classical optimality criterion in scheduling problems, allows us to obtain, in an efficient in way, optimized solution able to withstand a greater degree of incidents that occur in dynamic scenarios. Thus, a proactive approach is applied to the problem that arises with the presence of incidences and changes that occur in typical scheduling problems of a dynamic real world.

Resumen

A pesar de la continua evolución de los ordenadores y en la tecnología de la información, los problemas combinatorios de optimización del mundo real son problemas NP, en particular del dominio de la planificación y *scheduling*. Por eso, aunque técnicas exactas del campo de la investigación operativa, como la Programación Lineal, podrían ser aplicadas para resolver estos problemas de optimización, éstas son difíciles de aplicar a escenarios del mundo real ya que normalmente requieren demasiado tiempo de cómputo, por ejemplo,se requiere una solución optimizada en un tiempo de cómputo asequible. Además, los responsables a menudo se enfrentan a diferentes y típicamente objetivos opuestos, convirtiéndose en problemas de optimización multi-objetivo. Por lo tanto, técnicas aproximadas del campo de la Inteligencia Artificial (IA) normalmente se emplean para resolver los problemas del mundo real. Las técnicas de la IA proporcionan representaciones más ricas y flexibles del mundo real (Gomes 2000), y son ampliamente utilizados para resolver este tipo de problemas. Las técnicas heurísticas de IA no garantizan la solución óptima, sino que proporcionan soluciones cercanas a la óptima en un tiempo razonable. Estas técnicas se dividen en dos grandes grupos de clases de algoritmos: los métodos constructivos y los métodos de búsqueda local (Aarts and Lenstra 2003). Estos pueden guiar sus procesos de búsqueda a través de heurísticas o metaheurísticas dependiendo de como ellos escapan de los óptimos locales (Blum and Roli 2003). Haciendo referencia a los problemas de optimización multi-objetivo, el uso de las técnicas de IA pasan a ser imprescindibles debido a la complejidad de estos problemas (Coello Coello 2006).

Actualmente, el punto de vista para las tareas de planificación y *scheduling* ha cambiado. Debido a que el mundo real es incierto, impreciso y no determinístico, puede haber información desconocida, fallos, incidencias o cambios que convierten los planes iniciales invalidos. Por eso, hay una nueva tendencia para hacer frente a estos aspectos en las técnicas de optimización y buscar soluciones robustas (*schedules*) (Lambrechts, Demeulemeester, and Herroelen 2008).

De esta manera, estos problemas de optimización se vuelven más difíciles ya que una nueva función objetivo (medida de robustez) debe ser tenida en cuenta durante la búsqueda de solución. Por lo tanto, el concepto de robustez se ha estudiado y una medida de robustez general ha sido desarrollado para cualquier problema de *scheduling* (como Job Shop Problem, Open Shop Problem, Railway Scheduling o Vehicle Routing Problem). Con este fin, en esta tesis, algunas técnicas se han desarrollado para mejorar la búsqueda de soluciones optimizadas y robustas en problemas de planificación y *scheduling*. Estas técnicas ofrecen asistencia a los responsables para ayudar en las tareas de planificación y *scheduling*, determinar las consecuencias de los cambios, proporcionar asistencia para la resolución de incidentes, proporcionan planes alternativos, etc.

Como caso de estudio para evaluar el comportamiento de las técnicas desarrolladas, esta tesis se centra en problemas relacionados con terminales de contenedores. Las terminales de contenedores sirven como zona de transbordo entre los buques y otros medios de transporte (trenes o camiones). En (Henesey 2006a), se muestra como este mercado de transbordo ha crecido rápidamente. Las terminales de contenedores son sistemas abiertos con tres áreas distinguibles: el área del muelle, el patio de contenedores, y el área de la puerta de entrada y recepción de la terminal. Cada área presenta diferentes problemas de planificación y *scheduling* que deben ser optimizados (Stahlbock and Voß 2008). Por ejemplo, la asignación de muelles, la asignación de grúas, la planificación de la estiba de los buques, la planificación de las grúas deben ser gestionados en la zona del muelle; el problema del apilamiento de contenedores, la planificación de las grúas del patio y las operaciones del transporte horizontal deben ser llevados a cabo en el área del patio de contenedores; y, las operaciones con el interior del país deben ser resueltas en el área de la puerta de la terminal.

Las tareas de las terminales de contenedores tienen lugar en un entorno susceptible de fallos o incidencias. Por ejemplo, el motor de una grúa del muelle podría dejar de funcionar y necesitaría ser revisado, retrasando esta tarea una o dos horas. De ese modo, el concepto de robustez puede ser incluido en las técnicas de *scheduling* para tener en cuenta algunas incidencias y devolver un conjunto de planes robustos.

En esta tesis, se ha desarrollado un nuevo planificador dependiente del dominio a fin de obtener soluciones más eficientes en el problema genérico de la remoción de contenedores. Las heurísticas y criterios de optimización de planificación desarrollados han sido evaluados en problemas realistas y resultan aplicables para el problema genérico de remoción en escenarios del mundo de bloques.

Adicionalmente, se ha desarrollado un modelo de *scheduling*, aplicando técnicas metaheurísticas constructivas, sobre un complejo problema que combina secuencias de escenarios con distintas tipologías de recursos (Berth Allocation, Quay Crane Assignment, and Container Stacking problems). Estos problemas habitualmente son resueltos de forma desjunta y su integración permite obtener soluciones más optimizadas.

Por otra parte, a fin de tratar las incidencias y cambios que surgen en entornos dinámicos del mundo real, se ha desarrollado un modelo de robustez para tareas de *scheduling*. Este modelo se ha aplicado sobre esquemas metaheurísticos, basados en algoritmos genéticos.

La extensión de dichos esquemas, incorporando el modelo de robustez desarrollado, permite evaluar y obtener soluciones más robustas. Este criterio, combinado con el clásico criterio de optimalidad de los problemas de *scheduling*, permite obtener, de forma eficiente, soluciones optimizadas capaces de soportar un mayor grado de incidencias que ocurren en escenarios dinámicos. De esta forma, se aplica una aproximación proactiva al problema que surge con la presencia de incidencias y cambios, que ocurren en los típicos problemas de *scheduling* de un mundo real dinámico.

Resum

Tot i la contínua evolució dels ordinadors i la tecnologia de la informació, els problemes combinatoris d'optimització del món real són problemes NP, en particular del domini de la planificació i *scheduling*. Per això, encara que tècniques exactes del camp de la investigació operativa, com la Programació Lineal, podrien ser aplicades per a resoldre aquests problemes d'optimització, aquestes són difícils d'aplicar a escenaris del món real ja que normalment requereixen massa temps de còmput, per exemple, se requereix una solució optimizada en un temps de còmput assequible. A més, els responsables sovint s'enfronten a diferents i típicament objectius oposats, convertint-se en problemes d'optimització multi-objectiu. Per tant, tècniques aproximades del camp de la Intel·ligència Artificial (IA) normalment s'empren per resoldre els problemes del món real. Les tècniques de la IA proporcionen representacions més riques i flexibles del món real (Gomes 2000), i són àmpliament utilitzats per resoldre aquest tipus de problemes. Les tècniques heurístiques d'IA no garanteixen la solució òptima, sinó que proporcionen solucions pròximes a l'òptima en un temps raonable. Aquestes tècniques es divideixen en dos grans grups de classes d'algorismes: els mètodes constructius i els mètodes de cerca local (Aarts and Lenstra 2003). Aquests poden guiar els seus processos de cerca a través d'heurístiques o metaheurístiques depenent de com ells escapen dels òptims locals (Blum and Roli 2003). Fent referència als problemes d'optimització multi-objectiu, l'ús de les tècniques d'IA passen a ser imprescindibles per la complexitat d'aquests problemes (Coello Coello 2006).

Actualment, el punt de vista per a les tasques de planificació i *scheduling* ha canviat. Com que el món real és incert, imprecís i no determinístic, pot haver informació desconeguda, errors, incidències o canvis que converteixen els plans inicials invàlids. Per això, hi ha una nova tendència per fer front a aquests aspectes en les tècniques d'optimització i buscar solucions robustes (*schedules*) (Lambrechts, Demeulemeester, and Herroelen 2008).

D'aquesta manera, aquests problemes d'optimització es tornen més difícils ja que una nova funció objectiu (mesura de robustesa) s'ha de tenir en compte durant la recerca de la solució. Per tant, el concepte de robustesa ha de ser estudiat i una mesura de la robustesa

general ha estat desenvolupada per a qualsevol problema de *scheduling* (com Job Shop Problem, Open Shop Problem, Railway Scheduling o Vehicle Routing Problem).

Amb aquesta finalitat, en aquesta tesi, algunes tècniques s'han desenvolupat per millorar la recerca de solucions optimitzades i robustes en problemes de planificació i *scheduling*. Aquestes tècniques ofereixen assistència als responsables per ajudar en les tasques de planificació i *scheduling*, determinar les conseqüències dels canvis, proporcionar assistència per a la resolució d'incidents, proporcionar plans alternatius, etc

Com a cas d'estudi per avaluar el comportament de les tècniques desenvolupades, aquesta tesi se centra en problemes relacionats amb terminals de contenidors. Les terminals de contenidors serveixen com a zona de transbord entre els vaixells i altres mitjans de transport (trens o camions). A (Henesey 2006a), es mostra com aquest mercat de transbord ha crescut ràpidament. Les terminals de contenidors són sistemes oberts amb tres àrees distingibles: l'àrea del moll, el pati de contenidors, i l'àrea de la porta d'entrada i recepció de la terminal. Cada àrea presenta diferents problemes de planificació i *scheduling* que han de ser optimitzats (Stahlbock and Voß 2008). Per exemple, l'assignació de molls, l'assignació de grues, la planificació de l'estiba dels vaixells, la planificació de les grues han de ser gestionats a la zona del moll; el problema de l'apilament de contenidors, la planificació de les grues del pati i les operacions del transport horitzontal s'han de dur a terme en l'àrea del pati de contenidors; i, les operacions amb l'interior del país han de ser resoltes en l'àrea de la porta de la terminal.

Les tasques de les terminals de contenidors tenen lloc en un entorn susceptible d'errors o incidències. Per exemple, el motor d'una grua del moll podria deixar de funcionar i necessitaria ser revisat, retardant aquesta tasca una o dues hores. D'aquesta manera, el concepte de robustesa pot ser inclòs en les tècniques de *scheduling* per a tenir en compte algunes incidències i retornar un conjunt de plans robustos.

En aquesta tesi, s'ha desenvolupat un nou planificador dependent del domini per tal d'obtenir solucions més eficients en el problema genèric de la remoció de contenidors. Les heurístiques i criteris d'optimització de planificació desenvolupats han estat avaluats en problemes realistes i resulten aplicables per al problema genèric de remoció en escenaris del món de blocs.

A més, s'ha desenvolupat un model de *scheduling*, aplicant tècniques metaheurístiques constructives, sobre un complex problema que combina seqüències d'escenaris amb diferents tipologies de recursos (Berth Allocation, Quay Crane Assignment, and Container Stacking problems). Aquests problemes habitualment són resolts de forma desjunta i la seva integració permet obtenir solucions més optimitzades.

D'altra banda, per tal de tractar les incidències i canvis que sorgeixen en entorns dinàmics del món real, s'ha desenvolupat un model de robustesa per a tasques de *scheduling*. Aquest model s'ha aplicat sobre esquemes metaheurístics, basats en algorismes genètics.

L'extensió d'aquests esquemes, incorporant el model de robustesa desenvolupat, permet avaluar i obtenir solucions més robustes. Aquest criteri, combinat amb el clàssic criteri d'optimalitat dels problemes de *scheduling*, permet obtenir, de forma eficient, solucions optimitzades capaces de suportar un major grau d'incidències que ocorren en escenaris dinàmics. D'aquesta manera, s'aplica una aproximació proactiva al problema que sorgeix amb la presència d'incidències i canvis, que ocorren en els típics problemes de *scheduling* d'un món real dinàmic.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

1.1.1 Optimization in Planning & Scheduling

Large companies must make important decisions in different levels (Schmidt and Wilhelm 2000): strategic (long term), tactical (mid term) and operational (short term) (Figure 1.1). Examples of questions in each level might be:

- *Strategic decisions*: What is your vision for the business? Which direction is the business headed?
- *Tactical decisions*: Should the company increase the number of operators?
- *Operational decisions*: What is our procedure for delivering and order? How many quay cranes must be assigned to that container ship?

All these decisions involve many human and material resources (logistics) and need to be studied and planned carefully. Since they have reached a high degree of complexity, just operations experts or decision makers can not longer make these decisions. Companies need scientific methods to keep improving their activities.

Planning and scheduling are forms of decision making that are used on a regular basis in many manufacturing and service industries (Pinedo 2005). For instance, production, transportation and logistics, information processing, communication, etc. On the one hand, planning consists in, given an initial state and a finite set of actions, getting a plan

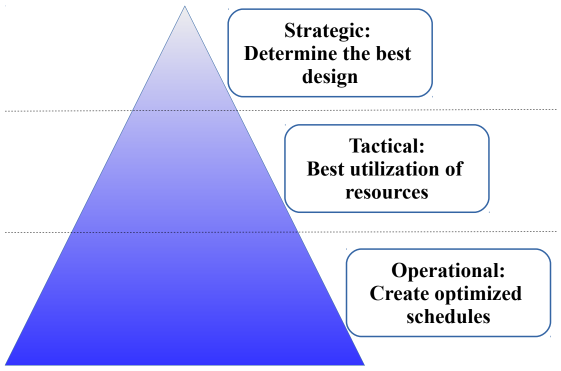

Strategic:
Determine the best design
Tactical:
Best utilization of resources
Operational:
Create optimized schedules

Figure 1.1: Representation of the decision levels

which reaches a goal state by choosing and establishing an ordering of those actions (Ghallab, Nau, and Traverso 2004). On the other hand, scheduling deals with assigning the given tasks according to the available resources and constraints over given time periods (Pinedo 2012). The goal of both planning and scheduling problems is to optimize one or more objective functions. Moreover, both planning and scheduling problems take advantage of mathematical techniques from the Operations Research field or heuristics and other complete techniques from the Artificial Intelligence field. In (Gomes 2000), it is showed how techniques from these two fields could be integrated to be competitive in real-world problems.

Operational research (OR) deals with the application of advanced analytical methods such as linear programming, queueing theory or simulation to help make better decisions.

Artificial Intelligence (AI) provides techniques which are able to produce near-optimal solutions in a reasonable computational time. Among AI techniques, metaheuristics such as Simulated Annealing (SA) (Kirkpatrick, Gelatt, and Vecchi 1983), Genetic Algorithms (GA) (Goldberg 1985) or Tabu Search (Glover and Laguna 1999) have been applied successfully to a wide range of different real-world combinatorial problems. Each of these metaheuristics has its own historical background. Some metaheuristics are inspired from optimization processes that occur in the nature such as evolution (e.g. genetic algorithms) or statistical mechanics (e.g. simulated annealing). Others are extensions of less sophisticated algorithms such as greedy heuristics and local searches (Greedy Randomized Adaptive Search Process or GRASP (Feo and Resende 1995)). A classification of the most common metaheuristics in the literature is depicted in Figure 1.2.

Over the last years, new algorithms are originated from algorithms of different research areas on optimization, exploiting their different optimization strategies. These new algorithms are commonly referred to as hybrid metaheuristics (Blum, Puchinger, et al. 2011).

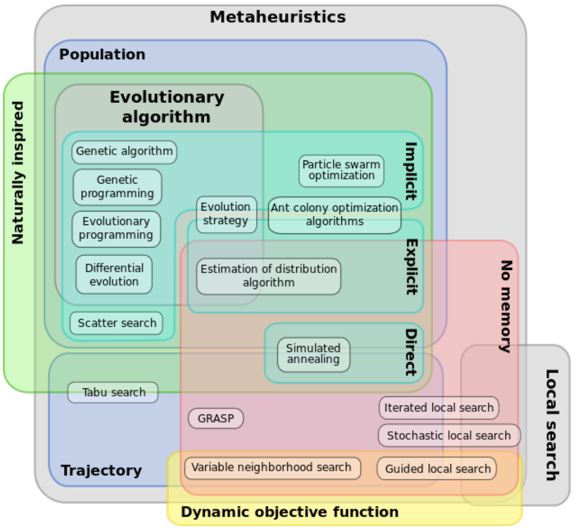


Figure 1.2: Classification of metaheuristics (image from http://en.wikipedia.org/wiki/Metaheuristic).

For instance, metaheuristics which are population-based (e.g. evolutionary algorithms or ant colony algorithms) are good concerning the exploration of the search space and therefore they are good to find promising regions of the search space. However, they are usually not so effective concerning the exploitation. On the other hand, local search methods (e.g. simulated annealing or tabu search) are good to find better solutions in the neighborhood of given starting solutions. Thereby, a hybrid metaheuristic composed by a population-based and a local search methods will identify the promising regions of the search space and find the good solutions in those regions.

All these previous algorithms just take into account single optimization problems. Nevertheless, real-world optimization problems commonly involve more than one conflicting objective function to be optimized. In a multi-objective optimization problem, usually there is no single solution wherein all its objectives are simultaneously optimized.

Example. *The routing school bus problem has two different objectives: (1) minimizing the total number of buses (operational cost); and, (2) minimizing the longest time a student would have to stay in the bus, or in other words, minimizing the maximum route length (service level) (Pacheco and Martí 2006).*

However, there may exist a set of Pareto optimal solutions with different trade-offs among their objective functions. Pareto efficient, or Pareto optimal, is a solution in which is impossible to make any one criteria better off without making at least one criteria worse off (Zhou et al. 2011).

The Pareto optimal set is the set of all the solutions that are Pareto optimal solution (Zhou et al. 2011). In general, generating the Pareto optimal set is expensive computationally and it is often impracticable. Therefore, algorithms try to find a good approximation of the Pareto optimal set. In this work, we refer to each approximation as Pareto front, which contains solutions that, although are non-dominated among them, could be dominated by other solutions not found by our algorithms. An example of a Pareto front is showed in Figure 1.3.

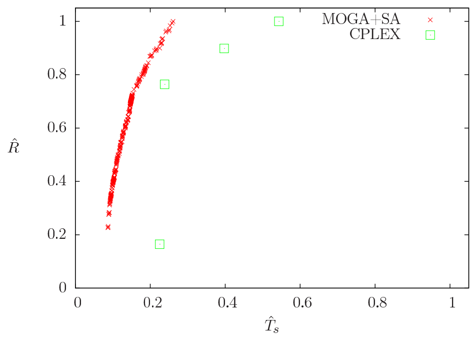


Figure 1.3: Example of a Pareto front in Berth Allocation and Quay Crane Assignment problem (Rodriguez-Molins, Salido, and Barber 2014b).

As mentioned above, multi-objective optimization problems do not have a unique optimal solution. Thus, multi-objective optimization algorithms cannot be compare directly. (Zitzler, Knowles, and Thiele 2008) propose different measures to compare Pareto front approximations. Among these measures, the size of the dominated space or the hypervolume of the Pareto front approximations obtained by the algorithms is one of the most used measures to compare two different algorithms (While, Bradstreet, and Barone 2012).

As in single optimization problems, hybrid metaheuristics have been also designed for multi-objective optimization problems (Ehrgott and Gandibleux 2008). These hybrid metaheuristics take advantage from each algorithm or techniques on which they are based on for achieving a better representation of the Pareto front. Examples of multi-objective hybrid metaheuristics are Multiple Objective Genetic Tabu Search (MOGTS) (Gambardella, Taillard, and Agazzi 1999) and Multi-Colony Ant System (MCAS) (Barichard and Hao 2002).

In optimization problems, there is a new trend focused on finding near-optimal solutions by handling uncertainty, dynamism or unknown information of the real world environtments. These strategies are classified mainly in two classes: proactives and reactives (Lambrechts, Demeulemeester, and Herroelen 2008). On the one hand, the objective of a proactive approach is to build robust schedules which could keep being valid after unexpected events during the schedule's execution. On the other hand, reactive approaches create a baseline schedule and then, if any unexpected event occurs, these baseline schedules are revised or re-optimized.

In this thesis, we have focused our attention on proactive strategies, and most of the proactive approaches in the literature make use of multi-objective optimization techniques to build robust schedules (Surico et al. 2007; Zhen and Chang 2012).

1.1.2 Contributions of this thesis

The first contribution of this thesis is the development of a domain-dependent planner and its comparison with domain-independent and domain-dependent planners. As domain-dependent planners can take advantage of the knowledge of the own problems, they could offer better solutions with less computational time. In this thesis, a domain-dependent planner has been developed to solve a real-world problem: the Container Stacking problem or Remarshalling problem (Salido, Sapena, et al. 2009; Rodriguez-Molins, Salido, and Barber 2012; Salido, Rodriguez-Molins, and Barber 2012). Furthermore, it is important to note that these domain-dependent planners offer the chance to handle different optimization criteria to cope different real-world scenarios.

The second contribution of this thesis is related to the development of a scheduling model, using constructive and population-based metaheuristics and mathematical models (Rodriguez-Molins, Barber, et al. 2012; Rodriguez-Molins, Salido, and Barber 2014a), on a complex problems that combines sequences of scenarios with different optimization problems: the above mentioned Container Stacking problem, and Berth Allocation and Quay Crane Assignment problem (BAP+QCAP) (Salido, Rodriguez-Molins, and Barber 2011; Salido, Rodriguez-Molins, and Barber 2012). These problems are usually solved separately, but it is likely that focusing on optimizing just one problem will not lead to the optimal solution. Thereby, their integration allows to obtain more efficient solutions and thus, a decision support system has been designed to help the decision makers and offer efficient solutions to all these problems at the same time.

The last contribution of this thesis is a proactive approach which is based on a new robustness model to deal with dynamism, uncertain or imprecise environments in scheduling problems without any previous knowledge about the incidences (Rodriguez-Molins, Salido, and Barber 2014b). This robustness model is based on the available operational buffers after each task related to the durations of these tasks. In general, the greater the operational buffers, the higher the robustness of the schedule. However, due to the lack of the knowledge about incidences, operational buffers should be distributed among vessels proportionally to be able to absorb as many incidences as possible. The aim of this proactive approach is to offer a wide range of different schedules to the decision makers where the trade-off between the classical optimality criterion and the robustness measure have been taken into consideration. These schedules must be part of an approximation of the Pareto optimal set. To this end, a multi-objective approach was considered to seek an approximation of the Pareto optimal set by means of a mixed-integer lineal programming model and a hybrid multi-objective genetic algorithm. The developed proactive approach was applied to and evaluated with the BAP+QCAP.

1.1.3 Case study: Container Terminals

Throughout our study, we use the Port of Valencia, specifically the Valencian Port Foundation and the maritime container terminal MSC (Mediterranean Shipping Company S.A.) as a case of reference, although our models are applicable to any container terminal.

The overall collaboration goal of our group at the Universitat Politècnica de València (UPV) with the Valencia Port Foundation and the maritime container terminal MSC (Mediterranean Shipping Company S.A.) is to offer assistance and help in the planning and scheduling tasks such as the allocation of spaces to outbound containers, to identify bottlenecks, to determine the consequences of changes, to provide support in the resolution of incidents, to provide alternative berthing plans, etc. Thus, the development of the techniques presented in this thesis will provide the terminal operators with different tools to simulate new configurations or settings for their container terminal (*what if. . . ?* questions) as well as to provide several plans which are able to deal with different scenarios, e.g. plans to integrate berthing allocation and container stacking problem or even berth schedules which keep valid after the ocurrence of incidences.

Containers were standardized by the International Organization for Standardization (ISO) based upon the US Department of Defense standards between 1968 and 1970, ensuring interchangeability between different modes of transportation worldwide. The standard sizes and fitting and reinforcement norms that exist now evolved out of a series of compromises among international shipping companies, European railroads, U.S. railroads, and U.S. trucking companies.

Four important ISO recommendations helped the standardization of the containerisation globally (Rushton, Croucher, and Baker 2006):

R-668 in January 1968 defined the terminology, dimensions and ratings;

R-790 in July 1968 defined the identification markings;

R-1191 in January 1970 made recommendations about corner fittings;

R-1897 in October 1970 set out the minimum internal dimensions of general-purpose freight containers.

The standardization of the containerization allowed the interchangeability between different modes of transport around the world (trucks, ships, trains and even aircrafts). Furthermore, it made managing multiple products easier, improved security, reduced loss

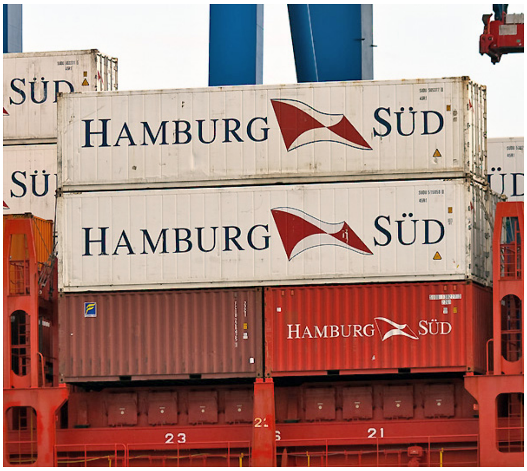


Figure 1.4: Comparison between containers of 20ft and 40ft (image from http://en.wikipedia.org/wiki/Twenty-foot_equivalent_unit).

and damage and above all speed up the whole process of freight transportation (Rushton, Croucher, and Baker 2006).

The term *twenty-foot equivalent unit* (TEU) is used to refer to one container with a length of twenty foot (approximately 6.1 meters). Thereby, a container of 40 feet (forty-foot equivalient unit or FEU) is expressed by 2 TEU (see Figure 1.4). This measure is also used to identify the capacity of the vessels, e.g. the vessel *Triple-E MÆRSK* (first delivered in 2013) is one of the largest vessels which can carry 18340 TEU[1]. Figure 1.5 showed how container ships have evolved over the last years.

Container terminals are open systems which generally serve as a transshipment on ships and land vehicles (trains or trucks) for containers. Operations related to move these containers can be divided into four different subsystems (Henesey 2006b) (see Figure 1.6):

1. *ship-to-shore* movements to unload the containers from ship to berth (or in reverse order, to load them onto the ship). Quay cranes are assigned to ships to perform these movements (loading/unloading).

2. *transfer* bi-directional movement of containers from berth to stack (storage area), from one stack to another stack and from the hinterland (or gate) to a stack;

3. *storage* stack or area where containers are placed to wait until their next ship, train or truck; and

[1]Updated on July 2014,
http://www.maerskline.com/link/?page=brochure&path=/our_services/vessels

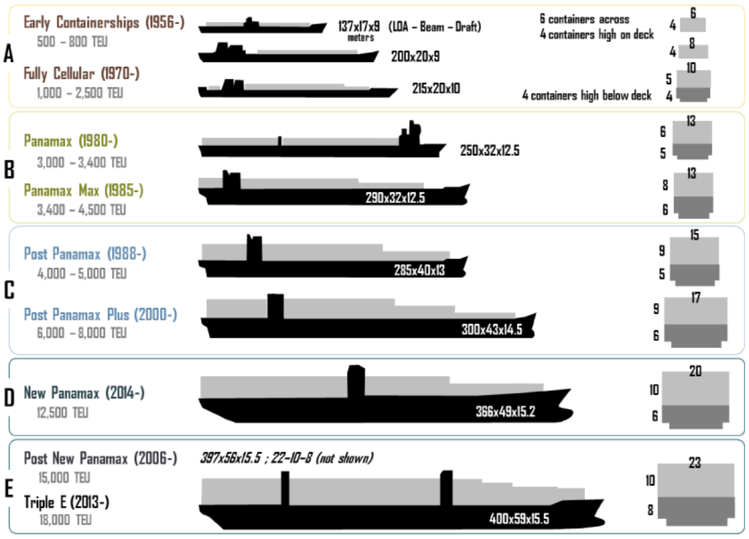


Figure 1.5: Evolution of container ships (image from http://people.hofstra.edu/geotrans/eng/ch3en/conc3en/containerships.html)

4. *delivery/receipt* movement of containers from stack to the hinterland transport, and vice versa.

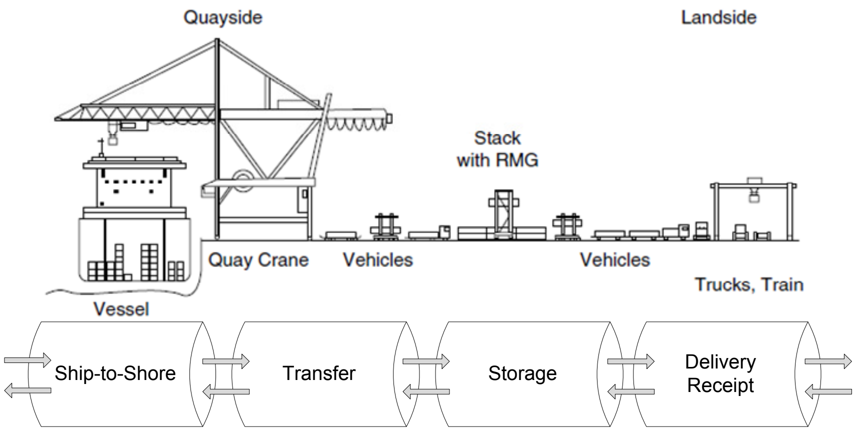
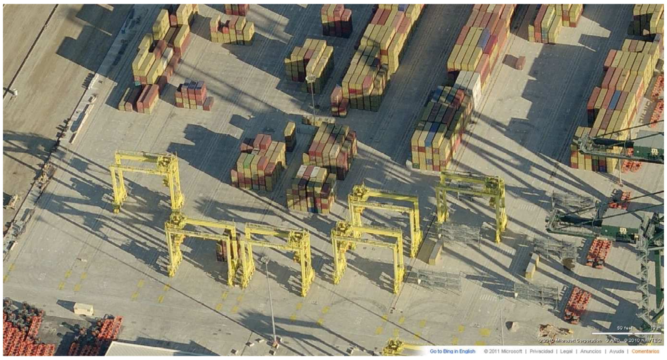
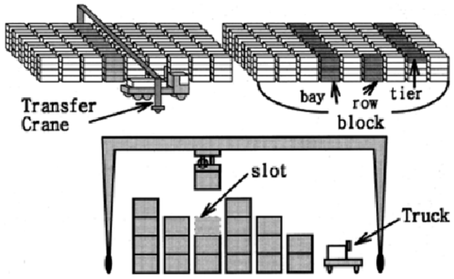


Figure 1.6: Subsystems in a Container terminal.

The different processes that a container terminal must perform can be described following the steps a container does within one terminal. All these activities are described in detail in (Vis and De Koster 2003).

Succinctly, when a container ship (or vessel) arrives at the port, it must wait at the roadstead until the permission is granted to moor at the quay. This permission is granted when all the resources needed are available (quay cranes, berth length, labour, etc.). Once the container ship moors, all the import containers must be unloaded to the container yard. This process is carried out by means of different handling equipments: quay cranes (QCs), automated guided vehicles (AGV), straddle carriers (SCs) and rubber-tired cranes (RTGs) among others. A description and a brief description of the handling equipments used in container terminals can be found at (Henesey 2006b; Stahlbock and Voß 2008). All this equipment must be handled wisely to avoid bottlenecks and congestions in the yard. For a detailed overview about the internal transport operations and material handling equipment see (Carlo, Vis, and Roodbergen 2014).


(a) Aerial view of a storage yard (courtesy of MSC).



(b) Schema of a block of containers.

Figure 1.7: Storage yard area.

The container yard consists of a number of lanes or blocks, and each block consists of 20-30 yard-bays (Figure 1.7(a)). Each yard-bay usually contains 6 rows (width) and a maximum height of 4 or 5 tiers, where containers can be stored for a certain period of time (Figure 1.7(b)). After this certain period, containers are retrieved to be loaded into other transportation modes (trucks, trains or barges) or even other container ships. This retrieval operation must be planned carefully in order to increase the handling time of the container ships as little as possible (Kim and Bae 1998). Containers should be stacked in the same order as the loading operation needs to be, but at the arrival of the containers this order is not available. Thereby, the layout of the block of the containers should be reorganized in order to speed up the loading operations of the incoming vessels. This operation is known as Remarshalling problem.

With 80 per cent of global merchandise trade by volume carried by sea and handled by ports worldwide, the competition among container terminals and the global economic crisis of the 2008/2009 make necessary to increase productivity and container throughput from quayside to landside and vice versa as well as ensure reliability (delivery dates or handling times) to the shipping companies. Thereby, as suggested in (Asariotis et al. 2013), investments in ports will lead to increases in efficiency diminishing transport costs by optimizing all the operations within a container terminal. (Vis and De Koster 2003; Steenken, Voß, and Stahlbock 2004; Stahlbock and Voß 2008) provide an extensive survey about operations at seaport container terminals.

1.2 Objectives

The main objective of this thesis is to contribute to the state of the art, by developing new robust planning and scheduling optimization techniques. The techniques developed will be applied in a case study related to container terminal problems. The detailed objectives can be summarized as:

- Review the single and multi-objective optimization techniques proposed in the literature for planning and scheduling problems, by analyzing their advantages and disadvantages.
- Review the two main strategies (proactive and reactive) reported in the literature to handle the uncertainty or dynamism of real-world environments.
- Review and analyze the robustness of the schedules from the proactive perspective.
- Develop new optimization planning techniques with domain-dependent heuristics.
  - Define the metric to evaluate the obtained plans.
  - Define domain-dependent heuristics to be applied to planning tasks within container terminals (Remarshalling tasks) .
  - Evaluate and compare the domain-dependent planner with a general and well-known planner: Metric-FF (Hoffmann 2003).

- Develop new optimization scheduling techniques to be applied to container terminals:
  - Define the metric to evaluate the obtained schedules.
  - Develop mathematical models to obtain the optimal schedules.
  - Apply these techniques to tasks within container terminals: Berth Allocation and Quay Crane Assignment Problems.
  - Evaluate and compare these techniques .
- Design a proactive approach with a new robustness model as well as develop new robust optimization techniques (multi-objective optimization) to be applied to container terminals:
  - Define and normalize the metric to evaluate (robust measure) how robust the obtained schedules are.
  - Develop mathematical models to obtain robust schedules.
  - Apply these techniques to tasks within container terminals: Berth Allocation and Quay Crane Assignment Problems.
  - Evaluate and compare the developed techniques.

1.3 Structure

This thesis is structured in four chapters organized as follows:

- **Chapter 1. Introduction**: In this chapter, the state of the art and the motivation of this thesis are described. It also presents the real-world environment chosen as case study: container terminals. All the developed techniques were applied and assessed using this case study.
- **Chapter 2. Selected Papers**: This chapter consists on a collection of articles (conferences and journals) published by the PhD. student which support this thesis. The articles are divided according to their scope.
- **Chapter 3. General Discussion of the Results**: This chapter provides a discussion about all the obtained results presented in the articles from the previous chapter.
- **Chapter 4. Conclusions**: This last chapter is devoted to present a final review of the conclusions as well as promising directions for further works.

1.4 Publications List

In this section, all the international publications related to this thesis are listed. They are classified according to their type (journals or international conferences) as well as whether they are listed in JCR[2] or in CORE[3], respectively.

- Journals listed in JCR.
  - Salido, M.A. and Rodriguez-Molins, M. and Barber, F.. *Integrated intelligent techniques for remarshaling and berthing in maritime terminals.* **Advanced Engineering Informatics**, Elsevier, 25(3), 435–451 (2011). (Q2, JCR: 1.489) DOI: 10.1016/j.aei.2010.10.001
  - Salido, M.A. and Rodriguez-Molins, M. and Barber, F.. *A Decision Support System for Managing Combinatorial Problems in Container Terminals.* **Knowledge Based Systems**, Elsevier, 29(0), 63–74 (2012). (Q1, JCR: 4.104) DOI: 10.1016/j.knosys.2011.06.021
  - Rodriguez-Molins, M. and Salido, M.A. and Barber, F.. *Intelligent Planning for Allocating Containers in Maritime Terminals.* **Expert Systems with Applications**, Elsevier, 39(1), 978–989 (2012). (Q2, JCR: 1.854) DOI: 10.1016/j.eswa.2011.07.098
  - Rodriguez-Molins, M. and Salido, M.A. and Barber, F.. *A GRASP-based Metaheuristic for the Berth Allocation Problem and the Quay Crane Assignment Problem by Managing Vessel Cargo Holds.* **Applied Intelligence**, Springer US, 40(2), 273–290 (2014). (Q2, JCR 2012: 1.853) DOI: 10.1007/s10489-013-0462-4
  - Rodriguez-Molins, M. and Salido, M.A. and Barber F.. *Robust scheduling for Berth Allocation and Quay Crane Assignment Problem.* **Mathematical Problems in Engineering**, 1–17 (2014). (JCR 2013: 1.082). DOI: 10.1155/2014/834927
- Other international journals.
  - Rodriguez-Molins, M. and Ingolotti, L. and Barber F. and Salido, M.A. and Sierra, M.R. and Puente J.. *A Genetic Algorithm for Robust Berth Allocation and Quay Crane Assignment.* **Progress in Artificial Intelligence**, Springer Berlin Heidelberg, 2(4), 177–192 (2014). DOI: 10.1007/s13748-014-0056-3

[2]Journal Citation Reports http://thomsonreuters.com/journal-citation-reports/
[3]Conference Ranking Exercise http://core.edu.au/index.php/categories/conference%20rankings/1

- International conferences listed in CORE.

  - Salido, M.A. and Sapena, O. and Rodriguez, M. and Barber, F.. *A Planning Tool for Minimizing Reshuffles in Container Terminals.* 21st International Conference on Tools with Artificial Intelligence, 2009. ICTAI'09, 567–571 (2009). (CORE B, CSC Ranking: 0.74)
    DOI: 10.1109/ICTAI.2009.53

  - Rodriguez-Molins, M. and Salido, M. and Barber, F.. *Domain-Dependent Planning Heuristics for Locating Containers in Maritime Terminals.* The Twenty Third International Conference on Industrial, Engineering & Other Applications of Applied Intelligent Systems IEA-AIE 2010, LNCS/LNAI 6096, 742–751 (2010). (CORE B, CSC: top 5 in Industrial Engineering Conference Ranking)
    DOI: 10.1007/978-3-642-13022-9_74

  - Salido, M. and Rodriguez-Molins, M. and Barber, F.. *Artificial Intelligence Techniques for the Integration of Berth Allocation and Container Stacking Problems in Container Terminals.* AI-2010 Thirtieth SGAI International Conference on Artificial Intelligence, 295–308 (2010). (CORE C)
    DOI: 10.1007/978-0-85729-130-1_23

- Other international conferences.

  - Salido, M. and Rodriguez-Molins, M. and Barber, F.. *Towards the Integration of Berth Allocation and Container Stacking Problems in Martime Container Terminals.* The International Conference on Harbor, Maritime & Multimodal Logistics Modelling and Simulation, HMS 2010, 79–84 (2010).

  - Rodriguez-Molins, M. and Barber, F. and Sierra, M. and Puente, J. and Salido, M.. *A Genetic Algorithm for Berth Allocation and Quay Crane Assignment.* Advances in Artificial Intelligence - IBERAMIA 2012, Springer Berlin Heidelberg, LNCS/LNAI 7637, 601–610 (2012).
    DOI: 10.1007/978-3-642-34654-5_61

1.5 Abbreviations and Acronyms

AI	Artificial Intelligence
BAP	Berth Allocation Problem
CSP, CStackP	Container Stacking Problem
CT	Container Terminal
FCFS	First-Come, First-Served
FEU	Forty-foot equivalent unit
GA	Genetic Algorithm
GRASP	Greedy Randomized Adaptive Search Procedure
ISO	International Organization for Standardization
LIFO	Last-In, Last-Out
LP	Linear Programming
LS	Local Search
MILP	Mixed Integer Linear Programming
MOGA	Multi-Objective Genetic Algorithm
MSC	Mediterranean Shipping Company S.A.
OC	Optimization Criteria
OR	Operations Research
PDDL	Planning Domain Definition Language
QC	Quay Crane
QCAP	Quay Crane Assignment Problem
RMG	Rail Mounted Gantry crane
RTG	Rubber Tyre Gantry crane
SA	Simulated Annealing
SC	Straddle Carrier
TEU	Twenty-foot equivalent unit
TS	Tabu Search
UPV	Universitat Politècnica de València

Chapter 2

Selected papers

2.1 Planning: Container Stacking Problem

A Planning Tool for Minimizing Reshuffles in Containers Terminals

Miguel A. Salido, Oscar Sapena, Mario Rodriguez, Federico Barber
Instituto de Automática e Informática Industrial.
Universidad Politécnica de Valencia.
Valencia, Spain

Abstract

One of the more important problems in container terminal is related to the Container Stacking Problem. A container stack is a type of temporary store where containers await further transport by truck, train or vessel. The main efficiency problem for an individual stack is to ensure easy access to containers at the expected time of transfer. Since stacks are 'last-in, first-out', and the cranes used to relocate containers within the stack are heavily used, the stacks must be maintained in a state that minimizes on-demand relocations. In this paper, we present a new domain-dependent planning heuristic for finding the best configuration of containers in a bay. Thus, given a set of outgoing containers, our planner minimizes the number of relocations of containers in order to allocate all selected containers in an appropriate order to avoid further reshuffles.

1 Introduction

Loading and offloading containers on the stack is performed by cranes. In order to access a container which is not at the top of its pile, those above it must be relocated. This reduces the productivity of the cranes.

Maximizing the efficiency of this process leads to several requirements. First, each incoming container should be allocated a place in the stack which should be free and supported at the time of arrival. Second, each outgoing container should be easily accessible, and preferably close to its unloading position, at the time of its departure. In addition, the stability of the stack puts certain limits on, for example, differences in heights in adjacent areas, the placement of empty and 'half' containers and so on.

Since the allocation of positions to containers is currently done more or less manually, this has convinced us that it should be possible to achieve significant improvements of lead times, storage utilization and throughput using appropriate and improved techniques.

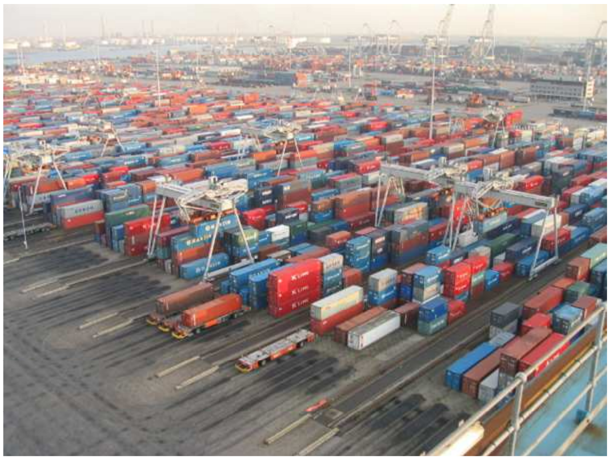


Figure 1: A container yard (courtesy of Hi-Tech Solutions)

Figure 1 shows a container yard. A yard consists of several blocks, and each block consists of 20-30 yard-bays [6]. Each yard-bay contains several (usually 6) rows. Each row has a maximum allowed tier (usually tier 4 or tier 5 for full containers). Figure 2 shows a transfer crane that is able to move a container within a stacking area or to another location on the terminal. For safety reasons, it is usually prohibited to move the transfer crane while carrying a container [7], therefore these movements only take place in the same yard-bay.

When an outside truck delivers an outbound container to a yard, a transfer crane picks it up and stacks it in a yard-bay. During the ship loading operation, a transfer crane picks up the container and transfers it to a truck that delivers it to a quay crane.

In container terminals, the loading operation for export containers is carefully preplanned by load planners. For load planning, a containership agent usually transfers a load profile (an outline of a load plan) to a terminal operating company several days before a ship's arrival. The load profile specifies only the container group, which is identified by container type (full or empty), port of destination, and size to be stowed in each particular ship cell. Since a ship cell can be filled with any container from its assigned group, the handling effort in the marshalling yard can be made easier by optimally sequencing export containers in the yard for the loading operation. In sequencing the containers, load planners usually pursue two objectives:

1. Minimizing the handling effort of quay cranes and yard equipment.

2. Ensuring the vessel's stability.

The output of this decision-making is called the ”load sequence list”. In order to have an efficient load sequence, storage layout of export containers must have a good

Figure 2: A Rubber-tired gantry crane (courtesy of Kalmar Industries).

configuration. The main focus of this paper is optimally reallocating outgoing containers for the final storage layout from which a load planner can construct an efficient load sequence list. In this way, the objective is therefore to plan the movement of the cranes so as to minimize the number of reshuffles of containers.

Given a layout, the user selects the set of containers that will be moved to the vessel. Our tool is able to organize the layout in order to allocate these containers at the top of the stacks in order to minimize the number of relocations. Thus a solution of our problem is a layout where all outgoing containers can be available without carrying out any reshuffle.

2 The problem modeled as an Artificial Intelligence planning problem

A classical AI planning problem can defined by a tuple $\langle A, I, G\rangle$, where A is a set of actions with preconditions and effects, I is the set of propositions in the initial state, and G is a set of propositions that hold true in any goal state. A solution plan to a problem in this form is a sequence of actions chosen from A that when applied transform the initial state I into a state of which G is a subset.

The container stacking problem is a slight modification of the *Blocks World* planning domain [8], which is a well-known domain in the planning community. This domain consists of a finite number of blocks stacked into towers on a table large enough to hold them all. The positioning of the towers on the table is irrelevant. The *Blocks World* planning problem is to turn an initial state of the blocks into a goal state, by moving one block at a time from the top of a tower onto another tower (or on a table). The optimal *Blocks World* planning problem is to do so in a minimal number of moves.

This problem is closed to the container stacking problem, but there are some important differences:

- The number of towers is limited in the container stacking problem: a yard-bay contains usually 6 rows, so it is necessary to include an additional constraint to limit the number of towers on the table to 6.

- The height of a tower is also limited.

- The main difference is in the problem goal specification. In the *Blocks World* domain the goal is to get the blocks arranged in a certain layout, specifying the final position of each block. In the container stacking problem the goal state is not defined as accurately, so many different layouts can be a solution for a problem. The goal is that the most immediate containers to load are in the top of the towers, without indicating which containers must be in each tower.

We can model our problem by using the standard encoding language for classical planning tasks called *PDDL* (Planning Domain Definition Language) [4]. Following this standard, a planning task is defined by means of two text files: the domain file, which contains the common features for all problems of this type, and the problem file, which describes the particular characteristics of each problem.

2.1 The container stacking domain

The main elements in a domain specification are (1) the types of objects we need to handle, (2) the types of propositions we use to describe the world, (3) the actions we can perform to modify the state of the world and (4) the function to optimize which is the number of relocations movements. For our proposals, we are going to consider two separate domains, D_1 and D_2, with a different complexity degree. In the first domain D_1, all containers have the same properties and the objective is to place a selected subset of containers, which must be loaded into the next vessel, on top of the stacks. In the second domain D_2, four subsets of containers are selected according to their departure time: *contF, contE, contM, contL*. *contF* is the first set of containers to leave, the next ones are *contE*, then *contM* and the last ones *contL*. In Figure 3 they are identified by means of different colors: red, light gray, dark gray and black, respectively. The red ones (*contF* containers) are the first ones to be loaded into the next vessel and, as it can be observed in the final layout in the figure, they are located on top of the stacks to facilitate their loading.

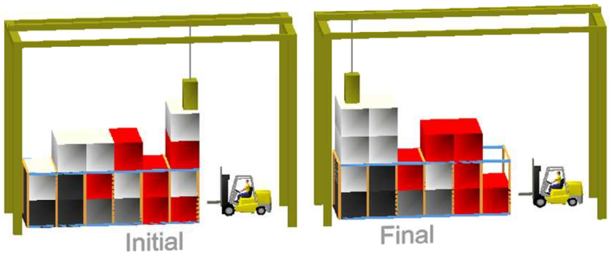


Figure 3: Initial state example (left), and final layout achieved (right) in the domain D_2.

3 A domain-dependent planning heuristic for solving the container stacking problem

Since the container stacking problem can be formalized in *PDDL* format, we can use a general planner to solve our problem instances. Currently we can found several general planners which work well in many different domains, such as *LPG-TD* [3], *MIPS-XXL* [2] and *SGPlan* [1]. However, and due to the high complexity of the domain we are handling, these planners are not able to find good plan solutions efficiently. *LPG-TD*, for example, spends too much time in the preprocessing stages, so it takes a long time to provide a solution. On the contrary, *MIPS-XXL* and *SGPlan* can compute a solution rapidly, but the quality of the obtained solution is not good enough, including some additional relocation movements to achieve the goal configuration.

Algorithm 1: Pseudo-code of the domain-dependent heuristic function

```
Data: $s$: state to evaluate
Result: $h$, heuristic value of $s$
1  $h$ = 0;
2  if $\exists$ $x$ - container / holding($x$) $\in s$ then
3      if goal-container($x$) then
4          $h$ = 0.1;
5      else
6          $h$ = 0.5;
7      end
8  end
9  for each row $r$ in the yard-bay do
10     $\Delta$h = 0;
11     for $x$ - container / at($x$, $r$) $\in s \wedge$ goal-container($x$) do
12         if $\nexists$ $y$ - container / goal-container($y$) $\wedge$ on($y$, $x$) $\in s$ then
13             $\Delta$h = max($\Delta$h, numContainersOn($x$));
14         end
15     end
16     $h$ += $\Delta$h;
17 end
```

We have implemented a local search domain-independent planner which can solve quite efficiently many problem instances. This planner has several interesting properties for the container stacking problem:

- It is an anytime planning algorithm [9]. This means that the planner can found a first, probably suboptimal, solution quite rapidly and that this solution is being improved while time is available.
- The planner is complete, so it will always find a solution if exists.
- The planner is optimal. It guarantees finding the optimal plan if there is time enough for computation.

This planner follows an enforced hill-climbing [5] approach with some modifications:

- We apply a best-first search strategy to escape from plateaux. This search is guided by a combination of two heuristic functions and it allows the planner to escape from a local minima very efficiently.
- If a plateau exit node is found within a search limit imposed, the hill-climbing search is resumed from the exit node. Otherwise, a new local search iteration is started from the best open node (the one with the best heuristic value).

This planner solves many problem instances but it can take too much time to find a solution in the hardest problems (usually when the number of containers is high and many reshuffles are required to achieve a goal layout). To improve the planning performance we have designed a new heuristic function specific for this domain. This heuristic computes an estimate of the number of container movements that must be carried out to reach a goal state (see Algorithm 1). Replacing the traditional heuristic function, based on a relaxed planning graph [5], by this domain-dependent heuristic function we outstandingly improve the planner performance: it solves many more problems and finds better quality plans with considerably less search effort. The plan is returned by the planner as a totally ordered sequence of actions that the transfer crane must carry out to achieve our objective.

3.1 Heuristic improvement

The main goal in the container stacking problem is to minimize the number of reshuffles required to reach a valid final layout. However, several different layouts can be usually achieved making the same number of reshuffles and some of them can be more interesting than the rest according to other important questions:

- It can be interesting to minimize the distance of the goal containers to the right side of the yard-bay, where the transfer crane is located. Achieving this we can spend considerably less time during the truck loading operations.
- It also could be interesting to balance the heights of the stacks to increase the containers stability.

These additional optimization functions have been easily incorporated in our planner by defining the heuristic function as a linear combination of two functions: h(s) = $\alpha * h_1(s) + \beta * h_2(s)$, where:

- $h_1(s)$ is the main heuristic function, which estimates the number of movements required to reach the goal layout (outlined in Algorithm 1). Since this is the main optimization function, α value should be significantly higher than β.
- $h_2(s)$ is the secondary function we want to optimize. This can be, for example, the sum of the distances of the selected containers to the right side of the yard-bay, which can be computed as Algorithm 2 shows.

Algorithm 2: Pseudo-code to calculate the distance

```
Data: $s$: state to evaluate
Result: $d$, distance value of $s$
1  $d = 0$;
2  for each row $r$ in the yard-bay do
3      for $x$ - container / at($x$, $r$) $\in s \wedge$ goal-container($x$) do
4          $d = d + (numRows(s) - r)$;
5      end
6  end
```

The benefits of using this combined heuristic function can be observed in Figure 4 and Figure 5. In the first one we want only to minimize the number of reshuffles, i.e. $h(s) = h_1(s)$. In the second one, we also want to minimize the distance of the selected containers to the forklift truck, so we have set $h(s) = 9 * h_1(s) + h_2(s)$. As a result, none of the selected containers (the red ones) are placed in the most left rows, reducing the required time to load the truck.

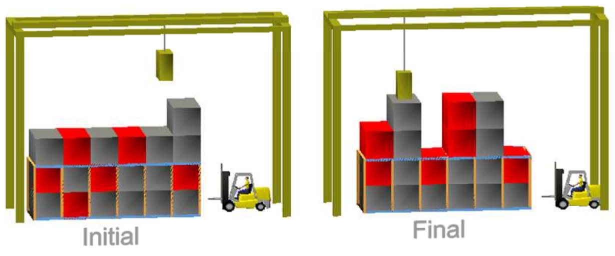


Figure 4: Obtained plan with the initial domain-dependent heuristic.

4 Evaluation

In this section, we have evaluated the minimum number of reshuffles needed to allocate all selected containers at the top of the stacks or under another selected containers in such a way that no reshuffles is needed to load outgoing containers.

The experiments were performed on random instances. A random instance is characterized by the tuple $< n, s >$, where n is the number of containers and s is the number of

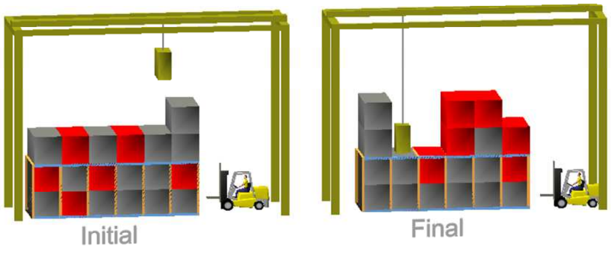


Figure 5: Obtained plan with the distance domain-dependent heuristic.

selected containers. Each instance is a random configuration of all containers distributed along the six stack with 4 or 5 tiers. We evaluated 100 test cases for each type of problem.

In Table 1, we present the average running time (in milliseconds) to achieve a solution in both a domain-independent heuristic and our domain-dependent heuristic in problems $< n, 4 >$. Thus, we fixed the number of selected containers to 4 and we increased the number of containers n from 15 to 20. It can be observed that our new domain-dependent heuristic is able to find a solution in a few milliseconds, meanwhile the domain-independent heuristic needs some more time for finding the first solution.

Table 1: Running time of the domain-independent heuristic and our domain-dependent heuristic in problems $< n, 4 >$ with 4 tiers.

Instance	**A Domain Independent Heuristic**	**Our New Domain Dependent Heuristic**
$< 15, 4 >$	180	6
$< 17, 4 >$	320	10
$< 19, 4 >$	533	15
$< 20, 4 >$	1210	40

In Table 2, we present the average sum of distances between the selected containers and the right side of the layout in both our domain-independent heuristic and our domain-dependent heuristic with distance optimization for problems $< n, 4 >$. As mentioned above, we fixed the number of selected containers to 4 and we increased the number of containers n from 13 to 19. It can be observed that the heuristic for distance optimization helps finding solution plans that place the selected containers closer to the right side of the yard-bay.

5 Conclusions and Further Works

This paper presents the modelling of the container stacking problem form the Artificial Intelligence point of view. We have developed a domain-dependent planning tool for

Table 2: Average distance obtained by considering distance or not in our domain-dependent heuristic $< n, 4 >$ with 4 tiers.

Instance	**without Heuristic distance**	**with Heuristic distance**
$< 13, 4 >$	8.15	7.45
$< 15, 4 >$	10.05	8.90
$< 17, 4 >$	10.70	9.55
$< 19, 4 >$	10.85	8.40

finding an appropriate configuration of containers in a bay. Thus, given a set of outgoing containers, our planner minimizes the number of necessary reshuffles of containers in order to allocate all selected containers at the top of the stacks or under another selected containers in such a way that no further reshuffles are needed to load them.

The initial heuristic function proposed to guide the search has been also improved to allow the optimization of secondary interesting functions as the containers' distance to the right side of the yard-bay, in order to reduce the required time during the loading operations. We have also presented an expanded version of the domain which allow to organize all the containers in the yard-bay according to their departure time. And, as expected, our domain-dependent tool outperforms the existing domain-independent planners, allowing to obtain high quality solution plans in few milliseconds.

In further works, we will focus our attention in the development of a more complex domain-dependent planning heuristic to manage new hard and soft constraints.

References

[1] Y. Chen, C.W. Hsu, and B.W. Wah. SGPlan: Subgoal partitioning and resolution in planning. *IPC-4 Booklet (ICAPS)*, 2004.

[2] S. Edelkamp. Taming numbers and durations in the model checking integrated planning system. *Journal of Artificial Intelligence Research (JAIR)*, 20:195–238, 2003.

[3] A. Gerevini, A. Saetti, and I. Serina. Planning through stochastic local search and temporal action graphs in LPG. *Journal of Artificial Intelligence Research (JAIR)*, 20:239–290, 2003.

[4] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. *AIPS-98 Planning Committee*, 1998.

[5] J. Hoffman and B. Nebel. The FF planning system: Fast planning generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[6] K.W. Kim, Y.M. Park, and K.R. Ryu. Deriving decision rules to locate export containers in container yards. *European Journal of Operational Research*, 124:89–101, 2000.

[7] Yusin Lee and Nai-Yun Hsu. An optimization model for the container pre-marshalling problem. *Computers & Operations Research*, 34(11):3295 – 3313, 2007.

[8] J. Slaney and S. Thiébaux. Blocks world revisited. *Artificial Intelligence*, 125(1-2):119–153, 2001.

[9] S. Zilberstein and S.J. Russell. Optimal composition of real-time systems. *Artificial Intelligence*, 82(1-2):181–213, 1996.

Intelligent Planning for Allocating Containers in Maritime Terminals

M. Rodriguez-Molins, M. A. Salido, F. Barber
Instituto de Automática e Informática Industrial
Universidad Politécnica de Valencia.
Valencia, Spain

Abstract

Maritime container terminals are facilities where cargo containers are transshipped between ships or between ships and land vehicles (tucks or trains). These terminals involve a large number of complex and combinatorial problems. One of them is related to the Container Stacking Problem. A container yard is a type of temporary store where containers await further transport by truck, train or vessel. The main efficiency problem for an individual stack is to ensure easy access to containers at the expected time of transfer.

Stacks are 'last-in, first-out' storage structures where containers are stocked in the order they arrive. But they should be retrieved from the stack in the order (usually different) they should be shipped. This retrieval operation should be efficiently performed, since berthing time of vessels and the terminal operations should be optimized. To do this, cranes can relocate containers in the stacks to minimize the rearrangements required to meet the expected order of demand for containers.

In this paper, we present a domain-dependent heuristically guided planner for obtaining the optimized reshuffling plan, given a stacking state and a container demand. The planner can also be used for finding the best allocation of containers in a yard-bay in order to minimize the number of reshuffles as well as to be used for simulation tasks and obtaining conclusions about possible yard configurations.

Keywords Planning, Heuristics, Optimizing, Container Stacking Problem

1 Introduction

Maritime container terminals are the most important locations for transshipment and intermodal container transfers (Figure 1). [5] shows how this transshipment market is growing fast (container throughput has increased by 58 per cent over 2000-2004) and needs further studies to analyze it. In order to ensure reliability, e.g. delivery dates or handling times, to the different shipping companies as well as increasing productivity and container throughput from the quayside and landside and vice versa, there are several issues which need optimization. [18, 17] provide an extensive survey about operations at seaport container terminals and methods for their optimization. Moreover, other problems could be faced as for instance planning the routes for liner shipping services to obtain the maximal profit [2]. Another important issue for the success at any container terminal is to forecast container throughput accurately [1]. With this data they could develop better operational strategies and investment plans.

Containers are an ISO standardized metal box and can be stacked on top of each other. Loading and offloading containers on the stack is performed by cranes following a 'last-in, first-out' (LIFO) storage. In order to access a container which is not at the top of its pile, those above it must be relocated. It occurs since other ships have been unloaded later or containers have been stacked in the wrong order due to lack of accurate information. This reduces the productivity of the cranes. Maximizing the efficiency of this process leads to several requirements:

1. Each incoming container should be allocated a place in the stack which should be free and supported at the time of arrival.
2. Each outgoing container should be easily accessible, and preferably close to its unloading position, at the time of its departure.

In addition, there exist a set of hard/soft constraints regarding the container locations, for example, small differences in height of adjacent yard-bays, dangerous containers must be allocated separately by maintaining a minimum distance and so on.

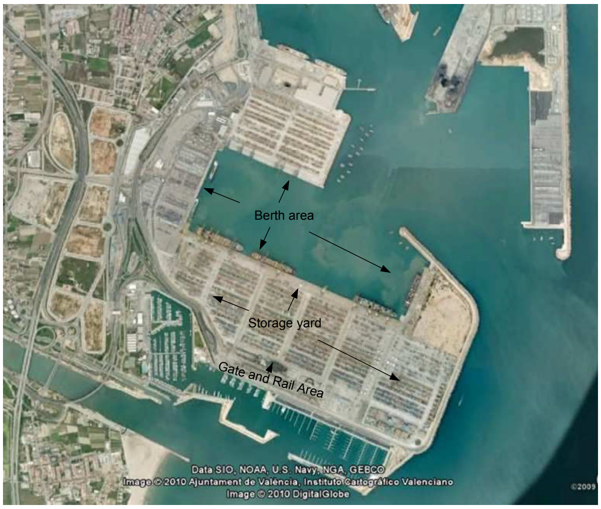

Berth area
Storage yard
Gate and Rail Area
Data SIO, NOAA, U.S. Navy, NGA, GEBCO
Image © 2010 Ajuntament de València, Instituto Cartográfico Valenciano
Image © 2010 DigitalGlobe
©2009

Figure 1: Container Terminal at Valencia

Nowadays, the allocation of positions to containers is usually done manually. Therefore, using appropriate Artificial Intelligent techniques is possible to achieve significant improvements of lead times, storage utilization and throughput.

