# Mathematical Models for Maritime Terminal Operations

by

**Celal Özgür Ünsal**

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Doctor of Philosophy

in

Industrial Engineering and Operations Management


January, 2019

**Mathematical Models for Maritime Terminal Operations**

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a doctoral dissertation by

**Celal Özgür Ünsal**

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Prof. Ceyda Oğuz

_____

Prof. Necati Aras

_____

Prof. Selçuk Karabatı

_____

Assoc. Prof. Sibel Salman

_____

Asst. Prof. Elvin Çoban Göktürk

Date: _____

*To my beloved family...*

# ABSTRACT

Maritime terminals are the key components of global freight transportation as they handle over 80% of global trade by volume and more than 70% of value according to United Nations. A steadily increasing workload causes maritime terminals around the world to face with a high competitive pressure. Therefore, it is essential for them to improve their performance levels in terms of service rate and costs. Maritime terminals can be classified into two based on the transported materials: container and bulk. Differently from container terminals in which standardized containers are processed, bulk terminals deal with unpackaged natural resources and agricultural products, such as iron ore, coal, grains, oil, and gas in large quantities. Operational problems observed in both classes of maritime terminals are the variants of well studied operations research problems such as machine scheduling, vehicle routing, and bin packing. However, they are more complex in nature because of the distinctive characteristics of terminals. In this dissertation, we study three different but related planning and scheduling problems of maritime terminals with a practical relevance: quay crane assignment problem (QCAP), reclaimer scheduling problem (RSP) and integrated dry bulk terminal (IDBT) problem. In each of these problems, we deal with one of the most challenging characteristics which is caused by the fact that bottleneck equipment in both classes of terminals, namely, quay cranes (QC) and reclaimers, are mounted on the same rail track, thus their movements are restricted by their respective positions over time.

In the first chapter, we introduce container and bulk terminals by presenting a concise overview of their operations as well as the related literature. Within these sections, we also describe our motivations and contributions for QCAP, RSP, and IDBT problem, respectively. In Chapter 2, we study QCAP. In this problem, QCs

are assigned to arriving vessels and handling time of a vessel depends on the number of assigned QCs. As QCs are the bottleneck equipment in container terminals, they need to be utilized efficiently. We study a QCAP by considering its various features, differently from the literature which deals with the simplest form of the problem. With our approach, we can estimate the vessel handling times accurately and hence improve the productivity as a result. We represent this problem as a moldable task scheduling problem with contiguous assignments, and we formulate an extended time-indexed model that additionally keeps specific task to machine information. Even though extended formulation is flexible in terms of modeling different features of the problem, it has multiple computational issues. Therefore, we develop an exact solution method by first hybridizing the formulation with a set of auxiliary variables to obtain a decomposable structure, and then implementing logic-based Benders decomposition (LBBD) with strong cuts. One limitation of this proposed method is excessive memory requirements for large instances, since it is based on a time-indexed formulation. Hence, we propose methods to derive lower and upper bounds for larger problem instances in Chapter 3. Since linear programming relaxations of time indexed formulations are known to be very tight, we developed a column generation procedure in which pricing problem can be solved in polynomial time. For an upper bound, we introduce a constraint programming model that finds near optimal solutions in a short time.

In Chapter 4, we introduce the reclaimer scheduling problem. Reclaimers handle the dry bulk cargo, which is stacked as a stockpile. Reclaimers are mounted on the multiple rail tracks. If there are two reclaimers on the same rail track, then they cannot cross each other. Furthermore, a reclaimer can only handle stockpiles located adjacent to its rail track. For this strongly NP-hard problem, we opt for a heuristic approach by developing an arc-time-indexed lower bound model and constraint programming model to find near optimal solutions. There are many relations between operational problems in maritime terminals. If we solve these problems hierarchically,

we often end up with plans with poor overall quality. Accordingly, in Chapter 5, we study the integrated problem dry bulk terminal operations, by considering berth allocation, yard assignment, and reclaimer scheduling operations simultaneously. After observing a key relation between problems, we decompose the problem into two easier problems and solve with a novel LBBD. In this decomposition, we model master and subproblems with mixed-integer programming and constraint programming, respectively, by exploiting the respective advantages of programming paradigms. Results show that the proposed method is able to solve considerably large instances to optimality in an acceptable time, compared to the monolithic approach. We conclude the dissertation with Chapter 6 in which we present a concise overview of our contributions and discuss future research directions.

# ÖZETÇE

Kıyı terminalleri küresel taşımacılık ağının en önemli bileşenlerinden biridir. Öyle ki, Birleşmiş Milletler raporlarına göre dünyada gerçekleşen tüm taşımacılığın hacim olarak %80, maddi değer olarak ise %70'inden fazlası bu terminaller tarafından elleçlenmektedir. Sürekli olarak artan işyükü, kıyı terminallerini rekabetten doğan baskıya maruz bırakmaktadır. Bu sebeple, kıyı terminallerin servis oranı ve maliyetler açısından performanslarını arttırmaları son derece önemlidir. Bu terminaller taşınan materyaller bakımından konteyner ve dökme yük terminalleri olmak üzere iki sınıfa ayrılır. Bütün yüklerin standartlaşmış konteynerler ile taşındığı konteyner terminallerinden farklı olarak, dökme yük terminallerinde demir cevheri, kömür, tahıllar ve benzin türevleri gibi paketli olmayan büyük hacimlerdeki yükler elleçlenir. Kıyı terminallerinde görülen operasyonel problemler makine çizelgeleme, araç rotalama ve paketleme gibi yoğun olarak çalışılmış yöneylem araştırması problemlerinin türevleridir. Ancak terminallerin ayırt edici bazı özellikleri sebebiyle karşılaşılan problemler daha karmaşık bir yapıdadır. Bu tezde birbiriyle ilişkili üç farklı operasyonel problem üzerinde çalıştık. Bunlar rıhtım vinci atama problemi (RVAP), boşaltıcı çizelgeleme problemi (BÇP) ve bütünleşik kuru dökme yük terminali problemidir (BKDYTP). Bu problemlerin her birinde, dar boğaz ekipmanların aynı ray üzerinde çalışmaları sebebiyle ortaya çıkan ve kıyı terminallerinin en zorlaştırıcı özelliklerinden olan birbirini geçememe kısıtı karşımıza çıkmaktadır.

İlk bölümde, mevcut literatürü de inceleyerek konteyner ve kuru yük terminallerine genel bir bakış ortaya koyuyoruz. Bu kısım ayrıca RVAP, BÇP ve BKDYTP için motivasyonlarımızı ve katkılarımızı da özetlemektedir. İkinci bölümde rıhtım vinci atama problemini ele almaktayız. Terminale gelen her gemilere rıhtım vinçlerini atadığımız bu problemde, bir geminin elleçleme süresi atanan rıhtım vinçlerinin sayısına bağlı

olarak değişmektedir. Konteyner terminallerinde dar boğaz ekipman olan rıhtım vinçlerinin verimli kullanılması çok önemlidir. Bu doğrultuda, problemin en basit halini ele alan literatürden farklı olarak, çeşitli özellikleri de göz önüne aldık. Yaklaşımımız, gemilerin elleçleme sürelerini isabetli olarak hesaplamayı ve sonuç olarak da sistemin verimliliğini arttırmamıza imkan sağlıyor. Her konteyner terminalinde karşımıza çıkan bu operasyonel problemi bitişik atama kısıtlı şekillendirilebilir iş çizelgeleme problemi olarak tasvir ederek iş-makine atamaları bilgisini göz önüne alan zaman indisli bir formülasyon geliştirdik. Bu genişletilmiş formülasyon problemin farklı özelliklerini modellemeyi mümkün kılmasına rağmen bazı hesaplama zorluklarına sahiptir. Bu doğrultuda, oluşturduğumuz formülasyonu ayrıştırılabilir bir yapıya kavuşturmak için yardımcı karar değişkenleri kullanarak genişlettik. Sonrasında ise geliştirdiğimiz mantık tabanlı Benders ayrıştırmasını kullanarak eniyiledik. Bu çözüm yöntemi zaman indisli karar değişkenleri kullanan bir formülasyon üzerinden oluşturulduğu için büyük problem örneklerini çözmek aşırı bellek gereksinimleri sebebiyle mümkün olmamaktadır. Bu sebeple, 3. bölümde büyük problemler için alt ve üst sınırlar elde edebileceğimiz yöntemler geliştirdik. Çizelgeleme problemleri için zaman indisli formülasyonların sıkı doğrusal programlama gevşetmelerine sahip oldukları bilinmektedir. Bu sebeple, bir önceki bölümde sunduğumuz formülasyondan yola çıkarak, yan problemin polinom zamanda çözülebildiği etkili bir sütun oluşturma algoritması aracılığıyla alt sınır elde ettik. Üst sınır için ise, kısa sürede eniyiye yaklaşık değerler üreten bir kısıt programlama formülasyonu sunduk.

Tezin 4. bölümünde kuru dökme yük terminalinde karşılaşılan boşaltıcı çizelgeme problemini ele aldık. Boşaltıcılar yığın olarak depolanan kuru dökme yükleri elleçler. Bir stok alanında boşaltıcılar birden fazla paralel ray üzerinde bulunabilir. Ayrıca bir ray üzerinde iki boşaltıcı makine bulunabilir ve bu durumda boşaltıcılarının birbirlerini geçmeleri engellenmelidir. Bu NP-zor problem için sezgisel yöntemler geliştirmeyi tercih ettik ve alt sınır için yay-zaman indisli bir formülasyon, en iyiye yaklaşık sonuçlar bulmak için ise kısıt programlama formülasyonu geliştirdik. Kıyı termi-

nallerinde karşılaşılan operasyonel problemler arasında pek çok ilişki bulunmaktadır. Bu ilişkili problemleri hiyerarşik olarak çözdüğümüzde ise sıklıkla düşük verimli planlar elde ederiz. Bu sebeple, 5. bölümde, en önemli kuru dökme yük terminal operasyonları olan rıhtım atama, depo atama ve boşaltıcı çizelgeleme problemlerini bütünleşik olarak ele aldık. Bu problemler arasındaki kritik ilişkileri tespit ettikten sonra bütünleşik problemi görece daha kolay iki probleme ayırdık ve mantık tabanlı Benders ayrıştırması ile eniyiledik. Bu yöntemde ana problemi karışık-tam sayılı doğrusal programlama, yan problemi ise kısıt programlama ile çözerek bu programlama tekniklerinin birbirlerine olan avantajlarını ortaya çıkardık. Nümerik sonuçlar geliştirdiğimiz ayrıştırma yönteminin bu zorlu bütünleşik problemi iki haftalık bir planlama periyodu için kısa bir sürede çözebildiğini gösterdi. Çalışmamızı literatüre olan katkılarımızı özetlediğimiz ve gelecek araştırma imkanlarına değindiğimiz 6. bölüm ile noktaladık.

# ACKNOWLEDGMENTS

First, I would like to gratefully thank my advisor Prof. Ceyda Oğuz for her guidance, support, and patience.

I also would like to express my appreciation to my thesis committee members Prof. Selçuk Karabatı and Prof. Necati Aras for spending their precious time to guide me with their insightful comments and examination committee members Assoc. Prof. Sibel Salman and Asst. Prof. Elvin Çoban Göktürk for their valuable suggestions that led me to improve the dissertation.

This thesis is financially supported in part by TÜBİTAK 1001 grant 113M486.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

Chapter 1

# INTRODUCTION

Over the last few decades, the distance between production and consumption points has increased because of globalized economy and maritime transportation became more important. According to UNCTAD (2017), more than 80% of global trade by volume and 70% by value are carried by sea and handled at maritime terminals.

Maritime terminals are the key components of global freight transportation, since they connect sea-freight and means of land transportation. We can classify maritime terminals into two based on the transported materials: container terminals and bulk terminals. Container terminals are the facilities where containers are transshipped between different means of transport for onward transportation. Bulk terminals handle unpackaged natural resources and agricultural products, such as iron ore, coal, grains, oil, and gas in large quantities.

Total volume of maritime transport has increased by a yearly average of 3% over the past four decades. This results in an increasing workload for maritime terminals. Figure 1.1 illustrates this growth in maritime transport over the years for containers and dry bulk loads (in millions of tons). The upward trend causes maritime terminals around the world to face with a high competitive pressure. Therefore, it is essential for them to improve their performance levels in terms of service rate and costs. Otherwise, inefficient ones are expected to lose potential customers to better performing nearby terminals.

A maritime terminal is a complex facility in which a set of logistic processes are performed. Operations research (OR) methods provide opportunities to these complex facilities for managing their operations efficiently (Steenken et al., 2004). OR

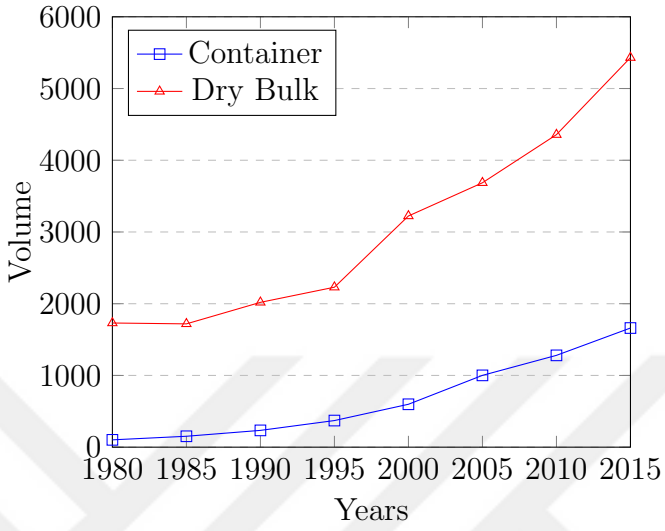



Figure 1.1: Growth of maritime transportation

methods used in terminals include but not limited to various optimization algorithms, queuing and simulation models. The rapid increase in number of related publications in the last two and a half decades indicates the importance of implementing OR methods in maritime terminals (WOS, 2018). Today, the number of terminals that utilize these methods is growing, in line with the success of previous implementations and easier accessibility to modern information technologies.

Operational problems observed in maritime terminals are the variants of well studied OR problems such as machine scheduling, vehicle routing, and bin packing. However, they are more complex in nature because of the distinctive characteristics of terminals (Meisel and Bierwirth, 2010). One of the most challenging characteristic is caused by the fact that bottleneck equipment in both classes of terminals, namely, quay cranes (QC) and reclaimers, are mounted on the same rail track. Hence, their movements along the rail track are limited (Unsal and Oguz, 2013).

In this dissertation, we study three challenging planning and scheduling problems of maritime terminals with a practical relevance: quay crane assignment problem (QCAP), reclaimer scheduling problem (RSP) and integrated dry bulk terminal (IDBT) problem. What these problems have in common is the existence of the char-

acteristic that we previously described. In the following, we introduce container and bulk terminals by presenting a concise overview of their operations as well as the related OR literature. Within these sections, we also describe our motivations and contributions for QCAP, RSP, and IDBT problem, respectively.

## *1.1 Container Terminals*

As the first container vessel is docked in 1956, containerization of cargoes is becoming ever more popular. Today, almost all types of goods are transported by containers. A container is a rectangular box which is used for transporting goods with different means of transport.

Total workload and capacity of the container vessels are represented in TEUs. A TEU stands for a 20-foot (6.1 meters) container. In addition to a standard 1-TEU container, other common types of containers are 2-TEU, open top, refrigerated, and flat rack. All of these types of containers are manufactured with respect to some international standards; thus, they are interchangeable between different companies and carriers. This standardization helps to carry and handle containers easily by various means of transportation, such as vessels, trains, and trucks. Furthermore, containers are manufactured from very strong materials. For this reason, they can be stacked easily. A container is also known to be the most secure way to transport goods without being damaged.

A container terminal (CT) is a huge facility where containers are handled and transshipped. Most container terminals are located at a seaside. They provide a connection for outbound and inbound container traffic between sea and land transportation. In addition to that, a CT allows transshipment of containers between vessels. CTs consist of seaside (i.e., quayside) and landside. In a CT, common activities that need to be performed are loading (unloading) of containers to (from) vessels, transporting cargoes between storage and berth areas, temporarily stacking and handling them in storage area. The most general representation of a CT is depicted in Figure 1.2.

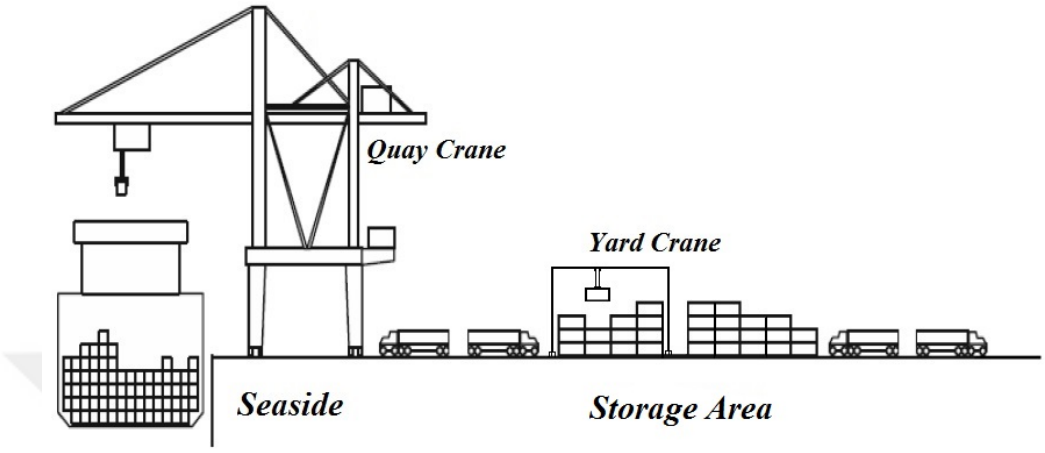



Figure 1.2: A general representation of a container terminal

At the seaside, container vessels moor to berths, and their loading/unloading operations are performed by a set of quay cranes. At the landside, there is a storage area in which containers are temporarily stored in stacks. Trucks and automated guided vehicles (AGVs) transfer containers between the seaside and the storage area continuously. Moreover, containers which are coming from (or going through) land transportation are also transferred between storage area and gates of CT by trucks and AGVs.

Containers support a vast increase in overseas trading by allowing the effective transportation of goods over the long distances. This intercontinental container traffic passes through a large number of container terminals all around the world. In Table 1.1, we list the top five terminals of the world and of Europe, in terms of annual workloads in units of thousands of TEU (Lloyd's List Intelligence, 2017). Based on the same data, Ambarli and Mersin terminals in Turkey are ranked as 54th and 96th with a combined annual throughput of 4,256,000 TEUs.

We can convert the numbers in Table 1.1 to a daily workload in terms of approximate number of vessels and containers. For example, based on 2017 data, average number of 450 container vessels arrived to Shanghai terminal daily. In other words, the terminal handled around 110,000 TEUs of containers each day. Such a huge

Table 1.1: Annual workloads of world's busiest terminals (in x1000 TEUs)

| World | | | | Europe | | | |
|---|---|---|---|---|---|---|---|
| Global Rank | Terminal, Country | 2010 | 2017 | Global Rank | Terminal, Country | 2010 | 2017 |
| 1 | Shanghai, China | 29,069 | 40,230 | 11 | Rotterdam, Netherlands | 11,146 | 13,600 |
| 2 | Singapore | 28,431 | 33,670 | 13 | Antwerp, Belgium | 8,468 | 10,450 |
| 3 | Shenzhen, China | 22,510 | 25,210 | 18 | Hamburg, Germany | 7,905 | 9,000 |
| 4 | Ningbo-Zhoushan, China | 13,144 | 24,610 | 26 | Bremerhaven, Germany | 4,871 | 5,700 |
| 5 | Busan, South Korea | 14,157 | 21,400 | 28 | Algeciras, Spain | 2,810 | 4,850 |

amount of workload suggests that the efficient operations planning is key for the success of any container terminal.

Operations research (OR) methods help terminals to improve their productivity by providing decision support systems (Stahlbock and Voss, 2008). The literature on the subject is steadily growing with the technological improvements and emerging trends in container terminals. OR problems in container terminals can be classified into two such that seaside operations and landside operations. At the landside, we mainly deal with storage operations and inter-terminal transportation. These operations are out of the scope of this dissertation, and we refer the reader to Carlo et al. (2014a, 2014b) for detailed reviews.

At the seaside of a container terminal, the literature defines operational problems of allocation of berth area to vessels, assignment of quay cranes to vessels, and scheduling of these quay cranes. In many container terminals, seaside operations are the bottleneck for the productivity (Carlo et al., 2015). All vessels that are moored at the seaside of a terminal need to be serviced with quay cranes (QCs), huge and expensive machines that unload (load) containers from (to) vessels. As a result, efficient allocation and utilization of QCs become vital for the overall performance of

terminals.

QCs move along the rail track to serve different vessels and different bays of a vessel, and in most of the cases, all QCs that serve a berth operate on the same rail track. This distinctive characteristic complicates the operational problems observed at the seaside of a terminal. Specifically, we must take two important constraints that restrict the movement of QCs into account: non-crossing and non-interference of QCs. Non-crossing constraint ensures that QCs cannot cross each other, thus they keep their respective ordering on the rail track at any time. Non-interference constraint consists of two components. First, a bay of a vessel can be serviced by at most one QC at a time. Second, a specific safety distance must be kept between two adjacent cranes.

In the following, we present a brief literature review of the related problems and our contributions on the subject. For more comprehensive reviews of OR problems on seaside operations, see Bierwirth and Meisel (2010, 2015).

**Berth Allocation Problem (BAP):**

This operational problem is to allocate mooring positions over the berth area to vessels such that vessels occupying the same berth position must be non-overlapping in time. In literature, three different berth layouts are considered: discrete (Imai et al., 2001), continuous (Park and Kim, 2003), and hybrid (Nishimura et al., 2001). Moreover, it can be assumed that all vessels are available for mooring at the beginning of planning horizon (static problem) or they arrive during the planning horizon (dynamic problem).

In discrete berth layout, each berth is a discrete resource with a single vessel capacity. Without any other constraint, a dynamic and a static BAP with a discrete berth layout are equivalent to parallel machine scheduling problem (PMSP) with and without release times, respectively. Imai et al. (2001) define and study the dynamic problem by developing a Lagrangian relaxation based heuristic. In reality, not all vessels are available at the beginning but they arrive in different time points over the planning horizon. That is the reason why the dynamic variant of the problem

is studied more. Imai et al. (2003) study the dynamic problem by also considering priorities among vessels and solve the resulting problem with a genetic algorithm. Hansen et al. (2008) develop a variable neighborhood search heuristic that generates near optimal solutions in a very short time. Bhurkal (2011) compare various mixed integer programming models and identify set partitioning formulation as the best performing one.

Continuous layout assumes that there is no partitioning of the berth area. That is, vessels can moor at arbitrary positions as long as they do not overlap in time and space. This type of layout can be modeled as a two dimensional bin packing problem where each vessel is a rectangle (Lim, 1998). In this representation, physical length and duration of stay of a vessel corresponds to width and height of a rectangle, respectively, assuming y-axis stands for the time space in corresponding bin packing problem. Lee et al. (2010) develop an effective GRASP heuristic for the dynamic problem. Imai et al. (2005) also study dynamic problem, by assuming handling time of a vessel depends on its berthing position. The same variant is also studied by Golias et al. (2009) in which an efficient genetic algorithm is presented.

Hybrid layout discretizes berth area as multiple sections of same or different sizes, similar to discrete layout. Differently, a vessel can occupy multiple berth sections at a time, based on its length. If we assume unit length berth sections, then we obtain an instance of a multiprocessor task scheduling problem with contiguous assignment of resources (Guan et al., 2002). Berth allocation with hybrid layout is studied by Imai et al. (2007), Dai et al. (2008), and Lee et al. (2012).

The main difference between these layouts is the level of utilization that they provide. Optimal solution to the problem with continuous layout provide a better utilization than those of discrete and hybrid layouts, since it can eliminate the wasted space caused by the difference between the lengths of vessel and berth section. At the same time, however, planning with continuous layout is harder than discretized layouts in general.

Furthermore, there are many other features of BAP considered in literature such

that uncertainty on arrival (Golias et al., 2014) and handling time (Zhen, 2015) of vessels, tidal time windows (Ernst et al., 2017, Song et al., 2018, Xu et al., 2012), fuel consumption and emissions (Venturini et al., 2017, Wang et al., 2018).

**Quay Crane Assignment Problem (QCAP):**

QCAP is to allocate QCs to each arriving vessel for loading/unloading operations by considering the fact that all QCs are mounted on the same rail track, thus, they cannot cross each other. In container terminals, handling time of a vessel depends on number of assigned QCs, which is a critical capacity allocation decision regarding to the bottleneck equipment. However, the relation of the number of assigned QCs to vessel and its handling time is not linear because of crane interference, and we observe marginal productivity losses as we assign more QCs to a vessel.

The only study that consider the problem individually is Blazewicz et al. (2011). They define the problem such that number of assigned QCs to a vessel does not change during its handling (time-invariant QCAP). Then, they show that the resulting problem can be modeled as a moldable task scheduling problem (MTSP) with contiguous assignments of QCs, since MTSP allows modeling number of QCs dependent processing times and can incorporate the effect of crane interference.

The rest of the literature studies this problem as a component of BAP, to determine vessel handling times. By assuming time-invariant QCAP, Liang et al. (2009) and Turkogullari et al. (2014) study the integrated problem by considering discrete and hybrid berth layouts, respectively.

On the other hand, some studies assume that the number of QCs can change during handling of a vessel (time-variant QCAP). Time-variant QCAP is first studied by Park and Kim (2003) with continuous berth layout. However, the authors ignored not only the fact that QCs operate on a same rail track but also the effect of crane interference. Meisel and Bierwirth (2009) study the same problem by taking both of these features into account. Even though time-variant QC assignment strategy can lead to better utilization compared to time-invariant, it may often not be desirable for the terminal management.

It is worth mentioning that plenty of studies solve QCAP by simply determining the number of assigned QCs without specifying vessel-QC assignments. As such an approach ignores the non-crossing of QCs, it is not guaranteed to have a feasible QC to vessel assignment even if total number of QCs assigned to vessels at any time is less than or equal to the number of available QCs (see Appendix B).

**Quay Crane Scheduling Problem (QCSP):**

As one can observe, QCAP is a berth level planning problem in which we assign a set of QCs to multiple vessels that arrive within a planning horizon. On the other hand, QCSP deals with a planning operation of a single vessel. Given a number of assigned QCs, QCSP is to schedule loading and unloading tasks of a vessel with respect to properties such as non-crossing constraint, safety distances between adjacent QCs, and travel times along the rail track.

Based on the definition of a task, we can group QCSP studies into three: complete bay, container groups, and a container. The first one defines a task as loading/unloading of all containers on a bay of a vessel. In reality, containers associated with same owner (and same destination) are stored adjacently on a bay. There can be multiple tasks on a single bay, and tasks that are located on the same bay cannot be handled at the same time. For this reason, most of the related literature defines a task as a container group. By assuming each container group as a task, a research stream that includes Kim and Park (2004), Moccia et al. (2006), Meisel and Bierwirth (2009), and Unsal and Oguz (2013) provide various solution techniques for the detailed problem with all of above mentioned properties. Legato et al. (2012) and Meisel (2011) further extend the problem considered on that stream with uniform QCs and QC unavailability, respectively.

**Integrated Operations:**

BAP, QCAP, and QCSP are highly interrelated operational problems. For example, the solution to a QCAP is an input to QCSP. QCAP and QCSP affect berth allocation decisions, as they determine handling times of vessels. If we solve these problems

hierarchically by ignoring their relations, we often end up with plans with poor overall quality (Bierwirth and Meisel, 2010). Accordingly, many studies address the deep integration of seaside operations, i.e., solving multiple problems simultaneously while considering their interrelationships. We have previously mentioned the studies that integrate BAP and QCAP. Deep integration of the time-variant QCAP with a detailed QCSP is studied by Unsal and Oguz (2013).

In one of the most comprehensive studies of seaside operations, Meisel and Bierwirth (2013) propose a framework to integrate all seaside operations by an algorithm which consists of preprocessing and feedback loops among problems. Note that their method is not exact since they utilize existing heuristics to solve various subproblems within their algorithm. Still, they show that it performs much better than the corresponding hierarchical approach, highlighting the importance of integrated planning once more.

**Our Contributions on Container Terminal Operations:**

As QCs are the bottleneck equipment in container terminals, they need to be utilized efficiently. Up to now, the literature deals with the simplest form of the QC allocation operation by assuming all QCs are identical and are available during the planning horizon. On the other hand, in real world terminals, we deal with the various features of the problem such as QCs with different processing rates, periodic maintenance of QCs, and QC eligibility. If we ignore these features, we cannot estimate the vessel handling times accurately and experience a loss of productivity in terminal operations as a result.

In this dissertation, to fill this gap in the literature, we study QCAP in more details to be able to cope with above mentioned features of the problem. We represent this problem as a variant of moldable task scheduling problem (MTSP) with contiguous assignments of QCs. We develop a flexible and robust logic-based Benders decomposition with strong cuts, based on the extended formulation of the MTSP that specifies task to machine assignments. Furthermore, we present a compact contiguous assignment constraint for the assignments of QCs. One limitation of the proposed

solution method is the excessive memory requirements for the large instances, since it is constructed with time-indexed variables. To solve the larger problem instances, we develop a constraint programming (CP) model to find near optimal solutions in a short time and a column generation procedure that provides very tight lower bounds that helps us to measure the performance of the CP model accurately.

## *1.2 Bulk Terminals*

Bulk terminals are seaside facilities in which products such as agricultural products and natural resources are stored and transshipped in very large volumes. Most of the cargo that is handled in these terminals is the major import and export products such as coal, grains, iron ore, and petroleum products. Each of these products is only exported by a few countries. For example, 82% of all iron ore is exported from Australia and Brazil, while 71% is imported by China. For this reason, bulk loads consist of 83% of all sea-freight in volumes. This indicates the critical role of bulk terminals on the global trade.

Similar to container terminals, bulk terminals also consist of seaside and yard area. However, they differentiate in terms of the type of cargo. In contrast to container terminals in which all cargo are transported in a standardized container, means of storage and handling of cargo varies for different types of materials in bulk terminals.

Bulk terminals can be classified as dry bulk terminals and gas/liquid terminals. In this dissertation, we consider dry bulk terminals (DBT) that handle dry loads such as grains, coals, and iron ore. We consider an export DBT, and the general structure of the terminal and the flow of loads can be described as follows: Dry cargo arrives to the terminal by train and is stored in the stockyard as stockpiles. After mooring of the vessels to the berths, related stockpiles are reclaimed by reclaimers and transferred to the quay area via conveyor belt systems where vessels are loaded by ship loaders. Berth allocation in DBTs is often affected by tidal conditions; that is, departure of a vessel is constrained by high tide periods. Moreover, there are very limited number of ship loaders, in contrast to the number of QCs in a typical container terminal.

Therefore, the number of vessels that can be loaded simultaneously is often limited by the number ship loaders, rather than the length of berth area.

Figure 1.3: A typical dry bulk terminal that exports coal (downloaded from http://shippingherald.com in May 2018)

The major operational problems observed in dry bulk terminals can be given as berth allocation, reclaimer scheduling, and yard allocation. Other problems observed in these terminals are blending of loads, design of stockyard and conveyor belt systems. Even though bulk loads constitute the largest portion of all loads maritime transportation, operational problems of bulk terminals are studied less in the literature compared to those of container terminals. As the OR literature on the subject only consists of a handful of studies, we will introduce them within the related chapters of the dissertation.

Our contribution on the subject is two fold. We first study reclaimer scheduling problem (RSP) to determine schedules for reclaiming stockpiles from the stockyard. Then, we study integrated operations planning problem of dry bulk terminals.

In our previous work, we show that constraint programming (CP) is a suitable tool to cope with non-crossing constraint (Unsal and Oguz, 2013). Based on this finding, we develop a CP model that can find near optimal solutions to RSP. We also present an arc-time-indexed lower bound model by relaxing non-crossing constraint of the problem to test the performance of the CP model. To the best of our knowledge, this is the first RSP study that considers all realistic features of the problem. Results of computational experiments indicate that the proposed method is able to provide quality solutions for different stockyard configurations within a minute.

It is easy to see that operational problems in these terminals are also strongly interrelated. For example, the schedule of the reclaimers will directly affect the loading of the vessels, and in turn the time that a vessel can leave the berth. Similarly, decisions on the positioning of stockpiles over the stockyard affect reclaimer scheduling, since they generate non-eligibility of some stockpile-reclaimer assignments.

Accordingly, we study the integrated dry bulk terminal (IDBT) problem. It consists of the deep integration of three important DBT operations, namely, berth allocation, yard allocation, and reclaimer scheduling, by taking various characteristics of the respective problems into account. After observing a key relation between problems, we decompose the integrated problem into two easier problems and develop a novel logic-based Benders decomposition. In this decomposition, we model master and subproblems with mixed-integer programming and CP, respectively, by exploiting the respective advantages of these programming paradigms. We develop CP subproblem by simply extending the model that we provide for RSP. Results show that the proposed method is able to solve considerably large instances to optimality in an acceptable time, compared to the monolithic approach. This is one of the very few studies in the whole maritime terminals literature and the first in bulk terminals literature that intends to simultaneously solve three operational problems to optimality.

## 1.3 Methodology

In this section, we introduce contraint programming (CP) and logic-based Benders decomposition (LBBD) techniques which we use in the multiple sections of the dissertation. We use CP (*i*) to find near optimal solutions to QCAP and (*ii*) RSP, as well as (*iii*) to solve a feasibility subproblem to LBBD implementation of IDBT problem. We implement LBBD with different decomposition structures to solve QCAP and IDBT problem to optimality, respectively.

In addition to these two methods, we also use column generation to solve QCAP. We left its description to the beginning of the related section.

### 1.3.1 Constraint Programming

Constraint programming (CP) is a technique that can be used for representing and solving combinatorial problems which are hard to solve. Similar to mathematical programming, CP is a combination of defining constraints about the problem via variables and finding a solution that satisfies all constraints. However, in CP, constraints are used actively to infer new constraints and to reduce domains of variables by removing certain values that violate constraints. Differently from the mathematical programming, the main focus of CP is on constraints and feasibility, rather than the objective function and the optimality.

Constraint programming also provides rich modeling tools (Lustig and Puget, 2001) to represent complex combinatorial problems compactly. In a nutshell, CP can be viewed as a heuristic method that consists of three main components: modeling, filtering of domains of variables based on the constraints, and the search phase that utilizes backtracking. In the following, some concepts related to the methodology used in this section are briefly introduced. For more detailed information on CP technique, see Apt (2003), Baptiste et al. (2001), and Unsal (2013).

**Activities and Interval Variables:**

Scheduling problems in CP can be represented in terms of activities and resources.

For example, if we consider QCAP, an activity corresponds to the complete handling operations of a vessel, and resources correspond to quay cranes. In the following, we define some important concepts of CP which we used in this dissertation. Each activity $A_i$ has a start time $start(A_i)$, a processing time $p(A_i)$, and an end time $end(A_i) = start(A_i) + p(A_i)$. Each activity also has a discrete domain of $domain(A_i)$ which represents the time interval that task $i$ can be processed.

An interval variable (Laborie and Rogerie, 2008) is an interval of time during which an activity is executed. The decision here is to select a time interval during the planning horizon to execute this activity. Each interval variable is characterized not only by a start time, an end time, and a processing time but also with a presence information, $presence(A_i)$. An interval variable can be optional; therefore, the activity can be left unexecuted. In these situations, unexecuted activity is indicated by $presence(A_i) = False$ (or 0). On the other hand, for all executed optional interval variables, $presence(A_i)$ takes the value of $True$ (or 1). This property of an interval variable helps us to model the problem when there are activities that can be executed on a set of alternative resources.

**Global Constraints:**

A global constraint in CP can represent the complex relationships between the problem variables as a single constraint. As global constraints deal with many problem variables at the same time, they provide faster and effective domain reduction by using specialized filtering algorithms. The most widely known global constraint is $allDifferent(\forall v_i)$, where $v_i, i \in \{1, \ldots, n\}$, is a decision variable with $domain(v_i)$ includes the possible values that $v_i$ can take. It ensures that each variable $v_i$ must take a different value from its domain. In contrast, $n.(n-1)/2$ constraints are required to represent such a relationship within a mixed-integer programming model. Global constraints have many specialized filtering algorithms; e.g. Lopez-Ortiz et al.. (2003) for $allDifferent$ constraint. This kind of algorithms usually offers more effective domain reductions. Therefore, in constraint programming, it is important to represent the model with global constraints as much as possible.

There are many global constraints in the literature and 6 of them are considered in this dissertation. The $alternative(A_i, (Y_{ij}|\forall j \in S_i), Z_i)$ constraint (Beck and Fox, 1999) simply assigns each activity $i$ to $Z_i$ different resources within a subset of resources which are elements of set $S_i$. The property of optionality of the interval variables makes sense here. Assume that activity $i$ is assigned to an only one resource ($j^{'}$). Then, $Y_{ij}$ variables are not taken into account for each $j \in S_i$ if $j \neq j^{'}$, by setting their domains as $domain(Y_{ij}) = \{\emptyset\}$.

Furthermore, a $disjunctive(A_i|\forall i \in P)$ global constraint ensures that the activities, which are the elements of some set $P$, do not overlap. $span(A_i, B_i|\forall i \in E)$ ensures that the interval represented by activity $A_i$ spans the resulting union of $B_i$ interval variables, $\forall i \in E$. In other words, $start(A_i) = \min_{i \in E} start(B_i)$ and $end(A_i) = \max_{i \in E} end(B_i)$. $EndBeforeStart(A_i, B_i)$ states that activity $B_i$ cannot start before activity $A_i$ ends and updates their domains accordingly. For example, let $domain(A_i) = [20, 50]$, $domain(B_i) = [10, 40]$ and their processing times are 5 and 10, respectively. Then, this constraint updates $domain(B_i)$ as $[25, 40]$. Similarly, constraints $StartAtStart(A, B)$ and $EndAtEnd(A, B)$ enforce activities $A$ and $B$ to start and end at the same time, respectively.

### *1.3.2 Logic-Based Benders Decomposition*

Benders decomposition (BD) (Benders, 1962) is a technique to solve large-scale optimization problems that uses the strategy of "learning from mistakes". It partitions the variables of the global problem into smaller subsets: first master problem is solved for a subset of variables and their optimal values are fixed. After that, subproblem is solved with these fixed values to determine values for remaining subset of variables. If subproblem determines that fixed variables are infeasible (or not optimal), then cuts that eliminate this portion of the search space are added to the master problem.

In BD, subproblem needs to be a linear programming (LP) model. However, it may not be possible to decompose many combinatorial optimization problems such that subproblem is an LP. For this reason, Hooker and Ottoson (2003) generalize

BD as logic-based Benders decomposition (LBBD) which relies on inferring cuts via inference dual rather than LP duality. In general, a solution to inference dual provides an explanation for why a solution is optimal, or is infeasible.

Consider the following general mathematical model. Note that we derive the following description from the Hooker (2007) which is one of the pioneering papers on the methodology.

$$\text{minimize } f(x, y)$$

$$\text{subject to:}$$

$$C(x, y) \tag{1.1}$$

$$x \in D_x, y \in D_y \tag{1.2}$$

$C(x, y)$ represents the set of constraints that contain variables $x$ and $y$. $D_x$ and $D_y$ stand for the domains of respective variables. If $x$ values are fixed to $\overline{x}$, we obtain the following subproblem for the rest of the variables.

$$\text{minimize } f(\overline{x}, y)$$

$$\text{subject to:}$$

$$C(\overline{x}, y) \tag{1.3}$$

$$y \in D_y \tag{1.4}$$

The inference dual of this subproblem is the problem of inferring the best possible lower bound for $f(\overline{x}, y)$ from $C(\overline{x}, y)$.

$$\text{minimize } v$$

$$\text{subject to:}$$

$$C(\overline{x}, y) \overset{P}{\Rightarrow} f(\overline{x}, y) > v \tag{1.5}$$

$$y \in D_y, P \in \mathcal{P} \tag{1.6}$$

That is, proof $P$ deduces $f(\overline{x}, y) > v$ from $C(\overline{x}, y)$ where $\mathcal{P}$ is a family of proofs. The solution to above inference dual is a proof of the tightest bound $v^{'}$ on $f(\overline{x}, y)$. We use the same proof schema to obtain a bound $B_{\overline{x}}$. This bound needs to satisfy two properties to be a Benders cut:

$(i)$ $B_{\overline{x}}$ provides a valid lower bound for not only $\overline{x}$, but also any given $x \neq \overline{x}$, $x \in D_x$.
$(ii)$ $B_{\overline{x}} = v^{'}$.

The bounding function $B_{\overline{x}}$ is obtained by examining the type of reasoning that led to a bound for $x = \overline{x}$ and we extend this reasoning to obtain a bound for any $x \in D_x$. In other words, the solution to inference dual provides an explanation for why a solution is optimal, or is infeasible. If objective of original problem (1.1)-(1.2) only consists of $x$ variables, a valid Benders cut rules out assigning $\overline{x}$ in next iterations. This simple cut is called "nogood" and it can be strengthened if we can find a subset of $\overline{x}$ that cause infeasibility, via inference dual.

If objective of original problem also contain $y$ variables, we should take $v^{'}$ ( best possible lower bound) into account at each iteration, while deriving a Benders cut. For that case, we refer readers to (Hooker, 2000).

Chapter 2

# QUAY CRANE ASSIGNMENT PROBLEM

## *2.1 Introduction*

In this chapter, we consider quay crane assignment problem (QCAP) which deals with allocation of these scarce resource to incoming vessels. In QCAP, the handling time of a vessel depends on a number of assigned QCs. As this number increases its handling time reduces. However, the relation between the number of assigned cranes and handling time is not linear due to crane interference.

We represent this problem as a variant of moldable task scheduling problem (MTSP) in which contiguous assignments of machines are considered (Blazewicz et al., 2011). In MTSP, there are $|J|$ tasks to be non-preemptively performed by $|K|$ parallel machines. The duration of a task is a discrete function of the number of machines assigned to this task. This number is determined at the beginning of the planning period and cannot be changed during the execution. In this representation of QCAP, a task stands for the complete handling operations of a vessel while machines are QCs. QCs operate on a single rail track, hence they cannot cross each other. Thus, we need to take two critical features of the problem into account: QC interference and contiguous assignment of QCs. MTSP formulation allows us to incorporate the effect of crane interference into handling times. Moreover, we can assign a set of QCs to a vessel if and only if these QCs are located consecutively on a rail track.

