Agent-Based Container Terminal Optimisation

Michael Winikoff
University of Otago
Dunedin, New Zealand
michael.winikoff@otago.ac.nz

Hanno-Felix Wagner
Universität Duisburg-Essen
Essen, Germany
hanno-felix.wagner@stud.uni-due.de

Thomas Young
University of Canterbury
Christchurch, New Zealand
thomas.young@pg.canterbury.ac.nz

Stephen Cranefield
University of Otago
Dunedin, New Zealand
stephen.cranefield@otago.ac.nz

Roger Jarquin
Jade Software Corporation
Christchurch, New Zealand
rjarquin@jadeworld.com

Guannan Li
University of Otago
Dunedin, New Zealand
gli@infoscience.otago.ac.nz

Brent Martin
University of Canterbury
Christchurch, New Zealand
brent.martin@canterbury.ac.nz

Rainer Unland
Universität Duisburg-Essen
Essen, Germany
rainer.unland@icb.uni-due.de

ABSTRACT

Container terminals play a critical role in international shipping and are under pressure to cope with increasing container traffic. The problem of managing container terminals effectively has a number of characteristics which make agents a suitable technology to consider applying. Container terminals involve the operation of distributed entities (e.g. quay cranes, straddle carriers) which coordinate to achieve competing goals in a dynamic environment. This paper describes a joint industry-university project which has explored the applicability of agent technology to the domain of container terminal management. We describe an emulation platform of a container terminal based on the JADE agent framework, along with two optimisations that have been developed and integrated with the emulator: allocating container moves to machines through negotiation, and allocating containers to yard locations through an evolutionary algorithm.

Categories and Subject Descriptors

H.4.2 [**Information Systems Applications**]: Types of Systems—*logistics*; I.2.11 [**Artificial Intelligence**]: Distributed AI—*multi-agent systems*

General Terms

Algorithms

Keywords

Container Terminal Management, Container Terminal Optimisation, Logistics

1. INTRODUCTION

As of 2005, some 18 million total containers make over 200 million trips per year. There are ships that can carry over 14,500 Twenty-foot equivalent units (TEU), for example the Emma Mærsk, 396 m long, launched August 2006. Today, approximately 90% of non-bulk cargo worldwide is transported by container, and modern container ships can carry up to 15,000 twenty-foot equivalent units (TEU). Container terminals play a crucial role in the process of shipping containerised goods, since they are the interface between sea and land transport (rail and trucks). Due to their critical role, and the growth in the amount of container traffic, container terminals are under pressure to increase their operating capacity and efficiency.

A number of characteristics of container terminals make them a natural candidate for agent-based solutions. Firstly, they can be naturally described as a system of interacting entities (e.g. cranes) which are distributed and autonomous, and which interact to solve a problem (e.g. loading and unloading ships) in an efficient way. Secondly, the problem being solved is complex: there are multiple competing goals (e.g. unloading a ship and loading a train) and there are multiple constraints (see Section 2). Thirdly, the environment is dynamic: the situation is subject to change, and things can (and do) go wrong. Taken together, these three characteristics make it natural to investigate agent-based techniques for container terminal management and optimisation.

This paper reports on a joint industry-university project that investigated the application of agents to container terminal optimisation. The industry partner was Jade Software Corporation, whose portfolio of products includes Jade Master Terminal (JMT), a comprehensive container terminal management solution. The project included visits to a local container terminal port in order to obtain a detailed understanding of the problem and its associated complexities. Additionally, real (but anonymised) data from the port was used for evaluation purposes. This data included machine movements and container arrival and departure information.

Note that the aim of this project is not to automate container handling: the domain is sufficiently complex that we cannot hope to capture all relevant situations and constraints. Instead, our ultimate aim is to develop an intelligent decision support tool that can assist human decision makers who are running a container terminal.

The key contributions of this paper are:

- An agent-based container terminal emulation platform, which can be used to assess (static) policies, or to guide decision makers during day-to-day operation (Section 4).
- A negotiation-based algorithm for optimising allocation of container moves to straddle carriers (Section 5).
- A genetic algorithm approach for optimising the allocation

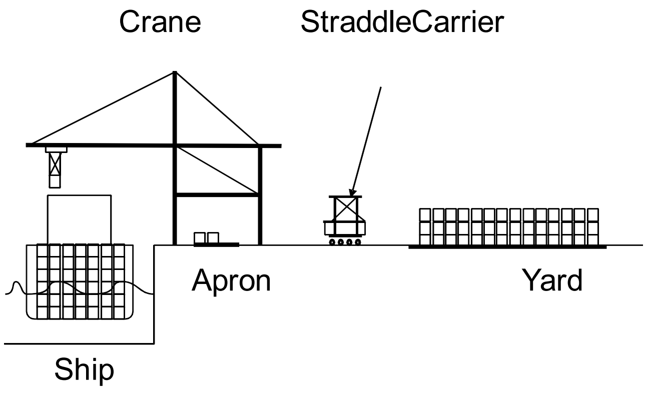


Figure 1: Port Scenario

of containers to yard locations (Section 6).

This paper is structured as follows. In Section 2 we introduce the domain of container terminals, along with associated problems; followed by a discussion of related work (Section 3). Section 4 presents the agent-based simulator which we have developed. We then discuss specific optimisations that address allocating container moves to straddle carriers (Section 5) and allocating container locations in the yard (Section 6). Finally, Section 7 concludes with discussions and future plans.