Figure 2 left shows a container yard. A yard consists of several blocks, and each block consists of 20-30 yard-bays [11]. Each yard-bay contains several (usually 6) rows. Each row has a maximum allowed tier (usually tier 4 or tier 5 for full containers). Figure 2 right shows a gantry crane that is able to move a container within a stacking area or to another location on the terminal. For safety reasons, it is usually prohibited to move the gantry crane while carrying a container [12], therefore these movements only take place in the same yard-bay.

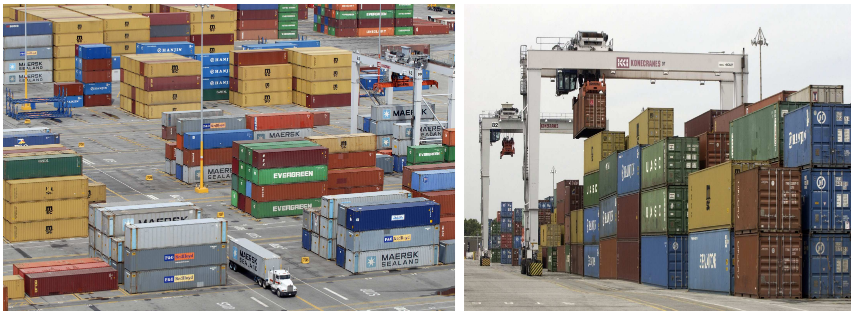


Figure 2: A container yard (left) and gantry cranes (right) (Photos by Stephen Berend)

When a container arrives at the terminal port, a transfer crane picks it up and stacks it in a yard-bay. During the ship loading operation, a transfer crane picks up the container and transfers it to a truck that delivers it to a quay crane.

In container terminals, the loading operation for export containers is pre-planned by load planners. For load planning, a containership agent usually transfers a load profile (an outline of a load plan) to a terminal operating company several days before a ship's arrival. The load profile specifies only the container group. In order to have an efficient load sequence, storage layout of export containers must have a good configuration.

The main focus of this paper is to present a planning system which optimally reallocates outgoing containers for the final storage layout from which a load planner can construct an efficient load sequence list. In this way, the objective is therefore to plan the movement of the cranes so as to minimize the number of reshuffles of containers in a complete yard. To this end, the yard is decomposed in yard-bays, so that the problem is distributed into a set of subproblems. Thus, each yard-bay generates a subproblem, but containers of different yard-bays must satisfy a set of constraints among them, so that subproblems will be sequentially solved taken into account the set of constraints with previously solved subproblems.

In the literature, generally this problem can be seen in two different ways according to when it should be done the optimization:

1. minimizing the number of relocations during the pickup operation.
2. getting a desirable layout for the bay before the pickup operation is done in order to minimize (or eliminate) the number of relocations during this process.

[8] proposes a methodology to estimate the expected number of rehandles to pick up an arbitrary container and the total number of rehandles to pick up all the containers in a bay for a given initial stacking configuration. In a similar way, [10] compares two methods, branch-and-bound algorithm and a heuristic rule based on an estimator, which they minimize the number of relocations during the pickup operation.

In [9], they also propose a methodology to convert the current bay layout into the desirable layout by moving the fewest possible number of containers (remarshalling) and in the shortest possible travel distance although it takes a considerable time since they use mathematical programming techniques. Cooperative coevolutionary algorithms have been developed in [13] to obtain a plan for remarshalling in automated container terminals.

This paper focuses on this latter issue. But we present a new heuristic with a set of optimization criteria in order to achieve efficiency and take into account constraints that should be considered in real-world problems in the provided solutions.

2 Problem description (The Container Stacking Problem)

The Container Stacking Problem can be viewed as a modification of the *Blocks World* planning domain [19], which is a well-known domain in the planning community. This domain consists of a finite number of blocks stacked into towers on a table large enough to hold them all. The *Blocks World* planning problem is to turn an initial state of the blocks into a goal state, by moving one block at a time from the top of a tower onto another tower (or on a table). The optimal *Blocks World* planning problem is to do so in a minimal number of moves.

Blocks World problem is closed to the Container Stacking Problem, but there are some important differences:

- The number of towers is limited to 6 because a yard-bay contains usually 6 rows.
- The height of a tower is also limited to 4 or 5 tiers depending on the employed cranes.
- There exist a set of constraints that involve different rows such as balanced adjacent rows, dangerous containers located in different rows, etc.
- The main difference is in the problem goal specification. In the *Blocks World* domain the goal is to get the blocks arranged in a certain layout, specifying the final position of each block. In the container stacking problem the goal state is not defined as accurately, so many different layouts can be a solution for a problem. The goal is that the most immediate containers to load are in the top of the towers, without indicating which containers must be in each tower.

We can model our problem by using the standard encoding language for classical planning tasks called *PDDL* (Planning Domain Definition Language) [3] whose purpose is to express the physical properties of the domain under consideration and it can be graphically represented by means of tools as [4]. A classical AI planning problem can be defined by a tuple $\langle A, I, G\rangle$, where A is a set of actions with preconditions and effects, I is the set of propositions in the initial state, and G is a set of propositions that hold true in any goal state. A solution plan to a problem in this form is a sequence of actions chosen from A that when applied transform the initial state I into a state of which G is a subset.

Following the *PDDL* standard, a planning task is defined by means of two text files. The **domain file**, which contains the common features for problems of this domain and the **problem file**, which describes the particular characteristics of each problem. These two files will be described in the following subsections.

2.1 Domain specification

In this file, we will specify the *objects* which may appear in the domain as well as the relations among them (*propositions*). Moreover, in order to make changes to the world state, *actions* must be defined.

- *Object types*: *containers* and *rows*, where the rows represent the areas in a yard-bay in which a tower or stack of containers can be built.
- *Types of propositions*:
  - Predicate for indicating that the container `?x` is on `?y`, which can be another container or, directly, the floor of a row (stack).

    `on ?x - container ?y - (either row container)`
  - Predicate for indicating that the container `?x` is in the tower built on the row `?r`.

    `at ?x - container ?r - row`
  - Predicate for stating that `?x`, which can be a row or a container, is clear, that is, there are no containers stacked on it.

    `clear ?x - (either row container)`
  - Predicate for indicating that the crane used to move the containers is not holding any container.

    `crane-empty`
  - Predicate for stating that te crane is holding the container `?x`.

    `holding ?x - container`
  - Predicates used to describe the problem goal. The first one specifies the most immediate containers to load, which must be located on the top of the towers to facilitate the ship loading operation. The second one becomes true when this goal is achieved for the given container.

    `goal-container ?x - container` and `ready ?x - container`

  - Numerical predicates. The first one stores the number of containers stacked on a given row and the second one counts the number of container movements carried out in the plan.

    `height ?s - row` and `num-moves`

- *Actions*:

  - The crane picks the container `?x` which is in the floor of row `?r`.

    `pick (?x - container ?r - row)`
  - The crane puts the container `?x`, which is holding, in the floor of row `?r`.

    `put (?x - container ?r - row)`
  - The crane unstacks the container `?x`, which is in row `?r`, from the container `?y`.

    `unstack (?x - container ?y - container ?r - row)`
  - The crane stacks the container `?x`, which is currently holding, on container `?y` in the row `?r`.

    `stack (?x - container ?y - container ?r - row)`
  - Finally, we have defined two additional actions that allow to check whether a given (goal) container is ready, that is, it is in a valid position. When a container is clear:

    `fict-check1 (?x - container)`

    The container is under another (goal) container which is in a valid position.

    `fict-check2 (?x - container ?y - container)`

As an example of *PDDL* format, we show in Figure 3 the specification of the stack operator. Preconditions describe the conditions that must hold to apply the action: crane must be holding container `?x`, container `?y` must be clear and at row `?r`, and the number of containers in that row must be less than 4. With this constraint we limit the height of the piles. The effects describe the changes in the world after the execution of the action: container `?x` becomes clear and stacked on `?y` at row `?r`, and the crane is not holding any container. Container `?y` becomes not clear and the number of movements and the containers in `?r` is increased in one unit.

2.2 Problem specification

Once the problem domain has been defined, we can define problem instances. These files describe the particular characteristics of each problem:

- *Objects*: the rows available in the yard-bay (usually 6) and the containers stored in them.

- *Initial state*: the initial layout of the containers in the yard.

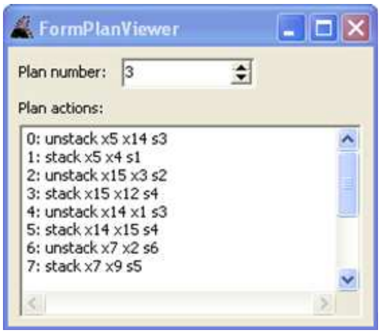
```
(:action stack
 :parameters (?x - container ?y - container ?r - row)
 :precondition (and
    (holding ?x) (clear ?y)
    (at ?y ?r)   (< (height ?r) 4))
 :effect (and
    (clear ?x)   (on ?x ?y)
    (at ?x ?r)   (crane-empty)
    (not (holding ?x))
    (not (ready ?y))
    (not (clear ?y))
    (increase (num-moves) 1)
    (increase (height ?r) 1)))
```

Figure 3: Formalization of the *stack* operator in *PDDL*.

- *The goal specification*: the selected containers to be allocated at the top of the stacks or under other selected containers.

- *The metric function*: the function to optimize. In our case, we want to minimize the number of relocation movements (reshuffles).

Since the Container Stacking Problem can be formalized with these two files, we can use a general domain independent planner to solve our problems as *Metric FF* [7]. The plan, which is returned by the planner, is a totally ordered sequence of actions or movements which must be carried out by the crane to achieve the objective. Figure 4 shows an example of the obtained plan for a given problem. The performance of this general planner will be analyzed in Section 6, which will be compared with the domain-oriented planner presented in next Sections.



```
3
0: unstack x5 x14 s3
1: stack x5 x4 s1
2: unstack x15 x3 s2
3: stack x15 x12 s4
4: unstack x14 x1 s3
5: stack x14 x15 s4
6: unstack x7 x2 s6
7: stack x7 x9 s5
```

Figure 4: The obtained plan solution to be carried out by the transfer crane.

3 A Domain-Dependent Heuristically Guided Planner

Metric FF planner might obtain plans, but it is very inefficient. Therefore, we propose a domain-dependent planner in order to provide more efficiency, it means at least reducing the number of crane operations required to achieve a desirable layout.

The proposed planner is built on the basis of a local search domain-independent planner called *Simplanner* [16]. This planner has several interesting properties for the container stacking problem:

- It is an anytime planning algorithm. This means that the planner can found a first, probably suboptimal, solution quite rapidly and that this solution is being improved while time is available.
- It is complete, so it will always find a solution if exists.
- It is optimal, so that it guarantees finding the optimal plan if there is time enough for computation.

It follows an enforced hill-climbing [6] approach with some modifications:

- It applies a best-first search strategy to escape from plateaux. This search is guided by a combination of two heuristic functions and it allows the planner to escape from a local minima very efficiently.
- If a plateau exit node is found within a search limit imposed, the hill-climbing search is resumed from the exit node. Otherwise, a new local search iteration is started from the best open node.

The initial approach, based on *Simplanner*, was firstly used to solve individual subproblems (yard-bays). To improve the solutions obtained by *Simplanner* we have further developed a domain-dependent heuristic to guide the search in order to accelerate and guide the search toward a optimal or sub-optimal solutions.

This heuristic (called h_1) was developed to efficiently solve one yard-bay. h_1 computes an estimator of the number of container movements that must be carried out to reach a goal state (see Algorithm 1). The essential part of this algorithm is to count the number of containers located on the selected ones, but also keeps track of the containers that are held by the crane distinguishing between whether they are selected containers or not. When the crane is holding a selected container, the value h has a smaller increase since, although this state is not a solution, this container will be at the top of some row in the next movement.

4 Optimization criteria for one-bay yards

Despite we are able to obtain good solutions (layouts) from *Simplanner* enhanced with h_1, we also want solutions more realistic for instance taking into account safety standards.

Algorithm 1: Pseudo-code of the domain-dependent heuristic h_1

```
   Data: $b$: state of the yard-bay;
   Result: $h$: heuristic value of $b$;
1  $h \leftarrow 0$;
   // Container hold by the crane
2  if $\exists x-$container / Holding($x$) $\in b$ then
3      if GoalContainer($x$) then
4          $h \leftarrow 0.1$;
5      else
6          $h \leftarrow 0.5$;
7      end
8  end
   // Increasing the $\Delta h$ value
9  for $r \leftarrow 1$ to numRows($b$) do
10     $\Delta h \leftarrow 0$;
11     for $x-$container / At($x, r$) $\wedge$ GoalContainer($x$) $\in b$ do
12         if $\nexists y-$container / GoalContainer($y$) $\wedge$ On($y, x$) $\in b$ then
13             $\Delta h \leftarrow \max(\Delta h,$ NumContainersOn($x$)$)$;
14         end
15     end
16     $h \leftarrow h + \Delta h$;
17 end
```

From this heuristic h_1, we have developed some optimization criteria each one of them achieving one of the requirements we could face at Container Terminals [15]. These criteria are centered in the next issues:

1. Reducing distance of the goal containers to the cargo side (OC_{1d}).
2. Increasing the range of the move actions set for the cranes allowing to move a container to 5th tier (OC_{1t}).
3. Applying different ways of balancing within the same bay in order to avoid *sinks* (OC_{1b}).

These criteria have been easily incorporated in our planner by defining a heuristic function as a linear combination of two functions:

$$h(s) = \alpha \cdot h_1(s) + \beta \cdot h_2(s) \tag{1}$$

being this secondary function a combination of these three criteria described:

$$h_2(s) = OC_{1d} + OC_{1t} + OC_{1b} \tag{2}$$

Note that although we want to guarantee balancing with this last optimization criterion, unbalanced states (states with *sinks*) are allowed during this process of remarshalling in order to get better solutions according to the number of reshuffles done.

4.1 OC_{1d}: Placing goal containers close to cargo side

Given an initial state, several different layouts can be usually achieved making the same number of reshuffles and some of them can be more interesting than the rest according to other important questions. In this case, since the transfer crane is located at the right side of the yard-bay, we want to obtain a layout where it is minimized the distance of the goal containers to this side of the yard-bay. Achieving this we can spend considerably less time during the truck loading operations.

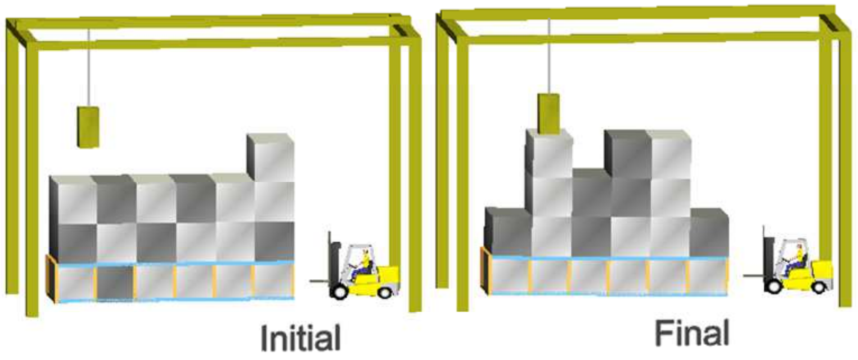


Figure 5: Obtained plan with the initial domain-dependent heuristic.

Following the heuristic function presented in Equation 1:

- $h_1(s)$ is the main heuristic function, which estimates the number of movements required to reach the goal layout (outlined in Algorithm 1). Since this is the main optimization function, α value should be significantly higher than β.
- $h_2(s)$ is the secondary function we want to optimize. In this case, it is just OC_{1d}. This means the sum of the distances of the selected containers to the right side of the yard-bay, which can be computed as Algorithm 2 shows.

Algorithm 2: Pseudo-code to calculate the distance

```
**Data**: $s$: state to evaluate
**Result**: $d$: distance value of $s$
1 $d \leftarrow 0$;
2 for $r \leftarrow 1$ to numRows($s$) do
3     for $x$−container / At($x, r$) $\in s \wedge$ GoalContainer($x$) do
4         $d \leftarrow d + (\texttt{numRows}(s) - r)$;
5     end
6 end
```

The benefits of using this combined heuristic function can be observed in Figure 5 and Figure 6. In the first one we want only to minimize the number of reshuffles, i.e.

$h(s) = h_1(s)$. In the second one, we also want to minimize the distance of the selected containers to the forklift truck, so we have set $h(s) = 9 * h_1(s) + h_2(s)$. As a result, none of the selected containers (the red ones) are placed in the most left rows, reducing the required time to load the truck.

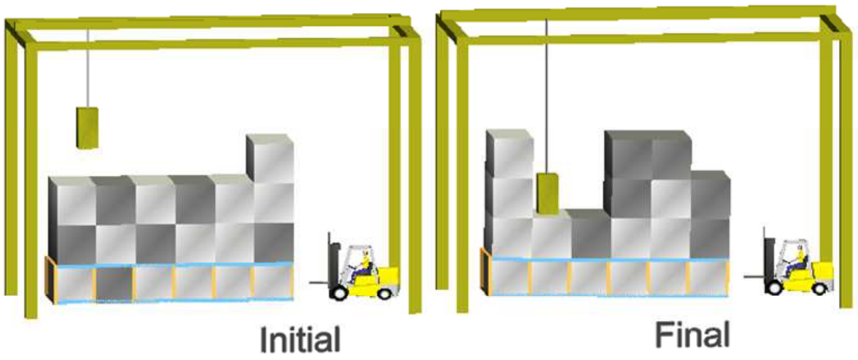


Figure 6: Obtained plan with the distance optimization function.

4.2 OC_{1t}: Allowing the 5th tier during the remarshalling process

In this optimization criterion as well as the next ones, we will include the new given heuristic value with the same factor as the initial one. One of the decisions that must be done in Container Terminals is about which cranes have to be bought depending on how many tiers cranes work. This topic has been considered in [14]. But, another approach is to reach the fifth tier only during the remarshalling process. Thereby, there would be 4 tiers at the beginning and the end keeping the first requirements.

Following this concept, we will use instances of problems $< n, 4 >$ with a domain whose move actions allow 5 tiers at the stacks. This function is showed in Algorithm 3 and it follows the same steps than the original but increasing the value of h when the height of one of the stacks is higher than 4. Thereby, we assure that the final layout will always have 4 tiers.

4.3 OC_{1b}: Balancing one yard-bay

In this section we present an extension for the heuristic h_1 (Algorithm 1) to include the balancing of the stacks within one yard-bay as a requirement. It is considered that there is a *sink* when the height difference between two adjacent stacks in the same yard-bay is greater than a maximum number of containers, in our case two containers.

Considering the time when the goal containers are removed from the yard, we can distinguish three ways to get balanced one yard-bay presented in the next subsections. The last mode is the consequence of applying the first two ones.

Algorithm 3: Pseudo-code of the domain-dependent heuristic function to allow 5 tiers

```
Data: $s$: state to evaluate
Result: $h$: heuristic value of $s$
h ← 0;
if $\exists x$−container / Holding($x$) $\in s$ then
    if GoalContainer($x$) then
        $h \leftarrow 0.1$;
    else
        $h \leftarrow 0.5$;
    end
end
for $r \leftarrow 1$ to numRows($s$) do
    $\Delta h \leftarrow 0$;
    if Height[$r, s$] $> 4$ then
        if $x$−container / Clear($x, r$) $\in s \wedge$ GoalContainer($x$) then
            $\Delta h \leftarrow 0.5$;
        else
            $\Delta h \leftarrow 1$;
        end
    end
    for $x$−container / At($x, r$) $\in s \wedge$ GoalContainer($x$) do
        if $\nexists y$−container / GoalContainer($y$) $\wedge$ On($y, x$) $\in s$ then
            $\Delta h \leftarrow \max(\Delta h,$ NumContainersOn($x$));
        end
    end
    $h \leftarrow h + \Delta h$;
end
```

1. **Balanced before loading operation** In this case, we consider that the *layout must be balanced before the goal containers are removed from that yard-bay*. This function is showed in Algorithm 4, it compares the height of each row of the yard-bay with the next one, and if the difference is higher than 2, the value heuristic h is increased. As it appears in Figure 7, this criterion avoids the *sinks* in the final layout while all the containers are still in the yard-bay.

   However, when these containers are removed, it might cause that the new layout is unbalanced as it happens in Figure 7(c).

Algorithm 4: Pseudo-code to balance before the goal containers are removed

```
Data: $s$: state to evaluate; $h$: Initial heuristic;
Result: $h$: heuristic value of $s$;
for $r \leftarrow 1$ to numRows($s$) $- 1$ do
    $\Delta h \leftarrow$ Abs(Height[$r, s$] $-$ Height[$r+1, s$]) $- 2$;
    if $\Delta h > 0$ then
        $h \leftarrow h + \Delta h$;
    end
end
```

2. **Balanced after loading operation** In contrast to the method seen above, we can consider that the *layout must remain balanced after the goal containers are removed from the yard-bay*. Figure 8 shows the layouts we get after execute the plan returned

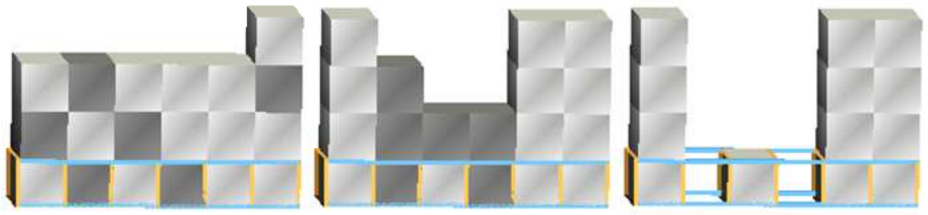

(a) Initial Layout (b) With goal containers (c) Without goal containers

Figure 7: Effects of using function seen in Algorithm 4

by our planner.

Algorithm 5: Function `HeightsWithoutGoals` to calculate heights of each row without taking into account the goal containers at the top

```
   Data: $b$: state of the *yard-bay*;
   Result: MinHeight, heights calculated;
1  for $r \leftarrow 1$ to numRows($b$) do
2      MinHeight[$r, b$] $\leftarrow$ Height[$r, b$];
       // Decrease till the first no goal-container
3      while MinHeight[$r, b$] $> 0 \wedge$ GoalContainer(MinHeight[$r, b$], $r$) $\in b$ do
4          MinHeight[$r, b$] $\leftarrow$ MinHeight[$r, b$] $- 1$;
5      end
6  end
```

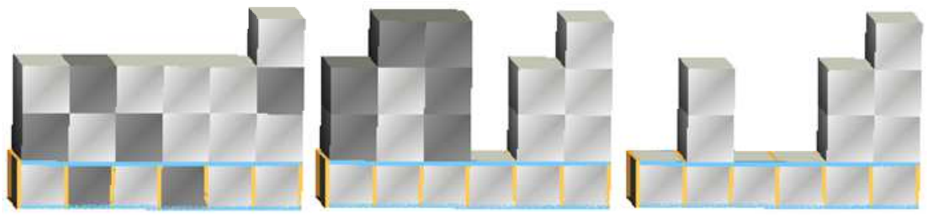
Algorithm 6 shows this function. It uses the Function *HeightsWithoutGoals* (Algorithm 5) in order to calculate for the yard-bay b the height for each stack where the first no-goal container is. These values are employed to get the difference of height between two adjacent stacks once the goal containers have been removed from the yard. Heights of each row are stored as soon as the planner gets the final solution plan for one yard-bay. After we obtain these values, we increase the heuristic value h according to whether or not there are goal containers on the floor. Then, we use the values given by *HeightsWithoutGoals* to calculate the difference between two adjacent stacks, when this difference is higher than 2 we consider that there is a *sink*, so h is increased again.

However, this process might also cause some unbalanced layouts (Figure 8(b)). But in this case, non-desirable layouts will appear while the goal containers are in the yard-bay. Once they have been removed from it, these layouts will be balanced ones (Figure 8(c)).

3. **Balanced before and after loading operation** Finally, we present an optimization criterion which obtains a *layout where is balanced both before and after the goal containers are removed from this yard-bay*. With this function we want to solve the problems seen in the last subsections as we can see it in Figure 9.

Algorithm 6: Pseudo-code to balance after the goal containers are removed

```
Data: $s$: state to evaluate; $h$: Initial heuristic;
Result: $h$: heuristic value of $s$;
HeightsWithoutGoals($s$);
$\Delta h \leftarrow 0$;
// Not allow containers on the floor
for $r \leftarrow 1$ to numRows($s$) do
    if $\exists x-$container / On($x, r$) $\wedge$ GoalContainer($x$) then
        if MinHeight[$r, s$] $> 0$ then
            $\Delta h \leftarrow \Delta h +$ NumContainersOn($x$);
        end
    end
end
$h \leftarrow h + \Delta h$;
for $r \leftarrow 1$ to numRows($s$) $- 1$ do
    $\Delta h \leftarrow$ Abs(MinHeight[$r, s$] $-$ MinHeight[$r+1, s$]);
    if $\Delta h > 2$ then
        $h \leftarrow h + \Delta h - 2$;
    end
end
```



(a) Initial Layout (b) With goal containers (c) Without goal containers

Figure 8: Effects of using function seen in Algorithm 6

This function (Algorithm 7) is a mixture of the last two ones. First, we increase h when there are goal containers on the floor. When this is achieved, we increase h when the difference between the heights values obtained by the function *HeightsWithoutGoals* (Algorithm 5) are higher than 2 for two contiguous rows. And finally, if h value is low enough (in our case lower than 1), we increase h again if the difference between the actual heights of two contiguous rows is higher than 2.

5 Optimization criteria for one block

This initial heuristic (h_1) was unable to solve a complete yard or block (in our case, one block consists of 20 yard-bays) due to the fact that they only solve individual yard-bays. In this paper, we also have developed two optimization criteria that include new constraints that involve several yard-bays. These constraints are:

Algorithm 7: Pseudo-code to balance the yard-bay before and after the goal containers are removed

```
Data: $s$: state to evaluate; $h$: Initial heuristic;
Result: $h$: heuristic value of $s$;
HeightsWithoutGoals($s$);
$\Delta h \leftarrow 0$;
// Not allow containers on the floor
for $r \leftarrow 1$ to numRows($s$) do
    if $\exists x-$container / On($x, r$) $\wedge$ GoalContainer($x$) then
        if MinHeight[$r, s$] $> 0$ then
            $\Delta h \leftarrow \Delta h +$ NumContainersOn($x$);
        end
    end
end
$h \leftarrow h + \Delta h$;
if $h < 2$ then
    $\Delta h \leftarrow 0$;
    // Balancing with containers which are not objective
    for $r \leftarrow 1$ to numRows($s$) $- 1$ do
        $\Delta h \leftarrow$ Abs(MinHeight[$r, s$] $-$ MinHeight[$r+1, s$]);
        if $\Delta h > 2$ then
            $h \leftarrow h + 0.6 \times (\Delta h - 2)$;
        end
    end
    if $h < 2$ then
        // Balancing with containers which are objective
        for $r \leftarrow 1$ to numRows($s$) $- 1$ do
            $\Delta h \leftarrow$ Abs(Height[$r, s$] $-$ Height[$r+1, s$]);
            if $\Delta h > 2$ then
                $h \leftarrow h + 0.4 \times (\Delta h - 2)$;
            end
        end
    end
end
```

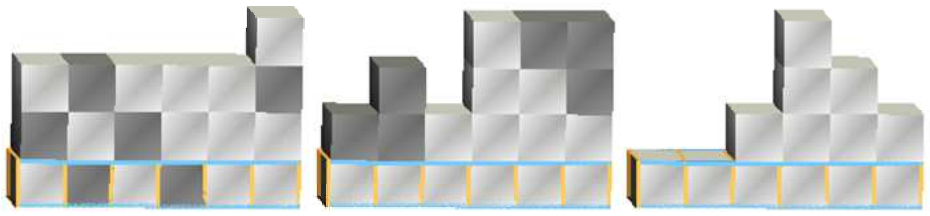

(a) Initial Layout (b) With goal containers (c) Without goal containers

Figure 9: Effects of using function seen in Algorithm 7

- Balancing contiguous yard-bays: rows of adjacent yard-bays must be balanced, that is, the difference between the number of containers of row j in yard-bay i and row j in yard-bay $i-1$ must be lower than a maximum (in our case lower than 3). Figure 10 shows which rows must be get balanced when we consider one yard-bay and Figure 11 left shows an example of non-balanced yard-bays (rows in dotted points).

- Dangerous containers: two dangerous containers must maintain a minimum security distance. Figure 11 right shows an example of two dangerous containers that does not satisfy the security distance constraint.

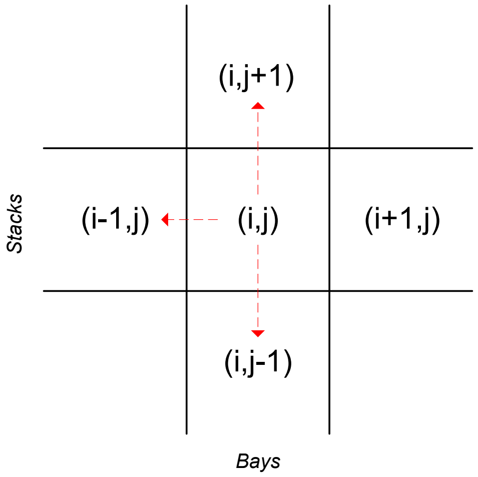


Figure 10: Balancing scheme

These constraints interrelate the yard-bays so the problem must be solved as a complete problem. However, it is a combinatorial problem and it is not possible to find an optimal or sub-optimal solution in a reasonable time. Following the previous philosophy of solving each subproblem independently (each yard-bay separately), we can distribute the problem into subproblems and solve them sequentially taken into account related yard-bays. Thus a solution to the first yard-bay is taken into account to solve the second yard-bay. A solution to the second yard-bay is taken into account to solve the third yard-bay. Furthermore, if there exist a dangerous container in a first bay, its location is taken into account to solve a dangerous container located in the third yard-bay (if it exists); and so on. Taken into account this distributed and synchronous model, we present two different optimization criteria to manage these types of constraints.

These two criteria are added to the heuristic function seen in Equation 1 as h_3 (Equation 3); and Equation 4 shows the exact combination of them. This makes possible to follow a criterion with major priority than the other one.

$$h = \alpha \cdot h_1 + \beta \cdot h_2 + \gamma \cdot h_3 \tag{3}$$

$$h_3 = \delta_1 \cdot OC_{nB} + \delta_2 \cdot OC_{nD} \tag{4}$$

As a consequence of the solving mode followed, depending on the order the yard-bays are resolved may not be possible to achieve a solution. Moreover, as mentioned in

Section 4, although we want to guarantee balancing and/or minimum distance between dangerous containers, during relocation of container process we will allow the presence of non-desirable sates, e.g. with some *sinks* between two contiguous rows or bays. These intermediate states are allowed because through them we will be able to get better solutions taking into account as metric function the number of reshuffles done.

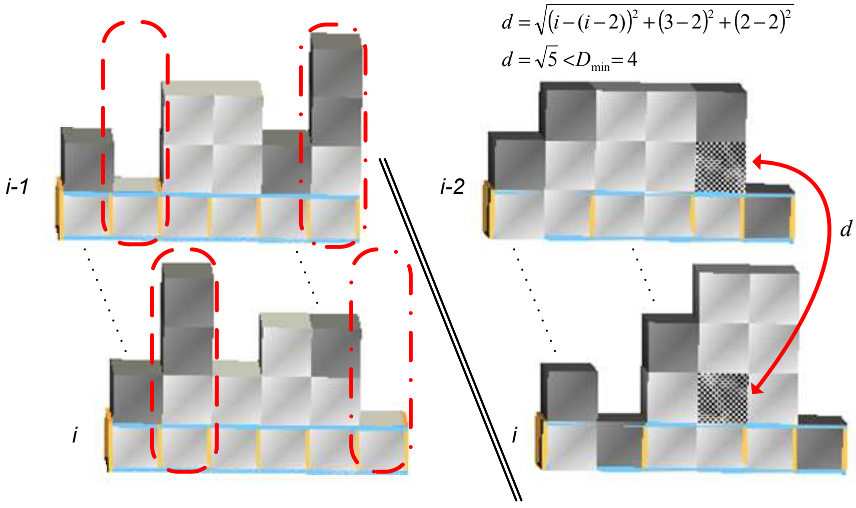


Figure 11: (Left) Non-balanced yard-bays. (Right) Proximity of two dangerous containers.

5.1 OC_{nB}: Balancing contiguous yard-bays

In this section we present an extension for the heuristic h_1 (Algorithm 1) to include the balancing of continuous yard-bays as a requirement. It is considered that there is a *sink* when a difference higher than two containers exists between two adjacent rows in contiguous yard-bays. This criterion is an extension of the *balanced heuristic* presented in Algorithm 7, which avoids *sinks* in the same yard-bay (horizontal balance) both before and after the outbound containers have been removed from the yard. However, in this case a *sink* represents a constraint between two subproblems. Thus, we also consider that there is a *sink* when a difference of two exits between the same row r in two contiguous yard-bays (vertical balance).

This process is showed in Algorithm 8. This also uses the Function *HeightsWithoutGoals* (Algorithm 5) in order to calculate for the yard-bay b the height for each stack where the first no-goal container is. Heights of each row are stored as soon as the planner gets the final solution plan for one yard-bay.

First, we apply the criterion seen in Algorithm 7 on the yard-bay b. Through `heights'` calculated by Algorithm 5 and the real heights of the actual yard-bay we obtain the differences between the row r and $r-1$ to calculate the value of h. When this value is zero (the yard-bay b is horizontally balanced), then we introduce our function to balance it with respect to the last yard-bay b_l. To do so, we must also calculate the `heights'` through the Algorithm 5 over b_l and use the real heights of it in order to obtain the differences between the row r situated in b and b_l. When these differences are higher than 2, we increase h proportionally. After that process, b will be balanced horizontally with respect to their rows, and vertically with respect to the last yard-bay. Repeating this process for each yard-bay in the block, this will be completely balanced.

Algorithm 8: Pseudo-code to balance two adjacent *yard-bays*

```
Data: $b$: state of the actual yard-bay; $h$: Initial heuristic; $b_l$: last yard-bay;
Result: $h$: heuristic value of $b$
// Getting the balance horizontally
HeightsWithoutGoals($b$);
$h \leftarrow h +$ BalBeforeAfter($b$);
// This heuristic will be executed after a partial solution
if $h < 1 \wedge$ NumBay($b$) $\neq 1$ then
    $\Delta h \leftarrow 0$;
    HeightsWithoutGoals($b_l$);
    // Balancing with containers which are not objective
    for $r \leftarrow 1$ to numRows($b$) do
        $\Delta h \leftarrow$ Abs(MinHeight$[r, b_l]$ $-$ MinHeight$[r, b]$);
        if $\Delta h > 2$ then
            $h \leftarrow h + 0.6 \times (\Delta h - 2)$;
        end
    end
    if $h = 0$ then
        // Balancing with containers which are objective
        for $r \leftarrow 1$ to numRows($b$) do
            $\Delta h \leftarrow$ Abs(Height$[r, b_l]$ $-$ Height$[r, b]$);
            if $\Delta h > 2$ then
                $h \leftarrow h + 0.4 \times (\Delta h - 2)$;
            end
        end
    end
end
```

5.2 OC_{nD}: Dangerous containers

Within a block, there are different types of containers depending on the goods they transport, being some of them dangerous. If they do not satisfy certain restrictions, it may become a hazard situation for the yard since e.g. if one of them explodes and they are not enough far between them, it will set off a chain of explosions.

With this added objective, the next optimization criterion (Algorithm 9) ensures a minimum distance (D_{min}) between every two dangerous containers (C_d) in the yard. D_{min} is set as one parameter for the planner and the distance is calculated as the Euclidean distance, considering each container located in a 3-dimensional space (X,Y,Z) where X is the number of yard-bays, Y is the number of rows and Z is the tier.

Generally, in container terminals, at most, there is only one dangerous container in two contiguous yard-bays, so that we take into account this assumption in the development of this function.

This function increases h value when a dangerous container C_{d1} exists in a yard-bay b and the distance constraints between dangerous containers are not hold. Thereby, for each dangerous container C_{d2} allocated in the previous D_{min} yard-bays is calculated by Euclidean distance to C_{d1}. If this distance is lower than D_{min}, for any dangerous container C_{d2}, then h value is increased with the number of containers n on C_{d1} because it indicates that removing those n containers is necessary to reallocate the container C_{d1}.

Algorithm 9: Pseudo-code to avoid locating two dangerous containers closer to a distance D_{min}

```
Data: $B$: whole block; $b$: state of the actual yard-bay; $h$: Initial heuristic; $D_{min}$: Minimum distance;
Result: $h$: heuristic value of $b$;
1  $nBay \leftarrow$ NumBay($b$);
2  if $nBay > 1 \wedge \exists C_{d1} \in b$ then
3      $\Delta h \leftarrow 0$;
4      $L_1 \leftarrow$ Location($C_{d1}$);
5      foreach $b_l \in Y$ / NumBay($b_l$) $\in \{\max(nBay - D_{min} + 1, 1), nBay - 1\}$ do
6          if $\exists C_{d2} \in b_l$ then
7              $L_2 \leftarrow$ Location($C_{d2}$);
8              $dist \leftarrow$ EuclideanDistance($L_1, L_2$);
9              if $dist < D_{min}$ then
10                 $\Delta h \leftarrow \Delta h +$ NumContainersOn($C_{d1}$);
11                 if Clear($C_{d1}$) $\in b$ then
12                     $\Delta h \leftarrow \Delta h + (D_{min} - dist)$;
13                 end
14             end
15         end
16     end
17     $h \leftarrow h + \Delta h$;
18 end
```

Algorithm 10: Sinks within a whole block

```
Data: $B$: whole block;
Result: $nSinks$: number of Sinks;
1  $nSinks \leftarrow 0$;
2  for $b \leftarrow 1$ to numYards($B$) do
3      for $r \leftarrow 1$ to numRows($b$) $- 1$ do
4          $\Delta h \leftarrow$ Abs(Height[$r, b$] $-$ Height[$r+1, b$]);
5          if $\Delta h > 2$ then
6              $nSinks \leftarrow nSinks + 1$;
7          end
8      end
9      if NumBay($b$) $> 1$ then
10         for $r \leftarrow 1$ to numRows($b$) do
11             $\Delta h \leftarrow$ Abs(Height[$r, b$] $-$ Height[$r, b-1$]);
12             if $\Delta h > 2$ then
13                 $nSinks \leftarrow nSinks + 1$;
14             end
15         end
16     end
17 end
```

Algorithm 11: Unfeasible relationships between two dangerous containers within a whole block

```
Data: $B$: whole block;
Result: $nDang$: number of Sinks;
$nDang \leftarrow 0$;
for $b \leftarrow 1$ to numYards($B$) do
    $nBay \leftarrow$ NumBay($b$);
    if $nBay > 1 \wedge \exists C_{d1} \in b$ then
        $L_1 \leftarrow$ Location($C_{d1}$);
        foreach $b_l \in Y$ / NumBay($b_l$) $\in \{\max(nBay - D_{min} + 1, 1), nBay - 1\}$ do
            if $\exists C_{d2} \in b_l$ then
                $L_2 \leftarrow$ Location($C_{d2}$);
                $dist \leftarrow$ EuclideanDistance($L_1, L_2$);
                if $dist < D_{min}$ then
                    $nDang \leftarrow nDang + 1$;
                end
            end
        end
    end
end
```

6 Evaluation

In this section, we evaluate the behavior of the heuristic with the set of optimization criteria presented in this paper. The experiments were performed on random instances. A random instance of a yard-bay is characterized by the tuple $< n, s >$, where n is the number of containers in a yard-bay and s is the number of selected containers in the yard-bay. Each instance is a random configuration of all containers distributed along six stacks with 4 tiers. They are solved on a personal computer equipped with a Core 2 Quad Q9950 2.84Ghz with 3.25Gb RAM.

First, we present a comparison between our basic domain dependent heuristic h_1 against a domain independent one (*Metric FF*). Thus, Table 1 presents the average running time (in milliseconds) to achieve a first solution as well as the best solution found (number of reshuffles) in 10 seconds for our domain-dependent planner and the average running time (in milliseconds) and the quality of the solution for *Metric FF*. Both planners have been tested in problems $< n, 4 >$ evaluating 100 test cases for each one. Thus, we fixed the number of selected containers to 4 and we increased the number of containers n from 15 to 21.

It can be observed that our new domain-dependent heuristic is able to find a solution in a few milliseconds, meanwhile the domain-independent planner (*Metric FF*) needs much time for finding a solution and also, this solution needs more moves to get a goal state. Furthermore, due to the fact that our tool is an anytime planner, we evaluate the best solution found in a given time (10 seconds).

Now we show the effects of using each one of the criteria described in Section 4 separately. In Table 2, we present the average sum of distances between the selected containers and the right side of the layout in both our domain-independent heuristic and our domain-dependent heuristic with distance optimization for problems $< n, 4 >$. As

Table 1: Average number of reshuffles and running time of *Metric FF* and h_1 in problems $< n, 4 >$.

Instance	***Metric FF***		**Heuristic (h_1)**	
	Running time	Solution	Time first solution	Best Solution in 10 secs
$< 13, 4 >$	22	3.07	2	3.07
$< 15, 4 >$	3102	4.04	5	3.65
$< 17, 4 >$	4669	5.35	11	4.35
$< 19, 4 >$	6504	6.06	22	4.72
$< 20, 4 >$	22622	7.01	33	5.22
$< 21, 4 >$	13981	6.82	62	5.08

mentioned above, we fixed the number of selected containers to 4 and we increased the number of containers n from 13 to 21. It can be observed that distance optimization function helps finding solution plans that place the selected containers closer to the cargo side of the yard-bay.

Table 2: Average distance obtained by considering distance or not in our domain-dependent heuristic $< n, 4 >$ with 4 tiers.

Instance	***Metric FF***		***OC_{1d}***	
	Distance	*Reshuffles*	*Distance*	*Reshuffles*
$< 13, 4 >$	11.28	3.07	10.91	3.07
$< 15, 4 >$	10.60	4.04	9.21	3.65
$< 17, 4 >$	10.58	5.35	8.87	4.46
$< 19, 4 >$	12.28	6.06	8.33	4.85
$< 20, 4 >$	12.71	7.01	7.75	5.55
$< 21, 4 >$	12.20	6.82	8.22	5.33

Applying the criterion or function showed in Algorithm 3 we obtain the results appeared in Table 3. These results are the comparison between the number of solved problems over 100 problems $< n, 4 >$ using or not that criterion in just one second. Through this table we can conclude that:

- The greater number of containers, the fewer problems are solved. This is because as we increase the number of containers there are less positions or gaps where containers could be remarshalled.
- Allowing movements to the 5^{th} helps us to solve more problems. It is remarkable with instances $< 23, 4 >$ with H_1 only three problems could be solved, however OC_{1t} solves 84 over 100 problems.

Table 3: Number of solved problems $< n, 4 >$ with 4 and 5 tiers during the process.

Instance	**4 tiers h_1**	**5 tiers OC_{1t}**
$< 19, 4 >$	100	100
$< 20, 4 >$	100	100
$< 21, 4 >$	95	99
$< 23, 4 >$	3	84

Last criterion for solving problems where we only take into account one yard-bay is showed in Section 4.3. As we mentioned in this section, since the last function (Algorithm 7) presents the best results after the whole process of remarshalling, we do the comparison in Table 4 among the solutions given by *Metric FF* planner, the initial one h_1 and OC_{1b} (*Both*) in 50 test cases. These results are the average of the best solutions found given a time limit of 1 second for the instances of both $< 15, 4 >$ and $< 17, 4 >$.

Sinks are calculated by Algorithm 10. As we mentioned above, we consider that there is a sink where the difference in tiers between two adjacent rows is higher than 2. Thereby, in this algorithm we are counting sinks produced between two contiguous stacks at the same yard-bay as well as between two rows in one yard-bay and the previous one. This process takes into account the goal containers in final yard-bays.

Table 4: Average number of movements, sinks and time for the first solution in problems $< 15, 4 >$ (1) and $< 17, 4 >$ (2) using or not balanced heuristics.

	Metric FF		**h_1**		**OC_{1b}**	
	(1)	(2)	(1)	(2)	(1)	(2)
Reshuffles	3.72	4.24	3.42	3.72	4.76	5.04
Sinks	0.62	0.50	0.94	0.66	0	0
Time First	2621	2961	5	9	32	44

From here we realize an evaluation for the criteria presented in Section 5. Table 5 shows the performance of the criteria for solving the whole block of yard-bays. These experiments were performed in blocks of 20 yard-bays and each one of them are instances $< 15, 4 >$. This evaluation was carried out in a yard with 2 blocks of 20 yard-bays. Thus, the results showed in Table 5 represent the average number of reshuffles, the average number of sinks generated along the block and the average number of unsatisfied dangerous containers. Results given by these optimization criteria are the average of the best solutions found in 10 seconds.

The number of unfeasible relationships between dangerous containers is calculated by means of Algorithm 11. Basically, we look for those pairs of dangerous containers whose distance between them is shorter than minimum distance (D_{min}).

In this table, it can be observed that h_1 still outperforms *Metric FF* in the average number of reshuffles. However, due to the fact that they do not take into account the balancing constraints, *Metric FF* generated an average of 18.00 sinks in the block of yard-bay and h_1 generated and average of 29.50 sinks. And the same thing happens for the average number of unfeasible constraints for dangerous containers, *Metric FF* gives us 16.00 and h_1 obtains 7.50.

Taking into account that OC_N is a junction of OC_{nB} and OC_{nD}, both OC_{nB} and OC_{nD} solved their problems, that is, OC_{nB} obtained its solutions with no sinks and OC_{nD} obtained its solutions by satisfying all dangerous constraints. Furthermore, OC_N was able to solve its problems by satisfying both types of constraints. However we could state that balancing problem is harder than the problem related to dangerous containers because OC_{nB} needs more reshuffles to obtain a solution plan than OC_{nD}. Moreover, we observe with OC_{nB}, OC_{nD} and OC_N ensure the established requirements however the average reshuffles is increased with respect to h_1.

Table 5: Average results with blocks of 20 *yard-bays* each one being a $< 15, 4 >$ problem.

	Metric FF	$\boldsymbol{h_1}$	$\boldsymbol{OC_{nB}}$	$\boldsymbol{OC_{nD}}$	$\boldsymbol{OC_N}$
Reshuffles	3.65	3.38	4.85	4.00	5.65
Sinks	18.00	29.50	0	40.33	0
Non-Safe Dangerous	16.00	7.50	8.00	0	0

7 Conclusions

This paper presents domain-dependent heuristics and a set of optimization criteria for solving the Container Stacking problem by means of planning techniques from Artificial Intelligence. We have developed a domain-dependent planning tool for finding optimized plans to obtain an appropriate configuration of containers in a yard-bay. Thus, given a set of outgoing containers, our planner minimizes the number of necessary reshuffles of containers in order to allocate all selected containers at the top of the stacks. The proposed planner is able to satisfy both balancing constraints and keeping a security distance among dangerous containers, as well as reducing the distance of the goal containers to the cargo side or allowing a fifth tier during the remarshalling process.

Additional criteria have been defined for management of blocks of yard-bays. However, as the problems involve a larger number of constraints, the solution becomes harder and the number of reshuffles increases. Due to the fact that a solution of a yard-bay influences on the solution of the following yard-bay, the order of solving the yard-bays will vary and determine the minimal number of reshuffles.

This proposed planner with a domain-dependent heuristic allows us obtaining optimized and efficient solutions. This automatic planner can help to take decisions in the port operations dealing with real problems. Moreover, it can help to simulate operations to obtain conclusions about the operation of the terminal, evaluate alternative configurations, obtain performance measures, etc. Particularly, in [14] the proposed planner has been applied for obtaining an evaluation of alternative 4 or 5 tiers stacks configuration.

Acknowledgment

This work has been partially supported by the research projects TIN2010-20976-C02-01 (Min. de Ciencia e Innovación, Spain), P19/08 (Min. de Fomento, Spain-FEDER) and the VALi+d Program of the Conselleria d'Educació (Generalitat Valenciana), as well as with the collaboration of the maritime container terminal MSC (Mediterranean Shipping Company S.A.).

References

[1] Shih-Huang Chen and Jun-Nan Chen. Forecasting container throughputs at ports using genetic programming. *Expert Systems with Applications*, 37(3):2054 – 2058, 2010.

[2] Tzung-Nan Chuang, Chia-Tzu Lin, Jung-Yuan Kung, and Ming-Da Lin. Planning the route of container ships: A fuzzy genetic approach. *Expert Systems with Applications*, 37(4):2948 – 2956, 2010.

[3] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. *AIPS-98 Planning Committee*, 1998.

[4] Ourania Hatzi, Dimitris Vrakas, Nick Bassiliades, Dimosthenis Anagnostopoulos, and Ioannis Vlahavas. A visual programming system for automated problem solving. *Expert Systems with Applications*, 37(6):4611 – 4625, 2010.

[5] L. Henesey. Overview of Transshipment Operations and Simulation. In *MedTrade conference, Malta, April*, pages 6–7, 2006.

[6] J. Hoffman and B. Nebel. The FF planning system: Fast planning generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[7] Jörg Hoffmann. The metric-ff planning system: translating ”ignoring delete lists” to numeric state variables. *J. Artif. Int. Res.*, 20(1):291–341, 2003.

[8] Kap Hwan Kim. Evaluation of the number of rehandles in container yards. *Computers & Industrial Engineering*, 32(4):701 – 711, 1997. New Advances in Analysis of Manufacturing Systems.

[9] K.H. Kim and J.W Bae. Re-marshaling export containers in port container terminals. *Computers & Industrial Engineering*, 35(3-4):655 – 658, 1998. Selected Papers from the 22nd ICC and IE Conference.

[10] K.H. Kim and G.P. Hong. A heuristic rule for relocating blocks. *Computers & Operations Research*, 33(4):940–954, 2006.

[11] K.W. Kim, Y.M. Park, and K.R. Ryu. Deriving decision rules to locate export containers in container yards. *European Journal of Operational Research*, 124:89–101, 2000.

[12] Yusin Lee and Nai-Yun Hsu. An optimization model for the container pre-marshalling problem. *Computers & Operations Research*, 34(11):3295 – 3313, 2007.

[13] K. Park, T. Park, and K.R. Ryu. Planning for remarshaling in an automated container terminal using cooperative coevolutionary algorithms. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1098–1105. ACM, 2009.

[14] M. Salido, O. Sapena, and F. Barber. What is better: 4 tiers or 5 tiers in the container stacking problem? *The International Workshop on Harbour, Maritime and Multimodal Logistics Modelling and Simulation*, 2009.

[15] M. Salido, O. Sapena, M. Rodriguez, and F. Barber. A planning tool for minimizing reshuffles in containers terminals. *ICTAI 2009: 21st International Conference on Tools with Artificial Intelligence*, 2009.

[16] O. Sapena and E. Onaindía. Domain independent on-line planning for strips domains. *In proc. IBERAMIA-02, 2527*, pages 825–834, 2002.

[17] R. Stahlbock and S. Voß. Operations research at container terminals: a literature update. *OR Spectrum*, 30(1):1–52, 2008.

[18] I.F.A. Vis and R. De Koster. Transshipment of containers at a container terminal: an overview. *European Journal of Operational Research*, 147:1–16, 2003.

[19] Terry Winograd. *Procedures as a representation for data in a computer program for understanding natural language*. MIT. Cent. Space Res., Cambridge, MA, 1971.

2.2 Scheduling: Berth Allocation Problem and Quay Crane Assignment problem

A genetic algorithm for berth allocation and quay crane assignment

Mario Rodriguez-Molins, Federico Barber, María R. Sierra, Jorge Puente, Miguel A. Salido
Instituto de Automática e Informática Indsutrial, Universidad Politécnica de Valencia (Spain)

Department of Computer Science, University of Oviedo (Spain)

Abstract

Container terminals are facilities where cargo containers are transshipped between different transport vehicles, for onward transportation. They are open systems that carry out a large number of different combinatorial problems that can be solved by means of Artificial Intelligence techniques. In this work, we focus our attention on scheduling a number of incoming vessels by assigning to each a berthing position, a mooring time and a number of Quay Cranes. This problem is known as the Berthing Allocation and Quay Crane Assignment problem. To formulate the problem, we first propose a mixed integer linear programming model to minimize the total weighted service time of the incoming vessels. Then, a meta-heuristic algorithm (Genetic Algorithm (GA)) is presented for solving the proposed problem. Computational experiments are performed to evaluate the effectiveness and efficiency of the proposed method.

Keywords Scheduling, Planning, Genetic Algorithms, Metaheuristics, Berthing Allocation, Quay Crane Assignment

1 Introduction

A container terminal is an open system with three distinguishible areas (berth, container yard and lanside areas) where there exist different complex optimization problems. For instance, berthing allocation or stowage planning problems are related to the berth area [13]; remarshalling problem or transport optimization in the yard area; and, planning and scheduling hinterland operations related to trains and trucks in the landside area [14].

Two planning and scheduling problems are studied in this paper, the Berth Allocation Problem (BAP) and the Quay Crane Assignment Problem (QCAP). The former is a well-known combinatorial optimization problem [10], which consists in assigning incoming vessels to berthing positions. The QCAP deals with assigning a certain number of QCs to each vessel that is waiting at the roadstead such that all required movements of containers can be fulfilled [1]. Once a vessel arrives at the port, it waits at the roadstead until it has permission to moor at the quay. The locations where mooring can take place are called berths. These are equipped with giant cranes, known as Quay Cranes (QC), that are used to load and unload containers which are transferred to and from the yard by a fleet of vehicles. These QCs are mounted on the same track (or rail) and, therefore they cannot pass each other. In a transshipment terminal, the yard allows temporary storage before containers are transferred to another ship or to another transportation mode (e.g., rail or road).

A comprehensive survey of BAP and QCAP is given by [1]. These problems have been mostly considered separately and with an interest mainly focused on BAP. However, there are some studies on the combined BAP+QCAP considering different characteristics of the berths and cranes ([2], [7], [9], [12], [15]). In this paper, we present a formal mixed integer lineal programming for the combined BAP+QCAP that extends the model presented in [8], by managing a continuous quay line. In order to obtain optimized solutions in an efficient way, we develop a metaheuristic GA, so that compared with mathematical solvers obtains near-optimal solutions in competitive computational times.

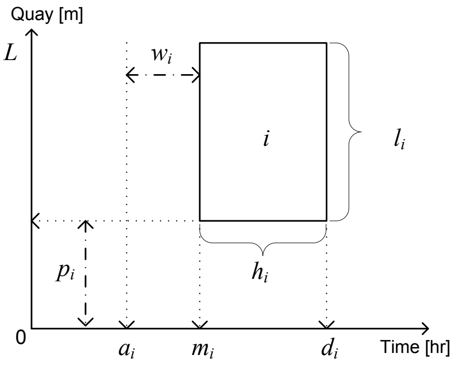
The rest of the paper is organized as follows. In the next two sections we give a thorough description and a mathematical formulation of the problem. In Section 4 we give the details of the GA designed for the BAP+QCAP. Section 5 reports the results of the experimental study. Finally, in Section 6 we give the main conclusions of this work.

2 Problem description

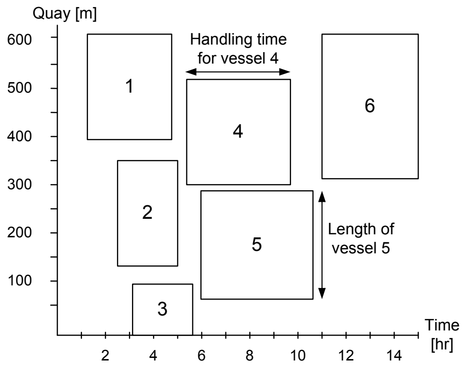
The objective in BAP+QCAP is to obtain an schedule of the incoming vessels with an optimum order of vessels mooring and a distribution of the docks and QCs for these vessels. Figure 1(b) shows an example of the graphical space-time representation of a berth plan with 6 vessels. Each rectangle represents a vessel with its handling time and length.

Our BAP+QCAP case is classified according to the classification given by [1] as:

- *Spatial attribute: Continuous layout.* We assume that the quay is a continuous line, so there is no partitioning of the quay and the vessel can berth at arbitrary positions within the boundaries of the quay. It must be taken into account that for a continuous layout, berth planning is more complicated than for a discrete layout, but it better utilizes the quay space [1].
- *Temporal attribute: Dynamic arrival.* Fixed arrival times are given for the vessels, so that vessels cannot berth before their expected arrival times.
- *Handling time attribute: Unknown in advance.* The handling time of a vessel depends on the number of assigned QCs (*QCAP*) and the moves required.
- *Performance measure: wait and handling times* The objective is to minimize the sum of the waiting (w_i) and handling times (h_i) of all vessels.



(a) Vessel



(b) Schedule

Figure 1: Representation of the BAP+QCAP problem

Let V be the set of incoming vessels. Following, we introduce the notation used for each vessel $i \in V$ (Figure 1(a)). The data variables are:

- QC : Available QCs in the container terminal. All QCs carry out the same number of movements per time unit $(movsQC)$, given by the container terminal.
- L : Total length of the berth in the container terminal.
- a_i : Arrival time of the vessel i at port.
- c_i : Number of required movements to load and unload containers of i.
- l_i : Vessel length.
- pr_i : Vessel priority.

The decision variables are:

- m_i : Mooring time of i. Thus, waiting time (w_i) of i is calculated as $(w_i = m_i - a_i)$.
- p_i : Berthing position where i moors.
- q_i : Number of assigned QCs to i.
- u_{ik} : Indicates whether the QC k works (1) or not (0) on the vessel i.

The variables derived from the previous ones are:

- h_i : Loading and unloading time at quay (handling time) of vessel i. This time depends on q_i and c_i, that is : $\left(\frac{c_i}{q_i \times \texttt{movsQC}}\right)$.
- t_{ik} : Working time of the QC k that is assigned to vessel i.

- d_i : Departure time of vessel i ($d_i = m_i + h_i$).
- s_i, e_i : indexes for the first and last QC used in vessel i, respectively.

Our objective is to allocate all vessels according to several constraints minimizing the total weighted waiting and service time for all vessels:

$$T_s := \sum_{i \in V} (w_i + h_i) \times pr_i \tag{1}$$

Note that this problem is a very special case of a multi-mode resource-constrained scheduling problem, where there exist shared resources (berth length), the duration of activities (mooring time) depends on the assigned resources (QCs), and the objective function is minimizing both the waiting as the processing times of vessels.

Moreover, the following assumptions are considered:

- Number of QCs assigned to a vessel do not vary along the moored time. Once a QC starts a task in a vessel, it must complete it without any pause or shift (non-preemptive tasks). Thus, all QCs assigned to the same vessel have the same working time ($t_{ik} = h_i, \forall k \in QC, u_{ik} = 1$)
- All the information related to the waiting vessels is known in advance (arrival, priority, moves and length).
- Every vessel has a draft that is lower than or equal to the draft of the quay.
- Movements of QCs along the quay as well as berthing and departure times of vessels are not considered since it supose a constant penalty time for all vessels.
- The components of the optimization function (Equation 1) can be independently weighted without requiring changes to our proposal.
- Simultaneous berthing is allowed, subject to the length of the berth.

And the following constraints must be accomplished:

- Moored time must be at least the same that its arrival time ($m_i \geq a_i$).
- It must be enough contiguous space at berth to moor a vessel of length (l_i).
- There is a safety distance (`safeDist`) between two moored ships. We assume 5% of the maximum length of two contiguous vessels.
- There must be at least one QC to assign to each vessel. The maximum number of assigned QCs by vessel depends on its length, since a safety distance is required between two contiguous QCs (`safeQC`), and the maximum number of QCs that the container terminal allows per vessel (`maxQC`). Both parameters are given by the container terminal.

3 Mathematical formulation

In this section, the mathematical formulation for BAP+QCAP is presented. The given MILP model solves the BAP+QCAP by minimizing the function given by the Equation 1, where M denotes a sufficiently large number, subject to the given constraints:

$$m_i \geq a_i \quad \forall i \in V \tag{2}$$
$$w_i = m_i - a_i \quad \forall i \in V \tag{3}$$
$$p_i + l_i \leq L \quad \forall i \in V \tag{4}$$
$$q_i = \sum_{k \in QC} u_{ik} \quad \forall i \in V \tag{5}$$
$$1 \leq q_i \leq QC_i^+ \quad \forall i \in V \tag{6}$$
$$1 \leq s_i, e_i \leq |QC| \quad \forall i \in V \tag{7}$$
$$s_i \geq e_i \quad \forall i \in V \tag{8}$$
$$q_i = e_i - s_i + 1 \quad \forall i \in V \tag{9}$$
$$\sum_{k \in QC} t_{ik} \times \texttt{movsQC} \geq c_i \quad \forall i \in V \tag{10}$$
$$h_i = \max_{k \in QC} t_{ik} \quad \forall i \in V \tag{11}$$
$$t_{ik} - u_{ik} \times M \leq 0 \quad \forall i \in V, \forall k \in QC \tag{12}$$
$$h_i - M \times (1 - u_{ik}) - t_{ik} \leq 0 \quad \forall i \in V, \forall k \in QC \tag{13}$$
$$u_{ik} + u_{jk} + z_{ij}^x \leq 2 \quad \forall i, j \in V, \forall k \in QC \tag{14}$$
$$M \times (1 - u_{ik}) + (e_i - k) \geq 0 \quad \forall i \in V, \forall k \in QC \tag{15}$$
$$M \times (1 - u_{ik}) + (k - s_i) \geq 0 \quad \forall i \in V, \forall k \in QC \tag{16}$$
$$p_i + l_i \leq p_j - sd_{ij} + M \times (1 - z_{ij}^x) \quad \forall i, j \in V,\ i \neq j \tag{17}$$
$$e_i + 1 \leq s_j + M \times (1 - z_{ij}^x) \quad \forall i, j \in V,\ i \neq j \tag{18}$$
$$m_i + h_i \leq m_j + M \times (1 - z_{ij}^y) \quad \forall i, j \in V,\ i \neq j \tag{19}$$
$$z_{ij}^x + z_{ji}^x + z_{ij}^y + z_{ji}^y \geq 1 \quad \forall i, j \in V,\ i \neq j \tag{20}$$
$$z_{ij}^x, z_{ij}^y, u_{ik} \quad \text{0/1 integer} \quad \forall i, j \in V,\ i \neq j, \forall k \in QC \tag{21}$$

The given formulation expands the model presented in [8] by adding the needed constraints to take into consideration QCs. Thereby, the handling time of vessels depends on the number of QCs and these QCs cannot pass each other when are relocated.