Up to date, most of the studies have analyzed MTSP in its simplest form, in line with the needs of their respective domains. They assume that the resources are identical, and there is no specific property of the problem other than precedence relationships. The resulting problem can be solved by determining the number of machines for each task while ensuring the total resource (machine) consumption at any

time to be less than the total amount of available resources. On the other hand, there are additional properties of scheduling problems associated with production/service systems such as QCAP. For QCAP, these properties are as follows:

- QC assignments to a vessel should be contiguous (Lu et al., 2012). That is, indices of the machines assigned to a task must be a sequence of consecutive numbers.

- QCs may have different processing rates based on the type of equipment and/or operators' experience (Legato et al., 2013).

- QCs may have unavailable time windows because of scheduled or periodic maintenance. Another reason for QCs to have unavailable time windows is crane interference based on berthing and QC assignment plans (Meisel, 2011).

- Some task-to-machine assignments may be forbidden because of two reasons:

  - Case 1: A vessel may not be within the reach of an assigned QC, since all QCs are located on the same rail track (Unsal and Oguz, 2013).

  - Case 2: QCs may have types. There are different types of vessels. For example, it may not be possible to handle jumbo container vessels with standard size quay cranes.

- While shifting from one vessel to another, a QC requires some travel time, i.e., sequence dependent setup times (Moccia et al., 2006).

The approaches developed for solving MTSP fail to cope with the problem having properties described above. The aim of this study is to provide a flexible exact solution framework for QCAP, which can solve the problem with such properties. For this purpose, we started with investigating the literature for similar scheduling problems to see how these properties are handled. We observed that these properties are intensively studied for parallel machine scheduling and resource constrained project

scheduling problems by specifying task to machine assignments, but have not been considered for MTSP yet.

In this study, we develop a logic-based Benders decomposition (LBBD) algorithm that utilizes time-indexed formulation for MTSP. This formulation provides a flexibility to implement different features of QCAP as it specifies task to machine assignments. However, this flexibility is achieved at the expense of a significant computational difficulty: for an instance with 5 machines and $|J|$ tasks, there are $\sum_{i=1}^{5} \binom{5}{i} = 31$ machine assignment alternatives for any task, and the total number of possible assignments for the problem is $31^{|J|}$. For MTSP without specific assignments, the number of possible assignments is only $5^{|J|}$.

The rest of the chapter is organized as follows. In Section 2.2, we present a review of related problems observed in the literature and discuss their relations. In Section 2.3, after presenting a novel formulation for MTSP with contiguous assignments, we propose an LBBD algorithm, a strong Benders cut, and a compact contiguous assignments constraint. In Section 2.4, we extend the solution approach for unavailable time windows for QCs and QCs with different processing rates. Then, we present an application of QCAP for a terminal with a specific configuration that requires combination of these properties of QCs. In Section 2.5, we measure the performance of the proposed method with a set of detailed computational experiments. We discuss the relationship of the proposed approach with other scheduling problems in Section 2.6 and conclude the chapter with Section 2.7.

## *2.2 Literature Review*

Most of the scheduling literature assumes that processing time of a task is a parameter of the problem. In many practical cases, though, processing time of a task can be controlled by allocating different amount of resources. These resources can be either discretely divisible (i.e. man power, machines) or continuously divisible (i.e. energy, money). A resource can also be either renewable or non-renewable. The problems with continuously divisible and/or non-renewable resources are beyond the scope of

this study, and we refer readers to Shabtay and Steiner (2007).

The problem we consider is a variant of a more general scheduling problem with a single renewable and discretely divisible resource where processing time of a task depends on the number of resource units assigned. This number is determined at the beginning of the planning horizon and cannot be changed during the execution. Scheduling problems with this combination of properties are observed in parallel processing, project management, and machine scheduling. In the following, we first present the literature for related problems and then discuss the existing literature on QCAP, while highlighting the contribution of our study.

In parallel processing context, by definition, a parallel task can be processed by more than one processor simultaneously (Drozdowski, 2009). Du and Leung (1989) present moldable task scheduling problem (MTSP) as a generalization of multiprocessor task (MPT) scheduling problem (Drozdowski, 1996). The authors define that processing time of a moldable task depends on the number of processors allocated, hence it is a generalization of MPT scheduling problem in which the number of processors needed to perform a task is fixed a priori.

Contiguous assignments of processors are also considered in this domain because it may be desirable to maintain the physical proximity of cooperating processors (i.e. non-fragmentable multiprocessor system). By considering a makespan minimization objective and contiguous assignment of processors, Turek et al. (1992), Ludwig (1995), and Mounie et al. (1999, 2008) develop approximation algorithms with performance ratios 2.5, 2, $\sqrt{3}$, and 1.5, respectively. The algorithms proposed in these studies assume non-decreasing processing times, except the one proposed by Ludwig (1995) which is valid for any speed up function. In another research stream, Lepere et al. (2002), Jansen and Zhang (2006), and Gunther et al. (2014) present approximation algorithms for MTSP in the presence of precedence constraints.

Even though approximation algorithms fit well with the needs of multiprocessor scheduling systems where large number of tasks need to be scheduled in a very short time, they have some important drawbacks (Feitelson et al., 1997). First of all, the

approximation factor is usually of little relevance to a practical problem, since a worst case schedule is often unacceptable for a production schedule. As we discussed for the problem of allocation of QCs to vessels, scheduling problems in production/service systems require many different constraints to be taken into account. However, approximation algorithms are developed based on restricted models that deviate significantly from real life situations. An addition of a very simple property such as release times of tasks can make the results of approximation algorithms invalid.

In project management context, resource dependent activity processing times are studied as an extension of resource constrained project scheduling problem (RCPSP), and this extension is entitled as multi-mode RCPSP (MRCPSP, Weglarz et al., 2011). In MRCPSP, processing time of a task is not fixed but a function of amount of resources (renewable, non-renewable and doubly constrained) is allocated. Here, a mode reflects a feasible way to combine duration and resource requests that allows accomplishing the underlying activity. Most of the time, however, there is only one of these three types of resources in the same model. If there only exists a single non-renewable resource, such as total budget, the problem is called as time/cost trade-off problem (DTCTP, Demeulemeester and Herroelen, 2002). If there is only a single renewable resource, as in the problem we study, this special case of MRCPSP is called time/resource trade-off problem (DTRTP).

Griorgiev et al. (2007) point out that time/resource trade-off problems have received much less attention compared to time/cost trade-off problems, although they are not less appealing from a practical viewpoint. For DTRTP, De Reyck et al. (1998) propose a tabu search heuristic to cope with the problem. Demeulemeester et al. (2000) develop a branch and bound procedure based on general multi-mode RCPSP formulation of Talbot (1982). Ranjbar and Kiafar (2007) present a genetic algorithm that outperforms the result of De Reyck et al. (1998). Ranjbar et al. (2009) further extend the problem with multiple renewable resources. Note that the solution methods developed for multi-mode RCPSP can also cope with DTRTP, since it is a special case of multi-mode RCPSP.

Strong precedence relations among tasks is a key property of project scheduling problems. Accordingly, the solution techniques for these problems are mainly based on precedence relations. The existence of this type of relations among tasks is not always the case for the scheduling problems observed in production/service systems, as in the case of QCAP. Thus, these solution techniques may not be effective for the scheduling problems with independent tasks.

We observe the same combination of properties in parallel machine scheduling (PMS) literature in terms of machine scheduling with resource dependent processing times (PMS-RDPT) (or PMS with additional resources). In most of the PMS studies in the literature, the only resource is a machine. However, as Edis et al. (2013) note, there are many real-world scheduling problems that require additional resources. Based on this fact, Daniels (1999) introduces a problem of PMS where processing time of a task is a non-increasing function of the number of additional allocated resource (single and renewable). This type of problem should solve an additional resource allocation problem to determine processing times of tasks, which adds significant complexity to the problem.

As one can observe, MTSP is a special case of PMS-RDPT, in which each task has its own machine. In Figure 2.1, we present reductions among related problems from above discussed problem domains. An arrow from problem $x$ to problem $y$ implies that problem $x$ is a special case of problem $y$. Hence, problem $y$ is as hard as problem $x$ for the same objective function. Even though most of these relations are already presented in literature, to the best of our knowledge, the relation between DTRTP and MTSP has not yet been analyzed.

**Proposition 1.** *MTSP and DTRTP are equivalent problems when we assume integer processing times.*

*Proof.* See Appendix A. □

Integer processing times are common for scheduling problems in production/service systems. It implies that processing times can be accurately represented by multiple

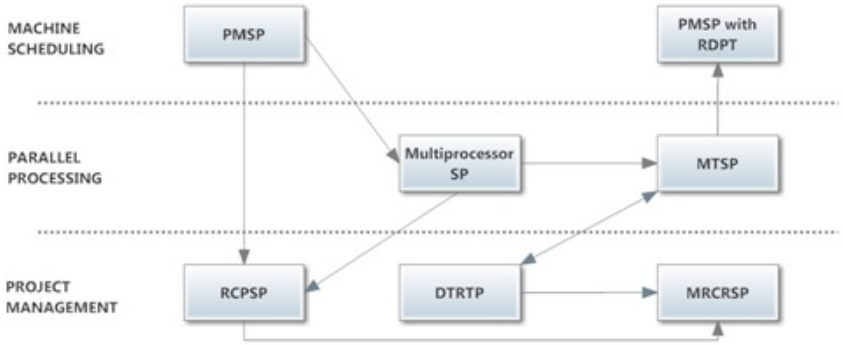



Figure 2.1: Relationship among related problems

equal length time intervals, as in man-hours (or machine-hours), contrary to multiprocessor systems where tasks can be executed even in milliseconds.

Quay crane assignment problem (QCAP) is an important application of the general scheduling problem that needs to be dealt in each maritime container terminal. The literature investigates QCAP in two different cases: time-variant and time-invariant. In both cases, processing time is a function of number of QCs. However, time-invariant QCAP assumes that the number of QCs assigned to a vessel does not change during the handling operation, while in time-variant case we relax this assumption. The problem we consider in this study is time-invariant. In theory, it is possible to increase productivity by assuming time-variant case, but as Blazewicz et al. (2011) point out, it might not be desirable from the point of view of terminal management.

The literature often investigates QCAP as a component of the berth allocation problem based on the fact that in reality QC allocations affect the handling times of vessels. Liang et al. (2009), Blazewicz et al. (2011), Yang et al. (2012), Chen et al. (2012), Turkogullari et al. (2014), and Iris et al. (2015) study integrated problem of berth allocation and quay crane assignments with an assumption of time-invariant allocation of QCs. While time-invariant QCAP is strongly related to the more general scheduling problem that we define at the beginning of this section, only two of these studies utilize previous work from the related literature: Blazewicz et al. (2011) approach this problem as a moldable task scheduling problem where handling of a

vessel is a task and QC is a processor, respectively. Turkogullari et al. (2014) use the mathematical formulation of multi-mode RCPSP provided by Talbot (1982) as the basis of their model.

Furthermore, all of the mentioned studies assume that QCs are identical and available during the whole planning period. QCs are identified as bottleneck equipment in a terminal, thus, allocation of these scarce resources is critical for the performance of terminals. If we do not take the various properties of QCAP into account, we may estimate the handling times of vessels inaccurately, resulting in a loss of productivity of overall terminal operations. This study differentiates from the literature by investigating QCAP in greater detail. We develop an exact solution method based on a mixed integer programming formulation of MTSP that specifies task to machine assignments. By this way, we can handle the properties that are commonly observed in container terminals, e.g. unavailability of QCs and different processing rates of QCs. Such a modeling flexibility is important because not every container terminal has the same configuration.

## *2.3 Mixed-Integer Programming Formulation for MTSP with Contiguous Assignments*

In this section, we present a time-indexed integer programming formulation for MTSP with contiguous assignments. A key feature of this formulation is specifying task to machine assignments. The sets and parameters to be used are presented as follows. Note that we use the terms machine and resource interchangeably, and both refers to a QC in a container terminal setting. Similarly, a task stands for the complete handling operations of a vessel.

- $J$ set of tasks; indexed by $i = \{1, \ldots, |J|\}$,
- $K$ set of machines; indexed by $l = \{1, \ldots, |K|\}$,
- $W$ set of number of total machines that can be assigned to a task; indexed by

$l = \{1, \ldots, \delta_i\}$, where $\delta_i$ stands for the maximum number of machines that can be assigned to task $i$,

- $T$ set of times; indexed by $t = \{1, \ldots, |T|\}$,
- $S$ set of all non-contiguous machine assignments,
- $w_i$ weight (importance) of task $i$,
- $p_{il}$ processing time of task $i$ if it is processed by $l$ machines.

Parameter $p_{i1}$ denotes the processing time of task $i$ when processed by a single machine. Speed up function $f_l$ then defines the relation between the processing time and the total number of assigned machines such that $p_{il} = \lceil p_{i1}/f_l \rceil$, and $lp_{il}$ stands for the total amount of work to be done by all assigned machines.

Let $Z_{ijlt}$ be a binary variable that takes the value of 1 if task $i$ is assigned to machine $j$ and a total number of $l$ machines, and starts at time $t$, 0 otherwise. As we mentioned, this variable keeps the information of specific task to machine assignments as well as the number of machines assigned to each task at the same time. The formulation $[MTSP - CA]$ is presented below.

$$\text{minimize} \sum_i \sum_j \sum_l \sum_t (t + p_{il}) w_i Z_{ijlt}/l$$

subject to:

$$\sum_j \sum_l \sum_t Z_{ijlt}/l = 1 \qquad \forall i \quad (2.1)$$

$$\sum_i \sum_l \sum_{h=max(1,t-p_{il}+1)}^{t} Z_{ijlh} \leq 1 \qquad \forall j, \forall t \quad (2.2)$$

$$\sum_{j \in u} \sum_t Z_{ij|u|t} \leq |u| - 1 \qquad \forall i, u \in S \quad (2.3)$$

$$Z_{ijlt} + Z_{iklh} \leq 1 \qquad \forall i, \forall j, \forall k, \forall t, \forall h, \forall l : t > h \quad (2.4)$$

$$Z_{ijlt} + Z_{ikl't} \leq 1 \qquad \forall i, \forall j, \forall k, \forall t, \forall l, \forall l^{'} : l > l^{'} \quad (2.5)$$

$$Z_{ijlt} \in \{0, 1\} \qquad \forall i, \forall j, \forall l, \forall t \quad (2.6)$$

The objective function is to minimize total weighted completion time of tasks. For each task, constraint (2.1) assigns a start time, a total number of machines to be used ($l$), and also specific machines to be used. Capacity constraint (2.2) ensures non-overlapping of tasks that are assigned to the same machine. By constraint (2.3), we ensure that a set of machines that are assigned to a task to be contiguous; that is, indices of these machines are a sequence of consecutive numbers. This enumeration based constraint forbids all possible non-contiguous machine assignments for each task: if there are $|u|$ machines in element $u$ of $S_3$, then we restrict the cases where a task is assigned to $|u|$ total machines, and these machines are $j \in u$.

Constraints (2.4) and (2.5) address two unique properties associated with the given MTSP formulation. These computationally expensive constraints describe the relation of machine index $j$ with indices of total number of machines ($l$) and time ($t$), respectively.

**Property 1** A task that is processed by more than one machine must start on all of these assigned machines at the same time $t$. Let task $i$ is assigned to machines 1 and 2. Without constraint (2.4), it is possible that $Z_{i12t}$ and $Z_{i22t'}$ both take the value of 1, where $t \neq t^{'}$.

**Property 2** A task that is processed by different machines must be assigned to the same number of total machines, denoted by $l$, for each of these machines (Constraint (2.5)). Assume that task $i$ is assigned to machines 1, 2, and 3. Without this constraint, $Z_{i12t}$, $Z_{i24t}$ and $Z_{i34t}$ can all take the value of 1 since these assignments satisfy constraint (2.1).

In addition to expensive constraints (2.3), (2.4), and (2.5), capacity constraint (2.2) also makes the formulation harder to solve. It is known that the capacity constraint of time-indexed scheduling problem formulations is a major cause of a computational difficulty (Van der Akker, 2000). Existence of machine index "$j$" in the formulation further complicates the capacity constraint: it has very large number of constraints and non-zero coefficients. As a result, the formulation $[MTSP - CA]$ fails to solve even small-sized instances because of excessive memory requirements.

## *2.4 Logic-Based Benders Decomposition for MTSP with Contiguous Assignments*

Due to the limitations of the model proposed, we develop a logic-based Benders decomposition (LBBD) procedure based on formulation $[MTSP - CA]$. LBBD is first implemented for scheduling problems by Hooker (2007). In this pioneering paper, the author proposes a novel decomposition structure for a variant of a parallel machine scheduling problem which is then adapted by many other studies. It simply decomposes the original problem into a master problem in which task to machine assignments are made but scheduling decisions are disregarded, and a subproblem which solves an easy single machine scheduling problem. In Appendix E, we discussed the implementation of this decomposition strategy for our problem and showed that it is not suitable for our problem.

Therefore, we develop another decomposition strategy for solving our problem. We first extend the formulation with a new set of variables to get rid of two expensive constraints associated with unique properties. Then, we decompose the resulting formulation such that the determination of specific task to machine assignments and contiguous assignments constraint are left to the subproblem. At the last part of the section, we propose a compact contiguous assignments constraint that allows restricting all non-contiguous assignments with a smaller number of constraints compared to enumerating all of these assignments.

Let $Q_{ilt}$ be a binary decision which is simply a restricted version of $Z_{ijlt}$ without

specific assignment information. It takes the value of 1 if task $i$ is processed by $l$ total machines simultaneously and starts at time $t$, 0 otherwise. We introduce $Q_{ilt}$ variables and add linking constraint (2.7) to formulation $[MTSP - CA]$:

$$\sum_j Z_{ijlt} = lQ_{ilt} \qquad \forall i, \forall l, \forall t \quad (2.7)$$

The resulting hybrid formulation makes it possible to remove constraints which are associated with two unique properties of formulation with $Z_{ijlt}$ variables.

**Proposition 2.** *If we introduce $Q_{ilt}$ variables and replace constraint (2.4) with constraint (2.7), then the model ensures that start time of a task on different machines cannot be assigned to two different time points.*

*Proof.* Assume that constraint (2.4) is replaced with constraints (2.7), and start time of a task on $l$ different machines ($l > 1$ and $l$ is bounded) is assigned to two different time points $t_1$ and $t_2$. That is, $Z_{i,j_1,l,t_1} = 1$ and $Z_{i,j_2,l,t_2} = 1$.

Constraint (2.7) states that for each $i$, $l$, and $t$, $\sum_j Z_{ijlt} = lQ_{ilt}$. Therefore, for $i$,$l$, and $t_1$, left hand side of (2.7) is at least 1 and at most $l - 1$ because $Z_{i,j_2,l,t_2} = 1$ and $t_1 \neq t_2$. Accordingly, $Q_{i,l,t_1}$ takes the value of at least $1/l$ and at most $(l - 1)/l$. However, domain of $Q_{ilt}$ is {0,1}: $Q_{ilt}$ variables can only take a value of 0 or 1. Thus, start time of a task on different machines must be assigned to the same time point. This contradicts our first assumption. □

We make the following claims for the second property which addresses the relation between $j$ and $l$ indices of $Z_{ijlt}$ variables.

**Claim 1** *If task $i$ is assigned to only one machine, then this task cannot be assigned to different total number of machines.*

*Proof.* By constraint (2.1), for each task $i$, we have $\sum_j \sum_l \sum_t Z_{ijlt}/l = 1$. Let this task is assigned to machine $j_1$, such that $Z_{i,j_1,1,t} = 1$. We cannot have another $Z_{ijlt}$ variable for task $i$ with a positive value. □.

**Claim 2** *If $|K|$=2, then a task cannot be assigned to different total number of machines.*

*Proof.* If we have two machines in total, then a task needs to be assigned to both one total machine ($l = 1$) and two total machines ($l = 2$) for this situation to occur. By Claim 1, it is not possible. □

**Proposition 3.** *If we introduce $Q_{ilt}$ variables and replace constraint (2.5) with constraint (2.7), then the model ensures that a task which is assigned to different machines cannot be assigned to different total number of machines.*

*Proof.* Assume there are $|K|$ total machines ($|K|$>2 and is bounded), and task $i$ is assigned to both $j_1$ and $j_2$ which are different machines as well as two different total number of machines $l_1$ and $l_2$. That is, $Z_{i,j_1,l_1,t}$=1 and $Z_{i,j_2,l_2,t}$=1. Constraint (2.7) states that for each $i$, $l$, and $t$, $\sum_j Z_{ijlt} = lQ_{ilt}$. For $i$, $l_1$ ,and $t$, left hand side of (2.7) takes a value of at least 1 (because $Z_{i,j_1,l_1,t}$=1) and at most $|K| - 1$ since $Z_{i,j_2,l_2,t}$=1, $l_1 \neq l_2$. Therefore, $Q_{i,l_1,t}$ can take values between $1/l_1$ and $(|K| - 1)/l_1$. Similarly, for $i$, $l_2$, and $t$, left hand side of (2.7) takes a value at least 1 and at most $|K|$-1 since $Z_{i,j_1,l_1,t}$=1, $l_1 \neq l_2$. So, $Q_{i,l_2,t}$ can only take values between $1/l_2$ and $(|K| - 1)/l_2$ . Observe that these values are positive as $1/l_1 > 0$ and $1/l_2 > 0$, and they should take the value of 1 because $Q_{ilt} \in \{0, 1\}$. There are four possible cases in which they both take the value of 1:

(i) $l_1 = 1$, $l_2 = 1$ Contradicts ($l_1 \neq l_2$)

(ii) $l_1 = 1$, $l_2 = |K| - 1$ Contradicts Claim 1

(iii) $l_1 = |K| - 1$, $l_2 = 1$ Contradicts Claim 1

(iv) $l_1 = |K| - 1$, $l_2 = |K| - 1$ Contradicts ($l_1 \neq l_2$)

Hence, a task cannot be assigned to different total number of machines, and this contradicts our assumptions. □

The resulting hybrid formulation contain both $Q_{ilt}$ and $Z_{ijlt}$ variables. In our LBBD implementation, we utilize respective advantages of these two sets of variables to decompose $[MTSP - CA]$ into two easier problems, even though such an integration increases total number of variables and constraints within the formulation. The master problem is a relaxation of the hybrid formulation in which computationally expensive capacity constraint (2.2) is replaced by the one that utilizes $Q_{ilt}$ variables to ensure that the total machine usage is less than or equal to available number of machines at any time. That is, we delay determination of specific task to machine assignments until the subproblem in which non-overlapping of tasks on each machine is ensured.

We add contiguous assignment constraint only to the subproblem because it is redundant in the master problem: the master problem can simply find a solution in which each task is assigned to machines with consecutive indices starting from $j = 1$ because we do not consider non-overlapping on tasks for each machine. In addition, the structure of the proposed LBBD allows the subproblem to further utilize this constraint as the subproblem only deals with a limited number of variables.

**[MP]: Master Problem of LBBD**

$$\text{minimize } \sum_i \sum_j \sum_l \sum_t (t + p_{il}) w_i Z_{ijlt}/l$$

subject to:

$$\sum_j \sum_l \sum_t Z_{ijlt}/l = 1 \qquad \forall i \quad (2.8)$$

$$\sum_j Z_{ijlt} = l Q_{ilt} \qquad \forall i, \forall l, \forall t \quad (2.9)$$

$$\sum_i \sum_l \sum_{s=max(1,t-p_{il}+1)}^{t} l Q_{ils} \leq |K| \qquad \forall t \quad (2.10)$$

$$\sum_{(i,l,t)\in CutSet_h} Q_{ilt} \leq |CutSet_h| - 1 \qquad \forall h \quad (2.11)$$

$$Z_{ijlt} \in \{0,1\}, Q_{ilt} \in \{0,1\} \qquad \forall i, \forall j, \forall l, \forall t \quad (2.12)$$

**[SP]: Subproblem of LBBD**

$$\text{minimize } \sum_i \sum_j \sum_l \sum_t (t + p_{il}) w_i Z_{ijlt}/l$$

subject to:

$$\sum_j Z_{ijlt} = l \qquad \forall (i,l,t) \in SOLUTION_{[MP]} \quad (2.13)$$

$$\sum_{\substack{(i,l,s)\in SOLUTION_{[MP]}: \\ s\in\{max(1,t-p_{il}+1),\ldots,t\}}} Z_{ijls} \leq 1 \qquad \forall j, \forall t \quad (2.14)$$

$$\sum_{j\in u} Z_{ij|u|t} \leq |u| - 1 \qquad \forall (i,l,t) \in SOLUTION_{[MP]}, u \in S : |u| = l \quad (2.15)$$

$$Z_{ijlt} \in \{0,1\} \qquad \forall (i,l,t) \in SOLUTION_{[MP]} \quad (2.16)$$

We solve the master problem $[MP]$ to optimality and then pass the solution in terms of optimal start times and number of machines assigned to each task to the subproblem $[SP]$, but not specific task to machine assignments, i.e., $SOLUTION_{[MP]} = \{(i,l,t) :\ Q^*_{ilt} = 1,\ \forall i, \forall l, \forall t\}$. Given $SOLUTION_{[MP]}$, the subproblem checks capacity constraint for each machine and enforce contiguous assignments over $Z_{ijlt}$ variables. In other words, [SP] simply removes the overlapping of tasks on machines by changing task to machine assignments while ensuring contiguous assignment of machines for each task.

Despite containing a capacity constraint which we previously identified as a source of computational difficulty, $[SP]$ is significantly easier to solve since we work with a

limited set of variables as we have already fixed number of assigned machines and start time for each task. Note that there are $O(|J||K||L||T|)$ total $Z_{ijlt}$ variables and we only work with $O(|J||K|)$ of them in the subproblem.

If $[SP]$ turns out to be infeasible, we add a cut to $[MP]$ that removes infeasible solution(s) from its solution space. This procedure is repeated until $[SP]$ identifies a feasible allocation of tasks to machines. We present the pseudocode of the proposed LBBD in Algorithm 2.1.

The simplest cut for LBBD is called nogood, a cut that removes current solution from the search space of master problem. Nogood cuts are often weak, thus it is better to develop a cut that removes a larger portion of solution space (Hooker, 2007). We propose a stronger Benders cut in the following.

Algorithm 2.1: Pseudocode of proposed LBBD

$initialize : \gamma = 0, LB = 0, iteration = 0,$
$SOLUTION_{[MP]} = \varnothing, CutSet = \varnothing$
**while** $\gamma$=0 **do**
  solve [MP]
    $SOLUTION_{[MP]} = \{(i,l,t) : Q^*_{ilt} = 1\}$
  solve [SP] given $SOLUTION_{[MP]}$
  **if** [SP] is feasible **then**
    $\gamma = 1$
  **else**
    $LB = z^*_{SOLUTION_{[MP]}}$
    $BendersCut(SOLUTION_{[MP]})$
      $CutSet \leftarrow CutSet \cup CutSet_{iteration}$
      add cut $z \leq LB$ to [MP]
  **end if**
  $iteration \leftarrow iteration + 1$
**end while**

### *2.4.1 Proposed Benders Cut*

**Definition** *A cut is a valid Benders cut if and only if it does not remove any optimal solution to global problem and guarantees the finite convergence of algorithm (Hooker, 2007).*

According to this definition, nogood cut is a valid Benders cut since ($i$) it removes a complete solution from master problem which is proved to be infeasible for global problem and ($ii$) master problem has a finite domain. Assume that we are given an optimal solution to the master problem and subproblem turns out to be infeasible. Let there exists a task $i$ (or set of tasks) which has a value of one in that solution to the master problem and the corresponding subproblem becomes feasible if we remove this set of tasks. Still, we cannot claim that would be a valid Benders cut since it may cut an optimal solution: there might exist an optimal solution in which related variable(s) takes a value of 1 with different assignments of other variables. That is, we need to consider the whole schedule generated by the master problem to be able to generate a valid Benders cuts.

This observation led us to develop a strong Benders cut, by considering the fact that $[SP]$ can be solved multiple times within a short time. Given $SOLUTION_{[MP]}$, we repeatedly solve $[SP]$ by eliminating a task with the largest completion time at once, until infeasibility resolves. The remaining solution together with the last eliminated task then constitutes a subset of $SOLUTION_{[MP]}$ which is still infeasible, independent of the eliminated part of the solution. This cut generation procedure proves via optimization that if $SOLUTION_{[MP]}$ contains this subset, then corresponding $[SP]$ is infeasible. Let this subset be an element of set $CutSet$. Then, we add constraint $\sum_{(i,l,t)\in CutSet_h} Q_{ilt} \leq |CutSet_h| - 1$ to $[MP]$ for each iteration $h$ as we need to change at least one element of the subset to be able to find a feasible solution to $[SP]$. Algorithm 2.2 presents the pseudocode for the proposed cut generation procedure.

Algorithm 2.2: Procedure for cut generation ($BendersCut(SOLUTION_{[MP]})$)

$initialize : \sigma = 0, S^{'} = \varnothing, CutSet_{iteration} = \varnothing$
sort $SOLUTION_{[MP]}$ in non-increasing order of $(t^* + p_{il})$
**while** $\sigma$=0 **do**
  $S^{'} \leftarrow SOLUTION_{[MP]} \setminus S^{'}$
  solve [SP] for $SOLUTION_{[MP]} \setminus S^{'}$
  **if** [SP] is feasible **then**
    add symmetric solutions to $SOLUTION_{[MP]}$
    $CutSet_{iteration} \leftarrow SOLUTION_{[MP]}$
    $\sigma = 1$
  **else**
    $SOLUTION_{[MP]} \leftarrow SOLUTION_{[MP]} \setminus SOLUTION_{[MP]}(1)$
  **end if**
**end while**

**Proposition 4.** *The proposed cut is as strong as nogood cut.*

If the procedure terminates after removing a single task from $SOLUTION_{[MP]}$ (i.e. the task with the largest completion time), then the proposed cut (without considering symmetric task pairs) is identical to nogood cut. In other words, $[SP]$ becomes feasible by removing the task with the largest completion time, hence the proposed cut includes all elements of $SOLUTION_{[MP]}$ by definition. If the proposed cut requires at least one task with the largest completion time to be removed from $SOLUTION_{[MP]}$, then it removes a strictly larger portion of the search space of the global problem which includes the part removed by nogood cut.

**Proposition 5.** *The proposed cut is a valid Benders cut.*

The procedure we previously explained identifies a subset of $SOLUTION_{[MP]}$ that causes infeasibility, by testing iteratively via optimization model $[SP]$. Therefore, it only removes a portion from the master problem which is infeasible for the global problem. The proposed cut is as strong as nogood cut, and finite convergence with nogood cut is guaranteed as long as variables have finite domain (Chu and Xia, 2004).

**Proposition 6.** *Without contiguous assignment constraint, the proposed LBBD is guaranteed to be solved to the optimality in the first iteration, if there exists a feasible solution to the master problem.*

*Proof.* Let us first solve the master problem to optimality. Given $(i, l, t) \in SOLUTION_{[MP]}$, task $i$ can be decomposed into $l$ identical tasks with one unit resource requirement. Start and end times of each of these identical tasks fixed to $t$ and $t + p_{il}$, respectively. The subproblem is now equivalent to tactical fixed job scheduling problem (Kolen et al., 2007), and in $[MP]$, the maximum number of tasks that overlaps at any time ("*job overlap*") is bounded by total number of machines $|K|$ by constraint (2.10). Dilworth's chain decomposition theorem suggests that there exists a feasible schedule, if we have "*job overlap*" machines which is less than or equal to $|K|$ (Dilworth, 1959). □

Recall that we represent QCAP as a MTSP with contiguous assignments. Proposition 6 is no more valid when we consider contiguous assignments of machines. In Appendix B, we present a counter example in which subproblem turns out to be infeasible, given $SOLUTION_{[MP]}$.

The proposed LBBD algorithm differentiates from the literature in terms of the decomposition strategy. Common approach in the literature is to separate variables of global problem associated with different decisions into easier problems. However, in our case, we further extend variable space of the original problem (i.e., $[MTSP-CA]$) by introducing new set of variables ($Q_{ilt}$) and integrate them with $Z_{ijlt}$ variables in the master problem. This integration utilizes $Z_{ijlt}$ and $Q_{ilt}$ variables such that the former allows modeling various problem constraints, as we described for QCAP while the latter ensures Properties 1 and 2 without expensive constraints (2.4) and (2.5) and also allows us to provide a relaxation for capacity constraint (2.2). The solution approach we proposed in this section provides a flexible framework such that various features of QCAP such as machine unavailability, contiguous assignments, and uniform machines can be implemented straightforwardly.

### 2.4.2 Contiguous Assignments Constraint

In a schedule constructed with contiguous assignment, indices of the machines assigned to a task must be a sequence of consecutive numbers (Bladek et al., 2014). Common applications of this case are assignment of check-in desks to flights in airports and assignments of quay cranes to container vessels in maritime terminals where non-contiguous assignments often generate infeasible solutions since quay cranes are operating on the same rail track (Bierwirth and Meisel, 2010). By utilizing the ability of the proposed approach to capture specific task to machine assignments, we can develop a more compact contiguous assignment constraint than enumerating all non-contiguous assignments. We define new sets and parameters as follows.

$S_1$ set of all combinations of possible machine assignments for each task,

$S_2$ set of contiguous machine assignments for each task,

$S$ set of all non-contiguous machine assignments, i.e., $S = S_1/S_2$.

As we mentioned within the MIP formulation of the problem, the straightforward approach would be to enumerate all non-contiguous assignments and then to forbid the existence of any of them in a schedule. For $|K|$ total number of machines, number of elements are $\sum_k \binom{|K|}{k}$ and $|K|(\frac{|K|-1}{2}) + |K|$ (i.e., $|K|(\frac{|K|+1}{2})$) for $S_1$ and $S_2$, respectively. For each element in set $S$, constraint (2.17) restricts a non-contiguous assignment by taking the number of total machines in such an assignment into account.

$$\sum_i \sum_{j \in u} \sum_t Z_{i,j,|u|,t} \leq |u| - 1 \qquad \forall i, \forall u \in S \quad (2.17)$$

By this constraint, we restrict the cases where a task is assigned to $|u|$ total machines if each of these machines is an element of $u$. For example assume that total of five machines are located in sequence of 1, 2, 3, 4, 5. Then $u = \{1, 2, 5\} \in S$ with $|u| = 3$. Constraint (2.17) states that if any task $i$ assigned to total of three machines, then

these machines cannot be 1, 2, and 5.

Alternatively, we take the advantage of the observation that all non-contiguous machine assignments can be restricted with a significantly lesser number of constraints compared to enumerating all non-contiguous assignments. In this approach, we need to check for non-contiguous assignments by considering machines two by two implicitly, without checking all 2-combinations of the machine set. Let us investigate this approach for $|K|$=4, in details. For machines (1,4), we need to check for total number of machines 2 and 3. That is, there are two constraints for each task, and more specifically, if any task $i$ is assigned to two total machines, these machines cannot be 1 and 4. Similarly, if any task $i$ is assigned to three total machines, two of these machines cannot be machines 1 and 4 at the same time because whatever the third machine assigned to task $i$, this will be a non-contiguous assignment. For (1,3) and (2,4), we need to check for total number of machines 2. So, there is one constraint for each of them. For (1,2), (2,3) and (3,4), there is no such case. As a result, for $|K|$=4, we need to restrict four assignments for each task by the proposed approach. By the same way, for $|K|$=5, we obtain this number as 10. We generalize these restrictions for MTSP with specific assignments formulation with the following constraint.

$$Z_{ijlt} + Z_{iklt} \leq 1 \qquad \forall i,\ \forall j,k,\ \forall t, \forall l \in \{2,\ldots,(k-j)\} :\ k > j+1 \quad (2.18)$$

Even though the total number of non-contiguous machine assignments grows exponentially with the number of machines $|K|$, by this approach, we can restrict all non-contiguous assignment by using a relatively small number of constraints.

**Proposition 7.** *In an instance with $|K|$ machines, there are $2^{|K|}$-1- $\frac{|K|(|K|+1)}{2}$ non-contiguous machine assignments for each task. At the same time, we can restrict all possible non-contiguous assignments by considering $\frac{\left(|K|^3-3|K|^2+2|K|\right)}{6}$ assignments.*

*Proof.* We previously explained the derivation of the number of non-contiguous machine assignments. By observation, we can calculate that the number of assignments required to be restricted by the proposed approach is $\sum_{k=1}^{|K|-2} k + \sum_{k=1}^{|K|-3} k + \cdots +$

$\sum_{k=1}^{2} k + \sum_{k=1}^{1} k$ for $|K|$ >5 total machines. That is, $\frac{(K-2)(K-1)}{2} + \frac{(K-3)(K-2)}{2} + \cdots + \frac{(2)(3)}{2} + \frac{(1)(2)}{2}$ which equals $\sum_{i=2}^{|K|-1} \frac{(|K|-i)(|K|-i+1)}{2}$. After some simplifications we obtain $\frac{\left(|K|^3-3|K|^2+2|K|\right)}{6}$. □

Table 2.1 shows that constraint (2.18) has a significant advantage over the enumeration based constraint especially when $|K|$ is getting larger, since the number of restricted assignments by the proposed constraint has polynomial growth.

Table 2.1: Comparison of two different contiguous assignments constraints

| $\|K\|$ | Enumeration | Proposed |
|---|---|---|
| 3 | 1 | 1 |
| 4 | 5 | 4 |
| 5 | 16 | 10 |
| 6 | 42 | 20 |
| 7 | 99 | 35 |
| 8 | 219 | 56 |
| 9 | 466 | 84 |
| 10 | 968 | 120 |
| 15 | 32647 | 455 |
| 20 | 1048365 | 1140 |

*Example for the proposed LBBD*

In the following, we present the execution of LBBD algorithm with proposed Benders cut iteration-by-iteration. It is the instance 9th of 20 tasks 6 machines instances with $p_{il} \in [20, 60]$ and speed-up function $f_l^1$.

Given an instance, tasks $i$ and $i'$ are symmetric if $p_{il} = p'_{il}$ and $w_i = w'_i$. Symmetric tasks can be swapped in $SOLUTION_{[MP]}$ without affecting the rest of the schedule. In that case, additional cuts that contain these symmetric solutions must be added

to the master problem as well. This instance, however, does not contain symmetric tasks and takes 94 iterations to solve (∼55 minutes) with nogoods cuts, while it can be solved with just 8 iterations (∼2 minutes) with the proposed Benders cut.

As a side note, if we add this contiguous assignments constraint directly to the hybrid formulation rather than to the subproblem of LBBD, it cannot be solved within even 5 hours of time-limit. This result shows the significance of the decomposition. In Table 2.2, we present all iterations of LBBD with the proposed cut.

As can be observed, the proposed Benders cut identifies that almost half of the tasks in the optimal solutions of master problem are independent of the subset of the tasks that causes infeasibility. That is, removing these tasks cannot eliminate the infeasibility, so we can derive a valid Benders cut without taking them into account. Note that the proposed cut removes larger portion of feasible space of master problem compared to nogood cuts. Such a strengthening helps the algorithm to find the optimal solution within significantly lesser number of iterations.

## *2.5 Extensions*

### *2.5.1 Machine Availability*

In this extension of the problem, we take into account that some machines may have limited availability over the planning period (Meisel, 2011, Schmidt, 2000). We define the following parameters associated with unavailability and add constraints (2.19) and (2.20) to the master problem.

- $f_{jt}$ parameter that indicates the unavailability of machine $j$ at time period $t$. It has a value of 1 if machine $j$ is available at time $t$, 0 otherwise.

- $v_t$ parameter that represents the number of available machines at time $t$, i.e., $v_t = \sum_j f_{jt}$.

Table 2.2: LBBD iterations for an instance with $|J|$=20 and $|K|$=6

| iteration | |
|---|---|
| 1 | Master Problem: $z^*$=6870 Subproblem: infeasible<br>Proposed Cut: $CutSet_1$={(6,2,44),(20,2,40), (10,2,37), (9,2,19), (13,1,13), (18,1,15), (14,1,13), (4,1,15), (16,2,1), (1,2,1), (17,2,1)} and $\|CutSet_1\|$=11 |
| 2 | Master Problem: $z^*$=6870 Subproblem: infeasible<br>Proposed Cut: $CutSet_2$={(6,2,40), (20,2,44), (10,2,37), (9,2,19), (13,1,13), (18,1,15), (14,1,13), (4,1,15), (16,2,1), (1,2,1), (17,2,1)} and $\|CutSet_2\|$=11 |
| 3 | Master Problem: $z^*$=6871 Subproblem: infeasible<br>Proposed Cut: $CutSet_3$={(6,2,44), (10,2,37), (20,2,37), (9,2,19), (13,1,13), (18,1,15), (14,1,13), (4,1,15), (16,2,1), (1,2,1), (17,2,1)} and $\|CutSet_3\|$=11 |
| 4 | Master Problem: $z^*$=6871 Subproblem: infeasible<br>Proposed Cut: $CutSet_4$={(6,2,44), (10,2,37), (9,2,19), (13,1,13), (18,1,15), (14,1,13), (4,1,15), (16,2,1), (1,2,1), (17,2,1)} and $\|CutSet_4\|$=10 |
| 5 | Master Problem: $z^*$=6872 Subproblem: infeasible<br>Proposed Cut: $CutSet_5$={(6,2,44), (20,2,40), (10,2,37), (9,2,19), (13,1,13), (18,1,16), (14,1,13), (4,1,15), (16,2,1), (1,2,1), (17,2,1)} and $\|CutSet_5\|$=11 |
| 6 | Master Problem: $z^*$= 6873 Subproblem: infeasible<br>Proposed Cut: $CutSet_6$={(6,2,37), (20,2,44), (10,2,40), (9,2,19), (13,1,13), (18,1,15), (14,1,13), (4,1,15), (16,2,1), (1,2,1), (17,2,1)} and $\|CutSet_6\|$=10 |
| 7 | Master Problem: $z^*$=6873 Subproblem: infeasible<br>Proposed Cut: $CutSet_7$={(6,2,37), (15,2,44), (10,2,40), (9,2,19), (13,1,13), (18,1,15), (14,1,13), (4,1,15), (16,2,1), (1,2,1), (17,2,1)} and $\|CutSet_7\|$=11 |
| 8 | Master Problem: $z^*$=6877 subproblem = feasible & optimal |

$$\sum_i \sum_l \sum_{s=max(1,t-p_{il}+1)}^{t} Z_{ijls} = 0 \qquad \forall j, \forall t : f_{jt} = 0 \quad (2.19)$$

$$\sum_i \sum_l \sum_{s=max(1,t-p_{il}+1)}^{t} lQ_{ils} \leq v_t \qquad \forall t \quad (2.20)$$

Constraint (2.19) restricts the cases in which machine $j$ processes task $i$ in one of its unavailable periods. We are able to model this extension without using capacity constraint with $Z_{ijlt}$ variables that checks capacity for each machine, even though the number of available machines changes over time: since constraint (2.19) does not allow processing of task while machines are unavailable, we can keep track of total capacity usage at any time with a capacity constraint that has a time dependent right-hand-side ($v_t$). We add constraint (2.19) to the subproblem as well.

