Optimierung von Kransystemen mit Interferenzen

Kumulative Dissertation
der Wirtschaftswissenschaftlichen Fakultät
der Universität Augsburg
zur Erlangung des Grades eines
Doktors der Wirtschaftswissenschaften
(Dr. rer. pol.)

vorgelegt von
Andreas Wiehl, M.Sc.

Erstgutachter:	Prof. Dr. Florian Jaehn
Zweitgutachter:	Prof. Dr. Axel Tuma
Drittgutachter:	Prof. Dr. Jens O. Brunner
Vorsitzender der mündlichen Prüfung:	Prof. Dr. Marco C. Meier
Datum der mündlichen Prüfung:	25.10.2018

Verzeichnis der Beiträge

Die vorliegende Dissertationsschrift enthält die folgenden Forschungsbeiträge:

R1 **Research Paper R1:**
Briskorn, D., Jaehn, F., & Wiehl, A. (2018). A generator for test instances of scheduling problems concerning cranes in transshipment terminals.

This is a pre-print of an article published in OR Spectrum.
The final authenticated version is available online at:
https://doi.org/10.1007/s00291-018-0529-z
(Beitrag der Kategorie A)

R2 **Research Paper R2:**
Jaehn, F., & Wiehl, A. (2018). Approximation algorithms for the twin robot scheduling problem.

This is a pre-print of an article published in Journal of Scheduling.
The final authenticated version is available online at:
https://doi.org/10.1007/s10951-019-00631-9
(Beitrag der Kategorie A)

R3 **Research Paper R3:**
Wiehl A. (2018). A decomposition procedure for different automated yard crane systems.

Die jeweils angegebenen Kategorien beziehen sich auf das Zeitschriftenranking JOURQUAL 3 des Verbands der Hochschullehrer für Betriebswirtschaft e. V. (VHB).

Inhaltsverzeichnis

Anmerkung: Die Tabellen, Abbildungen und Gleichungen sind in den Kapiteln fortlaufend nummeriert. Ein Literaturverzeichnis befindet sich am Ende der Kapitel I, IV und V. Die Forschungsbeiträge (Research Paper R1, R2 & R3) in Kapitel II, III und IV sind in Englisch verfasst.

I Einleitung

In den 30er Jahren des 20. Jahrhunderts hatte der US-amerikanische Lastkraftfahrer Malcolm McLean die bahnbrechende Vision, Waren und Güter weltweit in standardisierten Kisten zu transportieren (Levinson, 2016, S. 1). Eine stapelbare Box sollte direkt vom Lastkraftwagen in das Schiff gehievt werden, um so den Ladevorgang zu vereinfachen und die Produktivität an Seehäfen zu steigern. Vor der Einführung von Containern konnte der Warenumschlag nur durch einen hohen Einsatz an körperlicher Arbeit und teilweise wochenlangen Standzeiten der Schiffe realisiert werden. Für ein mit 5.000 Tonnen Stückgut beladenes Schiff waren ca. 60 Arbeiter nötig, um dieses innerhalb einer Woche zu löschen (Urwer, 2008, S. 3).
Allerdings sollte es noch 20 Jahre dauern, bis McLean seine Vision tatsächlich in der Praxis umsetzen konnte. Am 26. April 1956 war es schließlich soweit, als der umgebaute Frachter IDEAL X von Newark, New Jersey mit 58 Aluminium-Boxen beladen in Richtung Houston, Texas in See stach (Martin, 2012, S. 21). Der standardisierte Container war geboren — ein Meilenstein in der Logistik.
1966 landeten die ersten Container aus den USA im Hafen von Rotterdam und Bremen auf europäischem Boden (Jahns und Schüffler, 2008, S. 156). Ein paar Jahre später, im Jahr 1970, wurden die Maße der bis heute gängigen Standard-Container von der ISO (International Organization for Standardization) festgelegt. Als Standardmaß gilt seither die Twenty-foot Equivalent Unit (TEU), was einem Container mit 8 Fuß (2,44 m) Breite, 8 Fuß und 6 Zoll (2,59 m) Höhe sowie einer Länge von 20 Fuß (6,06 m) entspricht (Martin, 2016). Neben 20 Fuß (1 TEU) haben sich mittlerweile auch 2 TEU Container mit 40 Fuß Länge durchgesetzt, zudem existieren Spezial-Container mit Überlänge oder als High-Cube Ausführung.
Vor dem Zeitalter des Containers lohnte sich der internationale Export und Import für viele Produkte nicht (Levinson, 2016, S. 12). Die verstärkte Nutzung von Containern führte daher zu einer Revolution in der Logistik. Die gesamte Transportkette

an Land und Wasser hat sich an das System angepasst, was zu erheblichen Kosteneinsparungen, aber auch zu weltweiten Strukturveränderungen in Produktion und Handel geführt hat. Seit 2009 werden ca. 90% des Welthandels der Nicht-Schüttgutfracht über Seecontainer abgewickelt (Ebeling, 2009). Besonders beeindruckend ist die Entwicklung der letzten zwei Jahrzehnte: Im Zeitraum von 1996 bis 2016 hat sich der weltweite Containerhandel mehr als verdreifacht (vgl. Abb. 1).

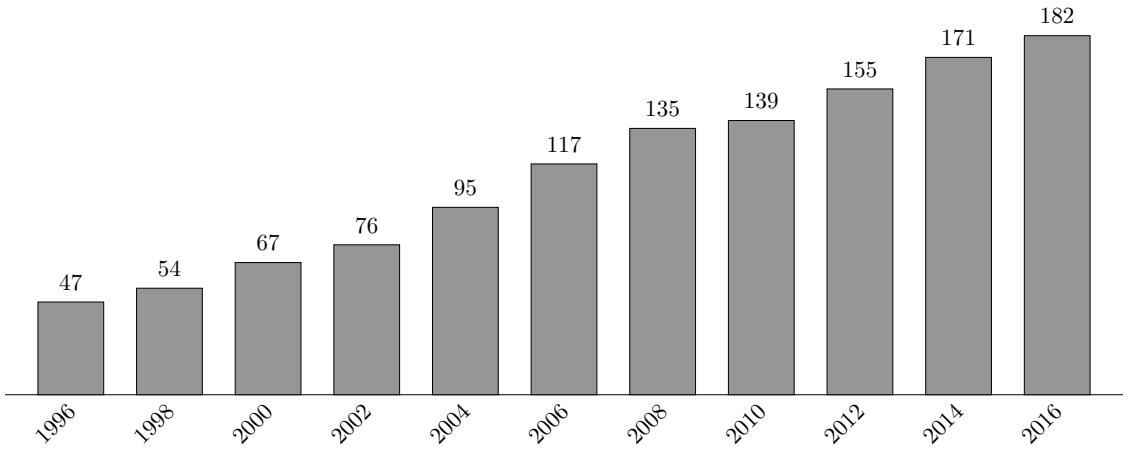


Abb. 1: Weltweiter Containerhandel in Mio. TEU (Saxon und Stone, 2017)

Mittlerweile liegen die zehn größten Häfen nach Containerumschlag ausschließlich auf dem asiatischen Kontinent (davon sieben in China), wobei der 'Port of Shanghai' den ersten Platz mit 40.230 Mio. TEU pro Jahr (2017) belegt. Die größten europäischen Häfen in Rotterdam, Antwerpen und Hamburg folgen auf den Plätzen 11, 13 und 18 (vgl. Maritime Intelligence (2017)). Obwohl das Wachstum durch die weltweite Bankenkrise im Jahr 2009 nachgelassen hat, rechnen Experten auf lange Sicht mit einem durchschnittlichen Wachstum von 1,9 bis 3,2% pro Jahr (vgl. Saxon und Stone (2017, S. 10)). Eine ähnliche Entwicklung ist bei der Größe von Containerschiffen zu beobachten. Hatte das größte Containerschiff im Jahr 1996 eine Kapazität von ca. 6.000 TEU (Reederei: Mærsk Line), so sind es mittlerweile 21.413 TEU (2017, Reederei: OOCL). Auch hier ist die Grenze des technisch Möglichen noch nicht erreicht, bereits heute existieren Pläne für Containerschiffe mit bis zu 24.000 TEU (Saxon und Stone, 2017, S. 13).

Durch den rasanten Anstieg des Welthandels und die steigende Kapazität von Frachtschiffen sind Umschlagsmengen von 6.000 Containern pro Schiff und Hafen kein Einzelfall mehr (Speer, 2017, S. 1). Aus Sicht der Reedereien sollte das Löschen und Laden der Container deshalb möglichst schnell und effizient durchgeführt werden, um kostspielige Standzeiten zu minimieren (Soriguera und Espinet, 2006; Schütt, 2011). Die Containerterminals unterliegen daher einem fortlaufenden Wettbewerb. Für den Erfolg des einzelnen Terminals ist es entscheidend, den Umschlagprozess noch effizienter und damit kostengünstiger als die Konkurrenz anzubieten (Min und Park, 2005).

Um den Anforderungen der Kunden gerecht zu werden, kommt seit der Jahrtausendwende verstärkt moderne Automatisierungstechnik zum Einsatz, mit der vor allem Kosten gesenkt und die Servicequalität verbessert werden sollen (Speer, 2017, S. 2). Insbesondere für Länder mit einem hohen Lohnniveau hat die Automatisierung einen bedeutenden Stellenwert. Im Sommer 2002 wurde das Containerterminal Altenwerder (CTA) in Hamburg eröffnet, bei dem der horizontale Containertransport nahezu vollautomatisiert abläuft (HHLA, 2017). Zwischen der Brücke und dem Containerlager werden für den fahrerlosen Transport Automated-Guided-Vehicles (AGVs) eingesetzt. In den Lagerblöcken arbeiten jeweils zwei Rail-Mounted-Gantries (RMGs) zwischen Land- und Seeseite. Ein solches Layout und der hohe Automatisierungsgrad des CTA in Hamburg gelten weltweit als Vorbild für viele moderne Terminals.

Dieser Trend ist in letzter Zeit auch in Asien angekommen. Im Jahr 2012 wurde das erste Terminal mit vertikal automatisiertem Transport in Korea eröffnet, das Busan-New-Container-Terminal (BNCT) (Slootweg, 2017). Zudem soll bis zum Jahr 2020 am Hafen von Shanghai das weltweit größte automatisierte Containerterminal mit 120 RMGs und 130 AGVs entstehen (The Straits Times, 2017).

Durch die neuen Technologien in der Automatisierung steigen allerdings auch die Anforderungen an die IT-Systeme, um diese zu steuern (Schütt, 2011, S. 104). Aus Sicht des Operations Research (OR) gibt es zahlreiche Optimierungsprobleme in einem Terminal zu beachten, wobei sich ein modernes teil- oder vollautomatisiertes Containerterminal am Seehafen in vier operative Abschnitte gliedern lässt (In Anlehnung an Gharehgozli et al. (2016), siehe Abb. 2).

Abschnitt **(1)** befindet sich an der Seeseite, hier wird das Schiff mithilfe von Kai-

Kränen be- und entladen. Zunächst muss das Schiff dabei einem Anlegeplatz und Kai-Kränen zugeordnet werden (Berth-Allocation-Problem und Quay-Crane-Assignment-Problem). Ziel ist es, unter anderem Servicezeiten und Verspätungen zu minimieren (Cordeau et al., 2005). Das Quay-Crane-Scheduling Problem muss anschließend gelöst werden, wobei durch einen optimalen Ablaufplan der Hafenkräne am Kai der Fertigstellungszeitpunkt minimiert und die Produktivität gesteigert werden soll (Bierwirth und Meisel, 2009). Beim Container-Stowage-Problem soll darüber hinaus ein Ladeplan für das Schiff erstellt werden. Ziele sind insbesondere die Minimierung der Servicezeit, die Gewährleistung der Stabilität und die Einhaltung der Belastungsgrenzen des Schiffes (Gharehgozli et al., 2016).

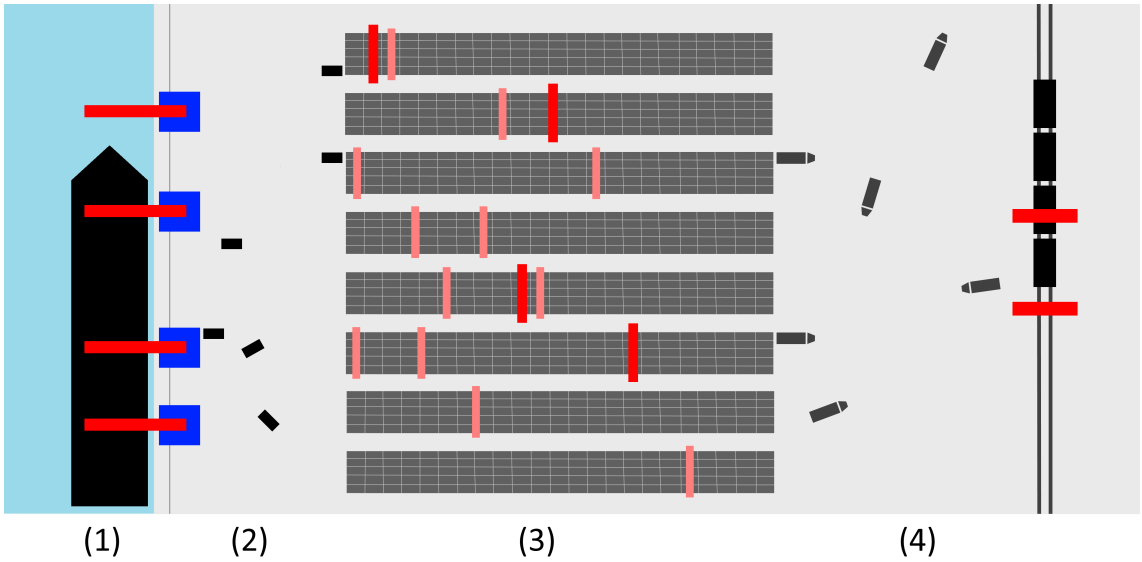


Abb. 2: Operative Einteilung von Containerterminals an Seehäfen

In Abschnitt **(2)** findet der interne Transport zwischen Kai und Lagerblock statt. Dabei wird zwischen automatisierten (z.B. AGVs) und von Menschen gesteuerten Fahrzeugen unterschieden. In modernen Systemen befinden sich zudem Übergabeplattformen an der Kai- und Lagerseite, um Wartezeiten zu vermeiden und das System unabhängig von den Kranbewegungen zu machen. Hierbei gilt es vor allem klassische Probleme der Touren- und Routenplanung zu lösen (Jeon et al., 2011) oder eine optimale Flottengröße zu bestimmen.

In Abschnitt **(3)** befindet sich der Kern des Terminals mit mehreren Lagerflächen bzw. Blöcken zur Zwischenlagerung der Container. Bei älteren Terminallayouts bewerkstelligen bspw. Portalhubwagen (engl. *straddle carriers*) oder gummibereifte Stapelkräne (engl. *rubber tired gantry*) den Transport der Container. In modernen automatisierten Terminals arbeiten bis zu drei RMGs an einer Fläche, um Container ein- und auszulagern, wobei sich die Übergabestellen häufig an den kurzen Seiten des Blocks befinden. Hier sind die Lagerflächen oftmals orthogonal zur Anlegestelle ausgerichtet und bilden somit eine Schnittstelle zwischen See- und Landseite bzw. Automatisierung und manueller Ausführung.
Ziele beim sogenannten Yard-Crane-Scheduling-Problem sind oftmals produktivitäts- und/oder serviceorientiert, bspw. sollen die Kräne Container ein- und auslagern, sodass der Fertigstellungszeitpunkt minimal ist oder Servicezeiten der Schiffe eingehalten werden können. Neben Ablaufplanungsproblemen existieren darüber hinaus zahlreiche Optimierungsprobleme, bei denen der Fokus auf dem geschickten Stapeln der Container liegt. Beim Pre-Marshalling-Problem sollen die Container so gestapelt werden, dass zu einem späteren Zeitpunkt keine weiteren Umstapler nötig sind (Lee und Hsu, 2007). Mehrere Forschungsarbeiten befassen sich daher mit verschiedenen Stapel-Strategien, bspw. sollen die Container stets nach den erwarteten Standzeiten sortiert werden (Dekker et al., 2006).
In Abschnitt **(4)** findet der Container-Austausch zur Landseite bzw. Straße, Schiene oder Fluss statt. Durch eine enge Zusammenarbeit mit Hinterland-Terminals und Spediteuren kann der Hafen die Auslastung im Laufe der Zeit besser steuern und Verzögerungen beim Be- und Entladen vermeiden. In der Studie von van Asperen et al. (2013) wird der Einfluss von Informationen zur Ankunft der LKWs auf die Effizienz des Terminals untersucht. Douma et al. (2009) erstellen ein Verfahren, um den Fahrplan der Binnenschifffahrt an die Abläufe am Hafen von Rotterdam anzupassen. Eine Verlagerung des Verkehrs auf Binnenschiffe (oder Züge) ist sinnvoll, um bspw. die Belastung auf den Straßen zu verringern und Emissionen einzusparen.
Wie bereits erwähnt, wird in modernen Containerterminals häufig Automatisierungstechnologie eingesetzt, um den steigenden Anforderungen an Service und Produktivität gerecht zu werden. In Abschnitt **(3)** des Terminals setzt man hierbei insbesondere auf RMG-Systeme, diese stellen jedoch einen potentiellen Engpass und eine kritische Ressource im System dar. Zahlreiche technische Innovationen und Er-

weiterungen in den anderen Bereichen machen diese besser skalierbar, dazu zählen bspw. Dual-Cycling am Kai-Kran **(1)**, Multitransporte, zusätzliche AGVs für den horizontalen Transport **(2)** sowie Vormeldesysteme für LKWs an der Landseite **(4)** (Speer, 2017, S. 121–125).
Aufgrund des hohen Platzbedarfs und der verhältnismäßig aufwendigen Erweiterbarkeit gelten RMG-Systeme als schlecht skalierbar, weshalb die Optimierung der vorhandenen Geräte von zentraler Bedeutung ist. Sobald mehr als ein Kran an der gleichen Fläche arbeitet, kann es zu Behinderungen kommen, wodurch Wartezeiten oder Umwege der beteiligten Kräne entstehen. Durch eine intelligente Steuerung bzw. Ablaufplanung lassen sich diese Behinderungen eingrenzen bzw. minimieren, sodass letztendlich eine höhere Produktivität gewährleistet wird.
An diesem Optimierungsproblem setzt auch die vorliegende kumulative Dissertationsschrift an. Bevor jedoch die einzelnen Forschungsbeiträge und Ergebnisse zu diesem Thema präsentiert werden (Abschnitt 2), soll im nachfolgenden Abschnitt 1 zunächst genauer auf den konkreten Forschungskontext eingegangen und die wesentlichen Forschungsziele dargestellt werden.

1 Forschungskontext und Forschungsziele

Heutzutage verlassen sich viele Wirtschaftszweige auf automatisierte Kransysteme, wenn es darum geht, Waren möglichst effizient von A nach B zu bewegen bzw. ein- und auszulagern. Es finden sich dabei zahlreiche Einsatzgebiete, bspw. in der Containerlogistik, in der Industrie und bei automatisierten Lagersystemen (engl. automated storage and retrieval systems (ASRS)). Um die Produktivität zu steigern, werden oft mehrere Kräne bzw. Roboter auf einer gemeinsamen Fläche oder Schiene eingesetzt, sodass Interferenzen bzw. Wartezeiten auftreten.
Zu vielen praktischen Problemstellungen im Bereich der Optimierung von Kransystemen mit Interferenzen sind bereits wissenschaftliche Beiträge erschienen, die sich mit verschiedenen Anwendungsbereichen befassen (siehe Abb. 3). Boysen et al. (2017) stellen fest, dass das Forschungsinteresse gerade in den letzten Jahren besonders hoch war, von insgesamt 82 klassifizierten Publikationen sind 63 allein im Zeitraum zwischen 2010 und 2016 erschienen, zudem sind über 75% der Beiträge der Containerlogistik an Seehäfen zuzuordnen. Davon betrachten wiederum 44 die Si-

tuation am Kai (insb. das Quay-Crane-Scheduling-Problem) und 20 den Lagerblock (insb. das Yard-Crane-Scheduling-Problem).

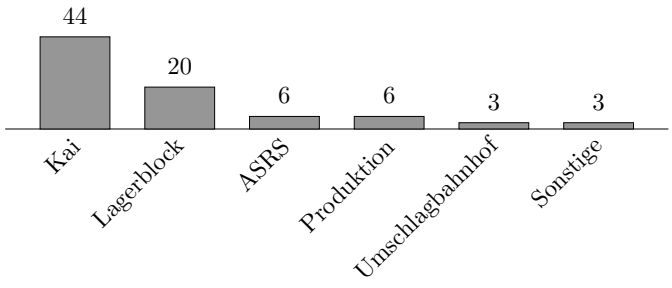


Abb. 3: Anzahl der Publikationen pro Anwendungsbereich (Boysen et al., 2017)

Ein Hauptfokus in diesem Forschungsbereich liegt auf der Minimierung des Fertigstellungszeitpunktes, wovon 61 Publikationen zeugen. Darüber hinaus werden bei 71 Beiträgen die Lagerpositionen auf einer eindimensionalen Geraden betrachtet bzw. angenommen, die restlichen elf Publikationen betrachten Interferenzen in einem zwei- oder dreidimensionalen Raum.
Das vorrangige Ziel der vorliegenden Dissertation besteht darin, in diesem Themenfeld einen Beitrag für Forschung und Praxis zu leisten, insbesondere in Bezug auf die Situation am Lagerblock in Seehäfen. Konkret lassen sich dabei vor allem die folgenden vier Forschungsziele formulieren:

Ziel 1: Ein häufiges Problem im Bereich des Operations Research ist es, die entwickelten Verfahren mit denen anderer Forscher zu vergleichen, sofern kein gemeinsames Testbed zur Verfügung steht (Beasley, 1990). Zudem sind in vielen Fällen keine praktischen Daten verfügbar, z.B. wenn neue Konzepte analysiert werden, die noch nicht in der Praxis etabliert sind oder aus vertraulichen Gründen nicht preisgegeben werden. Ziel ist es daher, für die Optimierung von automatischen Lagerkransystemen ein konsistentes Testbed zu etablieren, mit dessen Hilfe es möglich ist, praxisnahe und zukünftig relevante Situationen am Lagerblock in Containerterminals abzubilden, um repräsentative Stichproben von Testinstanzen zu erzeugen.

Ziel 2: Die Optimierungsprobleme in diesem Forschungsfeld sind teilweise $\mathcal{NP}$-schwer, daher kann es durchaus sinnvoll sein, Approximationsalgorithmen zu entwickeln, die der optimalen Lösung möglichst nahe kommen. Solche Algorithmen existieren in diesem Forschungsfeld bisher jedoch fast ausschließlich für das Quay-Crane-Scheduling-Problem (Lee und Chen, 2010; Lee und Wang, 2010), während in den anderen Anwendungsbereichen noch erheblicher Forschungsbedarf besteht. Einzig Erdoğan et al. (2014) beschreiben einen solchen Algorithmus auch für automatische Lagersysteme (ASRS). Ein wesentliches Ziel der vorliegenden Arbeit besteht daher vor allem darin, Approximationsalgorithmen für Problemstellungen am Lagerblock zu entwickeln.

Ziel 3: Aus dem Beitrag von Boysen et al. (2017) geht hervor, dass sich bislang nur wenige Forscher mit den für die Praxis relevanten cross-over Kränen befasst haben, z.B. ein kleiner RMG der unter bestimmten Umständen einen großen RMG passieren kann. Die existierenden Ansätze betrachten stets eher spezifische Krankonfigurationen, z.B. Twin RMG (ein kleiner und großer RMG) (Briskorn und Angeloudis, 2016) oder Triple RMG (zwei kleine und ein großer RMG) (Dorndorf und Schneider, 2010). Ein weiteres zentrales Forschungsinteresse bzw. Ziel der Dissertationsschrift besteht in diesem Zusammenhang darin, einen allgemeinen Ansatz zu entwickeln, der beliebige Kombination von kleinen und großen Kränen berücksichtigt, denn so betonen unter anderem auch Boysen et al. (2017):

> *„[...], there is a special need for generalized approaches, which can handle any combination of small and large cranes.“* (Boysen et al., 2017)

Ziel 4: Weitere Forschungslücken sehen Boysen et al. (2017) zudem in gemeinschaftlichen Betrachtungen von Ein- und Auslagerungen in einem Kranzyklus. Die abwechselnde Durchführung beider Bewegungen wird auch als Double- bzw. Dual-Cycle bezeichnet und führt zu einer Reduzierung der Leerfahrten eines Krans. Zwar existieren bereits zahlreiche Ansätze in ASRS oder an Containerterminals (Boysen und Stephan, 2016; Meisel und Wichmann, 2010; Gharehgozli et al., 2014), jedoch wird hier immer nur ein einziger Kran betrachtet. Eine detaillierte Betrachtung von Dual-Cycles unter Berücksichtigung von mehreren Kränen mit Interferenzen fehlt bislang dagegen. So schreiben Boysen et al. (2017) beispielsweise:

> *„[...] the combination of a storage move and a retrieval move within one crane cycle, proved very successful in ASRS and QC scheduling, [...]. However these studies consider only a single crane. Thus, a systematic in- vestigation of dual cycles for multiple cranes under non-crossing constraints is yet missing.“* (Boysen et al., 2017)

Mit der vorliegenden Dissertationsschrift soll daher ein Beitrag zur Schließung dieser Forschungslücke geleistet werden.

2 Zusammenfassung der Forschungsbeiträge

Im folgenden Abschnitt werden die in der Dissertationsschrift enthaltenen Beiträge zusammengefasst und in den Forschungskontext im Hinblick auf die oben genannten Ziele eingebettet.

Kapitel II – Research Paper R1: *A generator for test instances of scheduling problems concerning cranes in transshipment terminals*
In Beitrag R1 wird ein Instanzen-Generator vorgestellt, der zur Simulation von Kranprozessen an Containerterminals dient. Mithilfe des Generators ist es möglich, gängige Abläufe an Containerterminals anhand zahlreicher Parameter nachzustellen. Vorbilder für die parametrisierte Datengenerierung haben sich bereits in anderen Bereichen des Operations Research etabliert, z.B für Ablaufplanungsprobleme (Hall und Posner, 2001), in der Projektplanung (Kolisch et al., 1995) und in der Auktionsforschung (Leyton-Brown und Shoham, 2006).
Um ein möglichst gutes Abbild der Realität zu gewährleisten, ist das generische Modell in Zusammenarbeit mit einem Praxispartner aus diesem Gebiet entstanden. Eine einzelne Instanz repräsentiert dabei eine gegebene Menge an Containerbewegungen innerhalb eines Lagerblocks. Je nach Planungsebene bzw. Zielfunktion kann man den Containern bestimmte Eigenschaften mithilfe von Verteilungsfunktionen zuweisen, dazu gehören Positionsdaten, Termindaten, Vorrangbeziehungen und Weitere. Zudem können die Maße des Lagerblocks innerhalb einer Instanz mit bis zu drei Dimensionen spezifiziert werden. Der Instanzen-Generator ist online als Benutzeroberfläche (GUI, siehe www.instances.de/dfg/) sowie Programmierschnittstelle

(API) verfügbar und damit für Forscher weltweit zugänglich. Eine weitere wichtige Eigenschaft des Generators ist die Reproduzierbarkeit der Daten. Eine Sammlung von Instanzen kann unter einem Projektnamen veröffentlicht und mithilfe der Seed-generierten Zufallszahlen wiederhergestellt werden.

Kapitel III – Research Paper R2: *Approximation algorithms for the twin robots scheduling problem*

Forschungsbeitrag R2 stellt eine grundsätzliche Erweiterung der Arbeit *Scheduling twin robots on a line* von Erdoğan et al. (2014) dar. Das Praxisproblem des darin vorgestellten $\mathcal{NP}$-schweren Twin-Robots-Scheduling-Problem (TRSP) findet sich bei automatisierten Lagersystemen:

Zwei gleichartige Industrieroboter sind an gegenüberliegenden Enden (Depots) einer Schiene positioniert und liefern Produkte von Depot zu Slots entlang dieser Schiene, des Weiteren müssen die Roboter einen Sicherheitsabstand von einem Slot einhalten und können somit nicht aneinander vorbei fahren (engl. non-crossing). Ziel ist es, daher einen Ablaufplan für die Lieferaufträge zu erstellen, sodass der Fertigstellungszeitpunkt minimal ist. In der Praxis tritt dieses Problem beispielsweise beim automatischen Zusammenfassen von Produkten zu einer Ladeeinheit auf (Palettieren).

Das ursprüngliche Problem wird durch variable Abhol- und Lieferzeiten erweitert, um die Situation an Containerterminals besser abzubilden. Insgesamt wird zwischen vier verschiedenen Szenarien unterschieden. In Szenario A werden die Aufnahme- bzw. Lieferzeiten vernachlässigt (Erdoğan et al., 2014), bei B wird von konstanten Aufnahme- bzw. Lieferzeiten ausgegangen (Boysen et al., 2015) und bei C können diese Zeiten unterschiedlich ausfallen. An Containerterminals ist diese Annahme durchaus realistisch, da es bspw. bei der Aufnahme von Containern zu Verzögerungen durch Umstapler kommen kann. In Szenario D wird sowohl die Einlagerung als auch die Auslagerung innerhalb eines Prozesses berücksichtigt (*dual cycle*). Zur Erstellung von Ablaufplänen werden zwei Konzepte eingeführt: *semi-active schedule* und *active schedule*. Bei einem *active schedule* werden bspw. die Aufträge im Ablaufplan stets am frühestmöglichen Ort positioniert, ohne dass dabei ein einziger bereits eingeplanter Auftrag verschoben bzw. später ausgeführt wird.

Eine untere Schranke (engl. *lower bound*) ergibt sich, wenn die *non-crossing* Ne-

benbedingung aufgehoben wird, diese lässt sich mit einer Laufzeit von $\mathcal{O}(n \log n)$ feststellen, wobei n die Anzahl der Aufträge ist.
Ein weiterer zentraler Bestandteil von Beitrag R2 sind drei Approximationsalgorithmen. Für Instanzen von Szenario A und B werden drei Verfahren beschrieben, die eine approximative Lösung in Polynomialzeit finden. Mit der Decreasing-Sort-Procedure und Earliest-Fit-Procedure liegt die Lösung maximal 50% über der unteren Schranke und der Best-Fit Algorithmus liefert eine Lösung, die maximal $\approx 17{,}16\%$ über der unteren Schranke liegt, für hinreichend große Instanzen.
Neben den Heuristiken wird auch ein exaktes Branch-and-Bound Verfahren entwickelt. Mithilfe einer Dominanzregel (engl. *domination rule*) lassen sich die 50 von Erdoğan et al. (2014) zur Verfügung gestellten Instanzen signifikant schneller lösen als mit bisher bekannten exakten Verfahren.
In einer ausführlichen numerischen Studie werden Instanzen für die vier vorgestellten Szenarien untersucht. Dabei liegt ein Schwerpunkt darauf herauszustellen, unter welchen Begebenheiten Interferenzen zwischen den Robotern bzw. Kränen auftreten bzw. wann ein konfliktfreier Ablaufplan möglich ist. Um dies festzustellen, werden 22 Verteilungsmuster für Lagerpositionen untersucht.

Kapitel IV – Research Paper R3: *A decomposition procedure for different automated yard crane systems*
Forschungsbeitrag R3 beschreibt einen allgemeinen Ansatz zur Optimierung von verschiedenen Krankonfigurationen mit *cross over* Kränen. Das zugrunde liegende Praxisproblem kann folgendermaßen zusammengefasst werden: Eine verfügbare Anzahl an RMGs ist entlang eines Lagerblocks angeordnet und soll Container ein- bzw. auslagern. Es befinden sich zwei Übergabestellen an den kurzen Seiten des Blocks bzw. an der Land- und Seeseite. Die Positionsdaten der verfügbaren Container sind gegeben, jedoch ist die Bearbeitungsreihenfolge und Zuordnung der auszuführenden RMGs variabel. Ein kleiner RMG kann einen großen RMG immer dann passieren, wenn dieser gerade keinen Container aufnimmt bzw. ablädt. Zudem werden nur Systeme betrachtet, bei denen sich maximal zwei RMGs derselben Ausprägung (bzw. Größe) am Lagerblock befinden. Ziel des Forschungsbeitrages ist es daher, einen Ablaufplan zu erstellen, bei dem alle Aufträge durch die zur Verfügung stehenden RMGs ausgeführt werden, sodass der Fertigstellungszeitpunkt minimal ist.