2. CONTAINER TERMINAL OPERATION

In this section we briefly introduce the domain of container terminal operations, and the associated problems that confront a container terminal manager. Note that the discussion here is necessarily brief and omits various details and complexities. We aim to capture the essence of the problem, and give some sense of the various constraints and factors that make the problem challenging, without describing all such constraints and factors.

A container terminal consists of a number of different areas, depicted in Figure 1. The *apron* is the area directly beside the ship, into which containers are unloaded from the ship, and from which containers are loaded onto the ship. Note that the apron is of limited size. The bulk of the container terminal is taken up with the *yard* where containers are stored. Figure 1 does not show the structure of the yard, and some specific areas (such as empty container storage, or container cleaning). Nor does it show the rail and truck areas, where trucks and trains arrive to collect and/or drop off containers.

Whilst the basic areas (e.g. apron, yard) are somewhat consistent between different container terminals, the machines used vary. The local port that we have worked with has a particular setup that involves two types of machines: Quay Cranes (QCs) and Straddle Carriers (SCs). Quay Cranes ("Crane" in Figure 1) are able to move along the shore and can load containers from the apron to the ship, or unload them from the ship to the apron. Figure 2 shows a ship being unloaded by two (large blue) Quay Cranes. Straddle Carrlers (Figure 3) are mobile cranes, able to move freely within the container terminal. They are able to lift containers and stack them up to a certain height.

Given these areas and machines, the basic process of unloading a ship is as follows. Containers are unloaded from the ship to the apron by the QC. While this is being done, Straddle Carriers are clearing the apron by transporting containers from the apron to the yard, and stacking them. The process for loading a ship is the reverse (SCs bring containers from the yard to the apron, and the containers are loaded on to the ship by the QCs).

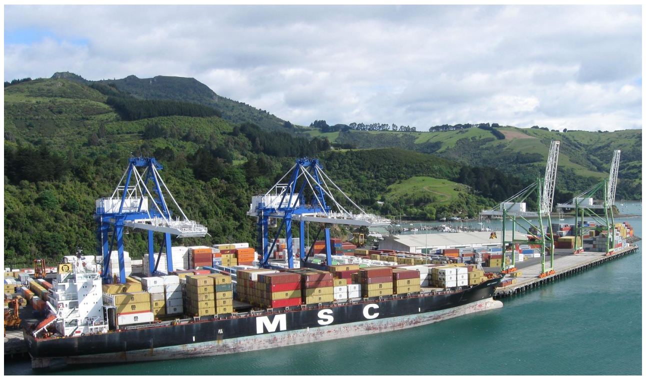
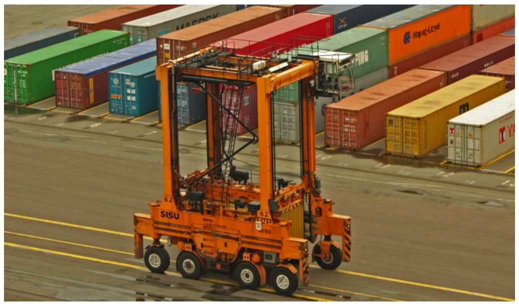


Figure 2: Container Terminal



Source: http://commons.wikimedia.org/wiki/File:Portalhubwagen.jpg

Figure 3: Straddle Carrier

This process sounds fairly simple, but, as mentioned earlier, is made complicated by a range of factors and constraints. For instance[1]):

- There may be more than one QC operating on a given ship, so SCs need to be shared between the QCs. Additionally, two QCs operating on the one ship need to maintain a safe separation distance.

- When retrieving containers from the yard, the container needed may be beneath other containers, which requires these containers to be moved in order to access the desired container. It is worth noting that yard space is limited, and that container terminals sometimes operate at a high level of yard capacity, so that one cannot avoid stacking containers in a sub-optimal way.

- Straddle Carriers need to divide their time between servicing QCs and dealing with trucks and trains that arrive. In the case of trucks, their arrivals are not predictable.

- Some containers are refrigerated ("reefers"), and these cannot be without power for an extended period. Port terminal operation guidelines aim to have reefers be disconnected for no longer than 10 minutes (and for at most 60 minutes).

[1]Some additional specific complicating factors are noted in Section 6

- For safety reasons, humans and machines cannot be in the same area at the same time. One situation where this constrains operations is that reefers need to be connected to power by humans, but they need to be moved by Straddle Carriers. This means that connecting/disconnecting reefers needs to be coordinated with Straddle Carriers accessing the area.

- The order in which containers are loaded into a ship is constrained by the need to maintain a ship's balance. If the ship begins to list, then the shipping order may need to be dynamically changed.

- Issues may arise during operations such as machines breaking down, or finding that certain containers cannot be stacked on top of certain other containers.

- Straddle Carrier operators are human and may make mistakes in data entry. This may lead to situations such as a later driver attempting to pick up container C1 from the yard, only to find that C1 is not where it is meant to be.

This list of factors and constraints is not exhaustive, but it hopefully does convey a sense of the complexity and challenge of this domain.

The key metric for container terminal efficiency is ship turnaround time: any delays to a ship's schedule are bad (and may involve a financial penalty to the port). Some of the decisions that the terminal operators need to make as part of day-to-day operations are:

- Where should an incoming ship dock? This matters because a ship may be closer to one part of the yard than another, which influences the cost of moving containers between the ship and yard.

