Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A.M. Uhrmacher, eds

EMBEDDING SIMULATION IN YARD CRANE DISPATCHING TO MINIMIZE JOB TARDINESS IN CONTAINER TERMINALS

Shell Ying Huang
Xi Guo
Wen Jing Hsu
Wei Lin Lim

School of Computer Engineering
Nanyang Technological University
SINGAPORE, 639798

ABSTRACT

Two optimal algorithms, MTA* and MT-RBA*, are presented to find the optimal yard crane (YC) job sequence for serving a fleet of vehicles for delivery and pickup jobs with scheduled deadlines and predicted vehicle arrival times. The objective is to minimize the total tardiness of incoming vehicle jobs. This is important for minimizing vessel turnaround time. In the search for an optimal job sequence, the evaluation of the total tardiness of (partial) job sequences requires sequence dependent job service times. Simulation is embedded in our optimization algorithms to help provide accurate YC service times. This results in a more accurate evaluation of job tardiness but incurs costs. Experimental results show that this is feasible despite the simulation costs. MTA* and MT-RBA* significantly outperform the Earliest Due Date First and the Smallest Completion time Job First heuristics in minimizing job tardiness. MT-RBA* is computationally more efficient.

1 INTRODUCTION

Container terminals serve as crucial hubs in the global transportation chain of the ever increasing cargo flows. When a vessel berths at a terminal, a number of Quay Cranes (QCs) are allocated to serve the vessel. QCs first unload containers from the vessel onto in-terminal vehicles for transferring them to the container storage yard. A vehicle would take the containers to specific job locations at various yard blocks in the storage yard. Yard Cranes (YCs) pick up the containers from the vehicles and temporarily store them in the yard blocks. The operation of loading containers onto a vessel is carried out in the reverse order. External trucks come into the storage yard through terminal gates to the yard blocks to get export containers unloaded or import containers loaded by YCs.

One of the most important objectives of terminal operations is to reduce vessel turnaround time (Steenken *et al*. 2004). Previous studies have pointed out that YC operations are of great importance and are likely to be a potential bottleneck to the overall terminal performance (Li *et al*. 2009). This is because when vehicles are delayed in the storage yard, they will not be able to reach their QCs on time. As a result, QCs' loading/unloading operations will be delayed and vessel turnaround time lengthened. In YC operation management there are two main problems: (i) deciding job sequence for an YC which we refer to as the YC dispatching problem; (ii) allocating YCs to different parts of the yard which we refer to as the YC deployment problem. We study the YC dispatching problem in this paper.

The YC dispatching problem was studied by Kim and Kim (1999) where they considered the loading operations only for a single YC with a given load plan and a given bay plan. A Mixed Integer Programming (MIP) model is proposed to minimize the total gantry time of the YC. Later, Kim and Kim (2003)

978-1-4673-4781-5/12/$31.00 ©2012 IEEE

and Kim *et al.* (2004) extended the study of this problem by comparing exact optimization, a beam search heuristic and a Genetic Algorithm (GA). Ng and Mak (2005) developed a heuristic to solve the single YC dispatching problem with different ready times to minimize the total job waiting time. It is known that for large problems, the MIP model has limited applicability due to the excessive computational times. On the other hand, heuristics cannot guarantee optimal solutions. Guo *et al.* (2011) applied A* search to compute optimal single YC dispatching sequence based on vehicle arrival times to minimize vehicle waiting times.

Jung and Kim (2006) considered 2 YCs working in one shared zone to support vessels loadings with a GA and a Simulated Annealing (SA) algorithm to minimize the make-span, i.e. the period between the starting time of the first YC operation and the finishing time of the last YC operation. Lee *et al.* (2006) considered 2 YCs working in 2 non-overlapping zones with a SA algorithm to minimize the make-span. Cao *et al*. (2008) considered Double-Rail-Mounted gantry (DRMG) crane systems where two YCs can pass through each other along a row of blocks with a combined greedy and SA algorithm to minimize the loading time of containers.

