Dynamic space and time partitioning for yard crane workload management in container terminals

Guo, Xi; Huang, Shell Ying

2012

Guo, X., & Huang, S. Y. (2012). Dynamic Space and Time Partitioning for Yard Crane Workload Management in Container Terminals. *Transportation Science*, 46(1), 134-148.

<https://hdl.handle.net/10356/96004>

<https://doi.org/10.1287/trsc.1110.0383>

© 2012 INFORMS. This is the author created version of a work that has been peer reviewed and accepted for publication by Transportation science, INFORMS. It incorporates referee' s comments but changes resulting from the publishing process, such as copyediting, structural formatting, may not be reflected in this document. The published version is available at: [<http://dx.doi.org/10.1287/trsc.1110.0383>].

Downloaded on 26 Apr 2023 16:58:13 SGT

Dynamic Space and Time Partitioning for Yard Crane Workload Management in Container Terminals

Xi Guo

School of Computer Engineering
Nanyang Technological University
Block N4, Nanyang Avenue, Singapore
639798

guox0006@ntu.edu.sg

Shell Ying Huang

School of Computer Engineering
Nanyang Technological University
Block N4, Nanyang Avenue, Singapore
639798

assyhuang@ntu.edu.sg

ABSTRACT

We propose a new hierarchical scheme for yard crane (YC) workload management in container terminals. We also propose time partitioning algorithm and space partition algorithm for deploying YCs to handle the changing job arrival patterns in a row of yard blocks. The main differences between our approach and most of the methods in literature are: (1) Average vehicle job waiting time instead of the number of jobs is used to balance YC workload and to evaluate the quality of a partition; (2) YC working zone assignment is not in units of yard blocks and our space partition algorithm generates more flexible divisions of the workload from all blocks; (3) YC deployment frequency is not fixed but is decided by our time partition algorithm with the objective of minimizing average vehicle waiting times. The scheme combines simulation and optimization to achieve our objective for a row of yard blocks. Experimental results show that the proposed binary partition algorithm TP2 makes substantial improvements in job waiting times over the basic partition scheme and another existing algorithm (Ng, 2005) in all tested job arrival scenarios.

Keywords

Yard crane dispatching, Simulation and Optimization, Decision-support, Container terminals

1. INTRODUCTION

Today, about 80 to 90% of global trade by volume is shipped in containers. As of July 2010, there exist 4815 container vessels with a capacity equal to 13.8 million TEUs (AXS-Alphaliner 2010). Container ports, which serve

Figure 1: Layout & container flows in a typical terminal

as hubs of container transshipment, are crucial nodes and play an important role in the marine transportation network. The demand on high quality services from container terminals includes efficiency and reliability in container handling which in turn requires the port to utilize its resources efficiently.

Container terminals are open systems of material flow. Figure 1 shows part of a typical layout of a container terminal. In conventional terminals, the storage yard is often divided into several tens of yard blocks in a number of rows in parallel to the quay. Each yard block may have more than 30 slots (yard bays) of containers stored in length (corresponding to the container bays on vessels). Vehicles travel along lanes to transfer containers between quay side and yard side as shown by the dotted lines in Figure 1. Each dotted line may represent a single traveling lane or multiple traveling lanes depending on the traffic volumes of the terminal. When a vessel is unloading, vehicles carry containers from the vessel to different slot locations in the yard to store them. When a vessel is loading, vehicles carry containers from different slot locations in the yard to the vessel. When multiple vessels are loading and unloading, vehicles may arrive at different slot locations at one side of a yard block for storing and retrieving containers. Trucks may also arrive through terminal gates to unload export containers or to load import containers. Both the Yard Crane (YC) and the vehicle must be at the same slot position for the loading or unloading of containers. As a result, YCs need to move across different slot locations in serving vehicle jobs with a mixture of operation types. When YCs are busy serving other vehicle(s), a vehicle needs to wait at the yard for YC services.