Aufgrund der hohen Entscheidungstiefe des zugrunde liegenden Problems wird ein heuristischer Ansatz verfolgt, um praxisrelevante Instanzen in angemessener Zeit zu lösen. Das Problem wird dabei in drei Teilprobleme zerlegt: (1.) Die Zuordnung von Aufträgen zu Kränen (Crane-Assignment-Problem), (2.) das Zusammenfügen von Ein- und Auslagerungen zu *dual cycles* bzw. *single cycles* (Single-Crane-Routing-Problem) und (3.) das Erstellen eines Ablaufplans unter Berücksichtigung von Interferenzen (Crane-Scheduling-Problem). Zur Lösung der Probleme (1.) und (2.) werden mathematische Modelle formuliert. Dabei liefert (1.) eine untere Schranke für das Gesamtproblem, d.h. der minimale Fertigstellungszeitpunkt ohne Berücksichtigung der Interferenzen. Zur Lösung des anschließenden Crane-Scheduling-Problem werden ein verkürzter Branch-and-Bound-Algorithmus sowie eine Tabu-Suche implementiert. Der resultierende Ablaufplan legt zum einen alle Bearbeitungsreihenfolgen der Container fest und zum anderen, welcher Auftrag Vorfahrt hat bzw. warten muss, sofern Interferenzen auftreten.
In einer numerischen Studie werden sieben verschiedene RMG-Systeme mit bis zu vier Kränen betrachtet, davon finden vier bereits Anwendung in der Praxis und die verbleibenden drei stellen mögliche zukünftige Erweiterungen dar. Die Systeme werden anhand von drei typischen Szenarien an Containerterminals in der Studie auf ihre Leistungsfähigkeit getestet. Das vorgestellte heuristische Verfahren liefert gute Ergebnisse nahe der unteren Schranke für alle Kransysteme, zudem können einige Instanzen auch optimal gelöst werden. Ein zentrales Forschungsergebnis ist, dass eine höhere Anzahl an Aufträgen in der Planung auch zu einer höheren Produktivität des Systems führt.

In den folgenden Kapiteln II, III und IV werden die einzelnen, hier kurz dargestellten Forschungsbeiträge schließlich präsentiert. Abschließend fasst Kapitel V *Fazit und Ausblick* die wichtigsten Ergebnisse der Dissertationsschrift nochmals gebündelt zusammen und gibt einen Ausblick auf zukünftige Forschungsfelder.

Literaturverzeichnis

Beasley, J. E. (1990). OR-Library: Distributing test problems by electronic mail. Journal of the operational research society, 41(11):1069–1072.

Bierwirth, C. und Meisel, F. (2009). A fast heuristic for quay crane scheduling with interference constraints. Journal of Scheduling, 12(4):345–360.

Boysen, N., Briskorn, D., und Emde, S. (2015). A decomposition heuristic for the twin robots scheduling problem. Naval Research Logistics (NRL), 62(1):16–22.

Boysen, N., Briskorn, D., und Meisel, F. (2017). A generalized classification scheme for crane scheduling with interference. European Journal of Operational Research, 258(1):343–357.

Boysen, N. und Stephan, K. (2016). A survey on single crane scheduling in automated storage/retrieval systems. European Journal of Operational Research, 254(3):691–704.

Briskorn, D. und Angeloudis, P. (2016). Scheduling co-operating stacking cranes with predetermined container sequences. Discrete Applied Mathematics, 201:70–85.

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

Dekker, R., Voogd, P., und van Asperen, E. (2006). Advanced methods for container stacking. OR Spectrum, 28(4):563–586.

Dorndorf, U. und Schneider, F. (2010). Scheduling automated triple cross-over stacking cranes in a container yard. OR Spectrum, 32(3):617–632.

Douma, A., Schutten, M., und Schuur, P. (2009). Waiting profiles: An efficient protocol for enabling distributed planning of container barge rotations along terminals in the port of rotterdam. Transportation Research Part C: Emerging Technologies, 17(2):133–148.

Ebeling, C. (2009). Evolution of a box. Invention and Technology, 23(4):8–9.

Erdoğan, G., Battarra, M., und Laporte, G. (2014). Scheduling twin robots on a line. Naval Research Logistics (NRL), 61(2):119–130.

Gharehgozli, A. H., Roy, D., und de Koster, R. (2016). Sea container terminals: New technologies and or models. Maritime Economics & Logistics, 18(2):103–140.

Gharehgozli, A. H., Yu, Y., de Koster, R., und Udding, J. T. (2014). An exact method for scheduling a yard crane. European Journal of Operational Research, 235(2):431–447.

Hall, N. G. und Posner, M. E. (2001). Generating experimental data for computational testing with machine scheduling applications. Operations Research, 49(6):854–865.

HHLA (2017). Geschichte Container Terminal Altenwerder - Der Klassenbeste feiert Geburtstag. https://hhla.de/de/container/altenwerder-cta/geschichte-cta.html. Letzter Zugriff 6. März 2018.

Jahns, C. und Schüffler, C. (2008). Logistik: Von der Seidenstraße bis heute. Wissensreihe des Siegfried-Vögele-Institut. Gabler Verlag.

Jeon, S. M., Kim, K. H., und Kopfer, H. (2011). Routing automated guided vehicles in container terminals through the Q-learning technique. Logistics Research, 3(1):19–27.

Kolisch, R., Sprecher, A., und Drexl, A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. Management Science, 41(10):1693–1703.

Lee, D.-H. und Chen, J. H. (2010). An improved approach for quay crane scheduling with non-crossing constraints. Engineering Optimization, 42(1):1–15.

Lee, D.-H. und Wang, H. Q. (2010). An approximation algorithm for quay crane scheduling with handling priority in port container terminals. Engineering Optimization, 42(12):1151–1161.

Lee, Y. und Hsu, N.-Y. (2007). An optimization model for the container premarshalling problem. Computers & Operations Research, 34(11):3295–3313.

Levinson, M. (2016). The Box: How the Shipping Container Made the World Smaller and the World Economy Bigger. Princeton University Press.

Leyton-Brown, K. und Shoham, Y. (2006). A Test Suite for Combinatorial Auctions, Seiten 451–478. MIT Press, Cambridge.

Maritime Intelligence (2017). Lloyd's list top 100 container ports: Rankings. Letzter Zugriff 6. März 2018.

Martin, A. (2012). Nachhaltigkeit in der Container Logistik: Wie relevant ist die Nachhaltigkeit für das Unternehmensimage? Reihe Nachhaltigkeit. Diplomica Verlag.

Martin, C. (2016). Shipping Container. Object Lessons. Bloomsbury Publishing.

Meisel, F. und Wichmann, M. (2010). Container sequencing for quay cranes with internal reshuffles. OR Spectrum, 32(3):569–591.

Min, H. und Park, B.-I. (2005). Evaluating the inter-temporal efficiency trends of international container terminals using data envelopment analysis. International Journal of Integrated Supply Management, 1(3):258–277.

Saxon, S. und Stone, M. (2017). Container shipping: The next 50 years. Travel, Transport & Logistics. McKinsey & Company.

Schütt, H. (2011). Simulation technology in planning, implementation and operation of container terminals. In Handbook of terminal planning, Seiten 103–116. Springer.

Slootweg, P. (2017). The merits of vertical-automated container terminals now proven in asia. PTI Journal, 57.

Soriguera, F. und Espinet, D. (2006). A simulation model for straddle carrier operational assessment in a marine container terminal. Journal of Maritime Research, 3(2):19–34.

Speer, U. (2017). Optimierung von automatischen Lagerkransystemen auf Containerterminals. Springer Fachmedien Wiesbaden.

The Straits Times (2017). World's largest automated container terminal opens in Shanghai. http://www.straitstimes.com/asia/east-asia/worlds-largest-automated-container-terminal-opens-in-shanghai. Letzter Zugriff 6. März 2018.

Urwer, A. (2008). Bedeutung und Entwicklung des Containerverkehrs im internationalen Transport. GRIN Verlag.

van Asperen, E., Borgman, B., und Dekker, R. (2013). Evaluating impact of truck announcements on container stacking efficiency. Flexible Services and Manufacturing Journal, 25(4):543–556.

II Research Paper R1

A generator for test instances of scheduling problems concerning cranes in transshipment terminals

Dirk Briskorn[1], **Florian Jaehn**[2], **Andreas Wiehl**[3]

[1] University of Wuppertal, [2] Helmut Schmidt University, [3] University of Augsburg

This work has been supported by the German Science Foundation (DFG) through the grant „*Scheduling mechanisms for rail mounted gantries with respect to crane interdependencies*“ (BR 3873/7-1 and JA 2311/2-1).

This is a pre-print of an article published in OR Spectrum.
The final authenticated version is available online at:
https://doi.org/10.1007/s00291-018-0529-z

Abstract

We present a test data generator that can be used for simulating processes of cranes handling containers. The concepts originate from container storage areas at seaports, but the generator can also be used for other applications, particularly for train terminals. A key aspect is that one or multiple cranes handle containers, that is, they store containers, receiving the containers in a designated handover area; outsource containers, handing the containers over in the handover area; or reshuffle containers. We present a generic model and outline what is captured by the test data itself and what is left to be estimated by the user. Furthermore, we detail how data are generated to capture the considerable variety of container characteristics,

which can be found in major terminals. Finally, we present examples to illustrate the variety of research projects supported by our test data generator.

Keywords: OR in maritime industry, test data generator, container terminals, crane scheduling.

1 Introduction

Containerization has been an important component of international trade for many years. The fastest and most cost-effective way to ship general cargo is typically by container. Accordingly, the volume of cargo being transshipped in containers has increased over the past twenty years; see Port of Hamburg (2017a). Not surprisingly, the throughput of most major ports is increasing; see Port of Hamburg (2017b). Similarly, this leads to new challenges for rail-road terminal operators since container transportation on rail is typically preferred over that on the road due to lower costs and because of a reduced environmental impact, e.g. rail-based freight systems emit less CO_2 than truck-only systems; see Kim and Van Wee (2009).

Automation of handling processes has been in the focus of terminal operators and equipment manufacturers over the past 20 years. Seen as the key to significant performance improvements and cost savings, many efforts have been devoted to automating several aspects of terminal operation. Automated stacking cranes (ASCs) are commonly used in major seaport terminals, such as the port of Rotterdam, the port of Hamburg, and the port of Antwerp. In contrast to straddle carriers and rubber-tired gantry cranes, which are not automated, ASCs are fixed to a certain block within the container storage area. In other words, ASCs manage containers only within the storage area and consequently have to hand over the containers to transport devices for the containers to be transported elsewhere or receive containers from other devices. Typically, such transport devices are automated guided vehicles or ship-to-shore cranes on the seaside and trucks on the landside. ASCs span the entire storage block in width and move on tracks installed alongside the block. They are not necessarily fully automated and may need an operator to guide them in the handover areas. Handover areas may either be on the short side of the blocks (e.g. Port of Hamburg) or on the long side of the blocks (e.g. Shanghai International

Port). In the paper on hand we present a test data generator to simulate processes of cranes handling containers at transshipment yards. The generator can be used to create a wide range of problems instances in this area and thus establish a basis for further research.

The following Section 1.1 provides a literature review on cranes in transshipment terminals and other test data generators. We first consider the situation of cranes at seaport terminals, then at train terminals, last we provide an overview on test data generators in general. In Section 1.2, the contribution and structure of our paper is presented.

1.1 Literature review

Aside from allowing automation from a technical perspective, optimizing the (stacking or gantry) cranes' performance potential requires effective and efficient scheduling mechanisms. These mechanisms are clearly substantial for both ASCs and non-automated stacking cranes, including quay cranes (QCs). Note that similar to ASCs, QCs move on a line, pick up or release containers on vessels, and have to hand over containers to other transport devices. In recent years, several pieces of literature have been published concerning ASCs and QCs. We refer to Stahlbock and Voß (2008); Steenken et al. (2004) for general surveys on optimization in seaport container terminals.

Regarding ASCs, several methods were developed for scheduling single cranes; see Kim and Kim (1999), Kim et al. (2003), Lee et al. (2007), Narasimhan and Palekar (2002), and Ng and Mak (2005a,b). Ng (2005) was among the first to investigate the optimal scheduling of multi-gantry cranes and presents an integer programming model that can be used to determine the sequence of crane activities for the execution of a series of container moves. In Froyland et al. (2008), a multiple crane system is considered, where the area for each stacking crane is restricted such that cranes do not interfere. Li et al. (2009) consider a similar system without restricting the operations of cranes to certain areas and provide several mixed integer programming-based techniques. Saanen and van Valkengoed (2006) compare single cranes, twin cranes, and crossover cranes through a computational study. Vis and Carlo (2010) present a scheduling approach for cross-over ASCs that employs an integer program-

ming model that can determine a lower-bound schedule, accompanied by a simulated annealing heuristic that can further balance the operations of both gantries. Speer et al. (2011) consider blocks with multiple ASCs. They evaluate different heuristics and an exact approach to minimize a weighted sum of job lateness (container drop-off with respect to a due date), makespan and the total flow times. Briskorn et al. (2016) consider the problem of importing containers arriving at one handover area via the joint transport of twin cranes. They aim to minimize the schedule's makespan. A scheduling algorithm for triple cross-over stacking cranes is discussed by Dorndorf and Schneider (2010). When assigning containers to be transported to cranes using a heuristic, they address the subproblem to decide the priority of cranes when the cranes' next operations are in conflict with each other. Then, avoiding collisions for a given sequence of operations is accomplished by employing a branch-and-bound algorithm. Briskorn and Angeloudis (2016) focus on this subproblem, which was addressed in Dorndorf and Schneider (2010) in two crane settings. For both twin cranes and crossover cranes, they provide efficient algorithms to optimally determine the priority of cranes in terms of makespan minimization if the sequence of containers to be transported is fixed for both cranes. This subproblem is employed in a branch-and-cut approach, determining the assignment of containers to cranes and the sequences of containers assigned to the same crane in a crossover crane setting in Nossack et al. (2017). In the work of Speer and Fischer (2016) the productivity of several different automated yard crane systems are tested.

The first optimization approaches for QC scheduling originate from Daganzo (1989) and Peterkofsky and Daganzo (1990). These studies, however, do not consider non-interference constraints of cranes. Kim and Park (2004) consider non-crossing constraints and present a model formulation along with exact and heuristic solution procedures. Alternative solution methods are presented by Lee et al. (2008b), who also provide an NP-hardness proof. Related contributions are also provided by Lee et al. (2008a), Lim et al. (2004, 2007) and Zhu and Lim (2006). A comprehensive review on QC scheduling is provided by Bierwirth and Meisel (2010, 2015).

Although it appears that seaport container terminals are more prominent as fields of application in the scientific literature, similar crane settings arise in train terminals; for a survey, see Boysen et al. (2013, 2017). Rail-road terminals have become one of the cornerstones of intermodal freight, with their main purpose being to serve as an

interface between different modes of transportation. In rail transport, there is also a trend toward automation, as discussed in Rotter (2004), although the degree of automation is still far behind that of ports. In a railway container terminal, freight trains are parked on parallel transshipment tracks of the terminal.

A terminal segment generally consists of between two and four parallel tracks, although a maximum gantry span of six tracks is possible; see, e.g., Steenken et al. (2004). Furthermore, a floor storage area enables intermediate container storage for cases in which a delivered container cannot be immediately shipped to the respective outbound truck or train. Typically, one or multiple gantry crane(s) span all three elements (i.e., tracks, storage area and truck lanes) such that containers can be directly moved to their destinations in a single step. Up to four of these gantry cranes serve a terminal segment in parallel.

Boysen and Fliedner (2010a) assign static and disjoint crane areas to a bundle of trains with given parking positions to minimize the makespan of train processing. They present a polynomial dynamic programming procedure for solving the resulting problem and test the solutions against typical real-world policies in a simulation of yard operations. As a complementary work, Briskorn and Fliedner (2012) consider the problem of determining parking positions for given crane areas. Alicke (2002) assigns a given set of crane moves to cranes with overlapping areas of operation, which are blocked whenever a crane enters an area. Whenever a start or target position falls in an overlapping area, the procedure dynamically determines which of two neighboring cranes processes the move.

At the border of two countries and railway systems, rail–rail terminals are also used to bridge different track gauges. This requires a special yard setting in which complete train loads are transshipped by cranes onto a train with the gauge width of the destination railway system. Martinez et al. (2004) investigate two simple rules for crane scheduling at a terminal on the border between France and Spain. Both rules were compared through a simulation study. The same terminal is investigated by Gonzalez et al. (2008), who provide a mixed integer model to jointly determine the load plan of outbound trains and crane schedules. Their objective is to minimize crane travel distances while observing the weight and length restrictions of wagons. The model is solved using an off-the-shelf solver that is shown to be suitable for real-world instances of small size. In Cichenski et al. (2017) several sub-problems of

the rail–rail transshipment yard scheduling problem are solved with an integrated MILP model as a single optimization problem.

For all the aforementioned systems, one or more cranes, which can pass each other only under certain circumstances or cannot pass each other at all, move along a line and deliver containers to handover areas, pick up containers in such areas or relocate containers within the storage area. A considerable amount of the models developed thus far are based on similar or even identical cores; see Boysen et al. (2017). However, to the best of our knowledge, no unifying test data generator has yet been established. Hartmann (2004) describe how to generate test scenarios for container terminal logistics. The scope is, however, the terminal as a whole. Therefore, scenarios are generated with respect to the arrival and departure of containers at the interface of the terminal to the outside. Expósito-Izquierdo et al. (2012) present an instance generator with varying degrees of difficulty for the pre-marshalling problem at container terminals. Here, the objective is to minimize the number of reshuffle movements so that no further relocations are necessary. In other fields of operations research, parameter-based generation of data has been successfully introduced; see Hall and Posner (2001) for machine scheduling, Kolisch et al. (1995) for project scheduling, and Leyton-Brown and Shoham (2006) and Leyton-Brown (2011) for auction settings.

1.2 Contribution and structure

In this paper, we present an instances generator that focuses on crane systems that handle containers. Consequently, data are generated from the perspective of the crane system operator, resulting in instances, which reflect real life situations. These instances might, but need not be particularly hard instances for the various potential problem settings.

The test data generator is accessible at www.instances.de/dfg and allows test data to easily be generated and shared. Instances can be generated directly by using the web page or with the help of the API documented there. The remainder of this paper is structured as follows. In Section 2, we outline environments where such problem settings arise for which we generate test data. Furthermore, a generic model as a base for the generator is developed. Section 3 details the data provided by the

test data generator, and Section 4 outlines how to derive complete test instances from such data. In Section 5, we present examples for research projects where our generator fits the optimization model investigated. Finally, Section 6 concludes the paper.

2 Problem settings and generic model

In this section, we first outline the relevant aspects of the problem environment with regard to scheduling gantry cranes in different terminals. Then, we present the basic decisions to be made when operating such cranes. Finally, we derive a generic model that captures substantial characteristics of the encountered problem settings.

2.1 Problem environment

2.1.1 Seaport terminals

The material flow through seaport container terminals can be divided into three classes. Land-sea containers arrive by truck or train and are received by gantry cranes. These cranes intermediately store the containers in the block. A situation may arise in which a container is relocated (potentially multiple times) before leaving the block. Later, the container is delivered by the crane to a transport device (if the crane is fixed to the block) or directly to the respective QC. Once the QC receives the container, it loads it onto the vessel. Sea-land containers basically travel through the terminal in reverse order. Finally, sea-sea containers share the travel direction with sea-land containers until they are stored in the block. Afterward, they share the travel direction with land-sea containers.

In stylized schemes, as shown in Fig. 1, we imagine the seaside area and landside area of the terminal as being separated by the storage area. In the landside area, land-sea containers arrive and sea-land containers are loaded on landside transport to depart the terminal. In the seaside area, sea-land containers arrive and land-sea containers are loaded on vessels to depart the terminal.

Considering the scheduling of cranes, both gantry cranes and QCs, we decouple the planning problems for adjacent transport devices from our focus. We assume that the scheduling of, e.g., AGVs has either been performed previously and therefore

imposes constraints for the crane schedule to be taken into account or (conversely) is performed afterward with accepting the crane schedule as a given.
Furthermore, we assume that reshuffle decision have already been made. If reshuffling becomes necessary because of blocked containers, this process usually follows predefined rules. Foresightful reshuffles are typically planned less frequently and with a considerably longer planning horizon. In seaport operations, this situation is commonly called housekeeping (see Ehleiter and Jaehn (2016)).
From the perspective of the crane operator, it is then convenient to consider the following three types of containers. Containers of every type essentially impose a transport job for the crane.

- Storage containers arrive at a certain point of time at the block and pose a transport job from the arrival point to the dedicated storage location.
- Retrieval containers initially have a position in the block and have to be delivered by the crane to a (possibly fixed) position in a handover area. There may be a due date for this delivery.
- Reshuffle containers initially have a position in the block and have to be delivered by the crane to a (possibly fixed) new position in the block.

Note that a single container is first a storage container, possibly a reshuffle container and finally a retrieval container. However, considering that we focus on scheduling

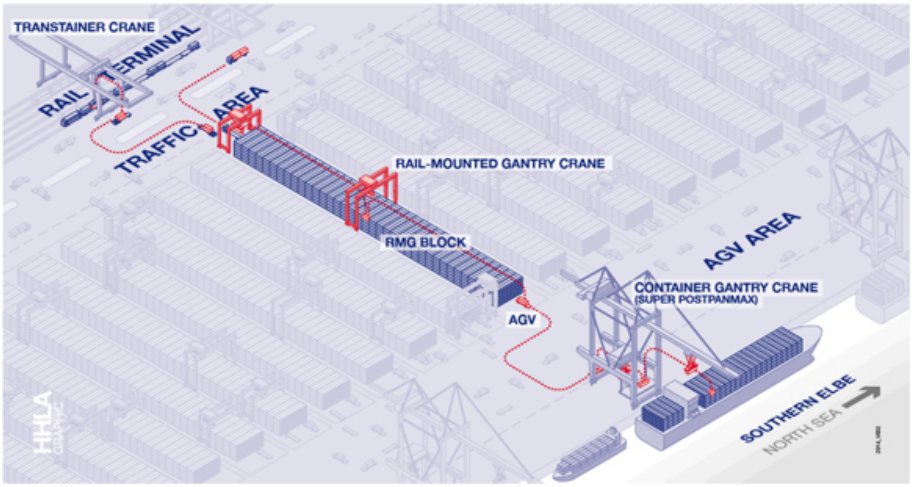


Figure 1: Schematic outline of the HHLA Container Terminal Altenwerder (CTA); HHLA (2017)

problems of cranes, an instance of the problem under consideration may have a planning horizon in which a container is only a storage container or only a retrieval container. However, we explicitly consider containers to be storage containers and retrieval containers.

Each individual container can be described according to the classes described above. However, the entirety of containers imposing transport jobs at a certain moment is hardly a set of independent containers. Depending on the crane under consideration and depending on the transport mode of containers, we may observe batches of containers that are arriving (or leaving) at the same time point or at least within a small time window. Major seaport container vessels have capacities of greater than 18,000 TEUs. Even if only a small portion of these containers are (un)loaded in a single port, a considerable number of containers is available for unloading at the same time or has to be loaded meeting the same deadline. This is obvious for QCs, but this effect also carries over to gantry cranes. Here, we observe periods where retrieval containers clearly outnumber storage containers and vice versa. The same holds true for containers being transported by train on the landside.

The stacking structure may also impose an interdependence of containers. Clearly, retrieval containers being stacked upon each other have to be picked up in the implied order. Conversely, for QCs, a stacking plan is predetermined that details how to locate containers within the vessel. Finally, for gantry cranes, the container loading sequence of vessels may impose precedence constraints since a container that is loaded earlier should leave the block earlier.

2.1.2 Train terminals

The typical layout of a rail-road transshipment terminal is shown in Fig. 2. In addition to containers, the scenario considered here may also include other standardized loading units that can be moved by gantry cranes. These units include swap bodies and semitrailers, although in our terminology, we will simply refer to containers. In contrast to applications at seaport terminals, the handover points for containers are parallel to the tracks that are used by the gantry cranes.

At rail-road transshipment terminals, we may also differentiate between three classes of material flows. Truck-train containers arrive by truck and must be loaded onto

a train. In contrast to the aforementioned situation at seaports, this task can be performed by a single crane move without intermediate storage. However, due to early arrival of the truck or other factors, an intermediate storage of the container might be reasonable or even necessary. Train-truck containers travel in the reverse order, also with a possible intermediate storage. Finally, train-train containers are to be moved from one position on a train to a different position on some other train. In this situation, an intermediate storage might be necessary if a train delivers a container dedicated for a train that has yet to arrive. Another reason for an intermediate drop off might be that the container's dedicated position is located far away. Because the crane should not interfere with all other cranes, which then have to yield way and therefore stop working, the container is handed over to the next crane using the storage area. This process might be supported by automated sorters in the storage area that move the container to its final horizontal position such that it can be moved onto the train by some other crane.

We focus on the processes of the cranes and omit the other vehicles involved, which include trucks, trains, and the aforementioned sorter. The container moves within a rail-road transshipment terminal have a very different structure than those at seaport storage areas. In the latter, there are only a few handover points on the short side of the working area. Here, the largest part of the working area consists of a handover area, namely, all truck lanes and tracks. Therefore, we do not further distinguish the jobs of cranes (e.g., as storage, retrieval, or reshuffle containers) because they can all be identified by an origin and by a destination somewhere in the working area (only moves from truck lanes to truck lanes and from storage area

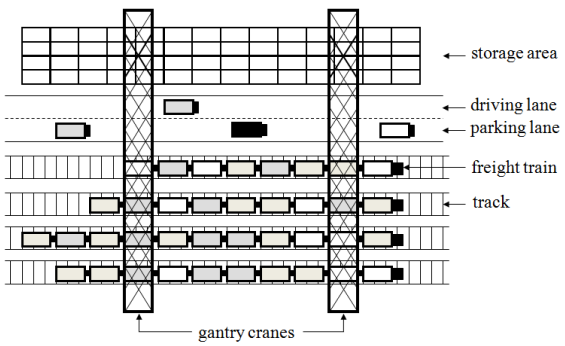


Figure 2: Schematic outline of a rail-road terminal

to storage area do not appear).
Note that the situation at train terminals differs from seaport terminals in even more aspects than those already mentioned. In particular, since there are commonly no containers going from one short side of the block to the other, crane interferences might not appear. Therefore, assigning a fixed working area to each crane is feasible in some cases. Additionally, the degree of capacity utilization is generally lower for the storage area at train terminals than for the one at seaport terminals. In combination with the fact that stacking heights of at most two tiers are common at train terminals, reshuffling aspects are completely exceptional. However, crane scheduling is also an important task here.

2.2 Crane scheduling problems

As outlined in Section 2.1, the sole purpose of the crane system is to conduct transport jobs. When scheduling the cranes at hand to accomplish transport jobs efficiently, there are a variety of decisions to be made depending on the actual crane configuration.

1. For each transport job, it has to be determined whether the corresponding container is set down once or multiple times before reaching its destination. Depending on the answer, the transport job is broken up into several parts, and it has to be determined where the container is set down.

2. For each transport job and each of its parts, it has to be determined which crane the job is assigned to. Note that in the case of a single crane, there is no decision to be made here, and it is highly unlikely that a transport job will be broken up into multiple parts.

3. For each crane, a sequence of assigned parts needs to be determined, prescribing the order in which these parts are conducted.

4. For given sequences of parts for each crane, the actual routing of cranes has to be determined. This routing determines the position of cranes over time.

Each of these decisions has been addressed in the scientific literature. Some approaches address a problem that integrates multiple such decisions, whereas others

focus on a single one. At the very core, however, we can find transport jobs (possibly with additional information) as input to the optimization methods.

2.3 Generic model for crane systems

To describe our generic model, we first outline the commonalities of the crane systems introduced in Section 1 with respect to operating decisions. First, we have a three-dimensional grid where containers are stored. For simplicity, we assume that all containers to be stored are identical (we return to this issue in Section 3.2).
We will refer to the entirety of storage positions as block in the following. Second, although cranes vary in detail, they share the following characteristics. The gantry consists of a beam spanning the entire block along one dimension and two columns that the beam rests upon. These columns (and therefore the entire gantry) are movable along a second dimension of the block. The beam carries a trolley with an attached spreader, which can be moved along the third dimension of the block (this dimension is generally oriented vertically); see Fig. 3. Additionally, the trolley can be moved along the beam. Therefore, by moving the trolley, the gantry and the spreader along the first dimension, the second dimension, and the third dimension, respectively, the spreader can reach each position in the grid. Additionally, the trolley can move outside the block area to handover positions for exchanging containers with other transport devices. This occurs either by moving the gantry along the second dimension or by moving the trolley along the first dimension. We will refer to the positions of containers along the first, second, and third dimensions as trolley slot, gantry slot, and spreader slot, respectively; see Fig. 3. In each dimension, we number slots in the block in orientation of this dimension beginning from 1. Handover positions exist on two opposite sides of the block only. Depending on whether they are reached by moving the entire gantry or only the trolley out of the block, they are addressed by gantry slots 0 and $N + 1$ or trolley slots 0 and $N + 1$, where N is the number of respective slots in the block.
One of the defining features of models is that they represent the original in a simplified way. A common approach to simplify the representation of the physical block is to reduce the number of dimensions considered. Consequently, we find models that consider two dimensions only (gantry slots and trolley slots), and it is even

more common to consider a single dimension (gantry slots) only. These models are well justified in the respective pieces of literature, but we shall at least provide an example for a justified simplification of that type here. Briskorn and Angeloudis (2016) and Briskorn et al. (2016) consider crane systems in which the trolley and the gantry can move in parallel. Considering that the travel time of the gantry is typically considerably higher than the travel time of the trolley, the total time for the trolley to be adjusted in the correct position can be approximated by the gantry travel time. The travel time of the spreader can be encoded in the processing time of the operation that the crane is conducting in the current position. One of the main concerns in both Briskorn and Angeloudis (2016) and Briskorn et al. (2016) is the interference of cranes, which can be accurately captured by their positions according to the gantry slot. Consequently, the corresponding model considers a block of one dimension only. To support such models, we also allow for one- or two-dimensional blocks. Note that we necessarily consider gantry position, whereas spreader slots are only considered if trolley slots are also considered. In the following, the term position refers to a slot number for each of the dimensions considered in the model at hand.

In the block, containers are stored temporarily. Considering a specific problem setting, containers either arrive at one of the handover areas during the planning horizon or are stored within the block at the beginning of the planning horizon. Conversely, containers are either handed over to another means of transport or are stored within the block at the end of the planning horizon. Consequently, each container has an origin; a release date, which marks the point of time it is available to be handled by the crane system; a destination position; and (possibly) a due date, which marks the point of time it is expected to have reached its destination position. Some containers may be more important than others, and there may be precedence relations stating that a certain container can be handled only after another container has been handled.

There may be multiple cranes operating in the same block, in which case they are necessarily aligned along the second dimension. Fig. 3 depicts the model components as described above.

Thus far, the generic part of a problem setting and therefore the part that is to be captured by the instance generator have been described. However, these pieces

of information do not fully specify a problem setting. The type of cranes and the operating regulations need to be defined. As we will show in the following, these are quite unique.

If there is more than one crane operating in one block, then these cranes can hardly operate independently from each other. The actual restrictions depend on the actual crane setting, and therefore, they are highly individual. Consequently, we do not cover these properties by the generic model or by the data provided by the instance generator. In the following, we present some examples for operating regulations, first to exemplify the problem context of our generic model and second to illustrate the substantial variety of regulations justifying us to draw the line between generic model and individual setting here.

- Independent from the actual crane setting, each crane has a specific operating speed. This speed refers to moving speeds along each of the dimensions. Furthermore, movement along one of the dimensions generally consists of an acceleration phase at the beginning, a speed phase in the middle, and a deceleration phase at the end. Whether to accelerate to the full speed also depends on the operating regulations. The characteristics of the acceleration and de-

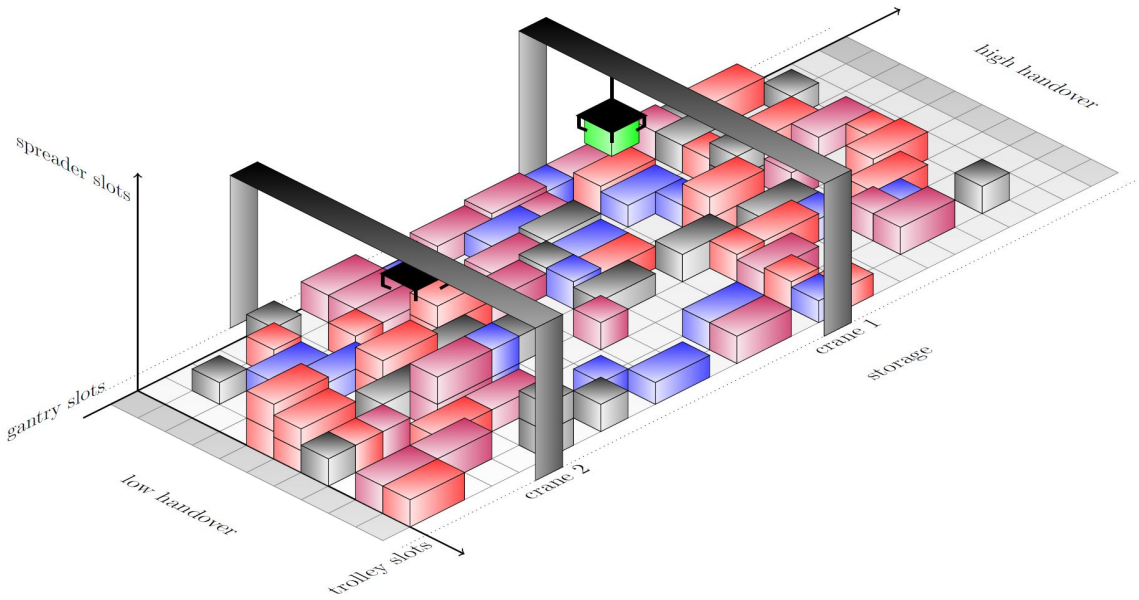


Figure 3: Schematic outline of a typical storage block with crossover cranes

celeration phases are very individual and may depend on whether a container is being carried.