In the proposed model, there are two auxiliary variables: z_{ij}^x is a decision variable that indicates if vessel i is located to the left of vessel j on the berth ($z_{ij}^x = 1$); and, $z_{ij}^y = 1$ indicates that vessel i is moored before vessel j in time (see constraint 21). Moreover, Constraint 2 ensures that vessels must moor once they arrive at the terminal. Constraint 4 guarantees that a moored vessel does not exceed the length quay. Constraints 5, 6, 7, 8 and 9 assign the number of QCs to the vessel i. Constraint 10 establishes the needed handling time to load and unload their containers. Constraint 12 ensures that QCs that are not assigned QCs to i have t_{ik} zero. Constraint 13 forces all assigned QCs to i working the same number of hours. Constraint 11 assigns the handling time for vessel i. Constraint 14 avoids that one QC is assigned to two different vessels at the same time. Constraints 15 and 16 force the QCs to be assigned contiguously (from s_i up to e_i). Constraint 17 takes into account the safety distance between each two vessels. Constraint 18 avoids that one vessel uses a QC which should cross through the others QCs. Constraint 19 avoids that vessel j moors while the previous vessel i is still at the quay. Finally, constraint 20 establishes the relationship between each pair of vessels.

This mathematical model has been coded in IBM ILOG CPLEX Optimization Studio 12.3 as detailed in the Evaluation Section 5.

4 Genetic Algorithm

Algorithm 1 shows the structure of the GA we have considered herein. The core of this algorithm is taken from [5, 4] and is quite similar to others generational genetic algorithms described in the literature ([6], [3] or [11]). In the first step, the initial population is generated and evaluated. Then, the genetic algorithm iterates over a number of steps or generations. In each iteration, a new generation is built from the previous one by applying the genetic operators of selection, reproduction and replacement. These operators can be implemented in a variety of ways and, in principle, are independent from each other. However, in practice all of them should be chosen considering their effect on the remaining ones in order to get a successful overall algorithm. The approach taken in this work is the following. In the selection phase all chromosomes are grouped into pairs, and then each one of these pairs is mated or not in accordance with a crossover probability (P_c) to obtain two offspring. Each offspring, or parent if the parents were not mated, undergoes mutation in accordance with the mutation probability (P_m). Finally, the replacement is carried out as a tournament selection (4:2) among each pair of parents and their offspring.

The coding schema is based on permutations of vessels, each one with a given number of QCs. So a gene is a pair (i, q_i), $1 \leq q_i \leq \min(maxQC_i, maxQC)$, and a chromosome includes a gene like this for each one of the vessels. For example, for an instance with 5 vessels where the maximum number of QCs are 2, 3, 4, 3 and 2 respectively, two feasible chromosomes are the following ones:

$$c_1\text{: ((1 1) (2 1) (3 1) (4 2) (5 1))}$$

$$c_2\text{: ((3 2) (1 2) (2 2) (5 2) (4 3))}$$

Note that, the same vessel may have different number of QCs in each chromosome. In accordance with this encoding, a chromosome expresses the number of QCs that each vessel is assigned in the solution and an order for building the schedule.

The order of vessels in chromosomes is used as a dispatching rule. Hence, we use the following decoding algorithm: the genes are visited from left to right in the chromosome

Algorithm 1: The genetic algorithm

```
Require: A BAP-QCAP instance P
Ensure: A mooring schedule for instance P
  1. Generate the initial population;
  2. Evaluate the population;
  while No termination criterion is satisfied do
    3. Select chromosomes from the current population;
    4. Apply the reproduction operators to the chromosomes selected at step 3. to
    generate new ones;
    5. Evaluate the chromosomes generated at step 4;
    6. Apply the replacement criterion to the set of chromosomes selected at step 3.
    together with the chromosomes generated at step 4.;
  end while
  return The schedule from the best chromosome evaluated so far;
```

sequence. For each gene (i, q_i) the vessel i is scheduled at the earliest mooring time with q_i consecutive QCs available, so that none of the constraints is violated. If there are several positions available at the earliest time, that closest to one of the berth extremes is selected. Also, the QCs are chosen starting from the same extreme of the berth.

For chromosome mating we have considered a classical crossover operator such as Generalized Position Crossover (GPX) which is commonly used in permutation based encodings. This is a two points crossover operator which work as follows. Let us consider two parents like:

p_1: ((1 1) | (2 1) (3 1) | (4 2) (5 1))

p_2: ((3 2) | (1 2) (2 2) | (5 2) (4 3))

Symbols "|" represent crossover positions, 1 and 3 respectively in this example, which are selected at random for each mating. Then two offsprings are built taking the substrings between positions 1 and 3 in each parent and then filling the remaining positions with the genes representing the remaining vessels taken from the other parent keeping their relative order. So in this case the two offsprings are:

o_1: ((1 2) | (2 1) (3 1) | (5 2) (4 3))

o_2: ((3 1) | (1 2) (2 2) | (4 2) (5 1))

For mutation we have implemented an operator that shuffles a random substring of the chromosome and at the same time changes the number of QCs assigned to each one of the shuffled genes at random, provided that the number of QCs is kept in between the proper limits for the vessel.

The initial population in generated at random, i.e. a random order for the vessels is chosen and each vessel i is assigned a number of QCs chosen uniformly in $[1, \min(maxQC_i, maxQC)]$.

The termination condition is given in one of these three forms: (1) a number of generations, (2) a time limit or (3) a number of evaluations.

5 Evaluation

The experiments were performed in a corpus of 100 instances generated randomly, each one is composed of a queue from 5 to 20 vessels. These instances follow an exponential distribution for the inter-arrival times of the vessels ($\lambda = \frac{1}{20}$). The number of required movements and length of vessels are generated uniformly in $[100, 1000]$ and $[100, 500]$ respectively. In all cases, the berth length (L) is fixed to 700 meters; the number of QCs is 7 (corresponding to a determined MSC berth line) and the maximum number of QCs per vessel is 5 (`maxQC`); the safety distance between QCs (`safeQC`) is 35 meters and the number of movements that QCs carry out is 2.5 (`movsQC`) per time unit.

The two approaches developed in this paper, the GA and the MILP model, were coded using C++ and the IBM ILOG CPLEX Optimization Studio 12.3, respectively. They were solved on a Linux PC 2.26Ghz.

In the GA, the population size is 200. Mutation and crossover probabilities are $P_m = 0.1$ and $P_c = 0.8$, respectively. Due to the stochastic nature of the GA process, each one of the instances were solved 30 times and the results show the average obtained values.

Table 1 shows the results form CPLEX and GA averaged for each group of 100 instances with the same number of vessels (5 to 20). The timeout was 10 seconds. For CPLEX, the reported values are the average value of T_s for the solutions reached, the number of instances solved to optimality (#Opt), the number of instances solved without certify optimality (#NOpt) and the number of instances for which no solution is reached by the timeout (#NSol) The last two columns show the best and the average values of the solutions obtained by the GA in 30 runs. Obviously, in all cases, the objective function (T_s) increases as the number of incoming vessels increases from 5 up to 20.

From these results, we can observe that CPLEX is not able to reach any optimal solution by the given timeout in at least 30% of the instances with 7 vessels or more. In addition, it can not get any optimal solution from 15 up to 20 vessels with this timeout. Moreover, for a number of instances with more than 14 vessels CPLEX is not able to reach a feasible solution. Regarding GA, all instances are solved and we can observe that the average values are better than those from CPLEX, the differences being in direct ratio with the number of vessels. Here, it is important to remark that GA reaches 1063 generations in 10 seconds. However, the GA is able to converge in lower times. Figure 2 shows the GA convergence for one representative instance of 20 vessels, so that near-optimal values are obtained after 100 generations, taking 0.94 seconds. Furthermore, Figure 3 shows how the average T_s for 10 vessels decreases as more computation time is allowed. In this experiment, the timeout was set to 5, 10, 20, and 60 seconds. As it can be observed, the GA approach does not require a large timeout (the improvement is lower than 1% beyond 5 seconds).

We remark that we have not been able to use previous test cases proposed in the

Table 1: Comparision CPLEX with GA (timeout 10 secs)

|V|	**CPLEX**				**GA**	
	Avg T_s	**#Opt**	**#NOpt**	**#NSol**	**Best T_s**	**Avg T_s**
5	1723.75	98	2	0	1723.75	1723.75
6	2193.06	88	12	0	2189.63	2189.63
7	2702.46	66	34	0	2681.14	2681.67
8	3287.66	41	59	0	3219.80	3222.13
9	3891.09	24	76	0	3729.78	3734.72
10	4642.23	14	86	0	4337.10	4350.23
11	5453.31	6	94	0	4946.66	4971.86
12	6557.60	3	97	0	5552.09	5589.16
13	7944.50	2	98	0	6181.67	6236.60
14	9332.26	1	98	1	6854.33	6931.59
15	11578.40	0	98	2	7526.27	7631.98
16	13518.00	0	97	3	8290.95	8438.06
17	15105.80	0	94	6	8972.13	9163.65
18	17253.80	0	85	15	9694.16	9927.06
19	18390.40	0	65	35	10506.20	10787.79
20	20410.50	0	46	54	11395.52	11725.64

literature because we assume a continuous berthing and non-preemtitive tasks ([15], [9]). However, even considering this more complex case, we can see that the results achieved are highly competitive against these previous approaches.

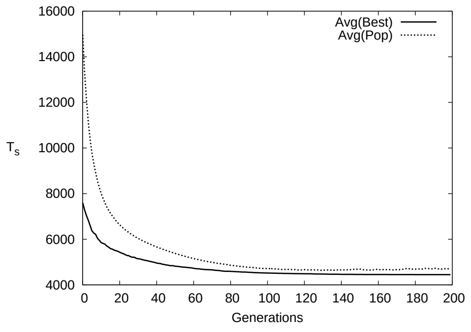


Figure 2: Convergence of the Genetic Algorithm

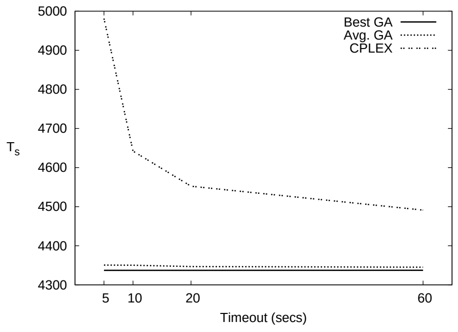


Figure 3: Average T_s for 10 vessels setting different timeouts

6 Conclusions

The competitiveness among container terminals causes the need to improve the efficiency of each one of the subprocesses that are performed within them. This paper focuses on two of the main related problems, the Berth Allocation and Quay Crane Assignment Problems, in an integrated way. To this end, a mixed integer lineal programming model and a Genetic Algorithm were developed. The MILP model was unable to get optimal solutions when a reasonable timeout is set or when the problem becomes harder (more than 10 vessels). Moreover, many of the instances were solved but without any guarantees of being the optimal ones since the timeout was reached. However, the GA approach is able to obtain near-optimal solutions in lower computational times and it also maintains a rapid convergence of the results even with large vessel queues. From these results, it is concluded the adequacy of a metaheuristic approach based on GA for solving the BAP+QCAP problem. This approach also extends the previous approaches given in the literature by adding features (as continuous quay line and non-preemptitive QC assignments) and it gives near-optimal solutions in a very competitive computational time. For future research, we propose devising some local search strategy that can be then combined with the GA or other metaheuristics such as GRASP, Tabu or Scatter Search.

Acknowledgments.

This research has been supported by the Spanish Government under research project MICINN TIN2010-20976-C02-01 and TIN2010-20976-C02-02, and the predoctoral FPU fellowship (AP2010-4405).

References

[1] C. Bierwirth and F. Meisel. A survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research*, 202:615–627, 2010.

[2] G. Giallombardo, L. Moccia, M. Salani, and I. Vacca. Modeling and solving the tactical berth allocation problem. *Transportation Research Part B: Methodological*, 44(2):232 – 245, 2010.

[3] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1985.

[4] I. Gonzalez-Rodriguez, C.R. Vela, and J. Puente. A memetic approach to fuzzy job shop based on expectation model. In *Proceedings of IEEE International Conference on Fuzzy Systems, FUZZ-IEEE2007*, pages 692–697. IEEE, 2007.

[5] I. González-Rodríguez, C.R. Vela, and J. Puente. A genetic solution based on lexicographical goal programming for a multiobjective job shop with uncertainty. *Journal of Intelligent Manufacturing*, 21(1):65–73, 2010.

[6] J.H. Holand. Adaptation in natural and artifcial systems: An introductory analysis with applications to biology, control and artificial intelligence, 1975.

[7] A. Imai, H.C. Chen, E. Nishimura, and S. Papadimitriou. The simultaneous berth and quay crane allocation problem. *Transportation Research Part E: Logistics and Transportation Review*, 44(5):900–920, 2008.

[8] K.H. Kim and K.C. Moon. Berth scheduling by simulated annealing. *Transportation Research Part B: Methodological*, 37(6):541–560, 2003.

[9] C. Liang, J. Guo, and Y. Yang. Multi-objective hybrid genetic algorithm for quay crane dynamic assignment in berth allocation planning. *Journal of Intelligent Manufacturing*, 22:471–479, 2011.

[10] A. Lim. The berth planning problem. *Operations Research Letters*, 22(2-3):105–110, 1998.

[11] Z. Michalewicz. *Genetic algorithms+ data structures = Evolution Programs, third, revised and extended.* Springer, 1996.

[12] Y.M. Park and K.H. Kim. A scheduling method for berth and quay cranes. *OR Spectrum*, 25(1):1–23, 2003.

[13] Miguel A. Salido, Mario Rodriguez-Molins, and Federico Barber. Integrated intelligent techniques for remarshaling and berthing in maritime terminals. *Advanced Engineering Informatics*, 25(3):435 – 451, 2011.

[14] R. Stahlbock and S. Voß. Operations research at container terminals: a literature update. *OR Spectrum*, 30(1):1–52, 2008.

[15] C. Zhang, L. Zheng, Z. Zhang, L. Shi, and A.J. Armstrong. The allocation of berths and quay cranes by using a sub-gradient optimization technique. *Computers & Industrial Engineering*, 58(1):40–50, 2010.

A GRASP-based Metaheuristic for the Berth Allocation Problem and the Quay Crane Assignment Problem by Managing Vessel Cargo Holds

Mario Rodriguez-Molins, Miguel A. Salido, Federico Barber
Instituto de Automática e Informática Industrial,
Universitat Politècnica de València,
Camino de vera, s/n, 46022, Valencia, Spain

Abstract

Container terminals are open systems that generally serve as a transshipment zone between vessels and land vehicles. These terminals carry out a large number of planning and scheduling tasks. In this paper, we consider the problem of scheduling a number of incoming vessels by assigning a berthing position, a berthing time, and a number of Quay Cranes to each vessel. This problem is known as the Berth Allocation Problem and the Quay Crane Assignment Problem. Holds of vessels are also managed in order to obtain a more realistic approach. Our aim is to minimize the total waiting time elapsed to serve all these vessels. In this paper, we deal with the above problems and propose an innovative metaheuristic approach. The results are compared against other allocation methods.

Keywords Planning, Scheduling, Optimization methods, Algorithms, Metaheuristic, GRASP

1 Introduction

Container terminals generally serve as a transshipment zone between ships and land vehicles (trains or trucks). In [12], it is shown how this transshipment market has grown rapidly. Between 1990 and 2008, container traffic grew from 28.7 million TEU (Twenty-foot Equivalent Unit) to 152.0 million TEU, an increase of about 430%. This corresponds to an average annual compound growth of 9.5%. In the same period, container throughput went from 88 million TEU to 530 million TEU, an increase of 500%, which is equivalent to an average annual compound growth of 10.5%. The surge of both container traffic and throughput is linked with the growth of international trade as well as to the adoption of containerization as the privileged vector for maritime shipping and inland transportation

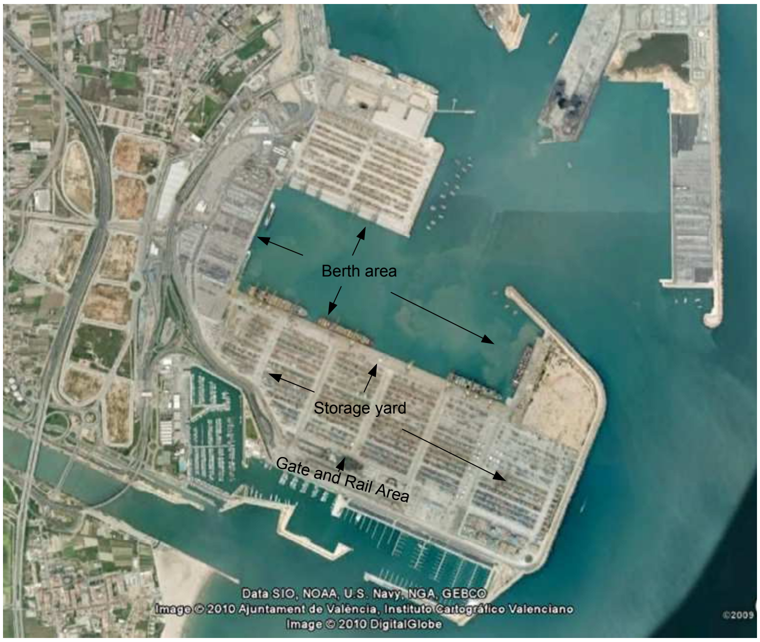

Berth area
Storage yard
Gate and Rail Area
Data SIO, NOAA, U.S. Navy, NGA, GEBCO
Image © 2010 Ajuntament de València, Institut Cartogràfico Valenciano
Image © 2010 DigitalGlobe
©2009

Figure 1: Container Terminal in Valencia.

[5]. However, the Global Economic Crisis of 2008 has had a negative impact over the container traffic [25].

The evolution of the container traffic has motivated the artificial intelligence research community to develop optimization techniques to manage combinatorial problems related to seaport terminals efficiently. These problems occur in transportation [34, 4] as well as within the container terminals. Container terminals are open systems where terminal operators must face up to combinatorial optimization problems to ensure the fast loading and unloading of the vessels. In container terminals, there are three distinguishable areas (see Figure 1): the berth area, where vessels are berthed for service; the storage yard, where containers are temporarily stored to be exported or imported; and the terminal receipt and delivery gate area, which connects the container terminal to the hinterland. Each one presents different planning and scheduling problems to be optimized [33, 15, 32]. For example, berth allocation, quay crane assignment, stowage planning, and quay crane scheduling must be managed in the berthing area; the container stacking problem, yard crane scheduling, and horizontal transport operations must be carried out in the yard area; and the hinterland operations must be solved in the land side area. Figure 2 shows the main planning and scheduling problems that must be managed in a container terminal [37].

We focus our attention on the Berth Allocation Problem (BAP) and the Quay Crane Assignment Problem (QCAP) taking into account the holds of each vessel. The former is a well-known, NP-hard combinatorial optimization problem [22], which consists of assigning incoming vessels to berthing positions. The latter deals with assigning a certain number of QCs to each vessel such that all required movements of containers can be fulfilled.

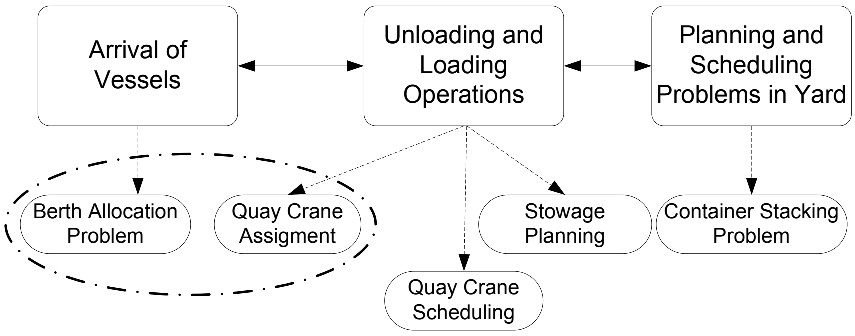


Figure 2: Planning and scheduling problems in container terminals.

Containers to be loaded/unloaded in the vessel are stored on the deck as well as in the holds. The holds of the vessels are structures that speed up both loading and unloading and keep the containers secure while at sea. Once a vessel arrives at the port, it waits at the roadstead until it has permission to moor at the quay. The quay is a platform that protrudes into the water to facilitate the loading and unloading of cargo. The locations where mooring can take place are called berths. These are equipped with giant cranes, called pier or Quay Cranes (QCs), that are used to load and unload containers, which are transferred to and from the yard by a fleet of vehicles. In a transshipment terminal, the yard allows temporary storage before containers are transferred to another ship or to another transportation mode (e.g., rail or road).

Managers at container terminals are confronted with two interrelated decisions: *where* and *when* the vessels should moor. First, they have to take into account physical restrictions such as length or draft, and they also have to take into account priorities and other aspects to minimize both port and user costs, which are usually opposites. Generally, this process is solved manually. It is usually solved by means of a policy to serve the first vessel that arrives (FCFS). Figure 3 shows an example of the graphical space-time representation of a berth plan with 6 vessels. Each rectangle represents a vessel with its handling time and length. For instance, vessel 4 must moor after vessels 1 and 2 depart.

The overall collaboration goal of our group at the Universitat Politècnica de València (UPV), the Valencia Port Foundation, and the maritime container terminal MSC (Mediterranean Shipping Company S.A.) is to offer assistance to help in planning and scheduling tasks such as to allocate spaces to outbound containers [29, 28], to berth incoming vessels [31], to identify bottlenecks, to determine the consequences of changes [30], to provide support in the resolution of incidences, to provide alternative berthing plans, etc. Researchers commonly develop mathematical or statistical models that represent real-world

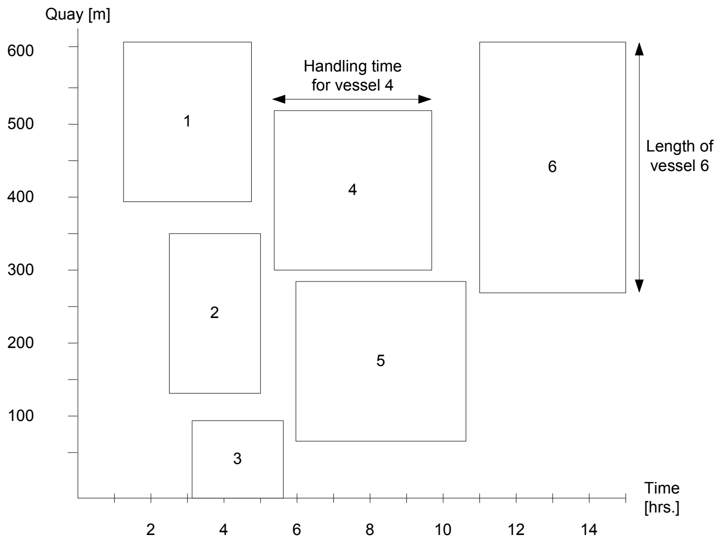


Figure 3: A berth plan with 6 vessels.

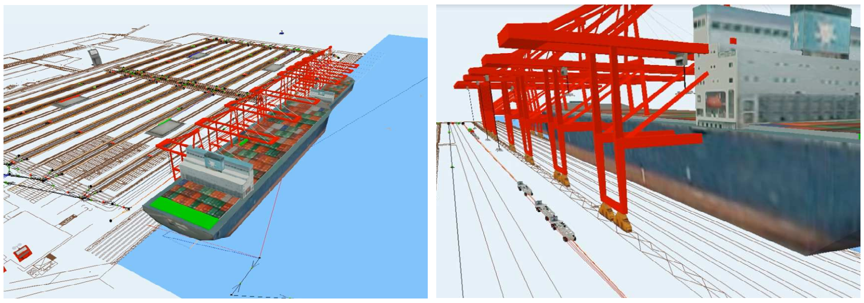


Figure 4: Simulator developed within the Masport Project [36].

systems. Nevertheless, these systems are very complex and composed of different problems that sometimes have opposing goals. These problems must be simplified with several assumptions in order to be appropriately modeled. Mathematical models are necessary but exact optimization methods algorithms cannot obtain the optimal solution in a reasonable time. Thus, techniques from artificial intelligence field (such as local search or metaheuristics) must be applied in order to solve these combinatorial optimization problems efficiently and get near-optimal solutions in an efficient way [1]. Our artificial intelligence techniques are included within a simulator that is able to represent a given state of the terminal and simulate the behavior in different parts of the terminal (see Figure 4). In this paper, a metaheuristic has been developed to schedule the incoming vessels and has been compared with several real world rules, commonly used by expert terminal operators.

The remainder of this paper is organized as follows: Section 2 presents a review of the literature about the BAP and QCAP and different techniques to manage them. Section 3 presents the problem definition. Section 4 explains the whole process of mooring one vessel and Section 5 describes the developed metaheuristic technique. Section 6 presents the computational results, and Section 7 summarizes our conclusions.

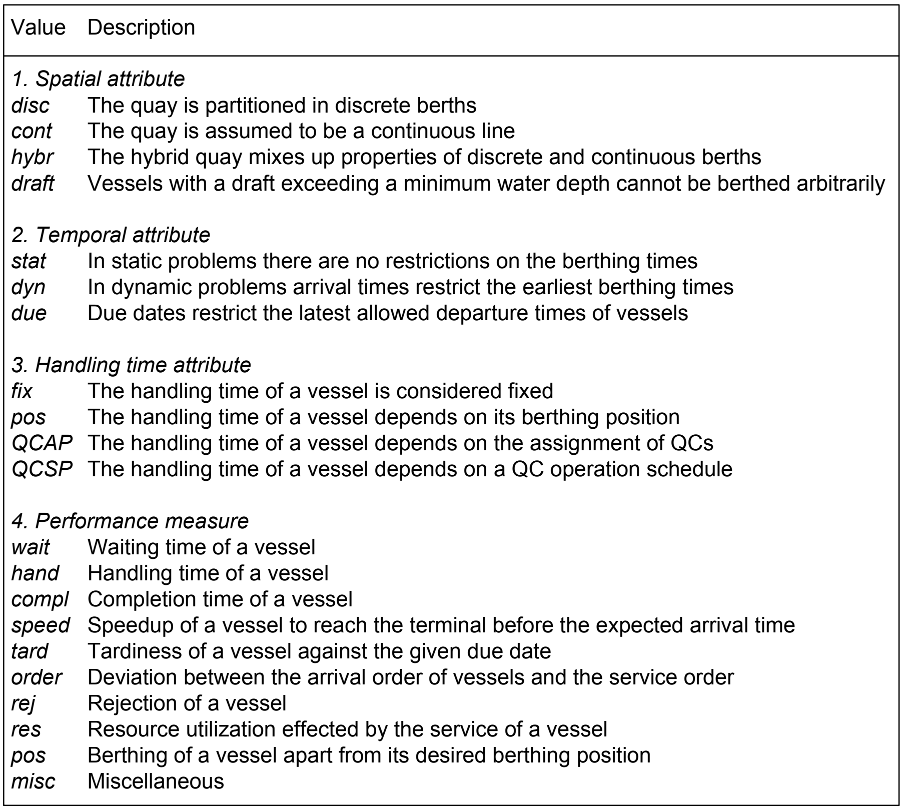
2 Literature review

In [35], the authors present a complete comparative study about different solutions for the BAP according to their efficiency in addressing key operational and tactical questions relating to vessel service. They also study the relevance and applicability of the solutions to the different strategies and contractual service arrangements between terminal operators and shipping lines.

To show similarities and differences in the existing models for berth allocation, a classification scheme is presented in [2] (see Figure 5). They classify the BAP according to four attributes. The spatial attribute concerns the berth layout and water depth restrictions. The temporal attribute describes the temporal constraints for the service process of vessels. The handling time attribute determines the way that vessel handling times are considered in the problem. The fourth attribute defines the performance measure to reflect different service quality criteria. The most important are the criteria that minimize the waiting time (*wait*) and the handling time (*hand*) of a vessel. Both these measures aim at providing a competitive service to vessel operators. If both objectives are pursued (i.e., *wait* and *hand* are set), the port stay time of vessels is minimized. Other measures are focused on minimizing the completion times of vessels. Thus, by using the above classification scheme, a certain type of BAP is described by a selection of values for each one of the attributes. For instance, given a problem where the quay is assumed to be a continuous line (*cont*). The arrival times restrict the earliest berthing of vessels (*dyn*), and handling times depend on the berthing position of the vessel (*pos*). The objective is to minimize the sum of the waiting time (*wait*) and the handling time (*hand*). According to the scheme proposed by [2], this problem is classified by *cont—dyn—pos—*Σ*(wait+hand).*

One of the early works that appeared in the literature developed a heuristic algorithm by considering a First-Come-First-Served (FCFS) rule [17]. However, some authors maintain the idea that for high port throughput, optimal vessel-to-berth assignments should be found without considering the FCFS policy [14]. Therefore, in this paper, we use the FCFS policy in order to get an upper bound. Nevertheless, this approach may result in some vessels' dissatisfaction regarding the order of service.

Several heuristic and metaheuristic approaches have been developed to solve different problems in container terminals. In [2], the authors give a comprehensive survey of berth allocation and quay crane assignment formulations from the literature. Some authors outline approaches more or less informally while others provide precise optimization models. More than 40 formulations that are distributed among discrete problems, continuous prob-



Value	Description
1. Spatial attribute	
disc	The quay is partitioned in discrete berths
cont	The quay is assumed to be a continuous line
hybr	The hybrid quay mixes up properties of discrete and continuous berths
draft	Vessels with a draft exceeding a minimum water depth cannot be berthed arbitrarily
2. Temporal attribute	
stat	In static problems there are no restrictions on the berthing times
dyn	In dynamic problems arrival times restrict the earliest berthing times
due	Due dates restrict the latest allowed departure times of vessels
3. Handling time attribute	
fix	The handling time of a vessel is considered fixed
pos	The handling time of a vessel depends on its berthing position
QCAP	The handling time of a vessel depends on the assignment of QCs
QCSP	The handling time of a vessel depends on a QC operation schedule
4. Performance measure	
wait	Waiting time of a vessel
hand	Handling time of a vessel
compl	Completion time of a vessel
speed	Speedup of a vessel to reach the terminal before the expected arrival time
tard	Tardiness of a vessel against the given due date
order	Deviation between the arrival order of vessels and the service order
rej	Rejection of a vessel
res	Resource utilization effected by the service of a vessel
pos	Berthing of a vessel apart from its desired berthing position
misc	Miscellaneous

Figure 5: A classification scheme for BAP formulation [2].

lems, and hybrid problems are presented. These problems have been mostly considered separately and with an interest mainly focused on BAP [6, 3, 11].

In the integration of both problems, BAP+QCAP, different approaches have been developed considering a discrete quay line, specially genetic algorithms. In [13], a solution based on genetic algorithms is presented for the integration of BAP with the QCAP with the objective of minimizing the total service time. A hybrid genetic algorithm is also presented in [21] where they minimize the sum of the handling time, the waiting time, and the delay time for every ship. In this sense, [10] presents the integration through two mixed integer programming formulations including a tabu search method (adapted from [6]), with the objective of minimizing the yard-related housekeeping costs that are generated by the flows of containers exchanged between vessels.

The integration of BAP+QCAP when the quay line is continuous was first introduced in [26] with a method of two phases. In the first phase, a Lagrangian relaxation based heuristic is used to obtain the berthing position and the number of QCs, and the second phase applies a dynamic programming to obtain the detailed schedule of the QCs. In [24], two different heuristics are presented to solve this model: squeaky wheel optimization and tabu search that show significant results compared with solutions reported by [26].

Focusing only on the holds of one vessel, a genetic algorithm able to solve the quay crane scheduling problem is presented in [20] by determining a handling sequence of holds for the quay cranes assigned to a vessel. Furthermore, in [20], the NP-completeness of this problem for one vessel was proved. The integration of BAP+QCAP problems considering the holds of vessels was first studied by [7] and [27], but they did not consider the interference among QCs. There are similar problems solved by MILP approaches [23]. Results obtained by these exact approaches show that this problems needs to be decomposed into two phases and they cannot solve realistic problems of medium size in a reasonable time.

Our approach deals with the integration of these two problems (BAP+QCAP) through a metaheuristic called Greedy Randomized Adaptive Search Procedures (GRASP) [8], taking into account the requirements of container operators of MSC (Mediterranean Shipping Company S.A.). This metaheuristic is able to find optimized solutions within an acceptable computation time in a very efficient way and it has been applied in a wide range of combinatorial optimization problems [9]. Moreover, several GRASP-based approaches have been developed and applied in different container terminal problems [19, 16].

Following the classification scheme in [2], our approach is represented by *BAP,QCAP*; and specifically, the BAP is defined as by *cont—dyn—QCAP—*Σ*wait*. Thus, we focused on the following attributes and performance measure:

- *Spatial attribute*: we assume that the quay is a continuous line (*cont*), so there is no partitioning of the quay and the vessel can berth at arbitrary positions within the boundaries of the quay. It must be taken into account that for a continuous layout, berth planning is more complicated than for a discrete layout, but it utilizes quay spaces better [2].

- *Temporal attribute*: we assume dynamic problems (*dyn*) where arrival times restrict the earliest berthing times. Since fixed arrival times are given for the vessels, vessels cannot berth before their expected arrival time.

- *Handling time attribute*: we assume that the handling time of a vessel depends on the assignment of QCs (*QCAP*).

- *Performance measure*: Our objective is to minimize the sum of the waiting time (*wait*) of all the scheduled vessels to be served.

This paper presents two approaches for modelling QCAP:

- *Static*: QCs are assigned to one vessel i and they cannot move to another vessel j until the vessel i leaves the container terminal.

- *Dynamic*: QCs are assigned to the holds of the vessel. Thus, once all the movements of one hold are done, the QC can move to another location (another hold in the same or other vessel).

Our approach presents a dynamic and continuous berthing model that takes into account the QCs and the holds of the vessels in order to obtain the handling time. Most of the studies presented above are focused on discrete models without managing the holds of the vessels. Unlike the models presented above having regard to the holds, our approach manages the constraints related to the cranes. Furthermore, our approach differs from crane scheduling problems in that several vessels with different arrival times are the input data.

In the rest of the paper, we specify the above problems (BAP+QCAP, including management of holds) and propose an innovative GRASP-based metaheuristic approach. The results obtained with several scenarios are compared to other allocation methods, contrasting the usefulness of our proposal by efficiently obtaining optimized solutions to these problems.

3 Problem description

The objective in BAP+QCAP is to obtain an optimal distribution of the docks and cranes for vessels waiting to berth. An optimal distribution that takes into account specific constraints (length and depth of vessels, ensuring a correct order for vessels that exchange containers, ensuring departure times, etc.) and optimization criteria (priorities, minimization of waiting and staying times of vessels, satisfaction with the order of berthing, minimizing of crane moves, degree of deviation from a pre-determined service priority, etc.). When the quay is discrete (it is divided in berths), the BAP could be considered as a special kind of machine scheduling problem, where the job and machine are the vessel and the berth, respectively. In machine scheduling, only the starting times of jobs are determined, but in continuous BAPs the berthing positions are also necessary for the output schedule.

In the following, we introduce the notation used for each vessel:

V The set of incoming vessels. Each vessel is denoted as $V_i \in V$.

QC Available QCs in the container terminal. These QCs are identical in terms of the productivity of loading/unloading containers. The parameters of QCs are:

- `movsQC` Number of QC moves per time unit.
- $\mathtt{HH}_{\mathtt{QC}}$ Time units required to reallocate the QC to another hold from the same vessel (Hold-to-Hold movement).
- $\mathtt{VV}_{\mathtt{QC}}$ Time units required to move the QC to another vessel (Vessel-to-Vessel movement).

L Total length of the berth in the container terminal.

$a(V_i)$ Arrival time of the vessel V_i at port.

$l(V_i)$ Length of V_i. There is a safety distance (`safeLength`) between two moored vessels: we assume 5% of the length of the vessels.

$pr(V_i)$ Vessel priority.

$h(V_i)$ Number of holds in V_i. All the holds of the vessels have the same width.

$c_j(V_i)$ Number of required movements to load or unload containers from/into the hold j, $1 \leq j \leq h(V_i)$. The handling time of each hold j ($1 \leq j \leq h(V_i)$) is given by: $\frac{c_j(V_i)}{\texttt{movsQC}}$.

$m(V_i)$ Mooring time of V_i.

$p(V_i)$ Berthing position where V_i will moor.

$q(V_i)$ Number of assigned QCs to V_i. The maximum number of assigned QCs by vessel depends on its length since a safety distance is required between two contiguous QCs (`safeQC`) and the maximum number of QCs that the container terminal allows per vessel (`maxQC`).

$st_j(V_i)$ Starting time of QC j at Vessel V_i, $1 \leq j \leq q(V_i)$. Only one QC can be assigned to one hold.

$ht_j(V_i)$ Handling time of the QC j, $1 \leq j \leq q(V_i)$.

$h_j(V_i)$ Set of handling times of each hold assigned to the QC j, $1 \leq j \leq q(V_i)$.

$d(V_i)$ Departure time of V_i, which depends on $m(V_i)$, $c(V_i)$, and $q(V_i)$.

$w(V_i)$ Waiting time of V_i from its arrival at port until it moors: $w(V_i) = m(V_i) - a(V_i)$.

To simplify the problem, we assume that mooring and unmooring does not consume time, simultaneous berthing is allowed, and every vessel has a draft lower than or equal to the water-depth of the berth. Furthermore, once a QC starts to work in a hold, it must complete it without any pause or shift (non-preemptive tasks). When a QC finishes the movements of one hold, it can move to another hold from the same vessel or to another vessel.

The goal of the BAP+QCAP is to allocate each vessel according to the existing constraints and to minimize the total weighted waiting time of all the vessels:

$$T_w = \sum_{V_i \in V} \left[\left[w(V_i)\right]^{\gamma} \times pr(V_i) \right] \tag{1}$$

The parameter γ ($\gamma \geq 1$) prevents lower priority vessels from being systematically delayed. Thus, the component of each vessel in the optimization function is not exactly linear with its waiting time ($w(V_i)$). In this way, vessels with large waiting times ($w(V_i)$) will have a proper weighting in the objective function, although they have low priority values ($p(V_i)$). For instance, let A and B be two alternative schedules for two vessels V_1 and V_2 with $pr(V_1) = 6$ and $pr(V_2) = 2$, respectively (see Table 1). In the schedule A, the waiting time of V_2 ($w(V_2) = 500$) is much larger than V_1 ($w(V_1) = 150$); whereas the waiting times in schedule B are closer to each other ($w(V_1) = 220; w(V_2) = 310$). Table 1 also shows the values of the objective functions with respect to the γ value for both schedules. If γ is not considered in the objective function ($\gamma = 1$), it is preferable to chose the schedule A ($T_w = 1900$) although V_2 must wait 500 time units. However, with a γ value greater than 1 ($\gamma = 1.2$), the schedule B, which balances the waiting times of V_1 and V_2, is chosen as the best schedule ($T_w = 5835$) avoiding that the vessel with the lowest priority (V_2) is delayed.

Table 1: Example of the γ parameter over two different schedules.

Schedule	$\mathbf{pr(V_1)}$	$\mathbf{pr(V_2)}$	$\mathbf{w(V_1)}$	$\mathbf{w(V_2)}$	$\mathbf{T_w}$ $(\gamma = \mathbf{1})$	$\mathbf{T_w}$ $(\gamma = \mathbf{1.2})$
A	6	2	150	500	1900	5917
B	6	2	220	310	1940	5835

There are two other key factors within container terminals: the ratio of berth usage (B_u), and the quay crane throughput (T_{qc}). Berth usage is obtained by Equation (4). It reflects the area held by vessels with respect to the maximum area. The maximum area depends on the length of the quay (L) and the mooring time of the first vessel calculated by (2) and the departure time of the last vessel calculated by (3).

$$firstArrival = \min_{V_i \in V}\{m(V_i)\} \tag{2}$$

$$lastDeparture = \max_{V_i \in V}\{d(V_i)\} \tag{3}$$

Thus, the ratio of the berth usage (B_u) is:

$$B_u = \frac{\sum\limits_{V_i \in V} \big[l(V_i) \times (d(V_i) - m(V_i))\big]}{L \times (lastDeparture - firstArrival)} \tag{4}$$

The QC throughput factor (T_{qc}) depends on the model for the QCAP under consideration. The static QCAP model calculates T_{qc} by means of (5), taking into consideration that one QC remains at the same vessel until it departs. Thereby, all the QCs are assigned to vessel V_i up to its departure time $d(V_i)$ even though these QCs are not moving any container from/to the vessel:

$$\text{Static QCAP: } T_{qc} = \sum_{V_i \in V} \big[q(V_i) \times (d(V_i) - m(V_i))\big] \tag{5}$$

$$\text{Dynamic QCAP: } T_{qc} = \sum_{V_i \in V} \sum_{1 \le j \le q(V_i)} ht_j(V_i) \tag{6}$$

However, the dynamic QCAP model uses (6). This model considers that once one QC finishes its task at one hold, it can move to another location. Therefore, the time that each QC is busy is just its handling time.

Figure 6 shows an example of a vessel containing 5 holds with different number of containers each, thus with different handling times each hold. In this example, a QC has been allocated to each hold. The shaded rectangle indicates the time a QC works on a hold and the arrow represents the time that the QC is assigned to the vessel. In the static QCAP model (see Figure 6(a)), they stay until the vessel departs, while the dynamic QCAP model allows to move one QC to another hold or vessel before the vessel departs (see Figure 6(b)).

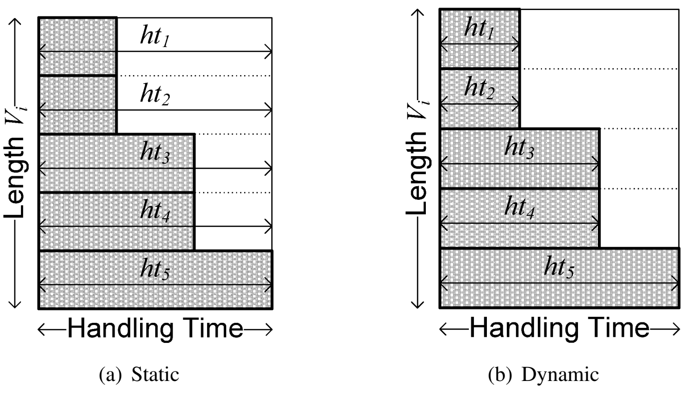
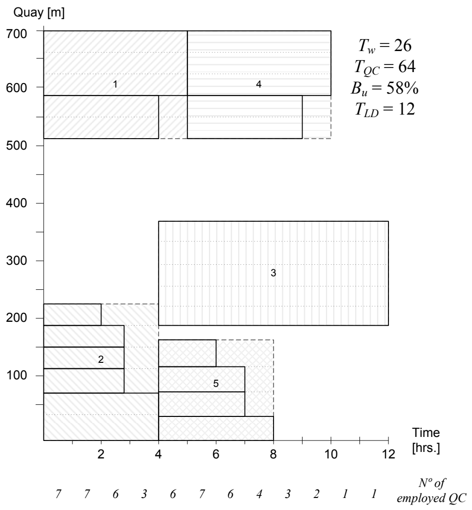


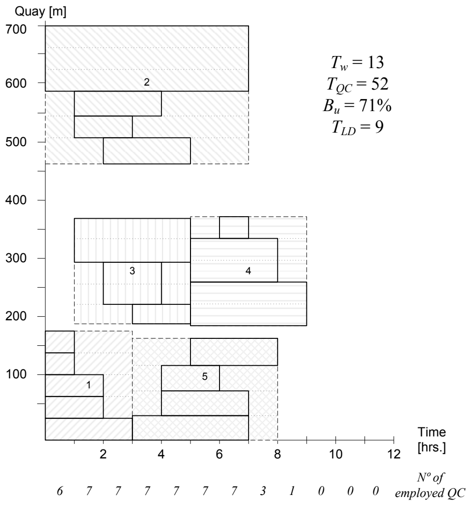
Figure 6: Differences between static and dynamic models for calculating T_{qc} for QCAP.

By taking into consideration the holds ($h(V_i)$) of each vessel, our model makes better use of the resources (QCs and berth) as shown in Figure 7. In this figure, a schedule of 5 vessels is shown. Each dashed rectangle represents a vessel with its id number and each bold rectangle represents the time that one QC is working on a hold.

Figure 7(a) shows the static QCAP model. Thus, when one QC is assigned to one vessel i, this QC cannot be moved to another vessel j until vessel i leaves the container terminal. Figure 7(b) shows the dynamic QCAP model and introduces the concept of holds. Once a QC finishes a task that is related to one hold in the vessel i, it could keep working on another hold of the same vessel or move to another vessel j. Thus, the dynamic model obtains the following: a departure time of the last vessel (T_{LD}) that is earlier than the static model (from 12 to 9 time units); a waiting time (T_w) that is lower than the static one (from 26 to 13 time units). Thus, we can see that the berth usage ratio is increased by 20$\%$. In other words, with dynamic QCAP model more QCs are used per time unit than with static QCAP, and the total time that the QCs are tied to one vessel (T_{qc}) is also reduced.



(a) QCs assigned to one vessel (static)



(b) QCs assigned to holds of one vessel (dynamic)

Figure 7: Plans obtained with and without the holds.

4 Mooring one vessel

Once the problem has been defined, the whole process to moor one vessel is presented. The *MoorVessel* function (Algorithm 1) moors one vessel V_i as early as possible, starting at its arrival time ($a(V_i)$). This function checks whether V_i can moor at time t or when a QC has completed its task (steps 9 to 16). The data required for this function is the vessel to allocate resources (V_i) and the set of moored vessels scheduled previously by the algorithm (V_m). There are three steps in this mooring process (see Figure 8):

1. To verify whether there are QCs available during the handling time of V_i (Algorithm 2).

2. To make sure there is enough continuous length at the berth to moor V_i.

3. To assign more QCs when it is possible (Algorithm 4).

The *InsertVessel* function (Algorithm 2) performs, if it is possible, the steps needed to allocate one vessel at the given time t. First, a quick check of the available quay length is carried out in order to know if there would be enough space to moor V_i (steps 1 to 4). Then, the maximum number of QCs for V_i is calculated from its mooring time until its departure (steps 5 to 31). This process starts assigning just one QC and increases this number according to the available QCs found during the vessel handling time (t, t_f). Available QCs are obtained by counting the number of cranes used by the other vessels (*CranesWorking* function) at their mooring time and whenever one of their assigned QCs

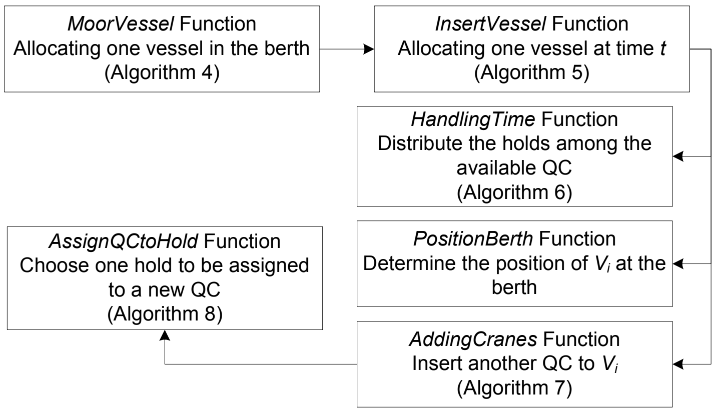

MoorVessel Function
Allocating one vessel in the berth
(Algorithm 4)
InsertVessel Function
Allocating one vessel at time *t*
(Algorithm 5)
HandlingTime Function
Distribute the holds among the available QC
(Algorithm 6)
AssignQCtoHold Function
Choose one hold to be assigned to a new QC
(Algorithm 8)
PositionBerth Function
Determine the position of *V_i* at the berth
AddingCranes Function
Insert another QC to *V_i*
(Algorithm 7)

Figure 8: Execution order of the presented algorithms.

has completed its task. As we have mentioned above, in this paper, we consider that each QC is assigned to just one hold; therefore initially, the holds of one vessel ($h(V_i)$) are distributed among the different QCs by the *HandlingTime* function (Algorithm 3). Finally, the berthing position is calculated by the *PositionBerth* function, and, when it is possible, additional QCs are assigned to V_i by the *AddingQuayCranes* function (Algorithm 4).

The allocation of the holds of V_i to the different QCs is done in two phases (Algorithm 3). Step 1 calculates the time needed to complete all the movements of each hold. Later, in steps 4-13, each hold is assigned to the given QCs in descending order according to their handling times.

After determining this first number of QCs, we must determine if there is enough continuous length and assign a berthing position. At this moment, the length of the safety distance between two contiguous vessels is taken into account (`safeLength`). Every space between each two vessels is examined and among all the possible positions, the one chosen will be the closest to the ends of the berth. This strategy is followed because if vessels are moored at these positions, the incoming vessels will have more contiguous available length each time that a vessel departs.

Then, if the vessel V_i has QCs and the length available to get moored at time t, more QCs are assigned in order to reduce the vessel service time. This process is carried out by the *AddingCranes* function (Algorithm 4) and is based on obtaining the period of time between $(m(V_i), d(V_i))$ in which there is at least one available QC without reaching the limit of QCs (`maxQC`) assigned to V_i. Once a period (t_j, t_k) has been found, the *AssignQCtoHold* function (Algorithm 5) searches among the holds assigned to QCs to determine which hold H carried out by the QC k begins later and can be completed in the given interval $(start, end)$. The selected hold H could be moved to the list of tasks of an already assigned QC to V_i (steps 11 to 17), or a new QC could be assigned to V_i to work on this selected hold H (steps 18 to 33). In either case, if any QC becomes idle because it has no assigned holds (steps 14 or 20), it performs the tasks of the last QC assigned to

Algorithm 1: `MoorVessel` function. Allocating exactly one vessel in the berth.

```
    Data: $V_i$: vessel to moor; $V_m$: Vessels already moored;
    Result: $V_i$: vessel with all the resources allocated;
 1  if $|V_m| = 0$ then
        // There is no other moored vessel
 2      $nQC := \max\left(1, \min\left(\texttt{maxQC}, \left\lfloor \frac{l(v)}{\texttt{safeQC}} \right\rfloor\right)\right)$;
 3      HandlingTime($V_i, nQC$);
 4      $m(V_i) := a(V_i)$;
 5      $d(V_i) := m(V_i) + \max_{1 \leq j \leq q(V_i)}(st_j(V_i) + ht_j(V_i))$;
 6      $q(V_i) := nQC$;
 7      $p(V_i) := 0$;
 8  end
    // There are other vessels. Moor at the earliest possible time
 9  $T \leftarrow \{a(V_i)\} \cup \{(st_k(j) + ht_k(j)) \mid j \in V_m, 1 \leq k \leq q(j) \wedge\ st_k(j) + ht_k(j) > a(V_i)\}$;
10  Sort($T$);                                  // Sort in ascending order
11  forall $t_k \in T$ do
12      $inst$ :=InsertVessel($V_i, t_k + \texttt{VV}_{\texttt{QC}}, V_m$);
13      if $inst$ then
14          break;
15      end
16  end
17  return $V_i$;
```

V_i, which then becomes available for the other vessels.

Finally, if the vessel V_i cannot be moored at time t, the whole process described above is repeated taking into consideration another time (t_k) to moor V_i (Algorithm 1). Each time t_k represents the moment in time that a QC finishes working on the hold of another vessel.

5 A metaheuristic method for BAP+QCAP

For the BAP+QCAP problem addressed in this paper, we developed different methods, which allow us compare their results. First, different rules R have been developed following different criteria. Algorithm 6 shows the schema to schedule all the incoming vessels according to a specific rule R. Following the order given by the rule R, all vessels are chosen one by one to be moored. Each scheduled vessel is added to the set of moored vessels V_m. Generally, a vessel can be allocated at time t when there is no vessel moored in the berth or there is available contiguous quay length as well as enough free QCs to be assigned.

The rules R implemented for its application in Algorithm 6 are:

- *FCFS*: Vessels moor according to their arrival order, thus $\forall i, m(V_i) \leq m(V_{i+1})$.
- *FCMP* (First Come Maximum Priority): Similar to FCFS where the next vessel is chosen according to their arrival order but, in this case, there is no restriction with the time the vessels can moor.
- *MWWT* (Maximum Weighted Waiting Time): Each vessel is ranked according to their weighted waiting time. The vessel with the greatest value is moored first.

Algorithm 2: `InsertVessel` function. Allocating one vessel in the berth at time t.

```
Data: $V_i$: Vessel to allocate; $t$: Actual time; $V_m$: Vessels already moored;
Result: A boolean indicating whether $V_i$ could moor or not;
// Check the length available at the quay
1  $L_{avail} \leftarrow L - \left(\sum_{v_j \in V_m} l(v_j) \mid m(v_j) \leq t \wedge d(v_j) > t\right)$;
2  if $L_{avail} \leq l(v)$ then
3      return false;
4  end
5  $cranes := -1$;   $cranes_m := -1$;
6  repeat
7      $nc := \max(1, cranes)$;
8      HandlingTime$(V_i, nc)$;                    // Handling time given the $nc$ QCs
9      $t_f := t + \max(st_j(V_i) + ht_j(V_i)),\ 1 \leq j \leq q(V_i)$;
       // Vessels which coincide with $V_i$
10     $W \leftarrow \{v \in V_m \mid d(v) > t \wedge m(v) < t_f\}$;
       // Find the maximum number of QCs available to be used in the interval
          $(t, t_f)$
11     $cranes_m := nc$;
12     $cranes := \max\left(1, \min\left(\texttt{maxQC}, \left\lfloor \frac{l(V_i)}{\texttt{safeQC}} \right\rfloor\right)\right)$;
13     $QC_u := \sum_{\forall v \in W}$ CranesWorking$(v, t)$;
14     $cranes := \min(cranes, QC - QC_u)$;
15     forall $i \in W$ do
16         if $m(i) \geq t$ then
17             $QC_u := \sum_{\forall v \in W}$ CranesWorking$(v, m(i))$;
18             $cranes := \min(cranes, QC - QC_u)$;
19         end
20         for $j \leftarrow 1$ to $q(i)$ do
21             $t_q := st_j(v) + ht_j(v)$;
22             if $t_q \geq m(V_i)\ \wedge\ t_q < d(V_i)$ then
23                 $QC_u := \sum_{\forall v \in W}$ CranesWorking$(v, t_q)$;
24                 $cranes := \min(cranes, QC - QC_u)$;
25             end
26         end
27     end
28     if $cranes \leq 0$ then
29         return false;
30     end
       // Repeat until the number of available QCs does not change
31 until $cranes_m \leq cranes$ ;
   // Assign the number of QCs and the mooring and departure times
32 $q(V_i) := cranes_m$;
33 $m(V_i) := t$;   $d(V_i) := t_f$;
   // Look for a berthing position for the vessel
34 $insert :=$ PositionBerth$(V_i, W)$;
35 if $insert$ then
       // Once $V_i$ is scheduled, try to assign it more QCs
36     AddingCranes$(V_i, V_{in})$;
37 end
38 return $insert$;
```

Algorithm 3: `HandlingTime` function. Distribute the holds among the available QCs.

```
   Data: $V_i$: Vessel to allocate; $nQC$: number of QCs;
   Result: $V_i$ with the holds allocated to the QCs;
   // Calculate the handling time required of each hold
1  $T \leftarrow \left\{ \frac{c_j(V_i)}{\texttt{movsQC}} \mid 1 \leq j \leq h(V_i) \right\}$;
2  Sort($T$) ;                                   // Sort the values of $T$ in descending order
3  $q(V_i) := nQC$;
   // Allocate the holds with more containers to each QC
4  for $j \leftarrow 1$ to $nQC$ do
5      $st_j(V_i) := m(V_i)$;
6      $ht_j(V_i) := \lceil T_j \rceil$;
7      $h_j(V_i) \leftarrow \{\lceil T_j \rceil\}$;
8  end
   // Do a greedy allocation for the rest of holds
9  for $j \leftarrow nQC + 1$ to $h(V_i)$ do
       // choose the QC which ends earlier
10     $q_m :=$ argmin$(st_j(V_i) + ht_j(V_i),\ 1 \leq j \leq nQC)$;
11     $h_{q_m}(V_i) \leftarrow h_{q_m}(V_i) \cup \{T_j\}$;
12     $ht_{q_m}(V_i) := ht_{q_m}(V_i) + \lceil T_j \rceil + \texttt{HH}_{\texttt{QC}}$;
13 end
14 Sort QCs in descending order according to their finish times;
15 return $V_i$;
```

Algorithm 4: `AddingCranes` function. Insert another QC to V_i.

```
   Data: $V_i$: Vessel to insert QCs; $V_m$: Vessels already moored;
   Result: $V_i$: Vessel with QCs reallocated;
   // Obtain the period of time $(t_j, t_k)$ that at least there is 1 available QC
1  repeat
2      changes:= 0;
       // Set of vessels $W$ which are moored at the same time as $V_i$
3      $W \leftarrow \{v \in V_m \mid d(v) > a(V_i) \ \wedge \ m(v) < d(V_i)\}$;
       // $T$ is the set of mooring and ending times of each QC of $W$
4      $T \leftarrow \{m(v) \mid v \in W \ \wedge \ m(v) \geq m(V_i)\}$;
5      $T \leftarrow T \cup \{st_j(v) + ht_j(v) \mid v \in W,\ 1 \leq j \leq q(v) \ \wedge$
       $(st_j(v) + ht_j(v)) \geq m(V_i) \ \wedge \ (st_j(v) + ht_j(v)) < d(V_i)\}$;
       // Sort the set of time units $T$ in ascending order
6      sort($T$);
7      forall $t_j \in T$ do
8          if CranesWorking($V_i$, $t_j$) ¡ maxQC then
9              forall $t_k \in T \mid t_k > t_j$ do
10                 Obtain the number of available QCs ($Q_a$) in the interval $(t_j, t_k)$;
11                 if $Q_a$ ¿ 0 then
12                     changes := changes + AssignQCtoHold($V_i$, $T_j$, $t_k$);
13                 else
                       // Continue with the next time unit $t_j$
14                     break;
15                 end
16             end
17         end
18     end
       // This loop is repeated while there is any change
19 until changes $= 0$ ;
20 return $V_i$;
```

Algorithm 5: `AssignQCtoHold` function. Choose one hold to be assigned to a new QC.

```
   Data: $V_i$: vessel to moor; $start$: starting time; $end$: ending time;
   Result: QC added (1) or not (0);
   // Search a hold whose handling task is lower or equal to $end - start$;
   // $last(h_j(V_i))$ points to the last hold assigned to this QC
1  Sort Cranes by their finish time in descending order;
   // Handling time of the hold
2  $H := \max(last(h_j(V_i))), 1 \leq j \leq q(V_i) \ \wedge \ last(h_j(V_i)) < (end - start)$;
   // QC $k$ which works on this hold
3  $k := \text{argmax}(last(h_j(V_i))), 1 \leq j \leq q(V_i) \ \wedge \ last(h_j(V_i)) < (end - start)$;
   // QC $m$ which finishes at $start$ time
4  $m := j \mid 1 \leq j \leq q(V_i) \ \wedge \ st_j(V_i) = start$;
5  if $H \neq \varnothing$ then
       // There is a hold $H$ that fits in the given interval
6      if $(start + H) \geq d(V_i) \vee (start + H) \geq (st_k(V_i) + ht_k(V_i))$ then
           // Move the $H$ to another QC would not improve the actual schedule
7          return 0;
8      end
       // Delete the hold from the list of tasks of QC $k$
9      $h_k(V_i) \leftarrow h_k(V_i) - \{H\}$;
10     $ht_k(V_i) := ht_k(V_i) - H - \mathtt{HH_{QC}}$;
11     if $m \neq \varnothing \wedge k \neq m$ then
           // There is a QC $m$ finishing its tasks at $start$.  Add task $H$ to QC $m$
12         $ht_m(V_i) := ht_m(V_i) + H + \mathtt{HH_{QC}}$;
13         $h_m(V_i) \leftarrow h_m(V_i) \cup \{H\}$;
14         if $|h_k(V_i)| = 0$ then
               // As QC $k$ becomes idle
15             Reallocate the tasks of the last QC of $V_i$ to the QC $k$;
16             $q(V_i) := q(V_i) - 1$;
17         end
18     else
           // There is no QC that finishes at $start$, so a new QC will be assigned
              to $V_i$
19         if $|h_k(V_i)| > 0$ then
               // Check whether the tasks can be joined in the same QC
20             for $j \leftarrow k + 1$ to $q(V_i)$ do
21                 if $st_k(V_i) + ht_k(V_i) = st_j(V_i)$ then
22                     $ht_k(V_i) := ht_k(V_i) + ht_j(V_i) + \mathtt{HH_{QC}}$;
23                     $h_k(V_i) \leftarrow h_k(V_i) \cup h_j(V_i)$;
                       // As QC $j$ becomes idle
24                     Reallocate the tasks of the last QC of $V_i$ to the QC $j$;
25                     $q(V_i) := q(V_i) - 1$;
26                 end
27             end
               // Assign the new QC to vessel $V_i$ and increase the number of QCs
28             $q(V_i) := q(V_i) + 1$;
29             $k := q(V_i)$;
30         end
31         $st_k(V_i) := start + \mathtt{VV_{QC}}$;   $ht_k(V_i) := H$;
32         $h_k(V_i) \leftarrow \{H\}$;
33     end
       // Obtain the new departure time of $V_i$ according to the QCs assigned
34     $d(V_i) \leftarrow \max_{\forall j \in q(V_i)} (st_j(V_i) + ht_j(V_i))$;
35     Sort cranes by their finish time in descending order;
36 end
37 return 1;
```

Algorithm 6: Vessels Allocation according to rule R.

```
Data: $V_{in}$: set of ordered incoming vessels; $R$: rule applied;
Result: $V_m$: set of vessels with all the resources allocated;
$V_{last} := \varnothing$;
$V_m \leftarrow \varnothing$;
while $V_{in} \neq \varnothing$ do
    // Next vessel according to the rule $R$
    $v :=$ nextByRule($R$);
    $t := a(v)$;
    // For FCFS rule, the earliest mooring time possible is the mooring time of
       the previous vessel
    if $R = FCFS$ then
        $t := \max(m(V_{last}), a(v))$;
    end
    // Schedule the chosen vessel at the earliest possible time
    $inst :=$ InsertVessel($v, t, V_m$);
    if ! $inst$ then
        $T := \{st_k(v_j) + ht_k(v_j) \mid v_j \in V_m, 1 \leq k \leq q(v_j) \wedge st_k(v_j) + ht_k(v_j) > t\}$;
        while $t_k \in T \wedge$ ! $inst$ do
            $inst :=$ InsertVessel($v, t_k + \mathtt{VV_{QC}}, V_m$);
        end
    end
    // Store the last scheduled vessel
    $V_{last} := v$;
    // Update the moored and the unscheduled vessels
    $V_m \leftarrow V_m \cup \{v\}$;   $V_{in} \leftarrow V_{in} - \{v\}$;
end
return $V_m$;
```

- *EWMT* (Earliest Weighted Mooring Time): Among the vessels that can moor earlier, the operator chooses the vessel with the highest priority.