The time an YC moves from one job location to another is referred to as gantry time. YC gantry times also contribute to vehicle waiting times and may become a significant part of an YC's busy time.

One most important performance target of container terminal operations is to minimize vessel turn-around time. It means the YCs need to serve jobs from in-terminal vehicles as efficiently as possible to reduce the delay of vehicles at the yard side in order that vehicles continuously feed the quay side to support Quay Crane (QC) operations. The main objective at the yard side is then translated to minimizing the average vehicle job waiting time for YCs service.

YC deployment affects the efficiency of YC services. It is a very complex problem because the vehicle arrival pattern is uneven and changes dynamically over time. To cater for this, YCs need to move not only within a yard block, but also from one block to another. The movements of a Rubber Tired Gantry Crane (RTG) include: intra-block linear gantry, inter-block linear gantry and inter-block cross gantry. Intra-block linear gantries are linear movements along the lane within a yard block. Inter-block linear gantries are movements from one block to another in the same row with a similar speed to intra-block gantry, e.g. from BLK5 to BLK4 in Figure 1. Inter-block cross gantries are movements from one block to another in a different row, e.g. from BLK 2 to BLK 6 in Figure 1. An YC doing cross gantry has to make two 90° turns which take much longer time than linear gantry and may delay the vehicle movements by blocking the vehicle traveling lanes. After equipment ordering at the beginning of a shift, a common practice is to initially assign YCs to various yard blocks to work. Then a re-distribution of YCs among the yard blocks is done from time to time to match the dynamically changing job arrival pattern. Since the number of YCs is often not the same as the number of yard blocks, partitioning in units of blocks may lead to one of the following scenarios: (1) some blocks do not have any YCs in charge; (2) some YC is in charge of two or three blocks; (3) some blocks have more than one YC. In the first situation it will result in long vehicle waiting times. The third scenario needs carefully synchronized YC operations to avoid YC clashes and may result in long YC waiting time. In all scenarios, the workload partitioning usually cannot achieve the best balance among YCs, which results in low productivity of YCs and long waiting time of vehicles at the yard side.

The successful deployment of multiple YCs highly depends on the accuracy of the future workload estimation. In real world applications, workload is commonly estimated by the number of jobs expected in a future time interval or a quantity proportional to the number of jobs (e.g. Zhang et al. 2002; Cheung et al. 2002). Therefore various YC workload partition methods usually try to minimize the difference in the number of jobs assigned to YCs among the

blocks. However, the estimations based on the number of jobs may widely deviate from the true “workload” as (1) For the same number of jobs, an YC will incur different gantry times due to the different job locations; (2) For the same number of jobs and the same job locations, different job arrival times may result in different job serving sequences and therefore different gantry times; (3) For the same number of jobs, the same job locations and the same job arrival times, different YC dispatching strategies may result in different job serving sequences and thus different gantry times. It can be seen that even when the average gantry times are involved in workload estimation, for example, workload is estimated by (average YC service time per job + average gantry time between two jobs) multiplied by the number of jobs, it is still not a good estimation. As we pointed out earlier, YC gantry time may be a significant part of its busy time. Different serving sequences may lead to different YC gantry times which result in different job waiting times. Therefore the number of jobs is a very coarse estimation of YC’s workload.

We propose to use estimated average job waiting time as metrics for workload partitioning and YC dispatching. The best partitioning and dispatching plans should assign working zones to YCs such that the average vehicle job waiting time calculated from all the jobs in the row in the planning window is minimized. It follows that the difference in the number of jobs done by YCs in a planning window is not an important concern. The yard operations would therefore, minimize the vehicle delays at the yard side. This will achieve better vehicle support to the quay side than using the number of jobs for the overall terminal objective of minimizing vessel turnaround time.

Figure 2: Two YCs need to be at least 8 slots apart to avoid blocking between 2 trucks.