- How should QCs be allocated to a ship?

- How should SCs be allocated between QCs, yard rearrangement operations, and trucks and trains? The management of straddle carriers has a big impact on the terminal's efficiency. If QCs are not adequately served, then they may need to wait for containers to be moved, which delays ship loading/unloading.

- Where should a given (incoming) container be placed in the yard? The placement of containers in the yard can make a big difference to the cost of moving the container later to where it is needed.

In our work we have focused on the last two questions.

3. RELATED WORK

There are a few papers that propose to apply multi-agent systems in the domain of container terminal management and optimisation.

Thurston and Hu [12] proposes to use a multi-agent system to automate container terminal operations. They focus on the loading process only. Like us, they have agents for each of the machines (quay cranes, straddle carriers). While their work is promising, the paper is early work: it outlines the approach and reports on an early Java prototype. No experimental evaluation is reported, and (from personal contact with the author) it appears that no subsequent work has been done.

Rebollo *et al.* [9] also propose to automate container terminal operations using a multi-agent system. Again, the paper is high-level: it provides a system architecture, but does not provide details for how the individual agent would operate. Implementation appears to have been in progress, but we have not found any subsequent papers describing the resulting implementation, or results from evaluation (the most recent paper [2] is shorter and does not contain any further details).

Note that such work, which attempts to address all of the problems of a container terminal, is quite ambitious, and is in danger of needing to make simplifying assumptions that render it inapplicable to real ports. Our approach is firstly to not attempt to automate a terminal, but to provide decision support; and secondly, to deal with parts of the problem separately, while trying to avoid oversimplifying the problem.

Other work that seeks to apply agents to container terminals has been more modest in scope, seeking either to tackle part of the problem only, or to simulate but not to control. An example of the latter is Henesey *et al.* [4], which describes a simulation tool ("SimPort"). Unlike the earlier described work, they do not aim to automate the operation of a container terminal, but instead to provide a tool that can be used to analyse the performance of (static) policies. This analysis can then be used to select static policies to implement.

Kefi *et al.* [5] is an example of work that tackles a part of the problem. They propose to use a multi-agent system to solve the yard allocation problem. They do not provide details of their approach, but do report results that appear promising. One potential issue with their approach is scalability, since they use an agent to represent each container (their results only consider up to 26 containers).

In addition to agent-based approaches, there have been other (non-agent-based) approaches that aim to tackle various aspects of the management and optimisation of container terminals (see [11, 10] for recent surveys). A common limitation of such work, which is often based on operations research techniques, is that it computes solutions up-front, but does not address the dynamic nature of the problem.

Kim and Park [6] tackle the QC allocation problem within an operational research framework. They propose an inefficient branch & bound approach, and then improve it using a Greedy Randomized Adaptive Search Procedure ("GRASP"). Their GRASP is similar to Tabu search (and to our negotiation process) in that it develops an initial solution, and then iteratively improves it by progressive cost-guided modification. This work does not address what happens when things go wrong, and deals with developing schedule for QCs, not for Straddle Carriers (although we would expect that the techniques could be adapted to be used for developing SC schedules).

The work of Chen *et al.* [3] is in some ways quite close to our work on optimising container moves (Section 5), and, indeed, their formalisation was the starting point for our work. However, they make a number of assumptions that are unreasonable in practice, and which we have relaxed. In particular, they assume that the yard maintains a clean separation between containers for loading and for unloading ("inbound and outbound containers are not mixed up in one block in storage yards"). Additionally, they assume a three-stage process and do not provide for "buffering" where, for example, a Quay Crane can unload a second or third container even though the first container has not yet been taken to the yard.

Considering now the yard allocation problem (which we cover in Section 6), there is a range of work that tackles this problem. Zhang *et al.* [13] consider the problem of allocating inbound and outbound containers to blocks in the yard, within the context of an overall planning hierarchy for a container terminal. A two-stage process first determines the number of containers that are to be assigned to each block to evenly balance the overall workload. This is followed

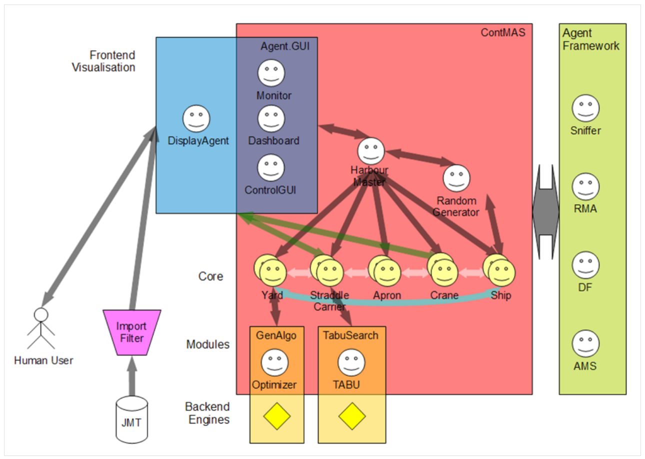


Figure 4: Emulation Architecture

by the allocation of individual containers to specific blocks, without considering the specific placement of containers within each block. The Integer Programming model used was replaced in [1] by an Evolutionary Algorithm, with operators designed to enforce the majority of the problem constraints.