Assuming contiguous assignments of machines, unavailability of machines during the planning period can further limit the distribution of QCs to vessels. For example, consider a terminal with 5 QCs, and QC 2 is unavailable within time interval $[t_1, t_2]$. If we assign 2 QCs to each of two vessels during that interval, then we can satisfy constraints (2.19) and (2.20). However, the subproblem turns out to be infeasible because of the contiguous assignment constraint: given the solution to the master problem, even though there are 4 total QCs available during $[t_1, t_2]$, there is no feasible distribution because of the position of unavailable machine. We can propose the following valid inequality to prevent such a situation in the master problem:

$$\sum_i \sum_{h=max(1,t-p_{il}+1)}^{t} Q_{i2h} \leq 1 \qquad \forall t \in [t_1, t_2] \quad (2.21)$$

This valid inequality depends on the terminal configuration. Hence, we need to investigate the configuration and manually add this constraint.

### 2.5.2 Uniform Machines

Up to this point, we assume that all machines are identical; that is, they have the same processing rate. Scheduling with identical machines has a large applicability (Belouadah and Potts, 1991) since machines often tend to have similar processing speeds in many environments. At the same time, there are lots of cases in which there are machines with different processing capabilities in parallel (Li and Yang, 2009). There are two types of non-identical machine scheduling problems in the literature:

- Unrelated machines: processing time of a task depends on the machine (Vredeveld and Hurkens, 2002).

- Uniform machines: each machine has processing rate (speed) and processing time of a task is proportional to processing rate of machine that it is assigned (Azizoglu and Kirca, 1999).

In this extension, we consider the uniform machine environment (Azizoglu and Kirca, 1999) and study the following variant of the problem: Each machine has a different processing rate and a task can be assigned to more than one machine. Processing time of a task then depends on the total processing rate of assigned machines, rather than the total number of them.

An example can be given as quay crane assignment and/or quay crane scheduling problem. Legato et al. (2013) study quay crane scheduling problem with the assumption of quay cranes having different processing rates. Their motivation comes from the fact that processing time of a task often depends on the particular crane, as it is affected by the experience of crane drivers. It is also possible that a terminal can have different types of quay cranes on the same rail track, such as cranes with single lift and twin lift. For this reason, we can claim that the handling time of a vessel also depends on the processing rates of assigned quay cranes.

For the mathematical formulation, we modify the definition of $l$ index of $Z_{ijlt}$ and $Q_{ilt}$ variables: the $l$ index still determines processing time of a task, however, this

time it stands for a total processing rate of assigned machines. Let $s_j$ to be the processing rate of machine $j$. The corresponding mathematical formulation for the master problem is given as follows.

$$\text{minimize} \sum_i \sum_j \sum_l \sum_t (t + p_{il}) w_i s_j Z_{ijlt} / l$$

subject to:

$$\sum_j \sum_l \sum_t s_j Z_{ijlt} / l = 1 \qquad \forall i \quad (2.22)$$

$$\sum_j s_j . Z_{ijlt} = l Q_{ilt} \qquad \forall i, \forall l, \forall t \quad (2.23)$$

$$\sum_i \sum_l \sum_{h=max(1,t-p_{il}+1)}^{t} l Q_{ilh} \leq \sum_j s_j \qquad \forall t \quad (2.24)$$

$$Z_{ijlt} \in \{0,1\}, Q_{ilt} \in \{0,1\} \qquad \forall i, \forall j, \forall l, \forall t \quad (2.25)$$

We first modify the objective function, assignment, and linking constraints with respect to new definition of variables. By this way, we can relate processing time of each task with the total processing rate of assigned machines. Consider an instance with one task and 2 machines. Machine 1 is two times faster than machine 2, therefore we can claim that $s_j$ equals to 2 and 1, respectively, without loss of generality. Now consider the assignment constraint with this instance: $Z_{1131}/3 + Z_{1231}/3 = 2/3$. The reason is obvious: in this extension, $l$ index stands for total rate (speed) of assigned machines, rather than the total number of them and we need to incorporate processing speed of each machine with this constraint. To accomplish that, we multiply each element of left-hand-side of constraint with $s_j$: $s_1 Z_{1131}/3 + s_2 Z_{1231}/3$, and it follows that $(2.1)/3 + (1.1)/3 = 1$. The same modification is also needed for objective function and linking constraint.

Furthermore, we need to revisit capacity constraint. Constraint (2.24) simply ensures that total processing rate usage must be less than the total processing rate of machines ($\sum_j s_j$), rather than the total number of machines ($|K|$).

For the subproblem, we need to make the same modification on assignment constraint to interpret machines with different processing rates. In addition to that, contiguous assignment constraint (2.18) needs to be modified because of the existence of machines with $s_j > 1$. We define additional processing rate of a machine $j$ as $s_j - 1$, by assuming base processing rate to be 1. Then, the expression $\sum_{j'=j}^{k} s_j^{'} - (k-j+1)$ stands for the amount of total additional processing rate from machines $j$ to $k$, $k > j$. The following constraint then ensures contiguous assignments of machines within the environment of uniform machines.

$$Z_{ijlt} + Z_{iklt} \leq 1$$

$$\forall i, \forall j, \forall k, \forall t, \forall l \in (2, ..., (k-j) + \sum_{j'=j}^{k} s_j^{'} - (k-j+1)) : k > j \quad (2.26)$$

For example, assume that machine 1 is two times faster than machines 2 and 3, i.e., $s_1$=2, $s_2$=$s_3$=1. Then, we should also restrict the case in which machines 1 and 3 are assigned to the same task and to total processing rate of 3 in addition to 2. Otherwise, we could mistakenly assign machines 1 and 3 to a same task and total processing rate of 3 by disregarding the existence of in-between machine. In other words, if machines 1 and 3 are assigned to same machine, then they cannot be assigned to any total processing rate less than 4, by considering not only the presence of machine 2, but also the fact that processing rate of machine 1 being 2.

Proposition 6 suggests that LBBD algorithm for MTSP without contiguous assignments is guaranteed to be solved within the first iteration. Unfortunately, it does not hold for any of the extensions/properties when we consider QCAP environment because all other constraints need to be implemented together with contiguous assignments of machines. Moreover, in LBBD, it is important to embed a relaxation of the subproblem in the master problem (Hooker, 2007). Thus, we can eliminate some of the solutions that do not satisfy the subproblem by adding the following

valid inequalities to the master problem.

- *"Total number of vessels that are being serviced at any time must be less than or equal to the total number of QCs."*

$$\sum_i \sum_l \sum_{h=max(1,t-p_{il}+1)}^{t} Q_{ilh} \leq |K| \qquad \forall t \quad (2.27)$$

- *"Total number vessels that are assigned to a total processing rate of one must be less than total number of QCs with $s_j$=1."* We only add this constraint to master problem if an infeasibility found at the first iteration because we observe that it does not affect the result of the first iteration while increasing solution time significantly. However, if an infeasibility is found, then it decreases the number of iterations.

$$\sum_i \sum_{h=max(1,t-p_{i1}+1)}^{t} Q_{i1h} \leq \sum_{j:s_j=1} s_j \qquad \forall t \quad (2.28)$$

- *"Some combinations of total processing rates cannot be assigned to vessels at the same time."* In container terminals, cranes with the same processing rate are located adjacently rather than arbitrarily. This makes simultaneous existence of some combinations of odd numbered total processing rates infeasible even it satisfies all other constraints. We should investigate terminal setting and manually add such constraints to the master problem.

  For example, consider a terminal with 6 QCs with respective processing rates of 2, 2, 2, 1, 1, and 1. In such a setting, we cannot assign two vessels with 3 total processing rate each, if they overlap in time. Even if it satisfies constraints (2.22), (2.23) and (2.24), there are only two possible assignments that result in total processing rate of 3, {QC 3, QC 4} and {QC 4, QC 5, QC 6}, which both require QC 4. Therefore, we should add the following constraint.

$$\sum_i \sum_{h=max(1,t-p_{i3}+1)}^{t} Q_{i3h} \leq 1 \qquad \forall t \quad (2.29)$$

### *2.5.3 Other Objective Functions*

In container terminals, different objectives are considered for operational problems by the terminal management (Bierwirth and Meisel, 2010). One advantage of the proposed solution technique is the flexibility to model different objective functions since it is developed based on mathematical programming.

In this chapter, we consider the objective of minimizing total weighted completion times of all vessels. A special case of this objective function, minimizing total completion times of vessels, can be simply derived by setting weight as 1 for all vessels. Another objective function which is frequently considered in the literature is to minimize makespan, i.e., minimizing the maximum completion time. Let $C_{max}$ be a continuous decision variable. Then, we add constraints (2.30) and (2.31) to the master problem.

$$C_{max} \geq \sum_{j} \sum_{l} \sum_{t} (t + p_{il}) Z_{ijlt} / l \qquad \forall i \quad (2.30)$$

$$C_{max} \geq (\sum_{i} \min_{l}(p_{il} l)) / |K| \qquad (2.31)$$

The objective function should be modified as minimization of $C_{max}$. Constraint (2.30) ensures that the value of makespan to be greater than or equal to the completion time of each task. When considered together with the objective function, it is equal to the largest completion time. In contrast to the problem with total completion times objectives, the LP relaxation is not tight for this objective. That is why we introduce preemptive lower bound by constraint (2.31).

In computational experiments section, we test the performance of the proposed approach with these two objective functions, in addition to minimizing total weighted completion times of vessels. Other objective functions such as minimization of earliness/tardiness, total tardiness, and number of late jobs can also be implemented within the proposed LBBD straightforwardly.

### *2.5.4 Application to Detailed QCAP*

Recall that we represent QCAP as a variant of MTSP in which contiguous assignments of QCs are considered because MTSP allows modeling of handling time of a vessel which depends on the number of assigned QCs. In any container terminal, QCs are located on the same rail track, and we ensure this via contiguous assignments constraint. Moreover, we consider an extended formulation of MTSP with specific task to machine assignments to be able to implement various properties of QCAP.

In the following, we introduce a detailed QCAP by utilizing the methods we previously presented: we determine specific QC to vessel assignments and schedules, while considering different vessel classes, tidal time windows, QC maintenance, and QCs with different processing rates. Up to date, the literature studied QCAP without considering any of these properties. The flexibility obtained by specifying assignments allows us to formulate this detailed problem. In this application, there are $|J|$ vessels and $|K|$ QCs, and each vessel $i$ have total workload of $p_{i1}$ given in time units of hours. Other properties as well as assumptions of the problem are listed as follows:

- There are three different classes of vessels: Feeder, Medium, Jumbo.
- There are $|R|$ tidal time periods, and a vessel can only depart during a high tide. Start and end times for each of these periods are represented as $(b_r, e_r)\,, \quad r = \{1, \ldots, |R|\}$.
- Number of QCs to handle a vessel is pre-specified between a lower bound $\underline{\delta}_i$ and an upper bound $\delta_i$, based on the class of this vessel.
- As they have twin-lifts, Some QCs are two times faster ($s_j = 2$) than others ($s_j = 1$).
- Let $M$ be the set of QCs that require maintenance in the planning period, $M \subset K$. Then maintenance period for QC $j$ is defined as $(mb_j, me_j)$, if $j \in M$.

The literature mostly addresses time-invariant QCAP together with berth allocation decisions. Even though we do not specify exact berthing position in the proposed model (as in Blazewicz et al., 2011), it can be derived from the solution to QCAP. That is, we consider contiguous assignments of QCs, thus we obtain relative positions of vessels over berth as well as predefined minimum and maximum QCs to handle each vessel by considering its size. Also note that, in the literature, the decision made for specific berthing positions are commonly used to determine the handling time of vessels based on the availability of QCs on that position (Imai, 2001). Therefore, it can be ignored as we already determine specific QCs to handle each vessel. This argument on the derivation of the berth allocation plan is only valid if the length of the berth area is not very limited for the number of available QCs. If not, we can consider extending the master problem of LBBD-MTSP with a constraint that limits the total length of vessels that moor at the same time.

### *2.6 Computational Experiments for LBBD-MTSP*

We design a set of computational experiments to evaluate the performance of the proposed LBBD for MTSP with contiguous assignments, its extensions, and application to QCAP. In container terminals, QCs operate on the same rail track, thus they cannot cross each other. When we increase the total number of QCs assigned to a vessel, cranes are more likely to interfere with each other, causing a decrease in their marginal productivity. This relation can be captured by interference exponent $\alpha$, such that $f_l^2 = l^\alpha$ (Meisel and Bierwirth, 2009) In this study, we set $\alpha$ as 0.85 in line with the literature. In addition, we perform tests with a logaritmic speed up function ($f_l^1 = \log_{1.75}(l + 0.75)$). Table 2.3 presents corresponding speed up factors, processing times, and total work to be performed by all machines for $f_l^1$ and $f_l^2$. For example, by assuming $f_l^1$, if we process this task with 4 machines, time required for its processing reduces from 60 to 22, while total work to be performed by all machines increases from 60 to 88. Clearly, $f_l^1$ tends to lose more workforce when $l$ increases compared to $f_l^2$ .

Table 2.3: Comparison of speed up functions $f_l^1$ and ($f_l^2$) for $p_{i1}$=60

| $l$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| $f_l$ | 1 (1) | 1.81 (1.81) | 2.36 (2.54) | 2.78 (3.25) | 3.13 (3.93) | 3.41 (4.59) | 3.66 (5.23) | 3.88 (5.86) | 4.07 (6.47) | 4.24 (7.08) |
| $p_{il}$ | 60 (60) | 34 (34) | 26 (24) | 22 (19) | 20 (16) | 18 (14) | 17 (12) | 16 (11) | 15 (10) | 15 (9) |
| $l.p_{il}$ | 60 (60) | 68 (68) | 78 (72) | 88 (76) | 100 (80) | 108 (84) | 119 (84) | 128 (88) | 135 (90) | 150 (90) |

The size of time-indexed models grows exponentially as maximum processing time $p_{max}$ increases (Sousa and Wolsey, 1992). To see the effect of $p_{max}$ on the solution performance, we uniformly select $p_{i1}$ from two different intervals, [10,30] and [20,60]. For each interval, 20 different instance sizes are considered with $|J|$=20, 25, 30, 35, 40 and $|K|$=4, 6, 8, 10. For each instance size, we generate 10 random instances. Without loss of generality, we set the maximum total number of machines that can process a task ($\delta_i$) as $|K|/2$. We set the length of the planning period as $|T| = \lceil p_{max}(|J|/|K|) \rceil$ and set time limit as one hour.

If there are different types of QCs in a container terminal, then QCs of the same type are placed adjacently on the rail track. Accordingly, to test QCAP with uniform machines, we assume that processing rates of QCs with first $|K|/2$ indices are twice as high as those of other QCs. For unavailability of QCs, we consider two different patterns. First, we assume that machines with the first two indices are unavailable within time interval [1,30]. Second, we test the proposed approach by assuming the unavailability of QC $|K|/2$ within [31,60]. Note that unavailable QCs are simply located at leftmost of rail track in the first pattern while the latter restricts more QC to vessel assignments since unavailable QC is located between others. We test these properties of QCAP by assuming speed up function $f_l^2$ as well as processing times are generated from interval of [20,60].

For the application of detailed QCAP, in line with literature (Meisel and Bierwirth, 2009), we assume that 60% of vessels belong to class Feeder, 30% of them belong to class Medium, and 10% of them belong to class Jumbo. The interval in which

processing times of vessels is uniformly determined and min-max QC requirements for each class are presented in Table 2.4. The schedule for tidal periods is derived from the Port of Newcastle, Australia. We also assume that processing rates of QCs 1 and 2 are two times faster than other QCs, and QC 5 has a planned maintenance within time interval [31,60]. We test this problem for 20, 25, and 30 vessels with total 10 QCs for a planning period of a week ($|T|$=168 hours).

Table 2.4: Parameters for QCAP

| Type | $p_{il}$ (hours) | $\underline{\delta}$ | $\delta$ |
|---|---|---|---|
| Feeder | [5,15] | 1 | 2 |
| Medium | [15,50] | 2 | 4 |
| Jumbo | [50,65] | 4 | 6 |

Initially, we test MIP formulation of MTSP with contiguous assignments [$MTSP-CA$]. We observe out of memory error at the root node in all instances, even without contiguous assignments constraint. It is mainly due to the constraints (2.4) and (2.5) that address two unique properties associated with $Z_{ijlt}$ variables. As we have shown in Section 2.2, these constraints can be removed by hybridizing [$MTSP-CA$] with $Q_{ilt}$ variables. The resulting hybrid formulation demands much less memory compared to [$MTSP-CA$] but still has a limited performance because of the existence of capacity constraint for each machine. That is, it can solve up to 25 tasks instances with 4 machines, 20 tasks instances with 6 machines.

Results suggests that the proposed LBBD for MTSP with contiguous assignments solves most of the instances to the optimality at the first iteration while only a few instances require cut(s). This means that the most important determinant of the performance of the proposed method is its master problem. For this reason, we disregard contiguous assignment constraint in the following experiment and evaluate

the performance of the algorithm with different speed up functions and processing time intervals. By Proposition 6, it is guaranteed to have an optimal solution at the first iteration.

In Tables 2.5 and 2.6, we present results of the experiments for the proposed LBBD for MTSP with specific assignments. For each instance size, we list:

($i$) total number of instances solved to optimality within the time limit,

($ii$) average percentage deviation between the objective value of LP relaxation and optimal solution value, and

($iii$) average solution time in seconds.

We also list ($iv$) average percentage deviation of the optimal solution value from the optimal value of the corresponding parallel machine scheduling problem, which indicates the advantage obtained by being able to process a task with multiple machines simultaneously. The subproblem is solved in less than two tenths of a second for all instances, hence we included it into solution time ($iii$).

Table 2.5 presents results for the experiments with the proposed LBBD for MTSP with specific assignments for both speed up functions when $p_{i1}$ is selected within interval of [10,30]. All instances are solved to the optimality with an average solution time of 16 and 74 seconds approximately, and the average percent deviation between the values of LP relaxation and the optimal solution are 0.17 % and 0.24% for speed up functions $f_l^1$ and $f_l^2$, respectively. Similarly, Table 2.6 presents the result for $p_{i1} \in$ [20,60]. All instances are solved to optimality with an average of 56 seconds with $f_l^1$. With $f_l^2$, 189 out of 200 instances are solved to the optimality within the time limit while the average solution time is jumped to 569 seconds. The average deviation between the values of LP relaxation and the optimal solution is almost identical with those observed for $p_{i1} \in$ [10,30]. For 11 instances in which solution process is terminated by the time limit, the average optimality gap is observed as 0.32%.

One important result from Tables 2.5 and 2.6 is the solution time differences between speed up functions. The reason can be explained as follows: $f_l^2$ provides faster processing times (and less total work to be performed by all assigned machines) as

Table 2.5: Results of LBBD-MTSP with $p_{i1} \in [10, 30]$

| | | Logarithmic ($f_l^1$) | | | | Exponential ($f_l^2$) | | | |
|---|---|---|---|---|---|---|---|---|---|
| $\|K\|$ | $\|J\|$ | $(i)$ | $(ii)$ | $(iii)$ | $(iv)$ | $(i)$ | $(ii)$ | $(iii)$ | $(iv)$ |
| 4 | 20 | 10 | 0.16 | 2.4 | 1.75 | 10 | 0.13 | 1.8 | 1.75 |
| | 25 | 10 | 0.10 | 7.2 | 1.16 | 10 | 0.09 | 3.5 | 1.16 |
| | 30 | 10 | 0.04 | 9.7 | 0.70 | 10 | 0.04 | 5.3 | 0.70 |
| | 35 | 10 | 0.03 | 12.4 | 0.41 | 10 | 0.03 | 8.0 | 0.41 |
| | 40 | 10 | 0.02 | 15.7 | 0.33 | 10 | 0.02 | 13.0 | 0.33 |
| | | avg. | 0.07 | 9.5 | 0.87 | avg. | 0.06 | 6.3 | 0.87 |
| 6 | 20 | 10 | 0.42 | 4.0 | 4.66 | 10 | 0.45 | 4.1 | 5.51 |
| | 25 | 10 | 0.23 | 10.3 | 3.06 | 10 | 0.26 | 7.4 | 3.56 |
| | 30 | 10 | 0.13 | 11.6 | 2.04 | 10 | 0.23 | 20.7 | 2.35 |
| | 35 | 10 | 0.09 | 21.1 | 1.38 | 10 | 0.12 | 33.5 | 1.55 |
| | 40 | 10 | 0.04 | 39.2 | 0.99 | 10 | 0.06 | 57.4 | 1.12 |
| | | avg. | 0.18 | 17.2 | 2.43 | avg. | 0.22 | 24.6 | 2.82 |
| 8 | 20 | 10 | 0.28 | 3.7 | 8.70 | 10 | 0.43 | 7.0 | 11.05 |
| | 25 | 10 | 0.29 | 6.9 | 5.73 | 10 | 0.39 | 16.7 | 7.19 |
| | 30 | 10 | 0.26 | 13.5 | 3.83 | 10 | 0.31 | 82.2 | 4.91 |
| | 35 | 10 | 0.19 | 29.9 | 3.01 | 10 | 0.22 | 80.8 | 3.91 |
| | 40 | 10 | 0.08 | 30.1 | 2.08 | 10 | 0.15 | 186.7 | 2.68 |
| | | avg. | 0.22 | 16.8 | 4.67 | avg. | 0.30 | 74.7 | 5.95 |
| 10 | 20 | 10 | 0.36 | 4.8 | 11.16 | 10 | 0.52 | 14.8 | 15.02 |
| | 25 | 10 | 0.23 | 8.3 | 8.98 | 10 | 0.54 | 33.2 | 12.09 |
| | 30 | 10 | 0.23 | 27.0 | 6.27 | 10 | 0.38 | 101.6 | 8.55 |
| | 35 | 10 | 0.19 | 25.9 | 4.87 | 10 | 0.30 | 171.2 | 6.37 |
| | 40 | 10 | 0.12 | 41.3 | 3.58 | 10 | 0.20 | 625.8 | 4.54 |
| | | avg. | 0.23 | 21.5 | 6.97 | avg. | 0.39 | 189.3 | 9.31 |

Table 2.6: Results of LBBD-MTSP with $p_{i1} \in [20, 60]$

| | | Logarithmic ($f_l^1$) | | | | Exponential ($f_l^2$) | | | |
|---|---|---|---|---|---|---|---|---|---|
| $\|K\|$ | $\|J\|$ | $(i)$ | $(ii)$ | $(iii)$ | $(iv)$ | $(i)$ | $(ii)$ | $(iii)$ | $(iv)$ |
| 4 | 20 | 10 | 0.19 | 6.6 | 2.32 | 10 | 0.18 | 6.0 | 2.27 |
| | 25 | 10 | 0.16 | 17.4 | 1.38 | 10 | 0.16 | 14.6 | 1.25 |
| | 30 | 10 | 0.06 | 23.2 | 0.93 | 10 | 0.06 | 21.8 | 0.87 |
| | 35 | 10 | 0.04 | 52.3 | 0.66 | 10 | 0.04 | 53.7 | 0.61 |
| | 40 | 10 | 0.02 | 121.0 | 0.41 | 10 | 0.02 | 131.7 | 0.38 |
| | | | 0.10 | 44.1 | 1.14 | | 0.09 | 45.62 | 1.08 |
| 6 | 20 | 10 | 0.24 | 7.1 | 6.70 | 10 | 0.34 | 10.7 | 7.65 |
| | 25 | 10 | 0.21 | 18.7 | 4.08 | 10 | 0.30 | 63.2 | 4.46 |
| | 30 | 10 | 0.18 | 42.5 | 2.66 | 10 | 0.19 | 144.3 | 2.92 |
| | 35 | 10 | 0.10 | 69.2 | 1.99 | 10 | 0.14 | 395.5 | 2.07 |
| | 40 | 10 | 0.05 | 147.5 | 1.29 | 10 | 0.08 | 929.6 | 1.33 |
| | | | 0.16 | 57.0 | 3.35 | | 0.21 | 308.65 | 3.69 |
| 8 | 20 | 10 | 0.14 | 5.9 | 10.78 | 10 | 0.50 | 22.5 | 13.01 |
| | 25 | 10 | 0.20 | 16.8 | 7.13 | 10 | 0.38 | 102.8 | 8.50 |
| | 30 | 10 | 0.19 | 53.1 | 5.15 | 10 | 0.24 | 365.7 | 6.01 |
| | 35 | 10 | 0.16 | 85.2 | 3.60 | 9 | 0.25 | 916.8 | 4.11 |
| | 40 | 10 | 0.14 | 209.4 | 3.83 | 9 | 0.16 | 1693.4 | 3.21 |
| | | | 0.16 | 74.1 | 5.90 | | 0.31 | 620.2 | 6.97 |
| 10 | 20 | 10 | 0.25 | 7.4 | 13.92 | 10 | 0.56 | 46.4 | 17.36 |
| | 25 | 10 | 0.14 | 20.8 | 10.46 | 10 | 0.46 | 317.6 | 12.81 |
| | 30 | 10 | 0.19 | 37.2 | 8.13 | 9 | 0.40 | 1377.7 | 10.07 |
| | 35 | 10 | 0.15 | 76.1 | 6.11 | 6 | 0.25 | 2148.7 | 7.16 |
| | 40 | 10 | 0.14 | 100.7 | 4.74 | 6 | 0.23 | 2626.9 | 5.48 |
| | | | 0.17 | 48.4 | 8.67 | | 0.38 | 1303.5 | 10.58 |

total number of assigned machines increases compared to $f_l^1$, so it is harder to eliminate nodes in the search tree. As a result, we need to investigate significantly higher number of nodes to find an optimal solution. These tables also show the advantage obtained by being able to process tasks by more than one machine compared to parallel machine scheduling in which tasks are processed by one and only one machine. Results indicates that such an advantage increases as the ratio of number of tasks to number of machines ($|J|/|K|$) decreases as it allows processing of tasks by more machines.

We use the same set of instances with $p_{i1} \in [20,60]$ and $f_l^2$ to test the proposed method with different objective functions: total completion times and makespan minimization. Table 2.7 shows that we can solve lesser number of instances with10 machines to optimality with total completion times objective compared to its weighted version, even though the former has a tighter LP relaxation on average. One possibility for such a behavior could be the number of symmetric task pairs. When we disregard the weights of tasks, number of symmetric task pairs increases and we might need to explore a larger portion of the search tree. For makespan minimization objective, we solve all 200 instances to optimality within the time limit while observing a smaller increase in solution time as the instance size increases. For example, if we consider makespan minimization, average solution time is 255.8 seconds for instances with four machines and it only increases to 525.1 seconds for instances with 10 machines. On the other hand, for total weighted completion time minimization objective, solution time increases from 45.5 to 1291 seconds for the same set of instances. Based on these results, we can claim that the proposed approach can cope with different objective effectively.

For MTSP with contiguous assignments 195 (out of 200) and 186 (out of 189) instances are solved to optimality in the first iteration of the LBBD for speed up functions $f_l^1$ and $f_l^2$, respectively. For these instances, solution times are approximately same with those presented in Table 2.6 because master problems of LBBD for MTSP with/without contiguous assignments are identical, and subproblems for all

Table 2.7: Results of LBBD-MTSP for different objective functions

| | | T.W.C.T. | | | T.C.T. | | | Makespan | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\|K\|$ | $\|J\|$ | (i) | (ii) | (iii) | (i) | (ii) | (iii) | (i) | (ii) | (iii) |
| 4 | 20 | 10 | 0.18 | 6.0 | 10 | 0.07 | 5.7 | 10 | 0 | 12.8 |
| | 25 | 10 | 0.16 | 14.6 | 10 | 0.03 | 8.2 | 10 | 0 | 63.4 |
| | 30 | 10 | 0.06 | 22.0 | 10 | 0.02 | 17.8 | 10 | 0 | 202.5 |
| | 35 | 10 | 0.04 | 53.8 | 10 | 0.01 | 20.4 | 10 | 0 | 335.6 |
| | 40 | 10 | 0.02 | 130.9 | 10 | 0.01 | 147.0 | 10 | 0 | 664.91 |
| | | | 0.09 | 45.5 | | 0.03 | 39.8 | | 0 | 255.8 |
| 6 | 20 | 10 | 0.34 | 11.2 | 10 | 0.33 | 28.4 | 10 | 0 | 13.9 |
| | 25 | 10 | 0.30 | 62.9 | 10 | 0.18 | 82.2 | 10 | 0 | 102.4 |
| | 30 | 10 | 0.19 | 146.9 | 10 | 0.05 | 150.1 | 10 | 0 | 319.8 |
| | 35 | 10 | 0.14 | 394.0 | 9 | 0.11 | 621.1 | 10 | 0 | 428.4 |
| | 40 | 10 | 0.08 | 936.3 | 10 | 0.07 | 615.2 | 10 | 0 | 1635.8 |
| | | | 0.21 | 310.3 | | 0.15 | 299.4 | | 0 | 500.1 |
| 8 | 20 | 10 | 0.50 | 23.5 | 10 | 0.43 | 54.0 | 10 | 1.00 | 41.3 |
| | 25 | 10 | 0.37 | 138.5 | 10 | 0.29 | 238.9 | 10 | 0.08 | 118.1 |
| | 30 | 10 | 0.25 | 754.0 | 10 | 0.21 | 865.1 | 10 | 0 | 324.0 |
| | 35 | 9 | 0.25 | 911.7 | 10 | 0.10 | 677.9 | 10 | 0 | 581.1 |
| | 40 | 9 | 0.16 | 1695.8 | 10 | 0.09 | 1110.8 | 10 | 0 | 1432.6 |
| | | | | 704.7 | | 0.22 | 589.3 | | 0.22 | 499.1 |
| 10 | 20 | 10 | 0.56 | 44.3 | 10 | 0.51 | 256.5 | 10 | 1.98 | 45.2 |
| | 25 | 10 | 0.46 | 311.0 | 10 | 0.33 | 603.3 | 10 | 1.07 | 94.0 |
| | 30 | 9 | 0.40 | 1319.3 | 7 | 0.22 | 1490.6 | 10 | 0.08 | 442.0 |
| | 35 | 6 | 0.25 | 2146.0 | 4 | 0.18 | 2153.2 | 10 | 0 | 1052.8 |
| | 40 | 6 | 0.23 | 2635.5 | 3 | 0.28 | 2372.2 | 10 | 0 | 991.6 |
| | | | 0.38 | 1291.2 | | 0.31 | 1375.2 | | 0.62 | 525.1 |

instances are still solved within two tenths of a second, despite containing an additional constraint for contiguous assignments. Results of other instances that cannot be solved in the first iteration are presented in Tables 2.8 and 2.9 for $f_l^1$ and $f_l^2$, respectively. For each instance, we present optimal values at the first iteration and the last iteration, number of cuts, and solution times for LBBD algorithm with nogood cuts and the proposed Benders cuts.

With speed up function $f_l^1$, our proposed cut is equal to nogood cut with symmetry elimination in three of these five instances, as the infeasibility is eliminated by removing the task which has the largest completion time in the optimal solution to the master problem. By considering symmetric task pairs within a nogood cut, we can reduce the number of total number of required iterations up to $2^n$ times where $n$ denotes the number of symmetric task pairs in a problem instance. Otherwise, the algorithm requires an iteration for each symmetric solution.

In two remaining instances, proposed Benders cut identifies that infeasibility is caused by a considerably smaller subset of all variables that take the value of 1 in optimal solution. For the instance with 20 tasks - 6 machines that has no symmetric task pairs, 94 nogood cuts are required, while it can be solved with only 7 proposed Benders cuts. In another instance with 35 tasks - 10 machines that has one symmetric task pair, 53 proposed Benders cuts are required to solve the instance to optimality, while 267 nogood cuts with symmetry elimination are not enough to solve the instance to optimality even within the extended time-limit of five hours. Excluding this instance, we observe that the proposed cut shortens the average solution times by approximately 8 times compared to nogood cuts. It is also worth mentioning that any of these five instances cannot be solved by the hybrid formulation within five hours. On the other hand, proposed LBBD algorithm can effectively solve this extension of the problem since five instances are solved in approximately 15 minutes on average. That indicates the significance of the proposed decomposition. Moreover, Table 2.9 presents similar results for speed up function $f_l^2$. Based on these results, we claim that the proposed Benders cut reduces the average number of iterations and

the solution time significantly.

Table 2.10 presents the results of computational experiments for the application of the LBBD-MTSP to a detailed QCAP setting. For each instance, we report the solution performance at the first iteration. If an instance require at least one Benders cut, then we also present the comparison of nogood and the proposed cuts in terms of the total number of iterations and the solution time. Out of 30 total instances, we are able to solve 28 of them to optimality within the time limit of an hour, while 17 out of these 28 instances are solved at the first iteration. In other 11 instances, we observe a significant advantage of proposed Benders cut. Note that we extend the limit to five hours for the tests with nogood cuts because it fails to solve most of these 11 instances within an hour.

For the instances which are solved to optimality within the respective time limits by both types of cuts, the average number of required iterations are 65 for nogood cuts and 10.1 for proposed cuts. This gap further increases if we consider the remaining instances. For example, in instance 5 of 30 vessels instances, 285 iterations with nogood cuts cannot solve the instance to optimality even within five hours. On the other hand, we are able to solve the same instance within less than three minutes with just 6 iterations with the proposed cut. Another example is the instance 3 of the instances with 30 vessels. Even though LBBD with both types of cuts is terminated by respective time limits, we observe that the proposed cut is able to increase lower bound to 4837 within an hour, while it is just 4825 with nogood cuts after five hours.

As we mentioned earlier, the length of the planning horizon $|T|$ is critical for the computational performance of time-indexed formulations. There are two factors that define $|T|$: (i) the largest processing time $p_{max}$ and (ii) the way we set $|T|$, given $p_{max}$.

We have previously presented the impact of $p_{max}$ for values of 30 and 60 in Tables 2.5 and 2.6, respectively. Now, we analyze the second factor, i.e. the selection of $|T|$, given $p_{max}$=60. While selecting the length of the planning horizon, we deal with the following trade-off: $|T|$ should be large enough to allow global optimal schedule, however if we select it too large, then it affects the computational performance of

Table 2.8: Results of LBBD-MTSP for the instances that require Benders cuts ($f_l^1$)

| $\|K\| - \|J\| - \|ins\|$ | $z^*$ end of the first iteration | $z^*$ end of the algorithm | LBBD nogood cut | | LBBD proposed cut | |
|---|---|---|---|---|---|---|
| | | | #its. | time | #its. | time |
| 20-6-9 | 6870 | 6877 | 94 | 3300.5 | 7 | 119.0 |
| 25-6-9 | 10020 | 10020 | 2 | 20.3 | 2 | 20.5 |
| 30-10-1 | 8669 | 8671 | 56 | 1253.1 | 14 | 368.8 |
| 35-10-1 | 11977 | 11996 | 267 | 18000[1] | 53 | 2587.3 |
| 40-10-9 | 13788 | 13789 | 14 | 858.2 | 3 | 176.4 |
| | | | 86.4 | 1358.0[2] | 15.6 | 171.2[2] |

[1] CPLEX cannot solve to optimality within the time limit

[2] Average solution time of the instances solved within the time limit

Table 2.9: Results of LBBD-MTSP for the instances that require Benders cuts ($f_l^2$)

| $\|K\| - \|J\| - \|ins\|$ | $z^*$ end of the first iteration | $z^*$ end of the algorithm | LBBD nogood cut | | LBBD proposed cut | |
|---|---|---|---|---|---|---|
| | | | #its. | time | #its. | time |
| 25-8-3 | 6881 | 6883 | 23 | 3209.4 | 2 | 342.5 |
| 30-8-5 | 10228 | 10231 | 31 | 18000[1] | 6 | 3185.3 |
| 30-8-7 | 11046 | 11047 | 11 | 5310.0 | 2 | 1020.8 |
| | | | 17 | 4260.3[2] | 2 | 681.6[2] |

[1] CPLEX cannot solve to optimality within the time limit

[2] Average solution time of the instances solved within the time limit

the formulation adversely. Note that, up to now, we use an adaptive strategy such that $|T| = \lceil p_{max}(|J|/|K|) \rceil$. In the following, we perform a new set of computational

Table 2.10: Results of LBBD-MTSP for the detailed QCAP application

| | | | First Iteration | | | Nogood Cut | | | Proposed Cut | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\|K\|$ | $\|J\|$ | ins. | $z^*$ | (ii) | (iii) | #its. | $z^*$ | (iii) | #its. | $z^*$ | (iii) |
| 10 | 20 | 1 | 2488 | 0.49 | 2.6 | | | | | | |
| | | 2 | 2219 | 0.38 | 2.1 | 243 | 2226 | 3711.1 | 23 | 2226 | 402.5 |
| | | 3 | 2221 | 0.64 | 2.7 | | | | | | |
| | | 4 | 3001 | 1.58 | 7.3 | 77 | 3006 | 1531.3 | 10 | 3006 | 199.0 |
| | | 5 | 1774 | 0.79 | 3.0 | | | | | | |
| | | 6 | 2968 | 1.53 | 5.9 | | | | | | |
| | | 7 | 2019 | 1.80 | 4.0 | | | | | | |
| | | 8 | 2229 | 1.27 | 3.9 | | | | | | |
| | | 9 | 3303 | 1.56 | 5.9 | | | | | | |
| | | 10 | 3677 | 1.75 | 28.1 | 12 | 3679 | 338.3 | 5 | 3679 | 134.5 |
| | | | 2589.9 | 1.18 | 6.6 | 110.7 | | 1860.2 | 12.7 | | 245.3 |
| 10 | 25 | 1 | 3854 | 1.23 | 21.0 | | | | | | |
| | | 2 | 2736 | 0.48 | 3.9 | 5 | 2737 | 66.1 | 2 | 2737 | 15.1 |
| | | 3 | 2226 | 1.12 | 17.1 | 27 | 2228 | 1020.8 | 8 | 2228 | 340.0 |
| | | 4 | 3167 | 1.40 | 12.3 | | | | | | |
| | | 5 | 4057 | 1.33 | 8.3 | 239 | 4060 | 18000[1] | 6 | 4061 | 161.7 |
| | | 6 | 2613 | 1.17 | 8.9 | | | | | | |
| | | 7 | 2966 | 0.82 | 6.8 | 28 | 2969 | 511.8 | 2 | 2969 | 22.9 |
| | | 8 | 6564 | 1.26 | 122.8 | 58 | 6571 | 18000[1] | 15 | 6575 | 3600[1] |
| | | 9 | 3938 | 1.09 | 11.2 | 23 | 3943 | 921.5 | 15 | 3943 | 698.2 |
| | | 10 | 3403 | 0.87 | 4.4 | | | | | | |
| | | | 3552.4 | 1.08 | 21.7 | 20.8[2] | | 630.1[2] | 6.8[2] | | 269.1[2] |
| 10 | 30 | 1 | 4325 | 0.71 | 28.4 | | | | | | |
| | | 2 | 3674 | 0.68 | 11.7 | | | | | | |
| | | 3 | 4819 | 0.62 | 11.4 | 285 | 4825 | 18000[1] | 45 | 4837 | 3600[1] |
| | | 4 | 5128 | 1.13 | 19.6 | 190 | 5132 | 9333.9 | 30 | 5132 | 1130.1 |
| | | 5 | 5114 | 1.06 | 59.7 | 75 | 5118 | 18000[1] | 17 | 5131 | 2755.4 |
| | | 6 | 7520 | 0.59 | 89.5 | | | | | | |
| | | 7 | 7901 | 0.67 | 254.2 | | | | | | |
| | | 8 | 4284 | 0.97 | 43.2 | 27 | 4288 | 787.2 | 2 | 4288 | 125.1 |
| | | 9 | 6096 | 0.50 | 7.4 | 18 | 6100 | 290.4 | 4 | 6100 | 104.8 |
| | | 10 | 4111 | 1.27 | 42.6 | | | | | | |
| | | | 5297.2 | 0.82 | 57.8 | 78.3[2] | | 3470.7[2] | 12[2] | | 453.3[2] |

[1] CPLEX cannot solve to optimality within the time limit

[2] Average solution time of the instances solved within the time limit

experiments by multiplying the previous $|T|$ value with 1.5.

In Table 2.11, we compare the new and the previous $|T|$ selection functions in terms of the ratio of the largest completion time in the optimal schedule to the schedule length (column (v)) and solution time in seconds (column (iii)).

Results show that $\lceil p_{max}(|J||K|)\rceil$ is large enough as it does not restrict global optimal schedules. That is, for each instance, we obtain the same optimal schedule when testing with a larger $|T|$ value. At the same time, it is not unnecessarily large because the ratio of the largest completion time in the optimal schedule to the length of the schedule is 0.776 on average. However, the effect of the way we select the $|T|$ value on the average solution time is lesser compared to the effect of $p_{max}$, as can be observed from Tables 2.5 and 2.6. In other words, even if we set an unnecessarily large $|T|$ value, we do not observe a significant decrease in the computational performance.

Table 2.11: Results of LBBD-MTSP for different length of schedule ($|T|$)

| | $\|T\| = \lceil p_{max}(\frac{\|J\|}{\|K\|})\rceil$ | | | $\|T\| = 1.5\lceil p_{max}(\frac{\|J\|}{\|K\|})\rceil$ | | |
|---|---|---|---|---|---|---|
| $\|K\|$ | (i) | (iii) | (v) | (i) | (iii) | (v) |
| 4 | 50 | 45.6 | 0.707 | 50 | 51.0 | 0.477 |
| 6 | 50 | 308.7 | 0.747 | 50 | 422.5 | 0.498 |
| 8 | 48 | 620.2 | 0.806 | 47 | 766.9 | 0.538 |
| 10 | 41 | 1305.5 | 0.844 | 39 | 1481.2 | 0.553 |
| | | 570.0 | 0.776 | | 680.4 | 0.517 |

## *2.7 Relationships with Other Scheduling Problems*

Assume that there are $n$ rectangular items with integer width $width_i$ and height $h_i$, and there is a strip of fixed width $W$. Then, strip packing problem is to pack all $n$ items into this strip without overlapping. The objective is to minimize makespan, i.e., strip length (Lodi et al., 2002).