In many works presented in the past, the objective of the YC dispatching algorithm is to minimize the total (average) vehicle waiting time or to minimize the make-span. While minimizing vehicle waiting times or minimizing make-span often helps reduce the vessel turnaround time, they may result in vehicles getting to the quayside earlier than they are needed. A more effective way to help reduce vessel turnaround time is to help minimize QC waiting time for a vehicle. This translates to minimizing vehicle job tardiness at a yard block. The vehicle job tardiness is with respect to the time a vehicle is scheduled to finish the transfer of container from/to a yard block. This time is referred to as the deadline of the vehicle job. It is not with respect to the time the vehicle arrives at the yard block. For example, consider a loading job, based on *the time a QC needs the vehicle at the quayside*, the time this vehicle should leave the yard block with the container to travel to the quayside (the deadline of this vehicle job) can be derived assuming no traffic congestions. The deadline of the job is the time the QC needs the vehicle minus the expected travel time from the yard block to the QC. Depending on which vehicle is assigned this loading job, *the vehicle's arrival time at the yard block* can be derived/predicted. Given the deadlines of the vehicle jobs and their predicted arrival times at the yard block, the YC dispatching algorithm that computes its serving sequence to minimize the average (total) job tardiness will be more effective in reducing vessel turnaround time.

When designing a YC dispatching algorithm, many previous work assumes that the YC service times for vehicle jobs are constant (Cao *et al*. 2008; Jung and Kim 2006; Lee *et al.* 2006 Guo *et al.* 2011). This is correct when the container to be retrieved/stored is on the top of the container stack in the yard. The scenarios where the containers to be retrieved/stored are not on top of the stack and thus the times to reshuffle some containers are not considered. This assumption simplifies the logic of the algorithms but reduces the applicability of the algorithms in real operations. However, it is not a simple matter of assuming that YC service time will be x times the time to move one container if the container to be retrieved is x^{th} from the top of stack at the time the YC job sequence is planned. Consider a container *a* that is 3^{rd} from the top of the stack at the time the job sequence is to be planned. The two containers above *a* may or may not be in the set of YC jobs to be planned or one of them may be in the set of jobs. If both of them are not in the set of jobs, they need to be moved to neighbouring rows before *a* can be retrieved. If both of them are in the set, one, two or none of them may need to be moved before *a* can be retrieved, depending on whether one or none or both have been retrieved earlier in the job sequence. Therefore, YC service time for a job is sequence dependent. When a YC dispatching algorithm searches for an optimal job sequence, the sequence dependent service times need to be used to evaluate the quality of a sequence among the alternatives.

Simulation of YC job sequence will enable accurate estimation of YC job service times which leads to accurate evaluation of total job tardiness of a sequence. Without employing simulation, an alternative may be to use the tier number of each container job at the time the YC job sequence is planned to estimate how many containers need to be moved and deduce the service times. This obviously will return wrong

service times from time to time. On the other hand, simulation of each job sequence incurs computational costs and YC dispatching is NP-hard (Narasimhan and Palekar 2002). Whether embedding simulation in the YC dispatching algorithm will incur prohibitive computational costs needs to be investigated.

We propose two provably optimal algorithms for an YC to handle the jobs in its assigned zone within a planning window efficiently. The algorithms take the predicted job arrival times and deadlines as inputs and minimize average job tardiness. We embed simulation to estimate the sequence dependent YC services times for jobs and to compute total job tardiness of a sequence. Our algorithms are evaluated against the Earliest Due Date first (EDD) and the Smallest Completion time Job First (SCJF). Our algorithms could find the best dispatching sequence with reasonable computational time in solving problems of practical sizes.

The rest of the paper is structured as follows. Firstly, a formal description of the YC dispatching problem is given in Section 2. Then two new algorithms with embedded simulation are proposed in Section 3 and the experimental evaluations of the proposed algorithms are presented in Section 4. Conclusion is drawn in the last section.

2 PROBLEM FORMULATION

2.1 General Description

The following assumptions are made in the YC dispatching model:

- Each vehicle job involves one container only. If a vehicle carries two containers stacked together and to be delivered at different locations, it will go to the delivery location of the top container first. A vehicle carrying multiple containers not stacked together and to be delivered at the same destination slot location is modeled as multiple container jobs as explained later.
- In each working zone in the yard, there is only one YC. Vehicles come to the zone for loading or unloading of containers.
- The deadlines of vehicle jobs are given.
- The vehicle arrival times can be predicted for a relatively short planning window, e.g. 30 minutes, and are given.
- YC gantry time between two job positions could be predicted with high accuracy as gantry speed is usually quite consistent.
- The slot location (yard bay), the row and tier numbers of the container in each vehicle job are given.
- The yard block status (how many containers are stored in each stack) is given.