We also propose not to assign working zones to YCs in units of yard blocks. Yard blocks in different rows are still in the charge of different YCs. Within a row of yard blocks any slot position may be a point of division of working zones. Ng (2005) also divides work among YCs at any point but we have an additional constraint that within a planning window, working zones have to be separated by a region at least 8-slot long, with no jobs in it. This is to guarantee that RTGs (a common type of YC) working in the same block are always separated by such a

distance. This is to observe the safety constraint to avoid collisions between two neighboring cranes. It is also to ensure that the two streams of vehicles to be served by the two YCs will not block each other. As shown in Figure 2, Truck 1 and Truck 2 are attempting to transfer containers to/from stacks in a block that lie beneath YCs 1 and 2 respectively. The trucks, each 55 feet long, are assumed to be mobile and the YCs are assumed to be stationary. If YCs 1 and 2 are less than 8 slots apart, Truck 2 does not have enough space to pull into the handling lane and to park parallel to the storage block beneath YC2. If Truck 2 is already in its position under YC2, Truck 1 will not have enough space to pull out of the handling lane and into the bypass lane without hitting Truck 2. The gap between two blocks is usually less than 8 slots but this region may also be a point of division of working zones. This is because vehicles at the two opposite ends of two neighboring blocks will not block each other.

We propose a hierarchical scheme for YC operation management which is organized into three levels as shown in Figure 3. Suppose a suitable number of YCs has been assigned to work for the current shift. **Level 1** distributes YCs among different rows at suitable times based on predicted future workload. This is done a few times during a shift of 8 hours. **Level 2** dispatches YCs to work in various non-overlapping working zones in each row for the time window in between two rounds of YC re-distributions at Level 1. **Level 3** determines the serving sequences of vehicle jobs for an YC in a working zone over a period of time (e.g. a sub-planning widow). The hierarchical scheme aims to (1) reduce the time consuming YC cross gantries between different rows of yard blocks; (2) minimize the overall average vehicle job waiting time through flexible re-distributions of YCs to cope with dynamically changing job arrival patterns over time in the yard.

Level 1	Distribution of YCs to different Rows	
Level 2	Dispatching YCs in a row	
	Time partition	Space partition
Level 3	YC dispatching in individual zones	

Figure 3: Hierarchical YC operation management scheme.

A very important task in the flexible re-distribution of YCs is done in Level 2 which is the focus of this paper. We assume that Level 1 will allocate a number of YCs to each row in proportion to the number of jobs for the planning window of Level 1. Then YCs are dispatched to work in various zones in each row for the planning

window in Level 2. It is done by first finding a proper time partition of the planning window into a number of sub-windows of variable lengths, and then a proper space partition of the row of yard blocks in each of the sub-planning windows. The time partition is necessary because sometimes the space partition into a number of zones does not have a solution. The reason is sometimes there is a large area with high workload so there are no partition points which are areas of at least 8-slots separation between adjacent jobs to carry out the partition. Or there are not enough partition points and working zones to divide among the YCs. By dividing the window into smaller sub-windows, the number of partition points in the sub-windows will increase. There are also times where the space partition would not be able to find a good solution. It is due to the fact that job arrival pattern is changing by a large margin within the planning window for Level 2. Our time partition algorithm aims to divide the planning window for Level 2 to enable dynamic space partitioning. This will cope better with the changing job arrival pattern than having the same space partition for the whole planning window. Two time partitioning algorithms are proposed by employing different approaches.