We already know that multiprocessor task (MPT) scheduling problem is a special case of MTSP in which number of machines to process each task is fixed a priori. Moreover, MPT scheduling problem is equivalent to strip packing problem when contiguous assignment of tasks are assumed. In such a case, each task can be represented as a rectangle where number of machines required to process a task and processing time stands for width and height of a rectangle, respectively.

An important variant of strip packing problem is the one that allows 90° rotation of items because the nature of the problem makes the orthogonal rotation of items possible. The resulting problem is a generalization of strip packing problem (SPP), which is categorized as 2D SPP-RF in the literature. By investigating this relation together with the relation between multiprocessor task scheduling and MTSP, we identify the following relation.

**Proposition 8.** *Strip packing problem with 90° rotations is a special case of MTSP with contiguous assignments.*

*Proof.* Each strip packing problem with 90° rotations can be transformed into an equivalent MTSP with contiguous assignments instance. Assume that $width_i \neq h_i$ and total number of machines $|K|$ equals strip width W. Then the following processing time structure completes the transformation:

$$p_{il} = \begin{cases} h_i, & if\ l = width_i \\ width_i, & if\ l = h_i \\ \infty, & otherwise \end{cases}$$

□

Furthermore, the methods we developed allow modeling different types of properties of SPP. The literature on the subject assumes that whole strip is available. However, there may be defects on the strip or parts of the strip may be cut previously to fulfill some urgent orders. Hence, it might be the case that strip may not be completely available. We can model this extension of strip packing problem with 90° rotations by simply adding time windows constraints that we developed.

Integrated problem of audit project scheduling and staffing is a variant of multi-mode RCPSP (Fernandes-Viagas and Framinan, 2014). In this problem, processing time of an audit task depends on the number of auditors assigned to this task, and auditors need to have some certain skills/qualifications to be assigned to a specific audit tasks. Such a problem can be coped with the proposed LBBD, if we extend it with an eligibility constraint. Since our MTSP formulation specifies task to machine assignments, we can easily eliminate non-eligible audit task to auditor assignments.

## *2.8 Conclusion*

In quay crane assignment problem (QCAP), we allocate quay cranes to perform unloading and loading operations of container vessels. A solution to QCAP is one of the major factors that determine efficiency of maritime terminals (Bierwirth and Meisel, 2010). In QCAP, handling time of a vessel depends on the number of quay cranes assigned, and its handling time reduces as this number increases. However, the relation between the number of cranes and the handling time is not linear because of crane interference. As Blazewicz et al. (2011) suggest that moldable task scheduling problem (MTSP) provides a basis for capturing these features, we model QCAP as a MTSP with contiguous assignments of machines.

By considering various features of QCAP, we study an extended mathematical formulation of MTSP. Differently from the literature, this formulation tracks specific task to machine assignment information. We develop a time-indexed formulation which can only solve small instances to optimality because of the computationally expensive constraints. For this reason, we hybridize that model by extending with an additional set of variables and constraints. Such a hybridization allows us not only to get rid of expensive constraints that address unique properties of initial formulation, but also to implement LBBD by dividing the problem into master and subproblems. In the master problem, we use two sets of variables together and ensure that the number of machines used at any time to be less than or equal to the total number of available machines. In the subproblem, we only consider a very limited subset of variables.

That is, given the optimal starting times and total machine assignments for each task determined by the master problem, we specify proper task to machine assignments by ensuring non-overlapping of tasks on each machine as well as contiguous assignments of machines. If the subproblem is infeasible, we add cuts to the master problem until a feasible solution is found. Moreover, we develop strong Benders cuts by utilizing the fact that the subproblem is very easy to solve hence it can be iteratively solved in a short time.

The resulting solution procedure provides a generic framework that allows applying different types of properties straightforwardly. Therefore, we extend LBBD with contiguous assignments of QCs, QC availability, and uniform QCs. Then, we present a realistic terminal scenario in which a combination of these properties are simultaneously considered.

Computational experiments show that the proposed approach provides a very tight linear programming relaxation for the base case and all of the extensions. This is the result of the novel decomposition strategy that considers all variables within the master problem, thus we are able to estimate feasible space of the global problem accurately. We show that we solve almost all instances of MTSP with contiguous assignments within the first iteration of the LBBD, while approximately 20% of the instances with uniform machines and unavailable time windows require one or more cuts. Experiments also indicate the robustness of the proposed approach since it can cope with a different objective and speed up functions.

# Chapter 3

# ALTERNATIVE SOLUTION APPROACHES TO QCAP

One limitation of this exact method proposed in Chapter 2 is the excessive computational requirements for solving large problem instances, since it is based on a time-indexed formulation. In Table 2.6, we showed that LBBD-MTSP can only solve 6 out of 10 instances with $|K|$=10 and $|J|$=35,40. While these instance sizes are sufficient for the majority of real world container terminals, some mega terminals handle larger number of vessels in a planning horizon of a day. In the following, we further test LBBD-MTSP with larger instances to see the limits of the method.

Table 3.1: Results of LBBD-MTSP for larger instances

| $\|K\|$ | $\|J\|$ | (i) | (iii) |
|---|---|---|---|
| 10 | 20 | 10 | 46.3 |
| | 25 | 10 | 317.6 |
| | 30 | 9 | 1377.7 |
| | 35 | 6 | 2148.7 |
| | 40 | 6 | 2626.9 |
| | 45 | 1 | 3106.5 |
| | 50 | 0 | - |

Table 3.1 shows that the performance of LBBD-MTSP in terms of the number of instances that can be solved within the time limit of an hour drastically decreases after $|J|$=40 instances. More specifically, it can only solve one instance with $|J|$=45 in 3106.5 seconds and cannot solve any of the instances with $|J|$=50.

To solve the problem instances with larger than 40 vessels, in this chapter, we

propose methods to derive lower and upper bounds. For an upper bound, we introduce a constraint programming model that finds near optimal solutions in a short time. Since linear programming relaxations of time indexed formulations are known to be very tight, we developed a column generation procedure in which pricing problem can be solved in polynomial time.

## *3.1 Constraint Programming*

In the following, we present a heuristic method for QCAP, namely, a constraint programming (CP) model, to obtain near optimal solutions within a short time. We previously modeled QCAP as a MTSP with contiguous assignment of machines. To be able to implement such a model with CP, we first introduce a novel CP model for MTSP with specific assignments. Then, we extend the resulting model with contiguous assignments of QCs and other features of the problem.

### *3.1.1 CP Model for MTSP with Specific Assignments*

All sets and parameters for mixed-integer programming formulation are also valid for the proposed CP model. In addition, we introduce the following variables.

- $\overline{Y}_i$ an interval variable that represents the interval in which task $i$ is processed by any machine/set of machines, $domain(\overline{Y}_i) = \{1, \ldots, |T|\}$,
- $\overline{X}_{ij}$ an optional interval variable that represents the interval in which task $i$ is processed by machine $j$, $domain(\overline{X}_{ij}) = \{1, \ldots, |T|\}$,
- $\overline{Z}_i$ an integer variable that defines total number of machines assigned to task $i$, $domain(\overline{Z}_i) = \{1, \ldots, |K|/2\}$.

We then present a novel CP model for MTSP with specific assignments as follows.

minimize $\sum_i w_i end(\overline{Y}_i)$

subject to:

$$alternative(\overline{Y}_i, (\overline{X}_{ij} | \forall j \in K), \overline{Z}_i) \qquad \forall i \quad (3.1)$$

$$disjunctive(\overline{X}_{ij}|\forall i \in J) \qquad \forall j \quad (3.2)$$

$$end(\overline{Y}_i) - start(\overline{Y}_i) = P_{i,\overline{Z}_i} \qquad \forall i \quad (3.3)$$

$$span(\overline{Y}_i, (\overline{X}_{ij}|\forall j \in K)] \qquad \forall i \quad (3.4)$$

The objective of the model is to minimize weighted sum of completion times. Constraint (3.1) assigns task $i$ to $\overline{Z}_i$ different machines, where $\overline{Z}_i$ is an integer decision variable that represents the number of machines assigned to task $i$. Previous CP studies that consider problems involving resource allocation with alternative resources, use *alternative* global constraint to assign each task to a single or fixed number of machines. These problems include resource allocation problems, machine scheduling problems, and multi-mode RCPSP. CP provides a very flexible modeling environment as the number of machines that is assigned to each task can even be a decision variable. By this way, we can obtain a more compact model for MTSP with specific task to machine assignments.

Constraint (3.2) ensures non-overlapping of tasks that are assigned to same machine. This constraint is equivalent to capacity constraint of MIP formuation with $Z_{ijlt}$ variables that checks capacity for each machine. So that, by solving this CP model, we obtain a schedule with a feasible task to machine assignments without requiring a subproblem, differently from the exact solution approach we introduced in Chapter 2 .

One of the key characteristics of MTSP is the controllable processing times; that is, processing time of task $i$ depends on the total number of machines assigned to this task. To implement this, we use another flexibility of CP modeling which allows the fact that indices of decision variables can be decision variables. Hence, we set the duration of $\overline{Y}_i$ interval variable as $P_{i,\overline{Z}_i}$, a parameter that indicates the processing time of task $i$ when processed by $\overline{Z}_i$ total machines.

Recall that MTSP with specific assignments has two unique properties from the perspective of mathematical programming:

- **Property 1:** A task that is processed by more than one machine must start on all of these assigned machines at the same time, i.e., if $j_1 \neq j_2$ and $t_1 \neq t_2$, then $Z_{ij_1lt_1}$ and $Z_{ij_2lt_2}$ cannot take the value of 1 simultaneously.

- **Property 2:** A task that is processed by different machines must be assigned to a same number of total machines, i.e., if $j_1 \neq j_2$ and $l_1 \neq l_2$, then $Z_{ij_1l_1t}$ and $Z_{ij_2l_2t}$ cannot take the value of 1 simultaneously.

Property 2 is simply enforced by constraint (3.1) because task $i$ can only be assigned to one and only one total number of machines, $\overline{Z}_i$. We model the Property 1 of MTSP with specific assignments formulation with a modeling trick, by relating interval variables $\overline{Y}_i$, $\overline{X}_{ij}$ with each other via constraint (3.4). By this constraint, we set the start time of task $i$ to the minimum of start times of this task on assigned machines and set its ending time to the maximum of corresponding ending times.

**Proposition 9.** *Constraints (3.3) and (3.4) together imply Property 1.*

*Proof.* Assume that task $i$ is assigned to two total machines ($l = 2$), which are $j_1$ and $j_2$, and this task starts on these machines on different times $t_1 < t_2$, respectively. Machine $j_1$ starts processing task $i$ at $t_1$ and ends at $t_1 + P_{i,2}$. Similarly, machine $j_2$ starts processing task at time $t_2$ and ends at $t_2 + P_{i,2}$. By constraint (3.4), the start and end times of task $i$ (activity $i$) is $t_1$ and $t_2 + P_{i,2}$, respectively. Constraint (3.3) requires the length of task to be equal to $P_{i,2}$, however $t_2 + P_{i,2} - t_1$ is strictly larger than $P_{i,2}$ as $t_1 < t_2$. This contradicts our initial assumption. □

### *3.1.2 Implemention to QCAP*

In previous chapter, we represent QCAP as an extension of MTSP with contiguous assignments of machines. For this reason, in this section, we extend the CP model with contiguous assignments.

**Contiguous Assignments:**

For LBBD-MTSP, we developed a compact contiguous assignments constraint that is based on $Z_{ijlt}$ variables. We cannot directly implement this constraint in CP because

decision variable $\overline{X}_{ij}$ keeps the task to machine assignment information but not the total number of assigned machines. We can still develop a more compact constraint compared to enumerating all non-contiguous assignments.

In this approach, for each task, we check all $(j_1, j_2)$ combinations of machine set where $j_1 < j_2$ and ensure that if $j_1$ and $j_2$ is assigned to same task, then all in-between machines $(j_1 < j_3 < j_2)$ must be assigned to that task as well.

$$presence(X_{ij_1}) - presence(X_{ij_2}) \leq 1 + \sum_{j_3 \in \{j_1+1,\ldots,j_2-1\}} presence(X_{ij_3})/(j_2 - j_1 - 1)$$
$$\forall i, \forall j_1, j_2 : (j_1 \leq |K| - 2) \;\&\; (j_2 \geq j_1 + 2) \quad (3.5)$$

**Uniform Machines:**

Here, we assume that machines are uniform rather than identical. That is, machines can have different processing rates but these rates do not depend on tasks. Differently from the MIP formulation, it is not possible to modify definition of $\overline{Z}_i$ index which stands for the $l$ index of $Z_{ijlt}$ of MIP formulation because of the role of this variable within constraint (3.1). We define new integer variable $\overline{Q}_i$ which tracks the total processing rate of machines assigned to task $i$. Then, for each task $i$, we add constraint $\sum_j S_j.(presence(\overline{X}_{ij})) = \overline{Q}_i$ to CP model. We also need to modify constraint (3.3) as $end(\overline{Y}_i) - start(\overline{Y}_i) = P_{i,\overline{Q}_i}$ to be able to define processing time of task $i$ by total processing rate of assigned machines.

**Unavailable Time Windows:**

We simply remove the intervals in which machine $j$ is unavailable from the domains of $\overline{X}_{ij}$ variables. Let $domain(Y_i) = (-\infty, \infty)$ and machine $j$ is unavailable within time interval $[mb_j, me_j]$. Then, domain of $\overline{X}_{ij}$ is $(-\infty, mb_j) \cup (me_j, \infty)$.

**Machine Eligibility:**

Let $M_i$ is a set of indices of eligible machines as we defined earlier. We modify constraint (3.1) as $alternative(\overline{Y}_i, (\overline{X}_{ij} | \forall j \in M_i), \overline{Z}_i)$ where $M_i \subset K$, set of all machines.

### *3.1.3 Computational Experiments*

We perform a set of computational experiments to evaluate the performance of CP approach. We consider the same data set of 200 total instances that we used in previous chapter, with $P \in [20, 60]$ and speed-up function $f_l^2$. CP is known to be a tool for finding near optimal solutions, and it does not have any advanced strategy to terminate its search with a good solution until all nodes are searched. Hence, it is often terminated with a time limit in practice. We test the performance of the proposed method by using IBM CP Optimizer for three different time limits: 10 seconds, 1 minute, and 5 minutes. Moreover, we execute each instance 5 times with different random seeds since the search phase of CP Optimizer involves randomness.

Table 3.2 shows the average percentage deviation of the results of CP model from the optimal value for10 instances and 5 trials for each $|K|$ - $|J|$ combination. For a fair comparison, we disregarded the 13 instances in which LBBD-MTSP fails to solve to optimality within an hour. The table also presents the average relative improvement of the results of 1 minute and 5 minutes tests to the results of 10 seconds tests.

According to Table 3.2, the average percentage deviation for all 187 instances are 2.99%, 1.70%, and 1.41% for time limits of 10 seconds, a minute, and 5 minutes, respectively. We do not present standard deviation of the method as it is very low in all instances, except the 10 seconds tests. Strictly speaking, for a time limit of a minute, we observe an average relative standard deviation of 0.35% over all instances with a maximum value of 1.23%.

The solution quality of the proposed CP model decreases as the total number of machines increases. For any time limit, average percentage deviation from the optimal values is approximately five times larger for $|K| = 10$ instances than those of $|K| = 4$ instances. This is not an unexpected result because the total number of task to machine assignments significantly grows with $|K|$: $(\sum_{k=0}^{|K|/2} \binom{|K|}{k})^{|J|}$. However, it is still less than 3% for instances with 10 machines if we set time limit as a minute.

Interestingly, CP model fails to find optimal values except two instances, even though it can find solutions which are very near to optimal. Additional tests with

Table 3.2: Results of CP model for MTSP with specific assignments

| $\|K\|$ | $\|J\|$ | 10 sec. | 60 sec. | improv. | 300 sec. | improv. | MIP time |
|---|---|---|---|---|---|---|---|
| 4 | 20 | 0.85 | 0.52 | 38.58 | 0.49 | 42.12 | 5.98 |
| | 25 | 0.86 | 0.50 | 42.52 | 0.39 | 55.28 | 14.85 |
| | 30 | 0.88 | 0.50 | 43.10 | 0.47 | 46.02 | 21.82 |
| | 35 | 0.87 | 0.53 | 38.66 | 0.47 | 46.54 | 53.73 |
| | 40 | 1.00 | 0.53 | 47.10 | 0.44 | 56.26 | 131.73 |
| | | 0.91 | 0.52 | 42.98 | 0.45 | 50.68 | 45.62 |
| 6 | 20 | 1.14 | 0.67 | 41.05 | 0.58 | 49.15 | 10.74 |
| | 25 | 1.75 | 1.31 | 25.38 | 0.95 | 45.86 | 63.18 |
| | 30 | 2.45 | 1.57 | 35.87 | 1.21 | 50.64 | 144.26 |
| | 35 | 2.64 | 1.74 | 34.08 | 1.32 | 50.14 | 395.52 |
| | 40 | 2.26 | 1.30 | 42.56 | 0.97 | 57.04 | 929.56 |
| | | 2.23 | 1.41 | 36.71 | 1.07 | 52.08 | 308.65 |
| 8 | 20 | 1.39 | 0.90 | 35.19 | 0.69 | 50.40 | 22.45 |
| | 25 | 2.44 | 1.69 | 30.53 | 1.03 | 57.71 | 102.81 |
| | 30 | 3.35 | 2.22 | 33.77 | 2.02 | 39.53 | 365.72 |
| | 35 | 3.86 | 2.29 | 40.73 | 1.91 | 50.47 | 916.84 |
| | 40 | 5.06 | 2.36 | 53.26 | 2.14 | 57.66 | 1693.40 |
| | | 3.70 | 2.08 | 43.75 | 1.76 | 52.40 | 620.24 |
| 10 | 20 | 2.51 | 1.33 | 46.97 | 1.10 | 56.03 | 46.35 |
| | 25 | 2.93 | 2.13 | 27.48 | 1.59 | 45.89 | 317.63 |
| | 30 | 4.44 | 2.61 | 41.17 | 2.10 | 52.71 | 1377.72 |
| | 35 | 5.47 | 3.02 | 44.73 | 2.51 | 54.13 | 2148.74 |
| | 40 | 7.12 | 3.39 | 52.39 | 3.17 | 55.52 | 2626.87 |
| | | 5.13 | 2.77 | 46.13 | 2.37 | 53.91 | 1303.46 |

30 minutes of time limit can only increase this number to four. Along with that, the improvement between a minute and 5 minutes of time limits are relatively low, which suggests us the time limit of a minute seems reasonable for the model.

In Table 3.2, we also present average solution time to find optimal solution with the LBBD that we proposed in previous chapter. That column suggests that it becomes advantageous over the exact method when the size of instances is getting larger because of the amount of time required to solve root node with LP. For example, in instances with 40 tasks and 8 machines, the average solution time for the exact method is 1693.40 seconds while proposed CP model obtain a solution with objective value deviates 2.36% in average within a minute.

Being able to capture specific task to machine assignments allows us to model different properties of the problem straightforwardly, as in the case of the proposed LBBD. For contiguous assignments, we observe very similar performance (not instance by instance, but in terms of average values) with the results in Table 3.2 for MTSP with specific assignments. When we consider uniform machines, in addition to contiguous assignments, we get even better performance from the CP model. For both patterns of unavailable time windows, the performance of the model slightly decreases compared to average results presented in Table 3.2. Still, the average deviation from the optimal value over all instances is less than 2%. All in all, the proposed CP model is a proper tool to generate near optimal solutions within a short time for large instances, even though it fails to find optimal solutions.

## *3.2 Column Generation*

### *3.2.1 Introduction*

In this section, we utilize the following hybrid formulation that we develop for MTSP by integrating two types of variables. Notice that this formulation is different than the master problem of LBBD. We obtain the hybrid formulation by integrating $Q_ilt$ variables into $[MTSP - CA]$. For the master problem of LBBD, we further replace capacity constraint over $Z_{ijlt}$ variables with one that only restricts total machine usage

at any time as we delay decision to specify machine assignments until the subproblem.

$$\text{minimize} \sum_{i} \sum_{l} \sum_{t} (t + p_{il}) w_i Z_{ijlt} / l$$

subject to:

$$\sum_{l} \sum_{t} Z_{ijlt} / l = 1 \qquad \forall i \quad (3.6)$$

$$\sum_{j} Z_{ijlt} = l Q_{ilt} \qquad \forall i, \forall l, \forall t \quad (3.7)$$

$$\sum_{i} \sum_{l} \sum_{h=min(1,t-p_{il}+1)}^{t} Z_{ijlh} \leq 1 \qquad \forall j, \forall t \quad (3.8)$$

$$Z_{ijlt} \in \{0,1\}, Q_{ilt} \in \{0,1\} \qquad \forall i, \forall j, \forall l, \forall t \quad (3.9)$$

As we mentioned in previous chapter, such an hybridization allows removing two computationally expensive constraints required for addressing the two unique properties associated with pure $Z_{ijlt}$ formulation, $[MTSP - CA]$. The resulting hybrid formulation is robust and flexible, i.e allows modeling and solving various problem variants straightforwardly. Moreover, hybrid formulation (3.6)-(3.9) is more unified compared to the LBBD approach because it determines feasible task to machine assignment without a subproblem (and adding any cuts).

At the same time, however, hybrid formulation has some important computational disadvantages.

- First, time-indexed formulations of scheduling problems are very large in size. Since variables are enumerated for each discrete time, the number of constraints and the number of variables can be large even for relatively small instances. As a result, the memory required to store an instance and the time required to solve just the linear programming (LP) relaxation may be prohibitively large. This situation is even worse for the hybrid formulation because it has a very expensive capacity constraint over extended variables (3.8) that ensures non-overlapping of tasks on each machine.

- Second, hybrid formulation has a symmetric structure because of $Z_{ijlt}$ variables that specify machine index. This symmetry causes a need for an evaluation of excessive number of nodes unnecessarily until optimal solution is found. In other words, the symmetry causes the solution time to be larger even though the quality of an LP relaxation at the root node is very near to optimal.

To address these issues, in this section, we propose a novel column generation (CG) approach by exploiting the advantages of the hybrid formulation to solve its LP relaxation. In addition to solving these issues, proposed CG offers the following substantial benefits. The problem we consider is hard to solve for large problem sizes. By solving its LP relaxation via CG, we obtain a very tight lower bound as we show in previous chapter. This lower bound allows us to evaluate the solution quality of the approximation methods developed for solving the large instances, accurately. Moreover, CG provides a basis for branch and price algorithm which can be utilized to solve integer problem to optimality as we will discuss in the last section of the chapter. We should note that, in this dissertation, our primary aim is to provide a quality lower bound for the large instances of the various variants of the problem.

### *3.2.2 Methodology*

Column generation (Lubbecke and Desrosiers, 2002) is a decomposition technique to solve large sized LP models. The basic idea can be described as follows: The original LP model is divided into two as restricted master problem (RMP) and subproblem(s). We start solving the RMP with a small number of variables. Subproblem (i.e. pricing problem) identifies the variable(s) with negative reduces costs by using the dual values of the RMP constraints. Then, we add these variables to the RMP. CG procedure terminates with an optimal solution to original LP model when there are no variables left with negative reduced costs. In other words, CG can be summarized as a technique that deals with only a subset of variables rather than considering all problem variables at once.

It is actually possible that we can directly decompose our formulation above into

master and subproblem(s). In that case, however, we need to add $Z_{ijlt}$ and $Q_{ilt}$ variables to the the RMP at each iteration, which requires examination of reduced costs for each of these variables. It is obvious that implying such a decomposition strategy will be computationally expensive. For this reason, almost all column generation studies in the literature consider three types of reformulations: Dantzig-Wolfe (Dantzig and Wolfe, 1960), set covering, and set partitioning. In each of them, a solution to the original problem can be represented as a single column in constraint coefficient matrix $A$.

In this study, we consider Dantzig-Wolfe (DW) reformulation to get benefit of time-indexed formulation as shown in Van der Akker et al. (2000). Any feasible solution to LP can be represented as a convex combination extreme points, and this fact provides a basis to DW reformulation (Bertsimas and Tsitsiklis, 1997). In DW, each column stands for a variable that corresponds to an extreme point of polyhedra of the original problem, and we find optimal solutions given a subset of extreme points of $X$ by convexity constraints (3.13) and (3.14).

**Original Formulation**

minimize $cx$

subject to:

$$Ax = b \tag{3.10}$$

$$x \in X \tag{3.11}$$

**Dantzig-Wolfe Reformulation**

$$\text{minimize} \sum_{i=1}^{Q} (cx_i)\lambda_i$$

subject to:

$$\sum_{i=1}^{Q} (Ax_i)\lambda_i = b \tag{3.12}$$

$$\sum_{i=1}^{Q} \lambda_i = 1 \tag{3.13}$$

$$\lambda_i \geq 0 \tag{3.14}$$

$$(x_i, i = 1, \ldots, Q \, extreme \, points \, of \, X)$$

By applying DW reformulation, we transform the polyhedra of original problem

**CG Master Problem**

$$\text{minimize} \sum_{i=1}^{Q} (cx_i)\lambda_i$$

subject to:

$$\sum_{i=1}^{Q} (Ax_i)\lambda_i = b \quad (u) \tag{3.15}$$

$$\sum_{i=1}^{Q} \lambda_i = 1 \quad (v) \tag{3.16}$$

$$\lambda_i \geq 0 \tag{3.17}$$

**CG Pricing Problem**

$$\text{minimize } (c - uA)x - v$$

subject to:

$$x \in X \tag{3.18}$$

into a new one in which $\lambda_i$ variables correspond to feasible solutions to the original problem, where $Q$ denotes the number of total feasible solutions. As can be noticed, applying DW reformulation can decrease number of constraints but at the same time it can increase the number of variables significantly. For example, $Q$ is extremely large for most of the scheduling problems. Column generation procedure emerges at this point; that is, it only considers a subset of variables and extends this subset as subproblem identifies variable(s) with negative reduced cost. Assuming $k$ to be less than $Q$, CG master and pricing problems can be constructed as follows.

CG pricing problem is based on the original problem variables while the RMP only contains $\lambda_i$ variables that represent feasible solutions (extreme points) of the original problem, a relatively small subset of all feasible solutions. The optimal values for the variables of the pricing problem stand for the coefficients of a new $\lambda_i$ variable to be added to the RMP. In other words, each extreme point of an original problem is added to the RMP as a coefficient column of a single variable. In the following, we present the DW reformulation of the hybrid formulation and the proposed column generation structure.

### 3.2.3 Implementation to Hybrid Formulation

Column generation approach for time-indexed formulation of scheduling problems is first presented by van den Akker et al. (2000) for a single machine scheduling problem. Even though this insightful paper shows that CG can cope with size disadvantage, CG is then rarely implemented to time-indexed formulations of other scheduling problems in the literature.

The reason for such a situation can be described as follows: van den Akker et al. (2000) shows that their decomposition exploits a special structure of the subproblem. That is, their pricing problem consists of the capacity constraint of the original problem, which is an interval matrix. By utilizing this observation with two other properties of the pricing problem, they show that it can be converted to an equivalent shortest path problem (SPP) that can be solved in polynomial time. Unfortunately, this structure is not observed in time-indexed formulations of many other scheduling problems. Consider RCPSP problem with a single renewable resource. If at least one task requires more than one units of resource, the resulting capacity constraint is no more an interval matrix, and we need to deal with a NP-hard pricing problem.

Note that this is also the case for the MTSP formulation without specific assignments (over $Q_{ilt}$ variables), since its capacity constraint consists of coefficients which are larger than 1. At this point, we should state that our hybridization of $Q_{ilt}$ variables with an extended $Z_{ijlt}$ variables not only creates an opportunity to exploit the same advantage with van den Akker et al. (2000) but also brings further advantages (in addition to modeling flexibility provided by $Z_{ijlt}$ variables) as we will describe in the following sections.

Accordingly, we keep assignment (3.6) and synchronization (3.7) constraints in the master problem and applied DW reformulation to enumerate extreme points of original problem. We then sent "complicating" capacity constraint (3.8) to the pricing problem whose objective function utilizes dual values from the RMP to find a variable with a negative reduced cost.

**Master Problem:**

$$\text{minimize} \sum_k \sum_i \sum_l \sum_t ((t + P_{il}) w_i Z_{ijlt}/l)\lambda_k$$

subject to:

$$\sum_k \sum_j \sum_l \sum_t (Z_{ijlt}/l)\lambda_k = 1 \qquad \forall i \quad (3.19)$$

$$\sum_k \sum_j Z_{ijlt}\lambda_k = \sum_k lQ_{ilt}\lambda_k \qquad \forall i, \forall l, \forall t \quad (3.20)$$

$$\sum_k \lambda_k = 1 \qquad (3.21)$$

$$\lambda_k \geq 0 \qquad \forall k \quad (3.22)$$

**Pricing Problem**

After solving the restricted master problem to optimality, we price $\lambda$ variables to be added to RMP, given the dual values associated with constraints (3.19), (3.20), and (3.21). Then, pricing problem is to find a non-basic variable with the most negative reduced cost.

Reduced cost of $\lambda$ variable :

$C_{\lambda_k} = \sum_i \sum_j \sum_l \sum_t (w_i(t + P_{il}) - D1_i - lD2_{ilt})Z_{ijlt}/l + \sum_i \sum_j \sum_l \sum_t D2_{ilt}Q_{ilt} - D3$

By constraint (3.20), we already set $lQ_{ilt}$ to be equal to $\sum_j Z_{ijlt}$ for each $i$, $j$, and $l$, hence parts of the expression associated with dual variable D2 cancels out. The resulting pricing problem is presented below.

$$\text{minimize} \sum_i \sum_j \sum_l \sum_t ((t + p_{il}) w_i - D1_i) Z_{ijlt}/l - D3$$

subject to:

$$\sum_{l}\sum_{l}\sum_{h=min(1,t-p_{il}+1)}^{t} Z_{ijlh} \leq 1 \qquad \forall j, \forall t \quad (3.23)$$

$$0 \leq Z_{ijlt} \leq 1 \qquad \forall i, \forall j, \forall l, \forall t \quad (3.24)$$

**Proposition 10.** *Pricing problem (3.23)-(3.24) is totally unimodular, hence variables are guaranteed to take integer values.*

*Proof.* Constraint matrix $A$ consists of constraint (3.23) which is an interval matrix. Interval matrices are totally unimodular and each extreme point of a totally unimodular matrix is an integer vector (Schiver, 1986). □

By implementing the Dantzig Wolfe reformulation of the problem, we transform the polyhedra of time-indexed formulation into another one with $\lambda_k$ variables. Together with the unimodularity of the pricing problem, each extreme point then corresponds to a pseudo-schedule for all machines, and coeefficients of an $\lambda_k$ variable are stored in terms of the values of $Z_{ijlt}$ variables. In these pseudo-schedules tasks can be executed more than once or even can be left unexecuted as long as we ensure that each machine can process at most one task at any time.

By solving the pricing problem, we obtain such a pseudo-schedule for each machine as the capacity constraint (3.23) prevents the non-overlapping of tasks. The pricing problem can be represented as a network $G_j(N, E)$ for each machine $j$ with the following transformation. Let discrete time points to be the nodes of the network $G$ with same labels, i.e., $N = \{1, 2, \ldots, |T|\}$. After that, we need to set edge weights between nodes to represent execution of tasks as follows:

- $((t + p_{il})w_i - D1_i)/l$ from node $t$ to $t + p_{il}$, which represents the execution of task $i$ starting at time $t$, on machine $j$ by using $l$ total machines.

- 0 from node $t$ to $t + 1$, which represents the idleness of machine $j$ from time $t$ to $t + 1$.

In Figure 3.1, we present the corresponding network for an instance with one task and two machines. Here, processing time of a task when processed with a single machine is two unit times, and in the figure, it is represented by the edges which are depicted over the nodes. Similarly, processing time of a task when processed by two machines is four, and it is represented by the edges depicted under the nodes in Figure 3.1. Moreover, each edge from node $t$ to $t+1$ represents the idleness of a machine. Note that the depicted network is only for one of the machines, but it is actually the same for the other machine.

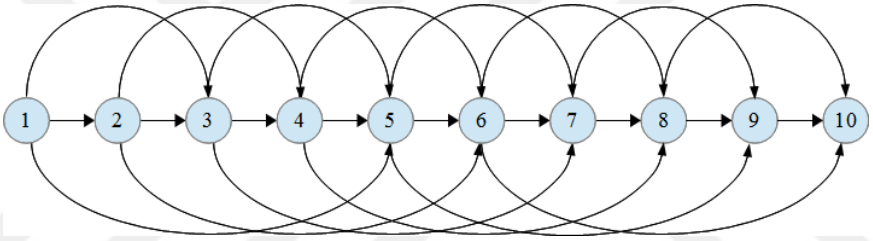



Figure 3.1: Representation of network $G$ for an example instance

In Figure 3.2, considering the example instance, we present a pseudo-schedule for a machine, on which the task starts at $t = 1$ with two machines and finishes at $t = 3$, then the machine stays idle for a time period, the task again starts at $t = 4$ with a single machine and finishes at $t = 8$, and finally the task starts with one machine at $t = 8$. One can observe that, in this pseudo-schedule, the task is executed three times on this machine while these executions did not overlap in time. More specifically, assuming that we add this extreme point to master problem as variable $\lambda_k$, the coefficients of this variable would be consisting of $Z_{1121} = 1$, $Z_{1114} = 1$, $Z_{1128} = 1$ for machine 1, $Z_{1221} = 1$, $Z_{1214} = 1$, $Z_{1228} = 1$ for machine 2, and all other $Z_{ijlt}$ take the value of 0.

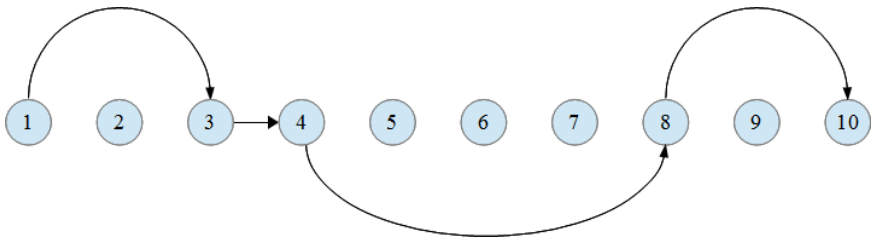



Figure 3.2: A pseudo-schedule for an example instance

By finding a shortest path on network $G_j$, we obtain a pseudo-schedule that minimizes reduced cost while ensuring non-overlapping of tasks on machine $j$. Let optimal value of SPP on network $j$ be $C_{SPP_j}$. Then, the reduced cost of $\lambda_k$ can be calculated as $C_{\lambda_k} = \sum_j C_{SPP_j} - D3$.

We continue adding new $\lambda$ variables to the restricted master problem until the pricing problem cannot identify a column with a negative cost. In the following, we present some important observations and enhancements on the CG algorithm we developed.

### *3.2.4 Observations and Enhancements*

**Observation 1** *Synchronization constraint (3.7) is redundant in the LP solution as we do not assume integrality of variables.*

Recall that we use constraint (3.7) together with the integrality of $Q_{ilt}$ variables to replace the functions of two expensive constraints that address unique properties of the time-indexed formulation of MTSP with specific assignments. Without integrality assumption, constraint (3.7) became redundant because its function is already enforced with constraint (3.6): In the RMP, constraint (3.7) suggests that $\sum_k \sum_j \lambda_k Z_{ijlt}/l$ can take any value between 0 and 1 for each $i$, $l$, and $t$. Let $H = \sum_k \sum_j \lambda_k Z_{ijlt}/l$. However, constraint (3.6) already states that $\sum_l \sum_t H = 1$, which means that $H$ is between 0 and 1. Note that the dual value associated with constraint (3.7) is not a part of the objective function of the pricing problem as it cancels out. Still, this constraint must be added to the RMP on demand based on the branching decisions

if we want to find an optimal solution to the original problem, rather than its LP relaxation. We will revisit this later.

**Observation 2** *Algorithm complexity is lower than the theoretical number of* $O(|J||L||T|)$. The network we consider is topologically sorted from 1 to $|T|$, i.e., there is no edge from any node $t$ to $t^{'}$, $t^{'} < t$ because edges are representing the processing of tasks outgoing from starting time. Therefore, we can solve SPP with an efficient dynamic programming algorithm that visits nodes by a topological sort and checking each outgoing edge. Since there are $|T|$ nodes and there can be $|J||L|$ possible edges from each node the theoretical complexity of such an algorithm is $O(|J||L||T|)$. However, we observe that we can have at most one edge from $t$ to $t+k, k>0$, even though there are possibly more nodes based on $p_{il}$ values of the input. As a result, the number of outgoing edges from each node is limited by $p_{max}$, which is an input of the instance and often is way lesser than $|J|.|L|$.

**Observation 3** *In the pricing problem, it is enough to consider edges with non-positive weights. Therefore, we can reduce the computational effort required to the solve pricing problem by removing the edges with positive weights.*

**Proposition 11.** *Edges with positive weights are not traversed in any optimal solution.*

*Proof.* Assume that an edge from node $j$ to $j+k$ $(k>0)$ is traversed in an optimal solution, and this edge has a positive weight $w_{j,j+k} > 0$. Let corresponding optimal value be $z^*$. We can replace this positive edge with a set of edges $(j, j+1), \ldots, (j+k-1, j+k)$ with zero weight that represent machine idleness . Resulting solution has an objective value of $z^* - w_{j,j+k}$ which strictly lesser than $z^*$. This contradicts our initial assumption. □

To observe the impact of this observation, we compare the maximum possible number of edges $|T|.p_{max}$, the total number of edges with and without positive edges.

For total of 10 instances with 30 tasks-10 machines with $|T| = 160, p_{max} = 60$, corresponding average numbers are 10800, 6715, and 4590, respectively. In other words, the average number of total edges are 36 percent less than $|T|.p_{max}$, and we can further reduce this number with 37 percent with this observation.

**Observation 4** *It is enough to solve SPP for only for one machine.*
Assuming we deal with the original problem (3.6)-(3.9), edge weights that represent the execution of tasks in SPP are $((t + P_{il}).w_i - D1_i)/l$. As it does not have a component with $j$ index, it is enough to solve SPP for only one machine. In other words, we have the same pseudo-schedule for each machine to minimize $C_{\lambda_k}$. We will revisit this observation for each problem extension.

**Observation 5** *CG can be modified for solving MTSP with contiguous assignments, machine unavailability, precedence relations, uniform machines and machine eligibility.*
The key here is to be able to make modifications without destroying the structure that allows us to solve pricing problem as a SPP. We will explain these modifications in the following section.

### *3.2.5 Implementation to Problem Extensions*

We list the modifications for the RMP and the pricing problem for each extension below.

*Machine Unavailability*

**Master Problem:** No modifications.
**Pricing Problem:** Modify the right-hand-side of constraint (3.23) as $F_{jt}$, a parameter which takes the value of 1 if machine $j$ is available at time $t$, 0 otherwise. The pricing problem is still unimodular as we are not changing $A$ matrix. Inthe corresponding network problem, we need to delete all arcs which are passing by unavailable

nodes, i.e. $t \leq f_{jt}$ *and* $t + p_{il} \geq f_{jt}$ if $f_{jt} = 0$. We can still solve only one SPP for machines without unavailable periods and one for each machine with a different unavailability pattern.

*Uniform Machines*

**Master Problem:** Multiply the left hand side of the constraints and the objective function with $s_j$, processing rate of machine $j$.

**Pricing Problem:** Edge weights are modified with respect to the modification of the master problem. We should solve one SPP for each processing rate.

*Precedence Relationships*

**Master Problem:** Add the following precedence constraint where $PREC$ is a set of precedence relationships between tasks:

$\sum_k \sum_j \sum_l \sum_t \lambda_k.t.Z_{i'jlt}/l \geq \sum_k \sum_j \sum_l \sum_t \lambda_k.(t + p_{il}).Z_{ijlt}/l \quad \forall (i, i') \in PREC$

**Pricing Problem:** Edge weights are modified with respect to precedence constraint of the master problem. Since weights are still independent of $j$ index, it is enough to solve only one SPP.

*Contiguous Assignments*

**Master Problem:** Add the following contiguous assignments constraint that we developed previously:

$\sum_k \lambda_k(Z_{ijlt} + Z_{ij'lt}) \leq 1 \quad \forall i, \forall j, j', \forall t, \forall l \in (2, ..., (j' - j)) : j' > j + 1$

**Pricing Problem:** Let dual variable associated with new constraint to be $\Omega_{ijj'lt}$. Then, reduced cost for $\lambda$ variables ($C_{\lambda_k}$) would be,

$\sum_i \sum_j \sum_l \sum_t((t+p_{il}).w_i - D1_i).Z_{ijlt}/l - \sum_i \sum_{j,j'} \sum_l \sum_t \Omega_{ijj'lt}.Z_{ijlt} - \sum_i \sum_{j,j'} \sum_l \sum_t \Omega_{ijj'lt}.Z_{ij'lt} - D3.$

Since we need to check for a pair of machines at the same time, one might think the special structure of the pricing problem would be destroyed. However, after a careful investigation, we are able to separate pricing problem for each machine as follows:

First, let us investigate the constraint for 4 total machines. In that case, we should consider contiguous assignments constraint for machines (1,3) and (2,4) for $l$=2 and (1,4) for $l$=3. Also let $A_j = \sum_i \sum_l \sum_t ((t + p_{il})w_i - D1_i).Z_{i1lt}/l$. Total reduced cost would be,

$C_{\lambda_k} = \sum_j A_j - \sum_i \sum_t \Omega_{i132t} Z_{i12t} - \sum_i \sum_t \Omega_{i132t} Z_{i32t} - \sum_i \sum_t \Omega_{i143t} Z_{i13t} - \sum_i \sum_t \Omega_{i143t} Z_{i43t} - \sum_i \sum_t \Omega_{i242t} Z_{i22t} - \sum_i \sum_t \Omega_{i242t} Z_{i42t}$.

By considering $Z_{ijlt}$ variables that we used in the pricing problem, this expression can be separated for each machine $j$ such that,

$\overline{C_1} = A_1 - \sum_i \sum_t \Omega_{i132t} Z_{i12t} - \sum_i \sum_t \Omega_{i143t} Z_{i13t}$

$\overline{C_2} = A_2 - \sum_i \sum_t \Omega_{i242t} Z_{i22t}$

$\overline{C_3} = A_3 - \sum_i \sum_t \Omega_{i132t} Z_{i32t}$

$\overline{C_4} = A_4 - \sum_i \sum_t \Omega_{i242t} Z_{i42t} - \sum_i \sum_t \Omega_{i143t} Z_{i43t}$.