In our formulation, the following notations are used:

$J = \{1, 2, \ldots, n\}$, the set of job identifiers in a YC's working zone for a planning window
a_i the arrival time of job i.
p_i the process time of job i by an YC.
d_i the deadline of job i
m_{ij} the time for YC gantry from the position of job i to that of job j.
S_i the time an YC starts processing job i.
C_i the time an YC completes processing job i.

J is the set of jobs to be sequenced. m_{0j} is the YC gantry time from its position at the start of the time window to the position of job j. C_0 is the time the YC is available to start to move to the position of its first job in the YC dispatching sequence.

For a set J of n vehicle jobs with predicted job arrival times a_i ($i = 1, 2, \ldots, n$), deadline d_i ($i = 1, 2, \ldots, n$) and YC gantry times m_{ij} ($i = 0, 2, \ldots, n$; $j = 1, 2, \ldots, n$), the tardiness of a job J_i is defined as $T_i =$

max (0, C_i - d_i), where C_i is the completion time of J_i. The objective of YC dispatching is to find a sequence so as to

$$\text{Minimize } \frac{1}{n}\sum_{i\in J} T_i$$

The completion time for job i is equal to its start time + process time, that is, $C_i=S_i+p_i$. When vehicle arrivals cannot be predicted, an YC can only start to move towards the next job location after the actual job arrival. Job starting time in this case could be derived as in Equation (1) where job *i* is the current job and job *j* is the next job. If vehicle arrivals can be predicted and the next job is decided, an YC is able to start moving towards the next job location before the actual vehicle arrival. This is referred to as the pre-gantry ability. Job starting time with YC pre-gantry ability is shown in Equation (2). The advantage of the pre-gantry ability is the possibility of utilizing YC idle time between jobs to transfer between different job locations.

$$S_j = \max\left(C_i, a_j\right) + m_{ij} \quad (1)$$

$$S_j = \max\left(C_i + m_{ij}, a_j\right) \quad (2)$$

Job process time p_i is the YC service time for the job. It is a variable which cannot be pre-determined. More discussions on p_i are given later.

The YC dispatching model is flexible to include operation conditions where some vehicles carry more than one container not stacked on top of each other, to be loaded/unloaded at the same slot location. They could be simply modeled as several container jobs with the same arrival time but different deadlines. For situations where several containers are to be loaded/unloaded at the same slot locations one after another, they will be individual jobs with their own vehicle arrival times, possibly one after another. In both situations, the YC dispatching algorithm will find a job serving sequence which returns the minimum total tardiness for all the jobs. The average job tardiness is the total tardiness divided by the number of jobs.

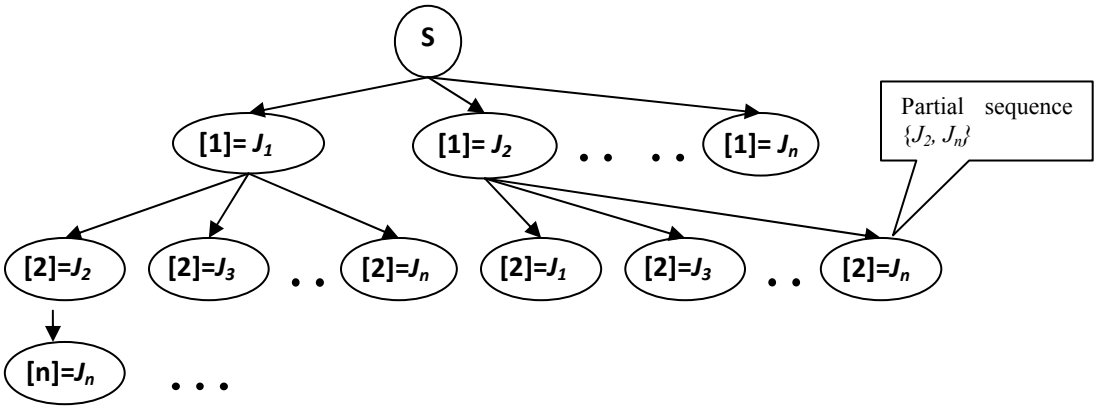

Figure 1: Search Space of the Problem.