- Although theoretically possible, moving along all three dimensions in parallel is rarely allowed. Often, movements along the first and second dimensions are allowed in parallel. For example, only after the trolley has been placed in the final gantry slot and trolley slot, the spreader can be lowered. Conversely, only after the spreader is at full height, further movements can be started.

- In a twin crane setting, we have two cranes that are identical (or can be assumed to be identical with respect to the operating regulations). Hence, these cranes cannot pass each other, which means that in each point of time, one crane must be at a smaller gantry slot than the other. This is surely the most characteristic operating regulation. Further regulations may involve safety distances between cranes to prevent collisions due to malfunctions or damages caused by swinging containers. Additionally, cranes may not be allowed to move toward each other in parallel, speed reductions must be followed if the distance of two cranes is below a threshold, or the cranes may be restricted to operate in designated operating areas (overlapping only in a rather small area). Note that regarding the above, we can easily consider blocks where more than two identical cranes are operating. In addition to actual container blocks in a storage area, QCs and cranes in train terminals serve as prominent examples here. Note that there may be more than two identical cranes, particularly in these cases.

- In a crossover crane setting, as shown in Fig. 3, we have two cranes, which differ in height and beam length. This allows the smaller crane to move under the larger crane. This may occur under specific circumstances. For example, it may be possible only when the larger crane is not carrying a container. In other settings, it is possible when the larger crane is carrying a container but the trolley of the larger crane has to be moved beside the beam of the lower crane.

- In a triple crossover crane setting, we have two twin cranes and a larger crane. Each pair of larger crane and a single smaller crane can be regarded as a

crossover crane for the purposes of operating regulations. Consequently, combinations of regulations are as described above.

To summarize, we have three types of information related to a problem instance concerned with operating a specific crane setting: first, information about the block itself; second, information about the transport jobs to be conducted; and third, information about the crane system. The former two are covered by our generator, whereas the third is not.

3 Test data

In this section, we describe the data created by the test data generator. We consider three levels of data collection. First, a basic instance describes the situation at a single block and a set of containers to be picked up and delivered somewhere else faced by the crane set-up under consideration. Second, we have files with multiple basic instances generated in a single execution of the generator and thus based on the same parameter setting. Finally, we have projects collecting multiple files that, e.g., have been used in the same computational study. In contrast to basic instances where all pieces of information are collected in a single file, we consider extended instances where complex information is spread over multiple files.
In Section 3.1, we describe the data fields that are given in files explicitly. Common structures imposed by the values of these fields are presented in Section 3.2. In Section 3.3 we describe the concept of projects.

3.1 Data format

A basic instance conveys the data related to a set of containers. We distinguish between containers to be transported and containers not to be transported. We refer to the former as moving containers and to the latter as stationary containers. Moving containers are clearly the focal objects in terms of transportation. Stationary containers are used to describe specifics of the load of the block, which are important particularly if one considers a setting in which the positions for moving containers are to be determined. Depending on the scenario, a basic instance can contain the data elements detailed below. Some of the elements are **mandatory** since we

assume that they are relevant irrespective of the scenario that is considered. Hence, all mandatory elements must be specified by the user so that a basic instance is valid.

- The *Number of Dimensions* is an integer number between 1 and 3 and is **mandatory**.
- The *Block Size* is **mandatory** and defines the number of slots available in each of the dimensions considered.
- The *Handover Position* is a **mandatory** integer number equal to 1 or 2, defining whether the handover position is aligned to dimension 1 (gantry slots) or to dimension 2 (trolley slots).
- Each container has an *ID*, which is a numeric identifier that is unique among containers of the basic instance. The ID is **mandatory**. A basic instance with c containers uses IDs $1, 2, ..., c$.
- We distinguish eight cases regarding the moving characteristics of containers. Consequently, the *Case* of a container is an integer number in $1, \ldots, 8$ (to be described in Section 3.2.2). The case is **mandatory**.
- An *Origin Position* for each container is mandatory. Recall that a position in the block is addressed by gantry slot and possibly by trolley slot and spreader slot.
- The *Destination Position* is addressed following the same pattern as the origin position.
- Each container has a *Stacking Value*, which is a numeric value that enables the user to implement stacking rules.
  For example, containers can be stacked upon each other only if they have identical stacking values, or a container can be stacked upon another container only if the stacking value of the first container does not exceed that of the second container. Note that the stacking rule itself is not covered by a basic instance but rather has to be represented in the model to be evaluated.

- A container's *Release Date* signals the first point of time that a container can be picked up.
- A container's *Due Date* specifies the point of time that the container should be delivered.
- Each container has a numeric *Abstract Value* that might represent other relevant characteristics (e.g., the assigned crane, block, the ship the container is ultimately to be loaded upon or its priority).
- The *Set of Predecessors* is a list of container IDs. This list contains containers that have to be handled prior to the container under consideration. To keep the representation as short as possible, only direct predecessors are listed explicitly.

There are clearly other important characteristics that are essential for a full description of the setting. However, most of them either can be derived from the above parameters or are too complex or diverse to be represented by a reasonable set of parameter values. We discuss this issue in Section 4.1.

3.2 Data structure

In the following, we take a closer look at the design and structure of data elements and provide examples on how they can be interpreted depending on the underlying scheduling model.

Clearly, it is useful to support randomly generated data elements. The generator uses a seed based mersenne twister algorithm to provide a fast generation of consistent pseudo-random numbers. The test data generator offers several discrete integer distribution functions. These functions can be employed for generating various data elements, e.g., positions, release dates, and due dates. The test suite provides functions that generate parameters according to a *uniform distribution* and the *normal distribution*. Furthermore, it offers a *truncated normal distribution* function, which resembles the characteristic of the *normal distribution* but limits realizations to a certain interval. Finally, an *arbitrary distribution* on a finite number of possible realizations is implemented. The user can specify the parameters of each of these distributions. We detail these built-in distributions in Appendix 6.

3.2.1 Block layout

The block layout consists of the three pieces of information. First, the number of dimensions under consideration is given. If we consider only one dimension, it is necessarily the one that the gantry moves along. If two dimensions are considered, we consider gantry slots and trolley slots. Only if three dimensions are considered do we take spreader slots into consideration. The block size is given as integer numbers of slots n_g and possibly n_t and n_s available in the first, second, and third dimensions by the gantry, trolley, and spreader. The handover position is necessarily reached by moving the gantry out of the block and is addressed by gantry slots 0 and $n_g + 1$ if only one dimension is considered. Otherwise, the handover position may be addressed by either gantry slots 0 and $n_g + 1$ or trolley slots 0 and $n_t + 1$. In the former (latter) case, the *Handover Position* evaluates to 1 (2) since it is reached by moving along the first (second) dimension. If three dimensions are considered, then the spreader slot of handover positions is 1, that is, we assume that handovers occur on the floor level.

3.2.2 Cases

We consider eight different cases, that is, moving characteristics, of containers. The cases differ in the origin and destination positions on the storage or handover area.

1. Low handover to storage: The container arrives at the lower handover position, that is, in slot 0, and is to be stored in the block.

2. Storage to low handover: The container is located in the block and is to be delivered to the lower handover position, that is, to slot 0.

3. Inner movements: The container is located in the block and is to be moved to a different location in the block.

4. High handover to storage: The container arrives at the higher handover position, that is, slot $n_g + 1$ or $n_t + 1$, and is to be stored in the block.

5. Storage to high handover: The container is located in the block and is to be delivered to the higher handover position, that is, slot $n_g + 1$ or $n_t + 1$.

6. Low handover to high handover: The container arrives at the lower handover position and is to be delivered to the higher handover position.

7. High handover to low handover: The container arrives at the higher handover position and is to be delivered to the lower handover position.

8. Stationary containers: The container is not moved.

For each case, the number of containers is prescribed by the user when starting the generation of instances. Each single container has properties as outlined in the following.

3.2.3 Positions

Positions are generated by applying the distribution functions detailed in Appendix 6 to derive slots for each dimension under consideration. Stationary containers only have an origin position. Note that by using the built-in distribution functions, we can generate scenarios where locations are spread evenly along each relevant dimension or are clustered around certain values. Positions are in line with the corresponding case. For example, an origin position is located in the respective handover position in cases 1, 4, 6, and 7 in Section 3.2.2, and the destination position is located within the block in cases 1, 3, and 4.
Once the generated position data exceed the corresponding block size dimensions, the position values are repeatedly determined according to the chosen distribution until the position is located inside the given block. Note that this might result in a distribution that differs from the one specified by the user.

3.2.4 Precedence relations

The test data generator offers three generation schemes that allow control over the structure of the precedence relations between containers to some degree.

- The generator supports *arbitrary* precedence relations using the probability density function (1) described by Hall and Posner (2001). The user can specify a target density D, $0 \leq D \leq 1$, of precedence relations. Setting $D = 1$ implies a precedence relation for each pair of containers, whereas $D = 0$ results in no precedence relation.

To guarantee acyclicity of the precedence relations, we never consider containers with a lower ID to be the successor of a container with a higher ID. We use this probability function to determine precedence relations between containers with IDs i_1 and i_2, $i_1 < i_2$.

$$P_{i_1,i_2} = \frac{D(1-D)^{i_2-i_1-1}}{1-D(1-(1-D)^{i_2-i_1-1})}. \tag{1}$$

As stated in Hall and Posner (2001), the value of P_{i_1,i_2} decreases more rapidly for higher values of D as the value of $i_2 - i_1$ increases. This illustrates the intuition that the existence of a precedence relation i_1 to i_2 is more likely to be implied by transitivity in higher density graphs.

- We consider a *chain* structure, that is, we have subsets of containers and precedence relations such that there is a precedence relation between each pair of containers within the same subset and there is no precedence relation between containers of different subsets. Here, the generator offers the ability to specify a minimum number $\underline{n}$ and a maximum number $\overline{n}$ of containers in a subset. The probability for a subset to have size k amounts to $1/(\overline{n} - \underline{n} + 1)$ for each $\underline{n} \leq k \leq \overline{n}$. Note that the chain structure can be found throughout fields of applications of our model. For example, stacks naturally imply linear precedence relations among subsets of containers. In some applications, it will be reasonable to have precedence constraints only between containers of the same case. Consequently, we allow the user to enforce this.

- We consider a *batches of chains* structure. Here, chains are generated as detailed above. Furthermore, chains (that is, the containers connected by being in the same chain) are grouped into numbered batches. No precedence relation is imposed between containers in different chains but in the same batch. However, each container in a batch with a smaller number must be handled before a container in a batch with a higher number.

  The user specifies a number of batches b. Each chain is assigned to a batch number according to a uniform distribution $\mathcal{U}[1, b]$. The presented structure reflects incoming ships or blocks on ships. Here, a batch may represent a ship, and the chains within a batch correspond to stacks stored on the ship.

3.2.5 Other container data

Stacking values, release dates, due dates, and abstract values are also generated by applying the distribution functions detailed in Appendix 6. In many applications, it will be reasonable to have containers with release dates that do not exceed the corresponding due dates. Consequently, we allow the user to enforce this constraint. In that case, the release and due dates are chosen randomly according to the corresponding distribution. Both values are repeatedly determined according to the chosen distribution until the realization is not smaller than the release date. Note that this might result in a distribution that differs from the one specified by the user.

3.3 Projects

Projects represent a collection of multiple files denoted with an arbitrary project name. By using the instances generator the user can specify a *Project Name* under which the respective file is stored. The project name serves as an unique alphanumeric identifier which can be used to recreate or share a collection of files, e.g. to simulate certain aspects of a specific real-world problem. Once all necessary files are attached to the project, the user can publish it and thus make sure that no further changes can be made. This enables the user to share the project without the worry of unauthorized changes and therefore establish fixed benchmark instances for a specific problem setting. A collection of all published projects can be found here: www.instances.de/dfg/published.php

4 Deriving test instances

In the section at hand we describe how to derive full test instances. Note that some parameters necessary for test instances might not be given explicitly by the test data itself, see Section 3. Thus, we need to complement the data to get full test instances. Some of the parameters are implied by the data in the file, others have to be rounded up by the user. Both is detailed in Section 4.1. In Section 4.2 we describe how several files can be combined to create richer instances. In Section 4.3, we present examples on how to derive instances based on practical scenarios.

4.1 Additional information

We use the data presented in 3.1 to derive additional information and describe which data can be estimated by the user.

4.1.1 Capacity of the block

The capacity is defined by the block size parameters. Once all three slot dimensions (gantry, trolley and spreader) are given, the maximum container capacity of the block can easily be derived.

4.1.2 Utilization of the block

The number of all containers in a basic instance divided by the capacity of the corresponding block results in an upper bound for the utilization. Once we only consider retrieval and stationary containers, we receive the utilization at the start of an instance. Whenever we consider all stationary and storage containers with a valid destination on the block, we can calculate the utilization at the end of the process.

4.1.3 Structure of the block

Stationary containers can be used to create a static structure with a certain utilization of the block at the beginning of the process. Once this option is used, the destination positions of storage containers might be undefined and solely depend on the container structure. Depending on how the stationary containers are distributed along the gantry and trolley slots, we achieve different accumulations of containers inside the block.

4.1.4 Assignment of cranes

The test data generator offers no direct assignment of containers to cranes; this is generally predefined by the selected case. Containers with the same moving behavior are often operated with the same crane. However, when a fixed assignment of containers to cranes is required, the abstract value can be used to assign a crane number to each container.

4.1.5 Missing information

There are clearly other characteristics that are important in seaport terminals. However, these characteristics are not suitable to be modeled using the generator and can easily be estimated by the user. Here is a list of missing information that the user should take into consideration.

- Number and speed of cranes
- Type and crossing constraints of cranes
- Size of gantry, trolley and spreader slots

4.2 Extended instances

The user can also generate extended instances by using multiple files to describe a single setting, for instance, to achieve a hierarchical structure with a child file and a parent file. Assume that we generate a file in which containers are clustered into batches of chains. We call this file child, where the batch ID serves as a foreign key. Hence, to achieve individual data for each batch ID, we generate a parent file in which the IDs (primary key) serve as reference to the batch IDs (foreign key) (see Example 4.3.3). Thus, we can generate individual release or due dates for each batch ID. In addition to batches, this procedure could also be used to generate individual data for other container subsets.

4.3 Examples

In the following, we provide practical examples on how to generate instances for specific real-world problems. We use three dimensions and a $30 \times 10 \times 5$ block layout for all example instances. The *Handover Position* is 1 for all Examples. The *Origin and Destination Positions* are uniformly distributed along the block dimensions for all examples. The presented examples are available under the project name 'examples'.

4.3.1 Example 1: Store containers

We store containers inside a block from both sides: the low and high handover areas. There are 100 incoming low handover to storage containers (case 1) and 100 high handover to storage containers (case 4). Furthermore, all containers have uniformly distributed *Release Dates* $\mathcal{U}[0, 500]$ and weightings $\mathcal{U}[1, 3]$. We use the *Abstract Value* to generate the container weightings.

4.3.2 Example 2: Inner movements

A total of 100 inner movement containers (case 3) are re-stacked inside the given storage block. We use the Abstract Value with the arbitrary distribution to assign containers to cranes: 40% are assigned to crane 1 and 60% to crane 2.

4.3.3 Example 3: Batches with child and parent files

A total of 100 low handover to storage containers (case 1) are divided into 5 batches. The chain structure is generated with a minimum and maximum of 5 containers, which results in equal-sized chains. Each chain is assigned to a batch number. We generate the data for each batch in a second file. A batch has normally distributed *Due Dates* $\mathcal{N}[500, 200]$. We generate two files: a child file with the container data and a parent file with the batch data.

4.3.4 Example 4: Online case; due dates with data availability times

A total of 100 low handover to high handover containers (case 6) pass through the block with uniformly distributed due dates $\mathcal{U}[0, 1000]$. The due dates define the arrival of trucks at the high handover area. To simulate an online case we use uniformly distributed data availability times $\mathcal{U}[0, 1000]$ (here: release dates). Furthermore, there is a gap of at least 100 time units between the data availability and the actual due dates.

5 Applications

In this section, we will outline examples for the research projects that could have been supported by our test data generator. Our intention is not to question any

previous results or existing instances. Rather, we want to emphasize that literature supports the purpose of the generator and its broad applicability.

5.1 Seaport terminals

In Briskorn et al. (2016), two cranes at a single block are considered that cannot pass each other. The authors use a simplified model that covers only a single dimension. A set of containers is waiting for pickup in one of the handover positions. A destination position within the block is given for each container. Hence, one of the cranes has to pick up each container and can either move it to its destination position or to another position. In the second case, the other crane picks up the container and delivers it to the destination position to share the total workload between both cranes. The pickup sequence of containers and the delivery position of each container are to be decided for the first crane. For the other crane, the pickup sequence of those containers not delivered to their destinations by the first crane has to be determined. Finally, in case both cranes interfere, the priority of way has to be decided. Test instances for this problem setting can be created using our generator as follows. Obviously, we consider a single dimension only for a block with a certain number of gantry slots. Due to symmetry, we can assume that all containers have a moving behavior according to case 1. This implies the origin position for each container. The destination position can be generated using one of the built-in distribution functions. In Briskorn et al. (2016), no further characteristics of containers are considered, but for future research, it would be easy to enrich instances with container's release dates or due dates. This has been done by Jaehn and Kress (2017) with the help of the generator at hand.

5.2 Robots

In Erdoğan et al. (2014), two robots are positioned on a rail with a non-crossing constraint. Both robots perform transport jobs, that is, they pick up objects at depots at both ends of the rail and deliver them to destination positions along the rail. The objective is to minimize the makespan. In the basic problem, no loading or unloading times are considered and only one dimension (along the rail) is relevant. The execution time of each job depends on the distance traveled between the depot

and destination position. In addition, waiting times in the depot may occur to avoid collisions with the other robot. Further, in Boysen et al. (2015), the setting is enriched by constant loading and unloading times. The paper by Jaehn and Wiehl (2018), which builds up on Erdoğan et al. (2014), uses instances that are created with the presented test data generator and are available under the project name '*trsp*'.

We use cases 1 and 2 for the first and cases 4 and 5 for the second to define the set of jobs performed by each robot. We consider a single dimension (the rail) with a certain number of gantry slots and use the built-in distribution functions to define the positions of the jobs along the rail. We can use the abstract values to add loading and unloading times to an instance. For future research, we could easily further enrich instances by adding another dimension, release dates, due dates or precedence relations.

5.3 Intermodal transshipment terminals

The paper by Boysen and Fliedner (2010b) treats a problem arising in rail-road terminals as depicted in Fig. 2. An operational task is considered in which a set of trains is located within the yard and containers or other loading units have to moved from the trains to trucks and vice versa. It is assumed that the cranes do not have to perform any movements of containers from a train to another train or to and from the storage area. Moreover, trucks delivering or receiving a container are always expected to be in a position such that the container is not moved in the direction of the gantry slots. Thus, it can be ensured that cranes do not interfere if each crane exclusively receives its own working area. The workload of a single container is a given input parameter, which is simply determined by the container's trolley slot. The objective is now to determine crane areas such that the maximum workload of the cranes is minimized.

Boysen and Fliedner (2010b) self-generated test data for which they provide a precise description of the chosen parameters; thus, an application of our instance generator would have been reasonable. Because containers only differ by their position in the storage area, it is sufficient to classify all containers to case 1. Two dimensions are considered. Although the alignment of the handover area is at dimension 2, this

information is not necessary for this specific problem setting. Boysen and Fliedner (2010b) use 50 gantry slots and up to 5 trolley slots. Containers are distributed along the gantry slots using a truncated normal distribution (to be more precise, the train length is chosen to be a truncated normal distribution, which in turn leads to a truncated normal distribution of the containers). Along the trolley slots, the containers are uniformly distributed. No further container-specific data are needed.

6 Conclusion

We present a test data generator that covers different scenarios reflecting common situations at seaport container terminals, warehouses and train terminals. The flexible generation schemes cover a large part of the problem settings already investigated in previous research projects but also aim to cover those that are yet to come.
We support eight common container moving characteristics within a storage area. Depending on the operational stage of the process, some information is already given or has yet to be determined. This information includes position data, container processing sequences, release and due dates, assignment to cranes and others. The combination of the aforementioned characteristics leads to a large selection of possible problem settings. The test data generator and thus the presented generation schemes are accessible for scientists worldwide. Furthermore, the generated instances remain comparable for novel optimization approaches, models, algorithms and tools developed in future projects. A consistent generation scheme avoids unintended biases in the data because the user develops algorithms for the instances (in other words, the problem setting) and not vice versa. Finally, the generated data are reproducible by using the project id, which supports future researchers in establishing fixed benchmark instances for different scenarios.
Various additional requirements might arise in future projects, which are not obvious at this time. Therefore, the design of test instances and the maintenance of the test data generator will be an ongoing process.
We encourage other scientists to use the generator and create consistent data for their specific project. However, the fundamental idea of this paper could also be applied to other operational fields, such as shunting yards, airports or warehouses.

Appendix

The test data generator offers the following distribution functions (version 1.08).

- Uniform distribution function:
  Input: $a, b \in \mathbb{Z}, a \leq b$
  Let $[a, b]$ be the interval that limits the output. We generate a random value x' using uniform distribution $\mathcal{U}(a-1, b)$. To achieve integer numbers, we round this value $x = \lfloor x' \rfloor + 1$. Output: $x \in \{a, a+1, ..., b\}$

- Normal distribution function:
  Input: $\mu, \sigma^2 \in \mathbb{R}^+$
  We generate a random value x' using normal distribution $\mathcal{N}(\mu, \sigma^2)$. To achieve integer numbers, we round this value $x = \lfloor x' + 0,5 \rfloor$. Output: $x \in \mathbb{Z}$

- Truncated normal distribution function:
  Input: $\mu, \sigma^2 \in \mathbb{R}^+$ and $a, b \in \mathbb{Z}, a \leq b$
  Let $[a, b]$ be the interval that limits the output. We generate a random value x' using normal distribution $\mathcal{N}(\mu, \sigma^2)$. To achieve integer numbers, we round this value $x = \lfloor x' + 0,5 \rfloor$. We reject x and restart the function if $x < a$ or $x > b$ applies. Output: $x \in \{a, a+1, ..., b\}$

- Arbitrary distribution function:
  Input: $p_1, p_2, ..., p_C \in \mathbb{R}^+$ and $v_1, v_2, ..., v_C$
  Let C be the number of classes, where each class $c = 1, ..., C$ has a probability p_c and a value v_c. The value can be a number or an alphanumeric value. First, we calculate normalized probabilities $p_c^n = p_c / \sum_{i=1}^{C} p_i \ \forall c = 1, ..., C$. In the next step, the value v_c is assigned to x with the probability of occurrence $P(X = v_c) = p_c^n \ \forall c = 1, ..., C$. Output: $x \in \{v_1, v_2, ..., v_C\}$

References

Alicke, K. (2002). Modeling and optimization of the intermodal terminal Mega Hub. *OR Spectrum*, 24: 1–17.

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

Bierwirth, C. and Meisel, F. (2015). A follow-up survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research*, 244(3):675–689.

Boysen, N., Briskorn, D., and Emde, S. (2015). A decomposition heuristic for the twin robots scheduling problem. *Naval Research Logistics (NRL)*, 62(1):16–22.

Boysen, N., Briskorn, D., and Meisel, F. (2017). A generalized classification scheme for crane scheduling with interference. *European Journal of Operational Research*, 258(1):343–357.

Boysen, N. and Fliedner, M. (2010a). Cross dock scheduling: Classification, literature review and research agenda. *Omega*, 38:413–422.

Boysen, N. and Fliedner, M. (2010b). Determining crane areas in intermodal transshipment yards: The yard partition problem. *European Journal of Operational Research*, 204: 336–342.

Boysen, N., Fliedner, M., Jaehn, F., and Pesch, E. (2013). A survey on container processing in railway yards. *Transportation Science*, 47(3):312–329.

Briskorn, D. and Angeloudis, P. (2016). Scheduling co-operating stacking cranes with predetermined container sequences. *Discrete Applied Mathematics*, 201:70–85.

Briskorn, D., Emde, S., and Boysen, N. (2016). Cooperative twin-crane scheduling. *Discrete Applied Mathematics*, 211:40–57.

Briskorn, D. and Fliedner, M. (2012). Packing chained items in aligned bins with applications to container transshipment and project scheduling. *Mathematical Methods of Operations Research*, 75: 305–326.

Cichenski, M., Jaehn, F., Pawlak, G., Pesch, E., Singh, G., and Blazewicz, J. (2017). An integrated model for the transshipment yard scheduling problem. *Journal of Scheduling*, 20(1):57–65.

Cramton, P., Shoham, Y., and Steinberg, R., editors (2006). *Combinatorial Auctions*. MIT Press, Cambridge.

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

Dorndorf, U. and Schneider, F. (2010). Scheduling automated triple cross-over stacking cranes in a container yard. *OR Spectrum*, 32: 617–632.

Ehleiter, A. and Jaehn, F. (2016). Housekeeping: Foresightful container repositioning. *International Journal of Production Economics*, 179:203–211.

Erdoğan, G., Battarra, M., and Laporte, G. (2014). Scheduling twin robots on a line. *Naval Research Logistics (NRL)*, 61(2):119–130.

Expósito-Izquierdo, C., Melián-Batista, B., and Moreno-Vega, M. (2012). Premarshalling problem: Heuristic solution method and instances generator. *Expert Systems with Applications*, 39(9):8337–8349.

Froyland, G., Koch, T., Megow, N., Duane, E., and Wren, H. (2008). Optimizing the landside operation of a container terminal. *OR Spectrum*, 30: 53–75.

Gonzalez, J., Ponce, E., Mataix, C., and Carrasco, J. (2008). The automatic generation of transhipment plans for a train-train terminal: Application to the Spanish-French border. *Transportation Planning and Technology*, 31: 545–567.

Hall, N. G. and Posner, M. E. (2001). Generating experimental data for computational testing with machine scheduling applications. *Operations Research*, 49:854 – 865.

Hartmann, S. (2004). Generating scenarios for simulation and optimization of container terminal logistics. *OR Spectrum*, 26:171–192.

HHLA (2017). HHLA Container Terminal Altenwerder. https://hhla.de/en/container/cta/how-cta-works.html. Last accessed on Nov 13, 2017.

Jaehn, F. and Kress, D. (2017). Scheduling cooperative gantry cranes with seaside and landside jobs. *Discrete Applied Mathematics*.

Jaehn, F. and Wiehl, A. (2018). Approximation algorithms for the twin robots scheduling problem. *Working Paper*.

Kim, K. and Park, Y.-M. (2004). A crane scheduling method for port container terminals. *European Journal of Operational Research*, 156: 752–768.

Kim, K. H. and Kim, K. Y. (1999). An optimal routing algorithm for a transfer crane in port container terminals. *Transportation Science*, 33:17–33.

Kim, K. H., Lee, K. M., and Hwang, H. (2003). Sequencing delivery and receiving operations for yard cranes in port container terminals. *International Journal of Production Economics*, 84:283–292.

Kim, N. S. and Van Wee, B. (2009). Assessment of CO2 emissions for truck-only and rail-based intermodal freight systems in europe. *Transportation planning and technology*, 32(4):313–333.

Kolisch, R., Sprecher, A., and Drexl, A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. *Management Science*, 41:1693–1703.

Lee, D.-H., Cao, Z., and Meng, Q. (2007). Scheduling of two-transtainer systems for loading outbound containers in port container terminals with simulated annealing algorithm. *International Journal of Production Economics*, 107:115–124.

Lee, D.-H., Hui, Q., and Miao, L. (2008a). Quay crane scheduling with handling priority in port container terminals. *Engineering Optimization*, 40: 179–189.

Lee, D.-H., Hui, Q., and Miao, L. (2008b). Quay crane scheduling with non-interference constraints in port container terminals. *Transportation Research Part E*, 44: 124–135.

Leyton-Brown, K. (2011). Cats website. http://www.cs.ubc.ca/ kevinlb/CATS/. Last accessed on Apr 13, 2017.

Leyton-Brown, K. and Shoham, Y. (2006). *A Test Suite for Combinatorial Auctions*, pages 451–478. In Cramton et al. (2006).

Li, W., Wu, Y., Petering, M. E. H., Goh, M., and de Souza, R. (2009). Discrete time model and algorithms for container yard crane scheduling. *European Journal of Operational Research*, 198:165–172.

Lim, A., Rodrigues, B., Xiao, F., and Xu, Z. (2004). Crane scheduling with spatial constraints. *Naval Research Logistics*, 51: 386–406.

Lim, A., Rodrigues, B., and Xu, Z. (2007). A m-parallel crane scheduling problem with a non-crossing constraint. *Naval Research Logistics*, 54: 115–127.

Martinez, M., Gutierrez, I., Oliveira, A., and Arreche Bedia, L. (2004). Gantry crane operations to transfer containers between trains: A simulation study of a Spanish terminal. *Transportation Planning and Technology*, 27: 261–284.

Narasimhan, A. and Palekar, U. S. (2002). Analysis and algorithms for the transtainer routing problem in container port operations. *Transportation Science*, 36:63–78.

Ng, W. C. (2005). Crane scheduling in container yards with inter-crane interference. *European Journal of Operational Research*, 164:64–78.

Ng, W. C. and Mak, K. L. (2005a). An effective heuristic for scheduling a yard crane to handle jobs with different ready times. *Engineering Optimization*, 37:867–877.

Ng, W. C. and Mak, K. L. (2005b). Yard crane scheduling in container terminals. *Applied Mathematical Modelling*, 29:263–276.

Nossack, J., Briskorn, D., and Pesch, E. (2017). Container dispatching and conflict-free yard crane routing in an automated container terminal. *Working Paper*.

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

Port of Hamburg (2017a). Container handling 1990 to 2016. https://www.hafen-hamburg.de/en/statistics/containerhandling. Last accessed on Apr 13, 2017.

Port of Hamburg (2017b). Top world container ports. https://www.hafen-hamburg.de/en/statistics/top-20-container-ports. Last accessed on Apr 13, 2017.

Rotter, H. (2004). New operating concepts for intermodal transport: The Mega Hub in Hanover/Lehrte in Germany. *Transportation Planning and Technology*, 27: 347–365.

Saanen, Y. and van Valkengoed, M. (2006). Comparison of three automated stacking alternatives by means of simulation. In Kuhl, M. E., Steiger, N. M., Armstrong, F. B., and Joines, J. A., editors, *Proceedings of the winter simulation conference 2005*, pages 1567–1576.

Speer, U. and Fischer, K. (2016). Scheduling of different automated yard crane systems at container terminals. *Transportation Science*, 51(1):305–324.

Speer, U., John, G., and Fischer, K. (2011). Scheduling yard cranes considering crane interference. *Lecture Notes in Computer Science*, 6971:321–340.

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

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

Vis, I. F. A. and Carlo, H. J. (2010). Sequencing two cooperating automated stacking cranes in a container terminal. *Transportation Science*, 44:169–182.

Zhu, Y. and Lim, A. (2006). Crane scheduling with non-crossing constraints. *Journal of the Operational Research Society*, 57: 1464–1471.

III Research Paper R2

Approximation algorithms for the twin robots scheduling problem

Florian Jaehn[1], **Andreas Wiehl**[2]

[1] Helmut Schmidt University, [2] University of Augsburg

This work has been supported by the German Science Foundation (DFG) through the grant „*Scheduling mechanisms for rail mounted gantries with respect to crane interdependencies*“ (JA 2311/2-1).

This is a pre-print of an article published in Journal of Scheduling.
The final authenticated version is available online at:
https://doi.org/10.1007/s10951-019-00631-9

Abstract

We consider the $\mathcal{NP}$-hard Twin Robot Scheduling Problem (TRSP), which was introduced by Erdoğan et al. (2014). Here, two moving robots positioned at the opposite ends of a rail have to perform automated storage and retrieval jobs at given positions along the gantry rail with a non-crossing constraint. The objective is to minimize the makespan. We extend the original problem by considering pick-up and delivery times and present exact and approximation algorithms with a performance ratio of ≈ 1.1716 for large instances. Further, we compare the presented algorithms in a comprehensive numerical study.