In this study, these rules are compared with our new method for the Berth Allocation and Quay Crane Assignment Problem: a metaheuristic GRASP approach. This is a randomly-biased multi-start method to obtain optimized solutions of hard combinatorial problems in a very efficient way. This method consists of two phases (Procedure 7) and these two phases are performed consecutively until a termination condition is met. This termination condition is given in one of these two forms: (1) a number of iterations; or (2) a time limit. The best solution obtained in those iterations is returned as the solution for the instance.

The first phase focuses on building a solution by means of adding one element at a time. In order to choose the next element for the solution, the elements that are not moored yet are evaluated using a greedy function that indicates how a candidate contributes to the final solution. Then, a random degree (δ) determines the number of candidates that could be eligible for this random choice election. If $\delta = 1$, all the elements are eligible, and therefore this choice is completely random. If $\delta = 0$, then it results in a completely greedy search. The second phase of the GRASP metaheuristic carries out a local search algorithm in order to improve each constructed solution in the previous phase. This local search algorithm works in an iterative manner by successively replacing the current solution by a better solution in the neighborhood of the current solution. It terminates when no better solution is found in the neighborhood [8].

Algorithm 7: GRASP framework

```
   Data: Max_iterations or Time Limit;
 1 Read input();
 2 while No Termination condition is satisfied do
       // Construction phase
 3     $S \leftarrow \varnothing$;
 4     Evaluate the incremental costs of the candidate elements;
 5     while $S$ is not a complete solution do
 6         Build the restricted candidate list (RCL);
 7         Select an element $s$ from the RCL randomly;
 8         $S \leftarrow S \cup \{s\}$;
 9         Reevaluate the incremental costs;
10     end
       // Local Search phase
11     $S \leftarrow$ LocalSearch($S$);
12     Keep track of the best solution $S$ found in BestSolution;
13     Increase the number of iterations;
14 end
15 return BestSolution
```

The GRASP-based procedure developed for the BAP+QCAP problem is detailed in Algorithm 8. This algorithm receives as parameters: the set of incoming vessels V_{in} waiting for mooring at the berth, the random degree (δ), the number of neighbors to explore in the local search algorithm (K) and the maximum number of iterations (I_{max}). These parameters will be discussed in Section 6. First, all the waiting vessels V_{in} are considered as candidates C. In step 9, each one of the candidate vessels is moored within the current state (being assigned the mooring and departure times ($m(V_i)$,$d(V_i)$), the number of QCs ($q(V_i)$), and the berthing position ($p(V_i)$)); and they are evaluated according to the greedy function f_c. Given a candidate vessel v_e, the greedy function assigned to v_e is the sum of weighted service time of each vessel v_o that is still waiting (steps 11 to 14).

According to the greedy function f_c and the random degree indicated by δ, a Restricted Candidate List (RCL) is created (step 18). Then, one vessel v is chosen randomly among the elements from the RCL to be moored and can no longer be modified (step 19). Once v is determined, this is added to the set of vessels V_s and eliminated from the candidate list C (step 21). This loop is repeated until C is empty, it means that all the vessels have been moored.

The solutions given by the construction phase of the GRASP metaheuristic always obtain valid solutions: The construction phase works as a dispatching rule by choosing each time a vessel from the RCL and inserting it into the partial schedule (set of vessels already scheduled, V_s). The Algorithms of the Section 4 check that all the constraints are met when a vessel is scheduled such as, among others, the safety distance between every pair of vessels or the number of QCs assigned to it. Repeating this operation for each incoming vessel obtains a feasible and valid schedule.

The second phase of the GRASP metaheuristic is shown in Algorithm 9. In order to define the neighborhood structure of the local search algorithm, a dispatching rule

Algorithm 8: Grasp metaheuristic adapted to BAP+QCAP.

Data: δ: random factor; V_{in}: incoming Vessels; K: number of neighbors; I_{max}: maximum number of iterations;
Result: V_m: set of vessels with all the resources allocated;

```
1  iters ← 0;
2  $V_m \leftarrow \varnothing$;
3  while No Termination condition is satisfied ($I_{max}$) do
       // Initialize the actual schedule and the candidates
4      $V_s \leftarrow \{\}$;
5      $C \leftarrow V_{in}$;
6      while $C \neq \varnothing$ do
           // Evaluate the incremental costs of each candidate
7          forall $v_e \in C$ do
8              $f_c(v_e) := 0$;
9              MoorVessel($v_e, V_s$);
10             $V_s' \leftarrow V_{in} \cup \{v_e\}$;
11             forall $v_o \in C \mid v_o \neq v_e \wedge a(v_o) \leq a(v_e)$ do
12                 MoorVessel($v_o, V_s'$);
13                 $f_c(v_e) := f_c(v_e) + ((d(v_o) - a(v_o)) \times pr(v_o))$;
14             end
15         end
           // Build the Restricted Candidate List
16         $c_{inf} := \min\{f_c(e) \mid e \in C\}$;
17         $c_{sup} := \max\{f_c(e) \mid e \in C\}$;
18         $RCL \leftarrow \{e \in C \mid f_c(e) \leq c_{inf} + \delta \times (c_{sup} - c_{inf})\}$;
           // Choose a vessel randomly
19         $v :=$ Random($RCL$);
20         MoorVessel($v, V_s$);
           // Insert $v$ in the partial schedule
21         $V_s \leftarrow V_s \cup \{v\}$;   $C \leftarrow C - \{v\}$;
22     end
       // Local search phase
23     $V_s \leftarrow$ LocalSearch($V_s, K$);
24     Keep track of the best schedule found $V_s$ in $V_m$;
25     iters ← iters +1;
26 end
27 return $V_m$;
```

Algorithm 9: *LocalSearch* function. Local Search based on Hill Climbing for BAP+QCAP

Data: S: Schedule (set of vessels) built by the construction phase; K: number of neighbors to generate;
Result: S^*: Best schedule found from the schedule S;

```
1  $S^* \leftarrow S$;        // $S^*$ is the current schedule.  Set of vessels with all resources
   allocated
2  repeat
3      improves := false;
       // Generate K neighbors from the current schedule
4      $S' \leftarrow \varnothing$;                                        // Best neighbour found
5      for $k \leftarrow 1$ to $K$ do
6          $S_n \leftarrow \varnothing$;                          // Generate a new schedule $S_n$
           // Considering the mooring times of the vessels in $S^*$ as dispatching
              rule, generate a new schedule $S_n$
7          Sort the vessels in $S^*$ by their mooring times;
8          Choose randomly two vessels $i, j$;
9          for $v \in S^*$ do
               // Interchange $i$ and $j$ within the order given by schedule $S^*$
10             if $v = i$ then
11                 MoorVessel($j$, $S_n$);
12             else if $v = j$ then
13                 MoorVessel($i$, $S_n$);
14             else
15                 MoorVessel($v$, $S_n$);
16             end
17         end
18         Keep track of the best neighbor schedule found in $S'$;
19     end
20     if $S'$ is better than $S^*$ then
21         $S^* \leftarrow S'$;
22         improves := true;
23     end
24 until ! improves ;
25 return $S^*$;
```

based on the order of the vessels according to their mooring times is applied. Thereby, a neighbor of a current schedule is created by means of interchanging (randomly chosen) two vessels in the dispatching rule (steps 7 to 17). This local search, based on the hill climbing technique, starts with the set of vessels V_s with all the resources allocated (step 1) as the current schedule S^*. K schedules from the neighborhood of the current schedule S^* are generated (step 5). If the best obtained neighbor schedule S' outperforms the current schedule S^* (step 20), according to the objective function T_w, then the current schedule S^* is replaced by S'. This loop is repeated until there is no neighbor schedule better than the current schedule (steps 2 to 24).

According to the GRASP metaheuristic framework, this search is repeated according to the number of iterations or to the time limit specified by the user. The best solution found according to the objective function T_w (V_m) is returned as the solution for the given instance of the problem.

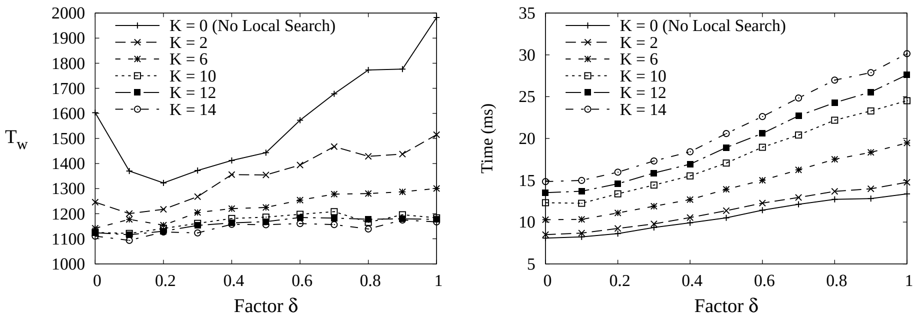
6 Evaluation

Several experiments have been performed with two different corpus: *Dens* and *Spar*. *Spar* means that the arrival time between two vessels is sparsely distributed, and *Dens* means that the arrival time between two vessels is densely distributed. Each corpus contains 100 instances generated randomly, each one composed of a queue from 5 to 20 vessels. The terminal operators gave us two inter-arrival distributions (exponential with parameters $\lambda_{Dens} = \frac{1}{2}$ and $\lambda_{Spar} = \frac{1}{5}$, and poisson with $\lambda_{Dens} = 1.5$ and $\lambda_{Spar} = 3$ distributions) for each corpus in order to generate the arrivals for the incoming vessels. The number of required movements and length of vessels are randomly generated between 100 and 1000 containers, and between 100 and 500 meters, respectively. In all cases, the berth length (L) is fixed to 700 meters; the number of Quay Cranes is 7 (corresponding to a determined MSC berth line) and the maximum number of QCs per vessel is 5 (`maxQC`); the safety distance between QCs (`safeQC`) is 35 meters and the number of movements that QCs carry out is 25 (`movsQC`) per time unit. The time needed for the QCs to move to another hold is 5 time units ($\mathtt{HH_{QC}}$), and 15 time units to another vessel ($\mathtt{VV_{QC}}$). These values were estimated by the terminal operators. Without loss of generality, all the experiments were conducted assuming $\gamma = 1$.

As we mentioned above, our goal is to minimize the total weighted waiting time elapsed to serve the set of n incoming vessels. A personal computer equipped with a Intel Core 2 Q9950 2.83Ghz with 4GB RAM was used in all the experiments.

Focusing on the static QCAP model, Figure 9 shows the objective function (T_w) and the computation times obtained by the GRASP metaheuristic by varying the parameter K of the local search. This experiment was carried out for the *Dens* corpus with an exponential inter-arrival distribution of arrivals. Using just the constructive phase of the GRASP metaheuristic ($K = 0$), the best value achieved was 1322.7 with $\delta = 0.2$ (see Figure 9(a)). In general, the greater the K value, the better T_w values since a deeper search in the neighborhood is carried out. For instance, the T_w obtained by $\delta = 0.2$

decreased to 1127 when $K = 14$ neighbors are generated in each step of the local search. However, we can see that for $K > 12$, we did not achieve a significance improvement in the objective function. Furthermore, it is important to note that the greater the K value for the local search, the greater the computation time. Given the $\delta = 0.2$, the computation time increased from 8.23 ms up to 15.97 ms per iteration. Therefore, a value $K = 12$ was set for the local search phase of the GRASP metaheuristic for all the following experiments.



(a) Values T_w (b) Computation times

Figure 9: Local search depending on the k value (*Dens* corpus with exponential inter-arrival distribution)

Table 2: Average T_w values for the rules and GRASP in the static QCAP model (20 vessels)

	Exp-Dens	**Exp-Spar**	**Poisson-Dens**	**Poisson-Spar**
FCFS	2222.26	484.45	2668.92	1116.9
FCMP	1973.21	443.26	2414.39	922.52
MWWT	2047.52	477.82	2473.57	1042.53
EWMT	1939.64	370.41	2427.8	841.79
GRASP	1414.75	273.06	1762.22	606.25

In Table 2, the dispatching rules detailed in Sect. 5 are evaluated. Each row represents the average T_w obtained by a rule on each corpus using the static QCAP model. The *EWMT* rule turned out to be the one that achieves the best results in three out of the four corpus studied. Thus, this rule will be used as a baseline for our GRASP metaheuristic algorithm.

The GRASP-based metaheuristic developed was compared with the *EWMT* rule using the two models presented: static and dynamic QCAP. Figure 10 shows the average values for the objective function (T_w) to allocate 10 vessels with an exponential inter-arrival distribution over the two corpus. As expected, the dynamic model obtained better solutions than the static one. For instance, for the *Dens* corpus (see Figure 10(a)), for $\delta = 0.2$, the value of T_w in the static QCAP model was 315.82 and decreased to 260.96 in the dynamic QCAP model. Moreover, it can be observed that the solutions given by the GRASP method always outperformed the *EWMT* solution in the two models, specially with δ values close to 1.0 for both the *Dens* and *Spar* corpus.

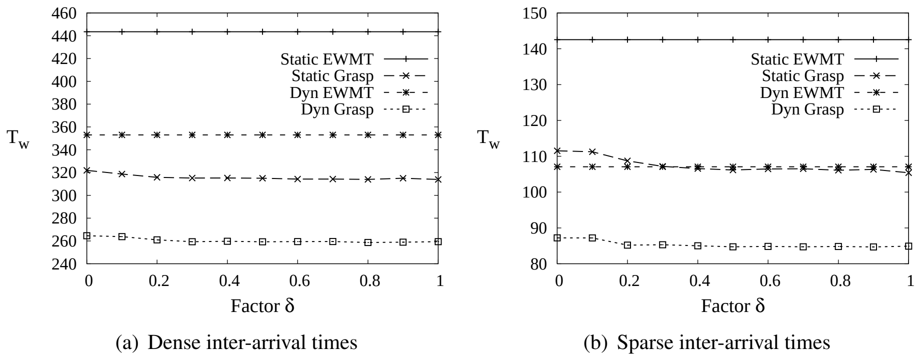


Figure 10: T_w for 10 vessels (Exponential).

Figure 11 shows the average T_w values obtained by the *EWMT* rule and the GRASP metaheuristic in the dynamic QCAP model. This experiment was carried out with instances of 20 incoming vessels with an exponential inter-arrival distribution of the *Dens* corpus using a different number of iterations for the GRASP method. It can be observed that as the number of iterations increased, the quality of our GRASP method also increased. For instance, for $\delta = 0.1$, T_w was 1135.4 with 100 iterations, while T_w decreased to 1110.21 with 400 iterations.

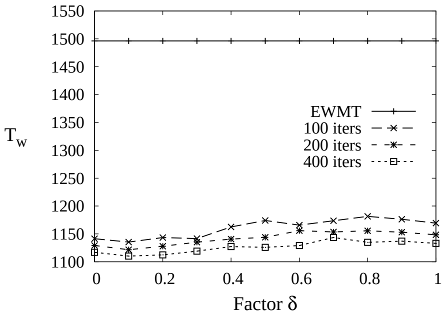
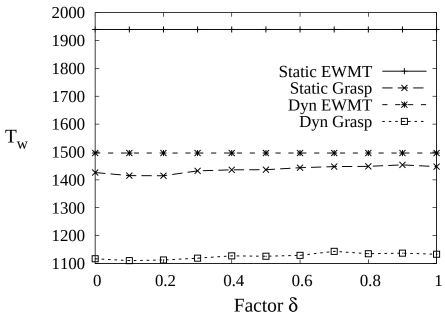
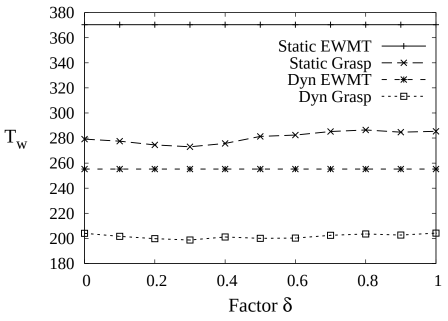


Figure 11: T_w depending on the number of iterations in GRASP.

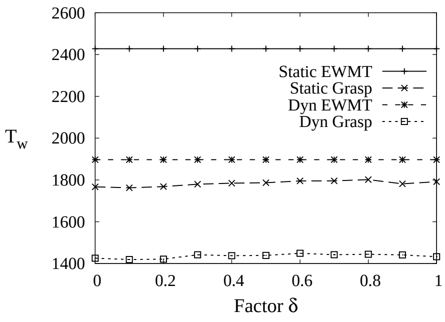
In Figure 12, the same evaluation was carried out for a queue of 20 vessels with the exponential and poisson inter-arrival distributions, respectively. The same tendency as in Figure 10 can be observed, but, in this case, $\delta \in [0.1 - 0.3]$ got the lowest values for both inter-arrival time distributions in both corpus (*Dens* and *Spar*). Moreover, the GRASP metaheuristic improved the average results given by the *EWMT* rule for each corpus. Taking into account the dynamic QCAP model, the results were improved by about $22\% - 25\%$ with respect to the dynamic *EWMT* rule; and, in the static QCAP model, they were improved by about $26\% - 27\%$ with respect to the static *EWMT*. These figures also show how the dynamic QCAP model always outperformed the static one. For instance, considering the poisson inter-arrival distribution and $\delta = 0.1$ (see Figure 12(d)), the average value of T_w for the dynamic and the static QCAP model were 606.25 and 388.24, respectively.



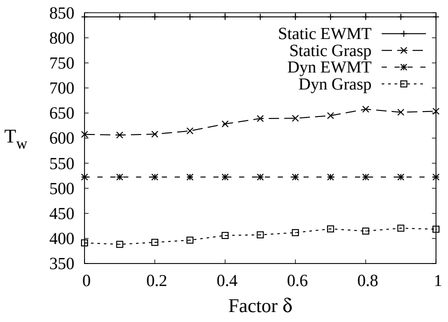
(a) Dense - Exponential inter-arrival times



(b) Sparse - Exponential inter-arrival times



(c) Dense - Poisson inter-arrival times



(d) Sparse - Poisson inter-arrival times

Figure 12: Weighted waiting time for 20 vessels in sparse and dense corpus with exponential and poisson inter-arrival distribution.

Table 3 shows the evolution of the rules with the static QCAP model (solutions that would be provided by terminal operators) against GRASP with the dynamic QCAP model varying the number of incoming vessels from 5 vessels up to 20 vessels with an exponential inter-arrival distribution both for the *Dens* and *Spar* corpus. For each number of incoming vessels, the GRASP metaheuristic outperformed the average results given by the rules. For instance, given the *Spar* corpus and 15 vessels, the best rule obtained 249.41 whereas GRASP achieved 136.75. It is important to note that GRASP decreases the objective function stronger with the *Dens* corpus (see Figure 13), since given the characteristics of the *Spar* corpus, the optimal solutions are close to the arrival order of the vessels.

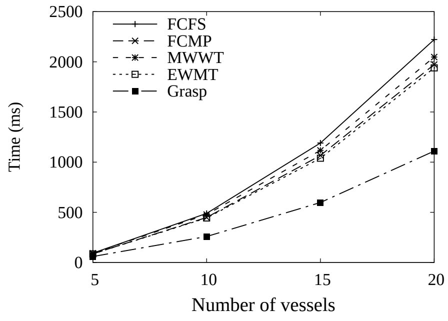
Table 3: Average T_w values for the rules and GRASP (Exponential)

(a) Dense inter-arrival times

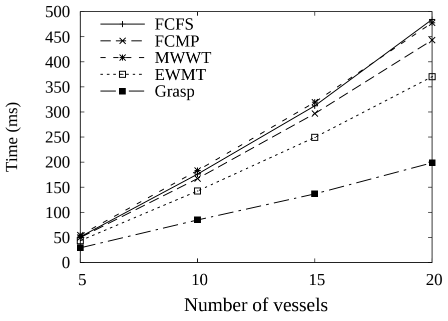
	5	**10**	**15**	**20**
FCFS	93.3	488.86	1190.86	2222.26
FCMP	85.66	448.55	1066.46	1973.21
MWWT	92.28	477.81	1117.59	2047.52
EWMT	85.67	443.51	1039.34	1939.64
GRASP	58.92	258.65	597.45	1110.21

(b) Sparse inter-arrival times

	5	**10**	**15**	**20**
FCFS	51.04	176.13	312.36	484.45
FCMP	49.65	167.29	296.87	443.26
MWWT	54.62	183.29	319.48	477.82
EWMT	43.34	142.54	249.41	370.41
GRASP	29.1	84.69	136.75	198.67



(a) Dense inter-arrival times



(b) Sparse inter-arrival times

Figure 13: Average T_w values for the rules and GRASP (Exponential)

Figure 14 shows the average computation times per iteration of the GRASP metaheuristic for the two models: static and dynamic QCAP. This experiment was performed for the exponential inter-arrival distribution both for the *Spar* (see Figure14(b)) and *Dens* (see Figure 14(a)) corpus. As the optimal solutions in the *Spar* corpus are close to the arrival order of the vessels, the average computation times for the *Spar* corpus are lower than the *Dens* corpus. Furthermore, the average time per iteration depends on the δ factor chosen since the size of the RCL is related to this parameter. Thereby, taking into account the dynamic QCAP model, this average time per iteration varied from 10.5 ms ($\delta = 0$) up to 30 ms ($\delta = 1$) for the *Spar* corpus; and, it varied from 23.2 ms ($\delta = 0$) up to 48.5 ms ($\delta = 1$) for the *Dens* corpus.

As mentioned in Sect. 3, the performance of container terminals is also evaluated

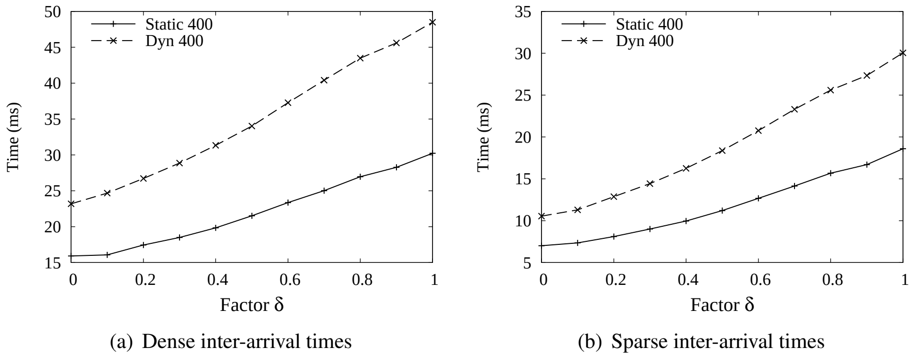


Figure 14: Average computation times for exponential inter-arrival distribution.

according to the berth usage (B_u) (see Table 4). Table 4(a) shows the relationship between the berth usage (B_u) and the weighted waiting time (T_w) for a queue of 20 incoming vessels from the *Dens* corpus with an exponential inter-arrival distribution. In this case, only the dynamic model is considered since it obtained the best results in the previous experiments. Note that the lower T_w is, the greater berth usage is. A value of 71.43% was achieved for $\delta = 0.2$. Furthermore, having evaluated the dynamic and static QCAP models (see Table 4(b)), the dynamic QCAP model always achieved a better berth usage of the quay, approx. 1.34% in average.

Table 4: B_u as a key factor in the container terminal.

(a) Relationship between B_u and T_w (dynamic QCAP model)

Factor δ	**0**	**0.2**	**0.4**	**0.6**	**0.8**	**1**
$B_u\%$	0.7136	0.7143	0.7064	0.7065	0.7042	0.7069
T_w	1116.97	1112.41	1127.21	1129.11	1134.88	1133.04

(b) Differences in B_u between Dynamic and Static QCAP models

Factor δ	**0**	**0.2**	**0.4**	**0.6**	**0.8**	**1**
Dynamic $B_u\%$	0.7136	0.7143	0.7064	0.7065	0.7042	0.7069
Static $B_u\%$	0.7009	0.7026	0.7047	0.6941	0.6962	0.6965

Another key factor studied is the QC throughput (T_{qc}). Table 5 shows that when holds are taken into account in the model (dynamic QCAP), T_{qc} is considerably improved for both *Dens* and *Spar* corpus. In other words, QCs spend less time to perform the same number of movements, e.g. with 20 vessels in Table 5(a), the T_{qc} was 482.27 in the static

QCAP model, whereas in the dynamic model was 406.68. Therefore, the dynamic QCAP model allows better use of the QCs, since they can be used in other vessels immediately.

Table 5: Average time that QCs are busy.

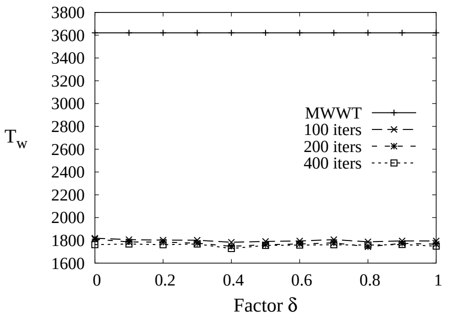
(a) Dense inter-arrival times

|V|	**Static QCAP**	**Dynamic QCAP**
5	122.06	101.05
10	240.36	202.02
15	359.98	303.08
20	482.27	406.68

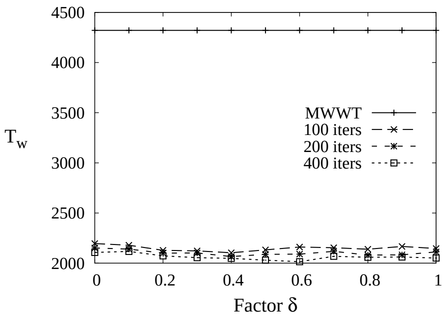
(b) Sparse inter-arrival times

|V|	**Static QCAP**	**Dynamic QCAP**
5	127.59	104.38
10	241.36	198.41
15	361.01	296.6
20	473.45	388.56

Finally, we remark that the GRASP metaheuristic search has also been applied to real data given by port operators from MSC where each instance consists of 15 incoming vessels. Figure 15 shows the average T_w values. For these experiments, the rule employed was *MWWT* since it obtained the best average results, and our GRASP method was able to reduce those T_w values in both models by approximately 53%. Comparing both models, the dynamic QCAP model reduced the T_w values by approximately 15.6% over the static model given the same δ factor ($\delta = 0.4$).



(a) Dynamic QCAP model



(b) Static QCAP model

Figure 15: T_w for the real data given by port operators.

7 Conclusions

We present a new process for allocating berth space for a number of vessels that uses the well-known GRASP metaheuristic. The developed method also adds the Quay Crane Assignment Problem into the model, taking into account the holds of each incoming vessel. The holds of the vessels are introduced in the dynamic QCAP model. The proposed GRASP metaheuristic has been compared to usual scheduling methods employed in container terminals (FCFS, FCMP, MWWT, EWMT). It can be observed how this metaheuristic reduces the waiting time and increases both the berth utilization and the throughput of QCs. These benefits are even greater when the dynamic QCAP model is employed since QCs are assigned in a more efficient way.

Due to the continuous increase of vessels traffic, our proposed metaheuristic could be employed since the difference between the GRASP-based method and the usual dispatching rules is becoming more and more significant. Therefore, allocation methods currently used in container terminals can be improved to a great extent by integrating metaheuristic approaches from areas of artificial intelligence.

BAP+QCAP solutions are executed in dynamic real-world environments where incidences can occur. Thus, an initial schedule might become invalid due to some incidences such as breakdowns in the QC engines, delays in the arrival of the vessels or deviations from the input data given by the shipping companies. Two main approaches are usually applied to manage these incidences: proactive and reactive [18]. The aim of a proactive approach is to obtain robust schedules that remain valid against incidences. A reactive approach gives rise to process of re-scheduling. These issues are interesting and open questions in real applications.

Acknowledgments

This work has been partially supported by the research projects TIN2010-20976-C02-01 (Ministerio de Ciencia e Innovación, Spain) the fellowship program FPU (AP2010-4405), and also with the collaboration of the maritime container terminal MSC (Mediterranean Shipping Company S.A.).

References

[1] D. Ayvaz, H.R. Topcuoglu, and F. Gurgen. Performance evaluation of evolutionary heuristics in dynamic environments. *Applied Intelligence*, 37(1):130–144, 2012.

[2] C. Bierwirth and F. Meisel. A survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research*, 202(3):615–627, 2010.

[3] C.Y. Cheong, K.C. Tan, and D.K. Liu. Solving the berth allocation problem with service priority via multi-objective optimization. In *Computational Intelligence in Scheduling, 2009. CI-Sched '09. IEEE Symposium on*, pages 95–102, 2009.

[4] M. Christiansen, K. Fagerholt, and D. Ronen. Ship routing and scheduling: Status and perspectives. *Transportation Science*, 38(1):1–18, 2004.

[5] Drewry Shipping Consultants. Global container terminal operators annual review and forecast. *Annual Report*, 2010.

[6] J.F. Cordeau, G. Laporte, P. Legato, and L. Moccia. Models and tabu search heuristics for the berth-allocation problem. *Transportation Science*, 39(4):526–538, 2005.

[7] C.F. Daganzo. The crane scheduling problem. *Transportation Research Part B: Methodological*, 23(3):159–175, 1989.

[8] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

[9] Paola Festa and Mauricio GC Resende. An annotated bibliography of grasp–part ii: Applications. *International Transactions in Operational Research*, 16(2):131–172, 2009.

[10] G. Giallombardo, L. Moccia, M. Salani, and I. Vacca. Modeling and solving the tactical berth allocation problem. *Transportation Research Part B: Methodological*, 44(2):232–245, 2010.

[11] Y. Guan and R.K. Cheung. The berth allocation problem: models and solution methods. *OR Spectrum*, 26(1):75–92, 2004.

[12] L. Henesey. Overview of Transshipment Operations and Simulation. In *MedTrade Conference, Malta, April*, pages 6–7, 2006.

[13] A. Imai, H.C. Chen, E. Nishimura, and S. Papadimitriou. The simultaneous berth and quay crane allocation problem. *Transportation Research Part E: Logistics and Transportation Review*, 44(5):900–920, 2008.

[14] A. Imai, K.I. Nagaiwa, and C.W. Tat. Efficient planning of berth allocation for container terminals in Asia. *Journal of Advanced Transportation*, 31(1):75–94, 1997.

[15] K. Kim and H.O. Günther. *Container terminals and cargo systems*. Springer, 2006.

[16] Kap Hwan Kim and Young-Man Park. A crane scheduling method for port container terminals. *European Journal of Operational Research*, 156(3):752–768, 2004.

[17] K. K. Lai and K. Shih. A study of container berth allocation. *Journal of Advanced Transportation*, 26(1):45–60, 1992.

[18] Olivier Lambrechts, Erik Demeulemeester, and Willy Herroelen. Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities. *Journal of scheduling*, 11(2):121–136, 2008.

[19] Der-Horng Lee, Jiang Hang Chen, and Jin Xin Cao. The continuous berth allocation problem: a greedy randomized adaptive search solution. *Transportation Research Part E: Logistics and Transportation Review*, 46(6):1017–1029, 2010.

[20] D.H. Lee, H.Q. Wang, and L. Miao. Quay crane scheduling with non-interference constraints in port container terminals. *Transportation Research Part E: Logistics and Transportation Review*, 44(1):124–135, 2008.

[21] C. Liang, Y. Huang, and Y. Yang. A quay crane dynamic scheduling problem by hybrid evolutionary algorithm for berth allocation planning. *Computers & Industrial Engineering*, 56(3):1021–1028, 2009.

[22] A. Lim. The berth planning problem. *Operations Research Letters*, 22(2-3):105–110, 1998.

[23] Jiyin Liu, Yat-wah Wan, and Lei Wang. Quay crane scheduling at container terminals to minimize the maximum relative tardiness of vessel departures. *Naval Research Logistics (NRL)*, 53(1):60–74, 2006.

[24] F. Meisel and C. Bierwirth. Heuristics for the integration of crane productivity in the berth allocation problem. *Transportation Research Part E: Logistics and Transportation Review*, 45(1):196–209, 2009.

[25] E.M. Mohi-Eldin and E.M. Mohamed. The impact of the financial crisis on container terminals (a global perspectives on market behavior). In *Proceedings of 26th International Conference for Seaports & Maritime Transport*, 2010.

[26] Y.M. Park and K.H. Kim. A scheduling method for berth and quay cranes. *OR Spectrum*, 25(1):1–23, 2003.

[27] R.I. Peterkofsky and C.F. Daganzo. A branch and bound solution method for the crane scheduling problem. *Transportation Research Part B: Methodological*, 24(3):159–172, 1990.

[28] M. Rodriguez-Molins, M.A. Salido, and F. Barber. Intelligent planning for allocating containers in maritime terminals. *Expert Systems with Applications*, 39(1):978–989, 2012.

[29] Mario Rodríguez-Molins, Miguel A. Salido, and Federico Barber. Domain-dependent planning heuristics for locating containers in maritime terminals. In *Proceedings of the 23rd International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems*, volume 6096 of *LNCS*, pages 742–751. 2010.

[30] M. Salido, O. Sapena, and F. Barber. An artificial intelligence planning tool for the container stacking problem. *Proceedings of the 14th IEEE International Conference on Emerging Technologies and Factory Automation*, pages 532–535, 2009.

[31] Miguel A. Salido, Mario Rodriguez-Molins, and Federico Barber. A decision support system for managing combinatorial problems in container terminals. *Knowledge-Based Systems*, 29:63–74, 2012.

[32] R. Stahlbock and S. Voß. Operations research at container terminals: a literature update. *OR Spectrum*, 30(1):1–52, 2008.

[33] D. Steenken, S. Voß, and R. Stahlbock. Container terminal operation and operations research-a classification and literature review. *OR Spectrum*, 26(1):3–49, 2004.

[34] Rafal Szlapczynski and Joanna Szlapczynska. On evolutionary computing in multi-ship trajectory planning. *Applied Intelligence*, 37:155–174, 2012.

[35] S. Theofanis, M. Boile, and M.M. Golias. Container terminal berth planning. *Transportation Research Record: Journal of the Transportation Research Board*, 2100:22–28, 2009.

[36] Fundació ValenciaPort. Automation and simulation methodologies for assessing and improving the capacity, performance and service level of port container terminals, 2009. Ministerio de Fomento (P19/08), Spain.

[37] I.F.A. Vis and R. De Koster. Transshipment of containers at a container terminal: an overview. *European Journal of Operational Research*, 147:1–16, 2003.

2.3 Planning and Scheduling: Integration Berthing and Container Stacking Problems

Integrated Intelligent Techniques for Remarshaling and Berthing in Maritime Terminals

Miguel A. Salido, Mario Rodriguez-Molins, Federico Barber
Instituto de Automática e Informática Industrial
Universidad Politécnica de Valencia.
Valencia, Spain

Abstract

Maritime container terminals are facilities where cargo containers are transshipped between ships or between ships and land vehicles (trucks or trains). These terminals involve a large number of complex and combinatorial problems. Two important problems are the Container Stacking Problem and the Berth Allocation Problem. Both problems are generally managed and solved independently but there exist a relationship that must be taken into account to optimize the whole process. The terminal operator normally demands all containers bound for an incoming vessel to be ready in the terminal before its arrival. Similarly, customers (i.e., vessel owners) expect prompt berthing of their vessels upon arrival. This is particularly important for vessels from priority customers who may have been guaranteed berth-on-arrival service in their contract with the terminal operator. To this end, both problems must be interrelated.

In this paper, a set of artificial intelligence based-techniques for solving both problems is presented. We develop a planning technique for solving the Container Stacking Problem and a set of optimized allocation algorithms for solving the Berth Allocation Problem independently. Finally we have developed an architecture to solve both problems in an integrated way. Thus, an algorithm for solving the berth allocation problem generates an optimized order of vessels to be served meanwhile our container stacking problem heuristics calculate the minimum number of reshuffles needed to allocate the containers in the appropriate place for the obtained ordering of vessels. Thus combined optimal solutions can be calculated and the terminal operator could decide which solution is more appropriate in each case. These techniques will minimize disruptions and facilitate planning in container terminals.

Keywords Planning, Heuristics, Optimization, Container Stacking Problem, Remarshaling, Berth allocation problem

1 Introduction

Container terminals have become an important component of global logistics networks. Henesey [11] shows how this transshipment market is growing fast (container throughput has increased by 58 per cent over 2000-2004) and needs further studies to analyze it. In order to ensure reliability, e.g. delivery dates or handling times, to the different shipping companies as well as increasing productivity and container throughput from the quayside and landside and vice versa, there are several issues which need optimization. Extensive surveys are provided [28, 26] about operations at seaport container terminals and methods for their optimization. Moreover, other problems could be faced as for instance planning the routes for liner shipping services to obtain the maximal profit [4]. Another important issue for the success at any container terminal is to forecast container throughput accurately [2]. With this data they could develop better operational strategies and investment plans.

In any case, the main objective in the container terminals is to reduce the berthing time of vessels. This global objective generates a set of interrelated problems divided into berth allocation, yard-side operation, storage operation and gatehouse operation. Generally, each problem is managed independently due to the fact that these problems are combinatorial problems so their complexities remain exponential. However, many of these problems are interrelated so an optimized solution of a problem could restring the possibility of obtaining a better solution in a related problem than before.

In this paper, we will focus our attention to the berth allocation problem and the container stacking problem (see Figure 1). Briefly, the berth allocation problem consists on allocating a given set of coming vessels to a berth under certain constraints such as priorities, length and depth of vessels, number of containers, etc. When a vessel berths, export containers to be loaded should be on top of the stacks in the container-yard. Therefore, the container stacking problem consists on rearranging the containers so that the yard crane does not need to do rehandling work at the time of loading. These two problems are related so that an optimal berth allocation plan may generate a large amount of relocations for export containers, meanwhile a suboptimal berth allocation plan could generate a small amount of relocations. The terminal operator could decide which solution is the most appropriate in every planning.

In this paper, we have developed a set of artificial intelligence based-techniques for solving both problems concurrently in order to achieve a solution that combines and optimizes both problems. To this end, we firstly present a planning technique for generating a rehandling-free intra-block remarshaling plan for a container yard. Then, we present a metaheuristic for solving the berth allocation problem as an independent problem. Finally, we present a integrated system for optimizing both problems. As we have pointed out before, the terminal operator will decide which solution is the most appropriate one in each stage due to the integrated problem will generate a multi-objective function: to minimize the waiting time of each vessel and to minimize the number of relocations.

The overall goal of collaboration between our group at the Technical University of Valencia (UPV) and the maritime container terminal MSC (Mediterranean Shipping Com-

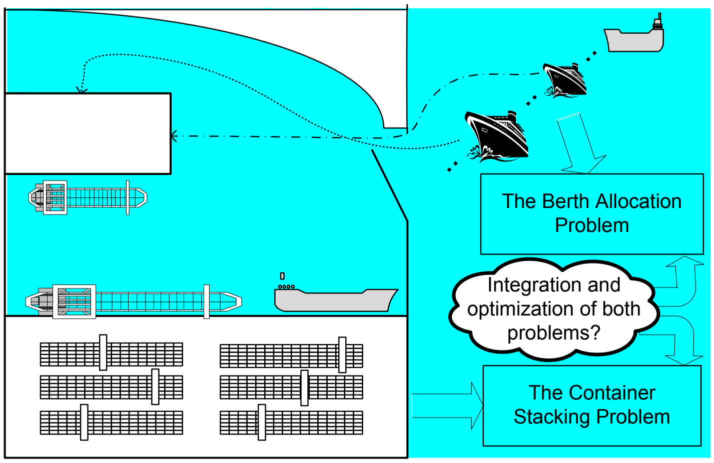

The Berth Allocation Problem
Integration and optimization of both problems?
The Container Stacking Problem

Figure 1: Integrated Remarshaling and Berthing problems in Maritime Terminals

pany S.A) is to offer assistance in planning and scheduling tasks, such as the allocation of spaces to outbound containers, the berthing allocation planning in order to identify bottlenecks, determine the consequences of changes, and to provide support in the resolution of incidents.

1.1 The Container Stacking Problem (CSP)

As we have pointed out, one of the performance measures in container terminals is the time spent by vessels in the port quays. This time is mainly composed of the container unloading/loading time. In order to reduce the loading time, it is necessary to maintain the out containers free of reshuffles by means of remarshaling tasks. This problem is known as Container Stacking Problem (CSP). It is a NP-complete and an intractable highly combinatorial optimization problem. Few studies are dealing with this problem. In [16], the authors proposed dynamic programming to attain an ideal configuration while minimizing the number of moved containers and the follow-on traveled distance.

Containers are an ISO standardized metal box and can be stacked on top of each other. The container capacity is often expressed in twenty-foot equivalent unit (TEU). Loading and offloading containers on the stack is performed by cranes following a 'last-in, first-out' (LIFO) storage.

In order to access a container which is not at the top of its pile, those above it must be relocated. It occurs since other ships have been unloaded later or containers have been stacked in the wrong order due to lack of accurate information. Contain relocation reduces the productivity of the cranes. Maximizing the efficiency of this process leads to several requirements:

1. Each incoming container should be allocated a place in the stack which should be

free and supported at the time of arrival.

2. Each outgoing container should be easily accessible, and preferably close to its unloading position, at the time of its departure.

In addition, there exist a set of hard/soft constraints regarding the container locations, for example, small differences in height of adjacent yard-bays, dangerous containers must be allocated separately by maintaining a minimum distance and so on.

Nowadays, the allocation of positions to containers is usually done manually. Therefore, using appropriate Artificial Intelligent techniques is possible to achieve significant improvements of lead times, storage utilization and throughput.

Figure 2 (a) shows a container yard. A yard consists of several blocks, and each block consists of 20-30 yard-bays [18]. Each yard-bay contains several (usually 6) rows. Each row has a maximum allowed tier (usually tier 4 or tier 5 for full containers). Figure 2 (b) shows a gantry crane that is able to move a container within a stacking area or to another location on the terminal. For safety reasons, it is usually prohibited to move the gantry crane while carrying a container [20], therefore these movements only take place in the same yard-bay.

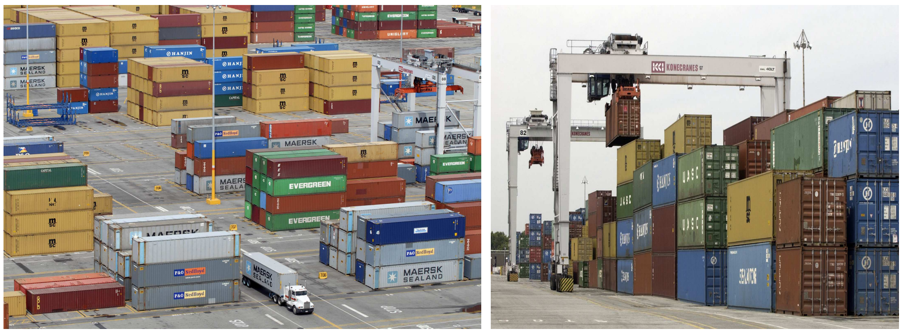


Figure 2: A container yard (a) and gantry cranes (b) (Photos by Stephen Berend)

When a container arrives at the terminal port, a transfer crane picks it up and stacks it in a yard-bay. During the ship loading operation, a transfer crane picks up the container and transfers it to a truck that delivers it to a quay crane.

In container terminals, the loading operation for export containers is pre-planned by load planners. For load planning, a containership agent usually transfers a load profile (an outline of a load plan) to a terminal operating company several days before a ship's arrival. The load profile specifies only the container group. In order to have an efficient load sequence, storage layout of export containers must have a good configuration.

One of the main focus of this paper is to present a planning system which optimally reallocates outgoing containers for the final storage layout from which a load planner can construct an efficient load sequence list. In this way, the objective of this problem is therefore to plan the movement of the cranes so as to minimize the number of reshuffles of containers in a complete yard. To this end, the yard is decomposed in yard-bays, so that the problem is distributed into a set of subproblems. Thus, each yard-bay generates a subproblem, but containers of different yard-bays must satisfy a set of common constraints among them, so that subproblems will be sequentially solved taken into account the common set of constraints.

In the literature, generally this problem can be seen in two different approaches according to when it should be done the optimization:

1. minimizing the number of relocations during the pickup operation.

2. getting a desirable layout for the bay before the pickup operation is done in order to minimize (or eliminate) the number of relocations during this process.

[15] proposes a methodology to estimate the expected number of rehandles to pick up an arbitrary container and the total number of rehandles to pick up all the containers in a bay for a given initial stacking configuration. In a similar way, [17] compares two methods, branch-and-bound algorithm and a heuristic rule based on an estimator, which they minimize the number of relocations during the pickup operation.

[16] also proposes a methodology to convert the current bay layout into the desirable layout by moving the fewest possible number of containers (remarshaling) and in the shortest possible travel distance although it takes a considerable time since they use mathematical programming techniques. Cooperative coevolutionary algorithms have been develop in [21] to obtain a plan for remarshaling in automated container terminals.

This paper focuses on the latter approach. But we present a new heuristic with a set of optimization criteria in order to achieve efficiency and take into account constraints that should be considered in real-world problems in the provided solutions.

1.2 The Berth Allocation Problem (BAP)

In [27], the authors show a complete comparison about different solutions for Berth Allocation Problem (BAP) according to their efficiency in addressing key operational and tactical questions relating to vessel service and their relevance and applicability to the different strategies and contractual service arrangements between terminal operator and shipping lines. They distinguish the models into the following 4 categories:

- *static BAP (SBAP)*, if all the vessels to be served are already in the port at the time that scheduling begins.

- *dynamic BAP (DBAP)*, if all the vessels to be scheduled for berthing have not yet arrived but their arrival times are known in advance.

- *discrete berthing space*, when the quay is viewed as a finite set of berths, in which each berth is described by fixed-length segments or points.

- *continuous berthing space*, in this case vessels can berth anywhere along the quay.

One of the early works that appeared in the literature was [19]. In this paper, they developed a heuristic algorithm by considering a First-Come-First-Served (FCFS) rule. However, the idea that for high port throughput, optimal vessel-to-berth assignments should be found without considering the FCFS bases was introduced by [14]. Therefore, we will use the FCFS rule in order to get an upper bound. But, this approach may result in some vessels' dissatisfaction regarding the order of service.

In [9] multiple vessel mooring per berth is allowed assuming that vessel arrivals can be grouped into batches. They have developed a tree search procedure which provides an exact solution and this is improved by a composite heuristic.

Some metaheuristics have been developed to solve the BAP. On the one hand, [5] introduce two Tabu Search heuristics to solve the discrete and continuous case, respectively to minimize is the weighted sum for every ship of the service time in the port. Both heuristics are inspired by a *Multi-Depot Vehicle Routing Problem with Time Windows* algorithm and can handle various features of real-life problems as time windows or favorite and acceptable berthing areas.

On the other hand, [3] follow an approach based on multi-objective optimization problem using evolutionary algorithms to minimize the makespan of the port, total waiting time of the ships, and degree of deviation from a predetermined service priority schedule.

[8] present the integration of BAP with the Quay Crane Assignment Problem (QCAP) through two mixed integer programming formulations including a tabu search method which is an adaption of the one of [5], however they minimize the yard-related housekeeping costs generated by the flows of containers exchanged between vessels. More information regarding new classification schemes for berth allocation problems and quay crane scheduling problems is presented in [1]. Particular focus of this paper is put on integrated solution approaches which receive increasing importance for the terminal management.

Our approach integrates the both problems (BAP and QCAP) through a metaheuristic method called Greedy Randomized Adaptive Search Procedure (GRASP) [6] which is able to find feasible solutions within an acceptable computational time.

2 An Approach for the Container Stacking Problem

The Container Stacking Problem can be viewed, from the artificial intelligence point of view, as a modification of the *Blocks World* planning domain [29], which is a well-known domain in the planning community. This domain consists of a finite number of blocks stacked into towers on a table large enough to hold them all. The *Blocks World* planning problem is to turn an initial state of the blocks into a goal state, by moving one block at a time from the top of a tower onto another tower (or on a table). The optimal *Blocks World* planning problem is to do so in a minimal number of moves.

Blocks World problem is closed to the Container Stacking Problem, but there are some important differences:

- The number of towers is limited to 6 because a yard-bay contains usually 6 rows.

- The height of a tower is also limited to 4 or 5 tiers depending on the employed cranes.
- There exist a set of constraints that involve different rows such as balanced adjacent rows, dangerous containers located in different rows, etc.
- The main difference is in the problem goal specification. In the *Blocks World* domain the goal is to get the blocks arranged in a certain layout, specifying the final position of each block. In the container stacking problem the goal state is not defined as accurately, so many different layouts can be a solution for a problem. The goal is that the most immediate containers to load are in the top of the towers, without indicating which containers must be in each tower.

We can model our problem by using the standard encoding language for classical planning tasks called *PDDL* (Planning Domain Definition Language) [7] whose purpose is to express the physical properties of the domain under consideration and it can be graphically represented by means of tools as [10]. A classical AI planning problem can be defined by a tuple $\langle A, I, G \rangle$, where A is a set of actions with preconditions and effects, I is the set of propositions in the initial state, and G is a set of propositions that hold true in any goal state. A solution plan to a problem in this form is a sequence of actions chosen from A that when applied transform the initial state I into a state of which G is a subset.

Following the *PDDL* standard, a planning task is defined by means of two text files. The **domain file**, which contains the common features for problems of this domain and the **problem file**, which describes the particular characteristics of each problem. These two files will be described in the following subsections.

2.1 Domain specification

In this file, we will specify the *objects* which may appear in the domain as well as the relations among them (*propositions*). Moreover, in order to make changes to the world state, *actions* must be defined.

- *Object types*: *containers* and *rows*, where the rows represent the areas in a yard-bay in which a tower or stack of containers can be built.
- *Types of propositions*:
  - Predicate for indicating that the container `?x` is on `?y`, which can be another container or, directly, the floor of a row (stack).

    ```
    on ?x - container ?y - (either row container)
    ```
  - Predicate for indicating that the container `?x` is in the tower built on the row `?r`.

    ```
    at ?x - container ?r - row
    ```

- Predicate for stating that `?x`, which can be a row or a container, is clear, that is, there are no containers stacked on it.

  `clear ?x - (either row container)`

- Predicate for indicating that the crane used to move the containers is not holding any container.

  `crane-empty`

- Predicate for stating that te crane is holding the container `?x`.

  `holding ?x - container`

- Predicates used to describe the problem goal. The first one specifies the most immediate containers to load, which must be located on the top of the towers to facilitate the ship loading operation. The second one becomes true when this goal is achieved for the given container.

  `goal-container ?x - container` and `ready ?x - container`

- Numerical predicates. The first one stores the number of containers stacked on a given row and the second one counts the number of container movements carried out in the plan.

  `height ?s - row` and `num-moves`

• *Actions*:

- The crane picks the container `?x` which is in the floor of row `?r`.

  `pick (?x - container ?r - row)`

- The crane puts the container `?x`, which is holding, in the floor of row `?r`.

  `put (?x - container ?r - row)`

- The crane unstacks the container `?x`, which is in row `?r`, from the container `?y`.

  `unstack (?x - container ?y - container ?r - row)`

- The crane stacks the container `?x`, which is currently holding, on container `?y` in the row `?r`.

  `stack (?x - container ?y - container ?r - row)`

- Finally, we have defined two additional actions that allow to check whether a given (goal) container is ready, that is, it is in a valid position. When a container is clear:

  `fict-check1 (?x - container)`

  The container is under another (goal) container which is in a valid position.

  `fict-check2 (?x - container ?y - container)`

As an example of *PDDL* format, we show in Figure 3 the specification of the stack operator. Preconditions describe the conditions that must hold to apply the action: crane must be holding container ?x, container ?y must be clear and at row ?r, and the number of containers in that row must be less than 4. With this constraint we limit the height of the piles. The effects describe the changes in the world after the execution of the action: container ?x becomes clear and stacked on ?y at row ?r, and the crane is not holding any container. Container ?y becomes not clear and the number of movements and the containers in ?r is increased in one unit.

```
(:action stack
 :parameters (?x - container ?y - container ?r - row)
 :precondition (and
    (holding ?x) (clear ?y)
    (at ?y ?r)   (< (height ?r) 4))
 :effect (and
    (clear ?x)   (on ?x ?y)
    (at ?x ?r)   (crane-empty)
    (not (holding ?x))
    (not (ready ?y))
    (not (clear ?y))
    (increase (num-moves) 1)
    (increase (height ?r) 1)))
```

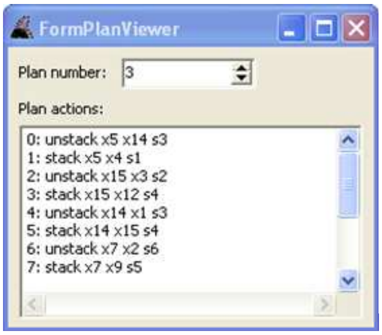
Figure 3: Formalization of the *stack* operator in *PDDL*.

2.2 Problem specification

In this file, we will specify the particular characteristics of each problem:

- *Objects*: the rows available in the yard-bay (usually 6) and the containers stored in them.
- *Initial state*: the initial layout of the containers in the yard.
- *The goal specification*: the selected containers to be allocated at the top of the stacks or under other selected containers.
- *The metric function*: the function to optimize. In our case, we want to minimize the number of relocation movements (reshuffles).

Since the Container Stacking Problem can be formalized with these two files (the domain specification and the problem specification), we can use a general domain independent planner to solve our problems as *Metric FF* [13]. The plan, which is returned by the planner, is a totally ordered sequence of actions or movements which must be carried out by the crane to achieve the objective. Figure 4 shows an example of the obtained plan for a given problem. The performance of this general planner will be analyzed in Section 5, which will be compared with the domain-oriented planner presented in next Sections. Once the problem domain has been defined, we can define problem instances.



```
0: unstack x5 x14 s3
1: stack x5 x4 s1
2: unstack x15 x3 s2
3: stack x15 x12 s4
4: unstack x14 x1 s3
5: stack x14 x15 s4
6: unstack x7 x2 s6
7: stack x7 x9 s5
```

Figure 4: The obtained plan solution to be carried out by the transfer crane.

2.3 A Domain-Dependent Heuristically Guided Planner

Metric FF planner might obtain plans, but it is very inefficient. Therefore, we propose a domain-dependent planner in order to provide more efficiency, it means at least reducing the number of crane operations required to achieve a desirable layout.

The proposed planner is built on the basis of a local search domain-independent planner called *Simplanner* [25]. Some characteristics of the tool can be seen in [22]. It has several interesting properties for the container stacking problem:

- It is an anytime planning algorithm. This means that the planner can found a first, probably suboptimal, solution quite rapidly and that this solution is being improved while time is available.
- It is complete, so it will always find a solution if exists.
- It is optimal, so that it guarantees finding the optimal plan if there is time enough for computation.

It follows an enforced hill-climbing [12] approach with some modifications:

- It applies a best-first search strategy to escape from plateaux. This search is guided by a combination of two heuristic functions and it allows the planner to escape from a local minima very efficiently.
- If a plateau exit node is found within a search limit imposed, the hill-climbing search is resumed from the exit node. Otherwise, a new local search iteration is started from the best open node.

Simplanner was firstly used to solve individual subproblems (yard-bays). To improve the solutions obtained, we have further developed a domain-dependent heuristic to guide the search in order to accelerate and guide the search toward a optimal or sub-optimal solutions.

A heuristic (called h_1) was developed to efficiently solve each individual subproblems. h_1 computes an estimator of the number of container movements that must be carried out to reach a goal state (see Algorithm 1). The essential part of this algorithm is to count the number of containers located on the selected ones, but also keeps track of the containers that are held by the crane distinguishing between whether they are selected containers or not. When the crane is holding a selected container, the value h has a smaller value since, although this state is not a solution, this container will be at the top of some row in the next movement.

Algorithm 1: Pseudo-code of the domain-dependent heuristic h_1

```
Data: $b$: state of the yard-bay;
Result: $h$: heuristic value of $b$;
1  $h = 0$;
2  Container hold by the crane if $\exists x-container/\texttt{Holding}(x) \in b$ then
3      if $\texttt{GoalContainer}(x)$ then
4          $h = 0.1$;
5      else
6          $h = 0.5$;
7      end
8  end
9  //Increasing the $\Delta h$ value for $r \leftarrow 1$ to $\texttt{numRows}(b)$ do
10     $\Delta h = 0$;
11     for $x-container/\texttt{At}(x, r) \wedge \texttt{GoalContainer}(x) \in b$ do
12         if $\nexists y-container/\texttt{GoalContainer}(y) \wedge \texttt{On}(y, x) \in b$ then
13             $\Delta h = \max(\Delta h, \texttt{NumContainersOn}(x))$;
14         end
15     end
16     $h+ = \Delta h$;
17 end
```

2.4 Optimization criteria for one-bay yards

Despite we are able to obtain good solutions (layouts) from *Simplanner* enhanced with h_1, we also want solutions more realistic for instance taking into account safety standards.

From this heuristic h_1, we have developed some optimization criteria each one of them achieving one of the requirements we could face at Container Terminals [24]. These criteria are centered in the issues as follows:

1. Reducing distance of the goal containers to the side (OC_{1d}) where the container will be loaded to a truck.

2. Increasing the range of the move actions set for the cranes allowing to move a container to 5th tier (OC_{1t}). Generally, the allowed number of tiers is limited to 4. In some cases, tier 5 is temporally allowed.

3. Applying different ways of balancing within the same bay in order to avoid *sinks* (OC_{1b}). It is considered that there is a *sink* when the height difference between two adjacent stacks in the same yard-bay is greater than a maximum number of containers, in our case two containers.

These criteria have been easily incorporated in our planner by defining an heuristic function as a linear combination of two functions:

$$h(s) = \omega_1 \times h_1(s) + \omega_2 \times h_2(s) \tag{1}$$

being this secondary function a combination of these three criteria described:

$$h_2(s) = OC_{1d} + OC_{1t} + OC_{1b} \tag{2}$$

Note that although we want to guarantee balancing with this last optimization criterion, unbalanced states (states with *sinks*) are allowed during this process of remarshaling in order to get better solutions according to the number of reshuffles done.

2.4.1 OC_{1d}: Placing goal containers close to cargo side

Given an initial state, several different layouts can be usually achieved making the same number of reshuffles and some of them can be more interesting than the rest according to other important questions. In this case, since the transfer crane is located at the right side of the yard-bay, we want to obtain a layout where it is minimized the distance of the goal containers to this side of the yard-bay. Achieving this we can spend considerably less time during the truck loading operations.

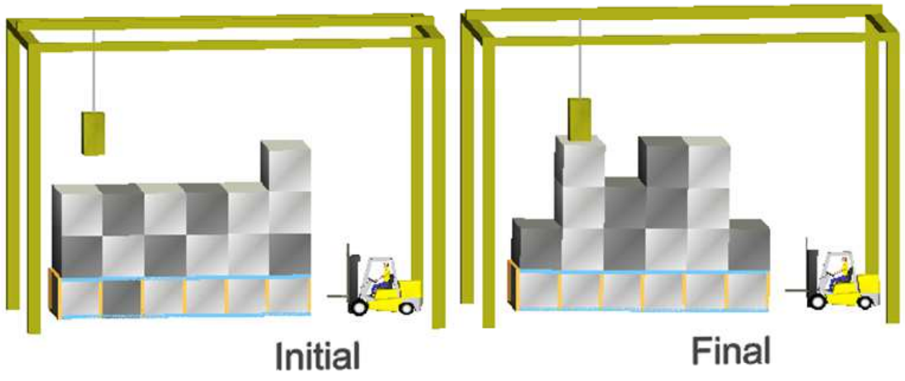


Figure 5: Obtained plan with the initial domain-dependent heuristic.

Following the heuristic function presented in Equation 1:

- $h_1(s)$ is the main heuristic function, which estimates the number of movements required to reach the goal layout (outlined in Algorithm 1). Since this is the main optimization function, α value should be significantly higher than β.

- $h_2(s)$ is the secondary function we want to optimize. In this case, it is just OC_{1d}. This means the sum of the distances of the selected containers to the right side of the yard-bay, which can be computed as Algorithm 2 shows.