2.2 YC Dispatching Reduced to a Tree-search Problem

We believe that solving the YC dispatching problem as described in the last section by integer programming approach will not be feasible for practical applications. Given an YC dispatching problem of *n* jobs, there are *n*! possible dispatching solutions in total. The solution space can be transformed into a tree-search problem as shown in Figure 1. The root of the tree is the start node before the first job is selected. Each path from the start node to a leaf node in the tree represents a complete dispatching sequence of height *n*. The edge weight from node *i* to node *j* has a value equal to the tardiness of job *j* if the YC is to do job *j* immediately after finishing job *i*. This edge weight is given by

$$W_{ij} = max\ \{0,\ max\ (C_i + m_{ij},\ a_j) + p_j - d_j\} \quad (3)$$

Note that m_{0j} is the YC gantry time from its position at the start of the job sequence to the position of job j and C_0 is the time the YC is available to start to move to the position of its first job in the YC dispatching sequence. The task is to find a path of minimum total distance (i.e. minimum total tardiness for a given n) from the start node to a leaf node that visits each job exactly once. From (3), it can be seen that each edge weight is not pre-defined and depend on i. In addition, the YC process time p_j depends on not only i but also all other predecessor jobs in the path from s to j. The YC process time depends on whether this job is at the top of the stack when YC comes to serve this job. A job which is not at the top of the stack at the beginning of the job sequence may become top of the stack if the containers above it are loaded onto vehicles in the early part of the job sequence. We assume that containers at the top of a stack at the beginning of the job sequence will not be blocked by other containers during reshuffling in the early part of the job sequence). Our algorithm will guarantee this. p_j needs to be dynamically computed in the process of searching for the optimal job sequence.

2.3 Dynamic Computation of Job Service Time by Simulation

At the beginning of the planning window, the number of containers in each stack of the yard block is given. The slot, row and tier numbers of the container of each vehicle job are also given. To determine the service time of a job [i] in a path in the search tree in Figure 1, simulation of the YC operations starting from the initial yard block status and the first job in the path will be an effective technique. Each job in the path will change the yard block status which may affect the service time of jobs later in the path. We have the following assumption in our simulation of the YC operations.

- When storing a container, the YC will put the container at the top of the allocated stack.
- When retrieving a container, the YC will simply take the container if it is on the top tier of the stack.
- When retrieving a container that is not on the top tier of the stack, the YC will move the container(s) to the top of a stack of a different row in the same slot (bay). In doing so, the YC will not choose a stack which has reached the maximum stack height or it will block the container of a job later in the job sequence. If a suitable stack cannot be identified, a stack in the neighbouring slot will be used. If a container moved is one for a job later in the job sequence, the job location of the affected job will be updated.
- The time to unload a container from a vehicle and put it on top of a stack, the time to take a container from the top of a stack and load it on a vehicle and the time to move a container from the top of one stack to the top of another stack are assumed to be approximately the same. Let this time be T_p.

3 OPTIMAL SEARCH ALGORITHMS

In finding the least-cost path of the YC dispatching problem, we need to traverse the tree to search for the optimal solution. As the problem is strongly NP-hard, exhaustive search would be time-consuming to perform. Here we propose to use modified A* search to reduce search time and yet to guarantee optimality. We derive a heuristic function using domain knowledge of YC operations for the modified A* search.

3.1 Modified A* Search: MTA*

A modified A* search algorithm (also called branch-and-bound algorithm) is employed to find the optimal job sequence for a YC in a working zone over a planning horizon. It involves simulating YC operations following various possible dispatching sequences. The algorithm takes as input the predicted vehicle job arrival times and their deadlines for jobs expected in the YC's working zone in the planning window. It also takes in the time the YC is available to start the first job in the planning window and its

initial location. This will be the time and position of the YC when this YC finishes its last job in the previous planning window.

A YC's dispatching sequence is built by adding jobs to a partial job sequence one at a time. A partial job sequence with job i as the last job is chosen to be expanded if the value of $f(i) = g(i) + h(i)$ is the minimum among all partial sequences. *g(i)* is the total tardiness from the start node to i and *h(i)* is the estimated lowest total tardiness from i to a goal node in the search tree in Figure 1. When a job j is added to a partial sequence, $f(j)$ is computed.