Instead of doing partition based on the number of jobs, our space partition algorithm considers a pool of candidate partition solutions using a combination of simulation and optimization. The candidate space partition solutions are evaluated with the support of Level 3 procedure. Level 3 generates the optimal YC dispatching sequence for each working zone of a candidate partition solution. It involves simulating YC operations following various possible dispatching sequences and uses a heuristic to reduce the computational time. The dispatching sequence with the smallest predicted vehicle waiting time will be the optimal sequence for one working zone of a candidate partition solution. The predicted vehicle waiting times in the working zones of a candidate partition solution would collectively provide the predicted total vehicle waiting time for all jobs in a row under that partition. Among all candidates, the partition plan with the smallest average job waiting time will be selected as the decision for space partition. In handling such inter-related complex problems, the efficiency of the lower level support is essential. Here the single YC dispatching algorithm at Level 3 may need to be called hundreds or even thousands of times to support decision-making at Level 2. A modified A* search algorithm (Guo and Huang 2008, Guo et al. 2009, Guo et al. 2011) is employed to find the optimal dispatching solution for an YC in a zone over a period of time at Level 3. The modified A* algorithm is able to find the optimal solution from over 2.4×10^{18} possible dispatching sequences within three seconds.

In the rest of the paper, we discuss the related work in Section 2 and present the framework of our system to combine simulation and optimization in Section 3. The proposed algorithms for time and space partition are described in Section 4. The algorithms are evaluated by simulation experiments against the traditional block-based partition approach and the approach in Ng (2005) in Section 5. Section 6 concludes the paper.

2. RELATED WORK

The problems of scheduling and dispatching resources in container terminals have been widely studied in recent years (Vis and de Koster, 2003; Steenken et al., 2004; and Stahlbock and Voss, 2008). On the topic of the YC management, two main problems are: (i) deciding job servicing sequence for an YC which we refer to as the YC dispatching problem in this paper; (ii) allocating YCs to different parts of the yard which we refer to as the YC deployment problem in this paper.

Kim and Kim (1999) studied the YC dispatching problem considering 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) and Kim et al. (2004) extended the study of this problem by comparing exact optimization, a beam search heuristic and a Genetic Algorithm (GA). 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. In addition, the assumption of having dedicated yard cranes just to support vessel loading operations in these works is not always the best for terminals with many berths and more yard blocks than cranes.

Several works studied the dispatching problem with 2 YCs. Due to the problem complexity, commonly MIP models were employed just to formulate the problems while heuristic methods were proposed to find near-optimal solutions. 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. Solutions with possible YC interference were considered not feasible in the MIP model. They assume that the QC schedule, which shows the sequence of ship-bays that each QC should perform, is given. The dispatching process attempts to determine the sequence of yard-bays for each YC to visit and the number of containers to be picked up at each yard-bay. Lee et al. (2007) 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.

Published works focusing on inter-block level YCs deployment are not abundant. Zhang et al. (2002) described the YCs deployment problem among blocks with forecasted workload per block per planning horizon (4hrs). The problem is formulated as a MIP model with the objective to minimize total unfinished workload at the end of each planning period and is solved by a modified Lagrangian relaxation method. The deployment process tries to find the optimal movements of YCs among blocks. However, only one transfer of one YC in and out of a block is allowed in the 4 hour planning period. This restricted YC transfers may not be enough to match the changing workload distribution. The forecasted workload is based on the number of container moves (equivalent to the number of YC jobs) only, which is not a good metric to partition workload as discussed in Section 1.

Cheung et al. (2002) also studied the problem of scheduling YC inter-block movements with the same objective. A MIP model is proposed with a Lagrangian decomposition solution and a successive piecewise-linear approximation approach is also proposed for large-sized problems. Experiments show that near-optimal solutions could be found in reasonable time. However, dispatching sequence of each YC in its block is not solved. To calculate the amount of unfinished work in a block, they assume the amount of work done is proportional to the number of YCs in a block. Intuitively this should be true, that is, work done $w = a \times m$ where m is the number of YCs in a block. However many factors including job arrival times, job locations and job serving sequences affect YC gantry times between two jobs and therefore YC productivity. In addition, some job arrival patterns may not allow multiple YCs to work fully in parallel, for examples, a number of jobs at the same slot location or many job arrivals near the end of a planning window. Having more YCs in these cases does not reduce the amount of unfinished work proportionally. So we cannot use the same value of a to apply to all yard blocks.