Keywords: Automated storage and retrieval systems, non-crossing constraints, approximation algorithms, crane scheduling.

1 Introduction

The first automated storage and retrieval systems (ASRS) originated in the 1950s, and since then, numerous applications in production and logistics have emerged. The main goal of these systems is to make the storage and retrieval of goods more efficient. This is achieved by replacing human labor with automated working units, such as industrial robots, cranes or shuttles. Advantages compared to the non-automated pendant are, among others, lower labor and land costs, reduced error rates, and a higher throughput. This compares to high initial investment costs and less flexibility, particularly during peak hours (see Bozer and White (1984)). Depending on the practical application, different types of ASRS have arisen over time. The survey of Roodbergen and Vis (2009) distinguishes between the movement directions, the load to be handled, and the type of racks (stationary or movable). Moreover, this results in various optimization problems, for instance, to determine the optimal path of a working unit (such as cranes or robots). Unlike other routing problems, like the $\mathcal{NP}$-hard Traveling Salesman Problem, Gademann et al. (1999) show that a special case of sequencing under certain storage policies at ASRS can be solved in polynomial time. Further, Kim and Kim (1997, 1999) describe algorithms for the optimal routing of a single crane at seaport terminals.

Both cases refer to the use of a single working unit, however productivity can be increased by using multiple units. Once several units work on an overlapping area, it is important to minimize interferences between them to ensure a smooth and efficient process. The first practical application described in literature can be found in the steel mill and cooper industry, where industrial cranes transport materials between furnaces and converters (see Lieberman and Turksen (1981)). Since then, many other applications with multiple working units have been successfully established. Besides industrial applications, others can be found in the logistical area, for instance, at ASRS or in container transportation (at seaports and rail-terminals). The survey of Boysen et al. (2017) provides a comprehensive overview of different systems with multiple working units. The presented classification scheme differentiates three characteristics. These can be briefly summarized as follows: First, the terminal layout, consisting of the number of working units, the dimensions considered and the alignment of the depots or transfer points. Second, other characteristics

such as movement, travel speed, information availability, precedence relations and the input data (like release and due date of items to be transported). And last, the objective, which is either productive orientated (e.g. minimizing the makespan or empty drives) or service orientated (e.g. minimizing lateness and earliness).
The paper on hand considers a problem that can be denoted by [1D, 2, ends|mv^X | C_{max}] using the classification scheme of Boysen et al. (2017). This defines a 1-dimensional layout with two working units and two handover areas at each end of the shared pathway [1D, 2, ends]. The working units store and retrieve items along the storage and handover area with a non-crossing constraint. Both units respect a constant travel speed, whereas the movements in other directions (e.g. trolley and spreader) can be neglected [mv^X]. The objective is to minimize the makespan [C_{max}].

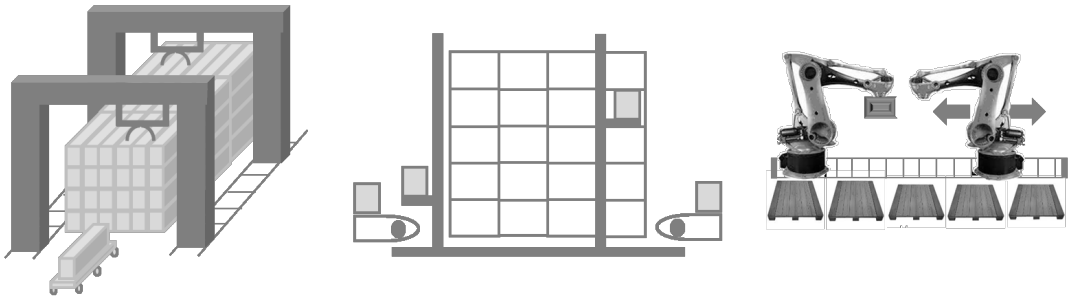


Figure 1: Examples for the applications of TRSP

We identify three practical applications for this generic model. First, the original real-world motivation for this problem can be found in the industrial area. Here two robots are (de-)palletizing boxes at given depots along a shared pathway (Fig. 1, right). The so-called Twin Robots Scheduling Problem (TRSP) was first introduced by Erdoğan et al. (2014). A similar problem with rail-bound transportation is formulated as a Blocking Job Shop by Bürgy and Gröflin (2016), who consider instances with up to 4 robots working on a common rail. Second, Kung et al. (2012, 2014) describe similar problems where two shuttles work jointly together in an automated storage and retrieval system (Fig. 1, middle). Last, twin automatic rail-mounted gantry cranes (Fig. 1, left) working on a common rail are used to ensure an efficient storage and retrieval of the containers at transshipment terminals.

Despite the fact that the spreader moves in a 3-dimensional space (gantry, trolley and spreader), we reduce it to a 1-dimensional problem. We neglect movement times of trolley and spreader so that all pick-up and delivery times are independent of previous operations. This assumption is commonly used in literature for yard crane operations (see Briskorn et al. (2016); Ehleiter and Jaehn (2016); Jaehn and Kress (2017)). A practical study about the sequencing of twin automated stacking cranes is presented by Hu et al. (2016). For a comprehensive study about the performance of other automated yard crane systems we refer to Speer and Fischer (2016).

In addition to practical applications, a few theoretical insights have already appeared in the literature. Erdoğan et al. (2014) provide a proof of $\mathcal{NP}$-hardness and present exact and heuristic solution procedures in which one procedure provides a performance ratio of $\frac{3}{2}$. Boysen et al. (2015) extend the problem with constant pick-up and delivery times $d > 0$ and provide a heuristic decomposition procedure tested with large instances with up to 500 jobs. Carlo and Martínez-Acevedo (2015), Briskorn et al. (2016) and Jaehn and Kress (2017) consider the case with cooperative rail-mounted gantry cranes at seaport terminals. Here preemptive container moves are allowed.

The paper on hand is structured as follows. The following section provides a formal problem description and a mathematical model including practical extensions of the TRSP and its properties. In Section 3, we present exact and approximative algorithms for the TRSP. We investigate the performance of the procedures in a comprehensive numerical study in Section 4. Finally, Section 5 concludes the article.

In the further course we will use the term robot as a designation for the working unit, however this study also refers to other applications in which the storage and retrieval process is carried out by cranes or shuttles.

2 Problem description

Two moving robots Robot 1 and Robot 2 are working on the same rail with a non-crossing constraint. The rail inside the storage area is divided into S slots that both robots can reach. At each end of the rail there is a depot that can only be reached by the corresponding robot, slot 0 is assigned to Robot 1 and slot $S + 1$ to Robot 2. Both robots share an equal speed of one slot per time unit. At the beginning

and at the end of the process the robots are located inside their depots. To avoid collisions, both robots are expected to respect a security distance of one slot unit. We assume a fixed assignment of jobs $J = J^1 \cup J^2$ to robots. Robot 1 performs the set of jobs $J^1 = \{1, ..., m\}$, whereas Robot 2 performs the jobs $J^2 = \{m+1, ..., n\}$. Each job $i \in J$ consists of a pick-up operation at slot a_i followed by a delivery operation at slot b_i. The minimum number of time units job i covers from the pick-up to the delivery destination is given by $\delta_i = |a_i - b_i|$. We do not consider inner movement jobs. Thus, either a_i or b_i corresponds to the depot at slot 0 or $S+1$, respectively. This defines that a job is either a storage ($a_i \in \{0, S+1\}$) or a retrieval operation ($b_i \in \{0, S+1\}$). Without loss of generality, we assume that the jobs in J^1 and J^2 are indexed with respect to decreasing distances from their depot, i.e. $\delta_i \geq \delta_{i+1}$. Moreover, there is a given time for the pick-up $q_i^a \geq 0$ and delivery $q_i^b \geq 0$ operation of each job $i \in J$. Further, we do not consider cooperative robots, preemptive jobs or idle times between the pick-up and delivery operation.

Given this input data, a feasible solution requires two components. First, we define a processing sequence of the jobs J^1 and J^2 for each robot, respectively. The jobs are then intended to be processed without waiting time of the robot if possible. Second, in order to resolve potential conflicts between the jobs of J^1 and J^2, we define another sequence that regulates which job is prioritized and which has to wait. We combine both requirements in sequence π, which defines the processing order of both robots as well as the priority order of jobs. Again, we assume that each robot performs its jobs consecutively until two jobs $i \in J^1$ and $j \in J^2$ appear so that both robots interfere. This conflict is resolved by letting Robot 1 wait if j appears before i in π. Otherwise, Robot 2 waits. The objective is to find a sequence $\pi = (\pi(1), ..., \pi(n))$ for all jobs J so that the makespan is minimal, where $\pi(k)$ is the kth job in π, $k \in \{1, ..., n\}$. In Section 2.1, we show how to derive a complete schedule from a given sequence π.

We divide the problem into four hierarchically structured scenarios. Scenario D is the problem described above and A, B, C are special cases with the following properties.

- Scenario A: Either all jobs are storage operations or all jobs are retrieval operations. We neglect pick-up and delivery times ($q_i^a = q_i^b = 0$).

- Scenario B: Either all jobs are storage operations or all jobs are retrieval operations. All jobs have equal pick-up and delivery times ($q_i^a = q_i^b = d \geq 0$).

- Scenario C: Either all jobs are storage operations or all jobs are retrieval operations. The pick-up and delivery times can have varying values ($q_i^a \geq 0, q_i^b \geq 0$).

- Scenario D: There are storage and retrieval operations with varying pick-up and delivery times in the same process.

The classification of the problems is a result of previous work in this area. Scenario A is the original problem introduced by Erdoğan et al. (2014), the pick-up and delivery times are omitted since industrial robots can perform this process very quickly. With respect to the situation at container terminals, Boysen et al. (2015) introduce constant pick-up and delivery times (Scenario B). In the paper on hand we extend the problem to varying pick-up and delivery times (Scenario C). Especially at the container terminals the times can be rather different, for instance, because of the stacking heights, re-stacking processes or other delays. To get even closer to the practical problem we have introduced Scenario D, which includes storage and retrieval operations in the same process.

2.1 Schedule generation

First, we define all necessary parameters and variables to create a complete schedule. The dependent variable p_i indicates the temporal position of job $i \in J$ in π in time units. This is the point in time when half of the pick-up time (if job i is a retrieval operations) or delivery time (if job i is a storage operations) is performed by the corresponding robot.

Let parameter $start_i$ represent the number of time units from the start of job i to reach point p_i, provided that the execution of i starts at the corresponding depot. Further, let parameter end_i represent the time units from point p_i back to the depot until the delivery operation of job i is fully executed. We calculate these parameters as follows:

$$start_i = \begin{cases} q_i^a + \delta_i + \frac{1}{2}q_i^b & \text{for } b_i \in \{1, ..., S\} \text{ (storage operation)} \\ \delta_i + \frac{1}{2}q_i^a & \text{for } a_i \in \{1, ..., S\} \text{ (retrieval operation)} \end{cases} \quad \forall i \in J \qquad (1)$$

$$end_i = \begin{cases} \frac{1}{2}q_i^b + \delta_i & \text{for } b_i \in \{1, ..., S\} \text{ (storage operation)} \\ \frac{1}{2}q_i^a + \delta_i + q_i^b & \text{for } a_i \in \{1, ..., S\} \text{ (retrieval operation)} \end{cases} \quad \forall i \in J \qquad (2)$$

Let completion time C_i be the earliest point in time the robot could be back in its depot after executing job i, i.e. no detour appears, for example, because of executing another job before returning to the depot. The completion time of job i depends on point p_i and can easily be derived from it: $C_i = p_i + end_i$. Note that for a given solution π, a robot is not necessarily in its depot at time C_i, i.e. a job is directly executed after job i without a detour to the depot (see job 3 in Fig. 2).
Let parameter d_{ij} be the minimum separation time between p_i and p_j due to robot processing if job i follows j in schedule π. First, we consider the case when i and j are performed by the same robot. Note that if a storage job i is followed by a retrieval job j, the robot moves directly from slot b_i to slot a_i. We calculate d_{ij} as follows:

$$d_{ij} = \begin{cases} \frac{1}{2}q_i^b + |b_i - a_j| + \frac{1}{2}q_j^a & \text{for } b_i, a_j \in \{1, ..., S\} \\ end_i + start_j & \text{otherwise} \end{cases} \quad \forall (i,j) \in J^1 \times J^1 \cup J^2 \times J^2 \qquad (3)$$

Next job i and j are processed by opposite robots. Let d_{ij} be the minimum separation time between p_i and p_j so that the corresponding robots of job i and j are not conflicting. Consequently, d_{ij} is $-\infty$ whenever both robots are not conflicting in any case, i.e. $\delta_i + \delta_j \leq S$. Further, let q_i^δ denote the pick-up or delivery operation, respectively, that takes place inside the storage area. Hence, q_i^δ is equal to q_i^b for storage and equal to q_i^a for retrieval operations. We calculate d_{ij} as follows:

$$d_{ij} = \begin{cases} -\infty & \delta_i + \delta_j \leq S \\ \delta_i + \delta_j - S + \frac{1}{2}(q_i^\delta + q_j^\delta) & \text{otherwise} \end{cases} \quad \forall (i,j) \in J^1 \times J^2 \cup J^2 \times J^1 \qquad (4)$$

In the following we present two algorithms to generate complete schedules under a given input sequence π. A complete schedule provides position data of all jobs in π and the objective value. This includes all dependent variables w_i, p_i and C_i $\forall i \in J$. We distinguish between two types of schedules, where Algorithm 1 generates a semi-active schedule and Algorithm 2 an active schedule. In a semi-active schedule waiting times occur only to overcome conflicts with the robot on the other side. Variable w_i denotes the waiting or idle time of robot c until a conflict-free execution of job i is possible. If job j^c is executed directly before job i by robot c then the waiting time is: $w_i = p_i - p_{j^c} - d_{j^c i}$ holds. If job i is the first job executed by robot c then $w_i = p_i - start_i$.

Definition 1 (Semi-Active Schedule)**.** *A feasible schedule is called semi-active if no operation can be positioned earlier without changing the processing and priority order of jobs in sequence π.*

Algorithm 1: Semi-active schedule - $\mathcal{O}(n^2)$

0. Initialization:
Sequence π
j^1, j^2: Indicates the last job processed by Robot 1 and 2
1. Compute:
for $k = 1$ to n **do**
  $c \in \{1, 2\}$: Denotes the robot that performs job $\pi(k)$
  $p_{\pi(k)} = \max\{start_{\pi(k)}, p_{j^c} + d_{j^c \pi(k)}\}$
  for $l = 1$ to $k - 1$ **do**
    if $p_{\pi(l)} - d_{\pi(k)\pi(l)} < p_{\pi(k)} < p_{\pi(l)} + d_{\pi(l)\pi(k)}$ **then**
      Resolve conflict with job $\pi(l)$: $p_{\pi(k)} = p_{\pi(l)} + d_{\pi(l)\pi(k)}$
    end if
  end for
  Set waiting time: $w_{\pi(k)} = p_{\pi(k)} - p_{j^c} - d_{j^c \pi(k)}$
  Set completion time: $C_{\pi(k)} = p_{\pi(k)} + end_{\pi(k)}$
  Update last job processed by c: $j^c = \pi(k)$
end for
return makespan $C_{max} := \max\{C_{j^1}, C_{j^2}\}$

Fig. 2 shows an example of a semi-active schedule with four jobs and a makespan of 42. None of the four jobs can be positioned earlier without changing the order in π. There are three possible input sequences π for Algorithm 1 to obtain exactly the schedule in Fig. 2: $(1, 2, 3, 4)$, $(1, 3, 4, 2)$ and $(1, 3, 2, 4)$. In all sequences, the processing order of Robot 1 is $(1, 2)$ and $(3, 4)$ of Robot 2, further job 1 has priority over job 3.

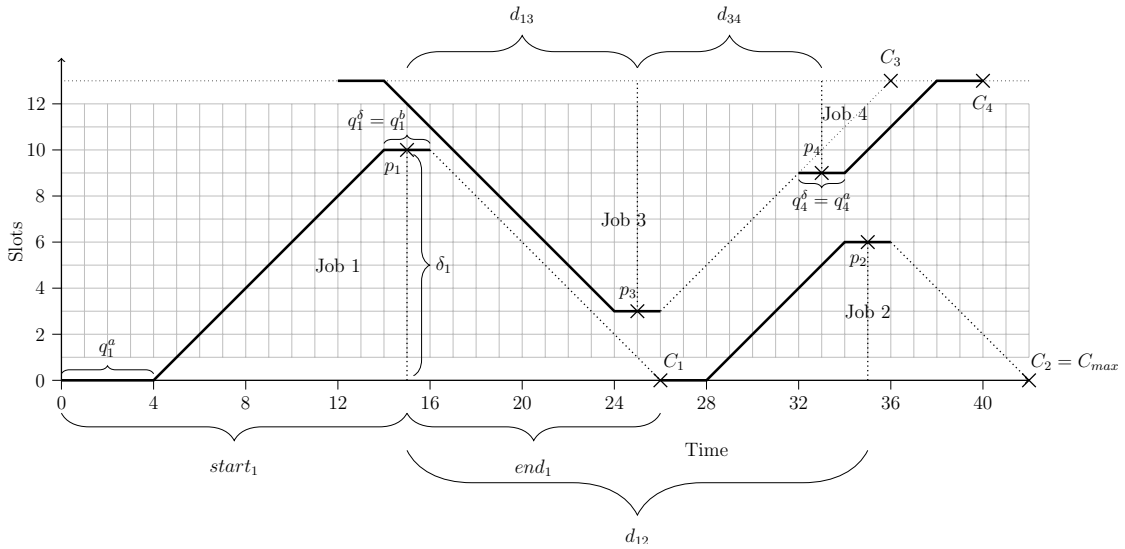


Figure 2: Schematic representation of a semi-active schedule

Definition 2 (Active Schedule)**.** *A feasible schedule is called active if it is not possible to construct another schedule, through changes in the order of π, with at least one job positioned earlier and no job positioned later.*

Algorithm 2 changes the order in sequence π so that Algorithm 1 generates an active schedule. The order in π is only modified if an earlier execution of a job is possible without changing the position of its predecessors in π. The algorithm examines each job iteratively according to sequence π. We move each job to the earliest position in π, under consideration of the minimum separation times to its predecessors in π. We obtain a modified sequence π which can be converted into an active schedule using Algorithm 1, as no job can be positioned earlier with no job positioned later. This always applies because every possible non-conflicting position before a job is checked in Algorithm 2.

Algorithm 2: Active schedule - $\mathcal{O}(n^2)$

0. Initialization:
Input sequence π
1. Compute:
for $k = 1$ to n **do**
  $p_{\pi(k)} = start_{\pi(k)}$
  $k^* = 1$: Indicates the new position of job $\pi(k)$ in π
  2. Check for conflicts:
  for $l = 1$ to $k - 1$ **do**
    if $p_{\pi(l)} - d_{\pi(k)\pi(l)} < p_{\pi(k)} < p_{\pi(l)} + d_{\pi(l)\pi(k)}$ **then**
      Resolve conflict with job $\pi(l)$: $p_{\pi(k)} = p_{\pi(l)} + d_{\pi(l)\pi(k)}$
      $k^* = l + 1$
      Go to **2.**
    end if
  end for
  Move job $\pi(k)$ from position k to k^*
end for
return Sequence π
Use Algorithm 1 to get an active schedule with sequence π

The semi-active schedule depicted in Fig. 2 can be converted into an active schedule. We position job 4 earlier at time $p_4 = 5$ without changing the positions of job $1, 2$ and 3. After that, no job can be positioned earlier without changing the positions of other jobs. For instance, with input sequence $\pi = (1, 3, 4, 2)$, Algorithm 2 returns $\pi = (4, 1, 2, 3)$, which can be converted into an active schedule using Algorithm 1. Obviously, an active schedule is also semi-active, however the reverse is not necessarily true. A schedule with an optimal makespan can either be active, semi-active or non-semi-active. However, there is at least one active schedule that is optimal. Thus, we can restrict ourselves to finding the best (semi-)active schedule.

2.2 Mathematical program

The following mathematical program represents the TRSP. It should be noted that a schedule derived from a solution of this model (i.e. the dependent variables p_i) is not necessarily a semi-active or active schedule.
Binary variables $z_{ij} \in \{0, 1\}$ ensure that separation times d_{ij} (Equations 3 and 4) are adhered to. Either $z_{ij} = 1$, then the minimum separation time d_{ij} from point p_i to p_j is respected, or $z_{ij} = 0$, then the minimum separation time d_{ji} from point p_j to p_i is respected. The auxiliary variable C_{max} is equal to the makespan. The

TRSP including all extensions can be formulated as follows:

Minimize

$$C_{max} \tag{5}$$

subject to

$$C_{max} \geq p_i + end_i \qquad \forall i \in J \tag{6}$$

$$p_i \geq start_i \qquad \forall i \in J \tag{7}$$

$$z_{ij} + z_{ji} = 1 \qquad \forall i, j \in J : i \neq j \tag{8}$$

$$p_j - p_i \geq d_{ij} - M \cdot z_{ji} \qquad \forall i, j \in J : i \neq j \tag{9}$$

$$z_{ij} \in \{0, 1\} \qquad \forall i, j \in J \tag{10}$$

Objective function (5) minimizes the makespan. Constraints (6) and (7) ensure that both robots are in their respective depot at the beginning and end of the process. We derive the objective value C_{max} from constraints (6). Equations (8) ensure that either $z_{ij} = 1$ or $z_{ji} = 1$ applies. Constraints (9) ensure that the minimum separation time between p_i and p_j is respected. And finally, (10) ensure that the variable z_{ij} is binary.

The problem without pick-up and delivery times (Scenario A) was shown to be $\mathcal{NP}$-hard by Erdoğan et al. (2014), even if each slot is only visited once. The generalization of this problem which allows multiple visits at each slot is $\mathcal{NP}$-hard in the strong sense. As Scenario A is a special case of B, C, and D, respectively, all Scenarios are $\mathcal{NP}$-hard.

2.3 Properties

Property 1 (Lower bound)**.** *Given is an instance of the TRSP. If we relax the non-crossing constraint, we obtain a lower bound using Algorithm 3 in* $\mathcal{O}(n \log n)$.

Let C^*_{max} be the optimal makespan and Ω_c be the minimal travel time of robot c if we relax the non-crossing constraint, then the following holds: $C^*_{max} = \max\{\Omega_1 +$

$w_i, \Omega_2 + w_j\} \; \forall i \in J^1$ and $\forall j \in J^2$. Hence, $\Omega := \max\{\Omega_1, \Omega_2\}$ is a lower bound, as $C^*_{max} \geq \Omega$ applies for all instances of the TRSP.

Executing storage or retrieval job $i \in J$ in a single command takes $e_i = q_i^a + q_i^b + 2\delta_i$ time units. This includes the time from the depot to pick-up slot a_i and the time from delivery slot b_i back to the depot. Obviously, a dual command of storage job $i \in J^S$ and retrieval job $j \in J^R$ leads to an improvement of $2\min\{\delta_i, \delta_j\} \geq 0$ time units compared to two successive single commands.

We formulate this as an assignment problem using a complete bipartite graph $G = (J^S, J^R, E)$. Edges $(i, j) \in E$ represent a dual command of storage job i and retrieval job j with a weight of $2\min\{\delta_i, \delta_j\}$. If necessary, we fill up either J^S or J^R with dummy jobs so that both subsets contain an identical number of jobs (see Fig. 3). A combination including a dummy job D represents a single command with zero weight and distance $\delta_D = 0$. The objective is to find a maximum weighted matching $M \subseteq E$ where all vertices in J^S and J^R are endpoints of exactly one edge in M.

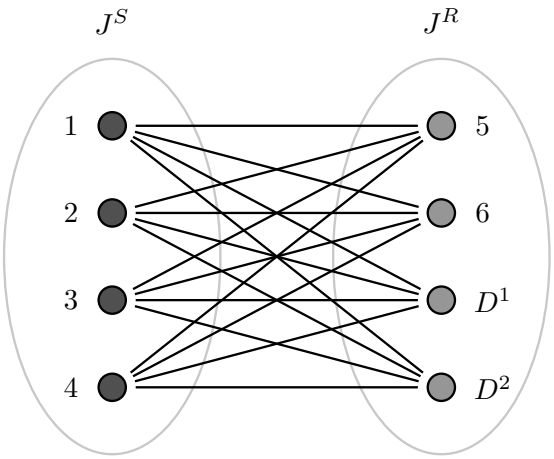


Figure 3: Complete bipartite graph $G = (J^S, J^R, E)$ with 8 jobs

We calculate the minimum travel times Ω_1 and Ω_2 as follows: $\Omega_c = \sum e_i - \max_M \sum_{(i,j)\in M} 2\min\{\delta_i, \delta_j\} \; \forall c \in \{1, 2\}$. Obviously, we only need to solve two assignment problems which can be efficiently solved in $\mathcal{O}(n^3)$ using the Hungarian Method (see Kuhn (1955)).

However, with Algorithm 3 we form optimal sets of storage and retrieval jobs simply by sorting subsets of jobs in descending order according to δ_i in $n \log n$ steps.

Algorithm 3: Lower bound - $\mathcal{O}(n \log n)$

0. Initialization:
J^{Sc}, J^{Rc}: Sets with storage and retrieval jobs to be processed by robot $c \in \{1, 2\}$
We sort all sets in descending order according to δ_i.
Ω_c: Lower bound of robot c
1. Compute:
for $c = 1$ to 2 **do**
  $\Omega_c = \sum_{i=1}^{J^{Sc}} e_i + \sum_{i=1}^{J^{Rc}} e_i$
  for each i in J^S **do**
    if J^R is empty **end for**
    let j be the next job in J^{Rc}; remove j from J^{Rc}
    let (i, j) be a new combined job
    $\Omega_c = \Omega_c - 2\min\{\delta_i, \delta_j\}$
  end for
end for
return $\Omega := \max\{\Omega_1, \Omega_2\}$

Proof: First, we consider the time when each job is executed separately: $\Omega_c = \sum_{i=1}^{J^{Sc}} e_i + \sum_{i=1}^{J^{Rc}} e_i \; \forall c \in \{1, 2\}$. Algorithm 3 combines storage job i with retrieval job j so that Ω_c decreases by $2\min\{\delta_i, \delta_j\} \geq 0$. After that Algorithm 3 forms another combination with storage job k and retrieval job l so that Ω_c decreases by another $2\min\{\delta_k, \delta_l\} \geq 0$. This is done until each job in J^S is paired with one job in J^R. Due to the sorting in Algorithm 3, the following applies: $\delta_i \geq \delta_k$ and $\delta_j \geq \delta_l$. A swap of storage jobs i and k or retrieval jobs j and l, does not lead to further reductions of Ω_c. However, Ω_c increases by up to $2(\delta_i - \delta_k)$ or $2(\delta_k - \delta_l)$ time units. Hence, the sorting proposed by Algorithm 3 leads to optimal combinations of storage and retrieval jobs, as no swap of jobs leads to a reduction of Ω_c, respectively Ω. □

Property 2 (Precedence relations)**.** *Given jobs $i, j \in J^1$ or $i, j \in J^2$ with: $a_i = a_j$, $b_i = b_j$, $q_i^a = q_j^a$ and $q_i^b = q_j^b$, then there exists an optimum schedule π in which i precedes j (denoted by $i \prec j$), i.e.*

Proof: As i and j are identical, swapping their position in π obviously does not affect the objective function. □

Let $\pi = \{\pi(1), ..., \pi(x), ..., \pi(n)\}$ be a solution with $x < n$, we call $\pi' = \{\pi(1), ..., \pi(x)\}$ a partial solution. Let $J(\pi') = J^1(\pi') \cup J^2(\pi')$ be the jobs in partial solution π'. Let $C^c(\pi') = \max\{C_i\} \; \forall i \in J^c(\pi')$ be the completion time of robot $c \in \{1, 2\}$ in partial

solution π'. Let $C_{max}(\pi')$ be the optimal makespan under consideration of partial solution π'.

Definition 3 (Domination)**.** *A partial solution π' dominates π'' if $J(\pi') = J(\pi'')$ and $C_{max}(\pi') \leq C_{max}(\pi'')$ applies.*

Property 3 (Domination rule: Scenario A, B, C)**.** *Partial solutions π' dominates π'' if $J(\pi') = J(\pi'')$, $C^1(\pi') \leq C^1(\pi'')$, $C^2(\pi') \leq C^2(\pi'')$ and $|C^1(\pi') - C^2(\pi')| \leq S$ holds.*

Proof: Proof by contradiction. Assume $C_{max}(\pi') > C_{max}(\pi'')$ holds. Given equal completion times $C^c(\pi') = C^c(\pi'')$ $c \in \{1, 2\}$ for both partial solutions, with $|C^1(\pi') - C^2(\pi')| \leq S$. We consider jobs $J^* = J \setminus J(\pi'')$ which are not included in partial solution π''. Jobs J^* can complement partial solution π' without conflicting with any of the already scheduled jobs $J(\pi')$.
Let job $k \in J^*$ start at time x and job $l \in J(\pi')$ be complete at time y. Job k and job l are executed by opposite robots and hold a potential conflict ($\delta_k + \delta_l > S$ applies). Both jobs are not conflicting if $|x + start_k - (y - end_l)| \geq \delta_k + \delta_l - S + \frac{1}{2}(q_k^\delta + q_l^\delta)$ applies (Equation 4). We obtain a minimum for $start_k + end_l$ if k is a retrieval and l is a storage operation with $\delta_k + \delta_l + \frac{1}{2}(q_k^\delta + q_l^\delta)$. We obtain a maximum for $\delta_k + \delta_l - S$ if $\delta_k = \delta_l = S$. This results in $|x - y + 2S + \frac{1}{2}(q_k^\delta + q_l^\delta)| \geq S + \frac{1}{2}(q_k^\delta + q_l^\delta)$.
Thus, if $|x-y| \leq S$ holds, job $k \in J^*$ is not conflicting with job $l \in J(\pi')$. This shows that jobs J^* provide a valid extension to partial solution π' if $|C^1(\pi') - C^2(\pi')| \leq S$ holds, with an objective value equal to $C_{max}(\pi'')$. This contradicts with the assumption $C_{max}(\pi') > C_{max}(\pi'')$.
Obviously, this also applies if $C^c(\pi') \leq C^c(\pi'')$ $c \in \{1, 2\}$. □
Property 3 also holds for instances of Scenario D if robots $c \in \{1, 2\}$ start from their respective depots at time $C^c(\pi')$ and $C^c(\pi'')$. This is true if the last job executed by Robot 1 and 2 is a retrieval job.

Property 4 (Waiting time, semi-active schedule: Scenario A, B)**.** *Given a semi-active schedule, then the waiting time w_i never exceeds its execution time e_i, i.g. $w_i < e_i$ holds $\forall i \in J$.*

Proof: Given a semi-active schedule and a sequence π. For Scenario A and B, $q_i^a = q_i^b = d \geq 0$ holds and either all jobs are storage or all jobs are retrieval operations, thus $e_i = 2\delta_i + 2d$ $\forall i \in J$.

Proof by contradiction. Assume $w_{\pi(k)} \geq e_{\pi(k)}$ holds. Job $\pi(k)$ starts at time x and is in conflict with job $\pi(l)$. In a semi-active schedule each job is positioned at the earliest non-conflicting point in time, which is: $p_{\pi(k)} = p_{\pi(l)} + d_{\pi(l)\pi(k)}$. This results in a waiting time of: $w_{\pi(k)} = p_{\pi(k)} - start_{\pi(k)} - x$.

The maximum $p_{\pi(l)}$ so that the conflict with $\pi(k)$ remains is $p_{\pi(l)} = x + start_{\pi(k)} + d_{\pi(k)\pi(l)} - \epsilon$ (ϵ is an arbitrarily small number). With this we also obtain a maximum for the waiting time: $w_{\pi(k)} = 2d_{\pi(k)\pi(l)} - \epsilon$. This is $w_{\pi(k)} = 2\delta_\pi(k) + 2\delta_\pi(l) - 2S + 2d - \epsilon$ (Equation 4).

Even if $\delta_\pi(l) = S$ the waiting time $w_{\pi(k)}$ never exceeds its execution time $e_{\pi(k)} = 2\delta_\pi(k) + 2d$. This contradicts with the assumption $w_{\pi(k)} = e_{\pi(k)}$, unless job $\pi(k)$ is not conflicting with any other job.