Algorithm 2: Pseudo-code to calculate the distance

```
   Data: $s$: state to evaluate
   Result: $d$: distance value of $s$
1  $d = 0$;
2  for $r \leftarrow 1$ to $\texttt{numRows}(s)$ do
3      for $x$−container/$\texttt{At}(x, r) \in s \wedge \texttt{GoalContainer}(x)$ do
4          $d = d + (\texttt{numRows}(s) - r)$;
5      end
6  end
```

The benefits of using this combined heuristic function can be observed in Figure 5 and Figure 6. In the first one we want only to minimize the number of reshuffles, i.e. $h(s) = h_1(s)$. In the second one, we also want to minimize the distance of the selected containers to the forklift truck, so we have set $h(s) = 9 \times h_1(s) + h_2(s)$. As a result, none of the selected containers (the dark ones) are placed in the most left rows, reducing the required time to load the truck.

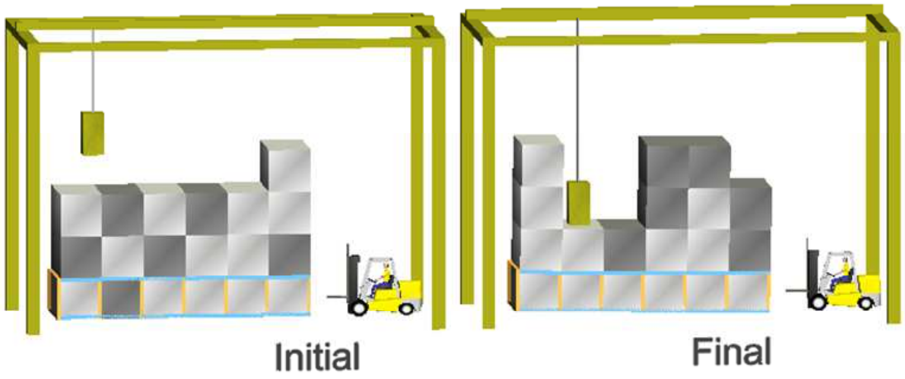


Figure 6: Obtained plan with the distance optimization function.

2.4.2 OC_{1t}: Allowing the 5th tier during the remarshaling process

In this optimization criterion as well as the next ones, we will include the new given heuristic value with the same factor as the initial one. One of the decisions that must be done in Container Terminals is about which cranes have to be bought depending on how many tiers cranes work. This topic has been considered in [23]. But, another approach is to reach the fifth tier only during the remarshaling process. Thereby, there would be 4 tiers at the beginning and the end keeping the first requirements.

Following this concept, we will use instances of problems $< n, 4 >$ with a domain whose move actions allow 5 tiers at the stacks. This function is showed in Algorithm 3 and it follows the same steps than the original but increasing the value of h when the height of one of the stacks is higher than 4. Thereby, we assure that the final layout will always have 4 tiers.

Algorithm 3: Pseudo-code of the domain-dependent heuristic function to allow 5 tiers

```
Data: $s$: state to evaluate
Result: $h$: heuristic value of $s$
h = 0;
if $\exists x-container/\texttt{Holding}(x) \in s$ then
    if $\texttt{GoalContainer}(x)$ then
        $h = 0.1$;
    else
        $h = 0.5$;
    end
end
for $r \leftarrow 1$ to $\texttt{numRows}(s)$ do
    $\Delta h = 0$;
    if $\texttt{Height}[r, s] > 4$ then
        if $x-container/\texttt{Clear}(x, r) \in s \wedge \texttt{GoalContainer}(x)$ then
            $\Delta h = 0.5$;
        else
            $\Delta h = 1$;
        end
    end
    for $x-container/\texttt{At}(x, r) \in s \wedge \texttt{GoalContainer}(x)$ do
        if $\nexists y-container/\texttt{GoalContainer}(y) \wedge \texttt{On}(y, x) \in s$ then
            $\Delta h = \max(\Delta h, \texttt{NumContainersOn}(x))$;
        end
    end
    $h+ = \Delta h$;
end
```

2.4.3 OC_{1b}: Balancing one yard-bay

In this section we present an extension for the heuristic h_1 (Algorithm 1) to include the balancing of the stacks within one yard-bay as a requirement.

Considering the time when the goal containers are removed from the yard, we can distinguish three ways to get balanced one yard-bay presented in the next subsections. The last mode is the consequence of applying the first two ones.

1. **Balanced Before loading operation** In this case we consider that the *layout must be balanced before the goal containers are removed from that yard-bay*. This function is showed in Algorithm 4, it compares the height of each row of the yard-bay with the next one, and if the difference is higher than 2, the value heuristic h is increased. As it appears in Figure 7, this criterion avoids the *sinks* in the final layout while all the containers are still in the yard-bay.

However, when these containers are removed, it might cause that the new layout is unbalanced as it happens in Figure 7(b).

Algorithm 4: Pseudo-code to balance before the goal containers are removed

```
Data: $s$: state to evaluate; $h$: Initial heuristic;
Result: $h$: heuristic value of $s$;
1 for $r \leftarrow 1$ to numRows$(s) - 1$ do
2     $\Delta h =$ Abs(Height$[r, s]$ − Height$[r + 1, s]$);
3     if $\Delta h > 2$ then
4         $h = h + \Delta h - 2$;
5     end
6 end
```

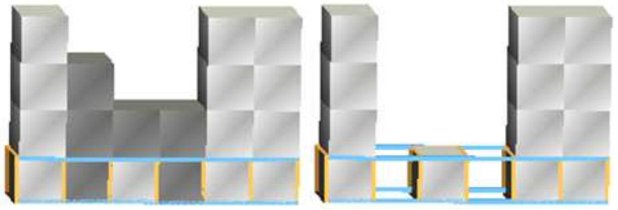

(a) With goal containers (b) Without goal containers

Figure 7: Effects of using function seen in Algorithm 4

2. **Balanced after loading operation** In contrast to the method seen above, we consider that the *layout must remain balanced after the goal containers are removed from the yard-bay*. Figure 8 shows the layouts we get after execute the plan returned by our planner.

Algorithm 5: Function `HeightsWithoutGoals` to calculate heights of each row without taking into account the goal containers at the top

```
Data: $b$: state of the yard-bay;
Result: MinHeight, heights calculated;
1 for $r \leftarrow 1$ to numRows$(b)$ do
2     MinHeight$[r, b]$ = Height$[r, b]$;
3     //Decrease till the first no goal-container
4     while MinHeight$[r, b] > 0 \wedge$ GoalContainer(MinHeight$[r, b], r) \in b$ do
5         MinHeight$[r, b]$ − −;
6     end
7 end
```

Algorithm 6 shows this function. It uses the Function *HeightsWithoutGoals* (Algorithm 5) in order to calculate for the yard-bay b the height for each stack where the first no-goal container is. These values are employed to get the difference of height between two adjacent stacks once the goal containers have been removed from the yard. Heights of each row are stored as soon as the planner gets the final solution plan for one yard-bay. After we obtain these values, we increase the heuristic value h according to whether or not there are goal containers on the floor. Then, we use the values given by *HeightsWithoutGoals* to calculate the difference between two adjacent stacks, when this difference is higher than 2 we consider that there is a *sink*, so h is increased again.

However, this process might also cause some unbalanced layouts (Figure 8(a)). But in this case, non-desirable layouts will appear while the goal containers are in the yard-bay. Once they have been removed from it, these layouts will be balanced ones (Figure 8(b)).

Algorithm 6: Pseudo-code to balance after the goal containers are removed

```
Data: $s$: state to evaluate; $h$: Initial heuristic;
Result: $h$: heuristic value of $s$;
HeightsWithoutGoals($s$);
$\Delta h = 0$;
//Not allow containers on the floor
for $r \leftarrow 1$ to numRows($s$) do
    if $\exists x$-container/On($x, r$) $\wedge$ GoalContainer($x$) then
        $\Delta h = \Delta h +$ NumContainersOn($x$);
    end
end
$h = h + \Delta h$;
for $r \leftarrow 1$ to numRows($s$) $- 1$ do
    $\Delta h =$ Abs(MinHeight[$r, s$] $-$ MinHeight[$r + 1, s$]);
    if $\Delta h > 2$ then
        $h = h + \Delta h - 2$;
    end
end
```

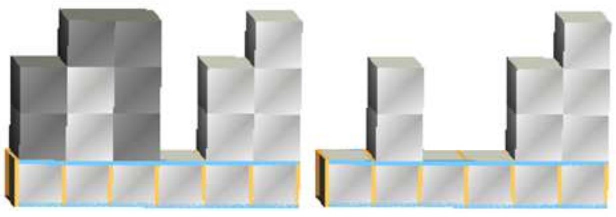

(a) With goal containers (b) Without goal containers

Figure 8: Effects of using function seen in Algorithm 6

3. **Balanced before and after the loading operation** Finally, we present an optimization criterion which obtains a *layout where is balanced both before and after the goal containers are removed from this yard-bay*. With this function we want to solve the problems seen in the last subsections as we can see it in Figure 9.

   This function (Algorithm 7) is a mixture of the last two ones. First, we increase h when there are goal containers on the floor. When this is achieved, we increase h when the difference between the heights values obtained by the function

HeightsWithoutGoals (Algorithm 5) are higher than 2 for two contiguous rows. And finally, if h value is low enough (in our case lower than 1), we increase h again if the difference between the actual heights of two contiguous rows is higher than 2.

Algorithm 7: Pseudo-code to balance the yard-bay before and after the goal containers are removed

```
Data: $s$: state to evaluate; $h$: Initial heuristic;
Result: $h$: heuristic value of $s$;
HeightsWithoutGoals($s$);
$\Delta h = 0$;
//Not allow containers on the floor
for $r \leftarrow 1$ to numRows($s$) do
    if $\exists x$−container/On($x, r$) $\wedge$ GoalContainer($x$) then
        $\Delta h = \Delta h +$ NumContainersOn($x$);
    end
end
$h = h + \Delta h$;
if $h == 0$ then
    $\Delta h = 0$;
    //Balancing with containers which are not objective
    for $r \leftarrow 1$ to numRows($s$) $- 1$ do
        $\Delta h =$ Abs(MinHeight[$r, s$] $-$ MinHeight[$r + 1, s$]);
        if $\Delta h > 2$ then
            $h = h + (\Delta h - 2)/2$;
        end
    end
    if $h < 1$ then
        //Balancing with containers which are objective
        for $r \leftarrow 1$ to numRows($s$) $- 1$ do
            $\Delta h =$ Abs(Height[$r, s$] $-$ Height[$r + 1, s$]);
            if $\Delta h > 2$ then
                $h = h + (\Delta h - 2)/2$;
            end
        end
    end
end
```

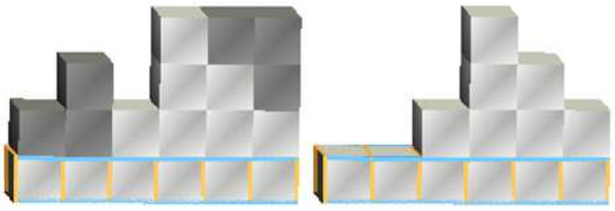

(a) With goal containers (b) Without goal containers

Figure 9: Effects of using function seen in Algorithm 7

2.5 Optimization criteria for one block

This initial heuristic (h_1) was unable to solve a complete yard or block (in our case, one block consists of 20 yard-bays) due to the fact that they only solve individual yard-bays. In this paper, we also have developed two optimization criteria that include new constraints that involve several yard-bays. These constraints are:

- Balancing contiguous yard-bays: rows of adjacent yard-bays must be balanced, that is, the difference between the number of containers of row j in yard-bay i and row j in yard-bay $i - 1$ must be lower than a maximum (in our case lower than 3). Figure 10 shows which rows must be get balanced when we consider one yard-bay and Figure 11 (a) shows an example of non-balanced yard-bays (rows in dotted points).

- Dangerous containers: two dangerous containers must maintain a minimum security distance. Figure 11 (b) shows an example of two dangerous containers that does not satisfy the security distance constraint.

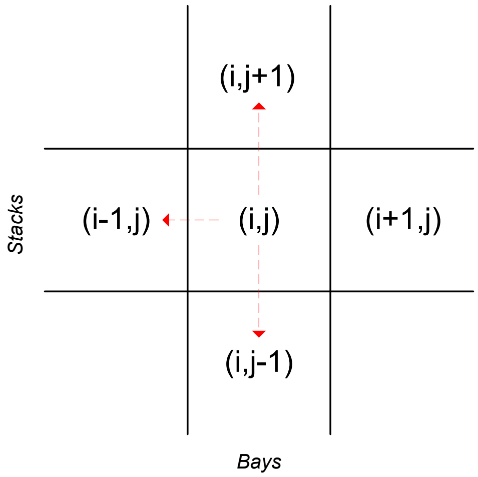


Figure 10: Bay and stack numbering

These constraints interrelate the yard-bays so the problem must be solved as a complete problem. However, it is a combinatorial problem and it is not possible to find an optimal or sub-optimal solution in a reasonable time. To this end, we can distribute the problem into subproblems and solve them sequentially taken into account related yard-bays. Thus a solution to the first yard-bay is taken into account to solve the second yard-bay. A solution to the second yard-bay is taken into account to solve the third yard-bay. Furthermore, if there exist a dangerous container in a first bay, its location is taken into account to solve a dangerous container located in the third yard-bay (if it exists); and so on. Taken into account this distributed and synchronous model, we present two different optimization criteria to manage these types of constraints.

These two criteria are added to the heuristic function seen in Equation 1 as h_3 (Equation 3); and Equation 4 shows the exact combination of them. This makes possible to follow a criterion with major priority than the other one.

$$h = \omega_1 \times h_1 + \omega_2 \times h_2 + \omega_3 \times h_3 \tag{3}$$

$$h_3 = \delta_1 \times OC_{nB} + \delta_2 \times OC_{nD} \tag{4}$$

As a consequence of the solving mode followed, depending on the order the yard-bays are resolved may not be possible to achieve a solution. Moreover, as mentioned in Section 2.4, although we want to guarantee balancing and/or minimum distance between dangerous containers, during relocation of container process we will allow the presence of non-desirable sates, e.g. with some *sinks* between two contiguous rows or bays. These intermediate states are allowed because through them we will be able to get better solutions taking into account as metric function the number of reshuffles done.

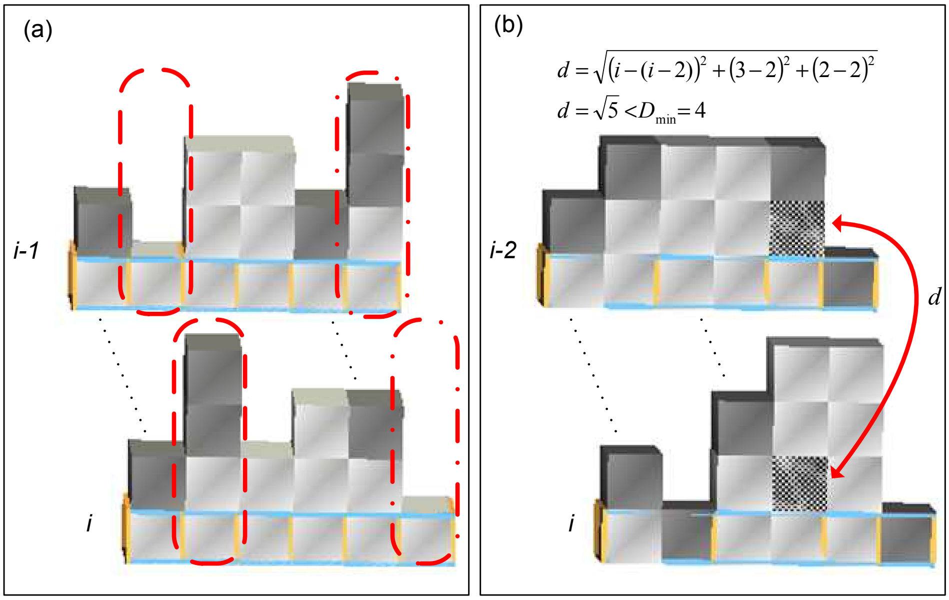


Figure 11: (a) Non-balanced yard-bays. (b) Proximity of two dangerous containers.

2.5.1 OC_{nB}: Balancing contiguous yard-bays

In this section we present an extension for the heuristic h_1 (Algorithm 1) to include the balancing of continuous yard-bays as a requirement. It is considered that there is a *sink* when a difference higher than two containers exists between two adjacent rows in contiguous yard-bays. This criterion is an extension of the *balanced heuristic* presented in Algorithm 7, which avoids *sinks* in the same yard-bay (horizontal balance) both before and after the outbound containers have been removed from the yard. However, in this case a *sink* represents a constraint between two subproblems. Thus, we also consider that there is a *sink* when a difference of two exits between the same row r in two contiguous yard-bays (vertical balance).

This process is showed in Algorithm 8. This also uses the Function *HeightsWithoutGoals* (Algorithm 5) in order to calculate for the yard-bay b the height for each stack where the first no-goal container is. Heights of each row are stored as soon as the planner gets the final solution plan for one yard-bay.

First, we apply the criterion seen in Algorithm 7 on the yard-bay b. Through `heights'` calculated by Algorithm 5 and the real heights of the actual yard-bay we obtain the differences between the row r and $r-1$ to calculate the value of h. When this value is zero (the yard-bay b is horizontally balanced), then we introduce our function to balance it with respect to the last yard-bay b_l. To do so, we must also calculate the `heights'` through the Algorithm 5 over b_l and use the real heights of it in order to obtain the differences between the row r situated in b and b_l. When these differences are higher than 2, we increase h proportionally. After that process, b will be balanced horizontally with respect to their rows, and vertically with respect to the last yard-bay. Repeating this process for each yard-bay in the block, this will be completely balanced.

Algorithm 8: Pseudo-code to balance two adjacent *yard-bays*

```
Data: $b$: state of the actual yard-bay; $h$: Initial heuristic; $b_l$: last yard-bay;
Result: $h$: heuristic value of $b$
1  //Getting the balance horizontally HeightsWithoutGoals($b$);
2  $h += $ BalBeforeAfter($b$);
3  //This heuristic will be executed after a partial solution
4  if $h == 0 \wedge b \neq 1$ then
5      $\Delta h = 0$;
6      HeightsWithoutGoals($b_l$);
7      //Balancing with containers which are not objective
8      for $r \leftarrow 1$ to numRows($b$) do
9          $\Delta h =$ Abs(MinHeight[$r, b_l$] $-$ MinHeight[$r, b$]);
10         if $\Delta h > 2$ then
11             $h = h + (\Delta h - 2)/2$;
12         end
13     end
14     if $h < 1$ then
15         //Balancing with containers which are objective
16         for $r \leftarrow 1$ to numRows($b$) do
17             $\Delta h =$ Abs(Height[$r, b_l$] $-$ Height[$r, b$]);
18             if $\Delta h > 2$ then
19                 $h = h + (\Delta h - 2)/2$;
20             end
21         end
22     end
23 end
```

2.5.2 OC_{nD}: Dangerous containers

Within a block, there are different types of containers depending on the goods they transport, being some of them dangerous. If they do not satisfy certain restrictions, it may become a hazard situation for the yard since e.g. if one of them explodes and they are not enough far between them, it will set off a chain of explosions.

With this added objective, the next optimization criterion (Algorithm 9) ensures a minimum distance (D_{min}) between every two dangerous containers (C_d) in the yard. D_{min} is set as one parameter for the planner and the distance is calculated as the Euclidean distance, considering each container located in a 3-dimensional space (X,Y,Z) where X is the number of yard-bays, Y is the number of rows and Z is the tier.

Generally, in container terminals, at most, there is only one dangerous container in two contiguous yard-bays, so that we take into account this assumption in the development of this function.

This function increases h value when a dangerous container C_{d1} exists in a yard-bay b and the distance constraints between dangerous containers are not hold. Thereby, for each dangerous container C_{d2} allocated in the previous D_{min} yard-bays is calculated by Euclidean distance to C_{d1}. If this distance is lower than D_{min}, for any dangerous container C_{d2}, then h value is increased with the number of containers n on C_{d1} because it indicates that removing those n containers is necessary to reallocate the container C_{d1}.

Algorithm 9: Pseudo-code to avoid locating two dangerous containers closer to a distance D_{min}

```
Data: $B$: whole *block*; $b$: state of the actual *yard-bay*; $h$: Initial heuristic; $D_{min}$: Minimum distance; $NC$:
    Number of containers;
Result: $h$: heuristic value of $b$;
$nBay = \texttt{NumBay}(b)$;
if $nBay > 1 \wedge h < NC \wedge \exists C_{d1} \in b$ then
    $\Delta h = 0$;
    $L_1 = \texttt{Location}(C_{d1})$;
    foreach $b_l \in Y/\texttt{NumBay}(b_l) \in \{\max(nBay - D_{min} + 1, 1), nBay - 1\}$ do
        if $\exists C_{d2} \in b_l$ then
            $L_2 = \texttt{Location}(C_{d2})$;
            $dist = \texttt{EuclideanDistance}(L_1, L_2)$;
            if $dist < D_{min}$ then
                $\Delta h = \Delta h + \texttt{NumContainersOn}(C_{d1})$;
            end
        end
    end
    $h+ = \Delta h$;
end
```

Algorithm 10: Sinks within a whole block

```
Data: $B$: whole block;
Result: $nSinks$: number of Sinks;
$nSinks = 0$;
for $b \leftarrow 1$ to $\texttt{numYards}(B)$ do
    for $r \leftarrow 1$ to $\texttt{numRows}(b) - 1$ do
        $\Delta h = \texttt{Abs}(\texttt{Height}[r, b] - \texttt{Height}[r + 1, b])$;
        if $\Delta h > 2$ then
            $nSinks + +$;
        end
    end
    if $\texttt{NumBay}(b) > 1$ then
        for $r \leftarrow 1$ to $\texttt{numRows}(b)$ do
            $\Delta h = \texttt{Abs}(\texttt{Height}[r, b] - \texttt{Height}[r, b - 1])$;
            if $\Delta h > 2$ then
                $nSinks + +$;
            end
        end
    end
end
```

Algorithm 11: Unfeasible relationships between two dangerous containers within a whole block

```
Data: $B$: whole block;
Result: $nDang$: number of Sinks;
$nDang = 0$;
for $b \leftarrow 1$ to $\texttt{numYards}(B)$ do
    $nBay = \texttt{NumBay}(b)$;
    if $nBay > 1 \wedge \exists C_{d1} \in b$ then
        $L_1 = \texttt{Location}(C_{d1})$;
        foreach $b_l \in Y/\texttt{NumBay}(b_l) \in \{\max(nBay - D_{min} + 1, 1), nBay - 1\}$ do
            if $\exists C_{d2} \in b_l$ then
                $L_2 = \texttt{Location}(C_{d2})$;
                $dist = \texttt{EuclideanDistance}(L_1, L_2)$;
                if $dist < D_{min}$ then
                    $nDang + +$;
                end
            end
        end
    end
end
```

3 A Metaheuristic Approach for The Berth Allocation Problem

The berth allocation problem is one of the most relevant problems arising in the management of container ports. The objective is to obtain an optimal distribution of the docks and cranes to vessels waiting to berth. Thus, this problem could be considered as a special kind of machine scheduling problem, with specific constrains (length and depth of vessels, ensure a correct order for vessels going to exchange containers, assuring departing times, etc.) and optimization criteria (priorities, minimization of waiting and staying time, satisfaction on order of berthing, minimizing cranes moves, etc.).

Following, we introduce the notation that we use through this paper:

- $a(V_i)$: Arrival time of the vessel at harbor.
- $m(V_i)$: Moored time of the vessel. All the constraints must be accomplished.
- $d(V_i)$: Departure time of this vessel, depending on:
  - $c(V_i)$: Number of movements to load and unload containers.
  - $q(V_i)$: Number of assigned QC.
- $w(V_i)$: Waiting time of the vessel from it arrives at harbor until it can moor:

$$w(V_i) = m(W_i) - a(W_i) \tag{5}$$

- $l(V_i)$: Length of the vessel.
- $dr(V_i)$: Draft needed at berth for this vessel.
- $pr(V_i)$: Vessels' priority.

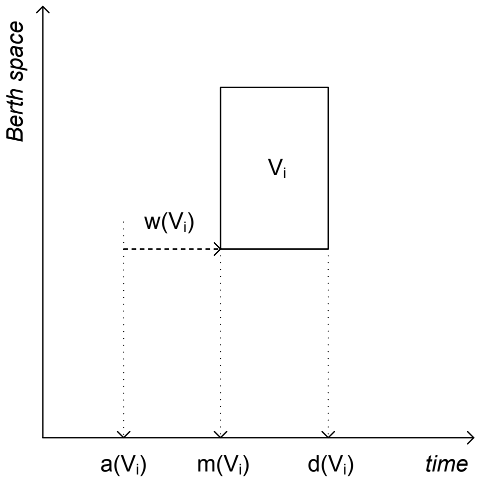


Figure 12: Representation of a vessel according its position and times

Basically, BAP is to find a good solution to allocate each vessel according to several constraints. Let's assume a priority of each vessel in order to avoid the vessels' dissatisfaction mentioned above, as:

$$pr(V_i) = \omega_l \times l(V_i) + \omega_c \times c(V_i) \tag{6}$$

In this paper, BAP is modeled as a dynamic BAP (DBAP) and time is discretized into integer units $(1, 2, \ldots, T)$. Moreover, we relax the problem by means of these assumptions:

- Number of quay cranes (QC) assigned to a vessel do not vary along all the moored time. Moreover, all QC do the same number of movements by unit time (`movsQC`).
- All the information related to the waiting vessels is known in advance.
- Every vessel has a draft lower or equal than the quay.
- Mooring and exiting is no consuming time.
- Each vessel has a priority according to its length and the number of movements (loading and unloading operations).

Therefore, in order to allocate one vessel at berth, the following constraints must be accomplished:

- Moored time must be at least the same that its arrival, therefore $m(V_i) \geq a(V_i)$.
- There is enough contiguous space at berth to moor the vessel ($l(V_i)$).
- There is a distance security between two moored ships. This distance is a percentage (in our case it is considered 5%) of its length (the maximum of these two contiguous ships).
- There is at least one QC to assign to each vessel.
- The maximum number of assigned QC by vessels depends on the length of it, since each QC needs a security distance (`seqQC`) to work. At most, $\texttt{maxQC}_{\texttt{V}}$ QC can be assigned to one vessel, in our case it is four.
- The handling time of one vessels is given by:

$$\frac{c(V_i)}{q(V_i) \times \texttt{movsQC}} \tag{7}$$

The goal of the allocation process for vessels berthing is to minimize the total weighted waiting time of vessels. That is:

$$T_w = \sum_i w(V_i) \times pr(V_i) \tag{8}$$

In order to prevent lower priority vessels are systematically delayed, it is necessary to introduce an adjustment factor γ $(1 < \gamma)$:

$$T_w = \sum_i w(V_i)^\gamma \times pr(V_i) \tag{9}$$

Note that this objective function is different to the tardiness concept in scheduling. The weighted optimization of tardiness of vessels would be:

$$T_{tard} = \sum_i w(V_i) \times (d(V_i) - dueTime(V_i)) \tag{10}$$

So that the departure time of vessels $d(V_i)$ with respect to their due times $dueTime(V_i)$ is optimized.

3.1 Strictly arrival order

Firstly, we can apply the simplest solution employed by operators to allocate vessels. This consists into inserting each ongoing vessel in a way 'First-Come-First-Served' (FCFS). It means each vessel V_i (i specifies the arrival order of that vessel) only could moor right after the entrance of the vessel V_{i-1}, such as $m(V_i) > m(V_{i-1})$ (Algorithm 12).

Algorithm 12: Allocating vessels following FCFS policy

```
Data: $V$: set of ordered incoming vessels; $b$: state of the berth
Result: Sequence for $V$
$V_{last} \leftarrow \varnothing$;
$V_m \leftarrow \varnothing$;
foreach $V_i \in V$ do
    $t \leftarrow \max(e(V_{last}), a(V_i))$;
    $inst \leftarrow insertVessel(V_i, t, b)$;
    if $!inst$ then
        $T \leftarrow d(V_j) | V_j \in V_m \wedge d(V_j) > t$;
        while $t_k \in T \wedge !inst$ do
            $inst \leftarrow insertVessel(V_i, t_k, b)$;
        end
    end
    $update(b)$ ;                                   /* state of the berth $b$ */
    $V_last \leftarrow V_i$;
    $V_m \leftarrow V_m \cup V_i$;
end
```

One vessel can be allocated directly at time t when there is no vessel moored in the berth. However, when there exists one vessel inside, it is likely that this vessel cannot moor. Two constraints must be accomplished in order to one vessel V_i might moor (Algorithm 13):

1. There must be available length and crane enough for V_i at time t.
2. During the whole stay of V_i (until $d(V_i)$) these resources must be available.

Therefore, available length and cranes are checked at time t. If there are enough resources, then departure time ($d(V_i)$) is calculated using the maximum possible number of cranes. Finally, it is checked that these resources are available from $m(V_i)$ until $(d(V_i)$ because it is likely that there were already some vessels scheduled after time t. If they are not guaranteed, then this process is repeated each time t', one scheduled vessel exists $d(V_j)$.

3.2 A Complete Algorithm

In this case, we use a complete algorithm for finding the best combination of the vessels which give us the lower total waiting time (T_w). We also follow the Algorithm 13 to know whether or not one vessel could be moored. However, now we do not take into account the constraint mentioned in the last method about one vessel could moor right after the previous vessel. In order to achieve that objective we do a search through a

Algorithm 13: Function insertVessel. Allocating one vessel in the berth at time t

```
Data: $V_i$: Vessel for allocating; $t$: actual time; $b$: state of the berth at time $t$;
Result: $V_i$ could moor
1  if $empty(b)$ then
2      $m(V_i) \leftarrow a(V_i)$;
3      $q(V_i) \leftarrow \min(\texttt{maxQC}_\texttt{V}, l(V_i)/\texttt{secQC})$;
4      $d(V_i) \leftarrow m(V_i) + \frac{c(V_i)}{q(V_i) \times \texttt{movsQC}}$;
5      return true;
6  else
7      $\texttt{freeQC} \leftarrow QC(b) - \sum q(V_i) | t \geq a(V_i) \wedge t < d(V_i)$;
8      $\texttt{freeL} \leftarrow l(b) - \sum l(V_i) | t \geq a(V_i) \wedge t < d(V_i)$;
9      if $\texttt{freeQC} > 0 \wedge l(V_i) <= \texttt{freeL}$ then
10         $q(V_i) \leftarrow \min(\texttt{freeQC}, l(V_i)/\texttt{secQC})$;
11         $m(V_i) \leftarrow t$;
12         $d(V_i) \leftarrow t + \frac{c(V_i)}{q(V_i) \times \texttt{movsQC}}$;
13         if checkDisponibility($V_i$,$m(V_i)$,$d(V_i)$) then
14             return true;
15         else
16             return false;
17         end
18     else
19         return false;
20     end
21 end
```

Branch & Bound method to explore the complete space of solutions, thereby if a new vessel is allocated and T_w is higher than the best one found earlier we do not continue examining this branch.

With this method we want to alter the arrival order established to get the best solution according to the T_w. As this function includes the vessels' priority ($pr(V_i)$), we also take into account the relevance of each vessel to overtake other vessels.

3.3 A Metaheuristic Allocation Method

With a complete search for this problem, only a limited number of ships can be taken into account since search space grows exponentially. For this reason, we introduce the use a metaheuristic allocation method (GRASP, [6]). This is a multistart method to solve hard combinatorial optimization problem and its basic process is showed in Algorithm 14. Principal characteristic is obtaining optimized solutions in a very efficient way. A local search should be at the end of this method, but in our problem this process would be very complex and its benefits would be insignificant.

Factor ρ is a parameter which the higher value has, the more random the process is. The elements E are the unmoored vessels and the cost function is the one it must be minimized, this is the waiting time T_w plus the priority of each vessel. And finally, the s set consists of the E elements with assigned times (entrance and exit) and number of QC.

Algorithm 14 describes the specification of the application of the general GRASP method described in Algorithm 13 to the berthing problem. The parameter $\rho \in (0 \leq \rho \leq 1)$ adjusts the random component of the GRASP procedure and the parameter

Algorithm 14: Pseudocode grasp metaheuristic

```
   Data: $\rho$ factor; $E$ elements
   Result: Solution $s$
1  $s \leftarrow \varnothing$;
2  $C \leftarrow E$;
3  evaluate costs $c(e) \forall e \in C$;
4  while $c \neq \varnothing$ do
5      $c_{min} \leftarrow \min\{c(e)|e \in C\}$;
6      $c_{max} \leftarrow \max\{c(e)|e \in C\}$;
7      $RCL \leftarrow \{e \in C|c(e) \leq c_{min} + \rho(c_{max} - c_{min})\}$;
8      $t \leftarrow random(RCL)$;
9      $s \leftarrow s \cup t$;
10     $C \leftarrow C - t$;
11     re-evaluate costs $c(e) \forall e \in C$;
12 end
```

$\gamma \in (0 \leq \gamma \leq 1)$ penalizes long waiting times of waiting vessels. This algorithm should be performed in combination with the Algorithm 13 for allocation vessel candidates. The Algorithm 15 orders the queue of waiting vessels and assigns the moor time for each vessel in order to minimize the balanced overall waiting time. Thus, it obtains and optimized allocation tail docking. The algorithm is successively executed until the termination condition (maximum number of executions, solution quality, etc.) is reached.

4 An Integrated Approach for the Container Stacking Problem and the Berth Allocation Problem

As we have pointed in the introduction, both the container stacking problem and the berth allocation problem are well-known problems and some techniques have been developed to solved them separately. However, no techniques have been developed to optimize both problems in an integrated way. Only some works integrate the berth allocation problem with the Quay Crane Assignment Problem [8] but there exit a relationship between the optimization of maritime operations (the berth allocation problem) and terminal operations (Quay Crane Assignment Problem, problem stacking problem, etc.). Figure 13 shows an example of three berth allocation plans and a block of containers to be loaded in the vessels. Containers of type A, B and C must be loaded in vessels A, B and C, respectively. In the first berth allocation plan the order of vessel is A-B-C, the waiting time for this plan is 205 time units and the number of reshuffles needed to allocate the white containers at the top of the stacks is 110. The second berth allocation plan is B-A-C. In this case the waiting time for this plan is 245 time units and the number of reshuffles is 260. Finally, The third berth allocation plan is C-B-A, the waiting time for this plan is 139 time units and the number of reshuffles is 450. The question is straightforward: what is a better solution? The answer could depend on many factors and the policy of each maritime terminal, so that we consider a lineal combination of these factors and the port manager could give the appropriate weight to each parameter α and β in order to minimize the global cost. In equation 11 a solution to the integrated problem is a lineal combination

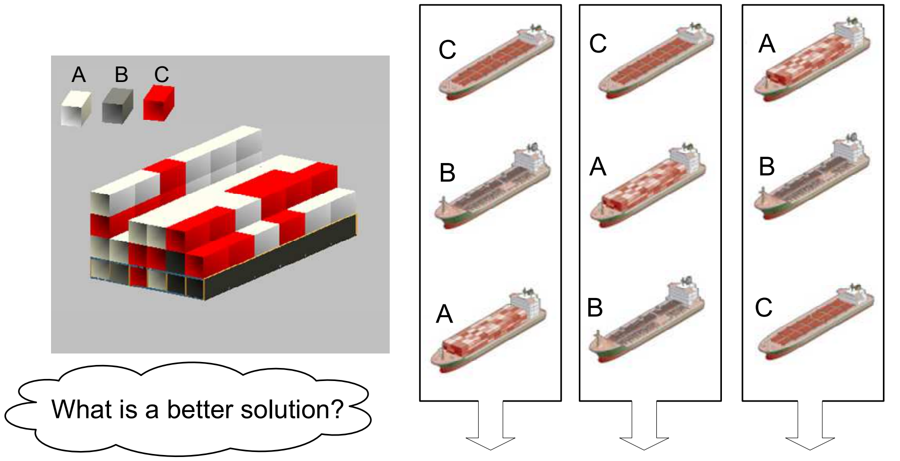
Algorithm 15: Allocating Vessels using GRASP metaheuristic

```
Data: $\rho$ factor; $E$ elements; $b$: state of the berth at $t_0$
Result: Solution $s$
/* current time at which algorithm is performed */
$T_{assig} \leftarrow t_0$;
$V \leftarrow \{V_i\}$;
$T_{exit} \leftarrow \{\}$;
while $V \neq \varnothing$ do
    /* $V_i$ which can be allocated in available quay */
    $SV \leftarrow \{V_i\} | \texttt{checkDisponibility}(V_i, \max(T_{assig}, a(V_i)), d(V_i)) = \texttt{True}$;
    if $SV \neq \varnothing$ then
        $T_{assig} \leftarrow first(T_{exit})$ ;    // Time of the next exiting vessel
    else
        foreach $V_i \in SV$ do
            $\texttt{Allocate}(V_i)$ ;    // Procedure allocate $V_i$
            $\texttt{Cost}(V_i) \leftarrow 0$;
            foreach $V_j \in SV | V_i \neq V_j \wedge \max(T_{assig}, a(V_i)), \max(T_{assig}, a(V_i)) + d(V_i) \cap$
            $\max(T_{assig}, a(V_i)), \max(T_assig, a(V_i)) + d(V_i) \neq \varnothing$ do
                $\texttt{Cost}(V_i) \leftarrow \texttt{Cost}(V_i)$
                $+pr(V_j)((\max(T_{assig}, a(V_j)) + d(V_j)) - \max(T_{assig}, a(V_j)))^{\gamma}$;
            end
            $\texttt{MaxCost} \leftarrow \max_{V_i \in SV}(\texttt{Cost}(V_i))$;
            $\texttt{MinCost} \leftarrow \min_{V_i \in SV}(\texttt{Cost}(V_i))$;
            $V_{assig} \leftarrow \{V_i \in SV | \texttt{Cost}(V_i) \in \{\texttt{MinCost}, \texttt{MinCost} + \rho(\texttt{MaxCost} - \texttt{MinCost})\}\}$;
            $V_k \leftarrow random(V_{assig})$;
            $\texttt{insertVessel}(V_k, \max(T_{assig}, a(V_k)), b)$;
            $\texttt{update}(b)$ ;    // update the state of the berth
            $\texttt{remove}(V_k, SV)$;
            $T_{exit} \leftarrow T_{exit} \cup d(V_k)$;
        end
    end
end
;    // All waiting vessels have been allocated
```

of the solution obtained by the berth allocation problem ($SBAP_i$) and the solution to the container stacking problem ($SCSP_i$).

$$Sol_i = \alpha \times SBAP_i + \beta \times SCSP_i \tag{11}$$

To this end, we have developed an integrated system to achieve the best combined solution. The data flow diagram of the Integrated System Functioning can be seeing in Figure 14. Firstly, both the berth allocation problem (BAP) and the container stacking problem (CSP) are loaded. Then, the parameters α, β and the number of solutions for the BAP are given by the port operator. In the next step the BAP is solved to achieve a solution based on the constraints and criteria given by the user. Once a solution to the BAP is obtained, the CSP is solved by taken into account the order of vessels obtained in the solution and following the constraints and criteria given by the user. In this step a global solution is calculated by using the formula 11. Thus, this is the best solution found so far. Then, iteratively new solutions are found for the BAP while $i < NSol$. In each loop, the obtained solution for the BAP is multiplied by α in order to analyze whether this solution could improve the current global solution. In this case the CSP is solved taken



Waiting Time	205 Time Units	245 Time Units	139 Time Units
Number of Reshuffles	110	260	450

Figure 13: Three different plans for the Berth Allocation Problem: What is better?

into account the plan obtained in the BAP. Then a new global solution is found and it is compared with the best obtained so far. If the new solution is better $Sol_i < BestSol$ then the best solution is updated $BestSol = Sol_i$. Thus, we obtain the best solution that minimizes both criteria. It must be taken into account that the BAP will be solved $NSol$ times meanwhile the CSP could be solved in a smaller number of cases.

For instance, in the example presented in Figure 13 if the user selects $\alpha = 0.75$ and $\beta = 0.25$, the first solution has a total cost of $0.75 \times 205 + 0.25 \times 110 = 181.25$. The second solution for the BAP is 245 time units, so $\alpha \times 245 = 0.75 \times 245 = 183.75$. In this case this solution will not take part of a global solution due to the fact that the best solution will not be achieved so the CSP for this plan is not executed. Some examples are presented in table 1 for the problem of Figure 13. It must be taken into account that depending on the parameters α and β the best solution will change. Thus the port operator will assign these parameters depending on the requirement of the terminal.

In the integrated system, the constraints and criteria for the BAP and the CSP are given by the user and customized constraints for each terminal could be included in order to achieve a solution for each problem.

5 Evaluation

In this section we evaluate the behavior of the algorithms developed in the paper. To this end, we have divided this section in three subsections concerning to the three problems presented in the paper: the CSP, the BAP and the problem that integrates the both above

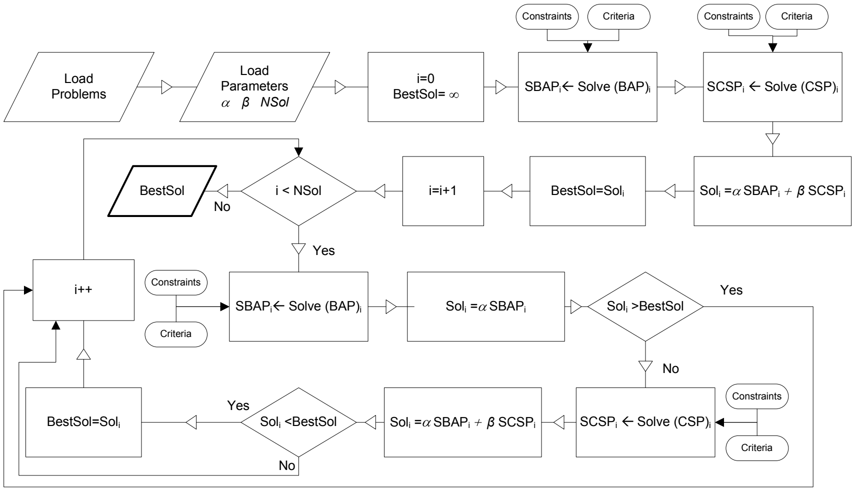


Figure 14: Data flow diagram of the Integrated System Functioning

Table 1: The best solution for the example of figure 13 by varying the parameters α and β

Parameters	A-B-C	B-A-C	C-B-A
$\alpha = 0.5$ $\beta = 0.5$	**157.5**	252.5	294.5
$\alpha = 0.75$ $\beta = 0.25$	**181.25**	-	216.75
$\alpha = 0.9$ $\beta = 0.1$	195.5	-	**170.1**

problems. The experiments were performed on random instances. A random instance of a yard-bay in the CSP is characterized by the tuple $< n, s >$, where n is the number of containers in a yard-bay and s is the number of selected containers in the yard-bay. Each instance is a random configuration of all containers distributed along six stacks with 4 tiers.

A random instance for the BAP has 20 vessels with an arrival exponential distribution with all needed factors randomly fixed (length, draft and moves). As we mentioned above, our goal is to minimize the total waiting time elapsed to served the n vessels.

All problem instances were solved on a personal computer equipped with a Core 2 Quad Q9950 2.84Ghz with 3.25Gb RAM.

5.1 Evaluation of the Container Stacking Problem

In this subsection, we evaluate the behavior of our heuristic with a set of optimization criteria presented in this paper. First, we present a comparison between our basic domain dependent heuristic h_1 against a domain independent one (Metric FF). Thus, Table 2 presents the average running time (in milliseconds) to achieve a first solution as well as the best solution found (number of reshuffles) in 10 seconds for our domain-dependent planner and the average running time (in milliseconds) and the quality of the solution for Metric FF. Both planners have been tested in problems $< n, 4 >$ evaluating 100 test cases for each one. Thus, we fixed the number of selected containers to 4 and we increased the number of containers n from 15 to 21.

It can be observed that our new domain-dependent heuristic is able to find a solution in a few milliseconds, meanwhile the domain-independent planner (Metric FF) needs much time for finding a solution and also, this solution needs more moves to get a goal state. Furthermore, due to the fact that our tool is an anytime planner, we evaluate the best solution found in a given time (10 seconds).

Table 2: Average number of reshuffles and running time of $Metric_F F$ and h_1 in problems $< n, 4 >$.

	Metric FF		**Heuristic (h_1)**	
Instance	Running time	Solution	Running time first solution	Best Solution in 10 secs
$< 13, 4 >$	22	3.07	2	3.07
$< 15, 4 >$	3102	4.04	6	3.65
$< 17, 4 >$	4669	5.35	12	4.35
$< 19, 4 >$	6504	6.06	24	4.72
$< 20, 4 >$	22622	7.01	36	5.22
$< 21, 4 >$	13981	6.82	66	5.08

Now we show the effects of using each one of the criteria described in Section 2.4 separately. In Table 3, we present the average sum of distances between the selected containers and the right side of the layout in both our domain-independent heuristic and our domain-dependent heuristic with distance optimization for problems $< n, 4 >$. As mentioned above, we fixed the number of selected containers to 4 and we increased the number of containers n from 13 to 21. It can be observed that distance optimization function helps finding solution plans that place the selected containers closer to the cargo side of the yard-bay.

Applying the criterion or function showed in Algorithm 3 we obtain the results appeared in Table 4. These results are the comparison between the number of solved problems over 100 problems $< n, 4 >$ using or not that criterion in just one second. Through this table we can conclude that:

- The higher number of containers, the lower problems are solved. This is because as we increase the number of containers there are less positions or gaps where containers could be remarshaled.

Table 3: Average distance obtained by considering distance or not in our domain-dependent heuristic $< n, 4 >$ with 4 tiers.

Instance	***Metric FF***		OC_{1d}	
	Distance	*Reshuffles*	*Distance*	*Reshuffles*
$< 13, 4 >$	11.28	3.07	10.91	3.07
$< 15, 4 >$	10.60	4.04	9.21	3.65
$< 17, 4 >$	10.58	5.35	8.87	4.46
$< 19, 4 >$	12.28	6.06	8.32	4.86
$< 20, 4 >$	12.71	7.01	7.75	5.56
$< 21, 4 >$	12.20	6.82	8.36	5.34

- Allowing movements to the 5^{th} helps us to solve more problems. It is remarkable with instances $< 23, 4 >$ with H_1 only three problems could be solved, however OC_{1t} solves 84 over 100 problems.

Table 4: Number of solved problems $< n, 4 >$ with 4 and 5 tiers during the process.

Instance	**4 tiers h_1**	**5 tiers OC_{1t}**
$< 19, 4 >$	100	100
$< 20, 4 >$	100	100
$< 21, 4 >$	95	99
$< 23, 4 >$	3	84

Last criterion for solving problems where we only take into account one yard-bay is showed in Section 2.4.3. As we mentioned in this section, since the last function (Algorithm 7) presents the best results after the whole process of remarshaling, we do the comparison in Table 5 among the solutions given by Metric FF planner, the initial one h_1 and OC_{1b} (*Both*) in 50 test cases. The last two ones look for solutions during 1 second in instances of $< 15, 4 >$ and 4 seconds in instances of $< 17, 4 >$. These times are used in order to achieve a solution for all the instances.

Sinks are calculated by Algorithm 10. As we mentioned above, we consider that there is a sink where the difference in tiers between two adjacent rows is higher than 2. Thereby, in this algorithm we are counting sinks produced between two contiguous stacks at the same yard-bay as well as between two rows in one yard-bay and the previous one. This process takes into account the goal containers in final yard-bays.

From here we realize an evaluation for the criteria presented in Section 2.5. Table 6 shows the performance of the criteria for solving the whole block of yard-bays. These experiments were performed in blocks of 20 yard-bays and each one of them are instances $< 15, 4 >$. This evaluation was carried out in a yard with 3 blocks of 20 yard-bays.

Table 5: Average number of movements, sinks and time for the first solution in problems $< 15, 4 >$ (1) and $< 17, 4 >$ (2) using or not balanced heuristics.

	Metric FF		$\boldsymbol{h_1}$		$\boldsymbol{OC_{1b}}$	
	(1)	(2)	(1)	(2)	(1)	(2)
Reshuffles	3.72	4.24	3.42	3.68	5.36	5.30
Sinks	0.62	0.50	0.92	0.66	0	0
Time First Sol.	2621	2961	6	10	16	22

Thus, the results showed in Table 6 represent the average number of reshuffles, the average number of sinks generated along the block and the average number of unsatisfied dangerous containers. Results given by these optimization criteria are the average of the best solutions found in 10 seconds.

The number of unfeasible relationships between dangerous containers is calculated by means of Algorithm 11. Basically, we look for those pairs of dangerous containers whose distance between them is shorter than minimum distance (D_{min}).

In this table, it can be observed that h_1 still outperforms *Metric FF* in the average number of reshuffles. However, due to the fact that they do not take into account the balancing constraints, *Metric FF* generated an average of 24.33 sinks in the block of yard-bay and h_1 generated and average of 32.67 sinks. And it occurs the same for the average number of unfeasible constraints for dangerous containers, *Metric FF* gives us 15.33 and h_1 obtains 7.67.

Taking into account that OC_N is a junction of OC_{nB} and OC_{nD}, both OC_{nB} and OC_{nD} solved their problems, that is, OC_{nB} obtained its solutions with no sinks and OC_{nD} obtained its solutions by satisfying all dangerous constraints. Furthermore, OC_N was able to solve its problems by satisfying both types of constraints. However we could state that balancing problem is harder than the problem related to dangerous containers because OC_{nB} needs more reshuffles to obtain a solution plan than OC_{nD}. Moreover, we observe with OC_{nB}, OC_{nD} and OC_N ensure the established requirements however the average reshuffles is increased with respect to h_1.

Table 6: Average results with blocks of 20 *yard-bays* each one being a $< 15, 4 >$ problem.

	Metric FF	$\boldsymbol{h_1}$	$\boldsymbol{OC_{nB}}$	$\boldsymbol{OC_{nD}}$	$\boldsymbol{OC_N}$
Reshuffles	3.98	3.60	5.68	4.30	6.53
Sinks	24.33	32.67	0	33.33	0
Non-Safe Dangerous	15.33	7.67	8.00	0	0

5.2 Evaluation of the Berth Allocation Stacking Problem

In this section, we evaluate the behavior of the three policies presented in this paper. Firstly, Table 7 shows the times of employing the complete search against the GRASP method with 1000 iterations. As observed Complete search is impracticable from 12 vessels (approximately 3 hours). However, GRASP method takes around 30 seconds to solve a schedule of 20 vessels.

Table 7: Computing time elapsed (seconds)

Number	**Complete search**	**GRASP**
5	< 1	1
10	112	8
11	1105	9
12	11830	10
13	57462	12
15	$\cdots$	15
20	$\cdots$	30

In Table 8 and 9, we show the average waiting times using FIFO and Complete Search methods described above with two different inter-arrival distributions. Through these data, it is demonstrated that FIFO method results a schedule which is far away from the best one.

Table 8: Total waiting time elapsed (Vessels separated)

Vessels	**FCFS**	**CS**
5	73.72	46.10
10	256.53	136.26

Table 9: Total waiting time elapsed (Vessels close)

Vessels	**FCFS**	**CS**
5	117.52	80.25
10	586.65	351.25

Using as minimization function the waiting time, we obtain the results given by Table 10 and Table 11. It is remarkable that using GRASP is more profitable when the inter-arrival distribution of the vessels is high since over 100 problems are improved around 90%.

Table 10: Instances that improve the FIFO solution for 10 vessels

Factor α	**Vessels separated**	**Vessels close**
0.0	53	70
0.1	54	74
0.2	57	77
0.3	66	81
0.4	72	91
0.5	65	85
0.6	62	84
0.7	63	85
0.8	63	86
0.9	57	87
1.0	61	81

Table 11: Instances that improve the FIFO solution for 20 vessels

Factor α	**Vessels separated**	**Vessels close**
0.0	42	86
0.1	48	90
0.2	53	90
0.3	56	91
0.4	68	92
0.5	56	93
0.6	55	91
0.7	61	95
0.8	60	91
0.9	59	91
1.0	51	91

5.3 Evaluation of the Integrated System

In this section, we evaluate the behavior of the integrated system in random instances. We randomly generated a set of 10 vessels for berthing and each vessel must load 17 containers from the container yard. The containers were randomly located in the yard. Thus the yard was composed of 170 containers so that it remained empty once the vessels were loaded. For each problem, we generated 50 random instances with a different configuration of containers in the yard. The average number of reshuffles is presented in the tables 12, 13 and 14.

Figure 12 shows the total cost of both the berth allocation problem and the container stacking problem by fixing the parameters $\alpha = 0.8$ and $\beta = 0.2$. Due to the fact that the makespan was pondered by 0.8, the best solution was achieved for the plan with lower makespan. The first solution generated a total cost of 307.1 so it was not needed to carry out the CSP for the second, third and fourth plan due to the fact that a better solution would not improve the first solution. The fifth plan was selected the best solution with a makespan of 349 time units and an average number of reshuffles of 100.6.

Table 12: Total cost of BAP and CSP for $\alpha = 0.8$ and $\beta = 0.2$

Plan	**Makespan**	**Number of Reshuffles**	$\alpha = 0.8$ $\beta = 0.2$
1	361.0	91.3	307.1
2	430.0	98.6	-
3	440.0	93.3	-
4	411.0	99.4	-
5	349.0	100.6	**299.3**
6	400.0	98.9	339.8
7	362.0	95.4	308.7
8	419.0	99.4	355.1
9	544.0	92,7	-
10	390.0	98.7	331.7

In table 13, the parameters α and β were set to $\alpha = 0.8$ and $\beta = 0.2$. In this case the plan number 5 is also considered the best solution with a total cost of 249.6.

Table 13: Total cost of BAP and CSP for $\alpha = 0.6$ and $\beta = 0.4$

Plan	**Makespan**	**Number of Reshuffles**	$\alpha = 0.6$ $\beta = 0.4$
1	361.0	91.3	253.1
2	430.0	98.6	-
3	440.0	93.3	-
4	411.0	99.4	286.4
5	349.0	100.6	**249.6**
6	400.0	98.9	279.5
7	362.0	95.4	291.2
8	419.0	99.4	355.1
9	544.0	92.7	-
10	390.0	98.7	273.5

In table 14, the parameters α and β were set to $\alpha = 0.4$ and $\beta = 0.6$. In this case the plan number 1 is considered the best solution with a total cost of 199.2. Thus, depending on the parameters given by the user, the best solution may change. It must be taken into account that in this last case it is necessary to solve almost all container stacking problems due to the fact that no partial solution can be ruled out by solving only the BAP.

Table 14: Total cost of BAP and CSP for $\alpha = 0.4$ and $\beta = 0.6$

Plan	**Makespan**	**Number of Reshuffles**	$\alpha = 0.4$ $\beta = 0.6$
1	361.0	91.3	**199.2**
2	430.0	98.6	231.1
3	440.0	93.3	232.0
4	411.0	99.4	224.1
5	349.0	100.6	199.9
6	400.0	98.9	219.3
7	362.0	95.4	202.1
8	419.0	99.4	227.3
9	544.0	92.7	-
10	390.,0	98.7	215.2

Due to the fact that all vessels have the same number of out containers, the average number of reshuffles is ranged between 91.3 and 100.6 so that the standard deviation is 3.3 . However if the number of out containers is randomly selected, the standard deviation increases considerably.

6 Conclusions

This paper presents a set of artificial intelligence-based heuristics for solving well-known problems presented in maritime terminals: the berth allocation problem (BAP) and the container stacking problem (CSP). To this end, we have developed a set of planning techniques for solving the container stacking problem and a set of algorithms for solving the berth allocation problem independently. Then we have developed an algorithm to solve both problems in an integrated way.

For the CSP, we have developed a domain-dependent planning tool for finding optimized plans to obtain an appropriate configuration of containers in a yard-bay as well as doing simulations to try different configurations or requirements as it is made in [23]. Thus, given a set of outgoing containers, our planner minimizes the number of necessary reshuffles of containers in order to allocate all selected containers at the top of the stacks. This proposed planner is able to satisfy both balancing constraints and dangerous container constraints, as well as reducing the distance of the goal containers to the cargo side or allowing a fifth tier during the remarshaling process.

We have developed two algorithms for solving the BAP. The first one is a complete algorithm in order to find the best combination of vessels to minimize the total waiting time. This algorithm is the base for developing new metaheuristics techniques such as

GRASP-based techniques [6]. Our grasp-based technique looks for an initial solution and then the greater the number of iterations, the better the solution obtained by this technique.

Finally, we propose an integrated system that combines the previous methods in order to achieve a solution to both problems. This integrated system obtains a solution to the BAP and then it carries out the CSP to minimize the number of reshuffles needed for the obtained berth plan. Thus, a solution that optimizes one problem could not be the more appropriate for the other problem. An expert human must assign values to the parameters in order to obtain the customized best solution.

In further work, we will include in the integrate system the Quay Crane Assignment Problem in order to analyze the relationship among these three problems and to obtained optimized solutions to the global problem.

Acknowledgment

This work has been partially supported by the research projects TIN2007-67943-C02-01 (Min. de Educación y Ciencia, Spain-FEDER), and P19/08 (Min. de Fomento, Spain-FEDER), as well as with the collaboration of the maritime container terminal MSC (Mediterranean Shipping Company S.A.).

References

[1] C. Bierwirth and F. Meisel. A survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research*, 202(3):615–627, 2010.

[2] Shih-Huang Chen and Jun-Nan Chen. Forecasting container throughputs at ports using genetic programming. Expert Systems with Applications, 37(3):2054 – 2058, 2010.

[3] C.Y. Cheong, K.C. Tan, and D.K. Liu. Solving the berth allocation problem with service priority via multi-objective optimization. In *Computational Intelligence in Scheduling, 2009. CI-Sched '09. IEEE Symposium on*, pages 95 –102, 2 2009-march 30 2009.

[4] Tzung-Nan Chuang, Chia-Tzu Lin, Jung-Yuan Kung, and Ming-Da Lin. Planning the route of container ships: A fuzzy genetic approach. Expert Systems with Applications, 37(4):2948 – 2956, 2010.

[5] J.F. Cordeau, G. Laporte, P. Legato, and L. Moccia. Models and tabu search heuristics for the berth-allocation problem. *Transportation science*, 39(4):526–538, 2005.

[6] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

[7] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. *AIPS-98 Planning Committee*, 1998.

[8] Giovanni Giallombardo, Luigi Moccia, Matteo Salani, and Ilaria Vacca. Modeling and solving the tactical berth allocation problem. *Transportation Research Part B: Methodological*, 44(2):232 – 245, 2010.

[9] Y. Guan and R.K. Cheung. The berth allocation problem: models and solution methods. *OR Spectrum*, 26(1):75–92, 2004.

[10] Ourania Hatzi, Dimitris Vrakas, Nick Bassiliades, Dimosthenis Anagnostopoulos, and Ioannis Vlahavas. A visual programming system for automated problem solving. Expert Systems with Applications, 37(6):4611 – 4625, 2010.

[11] L. Henesey. Overview of Transshipment Operations and Simulation. In *MedTrade conference, Malta, April*, pages 6–7, 2006.

[12] J. Hoffman and B. Nebel. The FF planning system: Fast planning generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[13] Jörg Hoffmann. The metric-ff planning system: translating ”ignoring delete lists” to numeric state variables. *J. Artif. Int. Res.*, 20(1):291–341, 2003.

[14] A. Imai, K.I. Nagaiwa, and TAT Chan Weng. Efficient planning of berth allocation for container terminals in Asia. *Journal of advanced transportation*, 31(1):75–94, 1997.

[15] Kap Hwan Kim. Evaluation of the number of rehandles in container yards. *Computers & Industrial Engineering*, 32(4):701 – 711, 1997. New Advances in Analysis of Manufacturing Systems.

[16] K.H. Kim and J.W Bae. Re-marshaling export containers in port container terminals. *Computers & Industrial Engineering*, 35(3-4):655 – 658, 1998. Selected Papers from the 22nd ICC and IE Conference.

[17] K.H. Kim and G.P. Hong. A heuristic rule for relocating blocks. *Computers & Operations Research*, 33(4):940–954, 2006.

[18] K.W. Kim, Y.M. Park, and K.R. Ryu. Deriving decision rules to locate export containers in container yards. *European Journal of Operational Research*, 124:89–101, 2000.

[19] KK Lai and K. Shih. A study of container berth allocation. *Journal of Advanced Transportation*, 26(1):45–60, 1992.

[20] Yusin Lee and Nai-Yun Hsu. An optimization model for the container pre-marshalling problem. *Computers & Operations Research*, 34(11):3295 – 3313, 2007.

[21] K. Park, T. Park, and K.R. Ryu. Planning for remarshaling in an automated container terminal using cooperative coevolutionary algorithms. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1098–1105. ACM, 2009.

[22] M. Salido, O. Sapena, and F. Barber. An artificial intelligence planning tool for the container stacking problem. *Proceedings of the 14th IEEE international conference on Emerging technologies and factory automation*, pages 532–535, 2009.

[23] M. Salido, O. Sapena, and F. Barber. What is better: 4 tiers or 5 tiers in the container stacking problem? *The International Workshop on Harbour, Maritime and Multimodal Logistics Modelling and Simulation*, 2009.

[24] M. Salido, O. Sapena, M. Rodriguez, and F. Barber. A planning tool for minimizing reshuffles in containers terminals. *ICTAI 2009: 21st International Conference on Tools with Artificial Intelligence*, 2009.

[25] O. Sapena and E. Onaindía. Domain independent on-line planning for strips domains. *In proc. IBERAMIA-02, 2527*, pages 825–834, 2002.

[26] R. Stahlbock and S. Voß. Operations research at container terminals: a literature update. *OR Spectrum*, 30(1):1–52, 2008.