Ng (2005) considered the problem of multiple YCs sharing a single bi-directional travel lane and modeled it as an integer program with the objective to minimize total job completion time. A two-phase algorithm is proposed where the first phase used dynamic programming approach to compute the best workload partition. The workload for each YC was roughly estimated by a simple greedy heuristic. The greedy heuristic works as follows: Among all the jobs not yet scheduled, compute the completion time for each of these jobs and select the job with the earliest job completion time as the one to be handled next. So the workload of the YC is the total working time following this job service order. This gives a much more accurate estimate of the YC's workload than the simplistic way by the

number of jobs. Based on the best solution found among non-overlapping partitions, the second phase swaps jobs between neighboring YCs to further improve the performance. Experiments show that the performance of the algorithm is on average 7.3% above a generated lower bound. However, safety constraint of separating two YCs by a minimal distance was not considered. This may lead to YC clashes. It may also lead to the streams of vehicles to be served by two neighboring YCs (RTGs) blocking each other when the two YCs are working side by side. As a result, the performance would downgrade from what the original paper expects as shown in our experimental results in Section 5.

Petering et al. (2009) presented a discrete event simulation study to show that a terminal's average quay crane rate depends on the performance of the YC dispatching system in the storage yard in the long run. Their work discussed the deadlock prevention problem when YCs are in charge of overlapping zones and claimed that YCs could only schedule for at most 1.5 container jobs to avoid deadlocks in a real-time setting. Therefore, only rule-based algorithms (e.g. First Come First Serve) were implemented in their experiments. This reinforces our belief that non-overlapping YC zones should be formed in our YC dispatching schemes to achieve high performance while maintaining safety constraints and avoiding deadlocks in real-time settings. Results in Petering et al. (2009) show that vehicles spend roughly 60% of their time waiting for quay cranes or yard cranes but yard operations are primarily responsible for the delays. Their results also show that the average quay crane rate has a strong negative correlation with the total waiting time of the vehicles. These findings strengthened our belief that we should use average vehicle job waiting time as the key indicator of YC workload and the objective of YC deployment is to minimize the delay of vehicles.

Other studies have extended the research to a broader scope by including several related sub-problems in terminals. Handling equipments of YCs, QCs and in-terminal vehicles were all considered in Chen et al. (2007), Lao and Zhao (2008), Zeng and Yang (2009). Froyland et al (2008) concerned the problem of operating a landside container exchange area including YC dispatching, short-term container stacking and allocation of vehicle delivery locations. Integrated models consider decision-makings of several types of resources or operations at the same time to achieve a global objective. Due to system complexities, models of subsystem are sometimes simplified to a certain degree, e.g. only rule-based dispatching like FAM(First Available Machine), LFM(Last Available Machine) and FCFS (First Come First Serve) were considered in scheduling (Zeng and Yang, 2009).

3. SIMULATION-BASED PARTITIONING-DISPATCHING SYSTEM

To support the hierarchical YC operation management, an integrated YC deployment scheme serves 3 purposes: (1) to find a time partition plan to split the entire planning time into several smaller planning windows (2) to find a space partition plan that partitions the jobs in one row into a number of zones so that each YC will be in charge of one non-overlapping zone in a planning window; (3) to determine an optimal YC dispatching sequence (i.e. job serving sequence) in each zone with minimized average job waiting time. The scheme covers Level 2 and Level 3 problems illustrated in Figure 3 in an integrated manner. It consists of two components (modules), namely the *optimization module* and the *simulation module* as shown in Figure 4.