Obviously, this can be generalized for any $|K|$ by simply utilizing the constraint that we developed for identifying non-contiguous assignments to be restricted:

$\overline{C_j} = A_j - \sum_i \sum_{j'=j+1}^{|K|} \sum_{l=2}^{j-j'} \sum_t \Omega_{ijj'lt} Z_{ijlt} - \sum_i \sum_{j'=j-1}^{1} \sum_{l=2}^{j'-j} \sum_t \Omega_{ij'jlt} Z_{ijlt}$

As a result, we obtain $C_{\lambda_k} = \sum_j \overline{C_j} - D3$, so that $\overline{C_j}$ can be calculated by solving SPP for each machine $j$ with weights calculated previously.

*Machine Eligibility*

**Master Problem:** Add the following machine eligibility constraint $\sum_k \sum_l \sum_t \lambda_k Z_{ijlt} = 0 \ \ \forall i, \forall j \notin M_i$ , where $M_i$ is the set of eligible machines for task $i$.

**Pricing Problem:** This constraint adds a component with $j$ index to weights. For this reason, we need to solve SPP for each machine separately. In real world, non-eligibility constraints are often based on the types of tasks and/or machines and therefore we can expect lesser number of SPPs to be solved. For example, let there are two types tasks and two types of machines such that the first type of tasks cannot be processed by the second type of machines. In that case, it should be enough to solve one SPP for first and one for the second type of machines.

Table 3.3: Comparison of the performance of CPLEX and the proposed CG

| $\|K\|$ | $\|J\|$ | CPLEX time | CG time |
|---|---|---|---|
| 10 | 20 | 1.6 | 1.2 |
| | 25 | 4.5 | 1.7 |
| | 30 | 8.0 | 2.1 |
| | 35 | 10.8 | 3.0 |
| | 40 | 16.5 | 5.5 |
| | 60 | 63.9 | 14.0 |
| | 80 | out of memory | 32.9 |
| | 100 | out of memory | 57.1 |

### *3.2.6 Computational Experiments*

We perform a set of computational experiments for instances with 10 machines. Accordingly, we compare the performance of the proposed column generation procedure, which is coded in C++, with those of the latest version of CPLEX (12.7.1) to the solve LP relaxation of (3.6)-(3.9). Table 3.3 shows that proposed CG strictly outperforms CPLEX in terms of the solution time and the maximum solvable instance size. For example, CPLEX fails to solve instances with 80 tasks with an out of memory error after approximately 120 seconds. On the other hand, CG solves all instances with 80 tasks within 32.9 seconds in average.

As we obtained lower bound values for large instances via proposed CG, we can evaluate the solution quality of CP model for those instances. In Table 3.2, we showed that the average percentage deviation from the optimal solution for $|K|$=10, $|J|$=40 is 3.39% within the time limit of a minute. For $|K|$=10 and $|J|$=60,80,100, the average percentage deviation of CP model from the lower bound value are observed as 3.71%, 4.09%, and 4.45%, respectively.

These results suggests us that it might be worth investigating the performance branch and price algorithm to solve integer problem: in addition to solve the size disadvantage of time-indexed models, branch and price algorithms can find the optimal by investigating lesser number of nodes compared to branch and bound (Savelsbergh, 1997).

### *3.2.7 Discussion on Branch and Price Algorithm*

When pricing problem identifies that there is no variable with a negative reduced cost, column generation algorithm terminates with an optimal solution to LP problem. Most of the time, however, this solution consists of variables with non-integer solutions. At this point, branch and price method emerges by integrating column generation with branch and bound (B&B) to solve integer problems. Accordingly, in branch and price (B&P), we do not need to solve the LP at each node of the search tree from the scratch and simply use the subset of columns which are previously added to the RMP based on branching decision. In this section, we discuss the implementation of B&P based on the proposed CG.

While it looks like B&P is a non-trivial combination of CG and B&B, there are some important differences with B&B. First of all, conventional integer programming branching strategies may not be effective because fixing variables can destroy the structure of pricing problem (Barnhart et al. 1998). Moreover, the decision of which branching over the original versus the master problem variables brings other issues as discussed in Derossiers and Lubbecke (2010).

For the proposed CG, we should branch on the original problem variables ($Z_{ijlt}$) rather than the master problem variables ($\lambda_k$). If we branch on the master problem variables, e.g. set $\lambda_k$=1 (and $\lambda_k$=0), we only state the fact that pseudo-schedule $k$ is the solution, and we cannot branch any more since constraint (3.21) states sum of all $\lambda$ vars to be equal to 1.

Assume that we select $Z_{ijlt}$ variable to branch, which has a non-integer value. Accordingly, we set $Z_{ijlt}$=1 and $Z_{ijlt}$=0 in two separate branches. Let us keep column

set $C$ which contains information of pseudo-schedules such that $\{k, i, j, l, t, 1\}$, which means we traverse node $t$ to $t + P_{il}$ on machine $j$ in column $k$. Then, for the former branch ($Z_{ijlt}$=1), we remove each master problem variable $k$ if $C$ does not contain $\{k, i, j, l, t, 1\}$ and add constraint $\sum_k \lambda_k.Z_{ijlt}$=1 to the RMP. For the latter branch, we similarly remove all $\lambda_k$ if from the RMP if $\{k, i, j, l, t, 1\}$ is an element of $C$. We should also add $Q_{ilt}$=1 along with $Z_{ijlt}$=1 but we cannot add $Q_{ilt}$=0 with $Z_{ijlt}$=0 as $Q_{ilt}$ can still take 1 because of $Z_{ij'lt}$ variables, $j^{'} \neq j$. On the other hand, branching on the original variables requires constant transition between the original and the RMP variables, so causes implementation issues.

It is worth emphasizing that we need to add synchronization constraint (3.7) on demand as we claimed in Observation 1. That is, we also generate rows, i.e., add constraints to the master problem based on branching decisions. Such a procedure distinguishes our algorithm from the related literature.

Chapter 4

# RECLAIMER SCHEDULING IN DRY BULK TERMINALS

## *4.1 Introduction*

As the volume of maritime transportation increases, workload at seaside terminals grows equally. This makes operating seaside terminals efficiently and less costly more important. Even though bulk loads constitute more than 80% of all loads carried by maritime transportation (UNCTAD, 2017), operational problems of bulk terminals are rarely studied in the literature, until recently.

In this chapter, we study the reclaimer scheduling problem (RSP) which is observed in many dry bulk terminals. In these terminals, cargo (i.e., load) is stored as a stockpile and is handled by reclaimer machines. A reclaimer is a huge equipment that reclaims (i.e., collects, recovers) stockpiles from the stockyard to be loaded to mooring vessels (Figure 4.1-a). Reclaimed cargo is then transported to the vessel via a conveyor belt system. In the stockyard of a dry bulk terminal, there are multiple stocking pads where stockpiles of different lengths are stacked. In between each two stocking pads, there is a rail track on which one or more reclaimers are mounted (Figure 4.1-b). Reclaimers are identical, and a reclaimer can only process one task at a time. Reclaiming of a stockpile is assumed to be non-preemptive.

The RSP is to find a schedule for reclaiming stockpiles which are stacked in the stockyard by using a set of reclaimers with respect to the various problem constraints. We define a task as a one type of load to be a sent to a predefined vessel. After finishing the reclaiming of a stockpile, a reclaimer travels along the rail track to go to the area where its next scheduled task is stacked. This movement requires a specific travel time based on the distance between these two stockpiles. Moreover, a reclaimer can rotate its boom to serve the pads that are adjacent to the rail track on which it is


(a) A reclaimer (b) A stockyard

Figure 4.1: Equipment and configuration of stockyard in dry bulk terminals (downloaded from http://gettyimages.co.uk in May 2018)

mounted, but it cannot reclaim the stockpiles that are located on other pads.

If there are more than one reclaimer on a rail track, then they cannot pass each other. Even though non-crossing constraint is widely studied in the context of quay crane scheduling problem (QCSP, Unsal and Oguz, 2013), there are some notable differences between non-crossing constraints of these two problems. In QCSP, all quay cranes operate on a single rail track while reclaimers are distributed across multiple rail tracks. When loading/unloading a container, a quay crane does not move along the rail track. On the other hand, a reclaimer performs bench cuts to reclaim a stockpile. That is, it travels along the rail track from one end of the stockpile to the other hand -55 times on average- to reclaim a stockpile completely (long travel bench cuts, Angelelli et al., 2016). This fact brings another restriction on the movements of reclaimers located on the same rail track. Consider two such reclaimers and two stockpiles which are located in different sides of the rail track. If these two stockpiles are overlapping in time and x-axis, they cannot be reclaimed simultaneously by these two reclaimers since it will cause them to cross each other.

We can view RSP as a variant of parallel machine scheduling problem (PMSP) in which non-crossing of machines, a specific pattern for machine eligibility (i.e., pro-

cessing set restrictions), and sequence dependent travel times are taken into account. Clearly, QCSP and RSP are related problems as both are the variants of PMSP with non-crossing constraint. We can determine the computational complexity of RSP by utilizing this relation: If we assume that there is only one pad, one rail track, two reclaimers, and all stockpiles has a unit length, we obtain an instance of QCAP with two QCs. The resulting problem is proved to be strongly NP-hard for makespan minimization objective (Zhu and Lim, 2006).

In addition, advanced reclaimers can also perform the stacking of stockpiles. For such machines, stacking rate is significantly higher than reclaiming rate. However, if stacking operation is ignored then we might generate infeasible reclaimer schedules, as stockpiles are often stacked and reclaimed dynamically during the planning horizon. For this purpose, in this chapter, we also extend the RSP with stacking operations by assuming that each stockpile to be stacked by a reclaimer before its reclaiming starts.

The literature on RSP is limited. Hu and Yao (2012) propose a mixed integer programming model and a genetic algorithm for RSP in which there is only one reclaimer in each rail track. They assume that a reclaimer can only reclaim stockpiles located in pads adjacent to its rail track and take travel times of reclaimers into account. The similar problem setting is also studied by Wang et al. (2018). They develop an approximation algorithm with 2 worst case ratio. They also show that this ratio can be reduced to 3/2 by assuming a task can be executed by two reclaimers simultaneously.

Angelelli et al. (2016) study an abstract scheduling problem inspired by the reclaiming operations of a coal export terminal. In their problem, there are two reclaimers mounted on a rail track which is located in-between two pads. Therefore, they consider the fact that these two reclaimers cannot cross each other at any time, namely non-crossing constraint. For this problem, they present some complexity results as well as an approximation algorithm. It should be noted that their abstract problem has one critical simplification: to reclaim this stockpile completely, a reclaimer must travel from the one end of stockpile to the other end once with predefined

constant speed. However, as we mentioned previously, this simplification is inconsistent with the real world terminal operations where reclaiming is mostly performed by long travel bench cuts. Kalinowski et al. (2017) extend the abstract problem of Angelleli et al. (2016) by relaxing their assumption that requires all stockpiles to be stacked at the beginning of planning period. That is, they study the dynamic version of the problem.

We develop a tight lower bound and a near optimal method that can solve RSP within a short time. We organize the rest of the chapter as follows. In Section 4.2, we present the methods we develop in this study. First, we discuss the mixed integer programming formulation of RSP. As it can only solve small sized instances, we develop a heuristic method for the problem. In Section 4.2.2 we present an arc-time-indexed formulation for the relaxation of the problem to generate a tight lower bound. By using that lower bound, we can measure the performance of the constraint programming model that we propose in Section 4.2.3 accurately. Lastly, in Section 4.3, we design a set of computational experiments to test the proposed method for different stockyard configurations and discuss their results.

## *4.2 Methodology*

### *4.2.1 Mixed Integer Programming Formulation*

For RSP, we initially formulate a mixed integer programming (MIP) model. It is based on assignment variable $X_{ik}$, which takes a value of 1 if stockpile $i$ is assigned to reclaimer $k$, 0 otherwise. Differently from our modeling efforts in previous chapters, decision variables of the model is not time-indexed because it would be computationally intractable, thus inadequate to model non-crossing constraint and sequence dependent setup times with that type of variables. At the same time, however, the performance of the MIP formulations of parallel machine scheduling problem with assignment variables are shown to be ineffective compared to the performance of time-indexed formulations as long as the length of planning horizon is not large (Unlu and Mason, 2010). This can be mainly attributed to its weak linear programming

relaxation, and the resulting RSP model is not an exception. We do not present this MIP model here, since it neither can solve problem instances of reasonable sizes nor will be utilized within the proposed lower bound and heuristic methods.

In the following, we present sets and parameters that we use in the lower bound and constraint programming models.

$J$ set of reclaiming tasks (stockpiles), $i = \{1, \ldots, |J|\}$,
$C$ set of reclaimers, $k = \{1, \ldots, |C|\}$,
$T$ set of time indices, $t = \{1, \ldots, |T|\}$,
$A$ set of pads in stockyard, $a = \{1, \ldots, |A|\}$,
$W$ set of rail tracks, $w = \{1, \ldots, |W|\}$,
$U$ set of stock positions located on a single pad, $u = \{1, \ldots, |U|\}$,
$Q_k$ index of a rail track on which reclaimer $k$ is mounted, $Q_k \in W$,
$P_i$ reclaiming time of stockpile $i$,
$K_i$ number of stock positions required to accommodate stockpile $i$,
$S_{i_1,i_2}$ time required to travel between midpoints of stockpiles $i_1$ and $i_2$,
$Pad_i$ index of pad in which of stockpile $i$ is stacked, $Pad_i \in A$,
$Pos_i$ index of stock position in which leftmost of stockpile $i$ is positioned,
$Weight_i$ weight associated with stockpile $i$.

For the extended problem of RSP with stacking operations, we also introduce the following sets and parameters.

$J'$ extended set of tasks, $i = \{1, \ldots, |J|, |J|+1, \ldots 2|J|\}$, where tasks indexed from $|J|+1$ to $2|J|$ stand for the stacking operations of stockpiles,
$\delta$ a constant that defines the estimated ratio of the reclaiming time to the stacking time of a stockpile,
$P_i'$ stacking and reclaiming times of stockpiles, i.e.,

$$P_i' = \begin{cases} P_i, & \text{if } i = \{1, \ldots, |J|\}. \\ \lceil P_i/\delta \rceil, & \text{if } i = \{|J|+1, \ldots 2|J|\}. \end{cases}$$

$Weight_i^{'}$ weight of each task, i.e.,

$$Weight_i^{'} = \begin{cases} Weight_i, & \text{if } i = \{1, \ldots, |J|\}. \\ 0, & \text{if } i = \{|J|+1, \ldots 2|J|\}. \end{cases}$$

$\gamma$ set of precedence relationships between stacking and reclaiming tasks, $\gamma = \{(i_1, i_2) | i_1 \in \{|J|+1, \ldots, 2|J|\}, i_2 \in \{1, \ldots, |J|\} : i_1 = i_2 + |J|\}$.

### *4.2.2 Relaxed MIP for Deriving a Lower Bound*

We develop an arc-time-indexed MIP model that provides a quality lower bound for RSP. This model helps us to measure performance of heuristic methods such as constraint programming accurately.

This type of model for scheduling problems is first presented for single machine by Tanaka et al. (2009) and is extended to parallel machines by Pessoa et al. (2010). In the proposed model that we extend their formulation with machine index, we simply relax non-crossing constraint but consider all other features of the problem. Arc-time-indexed formulations integrate respective advantages of network and time-indexed scheduling problem formulations: the former keeps a sequence of tasks for each machine while the latter provides a tight LP relaxation as it does not need "big M" to model disjunctions.

Accordingly, we introduce a binary variable $Y_{i_1 i_2 k t}$ that takes a value of 1 if machine $k$ starts processing task $i_2$ at time $t$, immediately after processing task $i_1$, 0 otherwise. In this formulation, for each machine, we determine a route over nodes. Here, a node represents a task-time tuple $(i, t)$, rather than a task as in pure network formulation. We also define a dummy task for modeling issues, indexed by 0, with null processing time and introduce a new set $T^0$, set of tasks extended with dummy task 0 ($T^0 = T \cup 0$). This relaxed MIP model for RSP is presented below.

$[RSP - LB]$:

$$\text{minimize} \sum_{i_1 \in J^0} \sum_{i_2 \in J} \sum_{k} \sum_{t} (t + P_{i2}) Weight_{i_2} Y_{i_1 i_2 k t}$$

subject to:

$$\sum_{i_1 \in J^0} \sum_{k} \sum_{t} Y_{i_1 i_2 k t} = 1 \qquad \forall i_2 \in J \quad (4.1)$$

$$\sum_{i_1 \in J^0} \sum_{t} Y_{i_1 i_2 k t} = 0 \qquad \forall i_2 \in J, \forall k : (Pad_i < Q_k) || (Pad_i > Q_k + 1) \quad (4.2)$$

$$\sum_{i_2 \in J} \sum_{t} Y_{0 i_2 k t} = 1 \qquad \forall k \quad (4.3)$$

$$\sum_{i_1 \in J} \sum_{t} Y_{i_1 0 k t} = 1 \qquad \forall k \quad (4.4)$$

$$\sum_{i_2 \in J^0} Y_{i_2 i_1 k t} - \sum_{i_2 \in J^0} Y_{i_1, i_2, k, (t + p_i + S_{i_1 i_2})} = 0 \qquad \forall i_1 \in J, \forall k, \forall t \quad (4.5)$$

$$Y_{i_1 i_2 k t} \in \{0, 1\} \qquad i_1, i_2 \in J^0, \forall k \in C, \forall t \in T \quad (4.6)$$

The objective of the model is to minimize total weighted completion of reclaiming stockpiles. The processing order of tasks on a reclaimer can be viewed as a route of a vehicle from the perspective of vehicle routing problem. By constraint (4.1), we ensure that each node is visited once. In other words, each task must be processed on one reclaimer by starting at one time point. Constraint (4.2) prevents the ineligible stockpile to reclaimer assignments.

Constraints (4.3) and (4.4) state that the route for each machine starts and ends at dummy node 0. Flow balance constraint (4.5) ensures that if a vehicle (reclaimer) visits a node, then it must leave that node as well, excluding the dummy task. Lastly, constraint (4.6) defines the domains of $Y_{i_1 i_2 k t}$ variables.

In this formulation, sub-tours are prevented by flow balance constraint (4.5), which also keeps account of start times, differently from the network formulation of scheduling problems (Cesaret et al., 2010).

**Proposition 12.** *The proposed formulation prevents sub-tours.*

*Proof.* Assume that there is a sub-tour of $i_1 - i_2 - i_1$ for a machine in a feasible solution to the formulation, where $i_1$ and $i_2$ are not dummy tasks, and their processing times are non-zero. In that case, if $i_1$ starts at time $t_1$, then $t_2$ starts at $t + P_{i_1} + S_{i_1 i_2}$ by flow balance constraint (4.5). In that sub-tour, we visit $i_1$ immediately after $i_2$. Then, $i_1$ starts at time $t_1 + P_{i_1} + S_{i_1 i_2} + P_{i_2} + S_{i_2 i_1}$. However, $P_{i_2} + S_{i_2 i_1} > 0$, and it contradicts our initial assumption, since a task can only start at one and only one time point by constraint (4.1). □

*Relaxed MIP Formulation for Deriving a Lower Bound for RSP with Stacking Operation*

We first integrate the additional sets and parameters into formulation $[RSP - LB]$ to model RSP with stacking operations. More specifically, we replace the set of tasks $(J)$ with $J^{'}$ as well as parameters $P_i$ and $Weight_i$ with $P_i^{'}$ and $Weight_i^{'}$, respectively. Furthermore, we introduce parameter $\delta$ and set of precedences $\gamma$. In this variant of the problem, a stockpile must be stacked in the stockyard before its reclaiming starts. Therefore, we add the following precedence constraint into the formulation.

$$\sum_{i_3} \sum_{k} \sum_{s \in \{1,\ldots,max\{1,t-p_{i_1}\}\}} Y_{i_3 i_1 k s} - \sum_{i_3} \sum_{k} \sum_{s \in \{1,\ldots,t\}} Y_{i_3 i_2 k s} \geq 0 \qquad \forall (i_1, i_2) \in \gamma, \forall t \quad (4.7)$$

### *4.2.3 Constraint Programming Model*

We introduce the CP model and associated decision variables as follows.

$Y_i$ An interval variable that represents the reclaiming of stockpile $i$. Its domain is $[1, |T| - P_i]$

$X_{ik}$ An optional interval variable that represents the reclaiming of stockpile $i$ by reclaimer $k$. This variable is active in the model if and only if reclaimer $k$ is assigned to stockpile $i$. Otherwise its domain will be $\emptyset$.

$Seq_{i_1 i_2 k_1 k_2}$ A sequence variable that is used for non-crossing constraint, and it determines relative order of interval variables $X_{i_1 k_1}$ and $X_{i_2 k_2}$. Its domain consists of the permutation of the order of these interval variables.

$[RSP - CP]$:

$$\text{minimize} \sum_{i_1 \in J} Weight_i end(Y_i)$$

subject to:

$$Alternative(Y_i, (X_{ik} | \forall k : (pad_i \leq Q_k) || (pad_i > Q_k + 1))) \quad \forall i \quad (4.8)$$

$$Disjunctive((X_{ik} | \forall i), S) \quad \forall k \quad (4.9)$$

$$Disjunctive(Seq_{i_1 i_2 k_1 k_2})$$

$$\forall i_1, i_2, \forall k_1, k_2 : Q_{k_1} = Q_{k_2}, k_1 < k_2, pad_{i_1} - pad_{i_2} \in \{-1, 0, 1\}, pos_{i_1} + K_{i_1} > pos_{i_2} \quad (4.10)$$

Constraint (4.8) assigns each stockpile to one of the eligible reclaimers and links related $Y_i$ with $X_{ik}$ variables. Constraint (4.9) ensures non-overlapping of stockpiles that are assigned to same machine by also considering sequence dependent travel times. We imply non-crossing constraint by (4.10). This constraint also respects the fact that reclaimers needs to perform bench cuts to reclaim stockpiles, by preventing the possible interference.

*Constraint Programming Model for RSP with Stacking Operation*

Similar to the modifications we made on the relaxed MIP model for incorporating stacking operations, we replace $P_i$, $Weight_i$, and $J$ with $P_i^{'}$, $Weight_i^{'}$, and $J^{'}$, respectively, and introduce sets $\delta$ and $\gamma$. Then, we simply add the following constraints to the formulation $[RSP - CP]$.

$$presence(X_{i_1k}) = presence(X_{i_1+|J|,k}) \qquad \forall k, \forall i_1 \in \{1, \ldots, |J|\} \quad (4.11)$$

$$endBeforeStart(X_{i_1k_1}, X_{i_2k_2}) \qquad \forall i_1, i_2, \forall k_1, k_2 \quad (4.12)$$

### *4.3 Computational Experiments*

We design a set of computational experiments for assessing the performance of the proposed CP method. We generate four different stockyard configurations with varying numbers of stocking pads ($|A|$), rail tracks ($|W|$), and reclaimers ($|C|$) . These configurations are as follows.

- **Configuration 1:** $|A| = 3$, $|W| = 2$, $|C| = 3$. Two reclaimers on rail track 1 and a single reclaimer on rail track 2.
- **Configuration 2:** $|A| = 3$, $|W| = 2$, $|C| = 4$. Two reclaimers on each rail track.
- **Configuration 3:** $|A| = 4$, $|W| = 3$, $|C| = 4$. Two reclaimers on rail track 2 and a single reclaimer on rail tracks 1 and 3.
- **Configuration 4:** $|A| = 4$, $|W| = 3$, $|C| = 6$. Two reclaimers on each rail track.

We assume that each stocking pad consists of $|U|$ =60 33.33 meters-long stocking positions. That is, the total length of a pad is 2000 meters. We generate tasks such that their lengths ($K_i$) are uniformly distributed between 2 and 10 units of stocking positions, and reclaiming times ($P_i$) are proportional to the their length, i.e. $P_i = 6K_i$. In reality, travel times are very small compared to reclaiming times. For this reason, we calculate the travel time between stockpiles $i_1$ and $i_2$ as their distance on x-axis from midpoints divided by 10 (in unit times). As a result, travel times are bounded within 1 and 6 unit times.

We generate 10 instances for each configuration and compare results of relaxed MIP model $[RSP - LB]$ with those of CP model $[RSP - CP]$. As CP method has random components on its search phase, we perform with five different trials for each instance. Recall that we obtain a relaxed MIP model for deriving a lower bound by relaxing non-crossing constraint of RSP. By removing non-crossing constraint from the CP model, we get an equivalent problem to the relaxed MIP model. Hence, we can also compare the performance of CP method without including the deviation caused by non-crossing constraint. We set a time limit of a minute and an hour for each CP trial and LB model, respectively.

Table 4.1: Results for RSP

| Configuration | Relaxed MIP | $CP^1$ | | | $CP^2$ | | |
|---|---|---|---|---|---|---|---|
| | GAP-LP | min | avg. | max | min | avg. | max |
| 1 | 0.05 | 0.01 | 0.36 | 0.95 | 0.03 | 0.62 | 1.33 |
| 2 | 0.06 | 0.01 | 0.37 | 0.75 | 0.31 | 0.93 | 1.80 |
| 3 | 0.05 | 0.11 | 0.25 | 0.45 | 0.17 | 0.56 | 1.09 |
| 4 | 0.09 | 0.32 | 0.72 | 1.17 | 0.90 | 1.27 | 1.96 |
| avg. | 0.06 | 0.11 | 0.43 | 0.83 | 0.35 | 0.85 | 1.54 |

Table 4.1 presents results of computational experiments for RSP. For configurations 1 and 2, the average number of tasks per instance is 28.8, and is 39.5 for configurations 3 and 4.

Out of 40 total instances, LB model solves 37 instances to optimality within time limit of an hour. For other three instances -one instance each from configurations 1, 3 and 4-, we use the lower bound value displayed by the CPLEX after an hour. The good performance of the LB model is mainly due to the very tight LP relaxation of the model: the average deviation of LP relaxations from the optimal values are approximately 0.06%, and LP relaxation is also optimal for more than half of the instances.

We present results for the computational experiments of CP method for RSP in

$CP^2$ column, in terms of the percentage deviation from the corresponding lower bound values. Average deviation is observed as 0.85%, and it increases for the configurations with larger number of reclaimers per track. In all 40 instances, the worst deviation value is less than just 2%. Moreover, it is robust: the maximum relative standard deviation between trials is observed as 0.48%, hence we do not present these values in Table 4.1.

In column $CP^1$ we also present the results for CP method without non-crossing constraint, which is equivalent to the relaxed MIP model. We observe 0.43% deviation from the lower bound in average, by excluding the effect of non-crossing constraint on the performance measure. This indicates that the CP model for RSP is able to cope with the challenging non-crossing constraint effectively.

To sum up, the proposed CP method can cope with the problem with various input configurations effectively, by providing near optimal results in time limit of a minute.

**Computational Experiments for RSP with Stacking Operation:**

This extension of RSP requires two tasks for each stockpile, a stacking task and a reclaiming task. In addition to that, precedence constraint (4.7) is computationally expensive since it adds large numbers of constraints and non-zero coefficients to the formulation. As the formulation is harder to solve, we perform experiments with a set of smaller instances to generate lower bound values. Specifically, we assume that the total length of each pad to be 1000 meters, rather than 2000. By this way, we aim to

Table 4.2: Results for RSP with stacking operation

| Configuration | $CP^1$ | | | $CP^2$ | | |
|---|---|---|---|---|---|---|
| | min | avg. | max | min | avg. | max |
| 1 | 0.03 | 0.29 | 0.65 | 0.01 | 0.50 | 1.45 |
| 2 | 0.05 | 0.44 | 0.88 | 0.11 | 0.79 | 1.58 |

reduce the number of stockpiles by half while the number of total tasks in an instance stays the same with respect to the instances of RSP without stacking operation.

We present the results in Table 4.2. It shows that the CP model can also solve this extension of the problem effectively with a similar performance to the results presented in Table 4.1.

Chapter 5

# INTEGRATED DRY BULK TERMINAL OPERATIONS

## *5.1 Introduction*

Similar to container terminals, bulk terminals also consist of seaside and yard area. They both need to perform common activities such as loading and unloading of cargoes from (to) vessels, transporting cargoes between the storage and the berth area, and temporarily stacking and handling them in the storage area. However, they vastly differ in terms of the type of cargo (load). In contrast to container terminals in which all cargo are transported in a standardized container, in bulk terminals means of storage and handling of cargo vary for different types of materials. Another important difference is that bulk terminals act as either import or export terminal. In this study, we consider export dry bulk terminals in which loads are handled by stacker-reclaimer machines and are stored as stockpiles. Our methodology is an integrated approach for the following operational problems in a dry bulk terminal (DBT):

- Berth Allocation Problem (BAP) to determine berthing time and berthing position for each vessel,
- Yard Allocation Problem (YAP) to determine stocking time and position of each stockpile in stockyard,
- Reclaimer Scheduling Problem (RSP) to determine a schedule for each reclaimer to reclaim stockpiles that are located at different positions of stockyard.

This study is motivated by the operational problems of a DBT exporting coal and the general structure of such a terminal can be described as follows: Coal arrives to

the terminal by train and is stored in the stockyard as stockpile(s). After mooring of empty vessels to the berths, related stockpiles are reclaimed from the stockyard by reclaimers and are transferred via conveyor belt systems to the quay area where empty vessels are loaded by ship loaders. Berth allocation in DBTs is often affected by tidal time windows; that is, the departure of the loaded vessels is constrained by high tide periods due to their weights.

As can be noticed easily, operational problems of DBTs are strongly interrelated. For example, the schedule of the reclaimers will directly affect the loading of the vessels, hence the time that a vessel can leave the berth. Similarly, decisions on the positioning of stockpiles at the stockyard affect reclaimer scheduling since they generate non-eligibility of some stockpile-reclaimer assignments.

In this study, by considering different characteristics of DBTs, we develop a novel logic-based Benders decomposition algorithm to solve berth allocation, reclaimer scheduling, and yard allocation problems simultaneously. We decompose these problems into two as master problem and subproblem, after identifying some key relationships between the problems. Then, we exploit the advantages of two different programming techniques, mixed integer programming (MIP) and constraint programming (CP), to cope with the master problem and the subproblem, respectively. To the best of our knowledge, this is the first exact method that integrates these major operational problems observed in DBTs.

The rest of the chapter is organized as follows. We first define the integrated dry bulk terminal (IDBT) problem and give a brief literature review of the bulk terminal operations. After introducing a monolithic formulation in which we solve IDBT problem within a single mathematical model in Section 5.2, we propose a decomposition structure for logic-based Benders algorithm. Section 5.3 presents the MIP master problem, the CP subproblem, cuts, and an example that explains execution of the proposed algorithm step by step. We also study IDBT problem with continuous berth structure in Section 5.4. In Section 5.5, we design and perform a set of computational experiments and present their results. Lastly, we conclude the chapter with discussion

and future work.

### *5.1.1 Problem Definition*

In the IDBT problem, there are $|V|$ vessels with $|J|$ loading tasks, $|M|$ berths, $|H|$ high tide periods, $|C|$ reclaimers mounted on $|W|$ rail tracks, and $|U|$ stocking positions located on each of $|A|$ pads. The structure of a DBT for an instance with $|M|$=3, $|C|$=3, $|W|$=2, $|U|$=13 and $|A|$=3 is given in Figure 5.1. We define a task as a stockpile of one type of coal to be loaded to a single vessel.

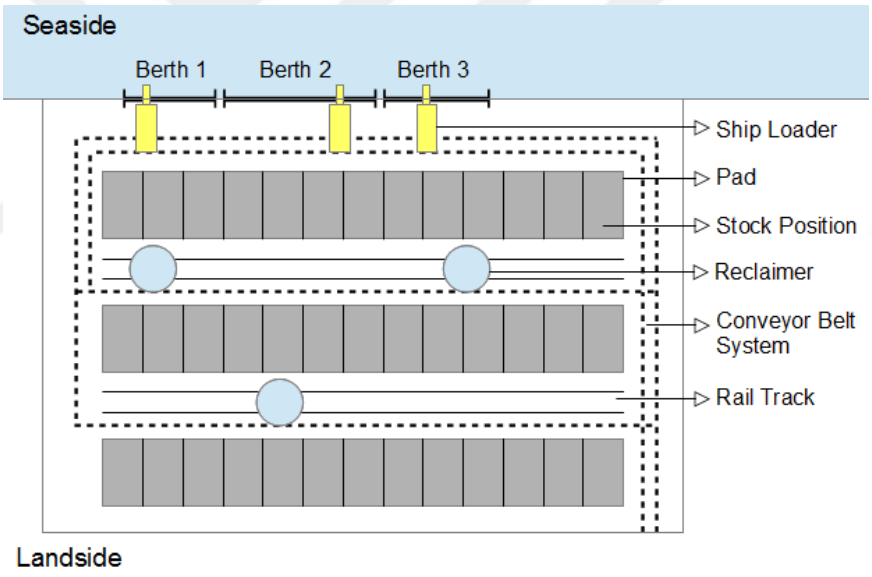



Figure 5.1: General structure of a dry bulk terminal

Then IDBT problem is to generate berth, reclaimer and yard schedules that minimize total weighted departure times of vessels with respect to various problem constraints. These schedules determine,

- for each vessel: where and when to moor, and when to depart,
- for each stockpile: where to stock, when to start reclaiming, and by which reclaimer it is reclaimed.

In the following, we state the assumptions and constraints of IDBT problem. In the literature, three different berth layouts are considered: discrete (Imai et al., 1997), continuous (Park and Kim, 2003), and hybrid (Nishimura et al., 2001). We assume a discrete berth layout in which each berth is a discrete resource with a single vessel capacity. In other words, vessels that are assigned to the same berth cannot overlap in time. Berths may have different lengths, therefore each vessel should moor to one of the berths that suits to its length. Vessels arrive dynamically during the planning period, and they cannot moor before their arrival times. Most important characteristic of berth allocation problem in dry bulk terminals is the common existence of tidal time windows. Because of tides, a vessel can only depart during a high tide period even if its loading is completed in low tide period. Since we consider an export terminal, vessels arrive empty, so they can moor even during a low tide period. Once a vessel is moored, its loading operations can be performed without such limitations.

A vessel can have more than one stockpile to be loaded. However, all loading tasks of a vessel must be performed consecutively by a single reclaimer because of the technological restrictions of in-terminal transportation system (connection of conveyor belts and ship loaders). Reclaimers are identical, and a reclaimer can process one task at a time. Reclaiming of a stockpile is assumed to be non-preemptive. A reclaimer can only reclaim stockpiles located on the pads adjacent to the rail track on which it is mounted. To start reclaiming a stockpile, the related vessel needs to be moored to a berth first. Similarly, a vessel needs to wait at berth until all related stockpiles are reclaimed.

Moreover, on each rail track there can be more than one reclaimer and these reclaimers cannot pass each other. Even though non-crossing constraint is widely studied in the context of quay crane scheduling problem (Unsal and Oguz, 2013), there is an important difference between non-crossing constraints of these two problems. When loading/unloading a container, a quay crane does not move along the rail track. In a dry bulk terminal, a reclaimer travels along the rail track from one

end of the stockpile to the other hand 55 times on average to reclaim a stockpile completely (Angelelli et al., 2016). This fact brings another restriction to movements of reclaimers which are located on the same rail track. Consider two such reclaimers and two stockpiles which are located at different sides of the rail track. If these two stockpiles are overlapping in time and x-axis, they cannot be reclaimed simultaneously by these two reclaimers, since this will require reclaimers to cross each other.

Compared to reclaiming, stacking operation has a higher handling capacity. Therefore, we disregard the scheduling of stacking operation and assume that stockpiles are stacked immediately once they arrived to the terminal. After a reclaimer completes loading of last stockpile of a vessel, there should be an additional time of transporting the very last part of the stockpile to the vessel with respect to the distance between the reclaimer and the berth that vessel is moored. We assume that this amount of time depends on reclaimer rather than the exact location of the related stockpile over the pad, because it depends on the conveyor belt configuration (design) between the berth and the reclaimer that reclaims stockpiles of this vessel.

In the stockyard, each pad consists of multiple discrete stock positions. A task (stockpile) is assumed to use all width of the pad however it may use more than one stock position due to its amount. In such a case it is stored as a rectangular pile that uses multiple adjacent stock positions. Stockpiles are assumed to be stacked at the moment that they arrived to the terminal and stored in the same stocking position until completely reclaimed. Furthermore, each stocking position can be occupied by one stockpile at a time.

### *5.1.2 Literature Review*

In the last two decades operational problems of container terminals are intensively studied and significant contributions have been made. For detailed reviews of the literature on container terminals we refer readers to Bierwirth and Meisel (2012) and Steenken et al. (2004). On the other hand, operational problems in bulk terminals have received very little attention, until recently.

The importance of effective management of bulk terminals is stated by Boland and Savelsbergh (2012). They also introduce the operational problems observed in dry bulk (coal) terminal of Port of Newcastle, Australia in details. In general, the properties of berth allocation operation in dry bulk terminals are similar to those of container terminals. For bulk terminals, Barros et al. (2011) study discrete berth layout by considering stock level constraints for minerals. Umang et al. (2013) assume a hybrid berth layout where a vessel can occupy more than one berth section; however, a berth section cannot be occupied by more than one vessel at any given time. They assume that each section has a fixed handling capacity and total loading/unloading time of a vessel depends on the occupied berth sections. Continuous berth layout in dry bulk terminals is studied by Ernst et al. (2017).

It is important to note that the inefficiencies of quayside operations caused by tides led researchers to develop berth allocation strategies that take tides into account (Du et al., 2015, Xu et al., 2012, Lalla-Ruiz et al., 2016). Recently, Ernst et al. (2017) present a new approach to model the effects of tidal time windows in export dry bulk terminals. They assume that vessels may berth and can be handled during a low tide. However, once loaded they may only depart during a high tide. In this study, we also follow this recent approach to model the effects of tidal periods. In export dry bulk terminals, vessels often demand one type of product, which is stacked as a single stockpile. For this reason each vessel is often loaded by a single ship loader (Boland et al., 2012). Therefore, in this study, we assume a discrete berth layout where each berth is served with a single ship loader.

Operational problems at stockyard area of dry bulk terminals widely differ from those of container terminals because of the absence of containers. These problems can be classified as reclaimer scheduling and yard allocation. For the former, Hu and Yao (2012) develop a mixed integer programming model by assuming only one reclaimer at each rail track which can only reclaim stockpiles located in pads adjacent to its rail track. Angelelli et al. (2016) study an abstract scheduling problem inspired by reclaiming operations of a coal export terminal. They assume that there are

two reclaimers mounted on a rail track which is located in-between two pads. They consider the fact that these two reclaimers cannot cross each other at any time, namely non-crossing constraint. For this problem, they present some complexity results as well as an approximation algorithm. Their abstract problem has one strong assumption: a reclaimer must travel from one end of stockpile to another end once with predefined constant speed to be able to reclaim this stockpile completely. In our integrated model, we allow that there can be two reclaimers on a rail track, so we take non-crossing constraint into account. We also assume that a reclaimer repeatedly travels back and forth to reclaim a stockpile as in reality.

In dry bulk terminals, loads are stocked in yard space as stockpiles. Stockpiles are stored on pads, which are lengthy storage areas located at both sides of rail tracks. A stockpile uses entire width of a pad but only a portion of its length. One might observe similarities of yard allocation problem with strip packing problem (Martello et al., 2003). However, dynamic structure of YAP largely differentiates it from strip packing problem. That is, many stockpiles arrive and leave stockyard during the planning period and one storage position in the stockyard can be used by many different stockpiles over time.

As in the case of Boland et al. (2012), operational problems observed in maritime terminals are often interrelated. Therefore, solving these problems sequentially by ignoring their relations often leads to plans with poor overall quality (Bierwirth and Meisel, 2010). Robenek et al. (2014) extend the work of Umang et al. (2013) integrating yard allocation and berth allocation operations in a bulk terminal. They assume a hybrid berth layout and discrete yard locations. In their study, yard locations have dedicated cargo types and a yard location can be assigned to only one cargo type. Then they assign vessels to yard locations to determine variable handling times comprised of carrying loads from stockyard to berth area. Also a yard location can be allocated to at most one vessel at any time. They develop a column generation scheme and a branch and price algorithm to minimize total completion times of vessels. In addition to an abstract reclaimer scheduling problem, Angelelli et al. (2016)

also present an extension that considers yard allocation and reclaimer scheduling simultaneously. In that model they also decide stocking positions of stockpiles that directly affect scheduling of reclaimers. Furthermore, Menezes et al. (2017) study an integrated problem that consider stockyard allocations and conveyor belt routes simultaneously. They solve medium sized problem instances to optimality via branch and price algorithm.

We present an exact solution method for the integrated problem of dry bulk terminal operations, consisting of berth allocation, reclaimer scheduling and yard allocation problems. We study this integrated problem by considering many different properties of individual problems, such as tidal periods, multiple reclaimers on a single rail track, non-crossing constraint, travel times of stockpiles between quay and yard side, and multiple rail tracks and pads that cause non-eligibility of some reclaimer-stockpile assignments. It is worth mentioning that a similar level of details is also considered by Boland et al. (2012) for an integrated problem of dry bulk terminal operations. However, they emphasize that problem components are very complex to use mathematical programming, so they develop constructive heuristics to solve the integrated problem. In the following section, we propose a logic-based Benders decomposition algorithm to solve IDBT problem to optimality.

## *5.2 Monolithic Model*

Integrated problem of dry bulk terminals can be modeled and solved within a single mixed mixed-integer programming formulation. In such a formulation, decisions regarding to allocations of vessels to berths, scheduling of reclaimers and assignments of stockpiles within the stockyard is made simultaneously.

In this chapter we initially develop a monolithic MIP formulation which is presented in Appendix C. As expected, this formulation comes up short in terms of computational performance because of the excessive number of variables and constraints. Therefore, we decompose the problem into parts which are easier to solve by utilizing the monolithic formulation.

## *5.3 Logic-Based Benders Decomposition*

Logic-based Benders decomposition (LBBD) is a generalization of Benders decomposition method (Benders, 1962) which relies on inferring cuts via inference dual rather than linear programming duality. By this way, it can be implemented to various combinatorial optimization problems in which subproblem cannot be modeled as a linear program. In this study, master problem is modeled as a mixed integer programming model (optimization) while a feasibility model for subproblem is constructed by constraint programming (CP). We refer readers to the related sections of Chapter 1 for more information on LBBD and CP methods.