3.1.1 The Total Tardiness of a Partial Sequence, g(*i*)

Simulation is employed to compute the sequence dependent total tardiness of a (partial) job sequence. Figure 2 shows the pseudocode.

```
sequenceTardiness(JobList) {  // JobList with job [i] as the last job
  ReInititalizeYardBlock; //start simulation with initial yard block status
  FOR each job j in JobList    {
   serveJob(JobList,j);
   Calculate job_tardiness;
   update Cummulative Tardiness of Joblist
  }
}

serveJob(JobList, k) {    // called from sequenceTardiness()
  YC moves to JobList[k].slot;   // YC gantry move
  IF JobList[k].isLoading  {
   //Check for reshuffle and update the process time and ContainerLoc
   IF container not at top tier     // reshuffle
      FOR each container above the job  {
         Set this container's location to unoccupied
         moveContainer(container's slot, container's row, container's tier)
      }
   Load JobList[k] to vehicle;
   Set container's location to unoccupied;
  }  ELSE  {   // unloading
   Unload from vehicle to store in stack;
   Increment the stack height;
   Set container's location to occupied
  }
  p_k = number of containers moved * T_p;
}

moveContainer(s,r,t) {   //Called from serveJob()
   FOR each yard bay b starting from s
    FOR each neighbouring row of r  //Shifting to a neighbouring row
     IF height of stack(b, row)< maximum and this stack has no future job {
          Move the container to the stack;
          Set the location to occupied;
          IF container at (s, r, t) is a future job
              Update job location in JobList;
          Return;
     }
}
```

Figure 2: Pseudocode for simulation to compute $g(i)$.

The function sequenceTardiness() in Figure 2 simulates the YC operations when following a (partial) job sequence JobList and computes the total job tardiness of this sequence. A (partial) job sequence is a path starting from node s and ending at a certain node in the tree in Figure 1. The total tardiness is the sum of W_{ij} as defined by Equation (3) where (i, j) is an edge in the path. The sequence dependent $\boldsymbol{p_j}$ is obtained dynamically by simulating the YC operations in the job sequence. Note that since YC operations change the yard block status, the block status has to be restored to its original state before the simulation of a job sequence is done each time (first line in sequenceTardiness() in Figure 2).

Simulating YC operations for each (partial) job sequence has a cost. For each sequence, the computation time is $O(n^2)$ where n is the number of jobs in the sequence. It is $O(n^2)$ because for each loading job simulated, the job sequence needs to be searched to see whether a reshuffled container is a future job.

3.1.2 Lower Bound of Total Tardiness, h(*i*), of Jobs Not Sequenced Yet

An admissible heuristic *h(i)* is designed to estimate the lower bound of the total tardiness of jobs not included in the job sequence to help evaluate a partial sequence and guide the search for an optimal one.

$$h(i) = [\sum_{j \in F_{later}} (\max(\ C_i + m_{ij}, a_j) + T_p - d_j)] + [1 + 2 + ... + (|\ F_{later}\ | - 1)] * T_p +$$

$$\sum_{k \in F_{earlier}} Max\quad \{0, \max(\ C_i + m_{ik}, a_k) + T_p - d_k\} \tag{4}$$

h(i) computes the lower bound in total job tardiness of jobs not yet included in the YC dispatching sequence if job i is chosen to be the next job in a partial job sequence. F_{later} is the set of jobs that will definitely miss their deadlines. These are the jobs such that even if they are processed immediately after the completion of job i, their completion time will be after their deadlines. The other group $F_{earlier}$ is the rest of the jobs. In other words, a job is in F_{later} if its completion time would be after d_j even when it is the immediate next job to be served after job i. For each job j in F_{later}, the minimum tardiness is $\max(C_i + m_{ij}, a_j) + p_j - d_j$, because the earliest time job j may be started after i is $\max(C_i + m_{ij}, a_j)$. Since p_i for each future job not included in the partial sequence is unknown, T_p is used in its place so that we will never overestimate the tardiness. T_p is The minimum processing time of a job (time to load or unload one container without the need to reshuffle other container(s)).

In addition to the above minimum tardiness just explained, the second job to be handled in F_{later} has an additional tardiness of at least one T_p, the minimum processing time of the first job handled in F_{later}. The third job to be handled in F_{later} has an additional tardiness of at least $2*T_p$, the sum of minimum job process times of the first and the second job handled in F_{later}. Likewise, the last job to be handled in F_{later} will have an additional tardiness of at least $(|\ F_{later}\ |-1)*T_p$, the sum of the job process times for the previous $(|\ F_{later}\ |-1)$ jobs in F_{later}. To sum up these minimum tardiness, we have $(1 + 2 + \ldots + (|\ F_{later}\ |-1))*T_p$.