Let job $\pi(j)$ precede job $\pi(k)$ in sequence π. Both jobs are processed by opposite robots. There is no conflict with job $\pi(k)$ if the following minimum separation times are respected: $d_{\pi(l)\pi(j)} \geq d_{\pi(l)\pi(k)} + d_{i\pi(k)\pi(j)}$. With Equations (3) and (4), we obtain $0 \geq 2\delta_{\pi(k)} - 2S$. This shows that even if $\delta_{\pi(k)}$ is equal to S, there are no further conflicts (see Example in Fig. 4). □

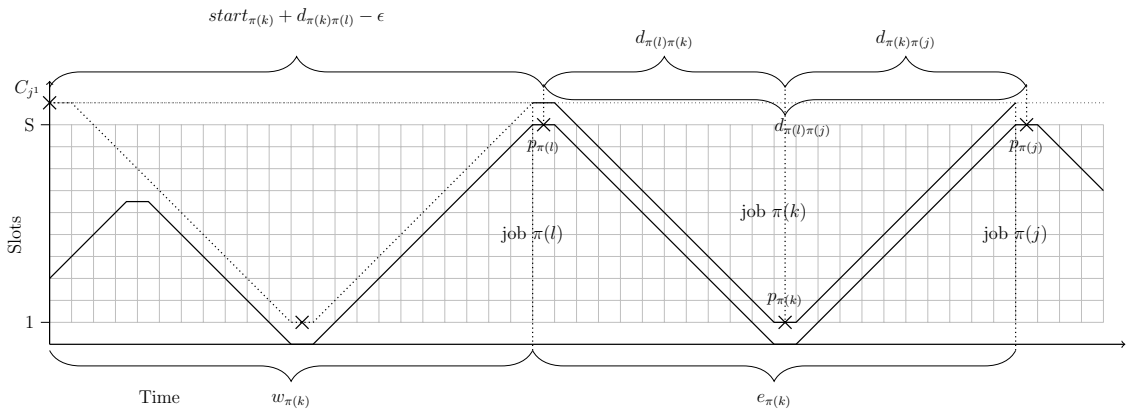


Figure 4: Waiting times in a semi-active schedule

3 Solution algorithms

We present three heuristic and one exact approach for the TRSP. First, we determine the bottleneck robot x and the non-bottleneck robot y using the lower bound presented in Property 1. If $\Omega_1 \geq \Omega_2$ holds, Robot 1 is the bottleneck robot, otherwise this is Robot 2.

In the Descending Sort Procedure (DSP) we fix the schedule of the bottleneck robot x so that all jobs J^x are predecessors of jobs J^y in sequence π. There are no waiting times for robot x, as all jobs in J^x are prioritized over all jobs in J^y ($w_i = 0 \; \forall i \in J^x$). In sequence π, we sort the jobs of robot x and y in descending order according to the distance from the depot δ_i. We obtain a semi-active schedule using Algorithm 1 and given sequence π.

Property 5 (Performance ratio: Scenario A, B)**.**
The performance ratio of DSP is $\frac{3}{2}$.

Proof: For Scenario A and B, $q_i^a = q_i^b = d \geq 0$ holds and either all jobs are storage or all jobs are retrieval operations, hence the execution time of jobs is given ($e_i = 2\delta_i + 2d$). A schedule obtained with DSP is semi-active and thus Property 4 applies ($w_i < e_i \; \forall i \in J$). The completion time of bottleneck robot x is equal to the lower bound $C^x = \Omega$. Let job $\chi \in J^y$ be the last conflicting job $w_\chi > 0$. We consider two cases: **First**, $C_\chi \leq \Omega$. Due to Property 4, $\sum_{J^y} w_i < \frac{1}{2}\Omega$ holds and the makespan is equal to $\Omega + \sum w_i$, hence the performance ratio is below $\frac{3}{2}$.

Second, $C_\chi \geq \Omega$. Let γ be the number of time units the completion of χ exceeds lower bound Ω, $\gamma = C_\chi - \Omega$. The performance ratio holds once we find job $\varphi \in J^y$ that starts before Ω and $w_\varphi + \gamma \leq e_\varphi$ holds.

The last conflicting job χ in sequence π is in conflict with job $j \in J^x$, $\delta_\chi + \delta_j > S$. Hence, $p_\chi = d_{j\chi} + p_j$ and $C_\chi = d_{j\chi} + p_j + end_\chi$. We get $\gamma = d_{j\chi} + p_j + end_\chi - \Omega$ with $p_j = \Omega - end_j$ and obtain: $\gamma = 2\delta_\chi - S + d$.

Let job $\varphi \in J^y$ be the first conflicting job in sequence π. Job φ is in conflict with job $i \in J^x$, $\delta_\varphi + \delta_i > S$. Due to the sorting $\delta_\varphi \geq \delta_\chi$ and $\delta_i \geq \delta_j$ applies. Waiting time w_φ is: $w_\varphi = p_\varphi - start_\varphi$ with $p_\varphi - p_i = d_{i\varphi}$ and $p_i = start_i$, we obtain $w_\varphi = 2\delta_i - S + d$. The execution time $e_\varphi = 2\delta_\varphi + 2d$.

We insert the values in equation $w_\varphi + \gamma \leq e_\varphi$, we obtain $2\delta_i - 2S + 2\delta_\chi \leq 2\delta_\varphi$. Even if $\delta_i = S$ the equation holds, since $\delta_\varphi \geq \delta_\chi$ applies due to the sorting in DSP. $\square$

The Earliest Fit Procedure (EFP) is similar to DSP, however we use Algorithm 2 to create an active schedule using the same sequence π. The schedule of bottleneck robot x remains unchanged, however the jobs of non-bottleneck robot y start at the earliest non-conflicting point in time under consideration of processing sequence π. However, EFP is an improvement procedure to DSP, the performance ratio of EFP remains $\frac{3}{2}$.

Definitions: In a schedule obtained with EFP, let L be a set of jobs that start after or exactly at time Ω, in other words $C_i - e_i \geq \Omega$ holds. Let W be a set of jobs with a waiting time w_i larger 0. Let γ be the number of time units the completion of the last conflicting job $\chi \in J^y$ exceeds lower bound Ω, $\gamma = \max\{C_\chi - \Omega, 0\}$. Then the following applies: $\sum_{i \in W} w_i = \gamma + \sum_{j \in L} e_j$.

For an instance of Scenario A and B, the schedule obtained with EFP has some properties. The execution time e_j of a job $j \in L$ is always larger than any wait time w_i of a job $i \in W$. We show this with Lemma 1. Under consideration of equation $\sum_{i \in W} w_i = \gamma + \sum_{j \in L} e_j$, the number of waiting slots $|W|$ is larger or equal to $|L|$. In Algorithm 4 we use this circumstance by moving each job of set L to a waiting slot in W.

Lemma 1 (Waiting time, active schedule: Scenario A, B)**.** *Given an active schedule obtained with EFP, then $w_i < e_j$ applies $\forall i \in W$, $\forall j \in L$.*

Proof: With regard to Property 4, $w_i < e_i$ holds and due to the sorting in ESA $e_i \leq e_j$ holds $\forall i \in W$, $\forall j \in L$. Waiting slot w_i offers enough space for a job with exactly the same size or smaller. Hence, $w_i \geq e_j$ is not feasible, as in an active schedule job j is scheduled at the earliest possible point in time with no job positioned later. □

The Best Fit Algorithm (BFA) uses sequence π obtained with EFP and inserts jobs from set L into waiting slots from set W. We do this by moving job j with the smallest execution time in L from position k to position $k^* - 1$ ($k^* < k$) in sequence π, where k^* is the position of job i with the largest waiting time in W. Hence, $k^* - 1$ is the position before job i and before the job causing the wait time w_i. We do this until no further jobs from L can be moved. It should be noted that the completion time of bottleneck robot x may rise. However, in Property 6 we show that each insertion leads to an improvement of the objective value (for Scenario A and B).

Due to the short runtime, we execute BFA twice so that each robot is the priority robot once. This has high improvement potential, especially if both robots have equal or almost equal workloads.

Algorithm 4: Best Fit Algorithm (BFA) - *heuristic* - $\mathcal{O}(n^2)$

0. Initialization:
Initialize with EFP; Sequence π
L: Set of jobs with $C_i - e_i \geq \Omega$, sorted in ascending order according to e_i
W: Set of jobs with $w_j > 0$, sorted in descending order according to w_i
1. Assignment:
for each $i \in L$ **do**
  Pull next job j from W
  Insert job i with execution time e_i into the waiting slot w_j
  k is the position of job i and k^* the position of job j in sequence π
  Move job i from position k to $k^* - 1$; Update π accordingly
end for
return Sequence π
Use Algorithm 1 to get a semi-active schedule with sequence π

Property 6 (Performance ratio: Scenario A, B)**.**
The performance ratio of BFA is $4 - 2\sqrt{2} + \frac{\gamma}{\Omega}$.

Proof: Given an active schedule obtained with EFP. For Scenario A and B, Lemma 1 holds, and therefore $w_i < e_j$ applies $\forall i \in W$, $\forall j \in L$. Due to equation $\sum_{i \in W} w_i = \gamma + \sum_{j \in L} e_j$, there is at least one waiting slot in W for each job in set L, $|W| \geq |L|$. Let $e = \min\{e_j\}\ j \in L$ and $w = \max\{w_j\}\ j \in W$ $(e > w > 0)$, then there are at least $\frac{e}{w}$ waiting slots in set W for each job in set L. Due to the sorting in EFP the following applies: $\delta_i \geq \delta_j$ and $w < e_i \geq e_j \geq e$ holds $\forall i \in W$, $\forall j \in L$. Each job in W has an execution time of at least e time units. Hence, each job in L is represented by at least $\frac{e}{w}(e + w)$ time units before lower bound $\Omega + \gamma$.
A Job with execution time e is put into a waiting slot with w time units. The completion time C^x of the non-bottleneck robot y will decrease by w, however the completion time C^y of the bottleneck robot x will increase by up to $e - w$ time units. If all jobs in L have a length of e and all waiting slots in W a length of w, then the objective value is up to $(e - w)/(\frac{e}{w}(e + w))$ times above the lower bound Ω (if $\gamma = 0$). We determine the worst case combination of any e and w, by solving the following equation:

$$\max_{e,w} \left\{ 1 + \frac{e - w}{\frac{e}{w}(e + w)} \right\} \tag{11}$$

$$\max_{e,w} \left\{ 1 + \frac{ew - w^2}{ew + e^2} \right\} = 4 - 2\sqrt{2} \quad at \;\; w = e\sqrt{2} - e \tag{12}$$

We obtain a maximum at $w = e\sqrt{2} - e$ with $4 - 2\sqrt{2}$. This is the performance ratio of BFA, even if the values $e_i \in L$ or $w_j \in W$ are not equal (and $\gamma = 0$ applies). Next, we take a closer look at γ. Due to Property 5 the maximum value for γ is $S + d$. As γ is not part of the insertion process, this has an effect of up to $\frac{\gamma}{\Omega}$ on the performance ratio of BFA. So we get a ratio of $4 - 2\sqrt{2} + \frac{\gamma}{\Omega}$. For large values of Ω, this is approximately ≈ 1.1716. □

The performance ratio of BFA depends on the ratio of $\frac{w}{e}$ (see Fig. 5). For values of $\frac{w}{e}$ near 0 and 1 we obtain a good approximation, however for $\frac{w}{e} = 4 - 2\sqrt{2}$ the deviation to the lower bound is the highest.

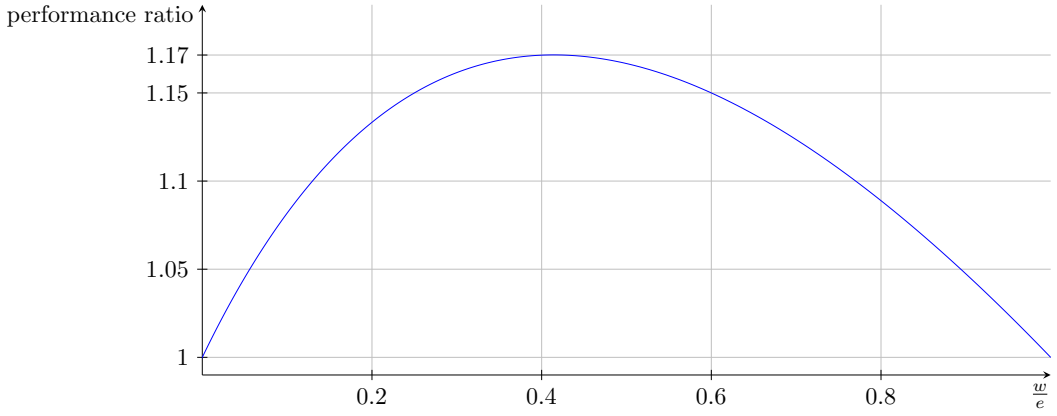


Figure 5: Worst-case performance ratio for $\frac{w}{e}$

We present a branch-and-bound algorithm (BB) that provides a semi-active schedule with an optimal solution. We use a recursive branch-and-bound implementation using deep search first. Further, we use an additive lower bound (Property 1) and

reduce the solution space by using Property 2 and 3. We memorize each solution set $[J(\pi'), C^1(\pi'), C^2(\pi')]$ in Λ if $|C^{1\prime} - C^{2\prime}| \leq S$ holds. Whenever we solve a new sub-problem, we check if there is a dominant partial solution in Λ if this is true we skip this branch. To prevent the algorithm from memory overload we reduce Λ by removing solution sets with high values for $|J'|$ first.

Algorithm 5: Branch-and-bound algorithm (BB) - *exact*

0. Initialization:
π': Empty sequence
Ω: Lower bound (Property 1)
$k = -1$: Current stage in sequence π'
$UB = \infty$: Upper bound
$\Lambda = \emptyset$: Set of solution sets
1. Branching:
for each j in J **do**
  if j is not in π' **then**
    Check precedence relations with Property 2
    Go to 2. Bounding (j)
  end if
end for
if $k = -1$ **then** UB is optimal **return** UB
else $k = k - 1$; BOUND
2. Bounding (Job j):
$k = k + 1$; $\pi'(k) = j$
Use Algorithm 1 to get a semi-active schedule considering jobs $\pi'(0)$ to $\pi'(k)$
Create solution set $[J(\pi'), C^1(\pi'), C^2(\pi')]$
Calculate additive lower bound $\Omega(\pi')$ (using Property 1)
if $\Omega(\pi') \geq UB$ **or** Λ contains solution set **then**
  $k = k - 1$; BOUND
end if
Add solution set to Λ (Property 3)
if π' contains all jobs J **then**
  save new best solution $UB = \Omega(\pi')$
end if
if $UB = \Omega$ **then**
  UB is optimal **return** UB
end if
Go to 1. Branching

Some adjustments in the implementation of BB and BFA were made for instances of Scenario D. We predefine the job combinations (i, j) obtained with Algorithm 3. So that storage job i and retrieval job j of each combination (i, j) is executed

directly after each other and no waiting times between both jobs occur ($w_j = 0$). Predefined combinations reduce the solution space so that BB obtains considerably better results in a shorter period of time. However, the downside is that an instance can only be proven to optimality if BB finds a solution in which $UB = \Omega$ holds. Further, in BFA, we also maintain the predefined combinations so that those do not tear apart during the insertion process.

4 Computational study

We investigate the performance of BFA and BB in a computational study. All algorithms are implemented in Java 1.8 and performed with an Intel i7-6700k 4GHz CPU and 32GiB of RAM. We use the following metrics to determine the performance of the algorithms.

- OPT: Share of instances proven to optimality
- GAP: Average relative gap to lower bound Ω (Property 1)
- MAX: Maximum relative gap to lower bound Ω (Property 1)
- CPU: Average solving time in seconds/milliseconds

4.1 Benchmark instances #1

The first set of instances #1 is provided by Erdoğan et al. (2014) and consists of 50 instances, which are considered hard to solve. In order to generate difficult instances, the authors have linked a random instance generator to a solver to sort out instances that take less than 1 CPU second to solve. Instances are generated with $12 - 27$ jobs and $9 - 39$ slots with $d = 0$ (Scenario A). We compare BB and our mathematical program from Section 2.2 with the results taken from Erdoğan et al. (2014). The authors use two integer programming formulations (TRSP1 and TRSP2) and a branch-and-bound implementation to solve the instances. Further, all procedures have a maximum solution time of 7200 seconds. If a procedure exceeds this time limit, it stops and optimality could not be proven. Hence, for all unsolved instances the solution time is 7200 seconds. We consider the average solution time in

seconds (CPU) and the share of instances proven to optimality (OPT). The results including all procedures are shown in Table 1.

Table 1: Benchmark Instances #1 - 50 instances

	OPT	CPU
TRSP1*	0.04	7089 s
TRSP2*	0.68	2767 s
BB*	0.80	1897 s
MP from Section 2.2	0.60	2997 s
BB no Property 3	0.72	2138 s
BB with Property 3	1.00	44 s

*by Erdoğan et al. (2014) (IRIDIS 4 computing cluster)

BB provides an optimal solution for all 50 instances, including six previously unsolved instances. Without using the domination rule (Property 3), BB only solves 36 instances within the 2 hour time limit.

4.2 Benchmark instances #2

Next we consider large instances for Scenario A with $n = \{10, 20, ..., 10240\}$ jobs and uniformly distributed slot distances $\delta = [1, S]$. For all instances $S = 80$ and $d = 0$ (Scenario A) holds. As in previous studies on the subject we use instances where $\bar{p} \geq 1.99$ applies (Equation 13). Thus, only instances are taken into account with a balanced workload. We use a random instances generator and skip instances where $\bar{p} < 1.99$ holds. For each set of jobs we use 100 valid instances with $\bar{p} \geq 1.99$. Each procedure has a maximum solution time of 30 seconds.

$$1.99 \leq \bar{p} = \frac{\Omega_1 + \Omega_2}{\Omega} \tag{13}$$

We illustrate the OPT values of #2 in Fig. 6. BB provides good results for instances with up to 80 jobs, for larger instances the 30 seconds limit is no longer sufficient. However, for 320 jobs and more, BFA solves around 90% of the instances, by providing solutions equal to lower bound Ω. We conclude that most instances

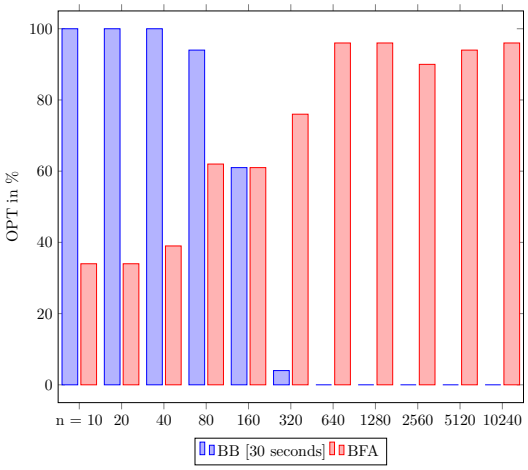


Figure 6: 1100 #1 benchmark instances

with uniformly distributed distances $\delta = [1, S]$ and a limited number of slots are easy to solve within 30 seconds, as the optimal solution C^*_{max} is mostly equal to lower bound Ω.

4.3 Benchmark instances #3

We suppose that instances where both robots share equal workloads ($\Omega_1 = \Omega_2$) are particularly hard to solve if no conflict-free schedule is possible. One method to achieve this is to sort out invalid instances where $\Omega_1 \neq \Omega_2$ holds. However, this can be rather time-consuming for complex instances and has a direct influence on the given distributions.

In #3 we use a different method. First, all instances are created in such a way that the expected workload of both robots is equal. Obviously, this does not mean that the actual workloads are equal. Indeed we receive an average $\bar{p}$ value of 1.97. To achieve $\bar{p} = 2$ ($\Omega_1 = \Omega_2$), we add a dummy job with sufficient length $e_0 = |\Omega_2 - \Omega_1|$ to the non-bottleneck robot (see Equation 14). This dummy job is then executed at the beginning of the process in depot slot 0 or $S + 1$, respectively, so that the non-bottleneck robot starts at time e_0.

$$2 = \bar{p} = \frac{\Omega_1 + \Omega_2 + e_0}{\Omega} \tag{14}$$

In #3 we vary the number of jobs and distribution of the pick-up and delivery times and the distribution of pick-up and delivery slots. We combine the following parameter in a full factorial design with 10 instances per case. Al in all, we receive $1 * 4 * 7 * 22 * \mathbf{10} = 6160$ problem instances.

- Number of slots: $S = 40$
- Number of jobs: $n = 20, 40, 80, 160$
- 22 Patterns to define the distribution of pick-up slot a_i and delivery slots b_i (see Table 2)
- 7 Patterns to define the distribution of pick-up times a_i^t and delivery times b_i^t (see Table 3)

Study #2 shows that instances with uniformly distributed slot distances δ from slot 1 to S are mostly easy to solve. In the following, we investigate other distributions of δ, and the influence of those on the performance of our algorithms. We generate 22 patterns (see Table 2) that define the distribution of the pick-up and delivery slots. For this purpose we divide the storage area into 5 successive blocks with an equal number of slots per block (Here 8 slots per block). To illustrate a pattern, we use the following logic: $\square$ means that this block is not approached by any robot, $\boxslash$ only by the robot from the left side, $\boxbslash$ only from the right side and $\boxtimes$ means that this block is processed by both robots. All storage or retrieval slots are subject to a discrete uniformly distribution within the assigned block(s). Overall, there are $5^4 = 1024$ possible block combinations. We limit the number of valid patterns to 22. We do this by excluding symmetrical patterns. Furthermore, the blocks processed by a robot are successive and both robots process the same number of blocks. In addition, only those patterns are taken into account where the working area of both robots completely overlap. With this we obtain the following supposedly hard to solve patterns:

Table 2: 22 Patterns: Distribution of pick-up a_i and delivery slots b_i

1	⊛ ⊛ ⊛ ⊛ ⊛	9	⧅ ⊛ ⧄ □ □	17	⧅ □ □ ⧄ □
2	⊛ ⊛ ⊛ ⊛ □	10	⧅ ⧅ ⧄ ⧄ □	18	⧅ □ □ □ ⧄
3	⧅ ⊛ ⊛ ⊛ ⧄	11	⧅ ⧅ □ ⧄ ⧄	19	□ ⊛ □ □ □
4	⊛ ⊛ ⊛ □ □	12	□ ⊛ ⊛ □ □	20	□ ⧅ ⧄ □ □
5	⧅ ⊛ ⊛ ⧄ □	13	□ ⧅ ⊛ ⧄ □	21	□ ⧅ □ ⧄ □
6	⧅ ⧅ ⊛ ⧄ ⧄	14	⊛ □ □ □ □	22	□ □ ⊛ □ □
7	□ ⊛ ⊛ ⊛ □	15	⧅ ⧄ □ □ □		
8	⊛ ⊛ □ □ □	16	⧅ □ ⧄ □ □		

We generate 7 patterns (see Table 3) to define the distribution of the pick-up and delivery times, which also defines the respective Scenario. In the first case, we neglect the pick-up and delivery times (Scenario A [1]). The rest of the data is created according to practical settings at seaport terminals with twin automatic rail-mounted gantry cranes (see Fig. 1 left). A common storage area at seaport terminals has a length of approximately 40 slots. Each slot has space for a 1 TEU container (Twenty-foot Equivalent Unit), so each slot has a dimension of 6x2.5x13m and the size of the storage area is 240x25x13m. We assume both cranes (or robots) share a constant speed of approximately 2 meters per second. Hence, in this scenario one time unit is equivalent to 3 seconds. An average pick-up and delivery process has a duration of 36 seconds or 12 time units, this is Scenario B [1]. Due to re-stacking processes and other delays, we assume that the pick-up and delivery time is equally distributed between 12 and 36 time units (Scenario C [1]). For the next pattern we assume that the pick-up process is longer than the delivery time, due to re-stacking processes at the respective pick-up slot (Scenario C [2]). We generate the same patterns for Scenario D [1-3] (except case Scenario A [1]), with 50% retrieval jobs and 50% delivery jobs.

Table 3: 7 Patterns: Distribution of pick-up a_i^t and delivery b_i^t times

	pick-up times (q_i^a)	delivery times (q_i^b)	share of retrieval jobs	share of storage jobs
Scenario A [1]	0	0	1.0	0.0
Scenario B [1]	12	12	1.0	0.0
Scenario C [1]	$[12, 36]$	$[12, 36]$	1.0	0.0
Scenario C [2]	$[24, 36]$	$[12, 24]$	1.0	0.0
Scenario D [1]	12	12	0.5	0.5
Scenario D [2]	$[12, 36]$	$[12, 36]$	0.5	0.5
Scenario D [3]	$[24, 36]$	$[12, 24]$	0.5	0.5

We create benchmark instances #3 using the test data generator from Briskorn et al. (2017). The instances are available online with the project name '*trsp*' using the following link: www.instances.de/dfg/project.php?project=trsp.

Finally, the BB procedure is subject to a 30-second CPU limit per instance. If BB exceeds this limit an instance is considered unsolved. Furthermore, we focus on the performance of BFA and BB, for results of EFP and DFP we refer to Table 7 in the appendix.

In Table 4 we compare the performance of different pick-up and delivery times. Most of Scenarios A [1] and B [1] instances can be easily solved and the average gap to the lower bound is rather low for both procedures with $2,2$ % up to 3.8 %. The instances of Scenario C [1,2] appear to be more difficult to solve. However, BFA produces better results in terms of the GAP values. Instances of Scenario C [2] where $q_i^a \geq q_i^b$ holds, are the most difficult to solve with only $13,3$ % of the instances proven to optimality. However, instances of Scenario D [1,2,3] perform better, which is basically due to the concept of combined jobs. By combining jobs, we reduce the solution space, thus BB obtains faster results compared to instances of Scenario C with an equal number of jobs.

Furthermore, the performance ratio of BFA $(4 - 2\sqrt{2} + \frac{\gamma}{\Omega})$ holds for all instances of Scenario A and B. All in all, BFA provides a good approximation for all scenarios with a maximum deviation to the lower bound of $20,7\%$, compared to BB with $88,3\%$. BFA appears to be a robust procedure despite the short runtime, as the GAP range $(2,5\%$ to $6,2\%)$ and MAX range $(14,2\%$ to $20,7\%)$ is rather small

compared to results of BB.

Table 4: Pick-up and delivery times - 7 * 880 instances

		BFA	BB
Scenario A [1]	OPT	0.197	0.697
	GAP	0.025	0.022
	MAX	0.142	0.173
	CPU	0 ms	9893 ms
Scenario B [1]	OPT	0.043	0.548
	GAP	0.038	0.038
	MAX	0.149	0.222
	CPU	0 ms	14739 ms
Scenario C [1]	OPT	0.000	0.355
	GAP	0.042	0.119
	MAX	0.160	0.660
	CPU	0 ms	19987 ms
Scenario C [2]	OPT	0.003	0.133
	GAP	0.062	0.273
	MAX	0.207	0.883
	CPU	0 ms	26214 ms
Scenario D [1]	OPT	0.026	0.283
	GAP	0.050	0.080
	MAX	0.194	0.405
	CPU	0 ms	16842 ms
Scenario D [2]	OPT	0.005	0.345
	GAP	0.052	0.095
	MAX	0.177	0.596
	CPU	0 ms	15398 ms
Scenario D [3]	OPT	0.006	0.261
	GAP	0.051	0.110
	MAX	0.195	0.751
	CPU	0 ms	17643 ms

In Table 5 we sort all 22 patterns in descending order according to the average gap to the lower bound of BFA (GAP). For 7 patterns BB delivers better results than BFA and for the other 15 patterns BFA has a lower average gap to the lower bound. We conclude that BB performs better for supposedly simple patterns, where GAP is close to 0. However, for patterns with a high deviation to the lower bound, BFA

performs significantly better. All in all, pattern 16 and 17 provide the instances with the highest GAP values for both procedures.

Table 5: 22 Patterns - GAP - 22 * 280 instances

id	pattern	BFA	BB
16	▧ □ ◪ □ □	0.130	0.348
17	▧ □ □ ◪ □	0.098	0.355
15	▧ ◪ □ □ □	0.098	0.262
9	▧ ⊛ ◪ □ □	0.088	0.186
10	▧ ▧ ◪ ◪ □	0.081	0.304
20	□ ▧ ◪ □ □	0.074	0.164
5	▧ ⊛ ⊛ ◪ □	0.047	0.098
19	□ ⊛ □ □ □	0.044	0.012
4	⊛ ⊛ ⊛ □ □	0.037	0.039
18	▧ □ □ □ ◪	0.036	0.213
11	▧ ▧ □ ◪ ◪	0.033	0.156
8	⊛ ⊛ □ □ □	0.033	0.036
14	⊛ □ □ □ □	0.033	0.033
12	□ ⊛ ⊛ □ □	0.032	0.005
6	▧ ▧ ⊛ ◪ ◪	0.029	0.087
2	⊛ ⊛ ⊛ ⊛ □	0.024	0.025
21	□ ▧ □ ◪ □	0.022	0.005
3	▧ ⊛ ⊛ ⊛ ◪	0.022	0.040
13	□ ▧ ⊛ ◪ □	0.019	0.015
1	⊛ ⊛ ⊛ ⊛ ⊛	0.016	0.021
22	□ □ ⊛ □ □	0.013	0.003
7	□ ⊛ ⊛ ⊛ □	0.012	0.002

In our previous tests we could find that many instances could be proven to optimality, because the optimal solution is equal to lower bound Ω. We call a schedule conflict-free if no waiting times occur $w_i = 0 \; \forall i \in J$. As for all instances in #3 $\bar{p} = 2$ holds, an instance is conflict-free if the objective value is equal to lower bound Ω. In our final test we use a fictive upper bound $UB = \Omega + 1$ for BB to determine the share of conflict-free schedules. We call this LB-TEST, which provides a positive answer once a conflict-free schedule with a solution equal to Ω is found within a given time limit (30 seconds).

In Table 6 we illustrate the share of conflict-free schedules according to the respec-

tive patterns and scenarios. Some patterns are never conflict-free (11,17,18 and 21), however others perform particularly well. We conclude that the proposed scenarios have an impact on the LB-TEST, most instances of Scenario A and B can be performed conflict-free, whereas instances of Scenario C and D have a higher conflict potential.

Table 6: LB-TEST (share of conflict-free schedules) - (40+40+80+120) * 22 instances

id	pattern	Scenario A	Scenario B	Scenario C	Scenario D
7	□ ⊛ ⊛ ⊛ □	1.000	1.000	0.838	0.783
1	⊛ ⊛ ⊛ ⊛ ⊛	1.000	1.000	0.562	0.750
4	⊛ ⊛ ⊛ □ □	1.000	1.000	0.375	0.900
19	□ ⊛ □ □ □	1.000	1.000	0.325	0.933
3	⧅ ⊛ ⊛ ⊛ ◪	1.000	1.000	0.362	0.455
22	□ □ ⊛ □ □	1.000	0.975	0.662	0.433
12	□ ⊛ ⊛ □ □	1.000	0.975	0.662	0.950
14	⊛ □ □ □ □	1.000	0.975	0.300	0.850
8	⊛ ⊛ □ □ □	1.000	0.975	0.275	0.783
2	⊛ ⊛ ⊛ ⊛ □	1.000	0.900	0.512	0.822
9	⧅ ⊛ ◪ □ □	1.000	0.200	0.050	0.050
13	□ ⧅ ⊛ ◪ □	0.975	0.475	0.325	0.100
5	⧅ ⊛ ⊛ ◪ □	0.900	0.250	0.150	0.533
6	⧅ ⧅ ⊛ ◪ ◪	0.900	0.225	0.015	0.050
20	□ ⧅ ◪ □ □	0.675	0.000	0.012	0.250
15	⧅ ◪ □ □ □	0.625	0.475	0.012	0.000
10	⧅ ⧅ ◪ ◪ □	0.075	0.000	0.012	0.217
16	⧅ □ ◪ □ □	0.000	0.000	0.000	0.017
21	□ ⧅ □ ◪ □	0.000	0.000	0.000	0.000
11	⧅ ⧅ □ ◪ ◪	0.000	0.000	0.000	0.000
17	⧅ □ □ ◪ □	0.000	0.000	0.000	0.000
18	⧅ □ □ □ ◪	0.000	0.000	0.000	0.000

5 Conclusion

We extend the original problem to varying pick-up and delivery times. In the course of this we also distinguish between storage and retrieval jobs. With the presented

extensions, the TRSP gains relevance for other practical problems where the pick-up and delivery times play a major role, e.g. for rail-mounted gantry cranes at container terminals. We provide an exact branch-and-bound procedure (BB) which dominates other exact solution approaches for Scenario A due to Property 3. Additionally, we show that we easily find optimal solutions for instances with a makespan equal to the lower bound. Furthermore, we present a $\mathcal{O}(n^2)$ approximation algorithm (BFA) with a performance ratio to the lower bound of $4-2\sqrt{2}+\frac{\gamma}{\Omega}$ for Scenario A and B. However, in our numerical study we show that BFA also provides good results for Scenario C and D. In a practical environment, e.g. to increase productivity at container terminals, it is beneficial to provide a conflict-free schedule. With the presented LB-TEST we can determine whether a pattern is mostly conflict-free or not. This data can be adapted to the slot assignment phase to make later operations more efficient. Future research should extend the presented problem to other relevant real-world settings. For instance, other systems like cross-over or triple cranes and service oriented objective functions like tardiness and/or earliness. However, We assume that the presented algorithms can be adapted to similar problems, especially those with a storage and retrieval characteristic, and thus serve as a meta-strategy.