As we stated before, IDBT consists of three operational problems. Even though decisions to be made for these individual problems are all interrelated, we identify two strong relationships among them. Schedules of reclaimers directly affect the departure times of vessels, since all associated stockpiles must be reclaimed by reclaimers. Also, reclaiming operation for a stockpile can only start after the associated vessel moors. Similarly, reclaimer scheduling and yard allocation are strictly related. Stockpiles that are assigned to a specific reclaimer must be located on pads adjacent to the rail track that the reclaimer mounted on. Otherwise, they cannot be reclaimed by this reclaimer. Moreover, non-crossing of reclaimers that are located on the same rail track cannot be achieved without knowing stocking positions of stockpiles.

In this study, we develop a novel LBBD algorithm to solve IDBT problem (LBBD-IDBT). We propose the following decomposition structure for LBBD-IDBT: we separate berth allocation and yard allocation problems into master and subproblem, respectively. Then we decompose reclaimer scheduling problem into two: non-overlapping of reclaiming operations that are performed by the same reclaimer is ensured in master problem while non-crossing and other constraints that interrelate reclaimer scheduling decisions with those of yard allocation are placed in subproblem. In this study, master problem is modeled as an MIP model, while a feasibility model for subproblem is constructed by CP. By solving master problem to optimality, we get:

($i$) berthing times of vessels, and berthing positions,

($ii$) departure times of vessels by considering tidal time windows,

($iii$) vessel to reclaimer assignments,

($iv$) reclaimer schedules, that is start time of reclaiming stockpiles.

We send ($i$), ($ii$), and ($iii$) of the solution to the subproblem, and it is solved to check whether there is a feasible yard allocation and a reclaimer schedule given these information from master problem. In other words, by using fixed mooring and departure times of vessels and vessel to reclaimer assignments which are determined by the master problem, subproblem tries to find a feasible solution in which:

- each stockpile is stored in stockyard until completely reclaimed,
- stockpiles are non-overlapping in both time and space (i.e. at any time a stock position can contain parts of at most one stockpile),
- each stockpile is reclaimed by the assigned reclaimer (by ($iii$)), therefore a stockpile is located on pads adjacent to the rail track of this reclaimer,
- reclaimers located on the same rail track do not pass each other at any time,
- reclaiming of stockpiles that are associated with a specific vessel starts after its mooring and ends before its departure by also taking the travel time required between berth and reclaimer into account.

If subproblem turns out to be feasible, then algorithm terminates with an optimal solution to the original problem. Otherwise, cuts are derived from the infeasibility and are added to the master problem for the next iteration.

In LBBD, there is no standard scheme for generating cuts; hence, we should devise cuts for each specific problem. A valid Benders cut needs to remove the infeasibility that is detected by subproblem without eliminating any optimal solution to the original problem (Hooker, 2007). The simplest Benders cut is "nogood", which states that we cannot obtain a feasible solution for original problem unless at least one variable

of the current solution is changed to a different value (Coban and Hooker, 2013). Nogood cuts are often ineffective, since it only cuts very small portion of the feasible space. Thus, we develop stronger cuts that remove larger portions.

One key property of LBBD-IDBT is not to send $(iv)$ to the subproblem, even though the master problem is able to determine exact start and end times for reclaiming operation of each stockpile. By this way, we can change reclaiming order of stockpiles associated with the same vessel without violating mooring and departure times of this vessel to avoid violation of non-crossing constraint. With this property, the algorithm is able to cope with non-crossing constraint easily, since the subproblem also determines stocking positions of stockpiles simultaneously. If $(iv)$ is sent to the subproblem, we observe that the algorithm may require many additional cuts because of infeasibility caused by the non-crossing constraint.

### *5.3.1 Master Problem*

In this section, we introduce sets, parameters and decision variables of the master problem. The same set of parameters is also valid for the subproblem. After presenting MIP model and describing its constraints, we propose two enhancements over the model that improve the performance.

**Sets and Parameters:**

$V$ set of vessels, $j = \{1, \ldots, |V|\}$,

$M$ set of berths, $m = \{1, \ldots, |M|\}$,

$J$ set of tasks (stockpiles), $i = \{1, \ldots, |J|\}$,

$C$ set of reclaimers, $k = \{1, \ldots, |C|\}$,

$T$ set of time indices, $t = \{1, \ldots, |T|\}$,

$H$ set of high tide periods, $r = \{1, \ldots, |H|\}$,

$A$ set of pads in stockyard, $a = \{1, \ldots, |A|\}$,

$U$ set of stock positions located on a single pad, $u = \{1, \ldots, |U|\}$,

$N$ set of all stock positions, $l = \{1, \ldots, |U|, \ldots, |A||U|\}$,

$D_l$ index of pad on which stock position $l$ is located, $D_l \in A$,

$\pi_l$ x-axis value of stock position $l$, $\pi_l \in U$,

$W$ set of rail tracks, $w = \{1, \ldots, |W|\}$,

$R_i'$ arrival time of stockpile $i$ to stockyard,

$Q_k$ index of a rail track in which reclaimer $k$ is mounted on, $Q_k \in W$,

$P_i$ amount of time required to reclaim stockpile $i$ from the stockyard,

$K_i$ number of stock positions required to accommodate stockpile $i$,

$L_j$ length of vessel $j$,

$G_m$ length of berth $m$,

$R_j$ arrival time of vessel $j$,

$B_r$ start time of high tide period $r$,

$E_r$ end time of high tide period $r$,

$Weight_j$ weight associated with vessel $j$,

$Distance_{mk}$ distance between berth $m$ and reclaimer $k$ (in time units),

$VesselOfTask_i$ index of a vessel that is associated with stockpile $i$,

$TotalLoadOfVessel_j$ amount of time required to reclaim all tasks of vessel $j$,

$AssignedTasksToVessel_j$ set of indices of stockpiles to be loaded to vessel $j$.

**Decision Variables:**

$S_j$ mooring time of vessel $j$,

$C_j$ departure time of vessel $j$,

$X_{mjj'}$ 1 if vessel $j'$ is moored to berth $m$, after vessel $j$; 0 otherwise,

$SR_i$ start time of reclaiming stockpile $i$,

$CR_i$ completion time of reclaiming stockpile $i$,

$XR_{kii'}$ 1 if stockpile $i'$ is reclaimed by reclaimer $k$, after stockpile $i$; 0 otherwise,

$Y_{mj}$ 1 if vessel $j$ is assigned to berth $m$; 0 otherwise,

$Z_{jr}$ 1 if vessel $j$ leaves at high tide period $r$; 0 otherwise,

$\Omega_{jk}$ 1 if reclaimer $k$ is assigned to vessel $j$; 0 otherwise,

$\beta_{jkm}$ 1 if vessel $j$ is assigned to reclaimer $k$ and berth $m$; 0 otherwise.

**Master Problem Model:**

$$\text{minimize} \sum_{j} Weight_j C_j$$

subject to:

$$\sum_{m} Y_{mj} = 1 \qquad \forall j \quad (5.1)$$

$$L_j . Y_{mj} \leq G_m \qquad \forall m, j \quad (5.2)$$

$$S_j \geq R_j \qquad \forall j \quad (5.3)$$

$$S_{j'} \geq C_j - |T|(1 - X_{mjj'}) \qquad \forall m, j, j^{'} : j \neq j^{'} \quad (5.4)$$

$$X_{mjj'} + X_{mj'j} \leq Y_{mj} \qquad \forall m, j, j^{'} : j \neq j^{'} \quad (5.5)$$

$$X_{mjj'} + X_{mj'j} \leq Y_{mj'} \qquad \forall m, j, j^{'} : j \neq j^{'} \quad (5.6)$$

$$X_{mjj'} + X_{mj'j} \leq 1 \qquad \forall m, j, j^{'} : j \neq j^{'} \quad (5.7)$$

$$X_{mjj'} + X_{mj'j} \geq Y_{mj} + Y_{mj'} - 1 \qquad \forall m, j, j^{'} : j \neq j^{'} \quad (5.8)$$

$$\sum_{r} Z_{jr} = 1 \qquad \forall j \quad (5.9)$$

$$C_j \geq \sum_{r} B_r Z_{jr} \qquad \forall j \quad (5.10)$$

$$C_j \leq \sum_{r} E_r Z_{jr} \qquad \forall j \quad (5.11)$$

$$\sum_{k} \Omega_{jk} = 1 \qquad \forall j \quad (5.12)$$

$$XR_{kii'} + XR_{ki'i} \leq \Omega_{VesselOfTask_i,k} \qquad \forall k, i, i^{'} : i \neq i^{'} \quad (5.13)$$

$$XR_{kii'} + XR_{ki'i} \leq \Omega_{VesselOfTask_{i'},k} \qquad \forall k, i, i^{'} : i \neq i^{'} \quad (5.14)$$

$$XR_{kii'} + XR_{ki'i} \leq 1 \qquad \forall k, i, i^{'} : i \neq i^{'} \quad (5.15)$$

$$XR_{kii'} + XR_{ki'i} \geq \Omega_{VesselOfTask_i,k} + \Omega_{VesselOfTask_{i'},k} - 1 \qquad \forall k, i, i^{'} : i \neq i^{'} \quad (5.16)$$

$$SR_{i'} \geq CR_i - |T|(1 - XR_{kii'}) \qquad \forall k, i, i^{'} : i \neq i^{'} \quad (5.17)$$

$$SR_i + P_i = CR_i \qquad \forall i \quad (5.18)$$

$$XR_{kii''} + XR_{ki''i'} \leq 1$$
$$\forall k, i, i^{'}, i^{''} : (VesselOfTask_i = VesselOfTask_{i'}) \& (VesselOfTask_i \neq VesselOfTask_{i''}) \quad (5.19)$$

$$SR_i \geq S_i \qquad \forall i \quad (5.20)$$

$$C_j \geq CR_i + \sum_{m} \sum_{k} Distance_{mk} \beta_{jkm} \qquad \forall j, i \in AssignedTasksToVessel_j \quad (5.21)$$

$$\sum_{k} \sum_{m} \beta_{jkm} = 1 \qquad \forall j \quad (5.22)$$

$$Y_{mj} + \Omega_{jk} \leq \beta_{jkm} + 1 \qquad \forall j, m, k : G_m \geq L_j \quad (5.23)$$

$$Y_{mj}, X_{mjj'}, XR_{kii'}, Z_{jr}, \Omega_{jk}, \beta_{jkm} \in \{0, 1\},$$
$$S_j, C_j, SR_j, CR_j \geq 0$$
$$\forall j, j^{'}, i, i^{'}, r, m, k \quad (5.24)$$

The objective of the master problem is to minimize total weighted departure times of vessels. We present constraints associated with berth allocation problem from (5.1) to (5.11). By constraints (5.1) and (5.2), each vessel must be assigned to one of the berths that its size allows. Constraint (5.3) simply ensures that a vessel cannot moor before its arrival. $X_{mjj'}$ and $Y_{mj}$ variables are put together by constraints from (5.5) to (5.8) to determine berthing order of vessels that are assigned to same berth. If vessels $j$ and $j'$ are assigned to same berth, then they must use that berth sequentially ($X_{mjj'} + X_{mj'j} = 1$). If at least one of $j$ and $j'$ is not assigned to berth $m$, then corresponding $X_{mjj'}$ variable takes the value of 0. Non-overlapping stays of vessels that use the same berth is ensured by constraint (5.4) using $X_{mjj'}$ variables. In other words, if $X_{mjj'}$=1, then vessel $j$ cannot moor before vessel $j'$ leaves.

Constraints (5.9), (5.10) and (5.11) together impose the effects of tidal time windows on berth allocations. The departure time of a vessel is assigned to one and only one high tide period by constraint (5.9) and its exact time is determined by constraints (5.10) and (5.11).

Constraints from (5.12) to (5.19) are associated with reclaimer scheduling problem. We assign a vessel to one and only one reclaimer to perform reclaiming of associated stockpiles by constraint (5.12). Constraints (5.13) to (5.16) determine processing order of tasks on reclaimers, similar to those of berth allocation problem. Then via constraint (5.17) tasks that are assigned to the same reclaimer cannot be reclaimed simultaneously.

Constraint (5.18) states that reclaiming of a stockpile requires a fixed amount of time and it must be performed non-preemptively. Note that duration of stay of a vessel is not fixed since it depends on factors such as reclaimer schedules and times of high tide periods. A reclaimer must perform reclaiming jobs of the same vessel consecutively because of technological restrictions of in-terminal transportation system (conveyor belts), and it is enforced by constraint (5.19).

To link the constraints of reclaimer scheduling and yard allocation, we use constraints (5.20) and (5.21). Constraint (5.20) indicates that reclaimers cannot start

reclaiming a stockpile before associated vessel moors. By constraint (5.21) we calculate the earliest time that a vessel can leave the terminal by taking completion time of reclaiming each stockpile as well as the distance between berth and reclaimer into account. Constraints (5.22) and (5.23) determine, for each vessel, the distance between the berth and the reclaimer that it is assigned. These constraints together enforce $\beta_{jkm}$ to take the value of 1 if vessel $j$ is both assigned to berth $m$ ($Y_{mj}$=1) and reclaimer $k$ ($\Omega_{jk}$=1). Lastly, constraint (5.24) determines domains of variables.

**Model Enhancements:**

An important observation on the bottleneck of the terminal leads us to add a new set of constraints that increases the computational performance of the master problem. As we mentioned previously, duration of stay of vessels also depends on the availability of reclaimers, while reclaiming times of stockpiles are fixed. Therefore, reclaimer scheduling can provide an insight on earliest possible completion times of vessels. We develop the following relaxation of reclaimer scheduling and embed it into the master problem to be able to get the benefit from the fact that reclaiming times are fixed. This relaxation provides a better lower bound for estimating departure time of vessels.

First, we aggregate reclaiming tasks associated with a vessel to a single task. Aggregated task $j$ then has processing (reclaiming) time of $TotalLoadOfVessel_j$ which represents a minimum possible duration of stay of vessel $j$. We can calculate the lower bound for departure time of vessels by assuming that there are unlimited number of berths and yard space, no low tide periods, no non-crossing constraint of reclaimers and no distance between reclaimers and berths. Note that the time that a vessel stays at berth is actually a variable because it depends on availability of empty berths, schedules of reclaimers, high tide periods and distances between berths and reclaimers.

We introduce constraints (5.25) to (5.28) and a time-indexed variable $\alpha_{jt}$ to apply this relaxation. $\alpha_{jt}$ takes the value of 1 if reclaiming of aggregated task of vessel $j$ starts at time $t$; 0 otherwise. This relaxation can be viewed as a time-indexed formulation of simple parallel machine scheduling problem with release dates, where

a reclaimer is a machine and aggregated reclaiming operations associated with a vessel is a task.

$$C_j \geq \sum_t (t + TotalLoadOfVessel_j).\alpha_{jt} \qquad \forall j \quad (5.25)$$

$$\sum_{t:t \geq R_j} \alpha_{jt} = 1 \qquad \forall j \quad (5.26)$$

$$\sum_{t:t < R_j} \alpha_{jt} = 0 \qquad \forall j \quad (5.27)$$

$$\sum_j \sum_{s \in \{max(1, t - TotalLoadOfVessel_j), \ldots, t\}} \alpha_{js} \leq |C| \qquad \forall t \quad (5.28)$$

$$\alpha_{jt} \in \{0, 1\} \qquad \forall j, t \quad (5.29)$$

Constraints (5.26) and (5.27) assign the start of reclaiming process for vessel $j$ to a time point $t$, which is after the arrival of this vessel. Capacity constraint (5.28) ensures that the number of vessels handled simultaneously must be less than or equal to the total number of reclaimers at any time. By constraint (5.25) we link variable $\alpha_{it}$ and original master problem variables. It also enforces departure time of a vessel to be not less than the theoretical minimum departure time which is dictated by group of constraints (5.25)-(5.29). We add these new constraints to the master problem formulation (5.1)-(5.24). Since the LP relaxation with these added constraints is very tight, it improves the performance of the model especially when the total output rate of reclaimers are less than that of berths, by reducing the search space for non-time-indexed variables.

Following Cire et al. (2015) who point out the importance of capturing some characteristics of the subproblem via their relaxations for a better performance of LBBD algorithm, we also add a relaxation of yard allocation problem as a constraint

to the master problem. In this relaxation, whole stockyard is aggregated into a single resource with capacity of total stocking positions ($|A||U|$), rather than considering each stocking position and pad individually. The resulting constraint (5.30) ensures that, at any time, the total length of stockpiles that are stacked in the stockyard but are not completely reclaimed must be less than the total available stocking positions.

By this way, we eliminate some solutions which are infeasible for the original problem. As a result, the master problem may have an optimal solution value to be much nearer to the global optimal and/or the algorithm may need fewer iterations until termination. In addition to that, constraint (5.30) also helps the model to identify the infeasibility of the instance caused by not enough yard space to accommodate all stockpiles given arrival times of vessels and stockpiles. We introduce an auxiliary parameter $ArrivalTime_{it}$ that has the value of 1 if arrival time of stockpile $i$ is $t$ and 0 otherwise. Further, we define an auxiliary time-indexed variable $\delta_{it}$ that takes the value of 1 if $SR_i = t$ and 0 otherwise. That is, $\sum_t t.\delta_{it} = SR_i$, $\forall i$. Hence, constraint (5.30) is stated as follows:

$$\sum_i \sum_{s\in\{max(1,min(R_i,t-P_i)),\ldots,t-P_i\}} K_i(ArrivalTime_{is} - \delta_{is}) + \sum_i \sum_{s\in\{max(1,t-P_i+1),\ldots,t\}} K_i ArrivalTime_{is} \leq |A||U| \quad \forall t \quad (5.30)$$

### *5.3.2 Subproblem*

In this section, we present constraint programming (CP) subproblem model. It is a feasibility model; that is, its objective is to find a feasible solution or to prove that the problem instance is infeasible. The subproblem of LBBD-IDBT must be represented in terms of activities and resources. In this representation, reclaiming and storing stockpiles are activities while reclaimers and stock positions correspond to resources. For a detailed information on the methodology, we refer readers to Chapter 1.

**Decision Variables:**

$\overline{Reclaim_i}$ an interval variable that represents reclaiming of stockpile $i$. Its domain is $[S_{VesselOfTask_i}, C_{VesselOfTask_i} - Distance_{rk} Y_{r,VesselOfTask_i} \Omega_{VesselOfTask_i,k}]$ and it is determined by the optimal solution to the master problem. Therefore this activity must be performed between associated vessel's ($VesselOfTask_i$) mooring and departure times by also considering the distance between reclaimer $k$ and berth $r$ assigned to this vessel.

$Reclaim_{ik}$ an optional interval variable that represents reclaiming of stockpile $i$ by reclaimer $k$. This variable is active in the model if and only if reclaimer $k$ is assigned to stockpile $i$. Otherwise its domain will be $\emptyset$.

$\overline{Stack_i}$ an interval variable that represents stacking of stockpile $i$. In our model we assume that it takes 0 unit time and its domain is fixed to $R'_i$, the time that stockpile $i$ arrives to the terminal.

$Stack_{il}$ an optional interval variable that represents stacking of stockpile $i$ such that its leftmost is placed to stock position $l$.

$\overline{Occupy_i}$ an interval variable that stands for the interval in which stockpile $i$ is stored in stockyard. Start of this interval variable is fixed to $R'_i$, i.e., arrival time of this stockpile. Its domain is $[R'_i, C_{VesselOfTask_i} - Distance_{rk} Y_{r,VesselOfTask_i} \Omega_{VesselOfTask_i,k}]$.

$Occupy_{il}$ an optional interval variable that represents the interval in which stockpile $i$ is stored in the stockyard such that its leftmost is assigned to stock position $l$.

$ReclaimerUsesPosition_{ikl}$ an optional interval variable that stands for reclaiming of stockpile $i$ which is located such that its leftmost is assigned to stock position $l$ and is reclaimed by reclaimer $k$. When it is active, its

domain is $[S_j, C_{VesselOfTask_i} - Distance_{rk} Y_{r,VesselOfTask_i} \Omega_{VesselOfTask_i,k}]$.

$NonCross_{ii'kk'll'}$ a sequence variable that is used for non-crossing constraint and it determines relative order of interval variables $ReclaimerUsesPosition_{ikl}$ and $ReclaimerUsesPosition_{i'k'l'}$. Its domain consists of permutation of order of these interval variables.

Note that the duration of interval variables $\overline{Occupy}_i$ and $Occupy_{il}$ are not fixed. Stockpile $i$ stays in the stockyard until completely reclaimed by reclaimers, differently from other variables of the model. In the following, we present constraint programming subproblem.

**Subproblem Model:**

Feasibility Problem:

$$Alternative(\overline{Reclaim}_i, (Reclaim_{ik}|\forall k)) \qquad \forall i \quad (5.31)$$

$$Alternative(\overline{Stack}_i, (Stack_{il}|\forall l)) \qquad \forall i \quad (5.32)$$

$$Alternative(\overline{Occupy}_i, (Occupy_{il}|\forall l)) \qquad \forall i \quad (5.33)$$

$$Presence(Reclaim_{ik}) = 1 \qquad \forall i,k : \Omega_{VesselOfTask_i,k} = 1 \quad (5.34)$$

$$1 - Presence(Stack_{il}) \geq Presence(Reclaim_{ik}) \qquad \forall i,k,l : (Q_k > D_l)||(D_l - Q_k > 1) \quad (5.35)$$

$$Presence(Stack_{il}) = 0 \qquad \forall i,l : (|U| - \pi_l + 1) - K_i < 0 \quad (5.36)$$

$$Disjunctive(Reclaim_{ik}|\forall i) \qquad \forall k \quad (5.37)$$

$$EndBeforeStart(\overline{Stack}_i, \overline{Reclaim}_i) \qquad \forall i \quad (5.38)$$

$$Presence(Stack_{il}) = Presence(Occupy_{il}) \qquad \forall i, l \quad (5.39)$$

$$EndAtEnd(\overline{Occupy}_i, \overline{Reclaim}_i) \qquad \forall i \quad (5.40)$$

$$Start(\overline{Occupy}_i) = R_i^{'} \qquad \forall i \quad (5.41)$$

$$Disjunctive(Occupy_{il} | \forall i, l^{'} : l^{'} \leq l, l^{'} \geq l - K_i + 1) \qquad \forall l \quad (5.42)$$

$$Disjunctive(NonCross_{ii^{'}kk^{'}ll^{'}})$$
$$\forall i, i^{'}, k, k^{'}, l, l^{'} : i \neq i^{'}, k < k^{'}, Q_k = Q_{k^{'}}, l + K_i > l^{'}, l \leq |U|, l^{'} \leq |U| \quad (5.43)$$

$$Presence(Stack_{il}) + Presence(Reclaim_{ik}) \leq ReclaimerUsesPosition_{i,k,vertical_l} + 1$$
$$\forall i, k, l : (Q_k > D_l) || (D_l - Q_k > 1) \quad (5.44)$$

$$\sum_k \sum_{l:l<\overline{U}} Presence(ReclaimerUsesPosition_{ikl}) = 1 \qquad \forall i \quad (5.45)$$

$$StartAtStart(ReclaimerUsesPosition_{ikl}, \overline{Reclaim}_i) \qquad \forall i, k, l : l < |U| \quad (5.46)$$

$$EndAtEnd(ReclaimerUsesPosition_{ikl}, \overline{Reclaim}_i) \qquad \forall i, k, l : l < |U| \quad (5.47)$$

Constraints (5.31), (5.32) and (5.33) assign stockpiles to reclaimers and stocking positions. Constraint (5.34) fixes reclaimer to stockpile assignments as determined by the master problem. Constraint (5.35) enforces that a reclaimer cannot process stockpiles which are not located on adjacent pads to the rail track it is mounted on. Otherwise, left hand side of the constraint becomes 0 and forces $Reclaim_{ik}$ interval variable to be inactive in the model, i.e., $Presence(Reclaim_{ik}) = 0$. Constraint

(5.36) states that a stockpile cannot be assigned to a stocking position $l$ if there is not enough stocking positions from $l$ to rightmost of the pad to accommodate all of the stockpile because a stockpile must be located in adjacent stocking positions as a single rectangular pile. Constraint (5.37) ensures non-overlapping of reclaiming operations of stockpiles that are assigned to the same reclaimer.

Constraint (5.38) states that a stockpile needs to be stacked in the stockyard to be able to reclaim this stockpile. After stacked, a stockpile stays at the same stocking positions, by constraint (5.39). Storage of a stockpile in the stockyard ends with the completion of reclaiming of this stockpile, which is denoted by constraint (5.40). For each stockpile, stacking time and start of its stay in the stockyard is fixed to its arrival time and it is ensured by constraint (5.41). Constraint (5.42) states that stockpiles that occupy stock position $l$ must be non-overlapping. If two stockpiles share at least one common stocking position, then they cannot stay in the stockyard at the same time.

We need constraint (5.44) for $ReclaimerUsesPosition_{ikl}$ variables to take proper values. By constraint (5.45), one and only one $ReclaimerUsesPosition_{ikl}$ variable is active for each stockpile because each stockpile is assigned to one reclaimer and its leftmost is stored in one stocking position. Constraint (5.43) ensures non-crossing of reclaimers by considering relative positioning of interval variables $ReclaimerUsesPosition_{ikl}$ and $ReclaimerUsesPosition_{i'k'l'}$ if and only if required conditions that cause reclaimers to cross each other are satisfied. This constraint also considers the fact that reclaimers repetitively travels back and forth by the length of the stockpile during reclaiming operation.

Lastly, constraints (5.46) and (5.47) together ensure that start and end times of active $ReclaimerUsesPosition_{ikl}$ interval variable to be identical with those of $\overline{Reclaim}_i$ . These variables both represent the interval in which stockpile $i$ is reclaimed, but the former additionally keeps stockpile to reclaimer and stocking position assignments.

### *5.3.3 Benders Cuts*

A "nogood" cut applies to all types of problems, however, it is often weak. Therefore, we investigate potential causes of infeasibility to derive more effective problem specific Benders cuts. Given an optimal solution of the master problem, there can be two potential causes of an infeasible subproblem in LBBD-IDBT.

- Stockyard Infeasibility: it may not be possible to distribute stockpiles to stocking pads since each stockpile should be stacked as a single pile.
- Non-Crossing Infeasibility: given the solution of the master problem, non-crossing constraint may be violated in the subproblem.

**Stockyard Infeasibility Cut:**

When the infeasibility is caused by the distribution of stockpiles to stocking pads, CP identifies that subproblem is infeasible. However, it cannot identify the constraints that are causing the infeasibility. In this case, we need to find the exact cause of the infeasibility with a specific cut generation procedure.

For this reason, we introduce the following two auxiliary expressions and two binary variables to implement the conditional statement that is used within this procedure.

$$A1_{ht} = \sum_{i \in S1CUT_h} \sum_{s \in \{max(1,min(R_i, t-P_i)),\ldots,t-P_i\}} (ArrivalTime_{is} - \delta_{is}) + \sum_{i \in S1CUT_h} \sum_{s \in \{max(1,t-P_i+1),\ldots,t\}} ArrivalTime_{is}$$

$$A2_h = \sum_j \sum_k Cut2_{hjk} \Omega_{jk}$$

$Var1_h$ a binary variable that takes the value of 1 if all stockpiles in $S1CUT_h$ stays at the same time, that is $A1_{ht} = |S1CUT_h|$, 0 otherwise.

$Var2_h$ a binary variable that takes the value of 1 if vessel to reclaimer assignments for related vessels are the same with previous iteration, that is $A2_h = |S2CUT_h|$, 0 otherwise.

At iteration $h$ of LBBD-IDBT, we derive a strong Benders cut for the stockyard infeasibility by the following procedure:

**Initialization:** Set $TSU_{ht} = \emptyset$, $S1CUT_h = \emptyset$, $S2CUT_h = \emptyset$.

**1:** Let $TSU_{ht}$ to be the list of total stockyard usage for each time period. For each $t$, calculate $TSU_{ht}$ as $\sum_i \{K_i | R_i^{'} \leq t \leq CR_i\}$.

**2:** Select time $t^{'}$ with the largest stockyard usage:

**2.1:** Find stockpiles that are stacked but not reclaimed until time $t^{'}$, and add these tasks to set $S1CUT_h$. That is, $S1CUT_h = \{i | R_i^{'} \leq t^{'} < CR_i\}$.

**2.2:** Solve an auxiliary MIP model, $SIM_1$ (Appendix D), to check whether it is possible to allocate these stockpiles to pads. If this MIP model is feasible, go to Step 2.3. Else, we obtain the proof of infeasibility: given the solution of the master problem, it is not possible to allocate the tasks in $S1CUT_h$ to pads. That is, these tasks cannot stay at the stockyard simultaneously. Add the following cut that prevents that situation to the master problem, and go to Step 3:

$$A1_{ht} \leq |S1CUT_h| - 1 \qquad \forall t \quad (5.48)$$

**2.3:** Solve a more restricted MIP model, $SIM_2$ (Appendix D), which takes vessel to reclaimer assignments into consideration. In this model, a task associated with a vessel can only be allocated to pads adjacent to the reclaimer that this vessel is assigned. If $SIM_2$ is feasible, then go to Step 2, and select the next $t^{'}$ value with the largest stockyard usage. Else, we obtain the proof of infeasibility: given the optimal solution of the master problem, it is not possible to allocate the tasks in $S1CUT_h$ to pads if we

consider vessel to reclaimer assignments. If all stockpiles in set $S1CUT_h$ are staying at the stockyard at the same time, then current vessel to reclaimer assignments must be changed to have a feasible solution. Go to Step 2.4 to derive this conditional cut.

**2.4:** Find stockpiles that are stacked but not completely reclaimed at time $t^{'}$, and add all related vessel to reclaimer $(j,k)$ assignments to set $S2CUT_h$. For each element of $S2CUT_h$, set $Cut2_{hjk} = 1$, and for each $(j,k) \notin S2CUT_h$, set $Cut2_{hjk} = 0$.

Next two constraints are required to have correct values for $Var1$ and $Var2$, the auxillary variables that we introduced previously.

$$1 - Var1_h \leq |S1CUT_h| - A1_{ht} \leq |S1CUT_h|(1 - Var1_h) \qquad \forall t \quad (5.49)$$

$$1 - Var2_h \leq |S2CUT_h| - A2_h \leq |S2CUT_h|(1 - Var2_h) \quad (5.50)$$

Now, we can straightforwardly implement the following condition: "if all stockpiles in set $S1CUT_h$ are staying at the stockyard at time $t$, then current vessel to reclaimer assignments must be changed to have a feasible solution", which can be stated as "if $Var1_h = 1$, then $Var2_h = 0$". That is,

$$Var2_h \leq 1 - Var1_h \quad (5.51)$$

Add constraints (5.49), (5.50), and (5.51) to the master problem and go to Step 3.

**3:** Go to the next iteration of LBBD-IDBT and solve the modified master problem.

**Example with Stockyard Infeasibility Cut:**

We describe the execution of LBBD-IDBT step by step on a small problem instance which requires a stockyard infeasibility cut. In this instance, there are four vessels

and each of them is associated with a single stockpile. There are four berths, so that these vessels can moor at the same time. Other parameters regarding to the stockpiles are presented in Table 1. In this instance, we assume no low tide periods and also zero travel time for stockpiles from stockyard to vessels. There are two pads with five stocking positions each. Each stockpile is served with one reclaimer, thus reclaimers are mounted on two different rail tracks.

Table 5.1: Parameters for the example instance

| Stockpiles / Parameters | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Weight | 10 | 5 | 1 | 1 |
| Arrival Time | 1 | 1 | 1 | 4 |
| Processing Time | 5 | 2 | 3 | 3 |
| Length | 3 | 2 | 3 | 3 |

**STEP 1:** Solve the master problem (iteration 1).

Stockyard operations for each stockpile in the optimal solution are presented in Figure 5.2(a). In this figure, each hatched rectangle represents that a stockpile is stacked but not being reclaimed, and solid rectangles stand for reclaiming operation. Thus,

$z^* = \sum_j Weight_j C_j = 10(1+5)+5(1+2)+1(3+3)+10(6+3) = 171.$

**STEP 2:** Solve the subproblem.

Even though the total stockyard usage at any time is less than the total stocking positions available by the relaxation of the yard allocation problem in the master problem, subproblem turns out to be infeasible.

**STEP 3:** Determine the cause of infeasibility and add cuts.

**STEP 3.1:** Calculate total stockyard positions to be occupied at any time. That is, $TSU_{1t} = \{8, 8, 6, 9, 9, 3, 3, 3\}$.

**STEP 3.2:** $t$=4 and $t$=5 has the highest total stockyard usage with 9 stocking positions. Arbitrarily select one of these time points, let us select $t$=4. Identify the indices of stockpiles that are stacked at the stockyard during time $t$: $S1CUT_1 = \{1, 3, 4\}$.

**STEP 3.3:** Solve MIP model $SIM_1$.

It is infeasible, hence proves that it is not possible to keep stockpiles 1, 3, and 4 at the stockyard simultaneously. Add a cut that forbids the overlapping of these stockpiles over time, and go to the next iteration.

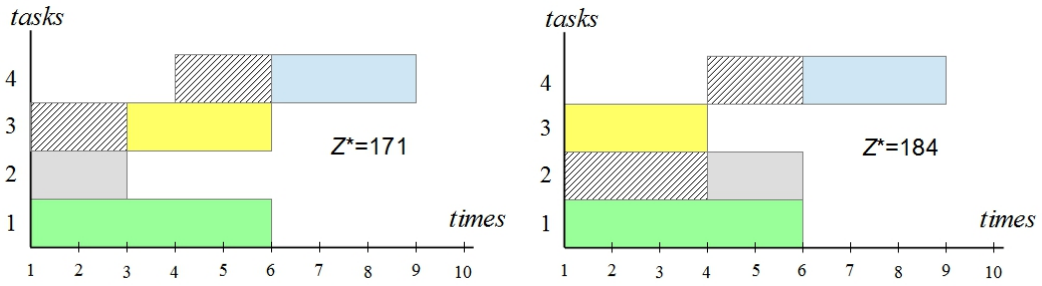
**STEP 4:** Solve the master problem (iteration 2).

Optimal stockyard operations for each stockpile are presented in Figure 5.2(b). Accordingly,

$z^* = \sum_j Weight_j.C_j = 10(1+5)+5(4+2)+1(1+3)+10(6+3) = 184.$

**STEP 5:** Solve the subproblem.

This time the subproblem finds a feasible solution in which stockpile 1 is located on pad 1 and other stockpiles are located on pad 2. Note that stockpile 4 is stacked after stockpile 3 is completely reclaimed from the stockyard. Algorithm terminates with an optimal solution.




(a) Iteration 1 (b) Iteration 2

Figure 5.2: Solutions from the master problem for different iterations

If the above procedure fails to generate a cut, then the infeasibility is caused by the violation of non-crossing constraint. However, it is not likely as the proposed decomposition substantially prevents such a violation. Hence, we generate a nogood cut in such a situation. According to Chu and Xia (2004), a Benders cut is valid if it satisfies the following two conditions: it does not remove any feasible solution to the original problem but does cut the current solution to the master problem.

**Proposition 13.** *The explained procedure generates a valid Benders cut.*

*Proof.* To prove the first condition, assume that a cut removes a feasible solution to the original problem. If subproblem turns out to be infeasible, by definition, the procedure can generate a cut when (*i*) $SIM_1$ is infeasible, (*ii*) $SIM_1$ is feasible but $SIM_2$ is infeasible, and (*iii*) $SIM_1$ and $SIM_2$ are both feasible.

(*i*) For a time point $t$, there is no feasible distribution of stockpiles that stays at the stockyard to pads, and it is verified by $SIM_1$. Thus, by constraint (44), we only remove this specific situation, that is "these stockpiles cannot be stored at the stockyard simultaneously".

(*ii*) For a time point $t$, there is no feasible distribution of stockpiles that stays at the stockyard to pads given the stockpile-reclaimer assignments. However, it is possible to distribute these tasks to pads by changing these assignments as $SIM_1$ is feasible. Therefore, by constraint (47), we remove a more restricted portion of the solution space by specifically addressing that "if these stockpiles are stored at the stockyard simultaneously, then it is not possible to have the current stockpile-reclaimer assignments".

In above cases, we only prevent the situations which are proved to be infeasible via $SIM_1$ and $SIM_2$, respectively.

(*iii*) Subproblem is infeasible, but both $SIM_1$ and $SIM_2$ are feasible. Then we add a nogood constraint that asserts "if we sent these set of values of variables to the subproblem, then we cannot obtain a feasible solution to the original

problem". Thus, we need to change the value of at least one variable. A nogood cut is valid as it only excludes the complete solution to the master problem, which is proven to be infeasible via the subproblem.

Therefore, the cut generation procedure does not cut any feasible solution to the original problem. This contradicts the initial statement. The proof of the second condition is trivial, as in all three cases, we forbid the occurrence of the same solution that is proved infeasible to the subproblem. □

Moreover, this cut generation procedure guarantees the finite convergence of the algorithm since the master problem has a finite domain and generates a valid Benders cut at each iteration.

## *5.4 IDBT Problem with Continuous BAP*

Previously, we assumed that each berth is a discrete resource that can be occupied by at most one vessel at any time. Alternatively, in a more general case, a berth is assumed to be a single continuous resource and a vessel can moor at any position of the berth area.

This variant of berth allocation problem can be represented on a space-time graph where vertical and horizontal axes correspond to berth area and time horizon, respectively. In this space-time graph each vessel is represented by a rectangle. Width of a rectangle stands for the length of the vessel and its height represents the time vessel stays at the berth. Rectangles for all vessels must be non-overlapping in two dimensional graph to be able to have a feasible solution to BAP.

Most of the related literature assumes that height of each rectangle is known in advance, hence, is an input of the problem. In such a case, the problem can be studied as a 2-D bin packing problem (Lim, 1998). However, in our study, the height of a rectangle is not fixed but depends on the reclaimer schedules, tidal conditions and the distance between mooring position of vessel and the reclaimer that handles this vessel.

Therefore, the resulting problem cannot be represented as a 2-D bin packing problem. In addition to sets, parameters, and decision variables which are presented for IDBT problem, we also introduce the following notation for the variant with continuous BAP.

**Additional Parameters:**

$L$ total length of berth area,

$L_j$ length of vessel $j$,

$CS_k$ an auxiliary parameter that defines the nearest side of berth area to reclaimer $k$ which has a value of 1 if it is left side of berth area and a value of 2 if it is right side,

$DistToBerth_k$ distance between reclaimer $k$ and the leftmost (or the rightmost) of berth area in unit times.

**Additional Decision Variables:**

$Y_j$ position at which the leftmost of vessel $j$ moors over berth area,

$X_{jj'}$ binary decision variable that determines the sequence of vessels over time, which takes a value of 1 if vessel $j$ departures before vessel $j'$ moors, 0 otherwise.

$I_{jj'}$ binary decision variable that determines the relative positions of vessels over berth area, if these two vessels are overlapping in time. It takes a value of 1 if vessel $j$ is handled at the right side of vessel $j'$ and these two vessels are overlapping in time, 0 otherwise,

$Dist_{jk}$ amount of time that is required to carry the material between the midpoint of vessel $j$ and reclaimer $k$ (in unit times).

A general structure of a dry bulk terminal with a continuous berth area is presented in Figure 5.4 below. In this figure, vessel $j$ is moored to berth as its leftmost is placed to point $Y_j$ over berth area.

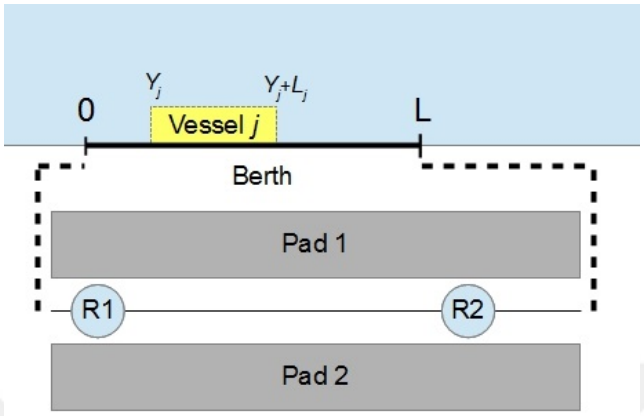



Figure 5.3: General structure of IDBT problem with continuous BAP

Moreover, dashed lines interpret the conveyor belt system that transfer materials from reclaimers to berth area and their length is given by the parameter $DistToBerth_k$. If vessel $j$ is assigned to reclaimer 1, then the amount of time that is required to carry material between the midpoint of vessel $j$ and reclaimer $k$ ($Dist_{jk}$) is calculated as a sum of $DistToBerth_1$ and the distance from leftmost of berth area to midpoint of vessel $j$, that is $Y_j + L_j/2$. This is because the conveyor system associated with reclaimer 1 is nearer to leftmost of berth rather than its rightmost, hence $CS_1 = 1$. Similarly, if this vessel is assigned to reclaimer 2, $Dist_{jk}$ will be the sum of $DistToBerth_2$ and $L - (Y_j + L_j/2)$.

In the following, mixed integer programming model for master problem of IDBT problem with continuous BAP is presented.

$$\text{minimize} \sum_i \sum_j Weight_j C_j$$

subject to:

$$Y_j \leq L - L_j \qquad \forall j \quad (5.52)$$

$$X_{jj'} + X_{j'j} + I_{jj'} + I_{j'j} = 1 \qquad \forall j, j' : j \neq j' \quad (5.53)$$

$$Y_{j'} \geq L_j I_{jj'} + Y_j - L(1 - I_{jj'}) \qquad \forall j, j^{'} : j \neq j^{'} \quad (5.54)$$

$$S_{j'} \geq C_j - |T|(1 - X_{jj'}) \qquad \forall j, j^{'} : j \neq j^{'} \quad (5.55)$$

$$Dist_{jk} = DistToBerth_k + Y_j + L_j/2 \qquad \forall j, \forall k : CS_k = 1 \quad (5.56)$$

$$Dist_{jk} = DistToBerth_k + L - (Y_j + L_j/2) \qquad \forall j, \forall k : CS_k = 2 \quad (5.57)$$

$$C_j \geq CR_i + Dist_{jk} - |T|(1 - \Omega_{jk}) \quad \forall i, \forall k, \forall j, j^{'} : j \neq j^{'} \& VesselOfTask_i = j \quad (5.58)$$

$$Constraints(5.9) - (5.20)$$

$$Y_j, X_{jj'}, I_{jj'}, XR_{kii'}, Z_{jr}, \Omega_{jk}, \{0, 1\},$$
$$Dist_{jk}, S_j, C_j, SR_j, CR_j \geq 0$$
$$\forall j, j^{'}, \forall i, i^{'}, \forall r, \forall k \quad (5.59)$$

In this model, all the constraints regarding scheduling of reclaimers as well as assignment of departure times of vessels to high tide periods are also valid for the new model. Therefore variables $XR_{kii'}$, $Z_{jr}$, $\Omega_{jk}$, $S_j$, $C_j$, $SR_j$ and $CR_j$ of IDBT problem are also exist in the model with continuous BAP. On the other hand, all constraints regarding berth allocation problem is reconstructed and it is presented by constraints (5.52)-(5.59).