Preston and Kozan [8] propose a Container Location Model (the plan for container storage in the yard) based on an Evolutionary Algorithm (EA) to optimise outbound transfers for a single ship assuming one of three deterministic types of loading schedule. This was extended in [7] by integrating a Container Transfer Model (the sequence of container moves between yard and vessel) with the Container Location Model; the integrated approach produced better results than having independent location and transfer optimizations.

Zhou *et al.* [14] made the point that it is unrealistic to ignore uncertainties in simulations of container yard operations and so adopted a fuzzy-coefficient model to capture the stochastic nature of operations.

4. AN AGENT-BASED SIMULATOR

As basis for the optimization approaches, we present the open source port emulation model *ContMAS*[2]. It consists of several types of agents, which cooperate through (asynchronous) message communication to achieve a common goal, namely the (intelligent) unloading of containers from a ship to the yard. Although it is configured to do so using QCs and SCs to fit the situation in the given local port, the model is flexible and can easily be configured to match any different port setup. *ContMAS* is designed as a tool for general port emulations. With its help results derived from different projects using the same model will be more easily comparable than current heterogeneous approaches. Also extensions developed during such projects may be integrated to form a better general model. In particular, *ContMAS* can be used for planning a new or re-planning a present port, to test and compare optimization and operation strategies, to simulate utilization scenarios or to run a just-in-time troubleshooting support tool in an actual port.

[2]LGPL, available at http://www-stud.uni-due.de/~sehawagn/contmas/page/index_en.html

Architecture

ContMAS utilizes the *Java Agent DEvelopment Framework* (*JADE*[3]) maintained by Telecom Italia. It is plain Java and uses *Jena* to access .owl files for certain features. It can be run as a project in the agent platform management tool *AgentGUI* [4], which then provides a bird's eye view of the physical layout. The *Protégé* ontology editor serves as a tool for the development of the communication ontology and configuration interface.

The system is structured into core agents, user interface agents, administrative agents and module agents (see Figure 4). Core agents represent the different handling devices in a port, but are generally modeled as one generic type with multiple options of configuration. This provides a flexible approximation of the reality while reducing the complexity by abstraction. *ContMAS* realizes a decentralised planning process since agents maintain and are solely responsible for individual plans for the execution order of container moves. In its basic version *ContMAS* allows the simple emulation of the activities within a harbor. However, in order to test different optimization strategies agents can be equipped with “intelligence”, which means that a specific strategy can be added to each agent. Alternatively, agents can use advisors in order to come to a better decision. This approach is the one which we will describe in more detail here. While *ContMAS* guarantees the autonomy of each agent it nevertheless allows agents to access advisors (called module agents) in order to get specific advice for their planning purposes. Accessing such advisors is just an option and does not imply that the agents need to do what the advisor suggests. An advisor can be seen as a central planning component and will be accessed by all agents in order to realize a specific optimization strategy. While the existence of a central planning tool contradicts in general the idea of a distributed and autonomous agent environment it provides an ideal solution for an emulation tool which does not need to react under real-time conditions. However, the fact that we have central components means that planning strategies can easily be replaced by other ones. This permits an easy testing and comparison of different optimization strategies. In this paper we will present two already developed optimization strategies, one which deals with the optimization of the allocation of container moves to straddle carriers (Section 5), and one to optimize the allocation of container positions in the yard (Section 6).

Agents

The core agents are called *ContainerHolderAgents*. Those are the agents which can pick up, transport ("hold") and drop containers, one for each individual device or other actor, such as cranes, ships, straddle carriers, yard areas or apron areas. *JADE* uses ontologies on a very basic level to support the communication between agents. *ContMAS* exploits the ontology objects as internal data storage for the current physical position and state of the agent, including all currently held containers and those which are about to get passed from or to other agents. This allows us to use *Protégé* as a graphical tool during development and as an editor for the run-time configuration of the emulation. Also it establishes an open interface for external applications, e.g. for the import and export of data in real-life systems.

There are several other agents in the model. The *HarbourMaster* controls the setup and events such as creation of a new agent

[3]Not to be confused with the Jade product from Jade Software Corporation.

[4]currently under development by Dipl.-Ing. Christian Derksen and Nils Loose at University Duisburg-Essen http://www.dawis.wiwi.uni-due.de/team/christian-derksen/

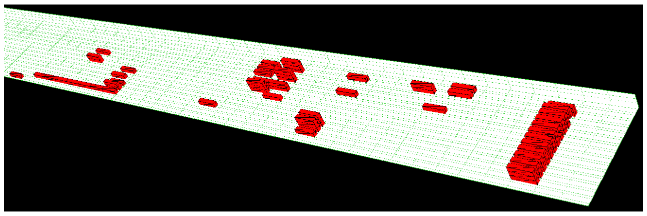


Figure 5: Example Yard

e.g. for a newly arriving ship. The *ControlGUIAgent* provides the graphical interface for the human user. The *RandomGenerator* provides random numbers or events for simulations. Agents are also used for visualisations such as a bird's eye map or 3D rendering of the containers held by an agent (e.g. Figure 5).

Environment

The logical structure of the port is modelled as a tree of *Domains*. Each *Domain* lies in exactly one parent *Domain* and can contain several sub-*Domains*. An agent lives in exactly one *Domain* and can be capable of accessing several other *Domains*. For example, a *Crane* lives in the *Domain CraneRails* and can access the *Domain TrainRails, ShipBerth* and *StraddleCarrierStreet* to exchange containers with agents living in those *Domains*. This structure permits to dynamically calculate the shortest logical path through the port for a container, along with possible alternatives for comparison and weighing.