Jobs in $F_{earlier}$ are the ones for which $C_i + m_{ik} + T_p \le d_k$. However, tardiness for a job k in this set will happen if a_k is after $C_i + m_{ik}$ and its completion time $a_k + T_p > d_k$. Note that $a_k + T_p$ is a very conservative estimation of the job completion time since the actual job service time may be longer than T_p if reshuffling of containers is required. The minimum tardiness for such a job is $\max(\ 0, \max(\ C_i + m_{ik}, a_k) + T_p - d_k)$.

3.2 Prioritized Recursive Backtracking with Heuristics: MT-RBA*

A* search has actually a best-first search feature and thus may take a long time to obtain a solution (i.e., not anytime) in the worst case or near-worst case. It is also memory intensive. We therefore propose a

Recursive Backtracking (RB) based algorithm. RB is complete and optimal if the depth of a search tree is finite and there is no time constraint. It greatly reduces memory usage by keeping only the nodes on current path during search. However, it needs to traverse the entire tree to find the optimal solution, which is time-consuming. We therefore introduce the evaluation function $f(i) = g(i) + h(i)$ which is explained in section 3.1 to trim the search space. The algorithm MT-RBA* will not miss the optimal solution and will greatly reduce the computation time of RB. At any time when expanding the current partial job sequence, if $f(i) = g(i) + h(i)$ is not smaller than the total tardiness of the current best solution, the search path will be pruned. Therefore, discovery of a good path which has near-optimal total tardiness at the early stage of the tree search is crucial to the performance of MT-RBA*. For this purpose, we proposed a technique called prioritized search order which is more likely to discover a good dispatching sequence early in the planning process, instead of choosing the next job randomly. The prioritized search order we use is the ascending order of the job deadlines. Intuitively, if the YC serves a job with an earlier deadline first, it will contribute to the minimization of total job tardiness.

4 PERFORMANCE EVALUATION

4.1 Design of Experiments

To evaluate the performance of the proposed YC dispatching algorithms, YC planning experiments were carried out. Parameter settings in the experiments were obtained from real world terminal models as in past projects (Guo *et al*. 2007). The linear gantry speed of an YC is 7.8km/hour. A yard block has a size of 36 slots. We conducted experiments where a YC is in charge of one yard block. A planning window of 10 jobs for a YC is simulated.

Vehicle jobs arrive at the yard block at four different arrival rates. We assume that the vehicle inter-arrival times follow exponential distributions. Mean inter-arrival time is set to 180, 240, 300 and 360 seconds respectively. The slot and row numbers of the jobs are generated randomly within the zone. Other recent studies using randomized container locations include, for example, Zeng and Yang (2009). The tier number of a job is generated according to Table 1.

Table 1: Mixture of jobs

Percentage	Job type	Tier number of job	Percentage
40%	Vessel loading	Top tier	50%
		2nd top tier	30%
		3rd top tier	20%
40%	Vessel unloading	Top tier	100%
10%	External truck loading	Tier 1, 2, 3, 4	Equal probability
10%	External truck unloading	Top tier	100%

The deadlines depend on the type of jobs: relatively tight deadlines for loading jobs and less tight deadlines for unloading jobs. This reflects the fact that it is more important to finish loading jobs on time than unloading jobs. Following the design of Chu (1992), deadlines are generated from a uniform distribution. Let T = a vehicle's arrival time + 2 minutes. 2 minutes is the YC time to load/unload one container from/to the top of a stack, not including the crane gantry time. In other words, T_p = 2 minutes. For a vessel loading job, its deadline is a random variable from a uniform distribution with a width of 360 seconds around T such that 95% of the time its deadline is after T. Since vehicle arrivals at the yard side may be late in special cases like traffic congestions or machine break downs, 5% jobs will have deadlines earlier than T. For a vessel unloading job, the uniform distribution has a range of 720 seconds around T. 5% jobs will have deadlines earlier than T. For external vehicle jobs, the deadline is 30 minutes after the

truck's arrival. This follows the practice of some terminals where they guarantee that external trucks will finish their loading/unloading jobs within half an hour after their arrivals to the terminal.