Appendix

Table 7: Number of Jobs - 4 * 1540 instances

n		DSP	EFP	BFA	BB
20	OPT	0.027	0.034	0.047	0.478
	GAP	0.337	0.187	0.065	0.035
	MAX	0.956	0.777	0.207	0.674
	CPU	0 ms	0 ms	0 ms	8071 ms
40	OPT	0.016	0.020	0.044	0.418
	GAP	0.348	0.169	0.053	0.058
	MAX	0.979	0.783	0.199	0.543
	CPU	0 ms	0 ms	0 ms	17729 ms
80	OPT	0.006	0.014	0.035	0.324
	GAP	0.355	0.155	0.047	0.108
	MAX	0.989	0.646	0.181	0.725
	CPU	1 ms	2 ms	0 ms	20536 ms
160	OPT	0.003	0.006	0.033	0.279
	GAP	0.354	0.145	0.044	0.152
	MAX	0.994	0.571	0.172	0.883
	CPU	3 ms	6 ms	1 ms	22645 ms

References

Boysen, N., Briskorn, D., and Emde, S. (2015). A decomposition heuristic for the twin robots scheduling problem. *Naval Research Logistics (NRL)*, 62(1):16–22.

Boysen, N., Briskorn, D., and Meisel, F. (2017). A generalized classification scheme for crane scheduling with interference. *European Journal of Operational Research*, 258(1):343–357.

Bozer, Y. A. and White, J. A. (1984). Travel-time models for automated storage/retrieval systems. *IIE transactions*, 16(4):329–338.

Briskorn, D., Emde, S., and Boysen, N. (2016). Cooperative twin-crane scheduling. *Discrete Applied Mathematics*, 211:40–57.

Briskorn, D., Jaehn, F., and Wiehl, A. (2017). A generator for test instances of scheduling problems concerning cranes in transshipment terminals. *Submitted*.

Bürgy, R. and Gröflin, H. (2016). The blocking job shop with rail-bound transportation. *Journal of Combinatorial Optimization*, 31(1):152–181.

Carlo, H. J. and Martínez-Acevedo, F. L. (2015). Priority rules for twin automated stacking cranes that collaborate. *Computers & Industrial Engineering*, 89:23–33.

Ehleiter, A. and Jaehn, F. (2016). Housekeeping: Foresightful container repositioning. *International Journal of Production Economics*, 179:203–211.

Erdoğan, G., Battarra, M., and Laporte, G. (2014). Scheduling twin robots on a line. *Naval Research Logistics (NRL)*, 61(2):119–130.

Gademann, A. N. et al. (1999). Optimal routing in an automated storage/retrieval system with dedicated storage. *IIE transactions*, 31(5):407–415.

Hu, Z.-H., Sheu, J.-B., and Luo, J. X. (2016). Sequencing twin automated stacking cranes in a block at automated container terminal. *Transportation Research Part C: Emerging Technologies*, 69:208–227.

Jaehn, F. and Kress, D. (2017). Scheduling cooperative gantry cranes with seaside and landside jobs. *Discrete Applied Mathematics*.

Kim, K. H. and Kim, K. Y. (1999). An optimal routing algorithm for a transfer crane in port container terminals. *Transportation Science*, 33(1):17–33.

Kim, K. Y. and Kim, K. H. (1997). A routing algorithm for a single transfer crane to load export containers onto a containership. *Computers & Industrial Engineering*, 33(3-4):673–676.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 2(1-2):83–97.

Kung, Y., Kobayashi, Y., Higashi, T., and Ota, J. (2012). Motion planning of two stacker cranes in a large-scale automated storage/retrieval system. *Journal of Mechanical Systems for Transportation and Logistics*, 5(1):71–85.

Kung, Y., Kobayashi, Y., Higashi, T., Sugi, M., and Ota, J. (2014). Order scheduling of multiple stacker cranes on common rails in an automated storage/retrieval system. *International Journal of Production Research*, 52(4):1171–1187.

Lieberman, R. and Turksen, I. (1981). Crane scheduling problems. *AIIE transactions*, 13(4):304–311.

Roodbergen, K. J. and Vis, I. F. (2009). A survey of literature on automated storage and retrieval systems. *European Journal of Operational Research*, 194(2):343–362.

Speer, U. and Fischer, K. (2016). Scheduling of different automated yard crane systems at container terminals. *Transportation Science*, 51(1):305–324.

IV Research Paper R3

A decomposition procedure for different automated yard crane systems

Andreas Wiehl[1]

[1] University of Augsburg

Abstract

This article addresses the problem of scheduling multiple cranes that execute storage and retrieval jobs along a line with non-crossing constraints. Each crane has a given height, where small cranes pass underneath larger ones under certain conditions. The objective is to minimize the makespan. A practical application for this scheduling problem occurs at transshipment yards, where one or multiple rail-mounted-gantries (RMGs) operate on rectangular yard blocks with handover points at the short sides of the block. We propose a heuristic decomposition procedure to solve this problem, which deals with three interrelated sub-problems. The assignment of jobs to cranes, the formation of crane cycles and the scheduling of cycles under consideration of the non-crossing constraints. In a computational study we investigate the performance of this approach with regard to seven different RMG systems, some of which are novel in practice and literature.

Keywords: Container terminals, yard crane systems, non-crossing constraints, decomposition procedure, crane scheduling

1 Introduction

As a result of the rapidly growing flow of goods in global economy, the requirements for container yards have changed significantly. At the Port of Hamburg container handling rises from two million in 1990 up to nine million TEU (Twenty-foot Equivalent Unit) in 2017 (Port of Hamburg (2017)). We observe similar trends for most large transshipment terminals around the world. During the same period the capacity of the largest deep-sea vessel rises from around 4,000 TEU in 1990 to 21,413 TEU in 2017. The increasing container flows have a strong impact on operational challenges at transshipment terminals. Typical objectives are to guarantee service times or to reduce average handling times of containers. Since the turn of the century, automation technology is increasingly being used at container terminals to meet these requirements.

A typical terminal can be divided into three operative sections. Quay crane operations at the berth, horizontal container transport between berth and storage and yard crane operations at the storage block. Due to the scalability of quay cranes and horizontal transportation, Speer and Fischer (2016) consider container yards to be a potential bottleneck in terminal operations. One reason for this is that adding additional yard blocks or cranes to improve yard productivity is highly cost and space intensive. Thus, the selection of suitable yard systems and the optimized scheduling of cranes is a crucial factor to ensure a reliable throughput.

In recent years, various yard crane layouts with one or multiple cranes have been established in practice. This paper treats a widespread layout at transshipment terminals, where automatic rail-mounted gantries (RMGs) operate on rectangular blocks arranged orthogonally to the quay with handover areas at both ends of the block. Such layout ensures a separation between sea- and landside operations and between automated and non-automated technology. One transfer point is facing the quay (at seaside) and the other exchanges in- and outbound containers with trucks and/or trains (at landside). Each crane moves on a rail along the gantry side and the hoist moves towards the trolley and spreader dimension, to execute container moves. Fig. 1 depicts five different automated yard crane systems: (1) Single RMG, (2) Twin RMG, (3) Double RMG, (4) Triple RMG, and (5) Double Twin RMG. For systems (1)-(4) there exist practical applications at container yards. However,

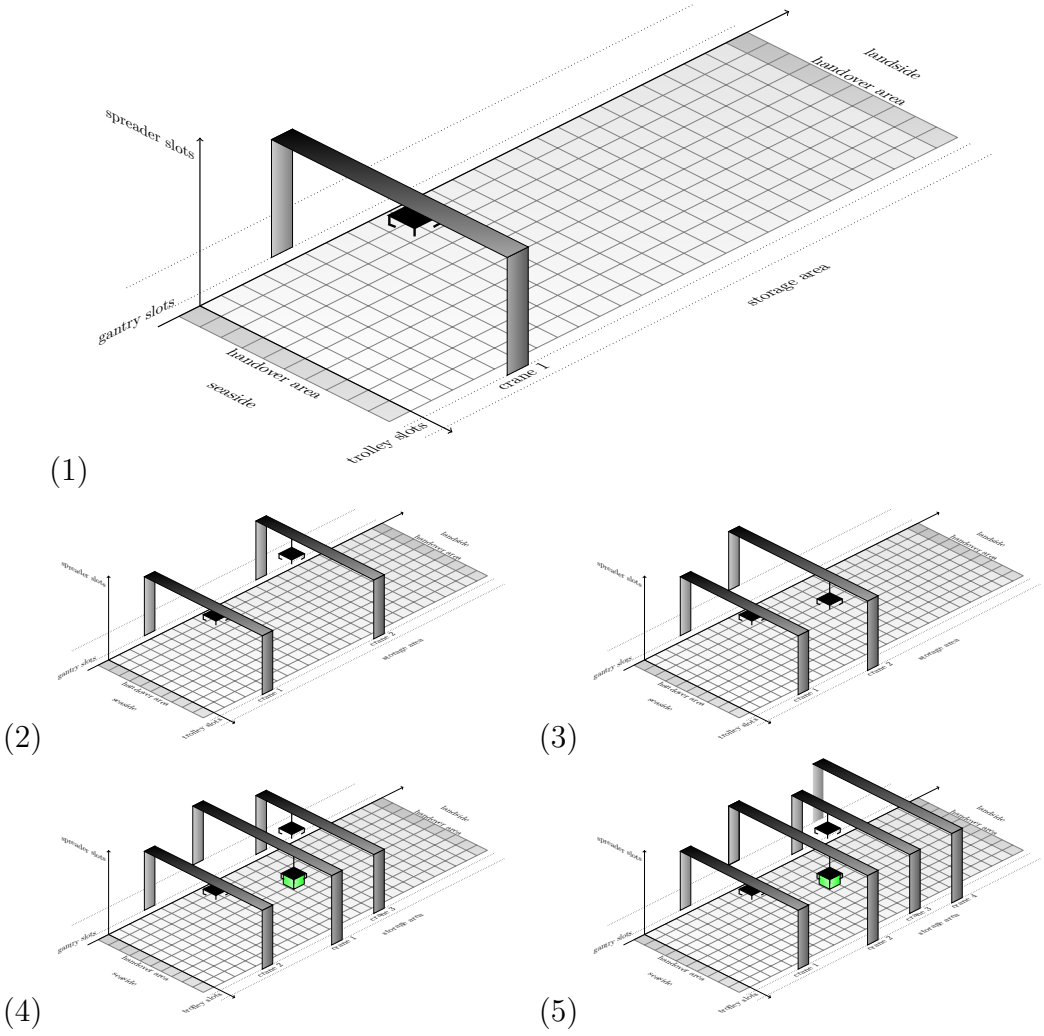


Figure 1: Different automated yard crane systems

system (5) is not used in practice, although it is technically possible.

RMGs have to perform different container moves within the storage block, whereas only some movements are known in advance. Typical objectives are to ensure a high productivity or to respect due dates of trains, ships or trucks for further transport. In this context, the paper on hand treats the following scheduling problem: A set of cranes have to execute storage and retrieval moves of containers at a single yard block with non-crossing constraints. However, small cranes can pass underneath larger ones if the spreader of the large crane is lifted. The following question arises: Which crane should execute which job in which timing sequence so that the overall makespan is minimal?

In the paper on hand we present a heuristic decomposition procedure that divides

this problem into three interrelated sub-problems. First, we introduce a mathematical model that defines the assignment of jobs (container moves) to cranes. Note that the solution of the model provides a lower bound for the overall problem if we relax the non-crossing constraints. After this, another mathematical model can be used to form crane cycles to reduce empty drives during the process. Finally, we present a truncated branch-and-bound and a tabu search heuristic to obtain feasible crane schedules. All three sub-problems pursue the objective to minimize the overall makespan.

The problem under consideration is subject to a few assumptions. We consider the situation at a single block, where the origin and destination positions of containers are predefined by the yard operators. We neglect the movement time of trolley and spreader. This assumption is not restrictive for real-world yard settings, as the movement of spreader and trolley is usually fast enough to complete its positioning during the crane movement along the gantry (Jaehn and Kress, 2017). Furthermore, all containers to be moved are available at the beginning of the process. Cranes have a constant travel speed. We neglected preemptive container moves and inner movements, which is a common assumption in situations of high workload.

The remainder of this article is organized as follows: In Section 2 we give a brief review on literature. A formal description of our problem is given in Section 3. In Section 4 we present the decomposition procedure, including three sub-problems, the Single Crane Routing Problem 4.1, the Crane Assignment Problem 4.2 and the Crane Scheduling Problem 4.3. We investigate the performance of our approach in a comprehensive numerical study in Section 5. Finally, Section 6 concludes the article.

2 Literature

For a fundamental review on logistic processes and operations in container terminals, we refer to Steenken et al. (2004); Stahlbock and Voß (2008). Further, Gharehgozli et al. (2016) provide a notable overview on recent developments in container terminal technologies and OR models.

Crane scheduling problems with interferences are widespread in literature. The survey of Boysen et al. (2017) provides a classification scheme on such systems, which

categorizes the presented problem as [1D,ends,pass|mv^x|C_{max}]. A brief summary of the abbreviations are: its one-dimensional nature (1D), cranes can pass each other under certain conditions (pass), cranes have a constant travel speed along the shared pathway (mv^x), handover slots are on opposite sides of the block (ends), and the objective is to minimize the makespan (C_{max}).

A majority of literature on yard crane scheduling problems focuses on specific applications using a given crane setting. First, we consider systems with only one working unit (e.g. cranes). Sequencing a set of operations on a line with respect to the makespan is similar to the $\mathcal{NP}$-hard Traveling Salesman Problem. However, Gademann et al. (1999) show that if only storage and retrieval jobs are considered this can be solved in polynomial time. Further, Kim and Kim (1997, 1999) present algorithms for the optimal routing of the Single RMG system at seaport terminals. For a survey on single crane/machine scheduling at automated storage and retrieval systems (ASRS) we refer to Boysen and Stephan (2016).

Complexity increases with a second crane, Erdoğan et al. (2014); Boysen et al. (2015) show that the scheduling problem [1D,2,ends|mv^x|C_{max}] including twin cranes/robots is $\mathcal{NP}$-hard in the strong sense. Note that this problem is similar to the Twin RMG case we cover in the paper on hand.

Carlo and Martínez-Acevedo (2015) evaluate priority rules for the Twin RMG system minimizing the makespan. Hu et al. (2016) explore three models for the Twin RMG in a practical study for the Shanghai Yangsha Terminal with respect to the makespan. The efficiency of the Double RMG system is examined in a simulation study by Stahlbock and Voβ (2009). The experiments are based upon practical scenarios at Container Terminal Altenwerder in Germany. Briskorn and Angeloudis (2016) present a polynomial time procedure for the Double and Twin RMG system under predefined processing sequences of containers. Ehleiter and Jaehn (2016) treat the Twin RMG system, where one crane performs a given schedule and the other is dealing with repositioning moves. This situation is commonly called housekeeping in seaport operations, which is usually an issue in situations of low workload during night time. Briskorn et al. (2016); Jaehn and Kress (2017) consider preemptive container moves with cooperative cranes for the Twin RMG system. Similar applications with two working units along a rail occur at ASRS with two stacker cranes (Kung et al. (2012, 2014)) or two industrial robots (Erdoğan et al. (2014);

Thomasson et al. (2017)).

Dorndorf and Schneider (2010) examine the Triple RMG system with an online approach by constructing a new crane schedule whenever a new job is available. The objective is to increase the productivity. Klaws et al. (2011) investigate the performance of the Triple RMG system if the handover area is along the long side of the block.

Only a few articles compare different crane settings. A notable study on the performance of the Single, Twin, Double and Triple RMG systems at seaport terminals is presented by Speer and Fischer (2016). Saanen and Valkengoed (2005) compare three different crane systems (Single, Double and Twin RMG) by means of simulation. The authors compare all systems according to their throughput, flexibility, complexity and costs. Another notable simulation study by Kemme (2012) evaluate the Single, Twin, Double and Trible RMG system with a high degree of detail. The author examines the performance of 385 block layouts with differing block length, width and height. In Emde (2017) quay or yard cranes are divided into groups, where only cranes of the same group interfere. The author finds optimal solutions for large instances within a few seconds and compares several crane configurations with multiple groups in a numerical study.

The paper on hand presents a branch-and-bound algorithm to solve the crane scheduling problem, which is a common approach in literature. In Speer et al. (2011); Guo et al. (2011) the branch-and-bound algorithm iteratively constructs a sequence of jobs for each crane. This approach reaches its limits considering multiple cranes and a high number of jobs. Speer et al. (2011) suggest to consider only a small subset of jobs (no more than 12 jobs) in each run of the branch-and-bound algorithm. This approach outperforms priority rules in most cases. In addition, the authors note that the subset size has a considerable effect on the productivity of crane systems. However, this weakens with an increasing number of jobs.

3 Problem description

For a formal definition consider a single storage block with one or multiple rail mounted gantry cranes $C = \{1, ..., m\}$. Assume that the slots (storage positions) in the block are arranged along a one-dimensional pathway and are consecutively

numbered from 0 to $S+1$ with slot 0 being the transfer slot of containers on the seaside and slot $S+1$ on the landside. Each gantry crane $c \in C$ is associated with a level $l_c \in \{1, ..., L\}$, whereby the level is also a representation of the height and width of a crane. We assume a maximum of two cranes with equal levels, i.e. a maximum of two cranes are mounted to a common rail and share the same height and width.

All cranes $c \in C$ have access to storage slots $\{1, ..., S\}$. However, cranes on equal levels cannot pass each other, therefore, only one has access to transfer slot 0 (seaside crane) and the other to slot $S+1$ (landside crane). Cranes that are alone on a level are denoted as cross-over cranes $C^0 \subseteq C$, with access to both transfer slots (0 and $S+1$). Let depot slot $h_c \in \{0, S+1\}$ be the initial and final location of each crane $c \in C$. Obviously, a crane can only be assigned to a depot slot to which it has access.

We consider a bidirectional flow of inbound and outbound containers from both transfer slots (sea- and landside). Each job $i \in J$ represents a container move from pick-up slot a_i to delivery slot b_i. As we do not consider inner movements, either a_i or b_i correspond to transfer slot 0 or $S+1$, respectively. This implies whether job $i \in J$ is a storage move $a_i \in \{0, S+1\}$ (inbound container) or retrieval move $b_i \in \{0, S+1\}$ (outbound container). Let J^S be the set of storage jobs, whereas J^R is the set of retrieval jobs. As stated before all jobs J are either storage or retrieval jobs ($J = J^S \cup J^R$). We assume that all inbound and outbound containers (i.e. jobs) are available at the beginning of the process. Preemption of container moves (i.e. jobs) is not allowed, meaning that a single container can only be lifted and dropped once before it reaches its delivery slot.

The time horizon under consideration is divided into equally sized intervals $[t-1, t]$ $t \in \mathbb{N}^+$. In the remainder of this paper we refer to a single interval as a time unit. During a time unit each crane $c \in C$ is either working, driving or idle. A driving crane moves to an immediately adjacent slot in a single time unit, this is with or without a loaded container. A working crane lifts or lowers its spreader to perform a pick-up or delivery operation at a specific slot. We assume a deterministic duration in time units of the pick-up $q_i^a \in \mathbb{N}^+$ and delivery $q_i^b \in \mathbb{N}^+$ operation of each job $i \in J$. Finally, if a crane neither works nor moves, this is an idle crane.

We assume no fix assignment of jobs to cranes. Obviously, some jobs can only be

executed by certain cranes. For instance, a job that starts at transfer slot $S+1$ can only be executed by a landside or cross-over crane. Let $J^c \subseteq J$ denote the set of jobs assigned to crane $c \in C$. We say a partition $J^1, J^2, ..., J^m$ of jobs is feasible if every job is assigned to exactly one crane, i.e. $J^1 \cup J^2 \cup, ..., \cup J^m = J$ and $J^c \cap J^{c'} = \emptyset$ $\forall c \neq c' \in C$ and all jobs $i \in J^c$ can be executed by crane c.

A feasible schedule of each crane $c \in C$ is defined by a sequence of crane cycles in which jobs J^c are executed. Each crane cycle starts and ends in transfer slot 0 or $S+1$. We distinguish three different variants of crane cycles. First, a single cycle is the individual execution of a container move (single command). Second, the combination of a storage and retrieval move within one cycle is a dual cycle (dual command). An empty drive from one transfer slot to another is called empty cycle. We assume an uninterrupted execution of crane cycles without detours or idle times within the process. Hence, during a cycle the crane is either driving to a pick-up, delivery or transfer slot, or the crane is working at a given slot to lift or drop a container. Fig. 2 displays a feasible crane schedule with all variants of crane cycles. The example depicts the Double RMG system, where the crane on a higher level is highlighted and starts at landside slot $S+1$. In this example the schedule length of the small crane is 50 and of the large crane 49 time units. We specify the generation of crane cycles in more detail in Section 3.1.

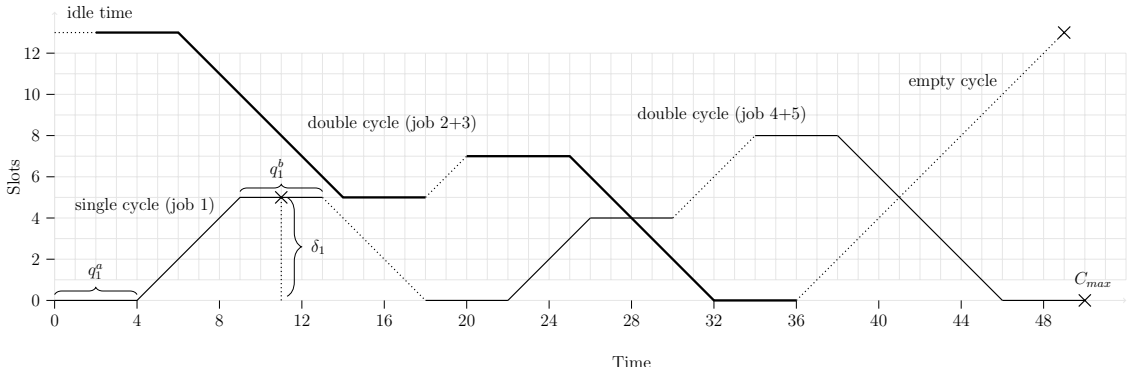


Figure 2: Schematic representation of a crane schedule

After each cycle the corresponding crane either directly starts another cycle or is idle in a transfer slot to avoid conflicts with other cranes. In Fig. 2 the large crane

is idle for two time units in depot slot $S+1$ to prevent a conflict in slot 5 with the smaller crane.

As all cranes C share the same pathway along the storage positions, some non-crossing constraints must hold to avoid collisions. For instance, smaller cranes can only pass a larger crane if the latter one is not currently lifting or dropping a container. Assume two different cranes $c, c' \in C$. The following constraints apply for any time unit $t \in \mathbb{N}^+$ and slot $s \in \{0, ..., S+1\}$ in a feasible crane schedule:

(1) If $l_c = l_{c'}$: Cranes on equal levels cannot be in the same slot s at time t and therefore cannot pass each other.

(2) If $l_c > l_{c'}$: Crane c on a higher level cannot work in slot s at time t if crane c' on a lower level is in the same slot s at time t.

The objective is to find a feasible partition $J^1, J^2, ..., J^m$ of jobs and feasible crane schedules for every crane $c \in C$ to process jobs J^c that the minimizes the makespan C_{max}. This is defined as the maximum schedule length in time units of all cranes $c \in C$ for delivering containers to their target slots and finally returning to their depot h_c under consideration of the non-crossing constraints. The schedule length of crane $c \in C$ is defined by the time unit when the last cycle of crane c ends.

3.1 Generation of crane cycles

In the following we define parameters and variables necessary for generating crane cycles. A dual command is a combination of storage job $i \in J^S$ and retrieval job $j \in J^R$ within a crane cycle. This also defines the initial slot a_i and final slot b_j of this cycle. However, if job $i \in J$ is executed within a single command, there are two possible slots (0 and $S+1$), in which the cycle ends (if i is a storage operation) or starts (if i is a retrieval operation). And further, we also consider empty crane cycles from one transfer slot to another.

To reflect this, we define two dummy jobs D^0 and D^{S+1} with no pick-up and delivery times ($q^a = 0$ and $q^b = 0$), wherein D^0 starts and ends in slot 0, and D^{S+1} in slot $S+1$. A dummy job can either be a storage or retrieval operation. We form another two sets of storage and retrieval jobs, each contains both dummy jobs, $J^{S'} = J^S \cup \{D^0, D^{S+1}\}$ and $J^{R'} = J^R \cup \{D^0, D^{S+1}\}$.

Each crane cycle is reflected by a combination of storage job $i \in J^{S'}$ and retrieval job $j \in J^{R'}$ denoted as (i, j). This also reflects single commands and empty crane cycles. For instance, a combination between dummy job D^0 and D^{S+1} represents an empty cycle from transfer slot 0 to $S + 1$. The distance in time/slot units from pick-up slot a_i to deliver slot b_i is denoted as $\delta_i = |a_i - b_i|$. Let e_{ij} be the execution time of crane cycle (i, j), with storage job $i \in J^{S'}$ and retrieval job $j \in J^{S'}$:

$$e_{ij} = q_i^a + \delta_i + q_i^b + |b_i - a_j| + q_j^a + \delta_j + q_j^b \tag{1}$$

Furthermore, each crane cycle starts and ends in transfer slot 0 or $S + 1$. We distinguish between four different types of crane cycles. Type 1 starts and ends in slot 0, for Type 2 this is slot $S + 1$. If the process starts in 0 and ends in $S + 1$, this is Type 3. Finally, from $S + 1$ to 0 we call this a Type 4 crane cycle. Let t_{ij} indicate a switch of handover slots during a crane cycle; otherwise 0 (Type 1 and 2). Let $t_{ij} = 1$, once $a_i = 0$ and $b_j = S + 1$ (Type 3) and $t_{ij} = -1$, once $a_i = S + 1$ and $b_j = 0$ (Type 4):

$$t_{i,j} = \begin{cases} 0 & \text{Type 1: } a_i = 0,\ b_j = 0 \\ 0 & \text{Type 2: } a_i = S + 1,\ b_j = S + 1 \\ 1 & \text{Type 3: } a_i = 0,\ b_j = S + 1 \\ -1 & \text{Type 4: } a_i = S + 1,\ b_j = 0 \end{cases} \tag{2}$$

Let S_{ij} be the start position and C_{ij} the completion time of cycle (i, j) in time units. Each crane cycle is generated with positions p_i^a and p_i^b in time units of the corresponding jobs. Whereas p_i^a and p_i^b indicate the point in time when half of the pick-up operation a_i or delivery operation b_i is executed. Note that job i might also be a dummy job, then we denote p_i^a and p_i^b as a dummy operation. We calculate the operational positions of each cycle as follows:

$$p_i^a = S_{ij} + \frac{1}{2}q_i^a \tag{3}$$

$$p_i^b = p_i^a + \frac{1}{2}q_i^a + \delta_i + \frac{1}{2}q_i^b \tag{4}$$

$$p_j^a = p_i^b + \frac{1}{2}q_i^b + |b_i - a_j| + \frac{1}{2}q_j^a \tag{5}$$

$$p_j^b = p_j^a + \frac{1}{2}q_j^a + \delta_j + \frac{1}{2}q_j^b \tag{6}$$

$$C_{ij} = p_j^b + \frac{1}{2}q_j^b \tag{7}$$

3.2 Non-crossing constraints

The following non-crossing constraints can be used to determine if there is a conflict between crane cycles of different cranes. Let p_i^x, p_j^x and p_k^x be the position of a storage, retrieval or dummy operation, whereas job i is executed by crane c and j, k is executed by a different crane c'. The following constraints must hold so that a conflict free execution is possible:

Non-crossing constraint (8): Let cranes c and c' be on the same level ($l_c = l_{c'}$). Both cranes cannot pass or touch each other. Hence, they cannot be in the same slot at the same time interval. There is a potential conflict between job $i \in J^c$ and job $j \in J^{c'}$ if $\delta_i + \delta_j > S$ holds. This conflict is resolved if the following equation applies:

$$|p_i^x - p_j^x| \geq \delta_i + \delta_j - S + \frac{1}{2}(q_i^x + q_j^x) \qquad \delta_i + \delta_j > S \tag{8}$$

Non-crossing constraint (9): Let cranes c and c' be on different levels ($l_c \neq l_{c'}$). Both cranes cannot work in the same slot and time interval. There is a potential conflict if both operations take place in equal slots $x_i = x_j$ and both are no dummy operations ($q_i^x > 0$ and $q_j^x > 0$). This conflict can be resolved if the following constraint holds:

$$|p_i^x - p_j^x| \geq \frac{1}{2}(q_i^x + q_j^x) + 1 \qquad x_i = x_j \tag{9}$$

Non-crossing constraint (10): Let crane c be on a higher level than crane c ($l_c > l_{c'}$). Crane c cannot work in the same slot and time interval if crane c' is driving or idle. Assume crane c' executes operation p_j^x and then drives to operation p_k^x. There is a potential conflict if $x_j < x_i < x_k$ or $x_j > x_i > x_k$ applies. This conflict can be resolved if the following constraint holds:

$$|p_j^x + \frac{1}{2}q_j^x + |x_i - x_j| - p_i^x| \geq \frac{1}{2}(q_i^x + q_j^x) + 1 \quad x_j < x_i < x_k \vee x_j > x_i > x_k \quad (10)$$

Assume crane c' is idle from operation p_j^x to p_k^x. There is a potential conflict if both operations take place in equal slots $x_i = x_j$. This conflict can be resolved if one of the following constraints holds:

$$p_j^x - \frac{1}{2}(q_i^x + q_j^x) > p_i^x \vee p_i^x > p_k^x + \frac{1}{2}(q_i^x + q_k^x) \qquad x_i = x_j = x_k \qquad (11)$$

4 Decomposition procedure

To solve instances of real-world size in acceptable time, we propose a heuristic decomposition procedure. We suggest to divide the overall problem into three interrelated sub-problems.

First, we create a feasible partition of jobs J to cranes C, i.e. $J^1, J^2, ..., J^m$ (Crane Assignment Problem (CAP)). Second, we define a set of crane cycles Δ_c for each crane c under consideration of the previously determined set of jobs J^c (Single Crane Routing Problem (SCRP)). And finally, we create a feasible schedule of all crane cycles Δ_c of cranes $c \in C$ under consideration of the non-crossing constraints (Crane Scheduling Problem (CSP)).

1. Crane Assignment Problem (CAP, Section 4.2)
2. Single Crane Routing Problem (SCRP, Section 4.1)
3. Crane Scheduling Problem (CSP, Section 4.3)

We will discuss the SCRP in the upcoming Section 4.1, although this is second in the hierarchy of the decomposition procedure, as this approach is used in the CAP.

4.1 Single Crane Routing Problem

We consider a single crane c and a given set of storage and retrieval jobs J^c. The objective is to determine an optimal sequence to carry out container moves J^c assigned to crane c and finally returning to the respective depot h_c so that the makespan is minimal.

The makespan of a single crane c depends on the generation of crane cycles using storage jobs J^{Sc} and retrieval jobs J^{Rc} assigned to crane c. We formulate this using a complete bipartite graph $G = (J^{S'c}, J^{R'c}, E)$. Edges $(i, j) \in E$ represent all possible crane cycles of crane c. Each crane cycle $(i, j) \in E$ has some properties, execution time e_{ij} in time units (see Equation 1) and type t_{ij} (see Equation 2). Further, let parameter q indicate if a switch of handover slots is necessary during scheduling. Hence, $q = 0$ if all crane cycles start and end at depot slot h_c of crane c, otherwise $q = 1$.

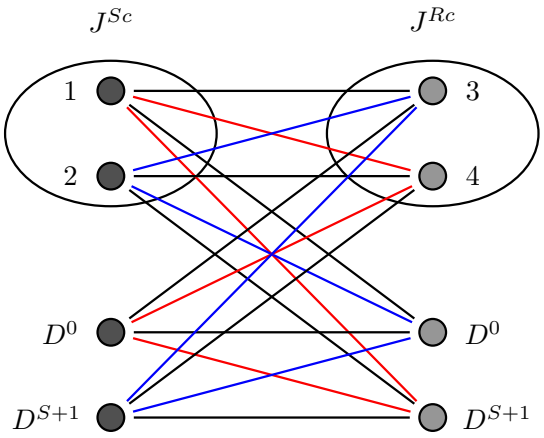


Figure 3: Complete bipartite graph $G = (J^{S'c}, J^{R'c}, E)$

The objective is to find a subset of crane cycles $\Delta^c \subseteq E$ so that $\sum_{(i,j)}^{\Delta^c} e_{ij}$ is minimal, where all vertices in J^{Sc} and J^{Rc} are endpoints of exactly one cycle (edge) in Δ^c. Dummy jobs D^0 and D^{S+1} can be part of more than one cycle in Δ^c. Further, the number of Type 3 ($t_{ij} = 1$) and Type 4 crane cycles ($t_{ij} = -1$) in Δ^c must be equal. And last, if parameter $q = 1$ there is at least one Type 3 and one Type 4 crane cycle in Δ^c. An optimal routing is obtained with a solution Δ^c and the routing rules

presented in this section.