Data structures

When communicating about containers and their transportation, agents always use a data structure called *TransportOrderChain* (*TOC*). It represents the path of a container through the port in terms of several *TransportOrders* as chain links, each for one step between two domains or agents. A *TransportOrderChain* can, for example, consist of *TransportOrders*, one for each step ship$\rightarrow$crane, crane$\rightarrow$apron, apron$\rightarrow$straddle carrier and straddle carrier$\rightarrow$yard. The agents administrate all containers (and therefore *TransportOrderChains*) which are known to them through a list; each is marked with one *TransportOrderChainState*, which indicates that the agent has sent a proposal for a *TOC*, a pick-up is planned, the container is currently held (transported/stored), a call for proposals has been issued, or the drop is planned. The data structures provide all information needed during the negotiations and at the same time work as flexible control artifacts as shown in the next sections. They abstract from the diverse ways of handling containers between the different devices towards a universalised view.

Operation

All negotiations between the agents are carried out by means of an extended contract net protocol: Any agent currently holding a container, e.g. a ship, initiates a call for proposals (CFP) to other suitable agents, e.g. cranes. They respond with a REFUSE or PROPOSE message, in the latter case containing the possible time of pick-up. The initiating agent then decides on one of the proposals and sends an ACCEPT message to that agent; all other agents get a REJECT message. Through this message exchange, the issuing agent and the determined contractor established a time and place to meet physically to hand over the container in question. Both agents move independently and can also negotiate with other agents about more containers in the meantime, thus building up a local plan.

When the agreed upon time is reached, both agents should have moved to their negotiated position and the initiating agent issues a REQUEST to execute the appointment, i.e. to hand over the container. The contractor will acknowledge with an INFORM message. At this point, the administration over the container changes from the initiating agent to the contractor, which can itself become an initiator and issue a CFP for the next step of transportation, e.g. from crane to apron, adding a new *TransportOrder* to the *TransportOrderChain*. The communication between very different types of handling devices and other agents therefore is homogeneous and the *TransportOrderChain* builds up during negotiation. The set of all TOCs reaching the yard in one run contains a complete record of all container transfers and the corresponding time represents a trace or resulting global plan for a given setup and can be used to advise human actors in the port.

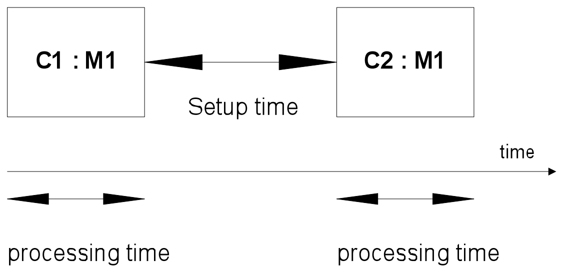


Figure 6: Setup and Processing Time

Additional Features

In *ContMAS*, it is also possible to import *TOC*s with varying amounts of detail to determine the level of freedom the agents have. This allows the emulation to be used for several purposes, such as testing pre-planned runs against plans that have been produced just-in-time through negotiation or "replay" data from a real port, as was done in our project. New features can easily be introduced by development of a new agent and/or by adding communication capability modules to the agents of the model.

5. OPTIMISATION OF ALLOCATION OF MOVES TO STRADDLE CARRIERS

As mentioned in Section 2, one of the problems that we focus on is the management of Straddle Carriers. If Straddle Carriers are not managed well, then Quay Cranes can be idle, waiting for containers to be provided for loading, or for apron space to clear up so that they can unload containers from the ship. Straddle Carrier management is thus an important sub-problem. We have thus developed a negotiation-based optimisation strategy to allocate container moves to Straddle Carriers.

A key feature that distinguishes this mechanism from the process used by *ContMAS* is that we develop a schedule of planned moves ahead of time, whereas in *ContMAS* until now container moves are put up for bids when the machine is ready to dispose of the container. Planning ahead and therefore negotiating over containers not yet held by a machine is a feature likely to be included in one of the next versions of *ContMAS*.

In our approach each machine agent, Quay Crane or Straddle Carrier, maintains a *schedule* of container moves. In essence this is a timeline: a list of container moves (each with associated source and destination locations) with associated timing information. The timing information is defined in terms of *processing* time and *setup* time. Processing time is the time taken to move a given container from its source location to its destination location. The processing time depends only on the container and the machine. The setup time is the time between finishing one container and being ready to pick up the next container. It depends on both containers. These times are depicted in Figure 6.

The container move allocation problem is how to assign the needed container moves to machines (QCs and SCs) in a way that meets all constraints, while attempting to reduce the overall processing time. A *solution* to the container move allocation problem is a set of machine, each with a schedule of container moves. Taken collectively, the moves proposed must correctly unload the ship[5] while respecting the various constraints. The overall goal is to reduce the ship turnaround time, and thus the overall quality of a solution is given by its cost: how much time is taken? This is simply the completion time of the last machine to complete (i.e. the maximum of the finishing time over the machines).

The process for deriving a solution has two phases: initial allocation and optimisation. In the initial, allocation, phase each container is considered in turn and is put up for "auction", with each (relevant[6]) machine bidding. The container is allocated to the cheapest bidding machine, and is inserted into its schedule. In doing the initial allocation we need to ensure that each container is unloaded from the ship before it is scheduled to be moved to the yard. This is done by tracking the time at which a container reaches the apron (its "minTime") and ensuring that the Straddle Carrier does not move the container earlier than its minTime.