Constraint (5.52) defines the feasible positions that leftmost of a vessel can be placed by considering its length. Relative positioning of two vessels $j$ and $j^{'}$ are expressed by constraint (5.53). Accordingly, position of vessel $j$ with respect to vessel $k$ must be one and only one of the following four cases, otherwise, it means that these two vessels are overlapping in time-space graph, which makes allocation of vessels to berth area infeasible.

(i) Vessel $j$ completes before arrival of vessel $j^{'}$ ($X_{jj'} = 1$),

(ii) vessel $j^{'}$ completes before arrival of vessel $j$ ($X_{j^{'}j} = 1$),

(iii) $j$ and $j^{'}$ overlaps in time and $j$ is handled at the right side of vessel $j^{'}$ ($I_{jj^{'}} = 1$),

(iv) $j$ and $j^{'}$ overlaps in time and $j^{'}$ is handled at the right side of vessel $j$ ($I_{j^{'}j} = 1$).

Constraint (5.54) enforces non-overlapping of vessels over berth area, by relating the variable that determines the position of a vessel over berth ($Y_j$) with $I_{jj^{'}}$ variables. If vessels $j$ and $j^{'}$ are not overlapping in time, i.e., $I_{jj^{'}} + I_{j^{'}j} = 0$, then the relative positioning of these vessels becomes irrelevant, since right hand side of constraint (5.54) becomes less than zero ($Y_{j^{'}} \geq Y_j - L$). Similarly, the sequence of vessels over time is determined by constraint (5.55). Distance between the berth position that a vessel moors and the assigned reclaimer is determined by constraints (5.56) and (5.57). It consists of two components: amount of time required to transport stockpiles ($i$) between reclaimer and nearest side of berth area and ($ii$) between this side of berth area to midpoint position that vessel is moored. This calculation is separated into two constraints since conveyor belt has different routes according to the position of reclaimer. Lastly, the earliest departure time of vessels are determined by constraint (5.58). That is, the departure time of a vessel must be larger than or equal to sum of the completion time of a stockpile assigned to this vessel and the time required to travel the distance between assigned reclaimer and the midpoint of mooring position of the vessel. The latter is enforced by the $Dist_{jk}$ variables together with the component that checks whether vessel $j$ is assigned to reclaimer $k$. Since a vessel is assigned to one and only one reclaimer, the earliest departure time of a vessel is calculated for that reclaimer. For other reclaimers, the right hand side of this constraint becomes less than or equal to zero.

It is important to note that the enhancements (relaxation of sub problems) implemented by constraints (5.25)-(5.29) and constraint (5.30) are still valid for the problem with continuous BAP since they are independent of the berth structure. Note that the LBBD-IDBT can be straightforwarly implemented for this variant of the problem as well, by replacing master problem (5.1)-(5.24) with (5.52)-(5.59).

### *5.5 Computational Experiments*

To test the performance of the proposed algorithm, we design a set of computational experiments by considering the experiments designed by Ernst et al.. (2017) and Boland et al.. (2012). We consider a terminal with 4 berths ($|M|$), a stockyard which consists of three parallel stocking pads ($|A|$) and two rail tracks in between each adjacent pad pairs. Each pad has a length of 2000 meters and is discretized as 100 meter-long stocking positions, hence there are 20 stock positions ($|U|$) in each pad. For this terminal structure, we consider two different numbers of reclaimers ($|C| = 3, 4$) to test our method with different material handling capabilities.

The other parameters of the problem are generated as follows:

- Number of vessels, $|V| = 10, 15, 20, 25$.
- Number of stockpiles per vessel, $|J|/|V| = 1, 1.4$. Therefore, instances have $|J| = 14, 21, 28, 35$ stockpiles in 1.4 stockpile per vessel setting.
- For the length of planning period ($|T|$), we considered three cases:

  $|T|$= a week = 168 hours for instances with 10 and 15 vessels.

  $|T|$= 1.5 week = 252 hours for instances with 20 vessels.

  $|T|$= 2 weeks = 336 hours for instances with 25 vessels.
- Arrival time of vessels ($R_j$) are generated uniformly within $[1, 2/3.|T|]$.
- Tidal time windows were obtained from the public website of Port of Newcastle, Australia which is available at http://www.bom.gov.au/australia/tides/.
- Length of each berth ($G_m$) is generated uniformly within $[7, 10]$, while lengths of vessels ($L_j$) are generated within $[3, 10]$.
- Workload of vessels is determined by the total workload of stockpiles that are assigned to them. Workload of each stockpile ($P_i$) is generated uniformly between

8 and 20 hours. Length of each stockpile ($K_i$) is determined proportionally to its workload, that is, $K_i = \lceil P_i/2 \rceil$.

We employ a full factorial design with 16 different scenarios. For each scenario, we generate 5 random instances. All computational experiments are performed by using CPLEX 12.7 and CP Optimizer on a computer with 2.4 Ghz Processor and 12 GB RAM. Time limit is set as one hour.

We first discuss the computational performance of the monolithic MIP model, i.e., the one without proposed decomposition. Even though it can solve all instances with 10 vessels, its performance is very limited for the remaining instances: only 20 of 60 instances are solved within the time limit, with an average of approximately five times larger solution times compared to that of the LBBD-IDBT. In 17 instances even no integer solution within time limit is found. Results of the computational experiments for LBBD-IDBT are presented in Tables 5.2 and 5.3. We also present a comparison of the computational performances of the proposed solution method and monolithic MIP model in Table 5.4. We use seven metrics in these tables to present our results:

($i$) Objective function value of the LP relaxation obtained from the master problem,

($ii$) optimal solution value of the master problem at the first iteration,

($iii$) percent deviation between ($i$) and ($ii$) to observe the performance of the master problem,

($iv$) optimal solution value at the end of the algorithm,

($v$) percent deviation between ($ii$) and ($iv$) to observe the performance of the LBBD-IDBT,

($vi$) solution time in seconds, and

($vii$) information on cuts added.

Table 5.2 shows that 39 out of 40 of instances with 4 berths and 4 reclaimers are solved within the time limit. Solution times increase not only with the number of vessels and stockpiles but also with the level congestion, i.e., the intensity of the total

Table 5.2: Results of LBBD-IDBT for a terminal with 4 reclaimers

| $\|V\|$ | $\|J\|$ | ins. | (i) | (ii) | (iii) | (iv) | (v) | (vi) | (vii) |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 10 | 1 | 3247.0 | 3247.0 | 0.00 | 3247.0 | 0.00 | 11.8 | - |
| | | 2 | 3328.0 | 3328.0 | 0.00 | 3328.0 | 0.00 | 13.0 | - |
| | | 3 | 3027.3 | 3048.0 | 0.68 | 3048.0 | 0.00 | 9.5 | - |
| | | 4 | 3050.0 | 3050.0 | 0.00 | 3050.0 | 0.00 | 14.7 | - |
| | | 5 | 4277.0 | 4277.0 | 0.00 | 4277.0 | 0.00 | 8.9 | - |
| | | avg. | 3385.9 | 3390.0 | 0.13 | 3390.0 | 0.00 | 11.6 | |
| | 14 | 1 | 4030.5 | 4037.0 | 0.16 | 4037.0 | 0.00 | 28.0 | - |
| | | 2 | 3203.0 | 3205.0 | 0.06 | 3205.0 | 0.00 | 27.8 | - |
| | | 3 | 3896.0 | 3896.0 | 0.00 | 3896.0 | 0.00 | 22.6 | - |
| | | 4 | 3621.0 | 3621.0 | 0.00 | 3621.0 | 0.00 | 29.4 | - |
| | | 5 | 3858.0 | 3858.0 | 0.00 | 3858.0 | 0.00 | 31.0 | - |
| | | avg. | 3721.7 | 3723.4 | 0.04 | 3723.4 | 0.00 | 27.8 | |
| 15 | 15 | 1 | 5704.0 | 5726.0 | 0.39 | 5726.0 | 0.00 | 35.8 | - |
| | | 2 | 4923.0 | 4985.0 | 1.26 | 4985.0 | 0.00 | 63.0 | - |
| | | 3 | 5289.8 | 5295.0 | 0.10 | 5295.0 | 0.00 | 21.5 | - |
| | | 4 | 4232.7 | 4269.0 | 0.86 | 4269.0 | 0.00 | 40.1 | - |
| | | 5 | 4998.0 | 4998.0 | 0.00 | 4998.0 | 0.00 | 37.2 | - |
| | | avg. | 5029.5 | 5054.6 | 0.52 | 5054.6 | 0.00 | 39.5 | |
| | 21 | 1 | 4749.8 | 4775.0 | 0.53 | 4775.0 | 0.00 | 104.2 | - |
| | | 2 | 6541.5 | 6615.0 | 1.12 | 6615.0 | 0.00 | 101.9 | - |
| | | 3 | 6474.5 | 6519.0 | 0.69 | 6519.0 | 0.00 | 143.0 | - |
| | | 4 | 4996.0 | 4996.0 | 0.00 | 4996.0 | 0.00 | 97.4 | - |
| | | 5 | 6051.3 | 6192.0 | 2.33 | 6355.0 | 2.63 | 331.5 | SI (1) |
| | | avg. | 5762.6 | 5819.4 | 0.93 | 5852.0 | 0.52 | 155.6 | |

SI($n$) indicates that stockyard infeasibility cuts are added where $n$ denotes the number of these cuts.

Table 5.2: Results of LBBD-IDBT for a terminal with 4 reclaimers - continued

| $\|V\|$ | $\|J\|$ | ins. | (i) | (ii) | (iii) | (iv) | (v) | (vi) | (vii) |
|---|---|---|---|---|---|---|---|---|---|
| 20 | 20 | 1 | 6414.3 | 6479.0 | 1.01 | 6479.0 | 0.00 | 75.4 | - |
| | | 2 | 6968.0 | 6968.0 | 0.00 | 6968.0 | 0.00 | 109.6 | - |
| | | 3 | 10106.9 | 10120.0 | 0.13 | 10120.0 | 0.00 | 76.8 | - |
| | | 4 | 9890.0 | 9938.0 | 0.49 | 9938.0 | 0.00 | 74.0 | - |
| | | 5 | 7982.5 | 8115.0 | 1.66 | 8115.0 | 0.00 | 72.5 | - |
| | | avg. | 8272.3 | 8324.0 | 0.66 | 8324.0 | 0.00 | 81.7 | |
| | 28 | 1 | 8862.0 | 8916.0 | 0.61 | 8916.0 | 0.00 | 252.2 | - |
| | | 2 | 9419.3 | 9459.0 | 0.42 | 9459.0 | 0.00 | 434.1 | - |
| | | 3 | 8476.8 | 8524.0 | 0.56 | 8524.0 | 0.00 | 1898.1 | - |
| | | 4 | 9426.1 | 9467.0 | 0.43 | 9467.0 | 0.00 | 290.6 | - |
| | | 5 | 10522.4 | 10604.0 | 0.78 | 10604.0 | 0.00 | 1415.4 | - |
| | | avg. | 9341.3 | 9394.0 | 0.56 | 9394.0 | 0.00 | 858.1 | |
| 25 | 25 | 1 | 14586.0 | 14605.0 | 0.13 | 14605.0 | 0.00 | 186.2 | - |
| | | 2 | 18298.0 | 18531.0 | 1.27 | 18531.0 | 0.00 | 175.3 | - |
| | | 3 | 14172.0 | 14224.0 | 0.37 | 14224.0 | 0.00 | 762.5 | - |
| | | 4 | 16681.0 | 16717.0 | 0.22 | 16717.0 | 0.00 | 243.0 | - |
| | | 5 | 17855.0 | 17873.0 | 0.10 | 17873.0 | 0.00 | 146.3 | - |
| | | avg. | 16318.4 | 16390.0 | 0.42 | 16390.0 | 0.00 | 302.7 | |
| | 35 | 1 | 18356.4 | 18395.5 | 0.21 | 18442.0 | 0.25 | 3600.0 | T.L. |
| | | 2 | 20098.0 | 20158.0 | 0.30 | 20158.0 | 0.00 | 2670.2 | - |
| | | 3 | 17252.0 | 17266.0 | 0.08 | 17266.0 | 0.00 | 846.0 | - |
| | | 4 | 16242.3 | 16299.0 | 0.35 | 16299.0 | 0.00 | 1160.8 | - |
| | | 5 | 15535.5 | 15581.0 | 0.29 | 15581.0 | 0.00 | 2562.8 | - |
| | | avg. | 17496.8 | 17539.9 | 0.25 | 17549.2 | 0.05 | 1810.0 | |

T.L. refers that instance is terminated by the time limit.

Table 5.3: Results of LBBD-IDBT for a terminal with 3 reclaimers

| $\|V\|$ | $\|J\|$ | ins. | (i) | (ii) | (iii) | (iv) | (v) | (vi) | (vii) |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 10 | 1 | 3747.0 | 3747.0 | 0.00 | 3747.0 | 0.00 | 8.1 | - |
| | | 2 | 2756.0 | 2756.0 | 0.00 | 2756.0 | 0.00 | 10.2 | - |
| | | 3 | 4102.0 | 4102.0 | 0.00 | 4102.0 | 0.00 | 7.8 | - |
| | | 4 | 3546.0 | 3555.0 | 0.25 | 3555.0 | 0.00 | 11.1 | - |
| | | 5 | 4040.0 | 4040.0 | 0.00 | 4040.0 | 0.00 | 7.4 | - |
| | | avg. | 3638.2 | 3640.0 | 0.05 | 3640.0 | 0.00 | 8.9 | |
| | 14 | 1 | 3943.6 | 3957.0 | 0.34 | 3957.0 | 0.00 | 18.1 | - |
| | | 2 | 3903.2 | 3917.0 | 0.35 | 3917.0 | 0.00 | 12.5 | - |
| | | 3 | 4332.1 | 4334.0 | 0.04 | 4334.0 | 0.00 | 14.4 | - |
| | | 4 | 3901.0 | 3901.0 | 0.00 | 3901.0 | 0.00 | 16.5 | - |
| | | 5 | 3379.2 | 3406.0 | 0.79 | 3406.0 | 0.00 | 20.9 | - |
| | | avg. | 3891.8 | 3903.0 | 0.31 | 3903.0 | 0.00 | 16.5 | |
| 15 | 15 | 1 | 5108.0 | 5136.0 | 0.55 | 5136.0 | 0.00 | 23.4 | - |
| | | 2 | 5128.7 | 5334.0 | 4.00 | 5334.0 | 0.00 | 51.4 | - |
| | | 3 | 4372.3 | 4381.0 | 0.20 | 4381.0 | 0.00 | 21.0 | - |
| | | 4 | 5576.0 | 5592.0 | 0.29 | 5592.0 | 0.00 | 26.1 | - |
| | | 5 | 4316.1 | 4320.0 | 0.09 | 4320.0 | 0.00 | 23.1 | - |
| | | avg. | 4900.2 | 4952.6 | 1.03 | 4952.6 | 0.00 | 29.0 | |
| | 21 | 1 | 5384.6 | 5416.0 | 0.58 | 5416.0 | 0.00 | 283.3 | SI(1) |
| | | 2 | 5914.9 | 5949.0 | 0.58 | 5949.0 | 0.00 | 100.1 | - |
| | | 3 | 6236.1 | 6277.0 | 0.66 | 6283.0 | 0.10 | 1080.7 | SI(2) |
| | | 4 | 8665.9 | 8691.0 | 0.29 | 8691.0 | 0.00 | 53.7 | - |
| | | 5 | 10159.6 | 10205.0 | 0.45 | 10205.0 | 0.00 | 71.2 | - |
| | | avg. | 7272.2 | 7307.6 | 0.51 | 7308.8 | 0.02 | 315.8 | |

SI($n$) indicates that stockyard infeasibility cuts are added where $n$ denotes the number of these cuts.

Table 5.3: Results of LBBD-IDBT for a terminal with 3 reclaimers - continued

| $\|V\|$ | $\|J\|$ | ins. | (i) | (ii) | (iii) | (iv) | (v) | (vi) | (vii) |
|---|---|---|---|---|---|---|---|---|---|
| 20 | 20 | 1 | 8665.8 | 8691.0 | 0.29 | 8691.0 | 0.00 | 54.0 | - |
| | | 2 | 10159.6 | 10205.0 | 0.45 | 10205.0 | 0.00 | 71.2 | - |
| | | 3 | 11563.9 | 11589.0 | 0.22 | 11589.0 | 0.00 | 49.6 | - |
| | | 4 | 8977.0 | 8977.0 | 0.00 | 8977.0 | 0.00 | 63.0 | - |
| | | 5 | 9893.0 | 10096.0 | 2.05 | 10096.0 | 0.00 | 101.5 | - |
| | | avg. | 9851.9 | 9911.6 | 0.60 | 9911.6 | 0.00 | 67.0 | |
| | 28 | 1 | 9279.3 | 9315.0 | 0.38 | 9315.0 | 0.00 | 2014.9 | - |
| | | 2 | 10533.6 | 10623.0 | 0.84 | 10623.0 | 0.00 | 816.8 | - |
| | | 3 | 10162.6 | 10292.0 | 1.27 | 10292.0 | 0.00 | 548.5 | - |
| | | 4 | 10944.5 | 11123.0 | 1.63 | 11123.0 | 0.00 | 1530.6 | - |
| | | 5 | 9201.0 | 9286.0 | 0.92 | 9286.0 | 0.00 | 3124.5 | - |
| | | avg. | 10024.2 | 10127.8 | 1.01 | 10127.8 | 0.00 | 1607.1 | |
| 25 | 25 | 1 | 18628.9 | 18645.0 | 1.21 | 18645.0 | 0.00 | 115.3 | - |
| | | 2 | 14646.0 | 14672.0 | 1.19 | 14672.0 | 0.00 | 105.0 | - |
| | | 3 | 14420.8 | 14463.0 | 1.08 | 14463.0 | 0.00 | 164.9 | - |
| | | 4 | 16222.0 | 16222.0 | 1.12 | 16222.0 | 0.00 | 105.3 | - |
| | | 5 | 19392.8 | 19742.0 | 1.15 | 19742.0 | 0.00 | 112.1 | - |
| | | avg. | 16662.1 | 16748.8 | 1.15 | 16748.8 | 0.00 | 120.5 | |
| | 35 | 1 | 19507.3 | 19603.7 | 0.49 | 19798.0 | 0.98 | 3600.0 | T.L. |
| | | 2 | 14968.4 | 14487.9 | 0.13 | 15002.0 | 0.09 | 3600.0 | T.L. |
| | | 3 | 18277.4 | 18306.0 | 0.16 | 18336.0 | 0.16 | 3600.0 | T.L. |
| | | 4 | 13053.0 | 13122.0 | 0.53 | 13122.0 | 0.00 | 3019.8 | - |
| | | 5 | 18273.0 | 18286.0 | 0.07 | 18306.0 | 0.11 | 3600.0 | T.L. |
| | | avg. | 16815.8 | 16267.1 | 0.27 | 16867.1 | 0.20 | 3484.0 | |

T.L. refers that instance is terminated by the time limit.

SI($n$) indicates that stockyard infeasibility cuts are added where $n$ denotes the number of these cuts.

Table 5.4: Comparison of LBBD-IDBT and monolithic MIP model

| | | | $4\,\|M\|, 4\,\|C\|$ | | $4\,\|M\|, 3\,\|C\|$ | | | | | $4\,\|M\|, 4\,\|C\|$ | | $4\,\|M\|, 3\,\|C\|$ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\|V\|$ | $\|J\|$ | ins. | time LBBD | time MIP | time LBBD | time MIP | $\|V\|$ | $\|J\|$ | ins. | time LBBD | time MIP | time LBBD | time MIP |
| 10 | 10 | 1 | 11.8 | 12.2 | 8.1 | 10.5 | 20 | 20 | 1 | 75.4 | T.L. | 54.0 | 3431.1 |
| | | 2 | 13.0 | 9.4 | 10.2 | 7.9 | | | 2 | 109.6 | 2791.9 | 71.2 | 2928.5 |
| | | 3 | 9.5 | 12.0 | 7.8 | 8.0 | | | 3 | 76.8 | 130.6 | 49.6 | 111.6 |
| | | 4 | 14.7 | 8.5 | 11.1 | 8.9 | | | 4 | 74.0 | 99.7 | 63.0 | 236.2 |
| | | 5 | 8.9 | 10.9 | 7.4 | 8.1 | | | 5 | 72.5 | T.L. | 101.5 | T.L. |
| | 14 | 1 | 28.0 | 141.1 | 18.1 | 110.0 | | 28 | 1 | 252.2 | T.L. | 2014.9 | T.L. |
| | | 2 | 27.8 | 27.0 | 12.5 | 37.4 | | | 2 | 434.1 | T.L. | 816.8 | T.L. |
| | | 3 | 22.6 | 17.0 | 14.4 | 17.4 | | | 3 | 1898.1 | T.L. | 548.5 | T.L. |
| | | 4 | 29.4 | 19.9 | 16.5 | 17.2 | | | 4 | 290.6 | T.L. | 1530.6 | T.L. |
| | | 5 | 31.0 | 33.9 | 20.9 | 29.5 | | | 5 | 1415.4 | T.L. | 3124.5 | T.L. |
| 15 | 15 | 1 | 35.8 | 130.5 | 23.4 | 159.5 | 25 | 25 | 1 | 186.2 | T.L. | 115.3 | T.L. |
| | | 2 | 63.0 | T.L. | 51.4 | 3015.0 | | | 2 | 175.3 | T.L. | 105.0 | 1173.5 |
| | | 3 | 21.5 | 30.3 | 21.0 | 626.4 | | | 3 | 762.5 | T.L. | 164.9 | 3507.4 |
| | | 4 | 40.1 | 1838.7 | 26.1 | 44.7 | | | 4 | 243.0 | T.L. | 105.3 | 162.4 |
| | | 5 | 37.2 | 128.7 | 23.1 | 145.0 | | | 5 | 146.3 | T.L. | 112.1 | 3185.8 |
| | 21 | 1 | 104.2 | T.L. | 283.3 | T.L. | | 35 | 1 | T.L. | T.L. | T.L. | T.L. |
| | | 2 | 101.9 | T.L. | 100.1 | T.L. | | | 2 | 2670.2 | T.L. | T.L. | T.L. |
| | | 3 | 143.0 | T.L. | 1080.7 | T.L. | | | 3 | 846.0 | T.L. | T.L. | T.L. |
| | | 4 | 97.4 | T.L. | 53.7 | T.L. | | | 4 | 1160.8 | T.L. | 3019.8 | T.L. |
| | | 5 | 331.5 | T.L. | 71.2 | T.L. | | | 5 | 2562.8 | T.L. | T.L. | T.L. |

T.L. refers that instance is terminated by the time limit.

Table 5.5: Comparison of the complexity of problem instances

| | Monolithic Formulation | Master Problem Formulation |
|---|---|---|
| 10 vessels-<br>10 stockpiles-<br>1 week | 1,960 variables<br>68,110 constraints | 1,070 variables<br>5,360 constraints |
| 25 vessels-<br>35 stockpiles-<br>2 weeks | 14,875 variables<br>1,458,365 constraints | 8,820 variables<br>194,480 constraints |

length of stockpiles that are stored at the stockyard over the planning horizon. So it takes less time to solve 20-vessel and 20-stockpile instances compared to 15-vessel and 21-stockpile instances since handling of stockpiles in 20-vessel instances spread over 1.5 week, rather than one week. Moreover, the LP relaxation of the master problem for the instances that are solved within the time limit is as tight as 0.5% on average. For the remaining instance, the optimality gap is observed as 0.25%.

Having a terminal with the same number of discrete berths and reclaimers makes sense because the reclaiming of stockpiles can only be performed if all related vessels are moored. On the other hand, the existence of tidal time windows may cause additional waiting for vessels even though all of their loading operations are completed. In such a situation, one or more reclaimers stay idle. For this reason, we also consider the case where the number of reclaimers is less than the number of berths. Table 5.4 presents the results of instances with 4 berths and 3 reclaimers. It can be observed that 36 instances out of 40 are solved within the time limit. In four other instances, the initial master problem cannot be solved within an hour, and the algorithm is terminated with an average optimality gap of 0.25%. In line with the results of instances with 4 berths and 4 reclaimers, the average solution time for instances with higher intensity are strictly larger. If an instance is terminated by the time limit,

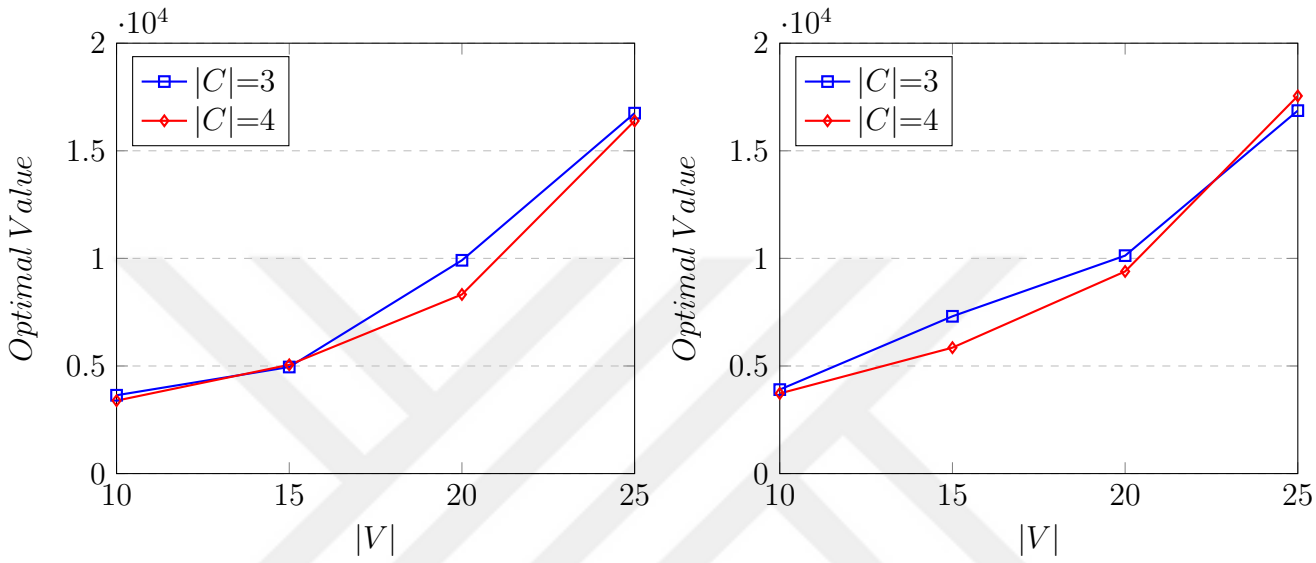
then columns (ii) and (iv) show lower and upper bound values at the termination, respectively.

These tables indicate that in 73 out of 75 total instances which are solved to optimality, optimal solution value equals to the lower bound of the algorithm, i.e., the optimal solution value of the master problem at the first iteration. One of the most important reasons of the success of the method is its master problem. In the IDBT problem, the duration of stay of a vessel is mainly determined by tidal windows and reclaimer schedules. Therefore, having tidal time windows constraints, vessel to reclaimer assignments, a relaxation of yard allocation problem, and non-overlapping constraint for reclaimers in the master problem allows us to estimate the duration of stay of vessels very accurately. As a result, the rest of the problem (after solving the master problem) turns into finding a feasible allocation of tasks to the stockyard with respect to subproblem constraints. While the master problem is still hard to solve, proposed decomposition structure allows finding a feasible solution to IDBT problem without adding any cuts in most of the problem instances.

Note that the instances with $|J|/|V|$=1 mimics the less congested terminals compared to the those with $|J|/|V|$=1.4. In other words, the average number of tasks to be handled at the same planning period is higher when some vessels have multiple stockpiles to be loaded. Tables 5.2 and 5.3 suggest that this difference also affects the solution performance of the method as less congested instances are easier to solve.

In Figure 5.4, we compare the average objective function values for the experiments with 3 and 4 reclaimers to evaluate the marginal improvement obtained by an additional reclaimer. Note that we study a terminal with 4 discrete berths with a single ship loader each, and a reclaimer can serve to a single vessel at any time. Because of tidal windows, some vessels should wait at the berth even if their loading is completed, which reduces the utilization of reclaimers especially if the number of reclaimers equals to the number of berths. Therefore, it might be sufficient to have a lesser number of reclaimers. Figure 5.4 shows that we can only obtain a slight improvement by adding the forth reclaimer to the stockyard. For this reason, this strategic decision on the

Figure 5.4: Comparison of the optimal values with 3 and 4 reclaimers




(a) 1 stockpile per vessel

(b) 1.4 stockpiles per vessel

stockyard configuration should be carefully evaluated by the terminal management as a reclaimer is a very expensive equipment.

Out of 80 instances, in only three instances the algorithm adds cut(s) to the master problem and all of them are stockyard infeasibility cuts. In addition to this, there may be a necessity for cuts in five instances in which the algorithm is terminated by the time limit. As expected, we only observe cuts in instances with intense workloads since finding a feasible solution that distributes stockpiles to pads is getting harder when the number of total stocking positions needed by the stockpiles is very near to the total number of stocking positions at the stockyard.

Table 5.5 compares the master problem with the monolithic MIP in terms of approximate number of variables and constraints for the largest and the smallest problem instances. One can observe that the proposed decomposition structure significantly reduce these numbers, especially those of constraints.

The decomposition structure also leads to the following interesting result regarding the computational experiments. That is, we did not observe a need for adding

cuts because of the violation of non-crossing constraint as we determine the stacking positions of stockpiles by also taking this constraint into account in the subproblem. In addition to that, there is a mechanism of LBBD-IDBT that helps to prevent the algorithm from adding non-crossing cuts in the congested instances. We observe that if we send start time of reclaiming stockpiles to the subproblem, we frequently obtain infeasible solutions because of non-crossing constraint. As we discuss, exact reclaimer schedules that define start and end times of reclaiming stockpiles determined by the master problem are not sent to the subproblem. This allows subproblem to avoid violating this constraint by switching the reclaiming order of tasks that are assigned to the same vessel without changing departure time of the vessel.

We can still generate some extreme instances that require one or more non-crossing cuts. Specifically, if we set the length of tasks (stockpiles) as near to the length of a pad, we observe some cases that given the optimal solution to the master problem, there is no feasible solution for the subproblem that satisfies non-crossing constraint. However, in reality, pads are long and tasks are relatively short based on the definition of a task (a stockpile of one type of coal to be loaded to a single vessel).

For such extreme instances, we can extend the cut generation procedure as follows. By constraint (5.43) of the subproblem, if stockpiles $i_1$ and $i_2$ such that $K_{i_1} + K_{i_2} > U$ are assigned to the reclaimers mounted on the same rail track, then their reclaiming cannot overlap in time. Therefore, if the subproblem is infeasible as well as $SIM_1$ and $SIM_2$ are both feasible, we should check whether this situation occurring in the solution to the master problem. This would be the proof of infeasibility and must be prevented in the master problem by a conditional cut as in the case of (5.51).

For the case with continuous berth structure, we modify problem instances by normalizing the distance between a reclaimer and a berthing position, as well as the length of vessels according to the total berth length, $\sum_m G_m$. Results show that we observe the same or slightly different objective values for most of the instances. For this reason, we do not present the results specifically. In dry bulk terminals, the number of vessels that can be loaded simultaneously is often depends to the

limited number ship loaders, rather than the length of the berth area. That is, while continuous berth structure provides a better utilization of the berth area as we can place more vessels than the number of discrete berth areas, the number of vessels that can be handled at a time is still limited by the number of ship loaders. That explains why the continuous berth structure cannot provide a considerable improvement to the IDBT problem with discrete BAP.

## *5.6 Conclusion*

Bulk cargoes constitute more than 80% of the total maritime transport in volumes. However, the research on its operational problem is typically disregarded until recently. In this study, we consider an integrated problem of dry bulk terminals which consists of operational problems of berth allocation, reclaimer scheduling and yard allocation. Our study is motivated by an export coal terminal, but it is also valid for other common dry bulk loads which are operated similarly.

We consider distinctive properties of these problems that include tidal time windows (Du et al., 2015), a realistic stockyard structure which consists of pads and rail tracks (van Vianen et al., 2015), multiple loading tasks for vessels (Boland et al., 2012), multiple reclaimers on a single rail track that require consideration of non-crossing constraint (Angelelli et al., 2016), and travel times between stockyard and berth area (Robenek et al., 2014). To the best of our knowledge, this is the most detailed and comprehensive exact solution approach that integrates operational problems of dry bulk terminals.

For this problem, we develop a novel LBBD algorithm (LBBD-IDBT) in which a master problem and a subproblem are modeled by using mixed integer programming (MIP) and constraint programming (CP), respectively. We identify some strong relationships among individual problems and decompose them into master and subproblems such that the complementary strengths of MIP and CP are exploited.

Accordingly, we model the master problem as a detailed optimization problem that minimizes weighted total departure times of vessels, by considering berth allo-

cation with tidal windows as well as schedules for reclaiming each stockpile. Even though the master problem is still not easy to solve, it estimates departure times very accurately, while an embedded relaxation of the reclaimer scheduling that uses time-indexed variables decreases its solution time. We model the subproblem as a feasibility problem with constraint programming to find a feasible schedule for reclaimer schedules and yard allocations given mooring and departure time of vessels. CP is not only very strong in feasibility problems (Hooker, 2012), but it is also shown to be a viable tool to cope with non-crossing constraint which is computationally expensive (Unsal and Oguz, 2013). Computational experiments show that LBBD-IDBT is able to solve problem instances of realistic size (up to 25 vessels, 28 stockpiles, 4 berths, 4 reclaimers and 60 stock positions) in a considerably short time.

# Chapter 6

# CONCLUSION

Maritime terminals are complex facilities in which a set of logistic processes are performed. As the volume of maritime transport is rapidly increasing, maritime terminals around the world face with a high competitive pressure. Therefore, it is essential for them to improve their performance levels in terms of service rate and costs. At that point, operations research methods provide opportunities to these complex facilities for managing their operations efficiently (Steenken et al., 2004).

Operational problems observed in both container and bulk terminals are the variants of well studied operations research problems, however, they are more complex in nature because of their distinctive characteristics. One of the most challenging characteristics is caused by the fact that the bottleneck equipment in both classes of terminals, namely, quay cranes (QC) and reclaimers, are mounted on the same rail track. Hence, their movements are limited by their respective positions.

In this dissertation, we study three different but related planning and scheduling problems of maritime terminals with a practical relevance: quay crane assignment problem (QCAP), reclaimer scheduling problem (RSP), and integrated dry bulk terminal (IDBT) problem. Note that all these operational problems involve the characteristic that we described above. In the following, we conclude the dissertation by emphasizing our contributions and discussing future research directions.

After presenting a concise overview of maritime terminals and their operations in the introductory chapter, in Chapter 2, we studied QCAP that allocates QCs to arriving vessels. Handling time of a vessel depends on the number of assigned QCs but this relation is not linear because of the crane interference. As QCs are the bottleneck equipment in container terminals, they need to be utilized effectively. If we ignore the

features of the problem such that unavailable time windows and different processing rates of QC, we cannot estimate vessel handling times accurately and experience a loss of productivity as a result. Up to date, the literature deals with the simplest form of the problem, and this is the first study that consider QCAP with the various problem features. We represented the QCAP as a variant of a moldable task scheduling problem (MTSP) with contiguous assignments of machines, and developed an extended mixed-integer programming (MIP) formulation that specifies QC to vessel assignments. As we show throughout the Chapter 2, this formulation is flexible in terms of modeling above mentioned features. However, it has multiple computational issues mainly due to the fact that the variables are time-indexed. Therefore, we first hybridized the formulation with a set of auxiliary variables to obtain a decomposable structure and then implemented a novel logic-based Benders decomposition (LBBD).

Compared to the original formulation which can only solve very small problem instances, the master problem has even higher number of variables and constraints. At the same time, the proposed decomposition allows us to solve the instances of realistic sizes by not only eliminating complicating constraints from the master problem, but also resulting in a very easy subproblem that can be iteratively solved to derive a strong Benders cut.

The resulting solution procedure provides a generic framework that allows implementing different features of the problem straightforwardly. Therefore, we extended LBBD with contiguous assignments of QCs, QC availability, and uniform QCs. Then, we presented a realistic terminal scenario in which a combination of these properties are simultaneously considered. Computational experiments showed that the proposed approach has provided a very tight linear programming relaxation for the base case (i.e., MTSP with contiguous assignments) and all of the extensions. This is the result of the novel decomposition strategy that considers all variables within the master problem, thus we are able to estimate the feasible space of the original problem accurately. We showed that almost all instances of MTSP with contiguous assignments were solved within the first iteration of the LBBD, while approximately 20% of the

instances with uniform machines and unavailable time windows required one or more cuts. Experiments also indicated the robustness of the proposed approach, since it can cope with a different objective and speed up functions.

Future work should consider the integration of the detailed QCAP that considers various problem features with the problem of allocation of vessels to berths as these two problems are highly interrelated. As in the case of a terminal with a very limited berth area, we can extent the proposed LBBD such that the total length of vessels moored at the same time to be less than or equal to the berth length in the master problem, while specific berthing positions are determined by the subproblem. Moreover, the proposed mathematical model has a symmetric structure because of the machine index. That is why we evaluate high number of nodes of the search tree to solve the master problem to optimality, even though the linear programming relaxation is very tight. This structure can be further investigated to reduce the negative effects of the symmetry.

The proposed exact solution method in Chapter 2 has one important drawback: It requires excessive memory to solve large problem instances because of the time-indexed decision variables. For this reason, in Chapter 3, we proposed solution methods to derive lower and upper bounds for the larger instances of QCAP.

While time-indexed formulations are large in size, they provide very tight linear programming relaxations. Hence, to obtain a tight lower bound, we took the advantage of the extended formulation that has specified task to machine assignments and developed a column generation procedure in which pricing problem was solved in polynomial time. We also discuss the implementation of branch and price over the proposed column generation procedure. We provided a novel framework that synchronizes two types of problem variables to the model on demand. However, we did not implement it for QCAP because, given the solution times of the linear programming relaxations of large instances by CG, at best, we can only expect a marginal improvement on the maximum instance size that can be solved within the time limit of an hour.

We should note that the proposed framework better suits for solving the variants of discrete time resource trade-off problem (DTRTP). One might think that we can also utilize LBBD-MTSP to solve this type of problems. In that scheduling environment, there are strong precedence relationships among tasks. Unfortunately, the constraint that ensures these relationships in time-indexed formulations is computationally inefficient. Hence, the success of the LBBD-MTSP cannot be replicated to DTRTP. On the other hand, we can cope with precedence relationships with the proposed column generation procedure as they can be modeled without destroying the special structure of the pricing problem.

For generating an upper bound for QCAP, we introduced a constraint programming model that has generated near optimal solutions within a short time for large instances, even though it failed to find optimal solutions. The ability of the model to find quality upper bounds in within a minute can be further utilized within exact methods by integrating CP and MIP methods. For example, we can first solve CP model and give the resulting solution to MIP as an initial solution (warm-start). Furthermore, the fact that proposed CP model is not being able to find optimal values while providing near optimal values in average is definitely worth investigating. In this study, we performed this set of experiments by the default search settings of CP Optimizer. We can also investigate the structure of optimal solutions and then use a user-defined search strategy to search tree more effectively.

In the rest of the dissertation, we studied the operations of a dry bulk terminal. In dry bulk terminals, cargo is stacked as a stockpile and is handled by reclaimers. As in the case of container terminals, the bottleneck handling equipment, i.e. a reclaimer, is located on the same rail track, albeit with a different configuration. Specifically, there are multiple parallel rail tracks, and on each rail track two reclaimers can operate simultaneously.

In Chapter 4, we studied the reclaimer scheduling problem. In this variant of a parallel machine scheduling problem, reclaimers cannot cross each other, and a reclaimer can only handle the stockpiles located adjacent to its rail track. As the problem is

strongly NP-hard, the mixed integer programming model that we formulated can only solve the small instances. Therefore, we opted for a heuristic approach by developing a constraint programming model. To generate tight lower bounds, we proposed an arc-time-indexed model by relaxing the non-crossing and non-interference requirements of reclaimers. By this way, we showed that the proposed CP model generates near optimal solutions for different stockyard configurations.

Advanced reclaimers can also perform the stacking of stockpiles. Stacking of a stockpile takes significantly lesser time than the reclaiming of the same stockpile. At the same time, if this operation is ignored then we might generate infeasible reclaimer schedules, as stockpiles are often stacked and reclaimed dynamically during the planning horizon. For this purpose, we also studied the extension of RSP in which we assumed each stockpile to be stacked by a reclaimer before its reclaiming has started.

One important observation is that the throughput and the utilization of reclaimers are affected by the design of the stockyard. Design parameters for the stockyard include but not limited to the placement of rail tracks, stocking pads, and reclaimers. The proposed methods within this chapter can be further utilized to develop managerial insights on the effective design of the stockyard.

Operational problems in dry bulk terminals are highly interrelated. For example, reclaimers' schedules directly affect the loading time of the vessels and hence the time that a vessel can leave the berth. Similarly, decisions on the positioning of stockpiles at the stockyard affect reclaimer scheduling since they generate non-eligibility of some stockpile-reclaimer assignments. If we solve these problems sequentially by ignoring their relations we often end up with plans with poor overall quality (Bierwirth and Meisel, 2010). Accordingly, in Chapter 5, we studied the integrated problem of dry bulk terminals, by considering berth allocation, yard assignment, and reclaimer scheduling operations simultaneously. We considered distinctive properties of these problems that included tidal time windows, a realistic stockyard structure which consists of pads and rail tracks, multiple loading tasks for vessels, multiple reclaimers

on a single rail track that require consideration of non-crossing constraint, and travel times between stockyard and berth area.

We developed a novel LBBD algorithm (LBBD-IDBT) in which a master problem and a subproblem were modeled by using mixed integer programming (MIP) and constraint programming, respectively. We identified some strong relationships among individual problems and decomposed them into the master and subproblems such that the complementary strengths of MIP and CP were exploited. To the best of our knowledge, this is the most detailed and comprehensive exact solution approach that integrates operational problems of dry bulk terminals. Results showed that LBBD-IDBT was able to solve considerably large instances to optimality in an acceptable time, compared to the monolithic approach.