Fig. 3 shows an illustration of graph G with two storage and two retrieval jobs. Thereby, job 1 starts in slot 0, job 2 in $S+1$, job 3 ends in slot 0 and job 4 in slot $S+1$. We have colored **Type 3** edges red and **Type 4** edges blue. An example for a valid solution is $\Delta^c = \{(1,4),(2,3)\}$ or $\Delta^c = \{(1,D^{S+1}),(2,4),(D^{S+1},3)\}$.

The following mathematical program defines this problem for a single crane c. Binary variable x_{ij} is one if crane cycle (i,j) is in Δ^c.

Minimize

$$\sum_{i}^{J^{S'c}} \sum_{j}^{J^{R'c}} x_{ij} \cdot e_{ij} \tag{12}$$

subject to

$$\sum_{i}^{J^{Sc}} x_{ij} = 1 \qquad \forall j \in J^{R'c} \tag{13}$$

$$\sum_{j}^{J^{Rc}} x_{ij} = 1 \qquad \forall i \in J^{S'c} \tag{14}$$

$$\sum_{i}^{J^{S'c}} \sum_{j}^{J^{R'c}} x_{ij} \cdot t_{ij} = 0 \tag{15}$$

$$\sum_{i}^{J^{S'c}} \sum_{j}^{J^{R'c}} x_{ij} \cdot |t_{ij}| \geq q \tag{16}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in J^{S'c},\ \forall j \in J^{R'c} \tag{17}$$

Objective function (12) minimizes the makespan. Constraints (13) ensure that each storage job in J^{Sc} is connected with exactly one retrieval job in $J^{R'c}$, and Constraints (14) provide that each retrieval job in J^{Rc} is combined with exactly one storage job in $J^{S'c}$. Constraints (15) enforce that the number of Type 3 and 4 edges in Δ^c is equal. Constraints (16) ensure that there is at least one Type 3 or 4 edge in Δ^c if $q = 1$. And finally, (17) guarantees that the variable x_{ij} is binary.

Single crane routing rules

Given an optimal set of crane cycles Δ^c for a single crane c derived from a solution of the mathematical model above. We obtain an optimal routing/makespan for crane c in polynomial time if the following rules apply:

- If $h_c = 0$, we start with a Type 1 or 3 crane cycle
- If $h_c = S + 1$, we start with a Type 3 or 4 crane cycle
- Each Type 1 or 4 cycle is followed by a Type 1 or 3 cycle
- Each Type 2 or 3 cycle is followed by a Type 2 or 4 cycle

Any violation of these rules will result in at least one additional empty cycle from one handover slot to another. For each additional empty cycle the makespan increases by $S + 1$ time units.

Obviously, we obtain an optimal routing for crane c if these rules apply during the entire process. Note that there is always at least one schedule where all routing rules apply. This is true as the number of Type 3 and 4 cycles (i.e. changes of the handover sides) is equal. And second, there is at least one set of Type 3 and 4 cycles if a job is on the opposite side of depot slot h_c (i.e. $q = 1$). Finally, we obtain the optimal makespan of a single crane c by summing up the execution times of all crane cycles under consideration: $\sum_{(i,j)}^{\Delta^c} e_{ij}$.

Complexity

The survey of Boysen and Stephan (2016) provides a classification scheme on such problems, which categorizes the problem above as $[\mathrm{E}^{free}|\mathrm{IO}^2|C_{max}]$. A brief summary of the abbreviations are: each loaded move starts or ends at a handover point, which handles both storage and retrieval jobs (E^{free}), two handover slots are located at both ends of the block (IO^2), and the objective is to minimize the makespan (C_{max}). Gharehgozli et al. (2014) proves that this problem can be solved in polynomial time with $\mathrm{O}(n^5)$ steps even if a two dimensional storage area is considered. The authors show that a maximum of n^2 Assignment Problems have to be solved, each of which can be efficiently solved in $\mathrm{O}(n^3)$ where n is the number of jobs. Note that an optimal solution for the Assignment Problem is obtained in $\mathrm{O}(n^3)$ using the Hungarian Method developed by Kuhn (1955) and improved by Munkres (1957).

4.2 Crane Assignment Problem

Given a set of storage and retrieval jobs J and a set of cranes C. The objective is to find a feasible partition $J^1, J^2, ..., J^m$ of jobs so that the maximum workload of each crane $c \in C$ to process jobs J^c is minimal. The workload of each crane $c \in C$ is defined as the minimum time units to execute jobs J^c and return to depot h_c if we relax the non-crossing constraints.
We achieve this by extending the SCRP from Section 4.1 to fit multiple cranes. Note that an optimal solution of the CAP provides a lower bound for the overall problem. First, we define which crane can be assigned to which job. We differentiated between 3 types of cranes according to the access to transfer slot 0 or $S+1$, respectively:

$$c_c = \begin{cases} -1 & \text{1: Seaside crane with access to handover slot 0} \\ 1 & \text{2: Landside crane with access to handover slot S+1} \\ 0 & \text{3: Cross-over crane with access to both handover slots} \end{cases} \qquad \forall c \in C \quad (18)$$

We distinguish three variants v_{ij} of crane cycles (i,j) $\forall i \in J^{S'}$ $\forall j \in J^{R'}$. A crane cycle either starts and ends in the same transfer slot (0 or $S+1$) or a crane cycle starts and ends in different transfer slots:

$$v_{ij} = \begin{cases} -1 & 1 : a_i = 0 \text{ and } b_j = 0 \\ 1 & 2 : a_i = S+1 \text{ and } b_j = S+1 \\ 0 & 3 : a_i \neq b_j \end{cases} \qquad \forall i \in J^{S'} \; \forall j \in J^{R'} \qquad (19)$$

Obviously, a seaside crane can only perform crane cycles bounded to slot 0 and a landside crane only those directed to transfer slot $S+1$. However, crossover cranes $C^0 \subseteq C$ can perform all three job variants. At the start and at end of the process each crane is located at their dedicated depot slot $h_c \in \{0, S+1\}$. Let parameter $d_c \in \{-1, 1\}$ specify this in our model. Let $d_c = -1$, if the start and end slot is 0, otherwise, for $d_c = 1$ the depot slot is $S+1$.
The following mathematical model represents the CAP. Variables x_{ijc} reflect the assignment of crane cycles (i,j) to subsets J^c $c \in C$, taking value 1 if the assignment

is done and 0 otherwise. The auxiliary variable Ω represents the maximum workload of all cranes $c \in C$. The CAP can be formulated as follows:

Minimize

$$\Omega \tag{20}$$

subject to

$$\Omega \geq \sum_{i}^{J^{S'}} \sum_{j}^{J^{R'}} x_{ijc} \cdot e_{ij} \qquad \forall c \in C \tag{21}$$

$$\sum_{i}^{J^{S}} \sum_{c}^{C} x_{ijc} = 1 \qquad \forall j \in J^{R'} \tag{22}$$

$$\sum_{j}^{J^{R}} \sum_{c}^{C} x_{ijc} = 1 \qquad \forall i \in J^{S'} \tag{23}$$

$$\sum_{i}^{J^{S'}} \sum_{j}^{J^{R'}} x_{ijc} \cdot t_{ij} = 0 \qquad \forall c \in C^0 \tag{24}$$

$$\sum_{k}^{J^{S'}} \sum_{l}^{J^{R'}} x_{klc} \cdot |t_{kl}| \geq x_{ijc} \cdot v_{ij} \cdot (-d_c) \qquad \forall i \in J^{S'},\ \forall j \in J^{R'},\ \forall c \in C^0 \tag{25}$$

$$x_{ijc} \cdot v_{ij} \cdot c_c = x_{ijc} \cdot |c_c| \qquad \forall i \in J^{S'},\ \forall j \in J^{R'},\ \forall c \in C \tag{26}$$

$$x_{ijc} \in \{0, 1\} \qquad \forall i \in J^{S'},\ \forall j \in J^{R'},\ \forall c \in C \tag{27}$$

Objective function (21) serves to minimize the maximum workload Ω of all cranes $c \in C$. Constraints (22) enforce that each storage job in J^S forms a crane cycle with a retrieval job in $J^{R'}$ assigned to a crane $c \in C$. Constraints (23) provide that each retrieval job in J^R forms a crane cycle with a storage job in $J^{S'}$ assigned to a crane $c \in C$. Constraints (24) ensure that the number of Type 3 and 4 cycles assigned to cross-over crane $c \in C^0$ is equal. Inequalities (25) enforce that there is at least one Type 3 and 4 cycle assigned to cross-over crane $c \in C^0$ if there is at least one Type 1 ($d_c = 1$) or Type 2 ($d_c = -1$) job variant v_{ij} assigned to crane c (see Equation 19). And finally, (27) ensures that the variable x_{ijc} is binary.

4.3 Crane Scheduling Problem

Given a set of crane cycles Δ^c for each crane $c \in C$. Let n' be the total number of crane cycles. A feasible solution of the proposed scheduling problem requires two components. First, we have to define a processing sequence of crane cycles $(i, j) \in \Delta^c$ for each crane $c \in C$. Each crane cycle is then intended to be processed without idle times if possible. To avoid additional empty cycles we assume that the single crane routing rules from Section 4.1 must apply in a feasible processing sequence. Second, in order to resolve conflicts between crane cycles of different cranes, we define another sequence that regulates which crane cycle is prioritized and which has to wait in the transfer slot. To resolve conflicts between two cycles the non-prioritized crane is idle at the current depot slot for a sufficiently large number of time units until the conflict is resolved. We use the non-crossing constraints defined in Section 3.2 to identify and resolve such conflicts.

We combine both requirements in sequence π, which defines a feasible processing order for all cranes C and the priority order of crane cycles. We prioritize a cycle if it is before another cycle in sequence π. The objective is to find a sequence $\pi = (\pi(1), ..., \pi(n'))$ for all crane cycles $\Delta_c \; \forall c \in C$ so that the makespan is minimal, where $\pi(k)$ is the kth crane cycle in π, $k \in \{1, ..., n'\}$. The makespan being defined as the maximum completion time of all crane cycles in π: $C_{\pi(k)} \; \forall k \in \{1, ..., n'\}$.

In the following subsections we propose two heuristic procedures for this scheduling problem.

4.3.1 Truncated branch-and-bound

In the following we present a recursive truncated branch-and-bound algorithm to generate a feasible schedule from given sets of crane cycles $\Delta_c \; c \in C$. This also computes the position data and completion times of all crane cycles.

We use an additive lower bound LB for effective bounding. Here, the execution times of all cycles not included in schedule π' are added to the current crane completion times. In the branching process each crane cycle is positioned at the earliest possible point in time under consideration of the current processing and priority sequence π' during branching. We do this by moving a cycle forward in time until no more conflicts occur. We identify conflicts using the non-crossing constraints presented in

Section 3.2.

A majority of conflicts can be resolved by shifting the crane cycle forward in time. However, a conflict between an idle crane and a prioritized working crane on a higher level cannot be resolved by this strategy. It can be very time consuming to resolve such deadlock situations in the process of branching. Hence, we prioritize all crane cycles executed by cranes on lower levels over those assigned to cranes on higher levels in sequence π'. As this restriction cuts off parts of the solution space, we call it a truncated branch-and-bound algorithm. However, we assume that this branching rule has a rather small effect on the optimal result in contrast to a significantly reduced solution space.

We initialize algorithm BB with an arbitrary sequence π and a sufficiently large number for the upper bound UB.

Algorithm Truncated branch-and-bound algorithm (BB)

0. Initialization:
π: Initial sequence of crane cycles; π': Empty sequence
UB: Upper bound; LB: Additive lower bound
$(i,j)^c$: Last crane cycle processed by crane $c \in C$; $k = -1$: Current position in π'
1. Branching:
for $j = 1$ to n **do**
  if $\pi(j)$ is not in π' **then**
    Let c be the crane to execute cycle $\pi(k)$
    if branching rule apply for $\pi(j)$ **then**
      $k = k + 1$; $\pi'(k) = j$
      $S_{\pi(k)} = C_{(i,j)^c}$
      go to 3. Bounding ($\pi(j)$)
    end if
  end if
end for
if $k = -1$ **then return** UB
else $k = k - 1$; BOUND
2. Check for conflicts:
for $l = 1$ to $k - 1$ **do**
  if non-crossing constraints 8 - 11 do not apply with $\pi(l)$ and $\pi(k)$ **then**
    Let x be the minimum number of idle time units for $\pi(k)$ to resolve the conflict with $\pi(l)$; Resolve the conflict: $S_{\pi(k)} = S_{\pi(k)} + x$
    Update position data of crane cycle $\pi(k)$ with Equations 3 - 7
    Go to 2. Check for conflicts
  end if
  Next l
end for

Update last crane cycle processed: $(i,j)^c = \pi(j)$; **Go to** 3. Bounding
3. Bounding:
Let Δ'_c be the crane cycles not in sequence π' for each crane $c \in C$
Calculate lower bound LB $= \max_c\{C_{(i,j)^c} + \sum_{(k,l)}^{\Delta'_c} e_{(k,l)}\}\ \forall c \in C$
if LB $\geq$ UB) **then** $k = k - 1$; BOUND **end if**
if π' contains all crane cycles **then** UB = LB (new best solution) **end if**
Go to 1. Branching

4.3.2 Tabu search

We suggest a tabu search heuristic to find good solutions in reasonable time for large instances. Given sets of crane cycles Δ_c $c \in C$.
We initialize the meta-heuristic with an arbitrary sequence π to which the single crane routing rules from Section 4.1 apply.

Algorithm Tabu search algorithm (TS)

0. Initialization:
π: Initial feasible sequence of crane cycles; UB: Upper bound;
Ψ: Empty tabu list; Add π to tabu list Ψ set $noimp = 0$
1. Neighbour solution:
Φ: Empty neighbourhood set
for $i = 1$ to n' **do**
  for $j = 1$ to n' **do**
    if cycles $\pi(i)$ and $\pi(j)$ are on the same level **then**
      Swap crane cycles: $\pi' = \pi$; $\pi'(i) = \pi(j)$; $\pi'(j) = \pi(i)$
      if Single crane routing rules hold for each crane in π' **and** $\pi' \notin \Psi$ **then**
        add π' to Φ
      end if
    end if
  end for
end for
2. Improvement:
UB$' = \infty$
for each $\pi' \in \Phi$ **do**
  UB$''$ = compute makespan of sequence π'
  if UB$''$ < UB$'$ **then** UB$'$ = UB$''$; $\pi = \pi'$
end for
if UB$'$ < UB **then** UB = UB$'$; $noimp = 0$ **else** $noimp++$
if $noimp = 10$ **then STOP**
Go to 1. Neighbour solution

In each iteration we generate a set of neighborhood sequences Φ. The neighborhood consists of all possible interchanges of two arbitrary crane cycles on equal levels

within sequence π. However, we only add those sequences to set Φ where the routing rules from Section 4.1 apply and those that are not included in tabu list Ψ. After computing all sequences in Φ we select sequence π with the lowest makespan for the next iteration. We obtain a feasible sequence π using algorithm BB and determine it after the first feasible sequence is found.
To avoid reconsideration of a solution we set it tabu for the next 1000 iterations using list Ψ. Afterwards we start the next iteration considering sequence π. Once we reach a certain number of iterations with no improvement, the algorithm determines (in this study we set $noimp = 10$). To prevent TS from getting stuck in a local optimum we suggest to restart the algorithm with different randomized initial sequences π.

5 Computational study

In the following we present a computational study based on realistic data. We empirically evaluate the performance of seven different RMG Systems for three typical scenarios at transshipment yards. During the course of this, we also investigate the performance of our proposed decomposition approach.
We implemented the mathematical models using the IBM ILOG CPLEX Optimization Studio (version 12.7.1.0) and the branch-and-bound and tabu search algorithm is implemented in Java 1.8. All tests are performed with an Intel® Core™ i7-6700k CPU running at 4GHz and 32GB of memory, on the operating system Windows 10 64 bit. The components of our decomposition procedure are implemented as follows:

- CAP: Mathematical model ⇒ CPLEX 12.7.1.0
- SCRP: Mathematical model ⇒ CPLEX 12.7.1.0
- CSP-BB: Branch-and-bound algorithm ⇒ Java 1.8
- CSP-TS: Tabu search algorithm ⇒ Java 1.8

The mathematical models CAP and SCRP are subject to a 30-second CPU limit. We use the CPLEX standard MIP gap of 0.0001 %. Further, we initialize CSP-BB with a sufficiently large number for the upper bound and a initial sequence π. The crane cycles in π are sorted in ascending order according to level of the assigned

crane, and in case of equality according to the index. CSP-BB is also subject to a 30-second CPU limit. We use the first feasible sequence π obtained with BB to which the rules from Section 4.1 apply to initialize CSP-TS. Finally, we determine the tabu search heuristic after 10 iterations with no improvement of the current best solution. We restart this procedure ten times with randomized feasible initial sequences π and return the best solution of all iterations, which defines the solution of CSP-TS.

5.1 Instances generation

All instances are created with the test data generator introduced by Briskorn et al. (2017) and are available online under the project name '*ASSIGN*' using the following link: www.instances.de/dfg/project.php?project=ASSIGN

The data is based on typical practical settings at seaport terminals. A common storage area has a length of approximately 40 TEU. Hence, $S = 40$ is a reasonable assumption. Each slot has space for a 1 TEU container (Twenty-foot Equivalent Unit) with a length of approximately 6m. The parameter settings are based on the simulation model of Speer (2017). We assume that all cranes share a constant speed of 2 meters per second. Hence, a time unit is equivalent to 3 seconds. An average pick-up and delivery process has a duration of 36 seconds, i.g. 12 time units. We assume constant pick-up and delivery times $a_i^t, b_i^t = 12 \; \forall i \in J$.

A bidirectional flow of containers is considered. Hence, each container is subject to one of four moving directions. We consider three scenarios of container movement (see Table 1). In Scenario 1, the four moving directions are uniformly distributed. In many practical scenarios the share of transshipment containers is significantly larger than the share of containers at landside. Scenario 2 considers only inbound and outbound containers at seaside. This is a practical situation at night time when there is no hinterland service. In Scenario 3, a majority are inbound container at seaside. This practical setting occurs when a ship needs to be unloaded quickly and only a few prioritized containers are passed to the landside. The pick-up and delivery slots are randomly drawn from a uniform distribution on the interval $[1, S]$. In this study we vary the number of containers $n = 12, 24, 36, 48, 60$, which is a practical assumption as there are usually no more than 50 container moves known

in advance that are accessible. We create 100 instances per case. All in all, we receive $3 * 3 * \mathbf{100} = 900$ problem instances with the following properties:

- Number of slots: $S = 40$
- Number of jobs: $n = 12, 24, 36, 48, 60$
- Pick-up and delivery times: $a_i^t, b_i^t = 12$
- Storage positions a_i and b_i inside the block are randomly drawn from a uniform distribution on the interval $[1, S]$
- Share of container moving directions (see Table 1)

Table 1: Moving directions of containers

	Seaside		**Landside**	
Scenario	Inbound	Outbound	Inbound	Outbound
1	0.25	0.25	0.25	0.25
2	0.50	0.50	0.00	0.00
3	0.75	0.00	0.00	0.25

We investigate the performance of seven different RMG systems with up to four cranes (see Table 2). To model a crane system we specify the crane type c_c and the depot slot d_c for each crane $c \in C$ in each system. It should be noted that systems $[3, 3]$, $[4, 2]$ and $[4, 3]$ are not yet in use in practice but represent realistic extensions of the existing systems.

Table 2: RMG systems

System	Name	$\|C\|$	$\|L\|$	Crane type(s) c_c	Depot slot(s) d_c
$[1,1]$	Single RMG	1	1	0	-1
$[2,1]$	Twin RMG	2	1	$-1,1$	$-1,1$
$[2,2]$	Double RMG	2	2	$0,0$	$1,-1$
$[3,2]$	Triple RMG	3	2	$-1,1,0$	$-1,1,-1$
$[3,3]$	-	3	3	$0,0,0$	$-1,1,-1$
$[4,2]$	-	4	2	$-1,1,-1,1$	$-1,1,-1,1$
$[4,3]$	-	4	3	$-1,1,0,0$	$-1,1,1,-1$

5.2 Computational results

Consider an instance I. Let $\omega(I) = \sum_i^J (\delta_i + q_i^a + q_i^b)$ be the total loaded workload of instance I if we do not consider empty drives and idle times. Let $E(sol_I) = \sum_{(i,j)}^{\Delta} e_{ij}$ be the sum of the execution time of all crane cycles $\Delta = \Delta^1 \cup, ..., \cup \Delta^m$ in a solution sol_I of instance I. Note that $E(sol_I)$ does not include idle times. We define the share of empty drives in all crane cycles Δ in solution sol_I as follows:

$$empty(sol_I) := \frac{E(sol_I) - \omega(I)}{E(sol_I)} \tag{28}$$

Let $\Omega(sol_I)$ be the maximum workload including empty drives and $C_{max}(sol_I)$ the total makespan of a solution sol_I using instance I. In our decomposition approach the value of $\Omega(sol_I)$ is obtained with the mathematical model for CAP and $C_{max}(sol_I)$ with CSP-BB or CSP-TS. We define the relative gap of solution sol_I with respect to lower bound $\Omega(sol_I)$ as follows:

$$gap(sol_I) := \frac{C_{max}(sol_I) - \Omega(sol_I)}{\Omega(sol_I)} \tag{29}$$

Note that $gap(sol_I)$ represents the share of additional time due to idle times (i.e. conflict time) of the bottleneck crane in solution sol_I, which is the last crane that returns to its depot in this solution.

We introduce another metric to measure the performance of different RMG systems. Let $per(sol_I^1, sol_I^2)$ be the minimum average handling time of containers in seconds of two solution, where sol_I^1 is obtained with CSP-BB and sol_I^2 by using CSP-TS. We divide the minimum overall makespan of both solutions by the number of container moves n of instance I and multiply this by 3 seconds, as a single time unit is equivalent to $3s$.

$$per(sol_I^1, sol_I^2) := \frac{\min\{C_{max}(sol_I^1), C_{max}(sol_I^2)\}}{n} \cdot 3s \tag{30}$$

In the following Subsections 5.2.1 - 5.2.5 we evaluate the average share of empty drives (*empty*) in crane cycles, the average relative gap (*gap*) and the average handling time per container move in seconds (*per*) for different sets of instances. Furthermore, we also examine the average solution times in milliseconds ($cpu(ms)$) and the number of optimal solved instances using mathematical model CAP ($sol(\#)$) within the 30 seconds time limit. And finally, the number of instances proven to optimality of the overall problem ($opt(\#)$).
All numerical values that correspond to the presented results are given in the appendix (see Tables 3, 4 and 5).

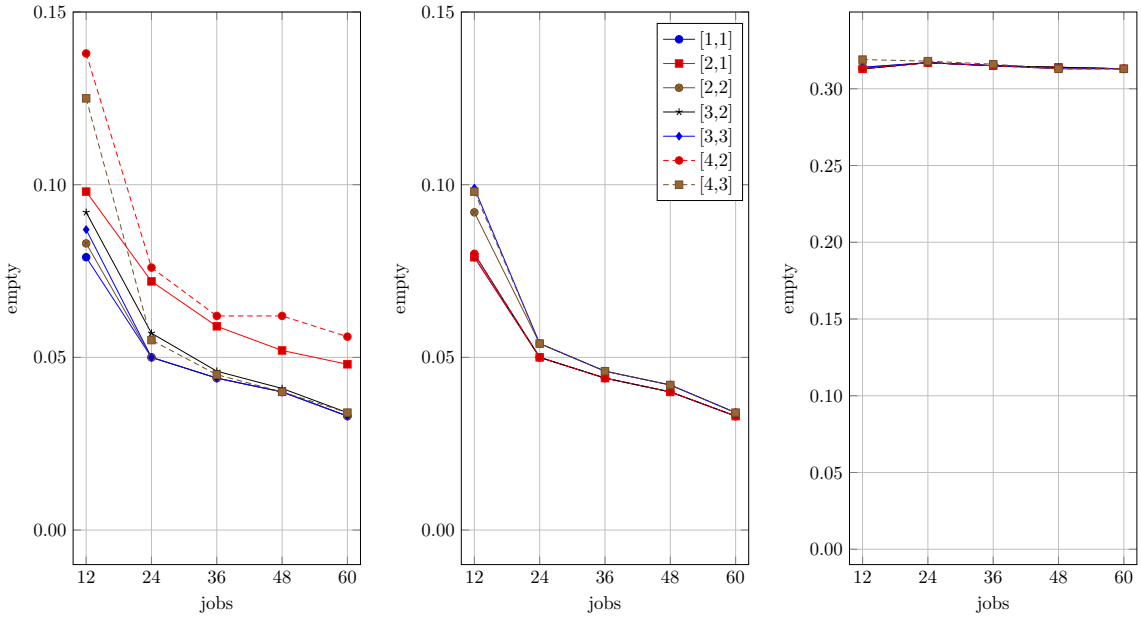
5.2.1 Solution times

The SCRP can be solved in a negligible time for all problem instances, thus we only consider the average solution times of the other procedures. Tables 3, 4 and 5 in the appendix illustrate the average solution times in milliseconds ($cpu(ms)$). Note that CAP and CSP-BB cannot solve all instances within the given 30 seconds time limit. Especially instances with more than 24 jobs are more challenging to solve. With regard to procedures CSP-BB and CSP-TS, it is noticeable that Scenario 3 instances have a higher average solution time than those of the other scenarios. Furthermore, CAP cannot solve all instances within the 30 seconds time limit ($sol(\#)$). The runtime increases with a higher number of jobs, and it is also noticeable that systems including crossover cranes for Scenario 1 and 2 have a higher runtime for CAP.

5.2.2 Share of empty drives

Fig. 4 suggest that including more storage and retrieval jobs leads to a reduction of the average share of empty drives (*empty*) in crane cycles for Scenario 1 and 2. This is basically due to the fact that dual cycles can be formed more and more beneficially using a higher number of jobs. It is also worth mentioning that the systems without cross-over cranes ($[2, 1]$ and $[4, 2]$) have a higher share of empty drives in Scenario 1, as there are fewer possibilities to form dual cycles.

This shows that the presented mathematical model (CAP) provides reasonable results for Scenario 1 and 2. However, for Scenario 3 the share of empty drives is much higher with up to 30 %. The reason for this is that almost all jobs are executed in a single command.



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Figure 4: Average share of empty drives

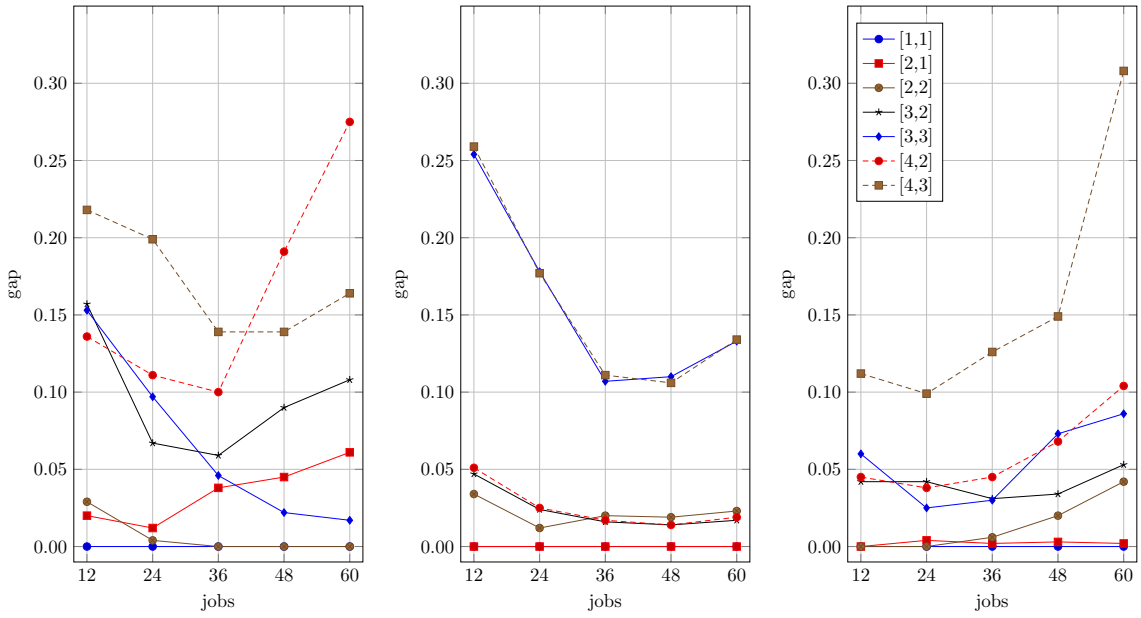
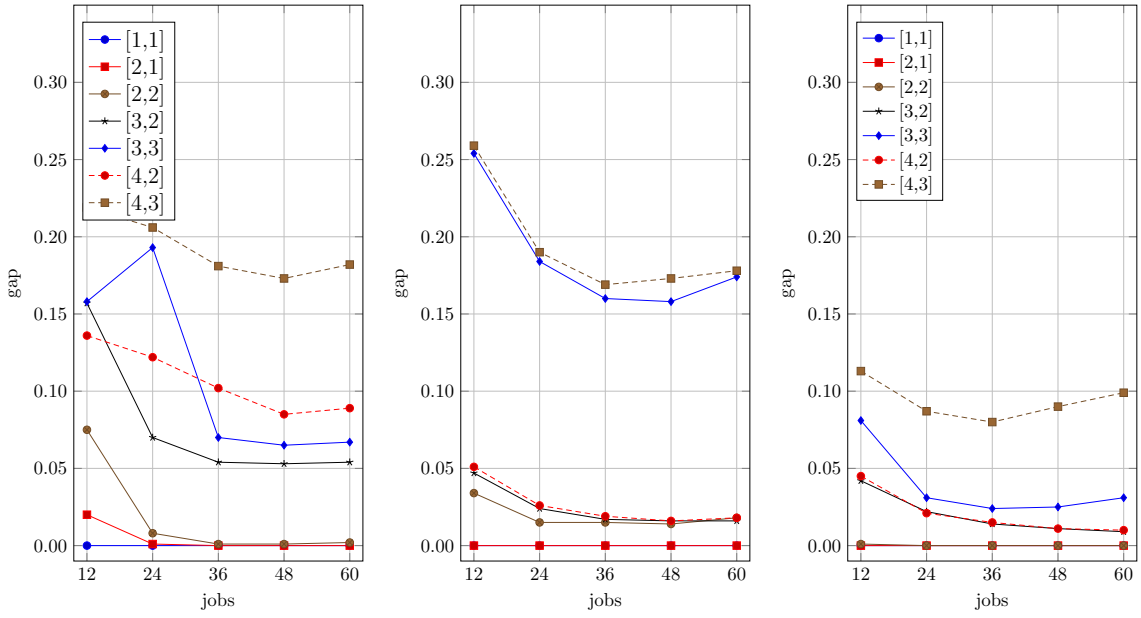


Figure 5: Average relative gap CSP-BB



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Figure 6: Average relative gap CSP-TS

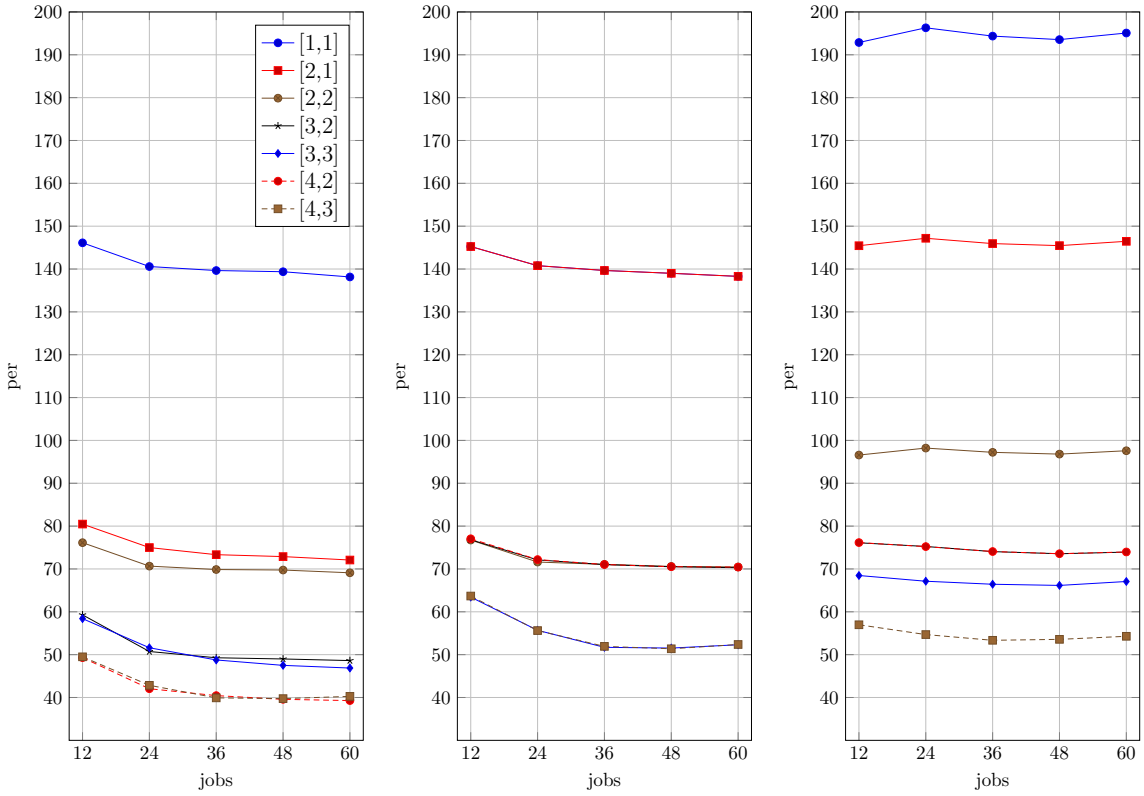
5.2.3 Relative gap

In Figures 5 and 6 we analyze the average relative gap of the CSP-BB and CSP-TS procedure. Obviously, the more crane cycles are available the more possibilities exist to schedule these, and thus idle times can be reduced. We assume that a high number of scheduling options leads to fewer conflicts in general and eventually to lower gap values. This effect can be shown especially for instances with up to 36 jobs, however the gap values for CSP-BB increase for larger instances. This shows that the 30 second time limit for CSP-BB is not suitable for instances with 48 jobs and more. CSP-TS performs better for larger instances. However, there are also limits, especially with regard to systems $[3, 3]$ and $[4, 3]$ and instances with more than 36 jobs. Furthermore, we determine that CSP-BB performs better for pure cross-over systems ($[2, 2]$ and $[3, 3]$) and CSP-TS is better with non-cross-over systems ($[1, 2]$ and $[4, 2]$).