[27] S. Theofanis, M. Boile, and M.M. Golias. Container terminal berth planning. *Transportation Research Record: Journal of the Transportation Research Board*, 2100(-1):22–28, 2009.

[28] I.F.A. Vis and R. De Koster. Transshipment of containers at a container terminal: an overview. *European Journal of Operational Research*, 147:1–16, 2003.

[29] Terry Winograd. *Procedures as a representation for data in a computer program for understanding natural language*. MIT. Cent. Space Res., Cambridge, MA, 1971.

A Decision Support System for Managing Combinatorial Problems in Container Terminals

Miguel A. Salido, Mario Rodriguez-Molins, Federico Barber
Instituto de Automática e Informática Industrial
Universidad Politécnica de Valencia.
Valencia, Spain

Abstract

A container terminal is a facility where cargo containers are transshipped between different transport vehicles. We focus our attention on the transshipment between vessels and land vehicles, in which case the terminal is described as a maritime container terminal. In these container terminals, many combinatorial related problems appear and the solution of one of the problems may affect to the solution of other related problems. For instance, the berth allocation problem can affect to the crane assignment problem and both could also affect to the container stacking problem. Thus, terminal operators normally demand all containers to be loaded into an incoming vessel should be ready and easily accessible in the yard before vessel's arrival. Similarly, customers (i.e., vessel owners) expect prompt berthing of their vessels upon arrival. However the efficiency of the loading/unloading tasks of containers in a vessel depends on the number of assigned cranes and the efficiency of the container yard logistic. In this paper, we present a decision support system to guide the operators in the development of these typical tasks. Due to some of these problems are combinatorial, some analytical formulas are presented to estimate the behavior of the container terminal.

Keywords Berth allocation problem, Quay Crane Assignment Problem, Container Stacking Problem, Decision Support System, Artificial Intelligence

1 Introduction

Container terminals generally serve as a transshipment between ships and land vehicles (trains or trucks). Henesey shows in [17] how this transshipment market is growing fast. Between 1990 and 2008, container traffic has grown from 28.7 million to 152.0 million of movements. This corresponds to an average annual compound growth of 9.5%. In the same period, container throughput went from 88 million to 530 million of containers, which represents an increase of 500%. The surge of both container traffic and throughput is linked with the growth of international trade in addition to the adoption of containerization as privileged vector for maritime shipping and inland transportation [1].

The efficient management of containers in port requires more analysis and development to ensure reliability, delivery dates or handling times in order to improve productivity and container throughput from quay to landside and vice versa. Extensive surveys are provided about operations at seaport container terminals and methods for their optimization [32, 30]. Moreover, other problems are faced on planning the routes for liner shipping services to obtain the maximal profit [7]. Another important issue for the success at any container terminal is to forecast container throughput accurately [5]. Thus, they could develop better operational strategies and investment plans.

The main research on optimization methods in container terminals is related to reduce the berthing time of vessels. This objective generates a set of interrelated problems such as berth allocation, yard-side operation, storage operation and gatehouse operation. Usually, each one of these problems is managed independently of others due to their exponential complexity. However, these problems are clearly interrelated so that an optimized solution of one of them restrings the possibility of obtaining a good solution in another.

The overall goal collaboration between our group at the Technical University of Valencia (UPV), Valencia Port Foundation, and the maritime container terminal MSC (Mediterranean Shipping Company S.A) is to offer assistance to help in planning and scheduling tasks such as the allocation of spaces to outbound containers, to identify bottlenecks, to determine the consequences of changes, to provide support in the resolution of incidents, to provide alternative berthing plans, etc.

In this paper, we focus our attention on three important and interrelated problems: the Berth Allocation Problem (BAP), the Quay Crane Assignment Problem (QCAP) and the Container Stacking Problem (CStackP) (see Figure 1). Briefly, the BAP and QCAP consist of the allocation of docks and quay cranes to incoming vessels under several constraints and priorities (length and depth of vessels, number of containers, etc.). On the other hand, when a vessel berths, export containers stacked to be loaded in the vessel should be on top of the stacks of the container yard. Therefore, the CStackP consists of relocating the containers so that the yard crane does not need to do re-handling work at the time of loading. These two problems are clearly related: an optimal berth allocation plan may generate a large amount of relocations for export containers; meanwhile a suboptimal berth allocation plan could require fewer rearrangements. Terminal operators should decide which solution is the most appropriate in each scenario.

In order to provide a computer-based decision support system, we integrate a set of intelligent techniques for solving these problems concurrently in order to achieve a mixed-solution that combines optimization of BAP, QCAP and CStackP. To this end, we developed a heuristically-guided planner for generating a rehandling-free intra-block remarshaling plan for container yards (CStackP problem). Due to the fact that this is a time consuming task, we present in this paper an analytic formula to estimate the number of reshuffles needed to solve this problem. Then, we present a meta-heuristic approach for solving the BAP+QCAP as an independent problem. Afterwards, we integrate solutions obtained from BAP+QCAP and StackP systems, so that terminal operators should

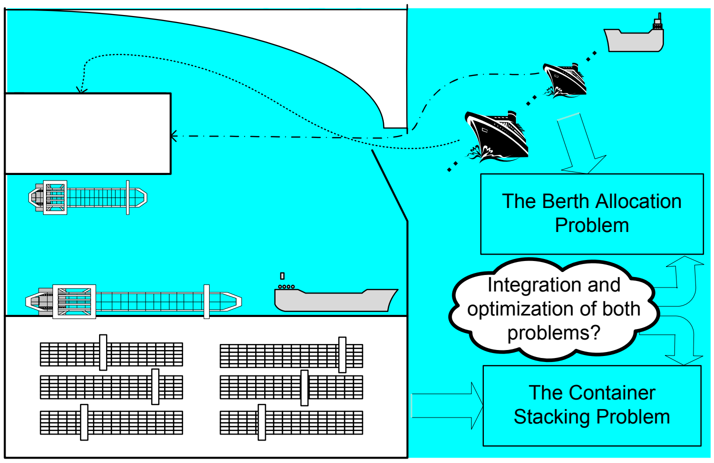

The Berth Allocation Problem
Integration and optimization of both problems?
The Container Stacking Problem

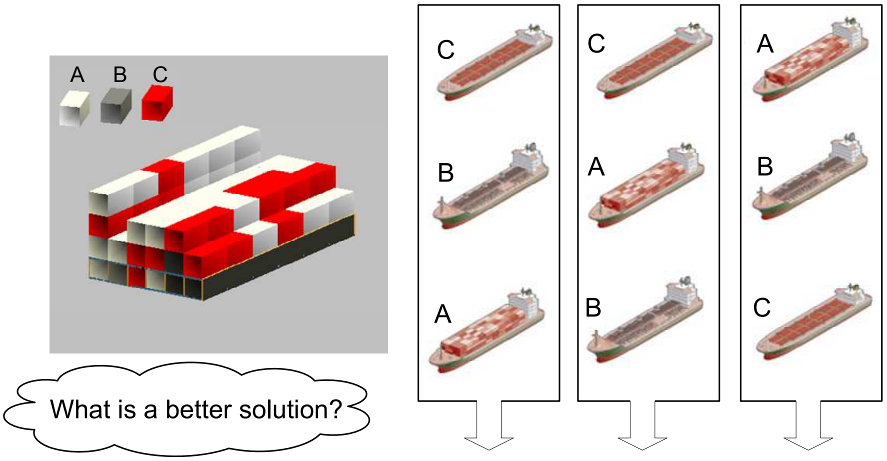
Figure 1: Integrated Remarshaling, Berthing and Quay Crane allocation problems in Maritime Terminals.

ultimately decide which solution is the most appropriate in relation to a multi-objective function: to minimize the waiting times of vessels and to minimize the amount of relocations of containers.

These techniques will be very useful for terminal operators due to berth allocation is especially important in case of ship delays because in this case a new berthing place has to be allocated to the ship whereas containers are already stacked in the yard [30] and a remarshaling plan remains necessary to minimize the berthing time.

2 Integrating BAP, QCAP and CStackP

As we have pointed out, both the CStackP and the BAP+QCAP are well-known problems and several techniques have been developed to solve them separately. However, few systems have been developed to relate and optimize both problems in an integrated way. Some works consider berth and yard planning in a common optimization model [2, 4, 10], but they are mainly focused on storage strategies. Moreover, only some works integrate the BAP with the QCAP. Giallombardo et al. [12] try to minimize the yard-related housekeeping costs generated by the flows of containers exchanged between vessels. However, there also exists a relationship between the optimization of maritime and terminal-sides operations (BAP, QCAP, CStackP, etc.). Figure 2 shows an example of three berth allocation plans with the corresponding quay crane allocations and a block of containers to be loaded in the vessels. Containers of type A, B and C must be loaded in vessels A, B and C, respectively. In the first berth allocation plan, the order of vessels is A-B-C and the quay crane allocation is two cranes, three cranes and one crane, respectively. The second berth allocation plan is C-B-A. In this case the quay crane allocation is three, two and one, respectively. Finally, the third berth allocation plan is B-C-A and two quay cranes are allocated to all vessels. Each configuration generates a different waiting time for berthing and different handling times, and the port operator probably selects the best solution to optimize these (BAP and QCAP) problems. However the best solution of these two problems could generate a large number of reshuffles in the yard so the question is straightforward: what is a better solution? Perhaps a solution that optimizes the BAP+QCAP could not be the more appropriate for the CStackP (and vice versa).



Waiting Time	205 Time Units	245 Time Units	139 Time Units
Number of Reshuffles	110	260	450

Figure 2: Different alternatives of BAP and QCAP.

Given a waiting queue of vessels to be allocated and a given state of the containers in the container yard, each solution for the BAP+QCAP ($SBAP_i$: a feasible sequence of mooring and a feasible quay crane allocation), requires a different number of container's re-locations in the associated CStackP solution ($SCStackP_i$) in order to put on top the containers to be loaded according to the order of berthing. We can associate a cost to each $SBAP_i + SQCAP_i$ related to the total weighted waiting time and handling time of vessels of this berthing order (T_w). Likewise, we can associate a cost to each $SCStackP_i$ as the number of required container relocations. Therefore, we can qualify the optimality of each global solution (Sol_i) of BAP+QCAP and CStackP as a lineal combination of the quality of each partial solution:

$$Cost(Sol_i) = \alpha * Cost(SBAP_i + SQCAP_i) + \beta * (SCStackP_i) \qquad (1)$$

The best decision will depend on the policy of each maritime terminal (α and β parameters). The data flow diagram of the Integrated System Functioning can be seeing in Figure 3. Firstly, the BAP, QCAP and the CStackP data are loaded in the integrated system. Next, the BAP+QCAP is solved to achieve a solution ($SBAP_i$) based on their constraints and optimization criteria. Then, the CStackP is estimated by taking into account the berthing order of vessels obtained in $SBAP_i$. This estimator returns the number of reshuffles needed to achieve a solution. After this step, the cost of the global solution (Sol_i) can be calculated by using the previous expression (Equation 1). By iterating this integrated process, the operators can obtain a qualification cost of each feasible Sol_i, as well as the best global solution, according to the given α and β parameters. A branch and bound method has been also applied in the integrated search for the best global solution (Sol_i), so that the search can be pruned each time the current solution does not improve the best solution found so far. Finally, once the best solution is obtained, the CStackP planner is executed to obtain the specific movements for all remarshaling tasks. This plan is sequentially obtained for each vessel according to the solution obtained in $SBAP_i$ and the current state of the container yard. Thus, the optimized remarshaling plan for the berthing order of vessels of $SBAP_i$ is obtained.

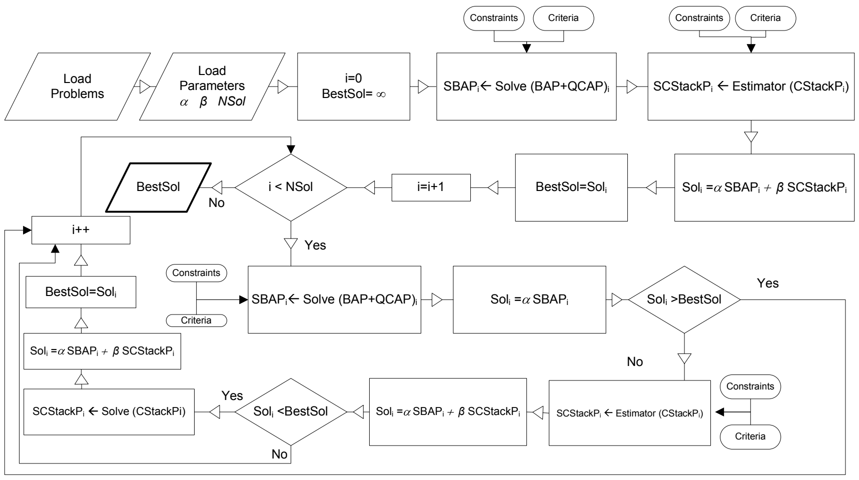


Figure 3: Data flow diagram of the Integrated System Functioning.

In next sections we develop our proposed techniques for estimating and solving the container stacking problem, the berth allocation problem and the quay crane allocation problem in order to achieve a global solution Sol_i to the integrated problems.

3 The Container Stacking Problem

Containers are ISO standardized metal boxes which can be stacked on top of each other. A container yard (see Figure 4) is composed of several blocks, each one consisting of (20 - 30) yard-bays . Each yard-bay contains several (usually 6) rows and each row has a maximum allowed tier (usually 4 or 5 tiers for full containers).

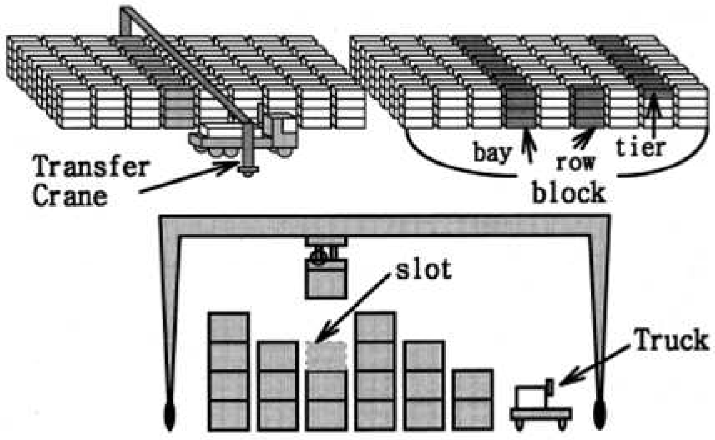


Figure 4: A container yard.

Loading and offloading containers on the stack is performed by cranes following a 'last-in, first-out' (LIFO) criteria. Containers are stacked in the order they arrive. However, in the loading process of vessels, to access a container which is not at the top of its pile, those above it must be relocated. This remarshaling process is required since the stacking order depends on the order in which ships unload or containers have been stacked. This remarshaling process reduces the productivity of cranes and its optimization would minimize the moves required. For safety reasons, it is usually prohibited to move the gantry crane while carrying a container [22], therefore these movements only take place in the same yard-bay. In addition, there exist a set of hard/soft constraints regarding container moves or locations where can be stacked, for example, small differences in height of adjacent yard-bays, dangerous containers must be allocated separately by maintaining a minimum distance, etc. The CStackP is a NP-complete combinatorial optimization problem and different approaches have been proposed ([27], [20], etc.). The CStackP can be viewed, from the artificial intelligence point of view, as a modification of the Blocks World planning domain [33].

In [28], a planning system for remarshaling processes was proposed. This system obtains the optimized plan of reshuffles of containers in order to allocate all selected containers at the top of the stacks, or under another selected containers, in such a way that no reshuffles will be needed to load these outgoing containers. This planner was specified by means of the standard Planning Domain Definition Language (PDDL) [11] and it was developed on the well-known domain-independent planner MetricFF [18]. The developed domain file contains the common features of the problem domain: (i) the domain objects: containers and rows, (ii) the relations among them (propositions), and (iii) allowed moves to change the status of the problem (actions). The problem file describes each particular instance: (i) the initial layout of the containers in the yard (Initial state), (ii) the export containers (goal) which must be allocated at the top of the stacks or under other export containers, and (iii) the function to optimize (minimizing the number of relocation movements). Figure 5 shows a simple example of a problem instance in PDDL. This problem describes a scenario with three containers (X1, X2 and X3) in the yard-bay and the crane is not holding any of them. X2 is an export container which must be relocated in order to be easily accessible at loading time of the next vessel.

```
(define (problem p1) (:domain CStackP)

 ;;Domain objects
 (:objects  X1 X2 X3 - container S1 S2 S3 S4 S5 S6 - slot )

 ;;Initial state
 (:init
(clear X1)(clear X3)(clear S1)(clear S4)(clear S5)(clear S6)
(on X2 S2)(on X1 X2)(on X3 S3)(at X1 S2)(at X2 S2)(at X3 S3)
(= (height S1) 0)(= (height S2) 2)(= (height S3) 1)
(= (height S4) 0)(= (height S5) 0)(= (height S6) 0)
(= (num-moves) 0)(handempty)
(goal-container X2)
 )

 ;;Goal state
 (:goal (and (handempty) (ready X2)) )

 ;;function to optimize
 (:metric minimize (num-moves))
)
```

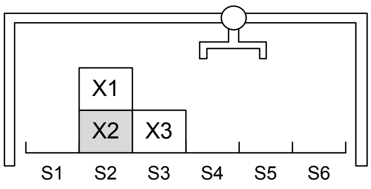


Figure 5: Example of problem instance in Container Stacking domain.

In [29] the Metric-FF-based initial planner was improved by integrating a domain-dependent heuristic (H1) in order to achieve efficiency. H1 computes an estimator of the number of container movements that must be carried out to reach a goal state, which it is used to guide search of solutions. However, new constrains and optimization criteria were included in order to take into account real-world requirements:

C.1. Reducing distance of the outgoing containers to the cargo side (to the left or right hand side of bays).

C.2. Increasing the range of the move actions set for the cranes allowing moving a container to 5th tier (in yard-bays with tiers 4).

C.3. Balancing the number of stacked containers within the same bay in order to avoid sinks, that is, the difference between the number of containers stacked in adjacent positions should be limited).

This planner was improved to manage a full container yard by distributing in small problems. On other issues such as monitoring, distributed approaches have been used [24]. The container yard was decomposed in yard-bays, so that the problem was distributed into a set of subproblems. Thus, each yard-bay generated a subproblem. However, containers of different yard-bays must satisfy a set of constraints among them. Therefore, subproblems were sequentially solved, so that each subproblem (yard-bay) took into account the set of constraints with previously solved subproblems. This decomposition required taking into account these new added constraints. With these new

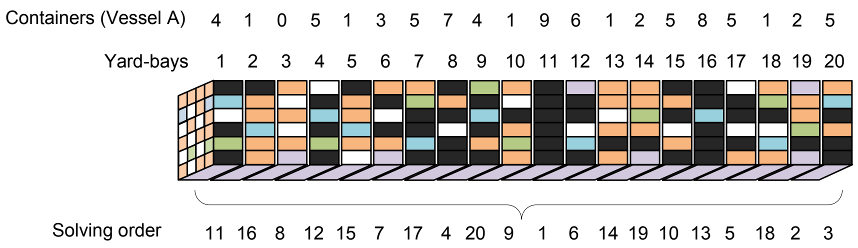

Containers (Vessel A) 4 1 0 5 1 3 5 7 4 1 9 6 1 2 5 8 5 1 2 5
Yard-bays 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Solving order 11 16 8 12 15 7 17 4 20 9 1 6 14 19 10 13 5 18 2 3

Figure 6: Order of execution of yard-bays.

added constraint and criteria, the developed planner could solve more real-world based problems:

1. Balancing contiguous yard-bays: rows of adjacent yard-bays must be balanced in order to avoid sinks inter yard-bays (CB).
2. Dangerous containers must maintain a minimum security (Euclidian) distance among them (DC).

Due to we manage the entire container yard, the requirement C.3 must be extended to balance the stacks of contiguous yard-bays. Thus the planning of a yard-bay is carried out taking into account the solution obtained by the previous yard-bay.

3.1 An Ordered Yard-Based Planner: Planner H1 Ordered

In order to insert our planner in the integrated system, we have also improved our version to minimize the number of reshuffles for a set of outgoing containers to be loaded in successive berthed vessels. Initially our planner was developed to minimize the number of reshuffles for one vessel (vessel A in Figure 6). However, the order of the rest of containers in the yard-bay did not matter. The new planner (Planner H1 Ordered) takes into account these features and it is able to organize the bay in order to adapt to the berth schedule. Thus, the reshuffles needed to allocate the outgoing containers for a vessel are carried out taking into account the outgoing containers for the following vessel to berth.

In the previous version of our planner (Planner H1 Sequential), the order of execution of yard-bays was sequential. Thus a first plan was obtained for the first yard-bay. Then a new plan is carried out for the second yard-bay taking into account the balance constraints generated with the solution obtained for the first yard-bay, and so on.

The new version of our planner classifies the yard-bays by means of tightness. Figure 6 shows the order of execution of a complete yard. A natural order of execution is the sequential order, from the first yard-bay to the end yard-bay (Planner H1 Sequential).

However, in this version (Planner H1 Ordered), the natural order of execution is modified. The tightest yard-bays (yard-bays with more export containers) are solved first. Thus, the number of reshuffles is minimized due to the fact that the tightest yard-bays are solved without the need of satisfying the balance constraints with the contiguous ones, meanwhile their neighbor yard-bays must be committed to these tasks. Thus, following the example of Figure 6, the yard-bay 11 is executed first without balance constraints because it has 9 goal containers meanwhile yard-bays 10 and 12 are executed later taking into consideration the balance constraints generated by the solution of yard-bay 11. In the evaluation section we will compare the behavior of this planner against the previous version. In general the order of execution of yard-bays is directly related with the efficiency of our planning tool. The ordering of yard-bays by tightness improves the efficiency of our planning tool.

3.2 An Analytic Formula to Estimate the number of Reshuffles

As we have pointed out, solving the CStackP is a NP-complete combinatorial optimization problem. Once the BAP returns a possible schedule of vessels to berth in the port, the plan of yard reshuffles must be carried out for each vessel. This is a very hard task so that a general formula that estimates the number of reshuffles for each vessel remains necessary. Once the global solution is achieved, the planning tool is executed to obtain the optimal plan for the yard reshuffles. To this end, we have identified the main parameters that affect the number of reshuffles in a yard-bay for a vessel:

1. Number of total slots in a yard-bay (P_1). For instance, a yard-bay with tier 5 and 6 rows, the number of slots is 30.

2. Number of current containers in the yard-bay (P_2), $P_2 \leq P_1$.

3. Number of goal (outgoing) containers (P_3), $P3 \leq P2$

4. Number of containers on top of goal containers (P_4). A lower bound for the minimal number of reshuffles is P_4.

These parameters influence in the number of reshuffles needed for each yard-bay. The estimator R is mainly depended on P_4 and it is bounded by:

$$R = P_4 + \alpha : \qquad \alpha \in [0, \infty) \tag{2}$$

where $\alpha = 0$ means that the problem is underconstrained meanwhile $\alpha \to \infty$ means that the problem is unsolvable.

Figure 7 shows an example of yard-bay with the value of each identified parameter. The rest of parameters also play an important role but the relationship among them is secondary. The simulation of one hundred yard-bays with different tiers and number of containers and objectives return a strong relationship among the parameters P_1, P_2 and P_3. If $(P_1 - P_2)/P_3$ is lower than 1.25, one more reshuffle is needed ($P_4 + 1$). In the same way if $(P_1 - P_2)/P_3$ is lower than 1, one more reshuffle is also needed $(P_4 + 2)$; and so on. Finally if $(P_1 - P_2) \to 0$ the problem is unsolvable. Thus, we estimate the number of reshuffles by the following formula:

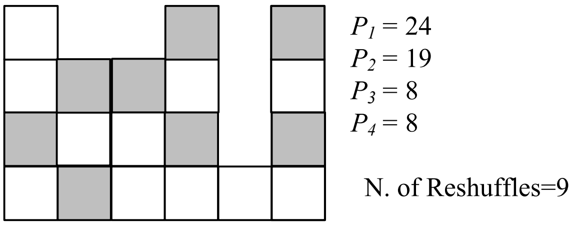


Figure 7: Example and values of parameters. Grey containers are goal containers.

$$R = P_4 + \left\lfloor \frac{2}{\frac{P_1 - P_2}{P_3}} \right\rfloor = P_4 + \left\lfloor \frac{2 \cdot P_3}{P_1 - P_2} \right\rfloor \tag{3}$$

This estimator R is accurate enough for us to do not need to execute our planner for each berthing plan. In the integrated system presented above we can select to run our planner or to use the estimator. In any case, once the best solution is found for the integrated problem, our planner must solve the best solution in order to determine the specific plan which will be carried out by the cranes to allocate the containers in the appropriate places.

4 The Berth Allocation and Quay Crane Assignment Problem

We will focus our attention on the Berth Allocation Problem (BAP), a well-known NP-Hard combinatorial optimization problem, which consists of assigning incoming vessels to berthing positions. Once a vessel arrives at the port, it enters in the harbor waiting time to moor at the quay. The quay is a platform protruding into the water to facilitate the loading and unloading of cargo. The locations where mooring can take place are called berths. These are equipped with giant cranes, called pier or quay cranes (QC), used to load and unload containers which are transferred to and from the yard by a fleet of vehicles. In a transshipment terminal the yard allows temporary storage before containers are transferred to another ship or to another mode (e.g., rail or road).

The BAP is one of the most relevant problems arising in the management of container ports. Several models are usually considered [31].

Managers at container terminals face two interrelated decisions: *where* and *when* the vessels should moor. First, they must take into account physics restrictions as length or draft, but also they have to take into account the priorities and other aspects to minimize both port and user costs, which are usually opposites. Figure 9 shows an example of graphical space-time representation of a berth planning with 6 vessels. Each rectangle represents a vessel with its service time and length.

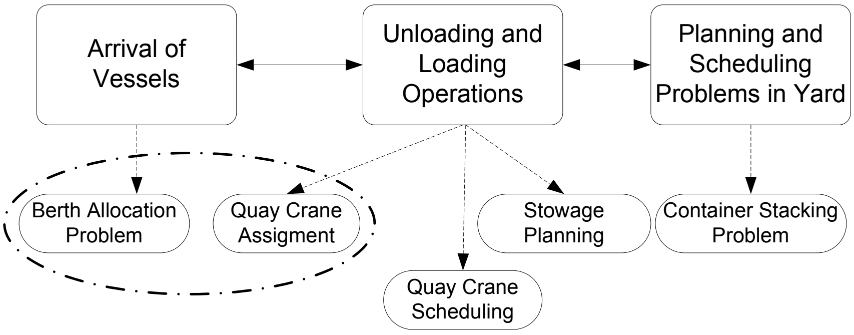


Figure 8: Planning and scheduling problems in Container Terminals.

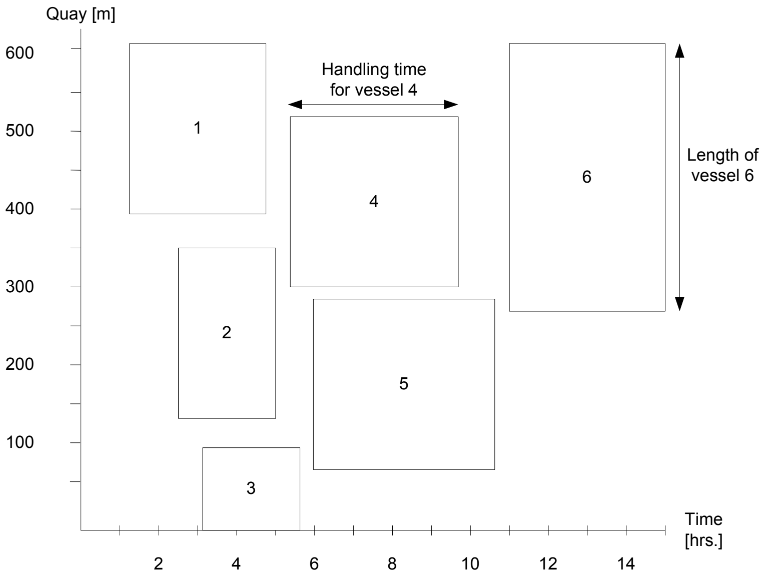


Figure 9: A berth planning.

In [31], the authors show a complete comparative study about different solutions for the BAP according to their efficiency in addressing key operational and tactical questions relating to vessel service. They also study the relevance and applicability of the solutions to the different strategies and contractual service arrangements between terminal operator and shipping lines.

To show similarities and differences in the existing models for berth allocation, Bierwirth and Meisel [3] developed a classification scheme (see Figure 10). They classify the BAP according to four attributes. The spatial attribute concerns the berth layout and water depth restrictions. The temporal attribute describes the temporal constraints for the service process of vessels. The handling time attribute determines the way vessel handling times are considered in the problem. The fourth attribute defines the performance measure to reflect different service quality criteria. The most important ones are minimizing the waiting time and the handling time of a vessel. Both measures aim at providing a competitive service to vessel operators. If both objectives are pursued (i.e. wait and hand are set), the port stay time of vessels is minimized. Other measures are focused on minimizing the completion times of vessels among others. Thus, by using the above classification scheme, a certain type of BAP is described by a selection of values for each of the attributes. For instance, a problem where the quay is assumed to be a continuous line (*cont*). The arrival times restrict the earliest berthing of vessels (*dyn*) and handling times depends on the berthing position of the vessel (*pos*). The objective is to minimize the sum of the waiting times (*wait*) and handling time (*hand*). According to the scheme proposed by [3], this problem is classified by *cont—dyn—pos—*Σ*(wait+hand)*.

Value	Description
1. Spatial attribute	
disc	The quay is partitioned in discrete berths
cont	The quay is assumed to be a continuous line
hybr	The hybrid quay mixes up properties of discrete and continuous berths
draft	Vessels with a draft exceeding a minimum water depth cannot be berthed arbitrarily
2. Temporal attribute	
stat	In static problems there are no restrictions on the berthing times
dyn	In dynamic problems arrival times restrict the earliest berthing times
due	Due dates restrict the latest allowed departure times of vessels
3. Handling time attribute	
fix	The handling time of a vessel is considered fixed
pos	The handling time of a vessel depends on its berthing position
QCAP	The handling time of a vessel depends on the assignment of QCs
QCSP	The handling time of a vessel depends on a QC operation schedule
4. Performance measure	
wait	Waiting time of a vessel
hand	Handling time of a vessel
compl	Completion time of a vessel
speed	Speedup of a vessel to reach the terminal before the expected arrival time
tard	Tardiness of a vessel against the given due date
order	Deviation between the arrival order of vessels and the service order
rej	Rejection of a vessel
res	Resource utilization effected by the service of a vessel
pos	Berthing of a vessel apart from its desired berthing position
misc	Miscellaneous

Figure 10: A classification scheme for BAP formulation [3].

One of the early works that appeared in the literature was [21], in which they developed a heuristic algorithm by considering a First-Come-First-Served (FCFS) rule. However, the idea that for high port throughput, optimal vessel-to-berth assignments should be found without considering the FCFS bases was introduced by [19]. Therefore, we will use the FCFS rule in order to get an upper bound of the function cost in BAP. Nevertheless, this approach may result in some vessels' dissatisfaction regarding the order of service.

In [13], multiple vessel mooring per berth are allowed assuming that vessel arrivals can be grouped into batches. They have developed a tree search procedure which provides an exact solution and this is improved by a composite heuristic.

Metaheuristics have been developed to solve the BAP but have also been used in other fields such as flowshop or flexible job-shop scheduling problems [25, 14]. On the one hand, [8] introduce two Tabu Search heuristics to solve the discrete and continuous case, respectively to minimize is the weighted sum for every ship of the service time in the port. Both heuristics are inspired by a *Multi-Depot Vehicle Routing Problem with Time Windows* algorithm and can handle various features of real-life problems as time windows or favorite and acceptable berthing areas. Mauri et al. [23] design a column generation approach for the problem of Cordeau et al. [8] which delivers better solutions in shorter runtime than Tabu Search. In the models of Han et al. [15] and Zhou et al. [34], a Genetic Algorithm (GA) is proposed to solve the problem. In both models the draft of vessels restricts the berth assignment decisions.

An approach based on multi-objective optimization problem using evolutionary algorithms ([6]) is followed to minimize the makespan of the port, total waiting time of the ships, and degree of deviation from a predetermined service priority schedule.

In [12], they present the integration of BAP with the Quay Crane Assignment Problem (QCAP) through two mixed integer programming formulations including a tabu search method which is an adaption of the one of [8], however they minimize the yard-related housekeeping costs generated by the flows of containers exchanged between vessels. In [3], the authors give a comprehensive survey of berth allocation and quay crane assignment formulations from the literature. Some authors outline approaches more or less informally while others provide precise optimization models. More than 40 formulations are presented distributed among discrete problems, continuous problems and hybrid problems. Hansen et al. [16] considered a discrete problem with a tardiness objective which accounts for departure time related costs including penalties for tardiness as well as benefits for early departures. This problem was solved by a variable neighborhood search which turns out to be superior to the GA of Nishimura et al. [26].

Nowadays, this process is generally solved manually and it is usually solved by means of a policy to serve the first vessel arrives.

Considering the requirements of container operators of MSC (Mediterranean Shipping Company S.A), our approach also studies the integration of these two problems (BAP and QCAP) through a metaheuristic called Greedy Randomized Adaptive Search Procedures (GRASP) [9]. This metaheuristic is able to find feasible solutions within an acceptable computational time. In this way and following the above classification scheme (see Figure 10), our approach is classified by $cont|dyn|QCAP|\Sigma wait$. Thus, we focused on the following attributes and performance measure:

- *Spatial attribute: cont*: we assume the quay is a continuous line, so that there is no partitioning of the quay and vessel can berth at arbitrary positions within the boundaries of the quay. It must be taken into account that for a continuous layout, berth planning is more complicated than for a discrete layout at the advantage of better utilizing quay space [3].

- *Temporal attribute: dyn*: we assume dynamic problems where arrival times restrict the earliest berthing times. Thus, fixed arrival times are given for the vessels, hence, vessels cannot berth before their expected arrival time.
- *Handling time attribute: QCAP*: we assume the handling time of a vessel depends on the assignment of QCs.
- *Performance measure: wait*: Our objective is to minimize the sum of the waiting time of all vessels.

The objective in BAP is to obtain an optimal distribution of the docks and cranes to vessels waiting to berth. Thus, this problem could be considered as a special kind of machine scheduling problem, with specific constrains (length and depth of vessels, ensure a correct order for vessels that exchange containers, assuring departing times, etc.) and optimization criteria (priorities, minimization of waiting and staying times of vessels, satisfaction on order of berthing, minimizing cranes moves, degree of deviation from a pre-determined service priority, etc.).

4.1 Notation for BAP and QCAP

Our approach follows an integration of the Quay Crane Assignment Problem (QCAP) and the BAP through the metaheuristic Greedy Randomized Adaptive Search Procedure (GRASP) [9] which is able to obtain optimized solutions in a very efficient way. Following, we introduce the used notation:

Q_C Available QCs in the Container Terminal. These QCs are identical in terms of productivity in loading/discharging containers.

L Total length of the berth in the Container Terminal.

$a(V_i)$ Arrival time of the vessel V_i at port.

$m(V_i)$ Moored time of V_i. All constraints must hold.

$pos(V_i)$ Berthing position where V_i will moor.

$c(V_i)$ Number of required movements to load and unload containers of V_i.

$q(V_i)$ Number of assigned QCs to V_i. The maximum number of assigned QC by vessel depends on its length since a security distance is required between two contiguous QC (`secQC`) and the maximum number of cranes that the Container Terminal allows per vessel (`maxQC`). Let's assume that the number of QC does not vary along all the moored time. Thus, the handling time of V_i is given by (where `movsQC` is the QC's moves per unit time):

$$\frac{c(V_i)}{q(V_i) \times \texttt{movsQC}} \tag{4}$$

$d(V_i)$ Departure time of V_i, which depends on $m(V_i)$, $c(V_i)$ and $q(V_i)$.

$w(V_i)$ Waiting time of V_i from it arrives at port until it moors: $w(V_i) = m(V_i) - a(V_i)$.

$l(V_i)$ Length of V_i. There is a distance security (`secLength`) between two moored ships: let's assume 5% of their lengths.

$pr(V_i)$ Vessel's priority.

In order to simplify the problem, let's assume that mooring and unmooring does not consume time and every vessel has a draft lower or equal than the quay. In each case, simultaneous berthing is allowed.

The goal of the BAP is to allocate each vessel according existing constraints and to minimize the total weighted waiting time of vessels:

$$T_w = \sum_i w(V_i)^\gamma \times pr(V_i) \tag{5}$$

The parameter γ ($\gamma \geq 1$) prevents lower priority vessels are systematically delayed. Note that this objective function is different to the classical tardiness concept in scheduling.

4.2 A meta-heuristic method for BAP and QCAP

We have developed three different methods for solving BAPs. Firstly, we applied the simplest solution, following the FCFS criteria: $\forall i, m(V_i) \leq m(V_{i+1})$. A vessel can be allocated at time t when there is no vessel moored in the berth or there are available contiguous quay length and QCs at time t (Algorithm 1).

Secondly, we also implemented a complete search algorithm to obtain the best (optimal) mooring order of vessels: the lowest T_w (lower bound of the function cost).

Finally, we developed a meta-heuristic GRASP algorithm for berth allocation (Algorithm 6). This is a randomly-biased multistart method to obtain optimized solutions of hard combinatorial problems in a very efficient way.

Figure 11 shows the application order of the different algorithms presented in this paper.

Algorithm 1 (FCFS) uses the function *insertVessel* described in Algorithm 2 to allocate one vessel in a given time t (the required data are: V_i: Vessel for allocating; t: time for mooring the vessel; V_{in}: set of vessels already moored). If V_i can not moor at time t, it is repeated again each time one vessel (already moored) departs.

Algorithm 2 assigns the number of QCs ($q(V_i)$) to the vessel V_i through the function *assignQC* (Algorithm 3). This function takes into account the length of V_i, the security

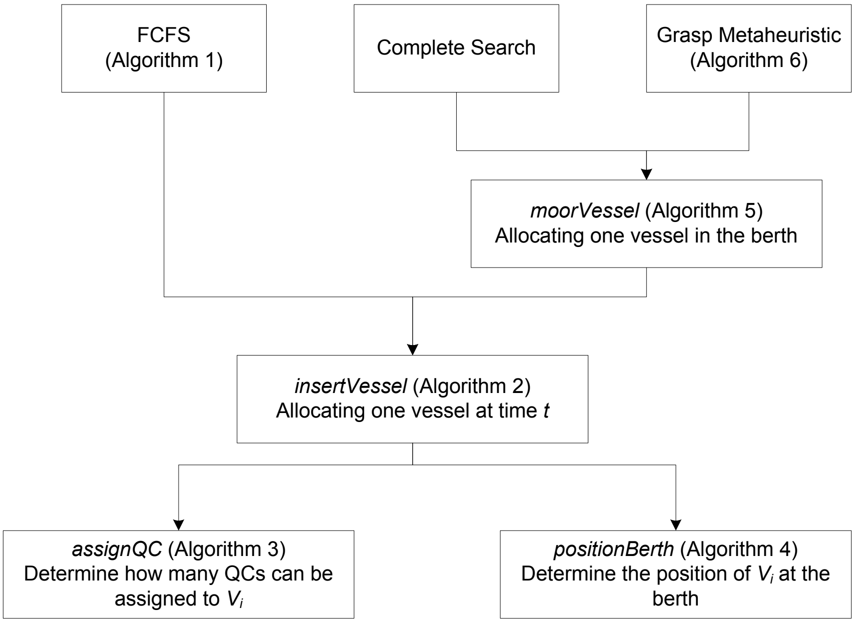

FCFS
(Algorithm 1)
Complete Search
Grasp Metaheuristic
(Algorithm 6)
moorVessel (Algorithm 5)
Allocating one vessel in the berth
insertVessel (Algorithm 2)
Allocating one vessel at time *t*
assignQC (Algorithm 3)
Determine how many QCs can be assigned to *V_i*
positionBerth (Algorithm 4)
Determine the position of *V_i* at the berth

Figure 11: Application order of the algorithms presented.

Algorithm 1: Allocating vessels following FCFS policy

```
   Data: $V$: set of ordered incoming vessels;
   Result: Sequence for $V$
1  $V_{last} \leftarrow \varnothing$;
2  $V_m \leftarrow \varnothing$;
3  foreach $V_i \in V$ do
4      $t \leftarrow \max(m(V_{last}), a(V_i))$;
5      $inst \leftarrow insertVessel(V_i, t)$;
6      if $!inst$ then
7          $T \leftarrow d(V_j) \mid V_j \in V_m \wedge d(V_j) > t$;
8          while $t_k \in T \wedge !inst$ do
9              $inst \leftarrow insertVessel(V_i, t_k)$;
10         end
11     end
12     $V_{last} \leftarrow V_i$;
13     $V_m \leftarrow V_m \cup V_i$;
14 end
```

distance between two QCs (`secQC`) as well as the QCs used by the others moored vessels. Furthermore, Algorithm 2 obtains the berthing position ($pos(V_i)$) by the function *PositionBerth* (Algorithm 4). In order to get this position, we look for the first position where the vessel V_i fits according to its length ($l(V_i)$) and the security length between two contiguous vessels (`secLength`).

In addition to the FCFS algorithm, the Complete Search was developed. The Com-

Algorithm 2: Function insertVessel. Allocating one vessel in the berth at time t

```
    Data: $V_i$: Vessel for allocating; $t$: actual time; $V_{in}$: moored vessels;
    Result: $V_i$ could moor
1   if $V_{in} = \varnothing$ then
2       $m(V_i) \leftarrow a(V_i)$;
3       $q(V_i) \leftarrow \min(\texttt{maxQC}, \frac{l(V_i)}{\texttt{secQC}})$;
4       $d(V_i) \leftarrow m(V_i) + \frac{c(V_i)}{q(V_i) \times \texttt{movsQC}}$;
5       return true;
6   else
7       $q(V_i) \leftarrow \texttt{assignQC}(V_i, t, V_{in})$;
8       if $q(V_i) = 0$ then
9           return false;
10      end
11      $m(V_i) \leftarrow t$;
12      $d(V_i) \leftarrow t + \frac{c(V_i)}{q(V_i) \times \texttt{movsQC}}$;
13      $pos(V_i) \leftarrow \texttt{positionBerth}(V_i, V_{in})$;
14      if $pos(V_i) < 0$ then
15          return false;
16      else
17          return true;
18      end
19  end
```

plete Search uses the functions *moorVessel* (Algorithm 5) which in turn also uses the function *insertVessel* (Algorithm 2) to allocate one vessel from its arrival time. The required data for the Algorithm 5 is: v: Vessel for allocating; V_{in}: set of vessels already moored. However, with this search only a limited number of vessels can be taken into account since search space grows exponentially.

Therefore, we developed the meta-heuristic GRASP algorithm for berth allocation (Algorithm 6). Algorithm 6 is guided by the parameter δ ($0 \leq \delta \leq 1$) which allows tuning of search randomization.

Algorithm 6 receives as parameters both the δ factor and the set of vessels V_{out} waiting for mooring at the berth. Firstly, all the waiting vessels V_{out} are considered as candidates C. Each one of the candidate vessels is moored within the same state of the berth (Algorithm 5) assigning the mooring and departure times ($m(V_i)$,$d(V_i)$), the number of QCs ($q(V_i)$) and the berthing position ($pos(V_i)$). These possibilities are evaluated according to the cost function f_c. This cost function is the sum of T_w that each vessel causes to the rest of unmoored vessels.

According to the cost function f_c, a restricted candidate list (RCL) is created. Then, one vessel v is chosen to be moored definitely following the random degree indicated by δ factor. Once v is determined, this is added to the set of vessels V_{in} and eliminated from the candidate list C. This loop is repeated until C is empty, it means, all the vessels are moored.

As metaheuristic GRASP indicates, this search is repeated according to the number of iterations specified by the user. Thus, the best solution according to T_w is returned as the solution for the BAP.

Algorithm 3: Function assignQC. Determine how many QCs can be assigned to V_i.

```
   Data: $V_i$: Vessel for allocating; $t$: actual time; $V_{in}$: moored already vessels;
   Result: Number of QCs
1  $cranes \leftarrow -1; cranes_m \leftarrow -1;$
2  repeat
3      $nc \leftarrow \max(1, cranes);$
4      $t_f \leftarrow t + \frac{c(V_i)}{nc \times \texttt{movsQC}};$
5      $cranes_m \leftarrow nc;$
6      $cranes \leftarrow \max\left(1, \min\left(\texttt{maxQC}, \texttt{floor}\left(\frac{l(V_i)}{\texttt{secQC}}\right)\right)\right);$
       // Vessels which coincide with $V_i$
7      $W \leftarrow v \in V_{in} \mid d(v) > t \wedge m(v) < t_f;$
8      foreach $v \in W$ do
           // $QC_m$ is the number of used cranes when $v$ moors
9          $QC_m \leftarrow \sum_{v' \in W} q(v') \mid m(v) \geq t \wedge m(v') \leq m(v) \wedge d(v') > m(v);$
           // $QC_d$ is the number of used cranes when $v$ departs
10         $QC_d \leftarrow \sum_{v' \in W} q(v') \mid d(v) \leq t_f \wedge m(v') < d(v) \wedge d(v') \geq d(v);$
11         $cranes \leftarrow \min(cranes, Q_C - QC_m);$
12         $cranes \leftarrow \min(cranes, Q_C - QC_d);$
13     end
14     if $cranes \leq 0$ then
15         return $0$;
16     end
17 until $cranes_m = cranes$ ;
18 return $cranes$;
```

Algorithm 4: Function positionBerth. Determine the exact position of V_i at the berth.

```
   Data: $V_i$: Vessel for allocating; $V_{in}$: moored already vessels;
   Result: Berthing position for $V_i$
   // Set of vessels which coincide during the stay of $V_i$
1  $W' \leftarrow v \in V_{in} \mid d(v) \leq m(V_i) \wedge m(v) \geq d(V_i);$
2  $W \leftarrow V_{in} - W';$
3  sortByPositionBerth($W$);
4  $lastPos \leftarrow 0; lastLength \leftarrow 0;$
5  foreach $v \in W$ do
       // Security lengths
6      $dLeft \leftarrow \max(lastLength, l(V_i)) \times \texttt{secLength};$
7      $dRight \leftarrow \max(l(v), l(V_i)) \times \texttt{secLength};$
8      $freeDist \leftarrow (lastPos + dLeft) - (pos(v) - dRight);$
9      if $freeDist \geq l(V_i)$ then
           // Assigning first free position
10         return $lastPos + dLeft$;
11     end
12     $lastPos \leftarrow pos(v); lastLength \leftarrow l(v);$
13 end
   // From the last vessel to the end of the berth
14 $dLeft \leftarrow \max(lastLength, l(V_i)) \times \texttt{secLength};$
15 $freeDist \leftarrow (lastPos + dLeft) - L;$
16 if $freeDist \geq l(V_i)$ then
17     return $lastPos + dLeft$;
18 end
19 return $-1$;
```

Algorithm 5: Function moorVessel. Allocating exactly one vessel in the berth.

```
   Data: $v$: vessel to moor; $V_{in}$: moored vessels;
1  $inst \leftarrow insertVessel(v, a(v))$;
2  if $!inst$ then
3      $T \leftarrow d(v_j) \mid v_j \in V_{in} \wedge d(v_j) > t$;
4      while $t_k \in T \wedge !inst$ do
5          $inst \leftarrow insertVessel(v, t_k)$;
6      end
7  end
```

Algorithm 6: Allocating Vessels using GRASP metaheuristic

```
   Data: $\delta$ factor; $V_{out}$ elements; $b$: state of the berth;
   Result: $V_{in}$: set of scheduled vessels;
1  $V_{in} \leftarrow \{\}$;
2  $C \leftarrow V_{out}$;
3  while $C \neq \varnothing$ do
4      foreach $v_e \in V_{out}$ do
5          $f_c(v_e) \leftarrow 0$;
6          $moorVessel(v_e, V_{in})$;
7          $V'_{in} \leftarrow V_{in} \cup \{v_e\}$;
8          foreach $v_o \in V_{out} \mid v_o \neq v_e$ do
9              $moorVessel(v_o, V'_{in})$;
10             $f_c(v_e) \leftarrow f_c(v_e) + (w(v_o) \times pr(v_o))$;
11         end
12     end
13     $c_{inf} \leftarrow \min\{f_c(e) \mid e \in C\}$;
14     $c_{sup} \leftarrow \max\{f_c(e) \mid e \in C\}$;
15     $RCL \leftarrow \{e \in C \mid f_c(e) \leq c_{inf} + \delta(c_{sup} - c_{inf})\}$;
16     $v \leftarrow random(RCL)$;
17     $moorVessel(v, V_{in})$;
18     $V_{in} \leftarrow V_{in} \cup \{v\}; C \leftarrow C - \{v\}$;
19 end
```

This metaheuristic process does not include a local search technique since it would involve testing the possible exchanges between the already ordered vessels, so that the computational cost would be increased considerably.

5 Evaluation

In this section, we evaluate the behavior of the algorithms developed in the paper. The experiments were performed on random instances. For the CStackP, containers are randomly distributed in blocks of 20 yard-bays, each one with six stacks of 4 tiers. A random instance of a yard-bay is characterized by the tuple $< n, s >$, where *'n'* is the number of containers and *'s'* ($s \leq n$) is the number of selected containers in the yard-bay. A random instance for the BAP has *'k'* vessels with an arrival exponential distribution with vessel's data randomly fixed (lengths, drafts, moves and priorities).

In Table 1, we show the performance of our planner H1 Sequential and H1 Ordered.

These experiments were performed on blocks of containers composed by 10 yard-bays in the form $< 17, s >$. Thereby, each yard-bay has a different number of goal containers. Additionally, we have established a timeout of 35 seconds to solve each yard-bay.

Table 1: Comparison of H1 Sequential and H1 Ordered.

Inst.	**Order (Yard-Bay/N. Goal containers)**										**Number**
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Reshuffles**
1-O	6	1	2	8	3	4	10	9	5	7	91
	9	7	6	6	5	5	5	4	2	2	
1-S	1	2	3	4	5	6	7	8	9	10	*Time Out*
	7	6	5	5	2	9	2	6	4	5	
2-O	2	5	6	1	4	3	8	9	10	7	64
	7	7	7	6	6	5	4	4	3	2	
2-S	1	2	3	4	5	6	7	8	9	10	*Time Out*
	6	7	5	6	7	7	2	4	4	3	
3-O	3	2	7	8	10	4	6	1	9	5	55
	8	6	6	6	6	5	5	4	4	1	
3-S	1	2	3	4	5	6	7	8	9	10	99
	4	6	8	5	1	5	6	6	4	6	
4-O	7	9	2	10	3	4	6	1	8	5	63
	10	7	6	6	5	5	4	3	3	2	
4-S	1	2	3	4	5	6	7	8	9	10	63
	3	6	5	5	2	4	10	3	7	6	
5-O	7	1	10	2	8	9	4	5	6	3	78
	9	7	7	6	6	6	5	5	4	2	
5-S	1	2	3	4	5	6	7	8	9	10	*Time Out*
	7	6	2	5	5	4	9	6	6	7	

The first column in Table 1 corresponds to each instance solved by each planner. Thereby, row 1 corresponds to instance 1 of planner H1 Ordered (1-O), row 2 corresponds to instance 1 of planner H1 Sequential (1-S), and so on. The following 10 columns have two rows for each instance. They show for each instance the order in which the yard-bays are executed (upper row) and the number of goal containers that each one of them has (lower row). As example, for the instance 1-O, the sixth yard-bay is the first one in being executed and it has 9 goal containers. Finally, the last column presents the number of reshuffles needed to solve the instance or *Time Out* if a solution is not found.

As it can be observed, there are instances which only can be solved through the H1

Ordered, e.g. instances 1, 2 and 5. Moreover, other instances (instance 3), through the H1 Ordered planner, give a more efficient plan than the sequential one. However, there are also other examples in which both planners return the same plan (instance 4).

Thus, we can conclude that H1 Ordered can be considered a better planner than H1 Sequential to solve the complete block of yard-bays.

The actual average number of reshuffles and the average value of our estimator R are presented in Table 2. In each row, we present the average number of reshuffles form a set of 100 random instances. In all cases, we considered yard-bays with 4 tiers, so that the number of possible containers to be allocated to each yard-bay is set to 24 ($P_1 = 24$). The rest of parameters were increased: P_2 from 15 to 19 and P_3 from 4 to 8. It can be observed the similitude of the average number of reshuffles and R in all cases. It can be observed that the estimator R achieved values close to the actual values in all cases. The average value of R in all instances was very similar to the average value of the actual number of reshuffles. The standard deviation of R was even lower than the actual number of reshuffles due to the fact that it does not depend on the original allocation of out containers.

Table 2: Values obtained through estimator R.

	$\mathbf{P_2 = 15}$		$\mathbf{P_2 = 17}$		$\mathbf{P_2 = 19}$	
	Reshuffles	**R**	**Reshuffles**	**R**	**Reshuffles**	**R**
$\mathbf{P_3 = 4}$	2.8	2.6	3.7	4.6	4.6	4.8
$\mathbf{P_3 = 6}$	4.2	4.8	6.2	5.9	5.6	6
$\mathbf{P_3 = 8}$	6.2	6	8.6	8	7.2	8
Average values	4.4	4.5	6.2	6.2	5.8	6.3
St. deviation	2.1	1.9	3.1	2.1	2.6	1.9

Table 3 shows the computational times (in seconds) required for solving BAP by using a complete search against the GRASP method with 1000 iterations. As observed, complete search is impracticable from 12 vessels (more than 3 hours). However, the GRASP method takes around 30 seconds to solve a schedule of 20 vessels.

Table 3: Computing time elapsed (seconds) for BAP.

No. vessels	5	10	11	12	13	15	20
Complete search	<1	112	1105	11830	57462	-	-
Grasp	1	8	9	10	12	15	30

Table 4 shows the average waiting times using FCFS and Complete Search (CS) methods described for the BAP, with two different inter-arrival distributions (temporal separation among arriving vessels). Through these data, it is demonstrated that FCFS criteria results a schedule which is far away from the best one (CS). Besides, it can be observed that the denser the arrivals of the vessels are, the longer the waiting times are for such vessels. For example, given 10 vessels, the value of T_w is 136.26 with separate arrival times and it becomes 351.25 with close arrival times.

Table 4: Total waiting time elapsed.

Vessels	**FCFS**	**CS**
5 (separate arrival times)	73.72	46.10
10 (separate arrival times)	256.53	136.26
5 (close arrival times)	117.52	80.25
10 (close arrival times)	586.65	351.25

Using as minimization function the total weighted waiting time (T_w), Figure 12 shows the results given by the FCFS criteria, and the GRASP procedure (with 1000 iterations) respect to the value of δ. The optimum value is $\delta = 0.3$, which indicates the suitability of the cost function used in the GRASP procedure (Algorithm 6). A total of 20 vessels are allocated, with two different inter-arrival distributions (separate and closest arrival times) among them.

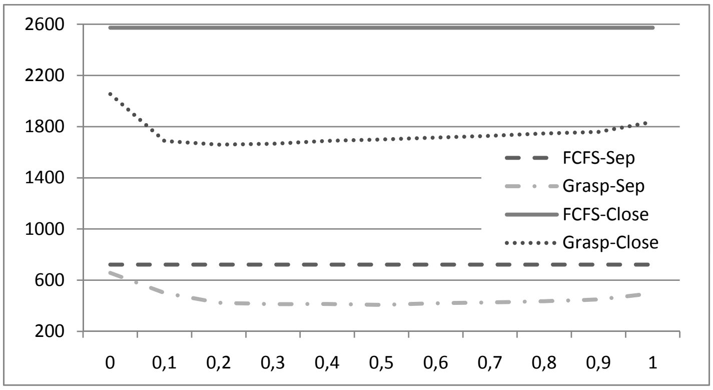


Figure 12: Weighted waiting time (T_w) with FCFS and GRASP procedures.

As it was expected, the GRASP procedure obtains a lower T_w than the FCFS criteria. It is also remarkable that using GRASP is more profitable when the inter-arrival distribution of the vessels is closer. It is not possible to know the optimal T_w due to the exponential computational time required by a complete search with 20 vessels.

Finally, Figure 13 shows the combined function cost Cost(Sol_i), introduced in Equation 1 which relates for ten different scenarios:

- The normalized total weighted waiting time of vessels, Cost($SBAP_i$), and

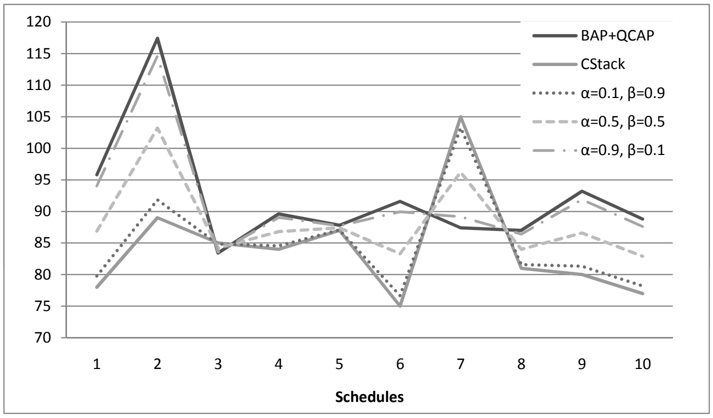


Figure 13: Relating the costs of BAP+QCAP and CStackP.

- The number of its required container relocations, Cost($SCStackP_i$).

In each one of this ten cases, the arrival times and data of vessels, as well as the initial state of the container yard, have been randomly generated. Figure 13 represents the combined function cost, Cost(Sol_i) with three different weights of the parameters α and β. We can see that better (or worst) berthing orders can require larger (or smaller) number of container relocations. For instance, with $\alpha = 0.5, \beta = 0.5$ the best choice is the sixth schedule. It does not get the best solution for BAP+QCAP, however it corresponds to the schedule with the smallest number of container relocations (CStackP).

6 Conclusions

The Container Stacking Problem, the Berth Allocation Problem and the Quay Crane Assignment Problem are three important and related problems in maritime container terminals. In this paper, we have presented a decision support system to manage these problems in a coordinated way. To this end, we have developed a domain-oriented heuristic planner for calculating the number of reshuffles needed to allocate the containers in the appropriate place. Furthermore, we have estimated this number in order to avoid the planning phase in unnecessary instances. Then, we have developed a GRASP metaheuristic for the Berth Allocation Problem and the Quay Crane Assignment Problem which generates an optimized order of vessels to be served according to existing berth constraints. By combining these optimized solutions in our integrated system, terminal operators can be assisted to decide the most appropriated solution in each particular case.

As future work, we are working on improving the function to estimate the number of reshuffles by adding some more specific parameters. Furthermore we plan to improve the GRASP method and to adequate the parameters (α, β and γ) to practical decisions and expert knowledge. Then, the developed system, as a computer-based aid system, could assist container terminal's operator to simulate, evaluate and compare different feasible alternatives.

Acknowledgment

This work has been partially supported by the research projects TIN2010-20976-C02-01 (Min. de Educación y Ciencia, Spain-FEDER), and P19/08 (Min. de Fomento, Spain-FEDER), in the ValI+D Program of the Conselleria d'Educació (Generalitat Valenciana), as well as with the collaboration of the maritime container terminal MSC (Mediterranean Shipping Company S.A.).

References

[1] Global container terminal operators annual review and forecast. *Annual Report*, 2010.

[2] M. Aleb-Ibrahimi, B. de Castilho, and C.F. Daganzo. Storage space vs handlingwork in container terminals. *Transportation Research-B*, 27B:13–32, 1993.

[3] C. Bierwirth and F. Meisel. A survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research*, 202:615–627, 2010.

[4] A. Bruzzone and R. Signorile. Simulation and genetic algorithms for ship planning and shipyard layout. *Simulation*, 71:74–83, 1998.

[5] S.H. Chen and J.N. Chen. Forecasting container throughputs at ports using genetic programming. Expert Systems with Applications, 37(3):2054 – 2058, 2010.

[6] C.Y. Cheong, K.C. Tan, and D.K. Liu. Solving the berth allocation problem with service priority via multi-objective optimization. In *Computational Intelligence in Scheduling, 2009. CI-Sched '09. IEEE Symposium on*, pages 95 –102, 2 2009-march 30 2009.

[7] T.N. Chuang, C.T. Lin, J.Y. Kung, and M.D. Lin. Planning the route of container ships: A fuzzy genetic approach. Expert Systems with Applications, 37(4):2948 – 2956, 2010.

[8] J.F. Cordeau, G. Laporte, P. Legato, and L. Moccia. Models and tabu search heuristics for the berth-allocation problem. *Transportation science*, 39(4):526–538, 2005.

[9] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, 1995.

[10] L.M. Gambardella, A.E. Rizzoli, and M. Zaffalon. Simulation and planning of an intermodal container terminal. *Simulation*, 71:107–116, 1998.

[11] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. *AIPS-98 Planning Committee*, 1998.

[12] G. Giallombardo, L. Moccia, M. Salani, and I. Vacca. Modeling and solving the tactical berth allocation problem. *Transportation Research Part B: Methodological*, 44(2):232 – 245, 2010.

[13] Y. Guan and R.K. Cheung. The berth allocation problem: models and solution methods. *OR Spectrum*, 26(1):75–92, 2004.

[14] C. Gutiérrez and I. García-Magariño. Modular design of a hybrid genetic algorithm for a flexible job-shop scheduling problem. *Knowledge-Based Systems*, 24(1):102 – 112, 2011.

[15] M. Han, P. Li, and J. Sun. The algorithm for berth scheduling problem by the hybrid optimization strategy GASA. *Proceedings of the Ninth International Conference on Control, Automation, Robotics and Vision (ICARCV 2006)*, pages 1–4, 2006.