The types of jobs that arrive at the yard block are a mixture of vessel loading/unloading jobs and external truck loading/unloading jobs. We tested a mixture of jobs as shown in Table 1. The percentage of each type of jobs is set which is similar to the scenarios in a terminal with high volumes for transhipment containers. When the tier number of a job is not the top tier, reshuffling may be needed depending on the job sequence as discussed in Section 3.3.

The algorithms MTA* and MT-RBA* are evaluated against the greedy heuristics Earliest Due Date first (EDD) and Smallest Completion Time Job First (SCJF). EDD and SCJF could find a solution fast but the optimal solution is not guaranteed. EDD generates job sequences in increasing order of jobs' deadlines. SCJF is a heuristic algorithm by Ng (2005). For each experimental setting, 30 independent runs were performed.

4.2 Results and Discussions

Table 2 shows the average job tardiness of the four algorithms tested for the four average inter-arrival time scenarios. Since MTA* and MT-RBA* compute the optimal job sequence for minimizing job tardiness, their results are the same even though the resulting job sequences may be different. As the average job inter-arrival time increases, average job tardiness decreases. However, the results confirm that our optimization algorithms make significant improvements from the greedy heuristics in all tested scenarios.

Table 2: Average job tardiness (seconds)

	IAT 180	IAT 240	IAT 300	IAT 360
MTA*	32.15	26.75	23.08	19.67
MT-RBA*	32.15	26.75	23.08	19.67
EDD	59.07	47.55	44.25	38.41
SCJF	74.96	58.04	47.94	36.90

Table 3: Computational time (seconds)

	IAT 180	IAT 240	IAT 300	IAT 360
MTA*	462.0	256.0	153.0	131.6
MT-RBA*	3.7	2.8	2.0	2.3
EDD	0.1	0.1	0.1	0.1
SCJF	0.1	0.1	0.1	0.1

Table 3 shows the average computational times taken by the four algorithms tested for the four average inter-arrival time scenarios. As expected, the greedy heuristics EDD and SCJF takes little computational time, much faster than MTA* and MT-RBA*. MTA* is more time consuming than MT-RBA*. Although MT-RBA* explores more nodes in the search tree than MTA* method because MTA* is optimally effective, it outperforms the MTA* method because it does not have the overhead to maintain the queue for deciding which node to visit next. The prioritized search order in MT-RBA* also helps to get a good solution at the beginning of the search which effectively prunes many branches of the search tree at early stage, reducing computational time.

For both MTA* and MT-RBA*, the computation for heavy workload scenario (average inter-arrival time of 180 seconds) is much higher than the other scenario. With our mixture of the different job types as shown in Table 1, the average YC service time is 166 seconds without including inter job gantry time. So this represents the situation where the YC is almost 100% busy. In such a situation, jobs have smaller in-

ter-arrival time differences and the optimal sequence is harder to find. This can be explained using queuing theory: the closer the job arrival rate is to the process rate (YC handling rate), the higher probability there is to see queuing jobs. When several vehicle jobs wait for the YC service, their relative job locations would be an important factor in determining the optimal service order. The algorithms may need to expand and evaluate more alternative job sequences in finding the optimal, resulting in longer computational times. However, MT-RBA* only takes a few seconds to find the optimal solution.

Our admissible heuristic *h(i)* designed to estimate the lower bound of the total tardiness of jobs not yet included in the job sequence has no knowledge of the actual YC process time for a job. However the results in Table 3 show that *h(i)* is effective to reduce search time for an optimal solution. The results in Table 3 also show that it is feasible to embed simulation to obtain the sequence dependent YC process times in the computation of the total tardiness of alternative job sequences in the optimal YC dispatching algorithms. For the tested scenarios, the total computational time which includes the time for the embedded simulation is not excessive and is in fact quite reasonable.

5 CONCLUSION

We propose to compute job sequences for YCs to minimize vehicle job tardiness instead of minimizing vehicle waiting time. This helps vehicles to support QC operations with minimal delays in the loading and unloading process and therefore reduce vessel turnaround time. We present a modified A* search algorithm MTA* with an admissible heuristic to compute optimal YC dispatching solutions to minimize vehicle job tardiness in the storage yard. To overcome the large memory usage limitation of the MTA* search, we further present MT-RBA* algorithm that combine the advantages of A*search and Backtracking with prioritized search order.