5.2.4 Handling time per container move in seconds

Figure 7 shows the average handling time per container move of all crane systems and scenarios. As defined above, the best solution from CSP-BB and CSP-TS is used for the *per* values. Note that we highlight these values in Tables 3, 4 and 5 in the appendix if CSB-BB leads to a better result than CSP-TS.

It can be observed that the number of available cranes in a system has a strong influence on the performance. This decreases with a higher number of cranes, mainly due to more conflicts in systems with multiple crane. Note that in Scenario 2 all landside cranes remain idle during the entire process. This has a negative impact on the performance, especially for systems $[2, 1]$ and $[4, 2]$. Note that this influence might be reduced by allowing preemptive moves, which we have not considered in this work.



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Figure 7: Average handling time per container move in seconds

It can also be seen that the handling times for Scenario 3 are particularly long. This is due to a higher share of empty drives in this scenario. Furthermore, the average handling time decreases for all crane systems in Scenario 1 and 2 with a higher number of jobs. This effect is particularly strong from 12 to 24 jobs and then declines steadily. In the interval from 12 to 60 jobs, the average handling time is reduced by about 6 up to 11 seconds depending on the system. This effect is especially strong for system $[3, 3]$ and Scenario 1, with an improvement of $11,53$ seconds.

5.2.5 Instances proven to optimality

Despite the heuristic character of the decomposition approach some instances can be proven to optimality. We consider an instance to be optimally solved if the solution

of CAP is optimal ($sol(\#)$) and one of the subsequent procedures (CSP-BB or CSP-TS) finds a solution that is equal to the lower bound obtained with CAP. In other words, if we find a conflict free schedule of the bottleneck crane.
Tables 3, 4 and 5 in the appendix illustrate the number of instances proven to optimality ($opt(\#)$). A majority of instances with up to two cranes can be solved optimally. However, this is not the case if we consider more than two cranes. One reason is the higher conflict potential of such systems. Another reason is that at least two cranes share the same depot slot, which most likely leads to conflicts at the beginning and end of the process.

6 Conclusion

This article treats a decomposition procedure for different automated yard crane systems. The objective is to minimize the makespan. We deal with the integration of three interrelated problems, and provide solution approaches for all sub-problems. As most literature in this area deals with specific crane settings (e.g. Double RMG), we present an approach that addresses various crane systems, which makes them comparable in different practical scenarios. The proposed approach provides good solutions near the lower bound in a short time for problem instances of realistic size. The presented computational study offers new insights to certain metrics regarding different RMG systems (e.g. share of empty drives or average handling time per move). The numerical results suggests that considering a large number of container moves leads to a reduction of empty drives and a higher productivity of crane systems.
Future research should explore the effects of other realistic scenarios on the performance of different crane systems. Some of the assumptions, like the negligible movement time of the trolley/spreader may be relaxed. We propose to extend the model for other practical relevant objectives, like minimizing crane completion times or tardiness of containers. In addition, the approach could also be altered to reflect the influence of new crane technologies or block layouts, like multi-transportation of containers or multiple handover points at each side.

Appendix

Table 3: Computational results of Scenario 1

		CAP			**CSP-BB**		**CSP-TS**			
	n	ms	$sol(\#)$	$empty$	$cpu(ms)$	gap	$cpu(ms)$	gap	$per(s)$	$opt(\#)$
$[1,1]$	12	9	100	0.079	0	0.000	0	0.000	146.12	100
$[1,1]$	24	30	100	0.050	0	0.000	0	0.000	140.58	100
$[1,1]$	36	102	100	0.044	0	0.000	0	0.000	139.65	100
$[1,1]$	48	284	100	0.040	0	0.000	0	0.000	139.37	100
$[1,1]$	60	519	100	0.033	0	0.000	0	0.000	138.15	100
$[2,1]$	12	7	100	0.098	0	0.020	6	0.020	80.48	55
$[2,1]$	24	22	100	0.072	6386	0.012	12	0.001	74.99	96
$[2,1]$	36	70	100	0.059	22372	0.038	42	0.000	73.33	97
$[2,1]$	48	186	100	0.052	24764	0.045	87	0.000	72.89	99
$[2,1]$	60	415	100	0.048	28639	0.061	187	0.000	72.09	100
$[2,2]$	12	58	100	0.083	0	0.029	0	0.075	**76.14**	52
$[2,2]$	24	1303	99	0.050	0	0.004	6	0.008	**70.66**	80
$[2,2]$	36	13227	59	0.044	9	0.000	34	0.001	**69.87**	59
$[2,2]$	48	16713	46	0.040	1885	0.000	154	0.001	**69.75**	45
$[2,2]$	60	14822	54	0.033	1471	0.000	556	0.002	**69.11**	53
$[3,2]$	12	22	100	0.092	0	0.157	6	0.157	59.29	3
$[3,2]$	24	295	100	0.057	348	0.067	97	0.070	**50.73**	0
$[3,2]$	36	2193	97	0.046	29738	0.059	431	0.054	49.27	0
$[3,2]$	48	3786	92	0.041	30000	0.090	1337	0.053	48.99	0
$[3,2]$	60	3254	96	0.034	30000	0.108	3391	0.054	48.62	0
$[3,3]$	12	121	100	0.087	0	0.153	0	0.158	**58.41**	0
$[3,3]$	24	3580	97	0.050	0	0.097	4	0.193	**51.61**	0
$[3,3]$	36	11755	64	0.044	31	0.046	61	0.070	**48.76**	0
$[3,3]$	48	11986	64	0.040	2395	0.022	255	0.065	**47.50**	0
$[3,3]$	60	12573	65	0.033	28609	0.017	714	0.067	**46.88**	0
$[4,2]$	12	13	100	0.138	0	0.136	22	0.136	49.30	4
$[4,2]$	24	42	100	0.076	284	0.111	114	0.122	**42.05**	0
$[4,2]$	36	133	100	0.062	29976	0.100	522	0.102	**40.44**	0
$[4,2]$	48	693	100	0.062	30000	0.191	2065	0.085	39.57	0
$[4,2]$	60	7263	85	0.056	30000	0.275	5112	0.089	39.28	0
$[4,3]$	12	72	100	0.125	0	0.218	10	0.219	**49.50**	0
$[4,3]$	24	2486	99	0.055	20	0.199	67	0.206	**42.84**	0
$[4,3]$	36	9178	87	0.045	7604	0.139	288	0.181	**39.91**	0
$[4,3]$	48	14944	58	0.040	30000	0.139	910	0.173	**39.74**	0
$[4,3]$	60	14874	64	0.034	30000	0.164	2267	0.182	**40.26**	0

Table 4: Computational results of Scenario 2

		CAP			**CSP-BB**		**CSP-TS**			
	n	ms	$sol(\#)$	$empty$	$cpu(ms)$	gap	$cpu(ms)$	gap	$per(s)$	$opt(\#)$
$[1,1]$	12	7	100	0.079	0	0.000	0	0.000	145.25	100
$[1,1]$	24	21	100	0.050	0	0.000	0	0.000	140.78	100
$[1,1]$	36	70	100	0.044	0	0.000	0	0.000	139.66	100
$[1,1]$	48	173	100	0.040	0	0.000	0	0.000	139.00	100
$[1,1]$	60	375	100	0.033	0	0.000	0	0.000	138.29	100
$[2,1]$	12	9	100	0.079	0	0.000	0	0.000	145.25	100
$[2,1]$	24	21	100	0.050	0	0.000	0	0.000	140.78	100
$[2,1]$	36	68	100	0.044	0	0.000	0	0.000	139.66	100
$[2,1]$	48	174	100	0.040	0	0.000	0	0.000	139.00	100
$[2,1]$	60	382	100	0.033	0	0.000	0	0.000	138.29	100
$[2,2]$	12	52	100	0.092	0	0.034	2	0.034	76.75	49
$[2,2]$	24	213	100	0.054	18	0.012	44	0.015	**71.61**	60
$[2,2]$	36	4717	90	0.046	18173	0.020	278	0.015	71.07	51
$[2,2]$	48	8022	75	0.042	21423	0.019	1010	0.014	70.59	36
$[2,2]$	60	8218	76	0.034	25936	0.023	3070	0.018	70.47	22
$[3,2]$	12	31	100	0.080	0	0.047	6	0.047	76.81	0
$[3,2]$	24	308	100	0.050	96	0.024	80	0.024	**72.12**	0
$[3,2]$	36	11254	70	0.044	30000	0.016	392	0.017	**71.02**	0
$[3,2]$	48	15818	48	0.040	30000	0.014	1304	0.016	**70.51**	0
$[3,2]$	60	16936	45	0.033	30000	0.017	3388	0.016	70.29	0
$[3,3]$	12	73	100	0.099	0	0.254	2	0.254	63.44	0
$[3,3]$	24	2257	96	0.054	2	0.178	52	0.184	**55.68**	0
$[3,3]$	36	2378	94	0.046	1893	0.107	281	0.160	**51.71**	0
$[3,3]$	48	3518	91	0.042	30000	0.110	954	0.158	**51.53**	0
$[3,3]$	60	6810	83	0.034	30000	0.133	2511	0.174	52.31	0
$[4,2]$	12	29	100	0.080	0	0.051	6	0.051	77.05	1
$[4,2]$	24	249	100	0.050	89	0.025	84	0.026	**72.19**	0
$[4,2]$	36	9504	75	0.044	30000	0.017	415	0.019	**71.04**	0
$[4,2]$	48	15817	48	0.040	30000	0.014	1354	0.016	**70.53**	0
$[4,2]$	60	16939	45	0.033	30000	0.019	3506	0.018	70.44	0
$[4,3]$	12	87	100	0.098	0	0.259	5	0.259	63.68	1
$[4,3]$	24	2459	95	0.054	2	0.177	32	0.190	**55.62**	0
$[4,3]$	36	3197	91	0.046	1812	0.111	165	0.169	**51.93**	0
$[4,3]$	48	3416	91	0.042	30000	0.106	571	0.173	**51.37**	0
$[4,3]$	60	6120	84	0.034	30000	0.134	1519	0.178	**52.37**	0

Table 5: Computational results of Scenario 3

		CAP			**CSP-BB**		**CSP-TS**			
	n	ms	$sol(\#)$	$empty$	$cpu(ms)$	gap	$cpu(ms)$	gap	$per(s)$	$opt(\#)$
$[1,1]$	12	9	100	0.313	0	0.000	0	0.000	192.87	100
$[1,1]$	24	21	100	0.317	0	0.000	0	0.000	196.30	100
$[1,1]$	36	61	100	0.315	0	0.000	0	0.000	194.36	100
$[1,1]$	48	147	100	0.314	0	0.000	1	0.000	193.54	100
$[1,1]$	60	304	100	0.315	0	0.000	0	0.000	195.08	100
$[2,1]$	12	8	100	0.313	1065	0.000	0	0.000	145.44	100
$[2,1]$	24	15	100	0.317	6601	0.004	0	0.000	147.18	100
$[2,1]$	36	47	100	0.315	4201	0.002	1	0.000	145.94	100
$[2,1]$	48	124	100	0.314	5100	0.003	1	0.000	145.46	100
$[2,1]$	60	270	100	0.315	5401	0.002	2	0.000	146.47	100
$[2,2]$	12	45	100	0.314	1	0.000	2	0.001	**96.59**	99
$[2,2]$	24	14913	51	0.317	832	0.000	45	0.000	98.21	51
$[2,2]$	36	13627	55	0.315	7034	0.006	343	0.000	97.22	55
$[2,2]$	48	14408	53	0.314	24569	0.020	2723	0.000	96.80	50
$[2,2]$	60	17077	45	0.315	30000	0.042	10775	0.000	97.59	41
$[3,2]$	12	12	100	0.313	625	0.042	86	0.042	76.11	1
$[3,2]$	24	47	100	0.317	30000	0.042	1424	0.022	75.25	0
$[3,2]$	36	10209	77	0.315	30000	0.031	8128	0.014	74.05	0
$[3,2]$	48	13852	55	0.314	30000	0.034	29180	0.011	73.55	0
$[3,2]$	60	14223	54	0.315	30000	0.053	78698	0.009	73.93	0
$[3,3]$	12	49	100	0.314	1	0.060	18	0.081	**68.47**	0
$[3,3]$	24	12081	60	0.317	19746	0.025	453	0.031	**67.14**	0
$[3,3]$	36	10705	65	0.315	30000	0.030	3249	0.024	66.43	0
$[3,3]$	48	11869	62	0.314	30000	0.073	12440	0.025	66.16	0
$[3,3]$	60	11714	64	0.315	30000	0.086	34718	0.031	67.07	0
$[4,2]$	12	9	100	0.313	415	0.045	90	0.045	**76.16**	0
$[4,2]$	24	32	100	0.317	30000	0.038	1403	0.021	75.22	0
$[4,2]$	36	353	100	0.315	30000	0.045	7950	0.015	74.07	0
$[4,2]$	48	1698	95	0.314	30000	0.068	28853	0.011	73.56	0
$[4,2]$	60	943	98	0.315	30000	0.104	75848	0.010	73.97	0
$[4,3]$	12	24	100	0.319	24	0.112	66	0.113	**56.97**	0
$[4,3]$	24	296	100	0.318	30000	0.099	1075	0.087	54.69	0
$[4,3]$	36	2042	94	0.316	30000	0.126	6340	0.080	53.35	0
$[4,3]$	48	3178	91	0.314	30000	0.149	22178	0.090	53.57	0
$[4,3]$	60	4276	88	0.316	30000	0.308	59902	0.099	54.28	0

References

Boysen, N., Briskorn, D., and Emde, S. (2015). A decomposition heuristic for the twin robots scheduling problem. *Naval Research Logistics (NRL)*, 62(1):16–22.

Boysen, N., Briskorn, D., and Meisel, F. (2017). A generalized classification scheme for crane scheduling with interference. *European Journal of Operational Research*, 258(1):343–357.

Boysen, N. and Stephan, K. (2016). A survey on single crane scheduling in automated storage/retrieval systems. *European Journal of Operational Research*, 254(3):691–704.

Briskorn, D. and Angeloudis, P. (2016). Scheduling co-operating stacking cranes with predetermined container sequences. *Discrete Applied Mathematics*, 201:70–85.

Briskorn, D., Emde, S., and Boysen, N. (2016). Cooperative twin-crane scheduling. *Discrete Applied Mathematics*, 211:40–57.

Briskorn, D., Jaehn, F., and Wiehl, A. (2017). A generator for test instances of scheduling problems concerning cranes in transshipment terminals. *Submitted*.

Carlo, H. J. and Martínez-Acevedo, F. L. (2015). Priority rules for twin automated stacking cranes that collaborate. *Computers & Industrial Engineering*, 89:23–33.

Dorndorf, U. and Schneider, F. (2010). Scheduling automated triple cross-over stacking cranes in a container yard. *OR Spectrum*, 32(3):617–632.

Ehleiter, A. and Jaehn, F. (2016). Housekeeping: Foresightful container repositioning. *International Journal of Production Economics*, 179:203–211.

Emde, S. (2017). Optimally scheduling interfering and non-interfering cranes. *Naval Research Logistics (NRL)*, 64(6):476–489.

Erdoğan, G., Battarra, M., and Laporte, G. (2014). Scheduling twin robots on a line. *Naval Research Logistics (NRL)*, 61(2):119–130.

Gademann, A. N. et al. (1999). Optimal routing in an automated storage/retrieval system with dedicated storage. *IIE transactions*, 31(5):407–415.

Gharehgozli, A. H., Roy, D., and de Koster, R. (2016). Sea container terminals: New technologies and OR models. *Maritime Economics & Logistics*, 18(2):103–140.

Gharehgozli, A. H., Yu, Y., Zhang, X., and Koster, R. d. (2014). Polynomial time algorithms to minimize total travel time in a two-depot automated storage/retrieval system. *Transportation Science*, 51(1):19–33.

Guo, X., Huang, S. Y., Hsu, W. J., and Low, M. Y. H. (2011). Dynamic yard crane dispatching in container terminals with predicted vehicle arrival information. *Advanced Engineering Informatics*, 25(3):472–484.

Hu, Z.-H., Sheu, J.-B., and Luo, J. X. (2016). Sequencing twin automated stacking cranes in a block at automated container terminal. *Transportation Research Part C: Emerging Technologies*, 69:208–227.

Jaehn, F. and Kress, D. (2017). Scheduling cooperative gantry cranes with seaside and landside jobs. *Discrete Applied Mathematics*.

Kemme, N. (2012). Effects of storage block layout and automated yard crane systems on the performance of seaport container terminals. *OR Spectrum*, 34(3):563–591.

Kim, K. H. and Kim, K. Y. (1999). An optimal routing algorithm for a transfer crane in port container terminals. *Transportation Science*, 33(1):17–33.

Kim, K. Y. and Kim, K. H. (1997). A routing algorithm for a single transfer crane to load export containers onto a containership. *Computers & Industrial Engineering*, 33(3-4):673–676.

Klaws, J., Stahlbock, R., and Voß, S. (2011). Container terminal yard operations–simulation of a side-loaded container block served by triple rail mounted gantry

cranes. In *International Conference on Computational Logistics*, pages 243–255. Springer.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 2(1-2):83–97.

Kung, Y., Kobayashi, Y., Higashi, T., and Ota, J. (2012). Motion planning of two stacker cranes in a large-scale automated storage/retrieval system. *Journal of Mechanical Systems for Transportation and Logistics*, 5(1):71–85.

Kung, Y., Kobayashi, Y., Higashi, T., Sugi, M., and Ota, J. (2014). Order scheduling of multiple stacker cranes on common rails in an automated storage/retrieval system. *International Journal of Production Research*, 52(4):1171–1187.

Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38.

Port of Hamburg (2017). Container handling 1990 to 2016. https://www.hafen-hamburg.de/en/statistics/containerhandling. Last accessed on Apr 13, 2017.

Saanen, Y. A. and Valkengoed, M. V. (2005). Comparison of three automated stacking alternatives by means of simulation. In *Simulation Conference, 2005 Proceedings of the Winter*, pages 10–pp. IEEE.

Speer, U. (2017). *Optimierung von automatischen Lagerkransystemen auf Containerterminals*. Springer.

Speer, U. and Fischer, K. (2016). Scheduling of different automated yard crane systems at container terminals. *Transportation Science*, 51(1):305–324.

Speer, U., John, G., and Fischer, K. (2011). Scheduling yard cranes considering crane interference. In *International Conference on Computational Logistics*, pages 321–340. Springer.

Stahlbock, R. and Voβ, S. (2009). Efficiency considerations for sequencing and scheduling of double-rail-mounted gantry cranes at maritime container terminals. *International Journal of Shipping and Transport Logistics*, 2(1):95–123.

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

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

Thomasson, O., Battarra, M., Erdoğan, G., and Laporte, G. (2017). Scheduling twin robots in a palletising problem. *International Journal of Production Research*, pages 1–25.

V Fazit und Ausblick

In diesem Kapitel werden die zentralen Ergebnisse der vorgestellten Beiträge nochmals gebündelt zusammengefasst und Ansatzpunkte für künftigen Forschungsbedarf aufgezeigt.

1 Fazit

Mit der vorliegenden Dissertationsschrift sollte ein wesentlicher Beitrag zur Optimierung von automatisierten Lagersystemen mit Interferenzen geleistet werden.
Im Zentrum standen dabei insbesondere Containerterminals, die infolge des zunehmenden Welthandels vor vielfältigen Herausforderungen stehen. Wie dargestellt wurde, arbeiten teilweise mehrere RMGs gemeinsam an einem Lagerblock, um Container möglichst effizient zwischenzulagern bzw. für den weiteren Transport bereitzustellen. Dieser Abschnitt des Containerterminals stellt jedoch einen potentiellen Engpass im System dar, weshalb ein Hauptfokus dieser Dissertation darauf lag, zugrunde liegende Optimierungsprobleme genauer zu untersuchen bzw. darzustellen, um darauf aufbauend schließlich entsprechende Optimierungsverfahren zu entwickeln.

- Kapitel II – Beitrag R1 verfolgte in diesem Zusammenhang das Ziel, zunächst ein gemeinsames Testbed für verschiedene Optimierungsprobleme am Lagerblock zu etablieren. Es wurde daher ein Generator vorgestellt, der es ermöglicht, Testinstanzen für eine Vielzahl praktischer Problemstellungen in Bezug auf die Kransteuerung an Containerterminals zu erzeugen. Wie aufgezeigt wurde, deckt das zugrunde liegende generische Modell dabei nicht nur Szenerien bereits existierender Arbeiten ab, sondern vor allem auch solche, die noch nicht von der Forschung erfasst wurden. Dazu gehört bspw. die Unterstützung von Containerbewegungen in drei Dimensionen oder die Modellierung von verschie-

denen praxisrelevanten Vorrangbeziehungen (engl. *precedence constraints*). Zu Beginn des Projekts war allerdings noch nicht ersichtlich, welche Szenarien der Generator letztendlich abbilden würde. Durch die Zusammenarbeit mit einem Praxispartner sowie mit weiteren Forschern aus diesem Fachbereich wurde das generische Modell in einem fortwährenden Prozess daher erweitert bzw. an praktische Situationen angepasst. Ein Beispiel hierfür war die Aufnahme der *stacking value* in das generische Modell. Wie in Beitrag R1 ebenfalls herausgearbeitet wurde, erfolgt die Vergabe der Container-Stellplätze in der Praxis häufig nach Prioritätsregeln, wobei unter anderem die Destination, Beschaffenheit oder Terminierung des Containers eine entscheidende Rolle spielen. Dieser Umstand wurde mit der *stacking value* abgebildet.
Durch das vorgegebene Generierungsschema können zudem unbeabsichtigte Verzerrungen bei der Erstellung von Testdaten verhindert werden. Ein weiterer Vorteil dieses Generators ist, dass erstellte Datensätze öffentlich zugänglich sind und somit auch für zukünftige Forschung genutzt werden können. Bislang existieren bereits zwei wissenschaftliche Publikationen, bei welchen der Generator Anwendung findet (siehe Ehleiter und Jaehn (2016); Jaehn und Kress (2017)). An weiteren Projekten, die ebenfalls auf diesen Generator zurückgreifen, wird derzeit zudem gearbeitet. Auch in Beitrag R2 und R3 wurden die Testdaten mithilfe des Generators erstellt und öffentlich zugänglich gemacht. Eine Übersicht der bisher zur Verfügung gestellten Datensätze kann online abgerufen werden (www.instances.de/dfg/published.php).

- Kapitel III – In Beitrag R2 lag der Schwerpunkt darauf, das bereits bekannte $\mathcal{NP}$-schwere *twin robots scheduling problem* (Erdoğan et al., 2014) weiter zu erforschen und an gleichartige Problemstellungen, insbesondere an Containerterminals, anzupassen. Wie bei dem hierzu beschriebenen Praxisproblem dargestellt wurde, führen zwei Industrieroboter Arbeiten an gegebenen Positionen entlang einer gemeinsamen Schiene aus. Neben der Entwicklung von exakten Verfahren zur Lösung des Optimierungsproblems ging es vor allem darum, die zeitliche Auswirkung der auftretenden Interferenzen näher zu betrachten. Mit dem für die vorliegende Dissertationsschrift entwickelten und in Beitrag R2

präsentierten Approximationsverfahren konnte für zwei der betrachteten Szenarien gezeigt werden, dass durch die Behinderungen der beiden Roboter im Worst-Case bei hinreichend großen Instanzen bis zu 17,16% zusätzliche Zeit (d.h. Wartezeit) entsteht.
Ein weiterer wichtiger Aspekt lag zudem darin festzustellen, bei welchen Probleminstanzen Behinderungen zwischen beiden Roboter auftreten und welche Auswirkungen dies auf die Gesamtbearbeitungszeit hat. Zu Beginn der Forschungsarbeit war dabei bekannt, dass eine ausgeglichene Arbeitslast beider Roboter eine wesentliche Voraussetzung darstellt, um schwer lösbare Instanzen zu erstellen. Eine zentrale Erkenntnis von Beitrag R2 war in diesem Zusammenhang, dass sich selbst große Instanzen mit bis zu 10.240 Aufträgen mittels der vorgestellten Verfahren zumeist exakt lösen lassen, sofern die Lieferpositionen gleich verteilt entlang der gemeinsamen Schiene angenommen werden. Aus diesem Grund wurden die Auswirkungen von vermeintlich konfliktreichen Verteilungsmustern der Lagerpositionen detaillierter untersucht. Mithilfe des vorgestellten LB-Tests konnte schließlich eine Aussage über den Anteil konfliktfreier Ablaufpläne in Bezug auf die vorgestellten Verteilungsmuster und Szenarien getroffen werden.

- Kapitel IV – In Beitrag R3 wurde das Ziel verfolgt, einen allgemeinen Ansatz für verschiedene RMG-Systeme mit mehreren Kränen zu entwickeln, bei welchen bspw. kleine Kräne unter bestimmten Voraussetzungen einen großen Kran passieren können. Hier stellt sich jedoch grundsätzlich die Frage, welche Auswirkung die Bereitstellung eines zweiten oder dritten Krans hat, bspw. in Bezug auf die Produktivität der RMG-Systeme.
  Aus diesem Grund wurde in Beitrag R3 ein Ansatz entwickelt, der verschiedene RMG-Systeme mit bis zu zwei Kränen pro Ebene bzw. Schiene unterstützt. Ziel war es, einerseits in der Praxis bekannte Systeme abzubilden (Twin, Double oder Triple RMG), andererseits aber auch solche, die sich in praktischen Anwendungen noch nicht etabliert haben. Das in Beitrag R3 vorgestellte Optimierungsproblem wurde dabei aufgrund seiner hohen Entscheidungstiefe in drei zusammenhängende Teilprobleme zerlegt. Damit verbunden wurde für jedes der Probleme ein Lösungsverfahren präsentiert. In einer umfangreichen

numerischen Studie konnte in diesem Zusammenhang gezeigt werden, dass der gewählte heuristische Dekompositions-Ansatz für die sieben untersuchten RMG-Systeme für Probleminstanzen mit bis zu 36 Containerbewegungen gute Ergebnisse nahe oder gleich der unteren Schranke liefert. Zudem konnte dargelegt werden, dass durch eine gemeinschaftliche Betrachtung von Ein- und Auslagerungen innerhalb eines Kranzyklus (*dual cycles*) eine Reduzierung der Leerfahrten erzielt werden kann. Dieser Effekt nimmt mit der Anzahl an betrachteten Containerbewegungen zu.
Des Weiteren liefert die in Beitrag R3 vorgestellte Studie grundlegende Erkenntnisse hinsichtlich der Produktivität von verschiedenen RMG-Systemen in drei praxisrelevanten Szenarien. Insgesamt steht mit dem in diesem Beitrag entwickelten Ansatz somit ein für praktische Anwendungen relevantes Verfahren zur Verfügung, das ausführlich untersucht wurde.

Zusammenfassend lässt sich schließlich festhalten, dass mit den drei vorgestellten Arbeiten ein bedeutender Beitrag zur Optimierung von Kransystemen mit Interferenzen geleistet wurde. Dennoch existiert durchaus noch weiterer Forschungsbedarf, weshalb abschließend ein kurzer Ausblick auf Anknüpfungspunkte für künftige Forschungsprojekte gegeben wird.

2 Ausblick

Einige Punkte wurden im Rahmen der vorliegenden Arbeit nicht untersucht und können Gegenstand weiterer Forschungsarbeiten sein. So stellt bspw. für den vorgestellten Dekompositions-Ansatz die Betrachtung von potentiellen Konflikten schon bei der Zuordnung von Containern zu Kränen eine vielversprechende Erweiterung dar, die in Zukunft einer genaueren Analyse bedarf. Mithilfe von weichen Nebenbedingen im Zuordnungsmodell könnte eingegrenzt werden, dass nicht mehrere Kräne am selben Stellplatz arbeiten bzw. sich dort behindern.
Ein weiterer Lösungsansatz der vorgestellten Probleme ist die intelligente Vorverarbeitung von Probleminstanzen. So könnten beispielsweise optimale bzw. konfliktfreie Teillösungen vorgerechnet und in Hashtabellen abspeichern werden. Diese können dann zu einem späteren Zeitpunkt zur effizienten Lösung von großen Probleminstan-

zen verwendet werden.
Die überwiegende Mehrheit der bisherigen Forschung betrachtet Interferenzen zudem lediglich in einem eindimensionalen Modell, obwohl beispielsweise ein RMG Containerbewegungen in einem dreidimensionalen Raum durchführt. Dieser Umstand sollte in Zukunft stärker als bisher geschehen hinterfragt und ggf. bestehende Modelle um eine bzw. zwei Dimensionen erweitert werden.
Zukünftige Forschungsarbeiten könnten sich darüber hinaus auch mit einer möglichen Erweiterung der vorgestellten Ansätze um serviceorientierte Zielfunktionen beschäftigen. Dazu gehören unter anderem die Einhaltung von Fristen oder die Minimierung von Verspätungen an der Land und Seeseite.
Auch die Berücksichtigung von ökologischen Aspekten stellt neben den aus der Forschung bekannten Zielfunktionen eine durchaus sinnvolle Erweiterung dar. Der Energiewandel von fossilen Brennstoffen hin zu erneuerbaren Energien bei gleichzeitig steigendem Welthandel führt auch zu einer zunehmenden Bedeutung der nachhaltigen Containerlogistik. Ergänzend zu ökonomischen Aspekten sollte auch ein energieeffizienter Ablauf bei der Optimierung von Kranabläufen eine tragende Rolle spielen. Beispielsweise kann kinetischen Energie, die durch das Abladen von Containern entsteht, wiederverwendet werden. Durch die kurzzeitige Speicherung der Energie und die Synchronisation von Ab- und Aufladeprozessen könnte so zukünftig ein energiesparender Ablauf am Lagerblock ermöglicht werden.

Literaturverzeichnis

Ehleiter, A. und Jaehn, F. (2016). Housekeeping: Foresightful container repositioning. International Journal of Production Economics, 179:203–211.

Erdoğan, G., Battarra, M., und Laporte, G. (2014). Scheduling twin robots on a line. Naval Research Logistics (NRL), 61(2):119–130.

Jaehn, F. und Kress, D. (2017). Scheduling cooperative gantry cranes with seaside and landside jobs. Discrete Applied Mathematics.