The second phase, optimisation, attempts to improve the initial solution by repeatedly modifying it. The modifications considered involve selecting a container and considering (a) moving it to a different position in its machine's schedule; or (b) moving it to a different machine. The possible reallocations are evaluated based on (an approximation) of the cost difference, and the cheapest one is chosen and applied to the current solution, in order to obtain a new solution.

In proposing and applying container move reallocations, we need to ensure that we do not reallocate a container move in a way that violates the "unload before move" constraint. We also need to ensure that we do not propose to deal with containers in a non-viable sequence (e.g. unloading C1 before C2 if C1 is below C2 in the ship).

When a container move is allocated to a machine, the machine's schedule may need to be adjusted, since moves that are after the new move are delayed. Similarly, when a container move is taken away from a machine, the container moves that were scheduled after it may be able to be done earlier, and so again, the schedule needs to be adjusted. Note that any adjustments to a container's start and end time may affect other container moves allocated to different machines.

The process described (briefly) above is performed before machines begin performing moves, and develops a complete scheduled plan for unloading a ship. A strength of the approach is that should something go wrong, the schedule can be updated to reflect necessary changes, and the allocation process re-run in order to deal with the change. For example, should a Straddle Carrier break down, the solution is updated by removing the Straddle Carrier in question, and putting its allocated container moves back into the list of moves to be allocated. The allocation process is then re-run to allocate these container moves to other Straddle Carriers. In order to support this style of usage the solution includes not just a set of machines (with their associated schedules), but also a list of unallocated container moves, and the "current time". The latter is used during the process of container move allocation and optimisation to ensure that container moves which have already been done (i.e. are in the past) do not get changed.

[5]We have focused on unloading only in our work so far.

[6]A machine is relevant to a container move if it is able to perform that type of move. For example, a Quay Crane is relevant to a move which involves the apron and a ship.

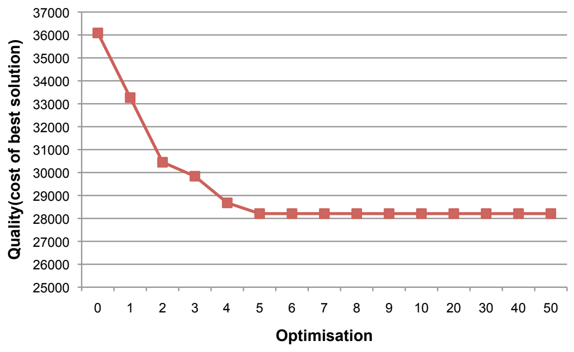


Figure 7: Experimental Results (Small data set)

We have implemented this negotiation-based approach for container management. The implementation makes use of a Tabu Search framework (OpenTS[7]). Although the framework is centralised, and thus not an accurate model of a distributed agent-based solution, in other ways it is a good model of the process for developing a solution. Specifically, the optimisation phase maps very nicely to a Tabu search approach in which an initial solution is developed, and then is progressively modified by applying small changes, selected based on their costs.

The implementation has been integrated with *ContMAS* as an "advisor": when machines bid for container moves in *ContMAS*, they may consult the prepared schedule, and use this to guide whether they submit bids or not.

We now briefly present some (initial) experimental results. These were derived using real (anonymised) data from the local port, with only data fitting our simplified scenario being used (unloading only, using only one area in the yard, and with no housekeeping moves considered). Specifically, the experimental data has been extracted from the JMT system which is used by the local port. We have used data from January 2009. This data contains all recorded machine activity (e.g. "SetDown", "PickUp") as well as a record of containers arriving and leaving the port (e.g. "ReceiveShip", "ReleaseShip"). Each record includes a date and time stamp, as well as (where relevant) the machine involved and its type (QC or SC).

Our experiments used two data sets: a small data set and a larger data set. Both data sets involve a single ship. The smaller data set has 30 containers, 1 Quay Crane and 4 Straddle Carriers. The larger data set has 82 containers, 3 Quay Cranes, and 8 Straddle Carriers. Figure 7 shows the solution quality (i.e. its cost: smaller is better) as the optimisation step is repeatedly applied. As can be seen, the initial solution resulting from the initial allocation phase (at x=0) is improved by the first few optimisation steps. Similarly, Figure 8 shows that for the larger data set, the initial solution is also improved by the first few (in this case five) optimisation steps, after which point no further improvement is found.

[7]http://www.coin-or.org/Ots

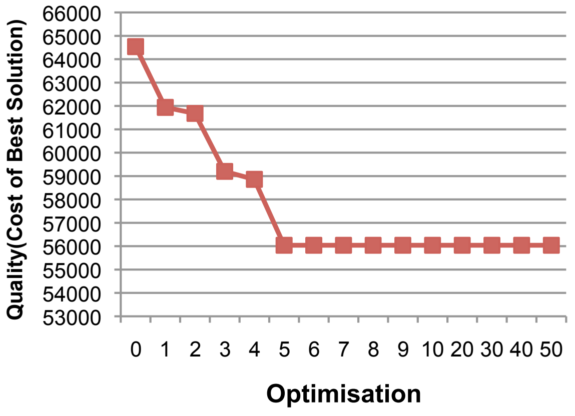


Figure 8: Experimental Results (Large data set)

6. OPTIMISATION OF CONTAINER YARD ALLOCATION

The method used to allocate containers to storage locations within the yard has a significant effect on the profitability of a container terminal. Beyond the expected operational costs of yard operations, penalties often apply if a vessel must be held at the terminal beyond its agreed berthing window to complete loading and unloading operations. Extra time may be needed to extract a container that is stored underneath other containers ("overstows".) Every extra container move ("rehandles") from one location to another within the yard adds to the operating cost. Therefore container terminal operators normally attempt to arrange the containers within the yard with the aim of minimising the berthing times for the vessels that are expected, and of avoiding double handling of containers. Finally, the allocation pattern affects the vehicle movements within the yard, which may have unintended consequences such as an increased likelihood of yard vehicle collisions.