In IDBT problem, we disregarded the scheduling of stacking operations, and assumed that the stockpiles were stacked immediately once they arrived to the terminal. On the other hand, in some terminals, stacking operations are also performed by reclaimers. In that case, reclaimer schedules should address both stacking and reclaiming operations as we mentioned in Chapter 4. Accordingly, future work should consider extending the integrated problem with stacking operations in such terminals.

# BIBLIOGRAPHY

[1] Alfares, H., Bailey, J. (1997). Integrated project task and man power scheduling. IIE Transactions, 29(9), 711-717.

[2] Angelelli, E., Kalinowski, T., Kapoor, R., Savelsbergh, M.W.P. (2016). A reclaimer scheduling problem arising in coal stockyard management. Journal of Scheduling, 19(5), 563-582.

[3] Apt, K.R. (2003). Principles of constraint programming. Cambridge University Press, London.

[4] Azizoglu, M., Kirca, O. (1999). On the minimization of total weighted flow time with identical and uniform parallel machines. European Journal of Operational Research, 113(1), 91-100.

[5] Baptiste, P., Jouglet, A., Savourey, D. (2008). Lower bounds for parallel machine scheduling problems, International Journal of Operational Research, 3(6), 661-682.

[6] Baptiste, P., Le Pape, C., Nuijten, W. (2001). Constraint-based scheduling. Kluwer Academic Publishers, Boston.

[7] Barnhart,C.E., Johnson, L., Nemhauser, G.L., Savelsbergh, M.W.P., Vance, P.H. (1998). Branch and price: column generation for solving huge integer programs. Operations Research, 46, 316-329.

[8] Barros, V.H., Costa, T.S., Oliveira, A.C.M., Lorena, L.A.N. (2011). Model and heuristic for berth allocation in tidal bulk ports with stock level constraints. Computers and Industrial Engineering, 60, 606-613.

[9] Belouadah, H., Potts, C. N. (1994). Scheduling identical parallel machines to minimize total weighted completion time. Discrete Applied Mathematics, 48(3), 201-218.

[10] Benders, J.F. (1962). Partitioning procedures for solving mixed-variables programming problems. Numerische Mathematik, 4(1), 238-252.

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

[12] Bierwirth, C., Meisel, F. (2015). A follow-up survey of berth allocation and quay crane scheduling problems in container terminals. European Journal of Operational Research, 244, 675-689.

[13] Bladek, I., Drozdowski, M., Guinand, F., Schepler, X. (2013). On contiguous and non-contiguous parallel task scheduling. Journal of Scheduling, 18(5), 487-495.

[14] Blazewicz, J., Cheng, T.C.E., Machowiak, M., Oguz, C. (2010). Berth and quay crane allocation: a moldable task scheduling model. Journal of Operational. Research Society, 62(7), 1189-1197.

[15] Boland, N., Gulczynski, D., Savelsbergh, M.W.P. (2012). A stockyard planning problem. European Journal on Transportation and Logistics, 1(3), 197-236.

[16] Boland, N., Savelsbergh, M.W.P. (2012). Optimizing the Hunter Valley coal chain. In: Gurnani H, Mehrotra A, Ray S (eds) Managing Supply Disruptions: Theory and Practice of Managing Risk. Springer, London, 275-302.

[17] Brucker, P., Knust, S. (2012). Resource-constrained project scheduling. In: Complex scheduling. GOR-Publications. Springer, Berlin.

[18] Buhrkal, K., Zuglian, S., Ropke, S., Larsen, J., Lusby, R. (2011). Models for the discrete berth allocation problem: A computational comparison. Transportation Research Part E: Logistics and Transportation Review, 47(4), 461-473.

[19] Carlo, H. J., Vis, I. F. A., Roodbergen, K. J. (2013). Seaside operations in container terminals: Literature overview, trends, and research directions. Flexible Services and Manufacturing Journal, 1-39.

[20] Carlo, H. J., Vis, I. F. A., Roodbergen, K. J. (2014a). Storage yard operations in container terminals: Literature overview, trends, and research directions. European Journal of Operational Research, 235(2), 412-430.

[21] Carlo, H. J., Vis, I. F. A., Roodbergen, K. J. (2014b). Transport operations in container terminals: Literature overview, trends, research directions and classification scheme. European Journal of Operational Research, 236(1), 1-13.

[22] Cesaret, B., Oguz, C., Salman, F.S. (2012). A tabu search algorithm for order acceptance and scheduling problem. Computers and Operations Research, 39, 1197-1205.

[23] Chen, J.H., Lee, D.H., Cao, J. (2012). A combinatorial benders cuts algorithm for the quayside operation problem at container terminals. Transportation Research Part E: Logistics and Transportation Review, 48(1), 266-275.

[24] Chu, Y., Xia, Q. (2004). Generating Benders cuts for a class of integer programming problems. J. C. Regin, M. Rueher, eds. Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problem,.Lecture Notes in Computer Science, Vol. 3011. Springer, 127-141.

[25] Cire, A., Coban, E., Hooker, J.N. (2015). Logic based Benders decomposition for planning and scheduling: a computational analysis. In Bartak, R., Salido, M.A., eds., COPLAS Proceedings, 21-29.

[26] Coban, E., Hooker, J.N. (2013). Single facility scheduling by logic-based Benders decomposition. Annals of Operations Research, 210, 245-272.

[27] Dai, J., Lin, W., Moorthy, R., Teo, C.-P. (2008). Berth allocation planning optimization in container terminals. In: Tang, C.S., Teo, C.-P., Wei, K.-K. (eds.), Supply chain analysis: A Handbook on the Interaction of Information, System and Optimization. Springer, New York, 69-105.

[28] Daniels, R.L., Hua, S.Y., Webster, S. (1999). Heuristics for parallel-machine flexible resource scheduling problems with unspecified job assignment. Computers and Operations Research, 26, 143-155.

[29] Demeulemeester, E.L., Herroelen, W.S. (2002). Project scheduling - a research handbook. Kluwer Academic Publishers, Boston.

[30] De Reyck, B., Demeulemeester, E., Herroelen, W. (1998). Local search methods for the discrete time/resource trade-off problem in project networks. Naval Research Logistic Quarterly 45(6), 553-578.

[31] Demeulemeester, E.L., de Reyck, B., Herroelen, W.S. (2000). The discrete time/resource trade-off problem in project networks - A branch-bound approach. IIE Transactions, 32, 1059-1069.

[32] Dilworth, R.P. (1950). A decomposition principle for partially ordered sets. Annals of Mathematics, 51, 161-166.

[33] Drozdowski, M. (2009). Scheduling for parallel processing. Springer, New York.

[34] Drozdowski, M. (1996). Scheduling multiprocessor tasks - an overview. European Journal of Operational Research, 94, 215-230.

[35] Du. J., Leung, J.Y.T. (1989). Complexity of scheduling parallel task systems. SIAM Journal of Discrete Mathematics, 2(4), 473-487.

[36] Du, Y., Chen, Q., Lam, J.S.L., Xu, Y., Cao, J.X. (2015). Modeling impacts of tides and the virtual arrival policy in berth allocation. Transportation Science, 49(4), 939-956.

[37] Dutot, P.F., Mounie, G., Trystram, D. (2004). Scheduling parallel tasks - algorithms and complexity. In J. Y-T. Leung (ed.), Handbook of Scheduling: Algorithms, Models, and Performance Analysis, CRC Press, 26-1-26-24.

[38] Edis, E.B., Oguz, C., Ozkarahan, I. (2013). Parallel machine scheduling with additional resources: Notation, classification, models and solution methods. European Journal of Operational Research, 230(3), 449-463.

[39] Ernst, A.T., Oguz, C., Singh, G., Taherkhani, G. (2017). Mathematical models for the berth allocation problem in dry bulk terminals. Journal of Scheduling, 20, 1-15.

[40] Feitelson, D.G., Rudolph, L., Schwiegelshohn, U., Sevcik, K.C., Wong, P. (1997). Theory and practice in parallel job scheduling. In Job Scheduling Strategies for Parallel Processing (IPPS-1997), 1-34, Springer-Verlag, London.

[41] Golias, M.M., Saharidis, G.K., Boilé, M., Theofanis, S., Ierapetritou, M.G. (2009). The berth allocation problem: optimizing vessel arrival time. Maritime Economics and Logistics, 11(4), 358-377.

[42] Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy, A.H.G. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics, 5, 287-326

[43] Grigoriev, A., Sviridenko, M., Uetz, M. (2007). Machine scheduling with resource dependent processing times. Mathematical Programming Series B, 110, 209-228.

[44] Guan, Y., Xiao, W.-Q., Cheung, R.K., Li, C.-L. (2002). A multiprocessor task scheduling model for berth allocation: heuristic and worst-case analysis. Operations Research Letters, 30(5), 343-350.

[45] Golias, M. Portal, I., Konur, D., Kaisar, E., Kolomvos, G. (2014). Robust berth scheduling at marine container terminals via hierarchical optimization. Computers and Operations Research, 41, 412-422

[46] Gunther, E., Konig, F.G., Megow, N. (2014). Scheduling and packing malleable and parallel tasks with precedence constraints of bounded width. Journal of Combinatorial Optimization, 27(1), 164-181.

[47] Hansen, P., Oguz, C., Mladenovic, N. (2008). Variable neighborhood search for minimum cost berth allocation. European Journal of Operational Research, 191(3), 636-649.

[48] Hartmann, S., Briskorn D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. European Journal of Operational Research, 207,1-14.

[49] Hooker, J.N. (2007). Planning and scheduling by logic based Benders decomposition. Operations Research, 55, 588-602.

[50] Hooker, J.N, Ottoson, G. (2003). Logic based Benders decomposition. Mathematical Programming, 96, 33-60.

[51] Hooker, J. N. (2012). Integrated methods for optimization. Springer, New York.

[52] Hu, D., Yao, Z. (2012). Stacker-reclaimer scheduling in a dry bulk terminal. International Journal of Computer Integrated Manufacturing, 25, 1047-1058.

[53] Imai, A., Nishimura, E., Papadimitriou, S. (2001). The dynamic berth allocation problem for a container port. Transportation Research Part B, 35(4), 401-417.

[54] Imai, A., Sun, X., Nishimura, E., Papadimitriou, S. (2005). Berth allocation in a container port: using a continuous location space approach. Transportation Research Part B, 39(3), 199-221.

[55] Imai, A., Nishimura, E., Hattori, M., Papadimitriou, S. (2007). Berth allocation at indented berths for mega-containerships. European Journal of Operational Research, 179(2), 579-593.

[56] Jansen, K., Zhang, H. (2006). An approximation algorithm for scheduling malleable tasks under general precedence constraints. ACM Transactions- Algorithms, 2(3), 416-434.

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

[58] Kolen, A.W., Lenstra, J.K., Papadimitriou, C.H., Spieksma, F.C. (2007). Interval scheduling: A survey. Naval Research Logistics, 54(5), 530-543.

[59] Lee, D.-H., Chen, J. H., Cao, J. X. (2010). The continuous berth allocation problem: A greedy randomized adaptive search solution. Transportation Research Part E: Logistics and Transportation Review, 46(6), 1017-1029.

[60] Lee, D.-H., Jin, J. G., Chen, J. H. (2012). Terminal and yard allocation problem for a container transshipment hub with multiple terminals. Transportation Research Part E: Logistics and Transportation Review, 48(2), 516-528.

[61] Legato, P., Trunfio, R., Meisel, F. (2012). Modeling and solving rich quay crane scheduling problems. Computers and Operations Research, 39(9), 2063-2078.

[62] Leung, J.Y.-T., Li, C.L. (2008). Scheduling with processing set restriction: a survey. International Journal of Production Economics, 116(2), 251-262.

[63] Legato, P., Trunfio, R., Meisel, F. (2013). Modeling and solving rich quay crane scheduling problems. Computers and Operations Research, 39, 2063-2078.

[64] Lepére, R., Trystram, D., Woeginger, G. (2002). Approximation algorithms for scheduling malleable tasks under precedence constraints. International Journal of Foundations of Computer Science, 13(4), 613-627.

[65] Laborie, P., Rogerie, J., Shaw, P., Vilim, P. (2009). Reasoning with conditional time-intervals - part II: an algebraical model for resources. In: Proceedings of the 22nd International FLAIRS Conference, Sanibel Island, Florida.

[66] Lalla-Ruiz, E., Exposito-Izquierdo, C., Melian-Batista, B., Moreno-Vega, J.M. (2016). A set-partitioning-based model for the berth allocation problem under time-dependent limitations. European Journal of Operational Research, 250(3), 1001-1012.

[67] Li, H., Womer, K. (2009). Scheduling projects with multi-skilled personnel by a hybrid MILP/CP Benders decomposition algorithm. Journal of Scheduling, 12(3), 281-298.

[68] Li, K., Yang, S. (2009). Non-identical parallel-machine scheduling research with minimizing total weighted completion times: models, relaxations and algorithms. Applied Mathematical Modeling, 33, 21-45.

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

[70] Lim, A., Rodrigues, B., Xu, Z. (2007). A m-parallel crane scheduling problem with a non-crossing constraint. Naval Research Logistics, 54(2), 115-127.

[71] Lloyd's List Intelligence (2017). One hundred ports, https://lloydslist.maritimeintelligence.informa.com/one-hundred-container-ports-2017. Retrieved in 03/20/2018.

[72] Lodi, A., Martello, S., Monaci, M. (2002). Two-dimensional packing problems: a survey. European Journal of Operations Research, 141(2), 241-252.

[73] Lu, Z., Han, X., Xi, L., Erera A.L. (2012). A heuristic for the quay crane scheduling problem based on contiguous bay crane operations. Computers and Operations Research, 39(12), 2915-2928.

[74] Lustig, I.J., Puget, J.F. (2001). Program does not equal program: constraint programming and its relationship to mathematical programming. Interfaces, 31, 29-53.

[75] Ludwig, W. (1995). Algorithms for scheduling malleable and nonmalleable tasks. Ph.D. Thesis, Department of Computer Science, University of Wisconsin, Madison.

[76] Martello, S., Monaci, M., Vigo, D. (2003). An exact approach to the strip-packing problem. INFORMS Journal on Computing, 15, 310-319.

[77] Meisel, F. (2011). The quay crane scheduling with time windows. Naval Research Logistics, 58(7), 619-636.

[78] Meisel, F.,Bierwirth, C. (2013). A framework for integrated berth allocation and crane operations planning in seaport container terminals. Transportation Science, 47(2),131-147.

[79] Meisel, F., Bierwirth, C. (2009). Heuristics for the integration of crane productivity in the berth allocation problem. Transportation Research Part E, 45 (1), 196-209.

[80] Menezes, G.C., Mateus, G.R., Ravetic, M.G. (2017). A branch and price algorithm to solve the integrated production planning and scheduling in bulk ports. European Journal of Operational Research, 258(3), 926-937.

[81] Mika, M. Waligora, G., Weglarz, J. (2008). Tabu search for multi-mode resource-constrained project scheduling with schedule-dependent setup times. European Journal of Operational Research, 187(3), 1238-1250.

[82] Moccia, L., Cordeau, J.F., Gaudioso, M., Laporte, G. (2006). A branch-and-cut algorithm for the quay crane scheduling problem in a container terminal. Naval Research Logistics, 53(1), 45-59.

[83] Mounie, G., Rapine, C., Trystram, D. (1999). Efficient approximation algorithms for scheduling malleable tasks. In Proceedings of SPAA, 23-32, ACM, New York.

[84] Mounie, G., Rapine, C., Trystram, D. (2008). A 3/2-dual approximation algorithm for scheduling independent monotonic malleable tasks. SIAM Journal of Computing, 37(2), 401-412.

[85] Ranjbar, M., Kianfar F. (2007). Solving the discrete time/resource trade-off problem in project scheduling with genetic algorithms. Applied Mathematics and Computation, 191, 451-456.

[86] Ranjbar, M., de Reyck, B. Kianfar, F. (2009). A hybrid scatter search for the discrete time/resource trade-off problem in project scheduling. European Journal of Operational Research, 193, 35-48.

[87] Nishimura, E., Imai, A., Papadimitriou, S. (2001). Berth allocation planning in the public berth system by genetic algorithms. European Journal of Operational Research, 131, 282-292.

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

[89] Robenek, T., Umang, N., Bierlaire, M., Ropke, S. (2014). A Branch-and-price algorithm to solve the integrated berth allocation and yard assignment problem in bulk ports. European Journal of Operational Research, 235, 399-411.

[90] Schmidt, G. (2000). Scheduling with limited machine availability. European Journal of Operational Research, 121, 1-15.

[91] Shabtay, D., Steiner, G. (2007). A survey of scheduling with controllable processing times. Discrete Applied Mathematics,155, 1643-1666.

[92] Song, Y., Zhang, J., Liu, M., Chu, C. (2018). The berth allocation optimisation with the consideration of time-varying water depths. International Journal of Operations Research, 1-29.

[93] Stahlbock, R., Voss, S. (2008). Operations research at container terminals: a literature update. OR Spectrum, 30(1), 1-52.

[94] Steenkeen, D., Voss, S., Stahlbock, R. (2004). Container terminal operation and operations research - a classification and literature review. OR Spectrum, 26(1), 3-49.

[95] Talbot, F.B. (1982). Resource-constrained project scheduling with time-resource tradeoffs: the non-preemptive case. Management Science, 28(10), 1199-1210.

[96] Turek, J., Wolf, J.L., Yu P.S. (1992). Approximate algorithms for scheduling parallelizable tasks. In Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, 323-332, San Diego, California.

[97] Turkogullari, Y.B., Taskin, Z.C., Aras, N., Altinel, I.K. (2014). Optimal berth allocation and time-invariant quay crane assignment in container terminals. European Journal of Operational Research, 235(1), 88-101.

[98] Umang, N., Bierlaire, M., Vacca, I. (2013). Exact and heuristic methods to solve the berth allocation problem in bulk ports. Transportation Research Part E: Logistics and Transportation Review, 54, 14-31.

[99] UNCTAD (2017). Review of maritime transport, United Nations Conference on Trade and Development, http://www.unctad.org.

[100] Unlu Y., Mason S.Y., (2010). Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems. Computers and Industrial Engineering, 58, 785-800.

[101] Unsal, O. (2013). Constraint programming approach to quay crane scheduling problem. M.Sc. Thesis, Koc University, Istanbul.

[102] Unsal, O., Oguz, C. (2013). Constraint programming approach to quay crane scheduling problem. Transportation Research Part E: Logistics and Transportation Review, 59, 108-122.

[103] Van den Akker, J.M., Hurkens, C.A.J., Savelsbergh, M.W.P. (2000). Time-indexed formulations for machine scheduling problems: Column generation. Journal on Computing, 12(2), 111-124.

[104] Venturini, G., Iris, C., Kontovasb, C.A., Larsena, A. (2014). The multi-port berth allocation problem with speed optimization and emission considerations. Transportation Research Part D, 54, 142-159.

[105] Vredeveld, T., Hurkens, C. (2002). Experimental comparison of approximation algorithms for scheduling unrelated parallel machines. Journal on Computing, 14 (2), 175-189.

[106] Wang, T., Wang, X., Meng, Q. (2018) Joint berth allocation and quay crane assignment under different carbon taxation policies. Transportation Research Part B, 117, 18-36.

[107] Weglarz, J., Jozefowska, J., Mika, M., Waligora, G. (2011). Project scheduling with finite and infinite number of activity processing modes - A survey. European Journal of Operational Research, 208,177-205.

[108] WOS (2018). Web of Science, www.webofknowledge.com.

[109] Xu, D., Li, C.L., Leung, J.Y.T. (2012). Berth allocation with time-dependent physical limitations on vessels. European Journal of Operational Research, 216(1), 47-56.

[110] Yan, S., Tang, C.-H., Chen, M. (2004). A model and a solution algorithm for airport common use check-in counter assignments, Transportation Research Part A, 38, 101-125.

[111] Yang, C., Wang, X., Li, Z. (2012). An optimization approach for coupling problem of berth allocation and quay crane assignment in container terminal. Computers and Industrial Engineering, 63(1), 243-253.

[112] Zheng, J., Gao, Z., Yang, D., Z. Sun, Z. (2015). Network design and capacity exchange for liner alliances with fixed and variable container demands. Transportation Science, 49(4), 886-899.

Appendix A

# EQUIVALENCY OF DTRTP AND MTSP

**Definition** *Discrete Time-Resource Trade Off Problem (DTRTP) (De Reyck et al., 1998)* In DTRTP the duration of an activity is a discrete, non-increasing function of the amount of a single renewable resource committed to it. Given the specified work content $w_i$ for activity $i$, all $mode_i$ efficient execution modes for its execution are determined based on time/resource trade-offs. When performed in mode $m$ ($1 \leq m \leq mode_i$), activity $i$ has a processing time $d_{im}$ and requires a resource in a constant amount of $r_{im}$. A mode is called efficient if every other mode has either a higher duration or a higher resource requirement. Without loss of generality, we assume that the modes of each activity are sorted in the order of non-decreasing duration. We assume that the dummy start node $|J| + 1$ and the dummy end node $|J| + 2$ have a single execution mode with zero duration and zero resource requirement. The renewable resource has capacity of $\alpha$ ; that is, total resource usage of tasks 1 to $|J|$ must be less than or equal to $\alpha$ at any time. There may be precedence relations between tasks, and in such cases $prec_i^{'}$ denotes the set of immediate predecessors of task $i$.

**Claim 1** *MTSP is a special case of DTRTP.*

*Proof* To prove this claim, we must show that there exists a corresponding DTRTP instance for each instance of MTSP. Let $prec_i$ denotes the set of immediate predecessors of task $i$ in MTSP:

$|K| = \alpha$ , $|J|^{MTSP} = |J|^{DTRTP}$

$prec_i = prec_i^{'}$

Based on the definition, we should extend the parameter set of DTRTP with some auxiliary parameters without loss of generality. Let $q_i$ be a set of $r_{im}$ values of efficient modes of activity $i$ and $t_i$ be a set of $(d_{im}, r_{im})$ tuples of efficient mode $m$ of activity $i$, sorted in increasing order of $r_{im}$. Furthermore, let $z_{ik}$ be the processing time of an efficient mode of task $i$ that uses $k$ units of resources. For each resource usage value for task $i$, there is 0 or 1 corresponding $d_{im}$ value based on the definition of an efficient mode. Accordingly,

$\forall i \in \{1, \ldots, |J|^{DTRTP} + 2\}, \forall k \in \{1, \ldots, \alpha\}$

$$z_{ik} = \begin{cases} \infty, \ if \ k \notin q_i \\ d_{im}, \ if \ k \in q_i, k = r_{im}, k > 0 \\ 0, \ if \ k \in q_i, k = 0 \end{cases}$$

Then, the rest of corresponding MTSP instance is defined as follows.

$p_{il} = Z_{il} \ \forall i \in \{1, \ldots, |J|^{DTRTP} + 2\}, \forall k \in \{1, \ldots, a\}$

$|w_i| = max\{k | k \in q_i\}, \forall i \in \{1, \ldots, |J|^{DTRTP}\}$

$|w_{(J^{MTSP}+1)}| = 1, |w_{(J^{MTSP}+2)}| = 1$

Hence, each MTSP instance can be transformed into an equivalent DTRTP instance.

**Example** Consider a following MTSP instance with two tasks, four machines.

$|J| = 2, |K| = 4, |w_1| = 4, |w_2| = 2$

$p_{11} = \infty, p_{12} = \infty, p_{13} = 5, p_{14} = 4.$

$p_{21} = 40, p_{22} = 30, p_{23} = \infty, p_{24} = \infty.$

$prec_2 = \{1\}, prec_1 = \{\}$

Accordingly, corresponding DTRTP instance will be;

$|J|^{DTRTP} = 2, \alpha = 4$

$mode_1 = mode_2 = mode_3 = mode_4 = 4$

$t_1 = \{(\infty, 1), (\infty, 2), (5, 3), (4, 4)\}$

$t_2 = \{40, 1), (30, 2), (\infty, 3), (\infty, 4)\}$

$t_3 = \{(0, 1), (\infty, 2), (\infty, 3), (\infty, 4)\}$

$t_4 = \{(0,1),(\infty,2),(\infty,3),(\infty,4)\}$

$prec_1^{'} = \{3\}, prec_2^{'} = \{1,3\}, prec_3^{'} = \{\}, prec_4^{'} = \{1,2\}$

**Claim 2** *DTRTP is a special case of MTSP.*

*Proof* To prove this claim, we must show that there exist a corresponding MTSP instance for each instance of DTRTP:

$|J|^{DTRTP} = |J|^{MTSP}, \alpha = |K|$

$mode_i = |K| \forall i \in \{1,\ldots,|J|^{MTSP}\}$

$r_{il} = l \ \ \forall i \in \{1,\ldots,|J|^{MTSP}+2\}, \forall l \in \{1,\ldots,|K|\}$

$d_{il} = p_{il} \ \ \forall i \in \{1,\ldots,|J|^{MTSP}\}, \forall l \in \{1,\ldots,|K|\}$

$$d_{il} = \begin{cases} 0, \ if \ \forall i \in \{|J|^{MTSP}+1, |J|^{MTSP}+2\}, l = 1 \\ \infty, \ if \ \forall i \in \{|J|^{MTSP}+1, |J|^{MTSP}+2\}, \\ \qquad\qquad \forall l \in \{2,...,|K|\} \end{cases}$$

$prec_i^{'} = prec_i \cup \{|J|^{MTSP}+1\} \forall i \in \{1,\ldots,|J|^{MTSP}\}$

$prec_{(|J|^{DTRTP}+1)}^{'} = \{\}$

$prec_{(|J|^{DTRTP}+2)}^{'} = \{1,\ldots,|J|^{MTSP}\}$

Hence, each DTRTP instance can be transformed into an equivalent MTSP instance.

hence each multiprocessor scheduling problem instance

**Example** Consider a following DTRTP instance with two tasks, four units of resource.

$|J| = 2, \alpha = 4, mode_1 = 2, mode_2 = 3$

$t_1 = \{(6,3),(5,4)\}, t_2 = \{(40,1),(20,2),(15,3)\}, t_3 = \{(0,1)\}, t_4 = \{(0,1)\}$

$q_3 = q_4 = \{0\}, q_1 = \{3,4\}, q_2 = \{1,2,3\}$

$z_{11} = \infty, z_{12} = \infty, z_{13} = 6, z_{14} = 5$

$z_{21} = 40, z_{22} = 20, z_{23} = 15, z_{24} = \infty$

$z_{31} = 0, z_{32} = \infty, z_{33} = \infty, z_{34} = \infty$

$z_{41} = 0, z_{42} = \infty, z_{43} = \infty, z_{44} = \infty$

$prec_1^{'} = \{3\}, prec_2^{'} = \{3\}, prec_3^{'} = \{\}, prec_4^{'} = \{1, 2\}$

Therefore, corresponding MTSP instance will be;

$|J|^{MTSP} = 2, |K| = 4$

$|w_3| = |w_4| = 1, |w_1| = 4, |w_2| = 3$

$p_{11} = \infty, p_{12} = \infty, p_{13} = 6, p_{14} = 5$

$p_{21} = 40, p_{22} = 20, p_{23} = 15, p_{24} = \infty$

$p_{31} = 0, p_{32} = \infty, p_{33} = \infty, p_{34} = \infty$

$p_{41} = 0, p_{42} = \infty, p_{43} = \infty, p_{44} = \infty$

$prec_1 = \{3\}, prec_2 = \{3\}, prec_3 = \{\}, prec_4 = \{1, 2\}.$

**Proposition** *DTRTP and MTSP are equivalent problems.*

*Proof* Claims 1 and 2 together indicate that any problem instance of these problems can be converted into a corresponding instance of each other. Therefore, these problems are equivalent.

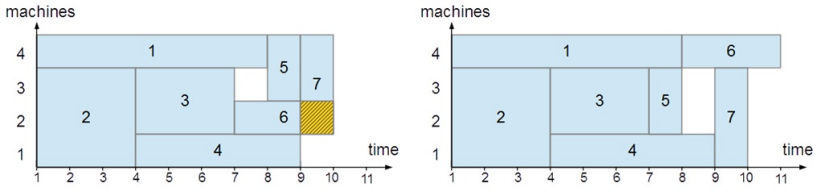
Appendix B

# MTSP WITH CONTIGUOUS ASSIGNMENTS: COUNTER EXAMPLE FOR PROPOSITION 6

Consider the following 7 tasks - 4 machines instance which is presented by Bladek et al. (2014) for multiprocessor task scheduling problem. In multiprocessor task scheduling, number of machines to process task $i$ is fixed and is denoted by $size_i$. MTSP is a generalization of multiprocessor task scheduling problem, hence each multiprocessor scheduling problem instance can be transformed into an equivalent MTSP instance by setting $p_{il} = p_i^{'}$ if $l = size_i$, and $p_{il} = \infty$ otherwise. We also modify this instance for total weighted completion time minimization objective by generating the following weights for each task.

$p_i^{'} = \{7, 3, 3, 5, 1, 3, 1\}, \quad w_i = \{34, 100, 50, 22, 13, 14, 10\}, \quad size_i = \{1, 3, 2, 1, 2, 1, 3\}$

When we solve this instance with the proposed LBBD, the resulting optimal solution has an objective value of 1577 (Figure B.01-a). However, this solution turns out to be infeasible for the subproblem. The optimal solution for this instance is obtained by the proposed LBBD after a single cut and has an objective value of 1578 (Figure B01-b) below.




(a) Solution in first iteration (b) Optimal solution

Figure B.01: Optimal solution to different iterations

# Appendix C

## IDBT PROBLEM: MONOLITHIC MIP FORMULATION

We add following variables and constraints to the formulation presented by (5.1)-(5.29) to obtain a monolithic model for IDBT problem. These new constraints simultaneously determine feasible yard allocations and reclaimer schedules based on these allocations with respect to non-crossing constraint.

**Additional Decision Variables:**

$V1_{ikp}$ binary decision variable that takes a value of 1 if stockpile $i$ is assigned to reclaimer $k$ and stocking position $p$, 0 otherwise.

$V2_{i_1 i_2}$ binary decision variable that takes a value of 1, if stockpile $i_1$ is reclaimed before stockpile $i_2$, 0 otherwise.

$V3_{isp}$ binary decision variable that takes a value of 1, if stockpile $i$ is assigned to pad $s$ and stocking position $p$, 0 otherwise.

$V4_{i_1 i_2}$ binary decision variable that takes a value of 1, if stockpiles $i_1$ and $i_2$ is allocated to the stockyard such that sharing at least one stocking position, and $i_1$ is reclaimed before the arrival of stockpile $i_2$, 0 otherwise.

**Additional Constraints:**

$$\sum_{k} \sum_{p} V1_{ikp} = 1 \qquad \forall i \quad \text{(C.1)}$$

$$V1_{ikp} + 1 \geq V3_{isp} + \Omega_{VesselOfTask_i,k} \qquad \forall i, \forall k, \forall s, \forall p \quad \text{(C.2)}$$

$$V2_{i_1 i_2} + V2_{i_2 i_1} \geq V1_{i_1 k_1 p} + \sum_{p_1 > p} V1_{i_2 k_2 p_1} - 1$$
$$\forall i_1, i_2, \forall k_1, k_2, \forall s, \forall p : i_1 \neq i_2, k_1 < k_2, Q_{k_1} = Q_{k_2}, p + K_{i_1} \leq |U| \quad \text{(C.3)}$$

$$CR_{i_1} \leq SR_{i_2} + |T|(1 - V2_{i_1 i_2}) \qquad \forall i_1, i_2 : i_1 \neq i_2 \quad \text{(C.4)}$$

$$\sum_p \sum_s V3_{isp} = 1 \qquad \forall i \quad \text{(C.5)}$$

$$V3_{isp} = 0 \qquad \forall i, \forall s, \forall p : p + K_i - 1 > |U| \quad \text{(C.6)}$$

$$1 - \sum_p \sum_{s:(Q_k > s)||(s - Q_k > 1)} V3_{isp} \geq \Omega_{VesselOfTask_i k} \qquad \forall i, \forall k \quad \text{(C.7)}$$

$$V4_{i_1 i_2} + V4_{i_2 i_1} \leq 1 \qquad \forall i_1, i_2 : i_1 \neq i_2 \quad \text{(C.8)}$$

$$V4_{i_1 i_2} + V4_{i_2 i_1} \geq V3_{i_1 sp} + \sum_{p_1 \in \{max(1, p - K_{i_2} + 1), \ldots, p + K_{i_1} - 1\}} V3_{i_2 s p_1} - 1$$
$$\forall i_1, i_2, \forall s, \forall p : i_1 \neq i_2, p + K_{i_1} - 1 \leq |U| \quad \text{(C.9)}$$

$$CR_{i_1} \leq R_{VesselOfTask_{i_2}} + |T|(1 - V4_{i_1 i_2}) \qquad \forall i_1, i_2 : i_1 \neq i_2 \quad \text{(C.10)}$$

$$V1_{ikp}, V2_{i_1 i_2}, V3_{isp}, V4_{i_1 i_2} \in \{0, 1\} \qquad \forall i, i_1, i_2, \forall s, \forall p, \forall k \quad \text{(C.11)}$$

Constraints (C.1)-(C.4) together ensure that reclaimers operating on the same rail track do not cross each other. By constraints (C.5)-(C.10), we state that if two stockpiles are allocated to the stockyard such that sharing at least one stocking position, then they cannot be stored at the stockyard simultaneously. Constraint (C.11) defines the domains of decision variables.

Appendix D

# IDBT PROBLEM: MODELS FOR DERIVING BENDERS CUTS

### Stockyard Infeasibility Model 1 ($SIM_1$)

As a part of Benders cut for infeasibility caused by allocation of stockpiles to pads, we introduce the following MIP model. This model checks whether it is possible to allocate given set of stockpiles to pads, without considering vessel to reclaimer assignments.

**Decision Variable:**

$\tau_{is}$ a binary variable that takes 1 if stockpile $i$ is allocated to pad $s$, 0 otherwise.

**Mathematical Model:**

$$\sum_{s} \tau_{is} = 1 \qquad \forall i \in S1CUT_h \quad \text{(D.1)}$$

$$\sum_{i \in SI1CUT_h} K_i \tau_{is} \leq |U| \qquad \forall s \quad \text{(D.2)}$$

$$\tau_{is} \in \{0, 1\} \qquad \forall i, s : i \in S1CUT_h \quad \text{(D.3)}$$

In this model, we assign each stockpile $i \in S1CUT_h$ to one of the stocking pads. Total length of stockpiles that are assigned to pad $s$ must be smaller than or equal to total stocking positions available in this pad.

**Stockyard Infeasibility Model 2 ($SIM_2$)**

We also introduce a more restricted MIP model than $SIM_1$, which checks whether it is possible to allocate given set of stockpiles to pads by considering vessel to reclaimer assignments determined by the master problem.

$\mu_{is}$ a parameter that has a value of 1 if stockpile $i \in S1CUT_h$ is assigned to a reclaimer which can serve pad $s$, 0 otherwise.

**Mathematical Model:**

$$\sum_{s:\mu_{is}=1} \tau_{is} = 1 \qquad \forall i \in S1CUT_h \quad \text{(D.4)}$$

$$\sum_{i \in SI1CUT_h : \mu_{is}=1} K_i \tau_{is} \leq |U| \qquad \forall s \quad \text{(D.5)}$$

$$\tau_{is} \in \{0,1\} \qquad \forall i, s : i \in S1CUT_h \quad \text{(D.6)}$$

In this model, we assign each stockpile $i \in S1CUT_h$ to one of the stocking pads by considering vessel-reclaimer assignments. Total length of stockpiles that are assigned to pad $p$ must be smaller than or equal to total stocking positions available in this pad.

Appendix E

# EVALUATION OF THE DECOMPOSITION STRUCTURE OF HOOKER (2007) FOR QCAP

In one of the most succesful implementations of logic-based Benders decomposition, Hooker (2007) considers planning and scheduling problem (PSP), which consists of assigning tasks to facilities (i.e., machines) and then scheduling these tasks on assigned facilities. This problem corresponds to a variant of parallel machine scheduling with uniform machines, release dates and deadlines. He studies minmax objectives, namely, makespan and maximum tardiness minimization.

The decomposition structure of Hooker (2007) is pretty straightforward yet effective as it exploits the respective advantages of mixed integer programming and constraint programming. In the master problem, he disregards the scheduling decisions and solves the problem of task to facility assignments via mixed integer programming. Let there are $|J|$ tasks and $|K|$ facilities. $X_{ij}$ is a binary decision variable that takes the value of 1 if task $i$ is assigned to machine $j$, and $p_{ij}$ represents the processing time of task $i$ and machine $j$. Then, the solution space of the master problem consists of $|J|^{|K|}$ possible task to machine assignments. Moreover, the relaxation of the scheduling subproblem is embedded into the master problem in terms of the master problem variables.

Given the optimal assignments determined by the master problem, the subproblem is to find an optimal solution for each machine, and it is modeled via constraint programming. The master and subproblems of the implementation of Hooker (2007) for the makespan minimization objective are as follows.

**Master Problem:**

minimize $C_{max}$

subject to:

$$\sum_j X_{ij} = 1 \qquad \forall i \quad \text{(E.1)}$$

$$Benders\ cuts \qquad \text{(E.2)}$$

$$C_{max} \geq \sum_i p_{ij} X_{ij} \qquad \forall j \quad \text{(E.3)}$$

$$X_{ij} \in \{0, 1\} \qquad \forall i, \forall j \quad \text{(E.4)}$$

**Subproblem for each machine $j$:**

Let $Q_i$ to be an optional interval variable that is active in the model if task $i$ is assigned to machine $j$. It has a domain $\{r_i, \ldots, d_i\}$ where $r_i$ and $d_i$ denote the release time and the deadline of task $i$, respectively.

minimize $\max_i Z_i$

subject to:

$$disjunctive(Z_i | \forall i : X_{ij} = 1) \qquad \text{(E.5)}$$

If the subproblem turns out to be infeasible, then optimality Benders cuts are added to the master problem for each machine. Hooker's implementation can be summarized as follows: the master problem only consists of assignment decisions, thus loosely projects the original problem. Then, for each machine, subproblem is solved very quickly to generate cuts. By this way, each iteration of LBBD is fast as the master and subproblems are easy to solve, but the procedure requires large number of cuts to be added to the master problem until it terminates with an optimal solution.

Differently from PSP, in our problem, a task can be processed by more than one machine at the same time, and the processing time depends on the number of assigned machines. In addition, machines that are assigned to a task must have consecutive indices (contiguous assignments). In the following, we implement LBBD to QCAP with makespan minimization objective by considering the decomposition structure of Hooker (2007). We select the test bed as the previously generated instances with $|K|$=10 and $|J|$=20,25,30,35,40.

As we consider contiguous assignment of machines and limit the number of machines that can be assigned to any task with $|K|/2$ , there are $\sum_{i=(|K|/2)-1}^{|K|}(i)$ possible assignments for each task. For $|K|$=10, there are 40 different assignments for each task. As a result the solution space of the master problem consists of $|J|^{|40|}$ assignments, compared to the $|J|^{|10|}$ for PSP.

In contrast to the LBBD implementation of PSP, the scheduling subproblem cannot be decomposed such that the problem for each machine is solved seperately. The reason is obvious: when a task is assigned to more than one machine, it should start and end at the same time on each machine. Therefore, the subproblem should solve the multiprocessor task scheduling problem given the fixed task to machine assignments. The master and subproblems then are presented as follows.

**Master Problem:**

Let $S_2$ is the set of contiguous machine assignments, e.g., $u = \{1, 2, 3\} \in S_2$. $Y_{iu}$ is a binary variable that takes the value of 1 if task $i$ is assigned to machine set $u \in S_2$, otherwise 0, and $\overline{p}_{il}$ represents the processing time of task $i$ when processed by $l$ machines at the same time. Furthermore, $A_{ju}$ is a parameter that has the value of 1 if $j$ is a member of machine set $u \in S_2$, otherwise 0.

minimize $C_{max}$

subject to:

$$\sum_{u} Y_{iu} = 1 \qquad \forall i \quad \text{(E.6)}$$

*Benders cuts* (E.7)

$$C_{max} \geq \sum_{i} \sum_{u} \overline{p}_{i,|u|} A_{ju}.Y_{iu} \qquad \forall j \quad \text{(E.8)}$$

$$Y_{iu} \in \{0,1\} \qquad \forall i, \forall j \quad \text{(E.9)}$$

**Subproblem:**

It is the same constraint programming model that we presented in section 3.1, however, this time it is easier to solve because task to machine assignments are already fixed by the master problem. In contrast to LBBD implementation of PSP, we are solving a harder subproblem as we cannot decompose it for each machine.

Constraint programming is relatively good for the scheduling problems with makespan minimization as it can prove the optimality, compared to minsum objectives. Therefore, given the solution of the master problem, subproblem can solve all $|K|$=10, $|J|$=20 instances within seconds. Not surprisingly though, while the master problem of PSP can be solved within 2 seconds for $|K|$=10, $|J|$=20 instances, the master problem mentioned above can only solve two out of 10 instances of the same size within 60 minutes of time limit. And both of these two solutions are proved to be infeasible in the subproblem.

As a result, by implementing the decomposition structure of Hooker (2007) which is then employed by many other studies, any of $|K|$=10, $|J|$=20 instances of MTSP with contiguous assignments cannot be solved within the time limit, while in 8 of them we cannot even solve the master problem.On the other hand, the decomposition structure that we proposed in the dissertation can solve all $|K|$=10, $|J|$=20 instances into optimality within 45.2 seconds on average.

Most probably, Hooker (2007) did not consider minsum objectives because the relaxations of scheduling subproblems would be very loose compared to minmax objectives. In line with that, even though many papers then followed his idea to assignment master problem, neither of them consider minsum objectives. Note that we also implemented the decomposition structure of Hooker (2007) to QCAP for total

weighted completion time minimization objective by presenting a relatively good relaxation of subproblem with the master problem variables. Compared to makespan minimization, the assignment master problem is solved very quickly but the subproblem fails to prove the optimality with this minsum objective within the time limit for any of the $|K|$=10, $|J|$=20 instances.

To sum up, Hooker (2007) studies a variant of parallel machine scheduling problem for minmax objectives, which can be decomposed into an assignment master problem with a relatively small search space, and very easy optimization subproblems that can be solved for each machine independently as a single machine scheduling problem. While master and subproblems can be solved in a short time, the master problem projects the feasible solution space of the original problem loosely as it disregards the scheduling of tasks, therefore, many Benders cuts need to be added to the master problem. When we implement this decomposition strategy to our problem, we obtain an assignment master problem with very large search space and a non-decomposable optimization subproblem that solves an instance of multiprocessor task scheduling problem. As a result, we cannot even complete the first iteration of the algorithm in most of the smallest instances of our test bed.