Figure 4: Simulation and optimization for YC deployment

Figure 4 shows the sequence of the events in the decision making process for YC deployment. A request for Row level YC deployment is generated (first event), e.g. every 2 hour after YC re-distribution among rows of yard blocks at Level 1. Upon such a request, real time vehicle and YC locations and other relevant information will be collected from the terminal yard and passed into the simulation module and the optimization module (also included in the first event). The simulation module simulates the terminal operations from this time point and makes predictions of the vehicle arrivals to the row of yard blocks in the planning period. The predicted vehicle arrivals (arrival times and job locations) are passed to the optimization module (second event in Figure 4). It will then generate *possible* partition plans and pass them to the simulation module (third event). The simulation experiments are conducted to simulate YC operations for each zone in each of these partition plans. The optimization module collects from the simulation module the predicted total vehicle waiting times zone by zone for each of these partition

plans (fourth event). The predicted total waiting time of a partition plan of the row is the sum of the total vehicle waiting time for each of the zones. The optimization module will choose the partition plan that returns the shortest total waiting time as the one to be used for assigning working zones to YCs. The average waiting time for all the jobs in the row is the total waiting time divided by the number of jobs. So the partition plan with the shortest total waiting time is the plan with the minimum average waiting time.

The algorithms of the optimization module to do both the time partition and the space partition are described in the next section. The time partition algorithm will call upon the space partition module to get the best space partition for various time windows. The space partition algorithm will call upon the simulation model in the process of obtaining the best dispatching sequence for each of the zones in a space partition. So at the end of the workload partitioning process, we have both the partitioning plan for the YCs and the dispatching sequence for each zone (crane).

Since the objective of partitioning a row is to achieve efficient yard performance, we use the predicted average vehicle waiting time returned by the simulation experiments to evaluate the quality of a candidate partition plan. If the average job waiting time in a zone for one YC is long, it means the YC's load is heavier compared with another zone where a smaller average job waiting time is expected. Because the expected performance of an YC in a zone is derived from simulating YC operations based on real time expected job arrivals, it is a much more precise estimation of the true YC workload.

Note that the main focus of this paper is the YC workload partition algorithms and not the prediction of vehicle job arrival times. We do not include the simulation to predict vehicle arrival information in this work. Our partition algorithms presented in the next section assume the availability of such information and take them as input.

4. PARTITION PLANS GENERATION AND SELECTION

YC workload distribution at the row level involves two parts: the space partitioning and the time partitioning. The space partitioning aims to partition a row into a number of non-overlapping zones for individual YCs in a given planning window. The time partitioning aims to achieve dynamic workload partition by partitioning the time between two rounds of YC re-distribution at Level 1 into a number of sub-planning time windows. Thus, the space partition for different sub-planning time windows may be different. This is our solution to cope with the uneven and changing job arrival pattern while maintaining the constraint of separating neighboring YCs. Since the time partition

algorithm relies on the space partition algorithm to get the best space partition for a sub-time window in a tentative time partition, we will present our space partition algorithm first.

4.1 Space Partition

The optimization module receives from the simulation module the predictions of the vehicle arrivals to the row of yard blocks in the planning window (Figure 4). Let N be the total number of jobs expected to arrive. Let slot locations for jobs be L_i for $i = 1, 2, \dots, N$. Obviously each zone for one YC has to be a continuous sequence of slot locations. YCs are also kept separated by at least 8 slots in a block or by the inter block gap at all times. So the possible boundaries in the partition plan are areas where two neighboring job slot locations satisfy the condition $|L_i - L_{i+1}| \geq 8$ or L_i and L_{i+1} belong to two different blocks. Suppose that there are R such potential boundaries labeled by $P_1, P_2, \dots, P_R$ as shown in Figure 5 with P_1 and P_R at the two ends of the row. This means each area between two boundary points is unbreakable in partitions. $R-1$ such unbreakable areas, (P_i, P_{i+1}) for $i=1, 2, \dots, R-1$, exist as shown in Figure 4.