Yard allocation is further complicated in practice by various uncertainties, for example:

- Containers that arrive into the yard by land may do so at any time within a nominal time period — 3 weeks at the local port; unlike many ports, the local port does not impose a cut-off date/time for container delivery. The order in which containers arrive in therefore has no relationship to the order in which they will leave, and there may be no time to rearrange containers in the yard prior to loading.

- Ships sometimes do not arrive at all. All of the containers scheduled for a particular voyage must then be rebooked on other vessels or returned (if that is possible.) More significantly for yard operations, those remaining containers may now be obstructing access to containers for later voyages.

- Tide or weather conditions may result in a ship being berthed with a reversed orientation — with the bow being where the stern was expected. As the loading sequence depends on the orientation this can create a significant problem of overstows.

For these reasons most prior research (see Section 3) has addressed various sub-areas within the overall problem of optimising container locations within the yard. This prior research indicates that simplified variants of the Yard Allocation Problem are amenable to optimisation, however no single approach to date has addressed the complete problem. Note that this is a multi-objective problem with many, often conflicting, constraints to be satisfied. To try to tackle this task we have implemented an optimisation interface within *ContMAS* that allows a variety of approaches to be tried.

Agents that need a yard location in which to store an incoming container make a request of a *BayMapOptimiser* agent, which calls one of potentially several optimisers. Each optimiser when called is given an ordered list of expected container moves (either from yard to ship or, as initially implemented, from ship to yard), and the current yard state, and must answer with a list of the optimal storage locations within the yard for each container.

Our initial module, described below, uses an Evolutionary Algorithm.

The Evolutionary Algorithm Module

Given a sequence of expected container moves and a representation of the current yard state, an Evolutionary Algorithm (EA) is used to attempt to optimise the location for each container placement within the yard. Straddle Carriers, such as are used within the yard we modelled, can only place or remove the uppermost container in a stack. Any time a Straddle Carrier needs to move a container that is part-way down in a stack, it must first remove all the overstowed containers to a free location in the yard, and this is obviously expensive. This implies that the order of operations is important; containers should be loaded in a stack in the opposite order to which they will be removed. We therefore hypothesised that the sequence of container moves should be explicitly addressed by each component of the EA, and so our EA consists of an ordered genome representation, a fitness function that simulates the sequence of moves when assessing costs, and custom order-preserving crossover and mutation operators.

Genome representation: the genome is made up of a sequence of (container id, yard location) genes, where each gene represents a move of a particular container to a [lane,bay,tier] location within the yard. Order is significant — the genome (AA12, [0,0,0]) (BB34, [1,1,1]) is not the same as the genome (BB34, [1,1,1]) (AA12, [0,0,0]). Containers usually appear more than once in the genome, to capture moves into and out of the yard; any additional entries for a given container signal potentially costly rehandles, or moves within the yard.

Fitness function: the fitness of each entity is calculated by simulating the sequence of moves encoded in the genome to a simplified representation of the container terminal. Each move between yard and ship has a cost equal to the 'Manhattan' distance of the move. Any genome that encodes an invalid sequence of moves is given a high penalty cost. Invalid sequences include ones where a required move is missing from the genome, or falls out of sequence, or a move involves an overstowed container. A binary tournament is used to select a sample of the best (lowest cost) genomes for the next generation.

Mutation operator: mutation acts only on the location encoded within a specified proportion of genes in the genome; mutation does not affect the order of container moves. A random set of genes is selected, and each gene in the set gets its location adjusted to a random free location in the yard:

```
function Mutation(Genome A):
    for 0 to number_of_mutations:
        randomly pick a gene from Genome A
        set the gene's location to a free location in the Yard
    return mutated Genome
```

Crossover operator: The order of genes in the genome is significant, so the crossover operator developed attempts to preserve the

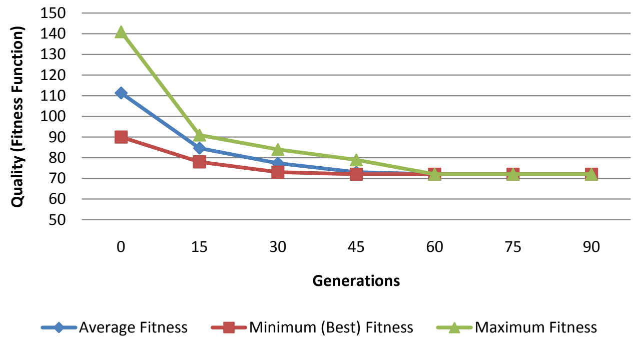


Figure 9: Preliminary Experimental Results (minimal complexity problem)

order of genes from each parent in the new child genome. It does this by identifying locations unique to the second parent, and then switching those for the locations of a random proportion of genes in the first parent, so leaving the order of moves untouched. This can result in invalid genomes, e.g., where the crossover results in a container to be in mid-air. Such invalid genomes are repaired by dropping mid-air containers down the stack to a supported position:

```
function Crossover(Genome A, Genome B):
    location_list = locations in Genome B that are not in Genome A
    new child Genome = copy of Genome A
    for 0 to some number_of_crossovers:
        randomly select a gene in the child Genome
        set that gene's location value to the next location in
                the location_list
    return child Genome
```