In real operations, containers involved in a YC job may not be on top of the stack. To handle such jobs, some containers have to be moved first. Such scenarios are very often ignored in some other studies. In order to consider such jobs in the planning of YC dispatching sequence, simulation is embedded in our optimization algorithms to help provide accurate YC service times. This results in a more accurate evaluation of job tardiness. Experiments were carried out to evaluate the algorithms under four levels of job arrival rates. Our results show that the proposed algorithms consistently perform very well in all tested cases, significantly reducing the vehicle job tardiness. Our planning window of 10 jobs represents 30 minutes (IAT = 180 seconds) to one hour (IAT = 360 seconds) terminal operation time. The computational time used in planning by MT-RBA* is only a few seconds.

Future work includes more comprehensive evaluation of our algorithms. For example, a different distribution of the job deadlines may be used. How the computational time will increase when a longer planning window has to be handled should also be investigated. In fact, the time spent in the embedded simulation can be reduced if the yard block status can be saved for partial sequences. In this way, we do not need to start simulation from the first job in the sequence each time. However, this will significantly increase the memory usage of the algorithms. Therefore whether the saving in time will out-weigh the higher costs in memory has to be evaluated.

ACKNOWLEDGMENTS

This project is funded by the NOL Fellowship program.

REFERENCES

Cao, Z., D. H. Lee, and Q. Meng. 2008. Deployment strategies of double-rail-mounted gantry crane systems for loading outbound containers in container terminals*, International Journal of Production Economics*, 115, 221–228.

Guo, X, S. Y. Huang, W. J. Hsu, M. Y. H. Low, T. H. Chan, and J.H. Liu. 2007. Vehicle Dispatching with real time location information in container terminals, In *Proceedings of the European Modeling and Simulation Symposium.*

Guo, X., S. Y. Huang, W. J. Hsu, M. Y. H. Low. 2011. Dynamic Yard Crane Dispatching in Container Terminals with Predicted Vehicle Arrival Information, *Advanced Engineering Informatics*, 25(3), 472-484.

Jung, S. H., and K. H. Kim. 2006. Load scheduling for multiple quay cranes in port container terminals, *Journal of Intelligent Manufacturing*, 17, 479–492.

Kim, K. H., J. S. Kang, and K. R. Ryu. 2004. A beam search algorithm for the load sequencing of outbound containers in port container terminals, *OR Spectrum*, 26, 93–116.

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

Kim, K. Y., and K. H. Kim. 2003. Heuristic algorithms for routing yard-side equipment for minimizing loading times in container terminals, *Naval Research Logistics*, 50, 498–514.

Lee, L. H., E. P. Chew, K. C. Tan, and Y.B. Han. 2006. An optimization model for storage yard management in transshipment hubs, *OR Spectrum*, 28, 539-561.

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

Narasimhan A. and U.S. Palekar. 2002. Analysis and Algorithm for the Transtainer Routing Problem in Container Port Operation, *Transportation Science* 36(1), 63–78.

Ng, W. C. 2005. Crane scheduling in container yards with intercrane interference, *European Journal of Operational Research*, 164, 64–78.

Ng, W. C. and K. L. Mak. 2005. An effective heuristic for scheduling a yard crane to handle jobs with different ready times, *Engineering Optimization*, 37(8), 867-877.

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

Zeng, Q. and Z. Yang. 2009. Integrating simulation and optimization to schedule loading operations in container terminals, *Computers & Operations Research*, 36(6), 1935–1944.

AUTHOR BIOGRAPHIES

SHELLYING HUANG is a senior lecturer in School of Computer Engineering at Nanyang Technological University (NTU), Singapore. Her research interests are in intelligent decision support systems, simulation optimization, heuristics and logistic systems. Her email address is ASSYHUANG@ntu.edu.sg.

XI GUO obtained her Ph.D. from School of Computer Engineering, NTU, Singapore. Her research interests include real time control, artificial intelligence, and efficiency enhancement techniques for simulation-based optimization. She is now with Murex (Singapore). Her email address is guox0006@ntu.edu.sg.

WEN JING HSU is an associate professor in School of Computer Engineering at NTU, Singapore. His research interests include parallel and distributed processing and Provable efficient algorithms. His email address is hsu@ntu.edu.sg.

WEI LIN LIM is a final year undergraduate student in School of Computer Engineering at NTU, Singapore. Her email address is LIMW0129@e.ntu.edu.sg.