[16] P. Hansen, C. Ceyda Oguz, and N. Mladenovic. Variable neighborhood search for minimum cost berth allocation. *European Journal of Operational Research*, 191:636–649, 2008.

[17] L. Henesey. Overview of Transshipment Operations and Simulation. In *MedTrade conference, Malta, April*, pages 6–7, 2006.

[18] Jörg Hoffmann. The metric-ff planning system: translating ”ignoring delete lists” to numeric state variables. *J. Artif. Int. Res.*, 20(1):291–341, 2003.

[19] A. Imai, K.I. Nagaiwa, and TAT Chan Weng. Efficient planning of berth allocation for container terminals in Asia. *Journal of advanced transportation*, 31(1):75–94, 1997.

[20] K.H. Kim and G.P. Hong. A heuristic rule for relocating blocks. *Computers & Operations Research*, 33(4):940–954, 2006.

[21] KK Lai and K. Shih. A study of container berth allocation. *Journal of Advanced Transportation*, 26(1):45–60, 1992.

[22] Yusin Lee and Nai-Yun Hsu. An optimization model for the container pre-marshalling problem. *Computers & Operations Research*, 34(11):3295 – 3313, 2007.

[23] G. Mauri, A. Oliveira, and L. Lorena. A hybrid column generation approach for the berth allocation problem. *Eighth European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP 2008)*, 4972:110–122, 2008.

[24] R. Micalizio and P. Torasso. On-line monitoring of plan execution: A distributed approach. *Knowledge-Based Systems*, 20(2):134 – 142, 2007.

[25] B. Naderi, R. Tavakkoli-Moghaddam, and M. Khalili. Electromagnetism-like mechanism and simulated annealing algorithms for flowshop scheduling problems minimizing the total weighted tardiness and makespan. *Knowledge-Based Systems*, 23(2):77 – 85, 2010.

[26] E. Nishimura, A. Imai, and S. Papadimitriou. Berth allocation planning in the public berth system by genetic algorithms. *European Journal of Operational Research*, 131:282–292, 2001.

[27] K. Park, T. Park, and K.R. Ryu. Planning for remarshaling in an automated container terminal using cooperative coevolutionary algorithms. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1098–1105. ACM, 2009.

[28] M. Salido, O. Sapena, and F. Barber. The container stacking problem: an artificial intelligence planning-based approach. *The International Workshop on Harbour, Maritime and Multimodal Logistics Modelling and Simulation*, 2009.

[29] M. Salido, O. Sapena, M. Rodriguez, and F. Barber. A planning tool for minimizing reshuffles in containers terminals. *ICTAI 2009: 21st International Conference on Tools with Artificial Intelligence*, 2009.

[30] R. Stahlbock and S. Voß. Operations research at container terminals: a literature update. *OR Spectrum*, 30(1):1–52, 2008.

[31] S. Theofanis, M. Boile, and M.M. Golias. Container terminal berth planning. *Transportation Research Record: Journal of the Transportation Research Board*, 2100(-1):22–28, 2009.

[32] I.F.A. Vis and R. De Koster. Transshipment of containers at a container terminal: an overview. *European Journal of Operational Research*, 147:1–16, 2003.

[33] Terry Winograd. *Procedures as a representation for data in a computer program for understanding natural language*. MIT. Cent. Space Res., Cambridge, MA, 1971.

[34] P. Zhou, H. Kang, and L. Lin. A dynamic berth allocation model based on stochastic consideration. *Proceedings of the Sixth World Congress on Intelligent Control and Automation (WCICA 2006)*, 2006.

2.4 Robust Scheduling: Robust Berth Allocation and Quay Crane Assignment Problem

Robust scheduling for Berth Allocation and Quay Crane Assignment Problem

M. Rodriguez-Molins, M.A. Salido, F. Barber
Instituto de Automática e Informática Industrial
Universidad Politécnica de Valencia.
Valencia, Spain

Abstract

Decision makers must face the dynamism and uncertainty of real-world environments when they need to solve the scheduling problems. Different incidences or breakdowns, e.g. initial data could change or some resources could become unavailable, may eventually cause the infeasibility of the obtained schedule. To overcame this issue, a robust model and a proactive approach is presented for scheduling problems without any previous knowledge about incidences. This paper is based on proportionally distributing operational buffers among the tasks. In this paper, we consider the Berth Allocation Problem and the Quay Crane Assignment Problem as a representative example of scheduling problems.

The dynamism and uncertainty is managed by assessing the robustness of the schedules. The robustness is introduced by means of operational buffer times to absorb those unknown incidences or breakdowns. Therefore, this problem becomes a multi-objective combinatorial optimization problem that aims to minimize the total service time, to maximize the buffer times and to minimize the standard deviation of the buffer times. To this end, a mathematical model and a new hybrid multi-objective metaheuristic is presented and compared with two well-known multi-objective genetic algorithms: NSGAII and SPEA2+.

Keywords Scheduling, Robustness, Metaheuristics, Multi-Objective Genetic Algorithms, Local Search, Simulated Annealing

1 Introduction

Within a container terminal, operations related to move containers can be divided into four different subsystems (ship-to-shore, transfer, storage and delivery/receipt) [15]. In each subsystem, terminal operators must deal with with different complex optimization problems that can be overcome by using Artificial Intelligence techniques. For instance, berthing allocation or stowage planning problems are related to the ship-to-shore area [17, 16, 28, 27]; remarshalling problem and transport optimization [24] to the storage and transfer subsystems, respectively; and, planning and scheduling hinterland operations related to trains and trucks to the delivery/receipt subsystem [29].

In this paper, we focus on two problems related to the ship-to-shore area, the Berth Allocation Problem (BAP) and the Quay Crane Assignment Problem (QCAP). The former is a well-known combinatorial optimization problem [22], which consists in assigning berthing positions and mooring times to incoming vessels. The QCAP deals with assigning a certain number of Quay Cranes (QCs) to each moored vessel such that all required movements of containers can be fulfilled [2].

A comprehensive survey of BAP and QCAP is given in [2]. These problems have been mostly considered separately, with an interest mainly focused on BAP. An interesting approach for BAP is presented by [18] where a Simulated Annealing metaheuristic is compared with a mathematical model. However, there are some studies on the combined BAP+QCAP considering different characteristics of berths and cranes [11, 21, 7, 25, 33].

Most of the research in scheduling has been focused on deterministic and complete information, but they are usually not satisfied in real-world environments. Due to the fact that the real world is uncertain, imprecise and non-deterministic, there might be unknown information, breakdowns, incidences or changes, which make the initial plans or the obtained schedules become invalid. Thus, there are new trends to cope these aspects in the optimization techniques: proactive and reactive approaches [20]. In this paper, a proactive approach is studied within the Berth Allocation and the Quay Crane Assignment problems. The uncertainty within these problems is due to lower movements per time unit than expected or engine failures in Quay Cranes, among others. Due to the introduction of this new objective in the scheduling optimization problem, a multi-objective optimization approach needs to be taken into consideration.

All the above studies do not take into consideration the uncertainty of the real world to obtain a robust scheduling. Robustness is a measure of the performance characterization of an algorithm in the presence of uncertainties [3]. However, there are some studies that address the robust scheduling. In [14], a robust optimization model for cyclic berthing for a continuous and dynamic BAP is studied by minimizing the maximal crane capacity over different arrival scenarios of a bounded uncertainty given by their arrival agreements. In [12], a proactive approach for a discrete and dynamic model of the BAP is presented taking into account uncertainties in the arrival and handling times given their probability density functions. They propose a mixed integer programming model and a Genetic Algorithm (GA) for both problems: discrete berth allocation and QC assignment. The objective is to minimize the sum of expected value, the standard deviation of the service time and the tardiness of the incoming vessels.

Robust scheduling based on operational buffers has already been introduced as a proactive approach in the BAP. An approach to robust BAP is presented in [8]. They presented a feedback procedure for the BAP that iteratively improves the robustness of the initial schedule. This feedback procedure determines the time buffers for each vessel by means of adjustment rules.

In [32], another approach to the robust BAP is solved by a scheduling algorithm that integrates simulated annealing and branch-and-bound algorithms. This study introduces the robustness as an objective to be maximized and an evaluation is carried out by varying the weights of these functions. The robustness is achieved by a constant buffer time assigned to all vessels.

In [34], the robust BAP problem is studied as a proactive strategy as a multi-objective optimization problem. They solved this problem with the Squeaky Wheel Optimization (SWO) metaheuristic. The first objective is to minimize the late departures and the deviation from the desired position; and, the second objective is to maximize the robustness of the schedule. They tackle the robustness measure as a diminishing return, specifically the exponential function, to capture the decreasing marginal productivity of slacks in a berthing schedule.

However, most of the above approaches consider discrete berths or previous knowledge about the uncertainty in arrival or handling times to produce robust schedules, but usually this knowledge is not available. Furthermore, other approaches propose how to obtain robust schedules by means of operational buffer times, but these buffers are set independently of the handling (or processing) time of the vessels.

Overcoming the above approaches, hybrid metaheuristics for both single and multi-objective combinatorial optimization problems have received a significant interest from the research community [4, 9], and also they have been used in a wide range of real-world applications [13].

In this paper, we introduce a robust model to deal with limited incidences with no previous knowledge about them (Section 3) as well as a multi-objective approach to face this problem (Section 4). A formal mixed integer programming (MIP) is presented for the dynamic and continuous Robust BAP+QCAP that extends the model presented in [18] (Section 5). Section 6 presents our proposed hybrid multi-objective genetic algorithm based on the scheme NSGAII [6] in order to obtain near-optimal solutions in an efficient way. This hybrid algorithm is used to solve the BAP+QCAP with a continuous quay and dynamic arrivals as well as to provide robust solutions by using operational buffers. As there is no previous knowledge about the incidences, these operational buffers are proportionally distributed among the tasks to be able to absorb as many incidences as possible. Thereby, a new objective function (standard deviation of robustness measures) was introduced to pursue this goal. This algorithm is compared with the mathematical model presented and two well-known Multi-Objective Genetic Algorithms: NSGAII and SPEA2+ [19] (Section 7). The development of the technique presented in this paper will provide the terminal operators with different robust berthing plans which are able to absorb limited incidences.

The overall collaboration goal of our group at the Universitat Politècnica de València (UPV) with the Valencia Port Foundation and the maritime container terminal MSC (Mediterranean Shipping Company S.A.) is to offer assistance and help in the planning and scheduling of tasks such as the allocation of spaces to outbound containers, to identify bottlenecks, to determine the consequences of changes, to provide support in the resolution of incidents, to provide alternative berthing plans, etc. Thus, the development of the technique presented in this paper will provide the terminal operators with different robust berthing plans which are able to absorb limited incidences.

2 Berthing Allocation and Quay Crane Assignment: BAP+QCAP

Let V be a set of incoming vessels, BAP+QCAP consists in obtaining an optimal (or near-optimal) schedule of the vessels V by assigning mooring times, berthing positions and QCs to each vessel. Our BAP+QCAP model is classified, according to the classification given by [2], as:

- *Spatial attribute: Continuous layout.* We assume that the quay is a continuous line, so there is no partitioning of the quay and the vessel can berth at arbitrary positions within the boundaries of the quay. It must be taken into account that for a continuous layout, berth planning is more complicated than for a discrete layout, but it better utilizes the quay space [2].
- *Temporal attribute: Dynamic arrival.* Fixed arrival times are given for the vessels, so that vessels cannot berth before their expected arrival times.
- *Handling time attribute: Unknown in advance.* The handling time of a vessel depends on the number of assigned QCs (*QCAP*) and the moves required.
- *Performance measure: wait and handling times* The objective is to minimize the sum of the waiting and handling times of all vessels V.

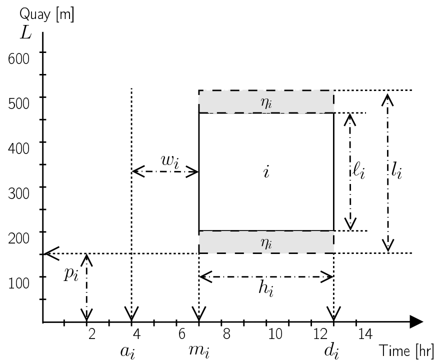


Figure 1: Data related to one vessel.

Following, we introduce the notation used for each vessel $i \in V$ (Fig. 1). The integer data variables are:

- K : Total number of QCs in the container terminal. We assume all QCs carry out the same number of movements per time unit (`movsQC`), given by the container terminal.
- L : Total length of the berth in the container terminal.

- a_i : Arrival time of the vessel i at port.
- c_i : Number of required movements to load and unload containers of vessel i.
- ℓ_i : Vessel length.

The decision variables are:

- m_i : Mooring time of i. Thus, waiting time (w_i) of vessel i is calculated as $(w_i = m_i - a_i)$.
- p_i : Berthing position where vessel i moors.
- q_i : Number of assigned QCs to vessel i.
- u_{ik} : Indicates whether the QC k $(1 \leq k \leq K)$ works (1) or not (0) on the vessel i.
- n_{ik} : Denotes that the number of QCs assigned to vessel i is k QCs ($n_{ik} = 1$). For instance, if vessel 3 has been assigned 4 QCs, then $n_{34} = 1$ and the others QCs $n_{3k} = 0, \forall k = 1 \ldots K, k \neq 4$.

The variables derived from the previous ones are:

- H_{ik} : Loading and unloading time at quay (handling time) of vessel i using k QCs $(1 \leq k \leq K)$. This handling time depends on c_i and it is defined by Eq. 1:

$$H_{ik} = \left\lceil \frac{c_i}{k * \texttt{movsQC}} \right\rceil \quad \forall i \in V, \forall k = 1 \ldots K \tag{1}$$

- h_i : Required handling time of vessel i when q_i QCs are assigned to it. This value is set by means of H_{iq_i}.
- t_{ik} : Working time of the kth QC $(1 \leq k \leq K)$ that is assigned to vessel i.
- d_i : Departure time of vessel i $(d_i = m_i + h_i)$.
- s_i, e_i : Indexes of the first and last QC assigned to vessel i, respectively.

In this study, the following assumptions are considered:

- All the information related to the waiting vessels is known in advance (arrival, priority, moves and length).
- Every vessel has a draft that is lower than or equal to the draft of the quay.
- Movements of QCs along the quay as well as berthing and departure times of vessels are not considered since it supposes a constant penalty time for all vessels.
- Simultaneous berthing is allowed, subject to the length of the berth.

Usually in container terminals, the number of QCs could vary during execution at the quay. This issue has been studied in [27]. However, in this paper and without loss of generality, we study the robustness of the schedules assuming that the number of QCs assigned to one vessel do not vary along the moored time. Once a QC starts a task in a vessel, it must complete it without any pause or shift (non-preemptive tasks). Thus, all QCs assigned to the same vessel i have the same working time on the vessel $(t_{ik} = h_i, \forall k = 1, \ldots, K, u_{ik} = 1)$.

The following constraints must be accomplished:

- Moored time of vessel i must be at least the same that its arrival time $(m_i \geq a_i)$.
- There is a safe distance between two moored ships. We assume that each vessel i has a 2.5% of this length at each side as a safe distance (η_i) (Fig. 1). This safe distance is added to the length of each vessel i: $l_i := \ell_i + 2\eta_i$.
- It must be enough contiguous space at berth to moor a vessel i of length (l_i).
- There must be at least one QC assigned to each vessel. Furthermore, there is a maximum number of QCs that can be assigned to vessel i (QC_i^+). This value, $\left(QC_i^+\right)$, depends on the length of each vessel (ℓ_i), since a safe distance is required between two contiguous QCs (`safeQC`), and the maximum number of QCs that the container terminal allows per vessel (`maxQC`) (Eq. 2). Both `safeQC` and `maxQC` parameters are given by the container terminal.

$$QC_i^+ = \min\left(\mathtt{maxQC}, \max\left(1, \left\lfloor \frac{\ell_i}{\mathtt{safeQC}} \right\rfloor\right)\right) \quad \forall i \in V \tag{2}$$

Our objective is to allocate all vessels according to several constraints minimizing the total waiting (T_w) and handling or processing time (T_h), known as the service time (T_s), for all vessels:

$$T_w = \sum_{i \in V} w_i \tag{3}$$

$$T_h = \sum_{i \in V} h_i \tag{4}$$

$$T_s = T_w + T_s \tag{5}$$

3 Robust BAP+QCAP Model

Uncertainty and non-determinism of real-world environments may cause difficulties in the initial plans made by the decision makers. In container terminals, the initial obtained schedules for the BAP+QCAP problem might become invalid due to different reasons: breakdowns in QCs, late arrivals of the vessels, extreme weather events, a lower ratio of movements per QC than expected, etc.

The robustness concept means that, given a schedule, this initial schedule remains feasible when minor incidences occur in its actual scenario.

The usual disruptions to be considered in BAP+QCAP are the followings:

- Early or late arrival of a vessel i from its expected arrival time (a_i).
- The handling time of a vessel i is larger than its expected handling time (h_i).

In this paper, we focus just on the disruptions affecting the handling time which eventually delay the departure time. In case of incidences related to late arrivals, they could also be modeled as delays in the handling time of the vessels which eventually also delay their departure time.

Definition. *Given the possible disruptions, we consider that a schedule is robust if a disruption in one vessel does not affect or alter the mooring times of the other vessels.*

The robustness of a schedule of BAP+QCAP might be guaranteed through two periods of time related to each vessel: waiting time of a vessel (w_i) and buffer time after the departure of each vessel (b_i) [26]. Without loss of generality, early arrivals are not taken into account since they only increase waiting times but they do not alter mooring times.

The schedule could absorb delays or breakdowns that do not exceed the sum of those two periods $(w_i + b_i)$. Therefore, both times should be maximized in order to achieve the maximum robustness and ensure that there is no need to re-schedule the involved vessels. However, it should be kept in mind that the first objective of the BAP+QCAP is to minimize the total service time of the incoming vessels $(w_i + h_i)$. Therefore, following the proposal given by [5], we focus on maximizing only the second period of time, buffer times $(\sum b_i)$, to obtain robust schedules.

Let φ_i be the vessels that succeed vessel i and occupy some berth space of vessel i (φ_i^p), or use any of QCs assigned to vessel i (φ_i^q). The buffer time of vessel i (b_i) is the minimum difference (τ_{ij}) between de departure time of vessel i (d_i) and the mooring time of vessel j $(m_j, j \in \varphi_i)$. In case there is no vessel in φ_i, the maximum buffer time is assigned to b_i (an infinite value). Fig. 2 shows an example of the buffer times (b_i) assigned for each scheduled vessel as an empty rectangle.

$$\varphi_i^p = \{\forall j \in V,\ m_j \geq d_i \ \wedge\ [p_i, p_i + l_i) \cap [p_j, p_j + l_j) \neq \varnothing\} \qquad \forall i \in V \qquad (6)$$

$$\varphi_i^q = \{\forall j \in V,\ m_j \geq d_i \ \wedge\ \exists k, 1 \leq k \leq K \wedge u_{ik} = 1 \wedge u_{jk} = 1\} \qquad \forall i \in V \qquad (7)$$

$$\varphi_i = \varphi_i^p \ \cup\ \varphi_i^q \qquad \forall i \in V \qquad (8)$$

$$\tau_{ij} = m_j - d_i \qquad \forall i \in V,\ \forall j \in \varphi_i \qquad (9)$$

$$b_i = \begin{cases} +\infty & , |\varphi_i| = 0 \\ \min_{j \in \varphi_i} (\tau_{ij}) & , \text{otherwise} \end{cases} \qquad \forall i \in V \qquad (10)$$

In this paper, we assume that the more handling time, the more likely to suffer incidences. Therefore, in general, the larger buffers, the more robust schedules. Nevertheless,

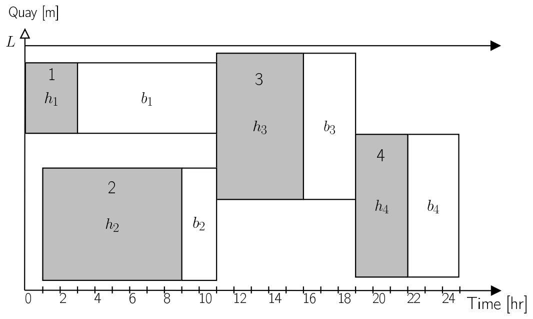


Figure 2: Buffer times b_i given an example schedule.

regarding the concept of decreasing productivity (or diminishing returns) presented in [34], there is a certain buffer size beyond which no more robustness is added to the schedule. Thereby, there is no need to assign large buffer times to each vessel. For instance, in Fig. 2, Vessel 1 would not need 8 time units of buffer time (b_1) since its handling time is only 3 time units. It is not likely that this vessel would suffer a delay of that magnitude. However, Vessel 2, with a handling time of 8 time units, has only 2 time units of buffer time (b_2). In this case, it is high likely that this vessel suffer some breakdown or delay and so it becomes invalid this schedule.

Furthermore, we consider that the magnitude of the incidence is related to the handling time of the vessel. Thus, the robustness measure of each vessel i ($r_i \in [0, 1]$) is related to the buffer time b_i and the average handling time h_i^* (Eq. 11). It should be mentioned that other functions, e.g. exponential function, could be adopted to define the robustness of each vessel.

$$h_i^* = \frac{c_i}{\left(\frac{1+QC_i^+}{2}\right)\texttt{movsQC}} \tag{11}$$

Given the robustness of each vessel, the robustness of a schedule $R \in [0, |V|]$ is defined by Eq. 13, where ω_i is a weighting factor ($\omega_i \geq 1$) which depends on historical data, if available. A $\omega_i = 1$ value represents that vessel i used to finish its tasks as expected, and $\omega_i > 1$ value denotes that vessel i used to be delayed.

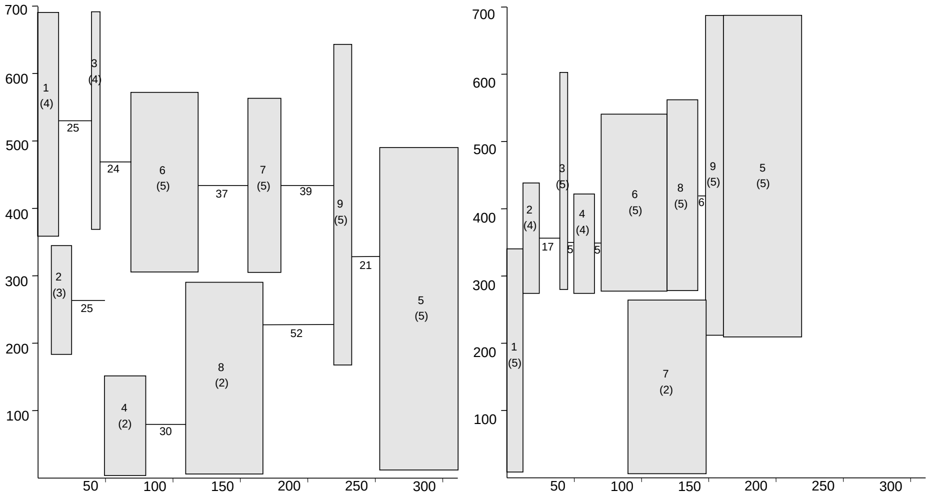
$$r_i = \min\left(1, \frac{b_i}{\omega_i h_i^*}\right), \quad \forall i \in V \tag{12}$$

$$R = \sum_{i \in V} r_i \tag{13}$$

In this paper, we address the BAP+QCAP problem without knowledge of the incidences, thus the weighting factor is the same for all the vessels ($\omega_i = 1, \forall i \in V$).

Example. *Fig. 3 shows two different schedules given the same set of 9 incoming vessels. Each vessel is labeled with its vessel's ID and the assigned QC number in brackets.*

Furthermore, the buffer time between vessels is also showed. On the one hand, Fig. 3(a) represents a robust schedule since limited incidences over any vessel could be absorbed. On the other hand, Fig. 3(b) shows a schedule with the optimal solution according to the objective function T_s. The latter schedule will be high likely unfeasible if any incidence occurs.



(a) Robust schedule. (b) Optimal schedule according to T_s.

Figure 3: Two possible schedules given the same incoming vessels.

Fig. 3(a) and Fig. 3(b) are an example of the well-known trade-off between optimality and robustness. However, a robust schedule is not only achieved by extending an optimal schedule over the time. A robust schedule must also consider an optimized allocation of vessels to achieve the maximum sum of buffer sizes with a proper distribution among all vessels. Note that the optimality is not directly the makespan of the schedule, but the total service time (waiting and handling times).

An important issue in this paper is that there is no available information about how likely the incidences or breakdowns occur. Therefore, it is interesting that these buffers are proportionally distributed among all the vessels. Thereby, a third objective is introduced into the model in order to improve the robustness of a schedule: minimizing the standard deviation (σ) of the robustness measures of all vessels ($r_i \ \forall i \in V$).

$$\sigma = \sqrt{\frac{1}{|V|}\sum_{i \in V}(r_i - \bar{r})} \tag{14}$$

where $\bar{r}$ is the average of the buffers of the schedule and $|V|$ is the number of incoming vessels.

Both measures presented above, robustness of a schedule (R) and standard deviation of these values (σ), represent the actual robustness of a schedule $\mathcal{R}$ to be maximized (see Eq. 15). This measure guarantees the absorption of incidences that imply at most a delay of a $\mathcal{R}\%$ of the weighted average handling time ($\omega_i h_i^*$).

$$\mathcal{R} = R - \sigma \tag{15}$$

Example. *Figure 4 shows two different schedules with a similar robustness value ($R = 0.7$), but different standard deviations, $\sigma_1 = 0.17, \sigma_2 = 0.45$. With these values, the first schedule has an actual robustness value of $\mathcal{R}_1 = 0.7 - 0.17 = 0.53$. Thus, in average this schedule guarantees that could absorb incidences that imply at most a delay of the 53% of the average handling time of the vessels. In contrast, the second schedule which has an actual robustness value of $\mathcal{R}_2 = 0.7 - 0.45 = 0.25$; and thus, it is able to absorb only incidences that imply at most a 25% of the average handling time of the vessels.*

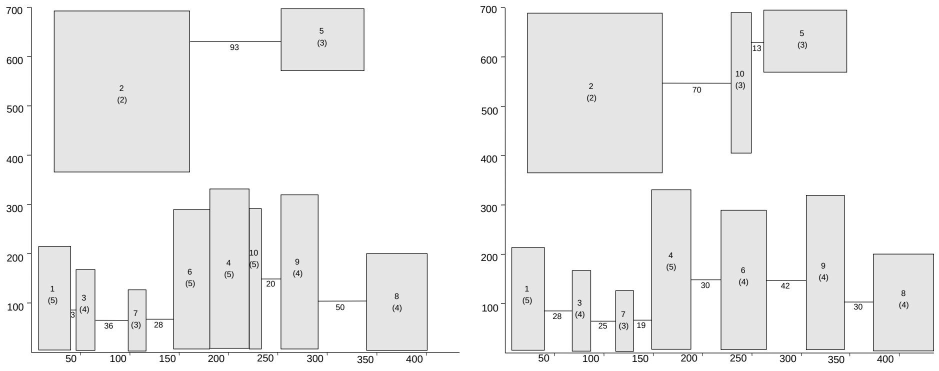
[30] presented a close function to measure the robustness of a schedule ($avg(w_i) - \alpha\sigma(w_i),\ \forall i \in V$). $avg(w_i)$ and $\sigma(w_i)$ denote the average and the standard deviation of the waiting times, respectively; and, α is a constant weighting factor that must be set. However, this measure does not reflect the relationship between the handling or processing time of the task and the buffer times. Thus, to our best knowledge, there is no other study which considering the BAP+QCAP with a continuous quay and dynamic arrivals tackles the robustness without any previous knowledge about the incidences.

Fig. 4 shows two different schedules of 10 vessels with the same high value of the robustness measure. However, schedule of Fig. 4(a) has a greater value for the standard deviation (σ) than schedule of Fig. 4(b). Thereby, it is important to note that buffer times from schedule in Fig. 4(a) are not equally distributed and this schedule will fail if an incidence which delays the departure time just 1 time unit over vessels 4 or 6 occurs or more than 3 time units over vessel 1. However, schedule in Fig. 4(b), it is high unlikely to be invalid since all vessels have enough buffer time after its schedule departure time.

4 Multiobjective Approach for the Robust BAP+QCAP

Three different objectives must be optimized to solve the robust BAP+QCAP: the service time (T_s) (Eq. 5), the robustness (R) (Eq. 13) and the standard deviation of the robustness measures $\sigma(R)$ (Eq. 14). These objective functions must be normalized in order to apply the search process correctly.

Eq. 18 shows how to normalize the service time objective into the interval $[0, 1]$ ($\hat{T}_s$) and it implies to normalize both the waiting time $\hat{T}_w$ (Eq. 16) and the handling time $\hat{T}_s$ (Eq. 17). On the one hand, the handling time is just a linear normalization since the maximum (h_i^+) and minimum (h_i^-) times are known by assigning the minimum (1) and the maximum number of QCs to vessel i (QC_i^+). On the other hand, normalizing the waiting time requires to determine a maximum total waiting time (W_F). In this case, W_F value is the total waiting time of the incoming vessels when a first-come, first-served



(a) High standard deviation.
(b) Low standard deviation.

Figure 4: Two different schedules with similar robustness and different standard deviation.

(FCFS) policy is applied, assigning 2 QCs to each vessel, and just one vessel is allowed in the berth at the same time (see example Fig. 5). The maximum total waiting time (W_F) could also be obtained by assigning just one QC to each incoming vessel, but in that case, W_F value would be too large and all the normalized waiting times would be close to zero.

$$\hat{T}_w = \frac{1}{W_F}\sum_{i\in V}(m_i - a_i) \qquad \hat{T}_w \in [0,1] \tag{16}$$

$$\hat{T}_h = \frac{1}{|V|}\sum_{i\in V}\left(\frac{h_i - h_i^-}{h_i^+ - h_i^-}\right) \qquad \hat{T}_h \in [0,1] \tag{17}$$

$$\hat{T}_s = \frac{\hat{T}_w + \hat{T}_h}{2} \qquad \hat{T}_s \in [0,1] \tag{18}$$

$$\tag{19}$$

Robustness objective function must also be normalized into the interval $[0,1]$ ($\hat{R}$) as defined by Eq. 20. The third objective, standard deviation of robustness measures, is already normalized due to the fact that r_i values are already in the interval $[0,1]$ (Eq. 14).

$$\hat{R} = \frac{R}{|V|} \qquad \hat{R} \in [0,1] \tag{20}$$

Thereby, the objective function for the robust BAP+QCAP is to minimize the function F (Eq. 21). Each coefficient λ_i $(0 \le \lambda_i \le 1)$ assigns different weights to each component or objective function in order to establish an aggregate function.

$$F = \lambda_1\hat{T}_s - \lambda_2\hat{R} + \lambda_3\sigma \tag{21}$$

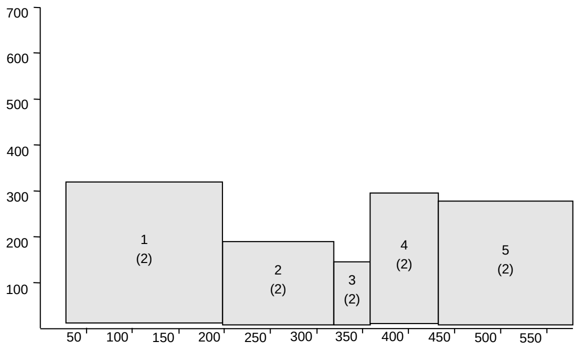


Figure 5: Schedule generated to obtain the maximum value of waiting time W_F.

These coefficients λ_i are subject to $\sum_i \lambda_i = 1$.

In a multi-objective optimization problem, usually there is no single solution wherein all its objectives are simultaneously optimized. However, there may exist a set of Pareto optimal solutions with different trade-offs between their objective functions. Pareto efficiency, or Pareto optimality, is a solution in which is impossible to make any one criteria better off without making at least one criteria worse off [35]. Pareto optimal solutions are defined by means of the dominance concept. Considering the robust BAP+QCAP, let x and y be two different solutions, x dominates y if at least one of the following conditions is satisfied:

$$\begin{array}{lll}
\hat{T}_s(x) < \hat{T}_s(y) & \wedge\, \hat{R}(x) \geq \hat{R}(y) & \wedge\, \sigma(x) \leq \sigma(y) \\
\hat{T}_s(x) \leq \hat{T}_s(y) & \wedge\, \hat{R}(x) > \hat{R}(y) & \wedge\, \sigma(x) \leq \sigma(y) \\
\hat{T}_s(x) \leq \hat{T}_s(y) & \wedge\, \hat{R}(x) \geq \hat{R}(y) & \wedge\, \sigma(x) < \sigma(y)
\end{array}$$

Given a set of feasible solutions D, a solution $x \in D$ is Pareto optimal solution if it is non-dominated by any other solution $x' \in D$. The Pareto optimal set is the set of all the solutions that are Pareto optimal solution [35].

In general, generating the Pareto optimal set is expensive computationally and it is often impracticable. Therefore, algorithms try to find a good approximation of the Pareto optimal set. In this work, we refer to each approximation as Pareto front, which contains solutions that, although are non-dominated among them, could be dominated by other solutions not found by our algorithms.

5 Mathematical Formulation

A mixed integer programming (MIP) model is presented to solve the robust BAP+QCAP. The objective function of this model is to minimize Eq. 21. This mathematical model is based on the model presented in [18] and [26].

In the proposed model, M denotes a sufficiently large number (as it is used in MIP). Furthermore, there are four auxiliary binary variables. z^x_{ij} is a decision variable that indicates if vessel i is located to the left of vessel j on the berth ($z^x_{ij} = 1$); and, $z^y_{ij} = 1$ indicates that vessel i is moored before vessel j in time. The auxiliary variable u_{ik} indicates whether the QC k works (1) or not (0) on vessel i; and $n_{ik} = 1$ denotes that the number of QCs assigned to vessel i is k.

$$\forall_{\substack{i,j \in V \\ i \neq j}} \forall_{k=1 \ldots K} \qquad z^x_{ij}, z^y_{ij}, u_{ik}, n_{ik} \quad \text{0/1 integer} \tag{22}$$

In the proposed model, there are four auxiliary binary variables. z^x_{ij} is a decision variable that indicates if vessel i is located to the left of vessel j on the berth ($z^x_{ij} = 1$); and, $z^y_{ij} = 1$ indicates that vessel i is moored before vessel j in time. The auxiliary variable u_{ik} indicates whether the QC k works (1) or not (0) on vessel i; and $n_{ik} = 1$ denotes that the number of QCs assigned to vessel i is k.

The constraints of this mathematical model are detailed below. Constraint 23 ensures that vessels must moor afer they arrive at the terminal.

$$\forall_{i \in V} \qquad m_i \geq a_i \tag{23}$$

Constraints 24 and 25 establish the waiting and departure times according to m_i and h_i.

$$\forall_{i \in V} \qquad w_i = m_i - a_i \tag{24}$$

$$\forall_{i \in V} \qquad d_i = m_i + h_i \tag{25}$$

Constraint 26 guarantees that a moored vessel does not exceed the length quay.

$$\forall_{i \in V} \qquad p_i + l_i \leq L \tag{26}$$

The number of QCs to the vessel i are assigned by means of constraints 27-32.

$$\forall_{i \in V} \qquad q_i = \sum_{k=1}^{K} u_{ik} \tag{27}$$

$$\forall_{i \in V} \qquad \sum_{k=1}^{K} n_{ik} = 1 \tag{28}$$

$$\forall_{i \in V} \qquad \sum_{k=1}^{K} n_{ik} k = q_i \tag{29}$$

$$\forall_{i \in V} \qquad 1 \leq q_i \leq QC^+_i \tag{30}$$

$$\forall_{i \in V} \qquad 1 \leq s_i \leq e_i \leq K \tag{31}$$

$$\forall_{i \in V} \qquad q_i = e_i - s_i + 1 \tag{32}$$

Constraints 33-35 establish the minimum handling time needed to load and unload their containers according to the number of assigned QCs.

$$\forall_{i\in V} \qquad \sum_{k=1}^{K} t_{ik}\,\texttt{movsQC} \geq c_i \tag{33}$$

$$\forall_{i\in V} \qquad \sum_{k=1}^{K} n_{ik} H_{ik} = h_i \tag{34}$$

$$\forall_{i\in V} \qquad h_i = \max_{\forall k=1\ldots K} t_{ik} \tag{35}$$

Constraint 36 ensures that QCs that are not assigned to vessel i have $t_{ik} = 0$.

$$\forall_{i\in V}\forall_{k=1\ldots K} \qquad t_{ik} - Mu_{ik} \leq 0 \tag{36}$$

Constraint 37 forces all assigned QCs to vessel i working the same number of hours.

$$\forall_{i\in V}\forall_{k=1\ldots K} \qquad h_i - M(1-u_{ik}) - t_{ik} \leq 0 \tag{37}$$

Constraint 38 avoids that one QC is assigned to two different vessels at the same time.

$$\forall_{i,j\in V}\forall_{k=1\ldots K} \qquad u_{ik} + u_{jk} + z^x_{ij} \leq 2 \tag{38}$$

Constraints 39 and 40 force the QCs to be contiguously assigned (from s_i up to e_i).

$$\forall_{i\in V}\forall_{k=1\ldots K} \qquad M(1-u_{ik}) + (e_i - k) \geq 0 \tag{39}$$

$$\forall_{i\in V}\forall_{k=1\ldots K} \qquad M(1-u_{ik}) + (k - s_i) \geq 0 \tag{40}$$

The safety distance between vessels is taken into account by constraint 41.

$$\forall_{\substack{i,j\in V\\ i\neq j}} \qquad p_i + l_i \leq p_j + M(1-z^x_{ij}) \tag{41}$$

Constraint 42 avoids that one vessel uses a QC which should cross through the others QCs.

$$\forall_{\substack{i,j\in V\\ i\neq j}} \qquad e_i + 1 \leq s_j + M(1-z^x_{ij}) \tag{42}$$

Constraint 43 avoids that vessel j moors while the previous vessel i is still at the quay.

$$\forall_{\substack{i,j\in V\\ i\neq j}} \qquad d_i \leq m_j + M(1-z^y_{ij}) \tag{43}$$

Constraint 44 establishes the relationship between each pair of vessels avoiding overlaps.

$$\forall_{\substack{i,j\in V\\ i\neq j}} \qquad z^x_{ij} + z^x_{ji} + z^y_{ij} + z^y_{ji} \geq 1 \tag{44}$$

Constraint 45 ensures that the total waiting time of the schedule does not exceed the maximum total waiting time (W_F).

$$\sum_{i \in V} w_i \leq W_F \tag{45}$$

Constraints 46-48 assign the time between the departure time of vessel i and the mooring time of vessel j. For those vessels j that $z^t_{ij} \neq 1$, they are assigned M as a value representing an unbounded time for the robustness.

$$\forall_{\substack{i,j \in V \\ i \neq j}} \qquad z^t_{ij} = z^x_{ij} + z^x_{ji} + z^y_{ij} \tag{46}$$

$$\forall_{\substack{i,j \in V \\ i \neq j \,\wedge\, (z^t_{ij}=0 \,\vee\, z^t_{ij}=2)}} \qquad \tau_{ij} = M \tag{47}$$

$$\forall_{\substack{i,j \in V \\ i \neq j \,\wedge\, z^t_{ij}=1}} \qquad d_i + \tau_{ij} = m_j + M(1 - z^y_{ij}) \tag{48}$$

Constraints 49 and 50 set the value of the available buffer time after vessel i and its robustness value, respectively.

$$\forall_{i \in V} \qquad b_i = \min\left(\min_{\substack{j \in V \\ i \neq j}} (\tau_{ij}),\ h^*_i\right) \tag{49}$$

$$\forall_{i \in V} \qquad r_i h^*_i = b_i \tag{50}$$

The decision variable z^t_{ij} (see constraint 51) indicates if a vessel j moors later than i and, at the same time, the vessel j intersects with the berth length occupied by vessel i (z^t_{ij}).

$$\forall_{\substack{i,j \in V \\ i \neq j}} \qquad 0 \leq z^t_{ij} \leq 2 \tag{51}$$

6 Multi-Objective Genetic Algorithms: MOGA+SA

Commonly approximations of the Pareto optimal sets of a multi-objective optimization problem are obtained by means of Multi-Objective Evolutionary Algorithms [35]. Furthermore, nowadays, metaheuristics are usually hybridized with other techniques or algorithms in order to enhance their effectiveness and performance [9, 4]. One of the most common forms of hybrid genetic algorithm involves incorporating local search to a canonical genetic algorithm. Genetic algorithm is used to perform global exploration among the population, and local search is used to perform local exploitation around the chromosomes. Because of the complementary properties of genetic algorithms and local search methods, the hybrid approach often outperforms either method operating alone [10].

Thereby, a hybrid multi-objective genetic algorithm (MOGA) has been implemented in this paper. The NSGAII schema has been extended with a multi-objective local search based on the multi-objective simulated annealing proposed by [1] (AMOSA), hereinafter named as MOGA+SA (see Algorithm 10). Moreover, two different schemes from the literature have been assessed NSGAII [6] and SPEA2+ [19].

The same chromosome structure is used in these three MOGAs. This chromosome has as many genes as incoming vessels ($|V|$). Each gene consists of three values (see Fig. 6): (1) the id of the next vessel to dispatch (i); (2) the number of QCs assigned (q_i); and (3) the buffer size after this vessel (b_i).

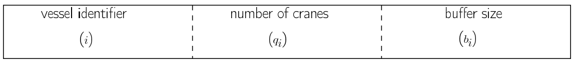


Figure 6: Structure of one gene of a chromosome.

It should be noted that each gene must be composed by feasible values with respect to vessel i. That is, according to the problem constraints, each vessel i can be assigned at most QC_i^+ cranes. Therefore, $1 \leq q_i \leq QC_i^+$. Likewise, if the berth length is L, then $\eta_i \leq p_i \leq L - l_i - \eta_i$.

In the following subsections, genetic operators that are used by the implementations of NSGAII and SPEA2+ are described.

6.1 Decoding and evaluation of one chromosome/solution

The structure of the chromosome, specifically the order of the vessels, is used as a dispatching rule. Hence, we use the following decoding algorithm: the genes are visited from left to right in the chromosome sequence. For each gene (i, q_i, b_i), the vessel i is scheduled at the earliest mooring time with q_i consecutive QCs available, so that none of the constraints is violated. In case there are several positions available at the earliest mooring time, the one closest to the berth extremes is selected. After the departure of the vessel i (d_i), it is ensured that there are b_i time units where no other vessel j ($\forall j \in V, j \neq i$) uses the QCs assigned to vessel i nor moors where vessel i does $[p_i, p_i + l_i)$.

Once a valid mooring time (m_i) and initial position (p_i) have been assigned to each vessel i, the fitness of the chromosome (Eq. 21) is obtained by computing each one of the objective functions: total service time ($\hat{T}_s$), robustness ($\hat{R}$) and standard deviation of the robustness (σ).

6.2 Generation of initial population

Construction of initial population ($generateInitialPopulation$ procedure) is performed so that the service time of a percentage of the initial population (GA parameter) is at least as good as the solution provided by the FCFS policy. The other chromosomes (or solutions) are constructed by instantiating each gene in the following way:

- Vessel identifier (i): an integer, between 1 and N is randomly chosen. Two genes of the same chromosome cannot have the same vessel identifier.

Algorithm 10: Implementation of MOGA+SA

```
Input: $\mathcal{I}$: instance of robust BAP+QCAP;
    $popsize$: number of chromosomes;
    $k$: number generations for local search
Output: $\mathcal{X}$: set of non-dominated schedules

    // Generate the parent population $P_0$
    $P_0 \leftarrow$ generateInitialPopulation($popsize$, $\mathcal{I}$)
    // Generate the offspring population $Q_0$
    $Q_0 \leftarrow$ evolvePopulation($P$, $popsize$)
    $t \leftarrow 0$
    while No termination criterion is satisfied do
        $unionSet \leftarrow$ makeUnionSet($P_t$, $Q_t$)
        $F \leftarrow$ fastNondominatedSort($unionSet$)
        $\mathcal{X} \leftarrow$ updateParetoFront($F_0$)
        // Create the next parent population $P_{t+1}$
        $i \leftarrow 0$
        $P_{t+1} \leftarrow \emptyset$
        while $i < |F| \wedge |P_{t+1}| + |F_i| < popsize$ do
            // Add the $i$th nondominated front ($F_i$) into the parent population $P_{t+1}$
            $P_{t+1} \leftarrow P_{t+1} \cup F_i$
            $i \leftarrow i + 1$
        end while
        if $|P_{t+1}| < popsize$ then
            // Sort $F_i$ according to the crowding distance measure
            crowdingDistance($F_i$)
            sort($F_i$)
            // Add the first $popsize - |P_{t+1}|$ elements of $F_i$
            $j \leftarrow 0$
            while $j < |F_i| \wedge |P_{t+1}| < popsize$ do
                $P_{t+1} \leftarrow P_{t+1} \cup \{F_i[j]\}$
                $j \leftarrow j + 1$
            end while
        end if
        // Use selection, crossover and mutation to create a new population $Q_{t+1}$
        $Q_{t+1} \leftarrow$evolvePopulation($P_{t+1}, popsize$)
        // Perform the local search each $k$ generations.
        if $t\%k = 0$ then
            $\mathcal{X}'$, $S_n \leftarrow$mosa($\mathcal{X}$)
            // Assign the new Pareto front to $\mathcal{X}$
            $\mathcal{X} \leftarrow \mathcal{X}'$
            $Q_{t+1} \leftarrow$ clustering($Q_{t+1}, popsize - |S_n|$)
            // The new solutions found by the local search are kept in the population
            $Q_{t+1} \leftarrow Q_{t+1} \cup S_n$
        end if
        // Increase the number of iterations
        $t \leftarrow t + 1$
    end while

    return Schedule of each element of the Pareto front $\mathcal{X}$
```

- Number of QCs (q_i): an integer, between 1 and QC_i^+, is randomly chosen.
- Buffer size (b_i): the initial buffer size is 0 for all genes of the initial population.

Once all chromosomes in the initial population have been instantiated, their fitness values are obtained as described in Section 6.1. Furthermore, the Pareto front $\mathcal{X}$ is updated considering all these chromosomes. Let x be a chromosome (or solution), x is added to the Pareto front $\mathcal{X}$ if there is no other solution $y \in \mathcal{X}$ such that y dominates x. If x is added to $\mathcal{X}$, then all solutions dominated by x are removed from $\mathcal{X}$.

6.3 Evolution of one population

In each iteration of the MOGA, a new population is built from the previous one (or the initial) by applying the genetic operators of selection, reproduction and replacement. The proposed approach follows the scheme:

- selection: all chromosomes in the actual population are randomly grouped into pairs;
- reproduction: 1) each one of these pairs is mated or not according to the cross-over probability P_c generating two offspring; and, 2) Each offspring, or parent if the parent were not mated, undergoes mutation in accordance with the mutation probability P_m.
- replacement: after evaluating the chromosomes previously generated, a tournament selection (4:2) is carried out among each pair of parents and their offspring as a replacement.

6.4 Crossover

The crossover operator receives one pair of chromosomes (P_1 and P_2), which are in the current population *pop* and have been randomly selected. The objective of this operator is to construct two offspring chromosomes (O_1 and O_2). For that, each time the crossover operation is performed, the following steps are made:

1. Two cross points are randomly chosen, k_1 and k_2 $(1 \leq k_1 < k_2 \leq N)$.
2. Each gene in chromosome P_1 and P_2 which is in position p, $k_1 \leq p < k_2$ is copied to the same position in chromosomes O_1 and O_2, respectively.
3. Each gene in chromosome P_1 and P_2 which is in position p, $1 \leq p < k_1$ is copied to the same position in chromosome O_1 and O_2, respectively.
4. Each gene in chromosome P_1 and P_2 which is in position p, $k_2 \leq p \leq N$ is copied to the same position in chromosome O_1 and O_2, respectively.

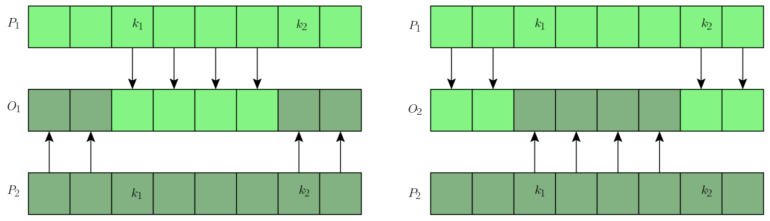


Figure 7: Crossover operation.

Fig. 7 is a graphical representation of the procedure that is used to perform the crossover operation, which is based on the technique Generalized Position Crossover [23] that is commonly used in permutation based encodings.

In one chromosome cannot be two genes with the same vessel identifier. Therefore, if the vessel identifier in the gene that will be copied already exists in the offspring (O_1/O_2) during steps 2 or 3, a new gene must be selected from the chromosome parent (P_1/P_2).

Once the vessel identifier of the selected gene does not exist in the offspring, then the gene is copied to the offspring in the corresponding position.

6.5 Mutation

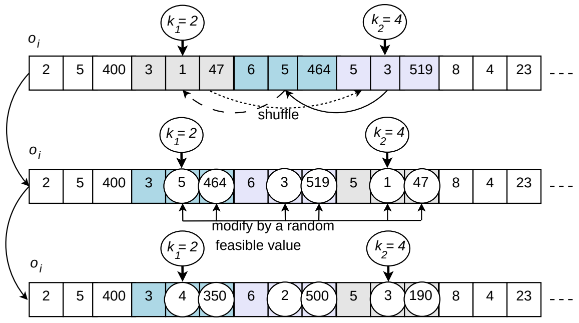


Figure 8: An example of mutation operation.

Mutation operation is performed on one chromosome, following these steps:

1. Two positions (k_1 and k_2) of the chromosome are randomly chosen $(1 \leq k_1 < k_2 \leq N)$.

2. Genes that are in positions between k_1 and k_2 (both included) are shuffled.

3. The number of QCs in each gene located between k_1 and k_2 (both included) is modified by a feasible random value with respect to the vessel in the same gene.

4. The buffer size in each gene located between k_1 and k_2, both included, is modified by a random value that is between 0 and the average handling time h_i^*, of the vessel i in the same gene.

Fig. 8 shows how the offspring o_i, which has been obtained after the crossover operation, is mutated. First, two values $k_1 = 2$ and $k_2 = 4$ are selected randomly. Then, all genes between both positions are shuffled. The gene 2 is moved to position 4, the gene 3 to position 1 and gene 4 to position 3. Finally, the number of QCs and the buffer size of each gene in position p, $2 \leq p \leq 4$ are modified by selecting feasible random values for each one.

6.6 Local search

Algorithm 11: createNeighbor

```
Input: $cur$: Actual solution/chromosome;
Output: $new$: Neighbor solution/chromosome;
    Copy chromosome $cur$ into $new$
    // Interchange the order of two vessels
    Vessels $v_1$ and $v_2$ randomly chosen from $new$
    Interchange position of $v_1$ and $v_2$ in $new$
    // Change the buffer assigned after its departure
    Randomly choose other vessel $v$ from $new$
    $r \leftarrow randInteger(0, 2)$
    if $r = 0$ then
        $b_v = 0$
    else if $r = 1$ then
        $b_v = h_i^*$
    else
        if $rand(0, 1) \leq 0.5$ then
            // Decrease a 10% the buffer size
            $b_v \leftarrow max\left(0, b_v - \frac{h_i^*}{10}\right)$
        else
            // Increase a 10% the buffer size
            $b_v \leftarrow min\left(h_i^*, b_v + \frac{h_i^*}{10}\right)$
        end if
    end if
    if $rand(0, 1) \leq 0.5$ then
        // Change number of assigned QCs
        $q_v \leftarrow rand(2, QC_v^+)$
    end if

    return Neighbour chromosome/solution $new$
```

The multi-objective simulated annealing presented by [1] has been modified and included into the MOGA as a local search to solve the Robust BAP+QCAP. The neighborhood structure of a solution takes advantage of the chromosome structure and their neighbors are obtained by changing the values of the variables presented in its genes (i, q_i and b_i). Algorithm 11 describes how to modify a given solution cur in order to create a neighbor new. In this process, two operations are applied to solution cur:

- interchanging the position of two vessels (v_1 and v_2) in the chromosome, randomly chosen.
- changing the values of number of QCs assigned (q_i) and buffer size (b_i) of a vessel i randomly chosen.

This multi-objective simulated annealing algorithm (*mosa* function) is computed every k iterations. It receives, as parameter, the Pareto front $\mathcal{X}$ of the actual iteration of the MOGA+SA. As a result, it returns two different sets of solutions or schedules:

- a Pareto front $\mathcal{X}'$ where the solutions from $\mathcal{X}$ have been improved to obtain a local optimal following the AMOSA scheme.
- a new set S_n consisting on the new non-dominated solutions found in the search which are part of $\mathcal{X}'$.

Unlike AMOSA [1], the simulated annealing algorithm implemented in this paper makes use of a different clustering method (see Algorithm 12) based on the crowding distance used in the NSGAII algorithm. This clustering method selects the representative solutions of the population according to the density of solutions surrounding a particular solution. After this local search process is performed, the solutions in S_n set replace the solutions in population $tmpPop$ whose crowding distance are the lowest ones. The solution to be replaced are chosen by means of the same clustering method used in the simulated annealing. The purpose is to improve the quality of the population by keeping the solutions that are most spread around the search space.

Algorithm 12: clustering

```
Input:  $P$: population;
    $N_m$; maximum number of chromosomes;
Output:  $P'$: population with exactly $N_m$ chromosomes;

    // calculate the crowding distance for each element/chromosome in $P$
    crowdingDistance($P$)
    // sort the elements in ascending order
    sort($P$)
    // choose the biggest $N_m$ elements according to the crowding distance
    $P' \leftarrow N_m$ elements of $P$ with the biggest crowding distances

    return $P'$
```

7 Evaluation

As no benchmark is available in the literature, the experiments were performed in a corpus of 100 instances randomly generated, where parameters (`maxQC`, `safeQC`, etc.) follow the suggestions of container terminal operators. All these benchmarks are freely available at http://gps.webs.upv.es/bap-qcap/. Each one is composed of a queue from 100 vessels. These instances follow an exponential distribution for the inter-arrival times of the vessels (scale parameter $\beta = 20$). The number of required movements and length of vessels are uniformly generated in $[100, 1000]$ and $[70, 400]$, respectively. In all cases, the berth length (L) was fixed to 700 meters; the number of QCs was 7 (corresponding to a determined MSC berth line) and the maximum number of QCs per vessel was 5 (`maxQC`); the safe distance between QCs (`safeQC`) was 35 meters and the number of movements that QCs carry out was 2.5 (`movsQC`) per time unit.

The approaches developed in this paper, NSGAII, SPEA2+ and MOGA+SA, were coded using C++; and their settings are showed in Table 1(a) and Table 1(b). Due to the stochastic nature of the GA process, each instance was solved 30 times and the results show the average obtained values. The mathematical model was coded and solved by using IBM ILOG CPLEX Optimization Studio 12.5. Due to the fact that the square root function defines concave region, standard deviation function could not be introduced into the objective function in the mathematical solver. They were solved on an Intel i7-2600 3.4Ghz with 8Gb RAM.

Table 1: Setting of the Algorithms

(a) NSGAII and SPEA2+ settings

Parameter	**MOGA scheme**
Number of Generations	500
Number of Chromosomes	100
Mutation probability (P_m)	0.1
Crossover probability (P_c)	0.9

(b) Multi-objective local search settings from MOGA+SA

Parameter	**MOGA+SA**
Initial temperature (T_{max})	20
Minimum temperature (T_{min})	0.001
Annealing factor (α)	0.9
Termination criterion	$T < T_{min}$
Cycle Length	2

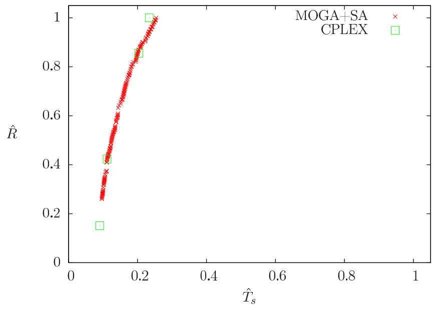
CPLEX is able to obtain a schedule of an instance for a given λ value. Algorithm 13 describes how to obtain a Pareto front using CPLEX solver for a given instance in order to be compared with the MOGA.

Algorithm 13: Pareto front from the mathematical model

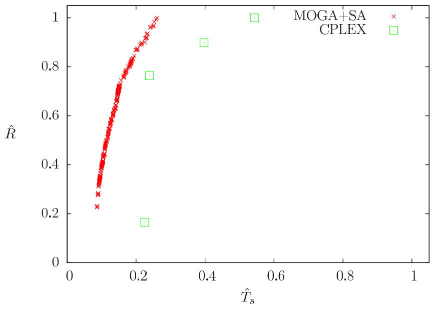
```
Input: $\mathcal{I}$: Instance;
    $T_o$: timeout;
Output: $\mathcal{X}$: Set of non-dominated solutions;

    Initialize set of solutions $S = \{\}$
    for $\lambda \in \{0, 1\}$ (steps of $0.1$) do
        Solve the mathematical model for the instance $\mathcal{I}$ and $\lambda$ value given the timeout $T_o$
        Add the schedule with the tuple of the objective functions ($\hat{R}$, $\hat{T}_s$) to the set $S$
    end for
    $\mathcal{X} \leftarrow$ non-dominated solutions from $S$.
```

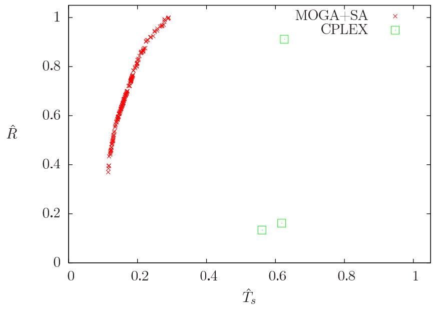
Fig. 9 shows the Pareto fronts obtained of a representative instance by both the MOGA+SA and CPLEX. In this experiment, the timeout for the CPLEX solver was set



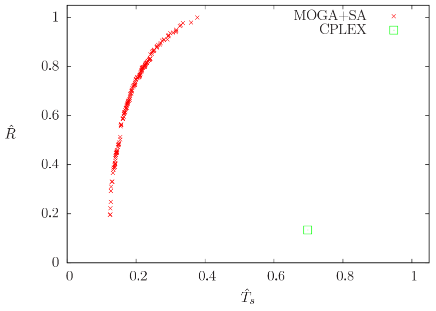
(a) $|V| = 10$ vessels



(b) $|V| = 20$ vessels



(c) $|V| = 30$ vessels



(d) $|V| = 40$ vessels

Figure 9: Pareto fronts of GA and CPLEX varying the number of incoming vessels ($|V|$).

to 1000 seconds for each λ value. It is important to note that the greater the incoming vessels, the fewer solutions obtained by CPLEX solver. Given this timeout, CPLEX was only able to get optimal solutions when $\lambda = 0.0$ and the incoming vessels was set to 10 and 20. Considering the Pareto fronts obtained by MOGA+SA and CPLEX, they were very similar with a queue of 10 vessels (see Fig. 9(a)). However, for instance with a queue 20 vessels, the solutions obtained by CPLEX were not able to reach the quality of the Pareto front of MOGA+SA (see Fig. 9(b)). Furthermore, it turned out that for 40 incoming vessels just one non-optimal solution was obtained (see Fig. 9(d)); and even more there was no solution with 50 vessels.

Multi-objective optimization algorithms are not comparable directly since there is no a unique optimal solution. [36] propose different measures to compare Pareto front approximations. Among these measures, the size of the dominated space or the hypervolume is one of the most used measures to differentiate two algorithms [31]. This measure is related to a reference point p and it is set according to the suggestion of [31]. To this end, for each objective, it is chosen the worst value from any of the sets being compared and increasing by an ϵ value.

Comparison among these different schemes has been performed by using the Kruskal-

Table 2: Kruskal-Wallis test over the hypervolumes obtained in 5 different instances

Instance	**p-value**	**Avg. Hypervolume**		
		NSGAII	**SPEA2+**	**MOGA+SA**
1	$< 2.2e - 16$	0.175315	0.207921	0.236660
2	$< 2.2e - 16$	0.174893	0.204749	0.252922
3	$< 2.2e - 16$	0.178945	0.216683	0.261251
4	$< 2.2e - 16$	0.180473	0.221138	0.258453
5	$3.699e - 15$	0.160139	0.173000	0.223231

Table 3: Wilcoxon test over the hypervolumes obtained for a given instance

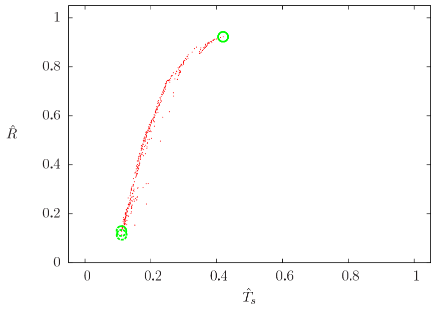
	NSGAII	**SPEA2+**
SPEA2+	0.00011	-
MOGA+SA	$< 2e - 16$	$< 2e - 16$

Wallis' non-parametric statistical test, according to [36]. This test assesses whether there are significant differences among different sets of values: in this case, sets of hypervolume measures. Table 2 shows the values obtained for this test given the results after solving five instances of 50 vessels with the three different algorithms. Kruskal-Wallis test revealed a significance effect of the algorithms on the hypervolumes (p-value< 0.01).

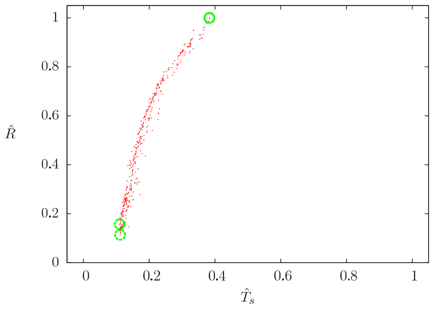
As there was a significance difference among them, a post-hoc test using a pairwise comparison test (Wilcoxon) with Bonferroni correction was carried out and showed the significant differences between the different algorithms. As example, Table 3 shows the results of the Wilcoxon test for the fifth instance. Note, MOGA+SA algorithm produces Pareto fronts which are statistically different with respect to the other algorithms. Examining the average values in Table 2, it can be determined that MOGA+SA obtained better Pareto front approximations.