For example, if GenomeA = (AA12, [1,1,1]) (BB34, [2,2,2]) and GenomeB = (CC56, [1,1,1]) (AA12 ,[3,3,3]), the unique locations in Genome B would be simply [3,3,3], and one possible child genome would be (AA12, [3,3,3]) (BB34, [2,2,2,]). The effect is to construct a child genome that maintains the move ordering of Genome A, while incorporating location material from Genome B.

Figure 9 shows results from a single EA run of an example scenario involving the extraction of 20 containers from a $10\times3\times2$ yard. The containers are requested according to a fixed schedule and the EA is initialised with a population of 200 individuals. The figure shows the population converging on a lower (that is, better) fitness value, where the fitness value is proportional to the total distance of Straddle Carrier moves to complete the scenario.

7. DISCUSSION

We have presented a port emulation platform (*ContMAS*) which has been implemented as a multi-agent system. The *ContMAS* platform can serve as an integrating framework for solutions to different aspects of the container terminal management and optimisation problem. *ContMAS* can be seen as a basic emulation tool which can be equipped with different optimisation strategies (called module agents). These module agents are easily replaceable, thus, allow an easy integration of different optimisation strategies. This has been illustrated with the description of two specific aspects of the problems that we have tackled, namely yard allocation and straddle carrier move optimisation, which have been investigated, implemented, and integrated with *ContMAS*. Furthermore, we have conducted evaluations (some of which is covered in this paper) which have shown that our optimisations are effective, e.g. that the EA model is able to improve upon random yard allocations.

Overall, our conclusion is that taking an agent-based approach has proven to be a natural choice because of the nature of ports in which many actors work together with a high degree of autonomy. The agent paradigm supports the modeling of such an environment with a high level of detail, flexibility and consistency with the archetype by assuring as little transfer friction as possible. The use of a free and mature agent framework reduced development overhead and enabled access to easy exploitation of parallelization potential and advanced features like agent mobility for further development.

There is a range of directions for future work. One direction is to allow the replace the centralized module agents by a distributed solution; i.e., an individual, agent-inherent mechanism for SC optimisation. The challenge is, among others, that the optimisation phase currently considers a very large number of possible reallocations, and we need to reduce this for a distributed implementation to be practical. We are in the process of investigating heuristics for doing this reduction.

Finally, regarding the EA-based yard allocation mechanism, our hypothesis that an order-based EA is required to optimise the complete yard allocation problem is as yet untested. We plan to conduct experimental tests against a control random allocation policy, and against the current 'block allocation' heuristic used by the local port.

8. REFERENCES

[1] M. Bazzazi, N. Safaei, and N. Javadian. A genetic algorithm to solve the storage space allocation problem in a container terminal. *Computers and Industrial Engineering*, 56(1):44–52, 2009.

[2] V. J. Botti. Multi-agent system technology in a port container terminal automation. *ERCIM News*, 56:37–39, January 2004.

[3] L. Chen, N. Bostel, P. Dejax, J. Cai, and L. Xi. A tabu search algorithm for the integrated scheduling problem of container handling systems in a maritime terminal. *European Journal of Operational Research*, 181:40–58, 2007.

[4] L. Henesey, P. Davidsson, and J. A. Persson. Agent based simulation architecture for evaluating operational policies in transshipping containers. *Autonomous Agents and Multi-Agent Systems*, 18:220–238, 2009.

[5] M. Kefi, O. Korbaa, K. Ghedira, and P. Yim. Container handling using multi-agent architecture. In N.T. Nguyen et al., editor, *KES-AMSTA*, volume LNAI 4496, pages 685–693. Springer, 2007.

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

[7] E. Kozan and P. Preston. Mathematical modelling of container transfers and storage locations at seaport terminals. *OR Spectrum*, 28(4):519–537, October 2006.

[8] P. Preston and E. Kozan. An approach to determine storage locations of containers at seaport terminals. *Computers & Operations Research*, 28(10):983–995, September 2001.

[9] M. Rebollo, V. Julian, C. Carrascosa, and V. Botti. A multi-agent system for the automation of a port container terminal. In *Workshop on Agents in Industry*, 2000.

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

[11] D. Steenken, S. Voß, and R. Stahlbock. Container terminal operation and operations research — a classification and

literature review. *OR Spectrum*, 26:3–49, 2004.

[12] T. Thurston and H. Hu. Distributed agent architecture for port automation. In *Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC)*. IEEE Computer Society, 2002.

[13] C. Zhang, J. Liu, Y. Wan, K. Murty, and R. Linn. Storage space allocation in container terminals. *Transportation Research Part B-Methodological*, 37(10):883–903, December 2003.

[14] P. Zhou, H. Kang, and L. Lin. A fuzzy model for scheduling equipments handling outbound container in terminal. In *6th World Congress on Intelligent Control and Automation*, pages 7267–7271, June 2006.