Figure 5: Possible boundary points between zones

Figure 6: Improved simulation and optimization for YC deployment

Since the arriving job locations could be predicted for the planning period by the simulation module, the total number of unbreakable areas $R-1$ is known at the beginning of the planning time. The number of YCs allocated to the row, denoted as M , should be no larger than the number of unbreakable areas, i.e. $M \leq R-1$. This ensures that one unbreakable area is covered by at most one YC to always satisfy the constraint of YC separation. In the event that M is greater than $R-1$, only $R-1$ YCs will be working for this specific planning window (set $M=R-1$). Other YCs will be idle since they cannot help in serving jobs without incurring the danger of potential collisions. The problem of choosing an YC workload partition plan in a planning window becomes a combinatorial problem. In other words, to allocate M yard cranes to cover the entire row, $(M-1)$ partition points are to be selected from $(R-2)$ candidates P_i , where $1 < i < R$. The total number of possible ways to partition the row into M zones is

$$\binom{R-2}{M-1} = \frac{(R-2)!}{(M-1)!(R-M-1)!}$$

To find the best way to partition the row into the M zones, we will need to simulate the YC operations under each such partition and predict their resulting vehicle waiting times. The problem could be solved naturally by dynamic programming (DP) approach. The DP approach will compute the performance of all candidate partition plans including those extremely unbalanced plans. There are two problems. Firstly, the extremely unbalanced plans are unlikely to be the optimal plans. For example, a problem of deploying 4 YCs for 40 jobs in a row may have a possible partition 2-1-2-35, in terms of the number of jobs. Intuitively YCs in the first three zones may result in long idle time while the YC in the last zone will be overloaded resulting in long vehicle waiting times. Secondly, since the problem of finding job serving sequence for a single YC in each zone is NP-Hard (Narasimhan and Palekar 2002), collecting performance from these extremely unbalanced plans would take a great portion of the entire simulation time.

We propose a space partition algorithm to avoid the unnecessary full evaluation of those extremely unbalanced plans while maintaining the guarantee that the best plan in the candidate pool is selected. Figure 6 illustrates the information flows when space partition is carried out (instead of Figure 4). Up to the point where the optimization module receives the predicted vehicle arrivals from the simulation module as inputs, Figures 4 and 6 are the same. The optimization module then generates candidate partition plans. A request for a quick evaluation of the plans will be sent to the simulation module for profiling their potentials. The quality of a partition plan is evaluated by a heuristic function based on a carefully controlled branch-and-bound method to return a performance Lower Bound

(LB) of the vehicle waiting time. This is essentially a branch-and-bound algorithm with the control of how many partial solutions are explored before the best bound value is returned as the LB of the plan. The LB is guaranteed never to overestimate the average job waiting time under a partition plan, no matter how small it is. After the quick evaluation, all candidate plans would form a priority queue in the order of decreasing likelihood to be the best partition plan. The plan with the smallest LB indicates a highest likelihood to be the best partition plan. Therefore it is the first plan in the priority queue. With this priority queue, full evaluations are conducted only for those promising plans to save simulation time while no optimal plan (of the candidate pool) will be missed. Pseudocode of the space partition algorithm is shown in Figure 7. `Get_LowerBounds()` in lines 10-21 is the branch-and-bound algorithm to get the LBs of candidate partition plans. `Get_Best_Partition()` in lines 23-35 is the algorithm to generate the best space partition plan.

As can be seen from Figure 7, function `Get_Best_Partition()` always gives priority to the un-explored plan that has the smallest LB of total job waiting time, that is, the top item in `planQ`. The search of the best plan will stop when the total job waiting time of the current plan is smaller than the smallest LB of all unexplored plans. Or, the search of the best plan stops when all plans have been evaluated. Each un-explored plan is evaluated by the “for” loop from line 30 to line 32. Line 31 is the step which involves simulation to get the best dispatching sequence for a zone in the evaluation of partition plans using the algorithm presented in Guo et al. (2011). The total vehicle waiting time from the best dispatching sequence for each zone will add up to the total vehicle waiting time of the partition plan.