Fig. 10 shows the Pareto fronts obtained by NSGAII and MOGA+SA algorithms. The schedules with the minimum and maximum values for each objective are highlighted by circles. It is important to note that MOGA+SA algorithm was able to produce a Pareto front with higher quality. Fig. 10(a) and Fig. 10(b) show the relationship between $\hat{R}$ and $\hat{T}_s$. MOGA+SA algorithm turned out to achieve schedules with greater robustness and lower $\hat{T}_s$ values ($\hat{R} = 1; \hat{T}_s = 0.3831$) than NSGAII algorithm ($\hat{R} = 0.9227; \hat{T}_s = 0.4196$). Furthermore, taking into account the relationship between $\hat{T}_s$ and $\hat{R}$ (see Figs. 10(e) and 10(f)), MOGA+SA algorithm achieved schedules with lower standard deviation of robustness ($\sigma(\hat{R}) = 0.0$) than the ones obtained by the NSGAII algorithm ($\sigma(\hat{R}) = 0.1557$).

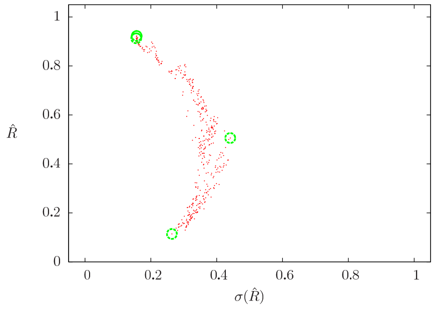
The performance of the schedules obtained by our approach (MOGA+SA) was evaluated by generating actual scenarios with some incidences in the actual handling time of the vessels. An incidence over a vessel i is modeled as a delay d in the handling time of vessel i. This incidence is absorbed if there is enough buffer time behind vessel i as to not alter the mooring time of the subsequent vessels. For each instance, the vessels that vary their handling times were uniformly chosen among all the scheduled vessels.



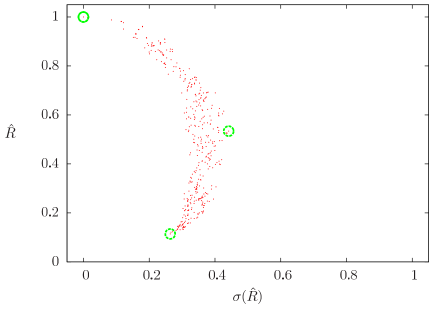
(a) NSGAII



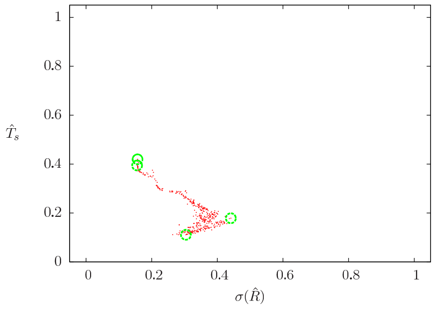
(b) MOGA+SA



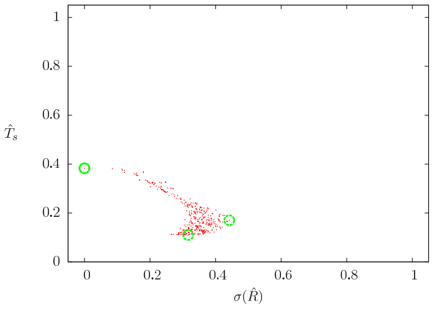
(c) NSGAII



(d) MOGA+SA



(e) NSGAII



(f) MOGA+SA

Figure 10: Pareto fronts obtained by using or not local search.

In this experiment, 100 instances of 100 vessels were evaluated. For each instance, three different schedules were chosen from the Pareto front according to their robustness (see Table 4(a)): the one with the minimum robustness (R_{min}), the one with the maximum robustness (R_{Max}) and one intermediate robust schedule (R_i where $R_{min} < R_i < R_{Max}$). Likewise, three schedules were chosen according to their service time (see Table 4(b)) and other three schedules according to their standard deviation of the robustness (see Table 4(c)).

Table 4: Percentages of incidences absorbed in schedules of 100 vessels

(a) Delays applied to schedules with different levels of $\hat{R}$

Range	$\mathbf{R_{min}}$	$\mathbf{R_i}$	$\mathbf{R_{Max}}$
$d \in [1,\ 0.2h_i]$	21.78	95.51	99.95
$d \in [1,\ 0.5h_i]$	18.73	93.58	99.85
$d \in [1,\ 0.8h_i]$	16.64	90.49	98.96
$d \in [1,\ 1.0h_i]$	13.92	87.10	98.01
$d \in [1,\ 1.2h_i]$	12.93	85.31	97.15

(b) Delays applied to schedules with different levels of $\hat{T}_s$

Range	$\mathbf{T_{s\,min}}$	$\mathbf{T_{s\,i}}$	$\mathbf{T_{s\,Max}}$
$d \in [1,\ 0.2h_i]$	20.17	79.91	99.94
$d \in [1,\ 0.5h_i]$	16.16	73.88	99.75
$d \in [1,\ 0.8h_i]$	13.89	65.17	98.94
$d \in [1,\ 1.0h_i]$	11.85	60.25	98.08
$d \in [1,\ 1.2h_i]$	10.32	56.76	96.74

(c) Delays applied to schedules with different levels of $\sigma(\hat{R})$

Range	$\sigma(\mathbf{R})_{\mathbf{min}}$	$\sigma(\mathbf{R})_{\mathbf{i}}$	$\sigma(\mathbf{R})_{\mathbf{Max}}$
$d \in [1,\ 0.2h_i]$	99.96	75.48	61.85
$d \in [1,\ 0.5h_i]$	99.95	72.23	54.42
$d \in [1,\ 0.8h_i]$	99.34	67.88	48.72
$d \in [1,\ 1.0h_i]$	98.87	65.34	47.77
$d \in [1,\ 1.2h_i]$	97.39	62.38	45.75

The incidences (or delays, d) introduced were randomly chosen from different ranges. These ranges vary from a minimum value (1) to a maximum value, which is related to the handling time (h_i) of the vessel affected by the incidence (see first column in Table 4). For each range, 100 incidences were uniformly created and applied to the four schedules of each instance.

Table 4(a) shows the percentage of incidences absorbed by each type of schedule. It can be observed that the more robust schedule, the more incidences absorbed. For instance, with delays $d \in [1, 0.5h_i]$, the R_{min} schedule only absorbed 18.73% of incidences in average, but the R_{Max} schedule absorbed up to 99.85.%. Note that as the delay became larger, fewer schedules can absorb the incidences. With delays in the range of $[1, 0.2h_i]$, the R_{Max} schedule can absorb 99.95% of incidences in average. However, with larger ranges, the incidences absorbed decreased down to 97.15% in average. This pattern was also repeated in Table 4(b). The lower T_s, the lower incidences absorbed due to the fact that either there would not be buffers among vessels or there would be small buffers.

Table 4(c) shows the percentage of incidences absorbed choosing three schedules by their standard deviation values. As expected, the highest percentages of incidences absorbed were obtained with the lowest values of standard deviation, e.g. 99.34% with delays in the range $[1, 0.8h_i]$. In general, schedules with the lowest standard deviation are related to those schedules with the greatest buffers proportionally distributed among all vessels (the most robust schedules).

The percentage of incidences absorbed by the most robust schedules using or not the local search algorithm are showed in Table 5. In this experiment, a timeout of 30 seconds was set for both algorithms. It is important to note that adding the local search to the Multi-Objective Genetic Algorithm allowed to increase the incidences absorbed in all the ranges. For instance, in range $[1,\ 1.0h_i]$, NSGAII was able to absorb 95.33% of incidences, whereas the MOGA+SA was able to absorb 97.34% of incidences.

Table 5: Percentages of incidences absorbed in schedules of 100 vessels obtained using or not LS (timeout 30 secs).

Range	$\mathbf{R_{Max}}$ no LS	$\mathbf{R_{Max}}$ with LS
$d \in [1,\ 0.2h_i]$	99.53	99.88
$d \in [1,\ 0.5h_i]$	99.40	99.70
$d \in [1,\ 0.8h_i]$	97.62	98.51
$d \in [1,\ 1.0h_i]$	95.33	97.34
$d \in [1,\ 1.2h_i]$	94.04	96.00

8 Conclusions

The competitiveness among container terminals causes the need to improve the efficiency of each one of the subprocesses or scheduling problems that are performed within them. However, this efficiency is affected by the uncertainty of the environment. This uncertainty might provoke delays in the arrivals of the vessels or handling times greater than expected due to extreme weather events, breakdowns in engines, delays, etc. Furthermore, these scheduling problems are even harder since they are interrelated and sometimes there is no previous knowledge about these incidences. To this end, we introduce the robustness into these scheduling problems. In this paper, we introduce the robustness into one of the main scheduling problems in Container Terminals: Berth Allocation and Quay Crane Assignment Problems. Its objective function is to minimize the total service time of the incoming vessels. The robustness, as second objective function, has been modeled as a measure related to the likelihood of a schedule to absorb incidences. This robustness has been related to the operational buffers found after each assigned vessel. The greater the operational buffers, the higher the robustness of the schedule. However, due to the lack of the knowledge about incidences, operational buffers should be distributed among vessels proportionally, and thus the third objective managed in this way is to minimize the standard deviation of the robustness measurements.

In this paper, a mixed integer programming (MIP) model and a new hybrid multi-objective genetic algorithm (MOGA+SA) were developed for the dynamic and continuous robust BAP+QCAP. They were compared with two well-known multi-objective Genetic Algorithms (MOGAs): NSGAII and SPEA2+. In multi-objective optimization problems there is no a unique optimal solution, and it is necessary to assess the trade-off between all the objectives by using the Pareto front. Visualizing Pareto fronts provides container terminals operators with a helpful system to decide which schedule is better depending on the actual state of the container terminal.

The results showed that the MIP model was able to obtain robust and efficient schedules up to 10 incoming vessels. However, MOGA+SA achieved better Pareto fronts than the MIP model for queues of incoming vessels greater than or equal to 20 vessels. Thereby, the schedules obtained by MOGA+SA were more efficient and robust than the schedules obtained by the MIP model. Furthermore, the MIP model was unable to found any solution with a given timeout for a queue of 50 incoming vessels. Additionally, differences between the MOGAs have been assessed by means of non-parametric statistical tests. It turned out to be that MOGA+SA obtained better Pareto fronts according to the hypervolume measures. Furthermore, different sets of incidences were simulated into the schedules obtained by the NSGAII and the MOGA+SA. The results returned that the schedules obtained by MOGA+SA were more robust due to the fact that they could absorb more incidences.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgements

This work has been partially supported by by the Spanish Government under research project MINECO TIN2013-46511-C2-1-P, the project PIRSES-GA-2011-294931 (FP7-PEOPLE-2011-IRSES) and the predoctoral FPU fellowship (AP2010-4405).

References

[1] Sanghamitra Bandyopadhyay, Sriparna Saha, Ujjwal Maulik, and Kalyanmoy Deb. A simulated annealing-based multiobjective optimization algorithm: Amosa. *Evolutionary Computation, IEEE Transactions on*, 12(3):269–283, 2008.

[2] C. Bierwirth and F. Meisel. A survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research*, 202:615–627, 2010.

[3] Jean-Charles Billaut, Aziz Moukrim, and Eric Sanlaville. *Flexibility and robustness in scheduling*, volume 56. Wiley. com, 2010.

[4] Christian Blum, Jakob Puchinger, Gnther R. Raidl, and Andrea Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6):4135 – 4151, 2011.

[5] Andrew J Davenport, Christophe Gefflot, and J Christopher Beck. Slack-based techniques for robust schedules. In *Proceedings of the Sixth European Conference on Planning (ECP-2001)*. Citeseer, 2001.

[6] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

[7] Ali Diabat and Effrosyni Theodorou. An integrated quay crane assignment and scheduling problem. *Computers & Industrial Engineering*, 73(0):115 – 123, 2014.

[8] Yuquan Du, Ya Xu, and Qiushuang Chen. A feedback procedure for robust berth allocation with stochastic vessel delays. In *Intelligent Control and Automation (WCICA), 2010 8th World Congress on*, pages 2210–2215. IEEE, 2010.

[9] Matthias Ehrgott and Xavier Gandibleux. Hybrid metaheuristics for multi-objective combinatorial optimization. In Christian Blum, MariaJosBlesa Aguilera, Andrea Roli, and Michael Sampels, editors, *Hybrid Metaheuristics*, volume 114 of *Studies in Computational Intelligence*, pages 221–259. Springer Berlin Heidelberg, 2008.

[10] Mitsuo Gen and Runwei Cheng. *Genetic algorithms and engineering optimization*, volume 7. John Wiley & Sons, 2000.

[11] G. Giallombardo, L. Moccia, M. Salani, and I. Vacca. Modeling and solving the tactical berth allocation problem. *Transportation Research Part B: Methodological*, 44(2):232 – 245, 2010.

[12] Xiao-le Han, Zhi-qiang Lu, and Li-feng Xi. A proactive approach for simultaneous berth and quay crane scheduling problem with stochastic arrival and handling time. *European Journal of Operational Research*, 207(3):1327–1340, 2010.

[13] Rosmalina Hanafi and Erhan Kozan. A hybrid constructive heuristic and simulated annealing for railway crew scheduling. *Computers & Industrial Engineering*, 70(0):11 – 19, 2014.

[14] Maarten Hendriks, Marco Laumanns, Erjen Lefeber, and Jan Tijmen Udding. Robust cyclic berth planning of container vessels. *OR spectrum*, 32(3):501–517, 2010.

[15] L.E. Henesey. *Multi-agent systems for container terminal management*. Citeseer, 2006.

[16] Qing-Mi Hu, Zhi-Hua Hu, and Yuquan Du. Berth and quay-crane allocation problem considering fuel consumption and emissions from vessels. *Computers & Industrial Engineering*, 70(0):1 – 10, 2014.

[17] A. Imai, H.C. Chen, E. Nishimura, and S. Papadimitriou. The simultaneous berth and quay crane allocation problem. *Transportation Research Part E: Logistics and Transportation Review*, 44(5):900–920, 2008.

[18] K.H. Kim and K.C. Moon. Berth scheduling by simulated annealing. *Transportation Research Part B: Methodological*, 37(6):541–560, 2003.

[19] Mifa Kim, Tomoyuki Hiroyasu, Mitsunori Miki, and Shinya Watanabe. Spea2+: Improving the performance of the strength pareto evolutionary algorithm 2. In *Parallel problem solving from nature-PPSN VIII*, pages 742–751. Springer, 2004.

[20] Olivier Lambrechts, Erik Demeulemeester, and Willy Herroelen. Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities. *Journal of scheduling*, 11(2):121–136, 2008.

[21] C. Liang, J. Guo, and Y. Yang. Multi-objective hybrid genetic algorithm for quay crane dynamic assignment in berth allocation planning. *Journal of Intelligent Manufacturing*, 22:471–479, 2011.

[22] A. Lim. The berth planning problem. *Operations Research Letters*, 22(2-3):105–110, 1998.

[23] D. C. Mattfeld. *Evolutionary search and the job shop: Investigations on Genetic Algorithms for production scheduling*. 1995.

[24] Kiyeok Park, Taejin Park, and Kwang Ryel Ryu. Planning for remarshaling in an automated container terminal using cooperative coevolutionary algorithms. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1098–1105. ACM, 2009.

[25] Y.M. Park and K.H. Kim. A scheduling method for berth and quay cranes. *OR Spectrum*, 25(1):1–23, 2003.

[26] Mario Rodríguez-Molins, Laura Paola Ingolotti, Federico Barber, Miguel A. Salido, María R. Sierra, and Jorge Puente. A genetic algorithm for robust berth allocation and quay crane assignment. *Progress in AI*, 2(4):177–192, 2014.

[27] Mario Rodriguez-Molins, MiguelA. Salido, and Federico Barber. A grasp-based metaheuristic for the berth allocation problem and the quay crane assignment problem by managing vessel cargo holds. *Applied Intelligence*, 40(2):273–290, 2014.

[28] Miguel A. Salido, Mario Rodriguez-Molins, and Federico Barber. Integrated intelligent techniques for remarshaling and berthing in maritime terminals. *Advanced Engineering Informatics*, 25(3):435–451, 2011.

[29] R. Stahlbock and S. Voß. Operations research at container terminals: a literature update. *OR Spectrum*, 30(1):1–52, 2008.

[30] Michele Surico, Uzay Kaymak, David Naso, and Rommert Dekker. A bi-objective evolutionary approach to robust scheduling. In *Fuzzy Systems Conference, 2007. FUZZ-IEEE 2007. IEEE International*, pages 1–6. IEEE, 2007.

[31] Lyndon While, Lucas Bradstreet, and Luigi Barone. A fast way of calculating exact hypervolumes. *Evolutionary Computation, IEEE Transactions on*, 16(1):86–95, 2012.

[32] Ya Xu, Qiushuang Chen, and Xiongwen Quan. Robust berth scheduling with uncertain vessel delay and handling time. *Annals of Operations Research*, 192(1):123–140, 2012.

[33] C. Zhang, L. Zheng, Z. Zhang, L. Shi, and A.J. Armstrong. The allocation of berths and quay cranes by using a sub-gradient optimization technique. *Computers & Industrial Engineering*, 58(1):40–50, 2010.

[34] Lu Zhen and Dao-Fang Chang. A bi-objective model for robust berth allocation scheduling. *Computers & Industrial Engineering*, 63(1):262–273, 2012.

[35] Aimin Zhou, Bo-Yang Qu, Hui Li, Shi-Zheng Zhao, Ponnuthurai Nagaratnam Suganthan, and Qingfu Zhang. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32–49, 2011.

[36] Eckart Zitzler, Joshua Knowles, and Lothar Thiele. Quality assessment of pareto set approximations. In *Multiobjective Optimization*, pages 373–404. Springer, 2008.

Chapter 3

General discussion of the results

In this chapter, the main results achieved in this thesis are discussed. They will be divided according to their scopes: planning; scheduling; planning and scheduling; and robust scheduling.

3.1 Planning

In planning, generally planners such as Metric FF (Hoffmann 2003), LPG-TD (Gerevini, Saetti, and Serina 2003), or LAMA (Richter and Westphal 2010) are used to build the needed plans to reach a goal state. They are general purpose planners, or also known as domain-independent planners. They could solve any problem, but they do not take advantage from the knowledge of the problem itself. Planners that make use of this knowledge are known as domain-dependent planners.

In the first part of this thesis, a general purpose planner (Metric FF) is compared with domain-dependent planners. Specifically, the domain-independent anytime planner called *Simplanner* (Sapena and Onaindía 2002) was extended in order to introduce heuristics to cope a real-world problem. In this thesis, the Remarshalling problem, related to the Container Stacking Problem, was chosen as a case study. As this problem is a slight modification of the *Blocks World* domain (Slaney and Thiébaux 2001), its domain and instances were modeled by using PDDL (Planning Domain Definition Language, (Ghallab, Howe, et al. 1998)).

Through this study, it could be observed how the performance of the domain-dependent planners outperforms the domain-independent planners. They are able to build plans quicker as well as they also achieve plans with a higher degree of optimality (Salido, Sapena, et al. 2009; Rodriguez-Molins, Salido, and Barber 2012). Table 3.1 shows the differences between these two different planners facing the remarshalling problem. For instance, with a scenario (yard-bay) of 20 containers and 4 export containers ($< 20, 4 >$), Metric FF obtained a solution in 22 secs with 7.01 reshuffles or moves in average. However, the domain-dependent planner (represented as h_1) was able to obtain in 10 secs a plan with 5.22 reshuffles in average.

Table 3.1: Average number of reshuffles and running time of *Metric FF* and h_1 in instances $< n, 4 >$.

Instance	***Metric FF*** Running time	***Metric FF*** Solution	**Heuristic (h_1)** Time first solution	**Heuristic (h_1)** Best Solution in 10 secs
$< 13, 4 >$	22	3.07	2	3.07
$< 15, 4 >$	3102	4.04	5	3.65
$< 17, 4 >$	4669	5.35	11	4.35
$< 19, 4 >$	6504	6.06	22	4.72
$< 20, 4 >$	22622	7.01	33	5.22
$< 21, 4 >$	13981	6.82	62	5.08

Another important advantage of domain-dependent planners is the possibility of introducing different optimization criteria to deal with real-world requirements. In the remarshalling problem within container terminals, three different optimization criteria were introduced into the planner to manage blocks of yard-bays (each yard-bay as a subproblem) (Rodriguez-Molins, Salido, and Barber 2012):

- Reduce the distance to the point of charge/discharge of containers.
- Avoid large differences in height between consecutive stacks of containers, known as sinks (OC_{nB}).
- Keep a certain distance between each pair of dangerous containers (OC_{nD}).

It is important to note that when optimization criteria are introduced into the search process to cope with requirements, the quality of the plans will be affected since the original problems are now more constrained. Figure 3.2 shows the average number of reshuffles obtained by the Metric FF and our domain-dependent planner (h_1) with each optimization criteria. The optimization criteria OC_N stands for the union of both optimization criteria OC_{nB} and OC_{nD}. For instance, note that if the optimization criteria of balance (OC_{nB}) or dangerous containers (OC_{nD}) are taken into consideration, the number of sinks (unbalanced stacks) or the number of dangerous containers in risk were reduced down to 0, but the number of reshuffles was increased.

Blocks of yard-bays of containers are modeled as a composition of subproblems in these experiments, and an important decision is what sequence of yard-bays is followed to solve

Table 3.2: Average results with blocks of 20 *yard-bays* each one being a $< 15, 4 >$ instance.

	Metric FF	h_1	OC_{nB}	OC_{nD}	OC_N
Reshuffles	3.65	3.38	4.85	4.00	5.65
Sinks	18.00	29.50	0	40.33	0
Non-Safe Dangerous	16.00	7.50	8.00	0	0

the whole block (Salido, Rodriguez-Molins, and Barber 2012). It is preferable to solve the tightest yard-bays or subproblems firstly. The tightest yard-bays are the ones which have more export containers (constraints). In (Salido, Rodriguez-Molins, and Barber 2012), a sequential planner (H1 Sequential) was compared with a planner (H1 Ordered) which solves first the tightest subproblems.

In Table 3.3, we show the performance of our planner H1 Sequential and H1 Ordered. These experiments were performed on blocks of containers composed by 10 yard-bays in the form $< 17, s >$. Thereby, each yard-bay has a different number of goal containers.

The first column in Table 3.3 corresponds to each instance solved by each planner. Thereby, row 1 corresponds to instance 1 of planner H1 Ordered (1-O), row 2 corresponds to instance 1 of planner H1 Sequential (1-S), and so on. The following 10 columns have two rows for each instance. They show for each instance the order in which the yard-bays are executed (upper row) and the number of goal containers that each one of them has (lower row). As example, for the instance 1-O, the sixth yard-bay is the first one in being executed and it has 9 goal containers. Finally, the last column presents the number of reshuffles needed to solve the instance or *Time Out* if a solution is not found.

As it can be observed in Table 3.3, there were instances which only can be solved through the H1 Ordered (instance 1). In other instances (instance 2), H1 Ordered planner gave a more efficient plan than the sequential one. However, there were also other examples in which both planners returned the same plan (instance 3). Thus, we can conclude that H1 Ordered can be considered a better planner than H1 Sequential to solve the complete block of yard-bays.

3.2 Scheduling

Combinatorial optimization problems have usually been solved by means of rules of thumb learnt through years of experience of the decision makers. They could be named as greedy solutions for these problems, since they cannot value all the variables to make proper decisions. Example of these rules are First-Come, First Served (FCFS) or Last-In, First-Out (LIFO).

Table 3.3: Comparison of H1 Sequential and H1 Ordered.

Inst.	**Order (Yard-Bay/N. Goal containers)**										**Number Reshuffles**
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	
1-O	6 9	1 7	2 6	8 6	3 5	4 5	10 5	9 4	5 2	7 2	91
1-S	1 7	2 6	3 5	4 5	5 2	6 9	7 2	8 6	9 4	10 5	*Time Out*
2-O	3 8	2 6	7 6	8 6	10 6	4 5	6 5	1 4	9 4	5 1	55
2-S	1 4	2 6	3 8	4 5	5 1	6 5	7 6	8 6	9 4	10 6	99
3-O	7 10	9 7	2 6	10 6	3 5	4 5	6 4	1 3	8 3	5 2	63
3-S	1 3	2 6	3 5	4 5	5 2	6 4	7 10	8 3	9 7	10 6	63

Other approach often used by decision makers is to employ mathematical solvers. These solvers always return the optimal plan or schedule for a given instance if they are given enough time (probably weeks or months). A well-known example of mathematical solver is the IBM ILOG CPLEX Optimization Studio. In this thesis, this solver has been used to assess the performance and quality of our approaches.

As a case study in this section, we focus on the ship-to-shore subsystem in container terminals. This subsystem involves operations related to the movement of containers from ship to berth. Two problems were studied, the Berth Allocation Problem (BAP) and the Quay Crane Assignment Problem (QCAP), hereinafter named as BAP+QCAP. The former is a well-known combinatorial optimization problem (Lim 1998), which consists in assigning berthing positions and mooring times to incoming vessels. The QCAP deals with assigning a certain number of QCs to each moored vessel such that all required movements of containers can be fulfilled (Bierwirth and Meisel 2010).

Mainly, two metaheuristics were developed to solve the BAP+QCAP problem:

- A constructive metaheuristic: Greedy Randomized Adaptive Randomized Procedure (also known as GRASP). This metaheuristic also needs to include a local search which was designed based on the hill-climbing technique (Rodriguez-Molins, Salido, and Barber 2014a); and,

- A population-based metaheuristic: Genetic Algorithm (GA) (Rodriguez-Molins, Barber, et al. 2012).

Both metaheuristics, GRASP and GA were evaluated with different corpora[1]. Besides these metaheuristics, a mixed integer lineal programming (MILP) model was developed to solve the BAP+QCAP. This MILP model extends the mathematical model presented in (Kim and Moon 2003).

Firstly, GRASP metaheuristic consists in two phases. First, a schedule is built by adding one element at a time according to a certain heuristic. In our problem, each element of the solution is one incoming vessel or container ship. Once the schedule is completed with all the incoming vessels, a local search is applied in order to improve locally this solution. This local search seeks promising solutions (schedules) among a certain number of neighbours K from the actual solution. Figure 3.1(a) shows why this second phase (local search) is necessary. Using just the constructive phase of the GRASP metaheuristic ($K = 0$), the best value achieved was 1322.7 with $\delta = 0.2$ (see Figure 3.1(a)). This metaheuristic without using any local search is also known as a semi-greedy heuristic (Hart and Shogan 1987).

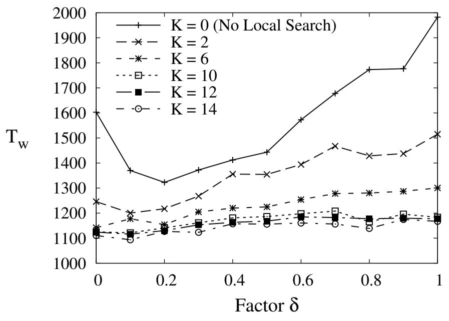
In general, the greater the K value, the better waiting times values (T_w) since a deeper search in the neighborhood is carried out. For instance, the T_w obtained by $\delta = 0.2$ decreased to 1127 when $K = 14$ neighbors are generated in each step of the local search. However, we can see that for $K > 12$, we did not achieve a significance improvement in the objective function. Furthermore, it is important to note that the greater the K value for the local search, the greater the computation time (see Figure 3.1(b)). Given the $\delta = 0.2$, the computation time increased from 8.23 ms up to 15.97 ms per iteration. Therefore, a value $K = 12$ was set for the local search phase of the GRASP metaheuristic for all the following experiments.

Metaheuristics became important due to the fact that mathematical solvers were unable to obtain solutions to realistic-size instances of real-world problems. Metaheuristics are designed in a way to achieve near-optimal solutions within a reasonable computation time.

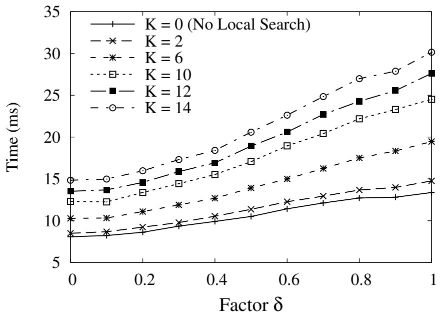
As an example, Table 3.4 shows the average results for CPLEX and the other metaheuristic developed, the GA, from 5 to 20 vessels. In this case, the objective to be minimized is the total service time (T_s). The timeout was set to 10 seconds. For CPLEX, the reported values are the average value of T_s for the solutions reached ($AvgT_s$), the number of instances solved to optimality (#Opt) and the number of instances solved without certifying optimality (#NOpt). The last two columns show the best and the average values of the solutions obtained by the GA in 5 runs, respectively.

From these results, it can be observed that CPLEX was not able to reach any optimal solution by the given timeout in at least 25% of the instances with 8 vessels or more. In addition, it cannot get any optimal solution from 18 up to 20 vessels given with timeout. Regarding GA, all instances were solved and it can be observed that the average values were better than CPLEX results, the differences being in direct ratio with the number of

[1]These benchmarks are freely available at `http://www.dsic.upv.es/~mrodriguez.`



(a) Waiting time values (T_w)



(b) Computation times

Figure 3.1: Local search results depending on the K value (*Dens* corpus with exponential inter-arrival distribution)

vessels. Here, it is important to remark that GA reached 2053 generations in 10 seconds. However, the GA was able to converge in lower times.

Furthermore, Figure 3.2 shows the average total service time obtained by CPLEX and GA for 10 vessels with different timeouts. Note that the average T_s for 10 vessels decreases as more computation time is allowed. In this experiment, the timeout was set to 5, 10, 20, and 60 seconds. As it can be observed, the GA approach does not require a large timeout (the improvement is lower than 1% beyond 5 seconds).

Summarizing, both metaheuristics were developed to solve a combinatorial optimization problem, the BAP+QCAP. Firstly, a constructive metaheuristic, GRASP, which builds schedules by choosing one element at a time (task, container ship, product, etc.). To this end, it is necessary to define a local heuristic to evaluate each possible next element. Once the schedule is built, a local search is performed to improve the schedule in the neighborhood space.

Table 3.4: Comparision CPLEX with GA (timeout 10 secs)

$\|V\|$	CPLEX			GA	
	Avg T_s	**#Opt**	**#NOpt**	**Best T_s**	**Avg T_s**
5	267.08	100	0	267.08	267.08
6	339.25	97	3	339.21	339.21
7	417.52	88	12	416.70	416.80
8	501.41	74	26	497.70	498.05
9	585.94	58	42	575.57	576.42
10	690.91	38	62	667.50	669.57
11	797.15	24	76	759.07	763.13
12	927.85	18	82	854.45	861.35
13	1065.32	12	88	949.55	959.35
14	1212.86	6	94	1055.05	1066.19
15	1406.21	3	97	1158.88	1173.59
16	1610.21	2	98	1276.12	1296.09
17	1796.58	1	99	1379.01	1407.02
18	2101.27	0	100	1497.33	1526.94
19	2333.46	0	100	1621.19	1658.57
20	2603.36	0	100	1798.63	2090.61

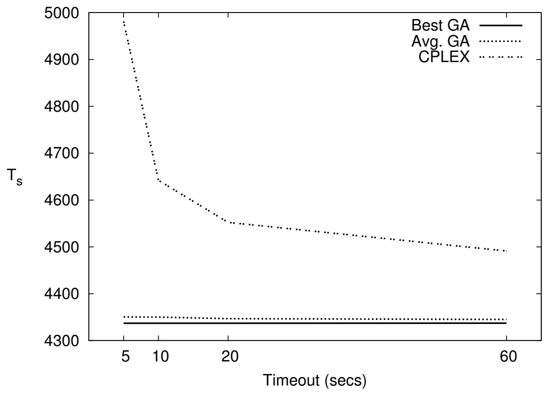


Figure 3.2: Average T_s for 10 vessels setting different timeouts.

The other metaheuristic evaluated was the genetic algorithm, a population-based metaheuristic. In this case, there are four operators to be defined: selection, mutation, crossover and replacement. Due to the fact that this metaheuristic consists of different solutions (chromosomes) evolving in each generation, it could do a better exploration in the search space.

3.3 Planning and Scheduling

In real-word environments, different optimization problems or processes must be taken into consideration at the same time to achieve an optimal solution. A small difference in the schedule or plan for one problem may cause that the global solution became not as optimal as expected. In this section, a decision support system was developed which integrates BAP+QCAP (scheduling) and CStackP (planning), hereinafter named as BAP+CStackP (Salido, Rodriguez-Molins, and Barber 2011; Salido, Rodriguez-Molins, and Barber 2012).

This decision support system seeks near-optimal solutions for BAP+CStackP. A near-optimal solution just for CStackP does not mean it fits the scheduling for the incoming vessels. As an example, Figure 3.3 shows, for ten different scenarios, the combined function cost Cost(Sol_i), introduced in (Salido, Rodriguez-Molins, and Barber 2012), which relates:

- The normalized total weighted waiting time of vessels, Cost($SBAP_i$);
- The number of its required container relocations, Cost($SCStackP_i$).

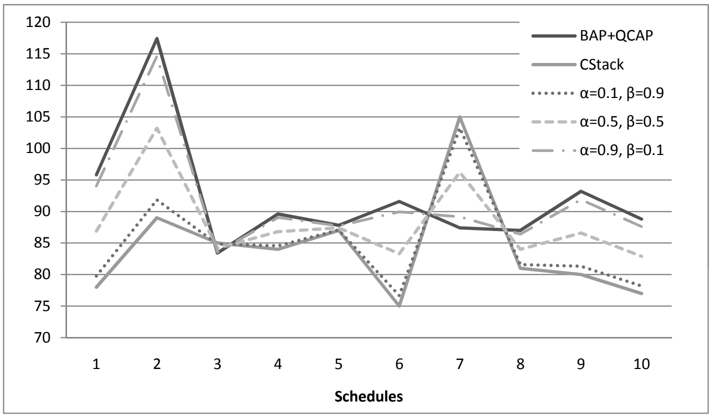


Figure 3.3: Relating the costs of BAP+QCAP and CStackP.

In each of this ten cases, the arrival times and data of vessels, as well as the initial state of the container yard, have been randomly generated. Figure 3.3 represents the combined function cost, Cost(Sol_i) with three different weights of the parameters α and β. We can see that better (or worst) berthing orders can require larger (or smaller) number of container relocations. For instance, with $\alpha = 0.5, \beta = 0.5$ the best choice is the sixth schedule. It does not get the best solution for BAP+QCAP, however it corresponds to the schedule with the smallest number of container relocations (CStackP).

In this decision support system, there are hard tasks such as obtaining the movements of relocations (remarshalling problem or CStackP) which they must be repeated for each berthing plan. Thus, in order to avoid this computation time to the decision makers (terminal operators), an estimation of the number of relocations or reshuffles needed for a certain layout of the storage yard is necessary. To this end, an estimator R for the Container Stacking Problem was defined in (Salido, Rodriguez-Molins, and Barber 2012). This estimator depends on the following four parameters:

- Total slots in a yard-bay (maximum number of allowed containers).
- Current containers stacked in the yard-bay.
- Goal (or export) containers to be loaded into the next container ship.
- Containers on top of other goal containers.

Once the system returns a solution, the planner must be executed on the actual layout of the container yard (CStackP) to determine the sequence of movements or reshuffles to be carried out.

3.4 Robust scheduling

Real-world problems are essentially non-deterministic, uncertain and imprecise: there might be unknown or incorrect information, breakdowns or failures of the resources, incidences or changes of the constraints, etc (Verfaillie and Jussien 2005). This dynamism of the real-world environments are addressed with new optimization techniques and classified according to reactive and proactive approaches (Lambrechts, Demeulemeester, and Herroelen 2008).

On the one hand, proactive approaches are those approaches that try to accommodate uncertainties or incidences in advance. On the other hand, reactive approaches react after the incidence. Thereby, proactive approaches aim to build schedules protected against disruptions during the actual scheduling execution. Reactive approaches are focused on restoring the feasibility after the occurrence of a disruption.

In this thesis, we focused on the proactive approach. Furthermore, it is assumed that there is no previous knowledge about the type or magnitude of the possible incidences (breakdowns, disruptions, extreme weather events, etc.). Thus, our aim was to offer robust schedules to the decision makers.

Our approach is based on hybrid multi-objective metaheuristics. The assessment of this algorithm was carried out in three phases:

- comparing it with a mathematical model (mixed integer programming model) which extends the mathematical model presented in Section 3.2.
- assessing the differences with two well-known multi-objective genetic algorithms (NSGAII and SPEA2+) by means of non-parametric statistical tests.
- simulating different incidences over the schedules obtained by our approach.

The dynamic and continuous BAP+QCAP problem from the container terminal was adopted as a case study in order to introduce the developed robustness model (hereinafter named as robust BAP+QCAP). Robust BAP+QCAP is characterized by three conflicting objective functions:

- from the BAP+QCAP: minimize the total service time (T_s) which includes handling and waiting times;
- from the robustness model:
  - maximize the robustness measure which is related to the size of the buffer among vessels (R).
  - minimize the standard deviation of the robustness measures (σ).

Due to the lack of the available information about how likely the incidences or breakdowns occur, it is interesting that these buffers are proportionally distributed among all the vessels. Thereby, a third objective is introduced into the model in order to improve the robustness of a schedule.

Note that all objective functions must be in similar ranges in order to obtain proper results. Thereby, all objective functions were normalized into the interval $[0, 1]$ (Rodriguez-Molins, Salido, and Barber 2014b).

Two different approaches were developed to solve this problem (Rodriguez-Molins, Salido, and Barber 2014b);

- a hybrid multi-objective metaheuristic, in particular a multi-objective genetic algorithm extended with a multi-objective local search[2] (MOGA+SA), was implemented in C++;
- a Mixed-Integer Programming (MIP) model was implemented in IBM ILOG CPLEX Optimization Studio.

[2]This multi-objective local search is based on the multi-objective simulated annealing proposed by (Bandyopadhyay et al. 2008) (AMOSA).

The evaluation of the robust BAP+QCAP model presented in (Rodriguez-Molins, Salido, and Barber 2014b), as a multi-objective optimization problem, requires the study of the dominance of solutions as well as the Pareto fronts. Thus, Figure 3.4 shows the Pareto fronts obtained of a representative instance by both the MOGA+SA and CPLEX. This Pareto front is an important tool for decision makers, to assess the trade-off between $\hat{T}_s$ (normalized total service time) and $\hat{R}$ (normalized robustness) of the different schedules obtained according to their preferences. The Pareto front of the CPLEX consists of the non-dominated solutions obtained by solving the mathematical model with different λ values ranging from 0 to 1.

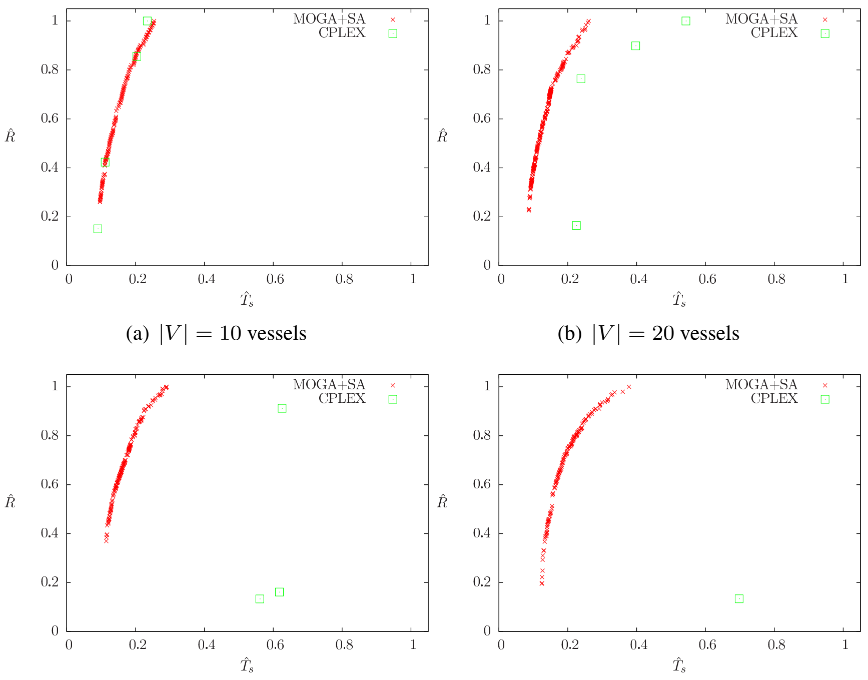
In this experiment, the timeout for the CPLEX solver was set to 1000 seconds for each λ value. It is important to note that the greater the incoming vessels, the fewer solutions obtained by CPLEX solver. Given this timeout, CPLEX was only able to get optimal solutions when $\lambda = 0.0$ and the incoming vessels was set to 10 and 20. Considering the Pareto fronts obtained by MOGA+SA and CPLEX, they were very similar with a queue of 10 vessels (see Figure 3.4(a)). However, for instance with a queue 20 vessels, the solutions obtained by CPLEX were not able to reach the quality of the Pareto front of MOGA+SA (see Figure 3.4(b)). Furthermore, it turned out that for 40 incoming vessels just one non-optimal solution was obtained (see Figure 3.4(d)); and even more there was no solution with 50 vessels.

These Pareto fronts show how the MOGA+SA approach achieved more schedules spread through the front and schedules with a better trade-off between the two conflicting objective functions $\hat{T}_s$ and $\hat{R}$ than the mathematical model (MIP).

Multi-objective optimization algorithms are not comparable directly since there is no a unique optimal solution. (Zitzler, Knowles, and Thiele 2008) propose different measures to compare Pareto front approximations. Among these measures, the size of the dominated space or the hypervolume is one of the most used measures to differentiate two algorithms (While, Bradstreet, and Barone 2012). This measure is related to a reference point p and it is set according to the suggestion of (While, Bradstreet, and Barone 2012). To this end, for each objective, it is chosen the worst value from any of the sets being compared and increasing by an ϵ value.

Comparison among these different schemes has been performed by using the Kruskal-Wallis' non-parametric statistical test, according to (Zitzler, Knowles, and Thiele 2008). This test assesses whether there are significant differences among different sets of values: in this case, sets of hypervolume measures. Table 3.5 shows the values obtained for this test given the results after solving five instances of 50 vessels with the three different algorithms. Kruskal-Wallis test revealed a significance effect of the algorithms on the hypervolumes (p-value< 0.01).

As there was a significance difference among them, a post-hoc test using a pairwise comparison test (Wilcoxon) with Bonferroni correction was carried out and showed the sig-



(a) $|V| = 10$ vessels

(b) $|V| = 20$ vessels

(c) $|V| = 30$ vessels

(d) $|V| = 40$ vessels

Figure 3.4: Pareto fronts of GA and CPLEX varying the number of incoming vessels ($|V|$).

nificant differences between the different algorithms. As example, Table 3.6 shows the results of the Wilcoxon test for the fifth instance. Note, MOGA+SA algorithm produces Pareto fronts which are statistically different with respect to the other algorithms. Examining the average values in Table 3.5, it can be determined that MOGA+SA obtained better Pareto front approximations.

These differences can be observed in the actual Pareto fronts obtained for a representative instance. Figure 3.5 shows the Pareto fronts obtained by NSGAII and MOGA+SA algorithms. The schedules with the minimum and maximum values for each objective are highlighted by circles. It is important to note that MOGA+SA algorithm was able to produce a Pareto front with higher quality. Figure 3.5(a) and Figure 3.5(b) show the relationship between $\hat{R}$ and $\hat{T}_s$. MOGA+SA algorithm turned out to achieve schedules with greater robustness and lower $\hat{T}_s$ values ($\hat{R} = 1; \hat{T}_s = 0.3831$) than NSGAII algorithm ($\hat{R} = 0.9227; \hat{T}_s = 0.4196$).

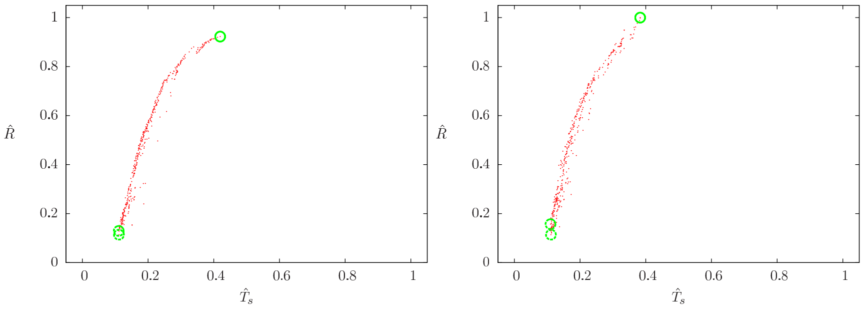
The next step is assess how these robust schedules, built by the MOGA+SA, face the incidences or disruptions that could happen during the execution of the schedule. In

Table 3.5: Kruskal-Wallis test over the hypervolumes obtained in 5 different instances

Instance	p-value	Avg. Hypervolume		
		NSGAII	SPEA2+	MOGA+SA
1	$< 2.2e-16$	0.175315	0.207921	0.236660
2	$< 2.2e-16$	0.174893	0.204749	0.252922
3	$< 2.2e-16$	0.178945	0.216683	0.261251
4	$< 2.2e-16$	0.180473	0.221138	0.258453
5	$3.699e-15$	0.160139	0.173000	0.223231

Table 3.6: Wilcoxon test over the hypervolumes obtained for a given instance

	NSGAII	SPEA2+
SPEA2+	0.00011	-
MOGA+SA	$< 2e-16$	$< 2e-16$



(a) NSGAII (b) MOGA+SA

Figure 3.5: Pareto fronts obtained by using or not local search.

this experiment, 100 instances of 100 vessels were evaluated. For each instance, three different schedules were chosen from the set of efficient solutions of the MOGA+SA according to their robustness: the one with the minimum robustness (R_{min}), the one with the maximum robustness (R_{Max}) and one intermediate robust schedule (R_i where $R_{min} < R_i < R_{Max}$).

The performance (robustness) of the schedules obtained by our approach (MOGA+SA) was evaluated by generating actual scenarios with some incidences in the actual handling time of the vessels. The incidences (delays, d) introduced were randomly chosen from a range. This range varies from a minimum value (1) to a maximum value, which is related to the handling time (h_i) of the vessel affected by the incidence (first column of Table 3.7). For each range, 100 incidences were uniformly created and applied to the three schedules (R_m, R_i and R_M) of each instance.

Table 3.7: Percentages of incidences absorbed in schedules of 100 vessels.

Range	$\mathbf{R_{min}}$	$\mathbf{R_i}$	$\mathbf{R_{Max}}$
$d \in [1,\ 0.2h_i]$	21.78	95.51	99.95
$d \in [1,\ 0.5h_i]$	18.73	93.58	99.85
$d \in [1,\ 0.8h_i]$	16.64	90.49	98.96
$d \in [1,\ 1.0h_i]$	13.92	87.10	98.01
$d \in [1,\ 1.2h_i]$	12.93	85.31	97.15

Table 3.7 shows the percentage of incidences absorbed by each type of schedule. It can be observed that the more robust schedule, the more incidences absorbed. For instance, with delays $d \in [1, 0.5h_i]$, the R_{min} schedule only absorbed 18.73% of incidences in average, but the R_{Max} schedule absorbed up to 99.85.%. Note that as the delay became larger, fewer schedules can absorb the incidences. With delays in the range of $[1, 0.2h_i]$, the R_{Max} schedule can absorb 99.95% of incidences in average. However, with larger ranges, the incidences absorbed decreased down to 97.15% in average.

Chapter 4

Conclusions

In this thesis, our main challenge has been the development of different techniques optimization for solving planning and scheduling problems as well as the search of robust solutions to deal with real-world instances of optimization problems.

As No-Free-Lunch theorem states (Wolpert and Macready 1997), a general purpose universal optimization is theoretically impossible, and the only way one strategy can outperform another is by building special purpose methods or algorithms to solve application-specific problems (Whitley and Watson 2005). In Artificial Intelligence community an expression is frequently used to reflect this fact: *Knowledge is Power*.

In this thesis, a domain-dependent planner is presented to obtain optimized and efficient solutions by means of local heuristics. This automatic planner can help to take decisions in real-world problems, such as port operations in container terminals (e.g. Container Stacking Problem or CSP). Moreover, it can help to handle different requirements and criteria, to simulate operations, to obtain conclusions about the operation of the terminal, to evaluate alternative configurations, to obtain performance measures, etc. For instance, in (Salido, Sapena, et al. 2009) the proposed planner was applied to evaluate different alternatives for the container stacking by minimizing the distance to the loading/unloading point of containers.

Well-known metaheuristics from the literature were studied and new metaheuristics were developed to achieve near-optimal solutions for hard combinatorial optimization problems. In particular, two metaheuristics from different fields were chosen: a constructive metaheuristic (GRASP) and a population-based metaheuristic (GA).

Actually, these optimization problems are harder than the ones that scientific community has been studying up to now. Usually, it is assumed that the environments were schedules or plans are going to be executed are deterministic, but this is not a real scenario. A real scenario or environment is non-deterministic, dynamic and imprecise. Therefore, techniques to deal with these situations must be designed. In this thesis, a robustness model based on buffers between tasks has been adopted and a proactive approach has been developed.

The robustness model presented in this thesis does not take into account any previous knowledge of the incidences. It assesses the buffers between two tasks which share any resource according to the duration time of the previous task. This model relies on the assumption that it is high likely that one task will not need a buffer of 10 hours if its expected duration is just one hour. Thereby, these buffers should be distributed among vessels proportionally.

Introducing this model into a combinatorial optimization problem makes necessary to adopt multi-objective optimization techniques. New objective functions must be handled along with the former objective function from the actual problem. In this thesis, a hybrid multi-objective genetic algorithm has been developed in order to offer to the decision makers a set of efficient solutions from the Pareto front. Solutions that vary between the one that could not absorb any incidence or disruption and solutions which could keep its execution after any limited incidence. This model was applied to the Berth Allocation and Quay Crane Assignment Problems (BAP+QCAP).

Proactive approaches offer the possibility to the decision makers to set a determined schedule able to deal with unexpected events. However, these approaches present some limitations, the incidences which are able to resist those schedules are usually limited. That's why these approaches should be applied together with reactive approaches. Once an unexpected event has made fail the execution of the schedule, the original schedule might be modified in order to face with this new scenario with minimum variations.

Real-world environments consist of several optimization problems. For instance, decision makers in container terminals must face with several problems at once: berthing allocation problem, quay crane assignment problem, quay crane scheduling problem, vehicle routing problem, etc. Usually, scientific community solves each problem independently. However, optimizing one of them (minimizing or maximizing) does not need to lead to the optimal solution. In this thesis, a decision support system has been developed to handle two different optimization problems from the container terminals by using the above approaches: BAP+QCAP and Container Stacking Problem. This decision support system is a cyclic process and obtains a solution to the BAP+QCAP and then it carries out the CSP to minimize the number of reshuffles needed for the obtained berth plan previously. By combining these optimized solutions in our integrated system, terminal operators can be assisted to decide the most appropriated solution in each particular scenario.

References

Aarts, E. and J. Lenstra (2003). *Local search in combinatorial optimization*. Princeton University Press.

Asariotis, R., H. Benamara, J. Hoffman, A. Jaimurzina, A. Premti, J. M. Rubiato, V. Valentine, and F. Youssef (2013). “Review of Maritime Transport, 2013”. In: *Review of Maritime Transport*.

Bandyopadhyay, S., S. Saha, U. Maulik, and K. Deb (2008). “A simulated annealing-based multiobjective optimization algorithm: AMOSA”. In: *Evolutionary Computation, IEEE Transactions on* 12.3, pp. 269–283.

Barichard, V. and J.-K. Hao (2002). “Un algorithme hybride pour le problème de sac à dos multi-objectifs”. In: *Huitiemes Journées Nationales sur la Résolution Pratique de Problemes NP-Complets JNPC*.

Bierwirth, C. and F. Meisel (2010). “A survey of berth allocation and quay crane scheduling problems in container terminals”. In: *European Journal of Operational Research* 202.3, pp. 615–627.

Blum, C. and A. Roli (2003). “Metaheuristics in combinatorial optimization: Overview and conceptual comparison”. In: *ACM Computing Surveys (CSUR)* 35.3, pp. 268–308.

Blum, C., J. Puchinger, G. R. Raidl, and A. Roli (2011). “Hybrid metaheuristics in combinatorial optimization: A survey”. In: *Applied Soft Computing* 11.6, pp. 4135–4151. DOI: 10.1016/j.asoc.2011.02.032.

Carlo, H. J., I. F. Vis, and K. J. Roodbergen (2014). “Transport operations in container terminals: Literature overview, trends, research directions and classification scheme”.

In: *European Journal of Operational Research* 236.1, pp. 1–13. DOI: 10.1016/j.ejor.2013.11.023.

Coello Coello, C. (Feb. 2006). "Evolutionary multi-objective optimization: a historical view of the field". In: *Computational Intelligence Magazine, IEEE* 1.1, pp. 28–36.

Ehrgott, M. and X. Gandibleux (2008). "Hybrid metaheuristics for multi-objective combinatorial optimization". In: *Hybrid metaheuristics*. Springer, pp. 221–259.

Feo, T. and M. Resende (1995). "Greedy randomized adaptive search procedures". In: *Journal of Global Optimization* 6.2, pp. 109–133.

Gambardella, L. M., É. Taillard, and G. Agazzi (1999). "Macs-vrptw: A multiple colony system for vehicle routing problems with time windows". In: *New ideas in optimization*. Citeseer.

Gerevini, A., A. Saetti, and I. Serina (2003). "Planning Through Stochastic Local Search and Temporal Action Graphs in LPG". In: *Journal of Artificial Intelligence Research (JAIR)* 20, pp. 239–290.

Ghallab, M., A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins (1998). "PDDL - The Planning Domain Definition Language". In: *AIPS-98 Planning Committee*.

Ghallab, M., D. Nau, and P. Traverso (2004). *Automated planning: theory & practice*. Elsevier.

Glover, F. and M. Laguna (1999). "Tabu Search". English. In: *Handbook of Combinatorial Optimization*. Ed. by D.-Z. Du and P. Pardalos. Springer US, pp. 2093–2229. DOI: 10.1007/978-1-4613-0303-9_33.

Goldberg, D. (1985). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley.

Gomes, C. (2000). "Artificial intelligence and operations research: challenges and opportunities in planning and scheduling". In: *The Knowledge Engineering Review* 15.1, pp. 1–10.

Hart, J. and A. Shogan (1987). "Semi-greedy heuristics: An empirical study". In: *Operations Research Letters* 6.3, pp. 107–114.

Henesey, L. (2006a). “Overview of Transshipment Operations and Simulation”. In: *Med-Trade Conference, Malta, April*, pp. 6–7.

Henesey, L. (2006b). *Multi-agent systems for container terminal management*. Citeseer.

Hoffmann, J. (2003). “The metric-FF planning system: translating "Ignoring delete lists" to numeric state variables”. In: *J. Artif. Int. Res.* 20.1, pp. 291–341.

Kim, K. and J. Bae (1998). “Re-marshaling export containers in port container terminals”. In: *Computers & Industrial Engineering* 35.3-4. Selected Papers from the 22nd ICC and IE Conference, pp. 655–658. DOI: 10.1016/S0360-8352(98)00182-X.

Kim, K. and K. Moon (2003). “Berth scheduling by simulated annealing”. In: *Transportation Research Part B: Methodological* 37.6, pp. 541–560.

Kirkpatrick, S., C. Gelatt, and M. Vecchi (1983). “Optimization by Simulated Annealing, Science, Vol. 220”. In: *Number* 4598, pp. 671–680.

Lambrechts, O., E. Demeulemeester, and W. Herroelen (2008). “Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities”. In: *Journal of scheduling* 11.2, pp. 121–136.

Lim, A. (1998). “The berth planning problem”. In: *Operations Research Letters* 22.2-3, pp. 105–110. DOI: 10.1016/S0167-6377(98)00010-8.

Pacheco, J. and R. Martí (2006). “Tabu search for a multi-objective routing problem”. In: *Journal of the Operational Research Society* 57.1, pp. 29–37.

Pinedo, M. (2005). *Planning and scheduling in manufacturing and services*. Vol. 24. Springer.

Pinedo, M. L. (2012). *Scheduling: theory, algorithms, and systems*. Springer.

Richter, S. and M. Westphal (2010). “The LAMA planner: Guiding cost-based anytime planning with landmarks”. In: *Journal of Artificial Intelligence Research* 39.1, pp. 127–177.

Rodriguez-Molins, M., F. Barber, M. Sierra, J. Puente, and M. Salido (2012). “A Genetic Algorithm for Berth Allocation and Quay Crane Assignment”. In: *Advances in Artificial Intelligence - IBERAMIA 2012*. Vol. 7637. Springer Berlin Heidelberg, pp. 601–610. DOI: 10.1007/978-3-642-34654-5_61.

Rodriguez-Molins, M., M. A. Salido, and F. Barber (2012). “Intelligent planning for allocating containers in maritime terminals”. In: *Expert Systems with Applications* 39.1, pp. 978–989. DOI: 10.1016/j.eswa.2011.07.098.

Rodriguez-Molins, M., M. A. Salido, and F. Barber (2014a). “A GRASP-based Metaheuristic for the Berth Allocation Problem and the Quay Crane Assignment Problem by Managing Vessel Cargo Holds”. In: *Applied Intelligence* 40.2, pp. 273–290. DOI: 10.1007/s10489-013-0462-4.

Rodriguez-Molins, M., M. Salido, and F. Barber (2014b). “Robust scheduling for Berth Allocation and Quay Crane Assignment Problem”. In: *Mathematical Problems in Engineering* 2014, pp. 1–17. DOI: 10.1155/2014/834927.

Rushton, A., P. Croucher, and P. Baker (2006). *The handbook of logistics and distribution management*. Kogan Page Ltd.

Salido, M. A., M. Rodriguez-Molins, and F. Barber (2011). “Integrated intelligent techniques for remarshaling and berthing in maritime terminals”. In: *Advanced Engineering Informatics* 25.3, pp. 435–451. DOI: 10.1016/j.aei.2010.10.001.

Salido, M. A., M. Rodriguez-Molins, and F. Barber (2012). “A decision support system for managing combinatorial problems in container terminals”. In: *Knowledge-Based Systems* 29, pp. 63–74.

Salido, M. A., O. Sapena, M. Rodriguez, and F. Barber (2009). “A Planning Tool for Minimizing Reshuffles in Container Terminals”. In: *Tools with Artificial Intelligence, 2009. ICTAI ’09. 21st International Conference on*, pp. 567–571. DOI: 10.1109/ICTAI.2009.53.

Sapena, O. and E. Onaindía (2002). “Domain-Independent Online Planning for STRIPS Domains”. English. In: *Advances in Artificial Intelligence — IBERAMIA 2002*. Vol. 2527. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 825–834. DOI: 10.1007/3-540-36131-6_84.

Schmidt, G. and W. E. Wilhelm (2000). “Strategic, tactical and operational decisions in multi-national logistics networks: a review and discussion of modelling issues”. In: *International Journal of Production Research* 38.7, pp. 1501–1523.

Slaney, J. and S. Thiébaux (2001). “Blocks World revisited”. In: *Artificial Intelligence* 125.1-2, pp. 119–153.

Stahlbock, R. and S. Voß (2008). “Operations research at container terminals: a literature update”. In: *OR Spectrum* 30.1, pp. 1–52.

Steenken, D., S. Voß, and R. Stahlbock (2004). “Container terminal operation and operations research-a classification and literature review”. In: *OR Spectrum* 26.1, pp. 3–49.

Surico, M., U. Kaymak, D. Naso, and R. Dekker (2007). “A bi-objective evolutionary approach to robust scheduling”. In: *Fuzzy Systems Conference, 2007. FUZZ-IEEE 2007. IEEE International*. IEEE, pp. 1–6.

Verfaillie, G. and N. Jussien (2005). “Constraint solving in uncertain and dynamic environments: A survey”. In: *Constraints* 10.3, pp. 253–281.

Vis, I. and R. De Koster (2003). “Transshipment of containers at a container terminal: an overview”. In: *European Journal of Operational Research* 147, pp. 1–16.

While, L., L. Bradstreet, and L. Barone (2012). “A fast way of calculating exact hypervolumes”. In: *Evolutionary Computation, IEEE Transactions on* 16.1, pp. 86–95.

Whitley, D. and J. Watson (2005). “Complexity Theory and the No Free Lunch Theorem”. English. In: *Search Methodologies*. Springer US, pp. 317–339. DOI: 10.1007/0-387-28356-0_11.

Wolpert, D. and W. Macready (Apr. 1997). “No free lunch theorems for optimization”. In: *Evolutionary Computation, IEEE Transactions on* 1.1, pp. 67–82. DOI: 10.1109/4235.585893.

Zhen, L. and D.-F. Chang (2012). “A bi-objective model for robust berth allocation scheduling”. In: *Computers & Industrial Engineering* 63.1, pp. 262–273.

Zhou, A., B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang (2011). “Multiobjective evolutionary algorithms: A survey of the state of the art”. In: *Swarm and Evolutionary Computation* 1.1, pp. 32–49.

Zitzler, E., J. Knowles, and L. Thiele (2008). “Quality assessment of pareto set approximations”. In: *Multiobjective Optimization*. Springer, pp. 373–404.

Index