The accuracy of the LB assessment affects the computational time distribution between Function `Get_LowerBounds()` and Function `Get_Best_Partition()`. When the LBs are rough, they cannot profile well the likelihood of plans to be the optimal one. Little time might be spent in Function `Get_LowerBounds()` while much longer time will be spent in Function `Get_Best_Partition()`. In the extreme case, if LBs for all plans are zero (not computed), no knowledge of the plan potentials is available and all plans need to be fully explored in `Get_Best_Partition()`. On the other hand, when more time is consumed in `Get_LowerBounds()` and the LBs are highly accurate, much time will be saved in `Get_Best_Partition()`. As optimality likelihoods are well profiled by the priority queue, only a small proportion of the partition plans in the priority queue needs to be explored by `Get_Best_Partition()` to confirm the optimal plan. In the extreme case, if LBs generated in `Get_LowerBounds()` are exactly the true performances of the plans, no time is needed in `Get_Best_Partition()`. It could be seen easily that in

the two extreme cases, one stage is exactly the full exploration of all partition plans and the other stage is doing nothing. The issue is how much `Get_LowerBounds()` should do to achieve substantial savings in computational time for the YC workload partitioning and job sequencing problem.

```

1  Function Space_Partition ()
2      Create a Priority Queue planQ
3      // Generate all candidate partition plans and get the LBs of Total JobWaitTime of each plan
4      Call Function Get_LowerBounds ()
5      // Now planQ keeps all plans in ascending order of their LBs
6      Call Function Get_Best_Partition ()
7      Output the plan currentBestPlan
8  END Function Space_Partition
9
10 Function Get_LowerBounds ()
11     FOR each candidate partition plan
12         planLB = 0 // LB of total job waiting time of the row
13         FOR each zone in this plan
14             IF LB of the zone stored in DMap (flag ==1)
15                 Retrieve LB from DMap
16             ELSE
17                 Compute LB of this zone
18                 Update the LB to DMap (flag =1)
19             planLB += zoneLB
20         Insert the plan to priority queue planQ
21 END Function Get_Lower_Bounds
22
23 Function Get_Best_Partition ()
24     create planBest, let planBest.TotalJobWaitTime = infinity.
25     WHILE planQ is not empty
26         Pop out the top plan x of planQ,
27         x.TotalJobWaitTime = 0
28         IF planBest.TotalJobWaitTime <= LB of plan x
29             STOP the searching
30         FOR each zone in the partition plan x
31             Find YC job sequence and get the total JobWaitTime for the zone (Guo et al., 2011)
32             x.TotalJobWaitTime = x.TotalJobWaitTime + total JobWaitTime of the zone
33         IF x.TotalJobWaitTime < planBest.TotalJobWaitTime
34             planBest = x
35 END Function Get_Best_Partition

```

Figure 7: Pseudocode of the Space_Partition algorithm

In `Get_LowerBounds()`, the LB of each partition plan is the sum of the LBs of its YC working zones. We use the same A* search method to compute the LB of each zone (line 17 in `Get_LowerBounds()`) as that in the generation of the complete YC job sequence which is Line 31 in `Get_Best_Partition()`. The details of the algorithm to generate the complete job sequence are in Guo et al. (2011). The main idea is to build/simulate YC job sequences incrementally always choosing the most promising job to expand next. Each partial job sequence is evaluated by a lower bound of

the total waiting times of the complete sequences which have the partial sequence as the prefix. The difference from the algorithm in Guo et al. (2011) is that A* search is used to find a lower bound of the optimal solution instead of generating the optimal solution. Only a portion of the A* search graph is explored, that is, only partial job sequences are built. The lowest estimate of the total waiting time among the partial sequences in the search graph is used as the LB for optimal job waiting time of a single YC zone. How much exploration is done (or how many partial sequences are generated) determines the accuracy of the LB of the zone and this is controlled. We exercise this control by exploring $\frac{I}{|Jobs|} \times 100$ percent of the estimated total number of nodes in the search graph:

$$\text{No. of nodes explored in LB} = \text{Estimated No. of nodes explored to find optimal solution: } N * \frac{I}{|Jobs|}$$

To estimate the number of nodes N to be explored to find the optimal solution, we use the concept of effective branching factor (EBF). EBF is usually used to characterize the quality of a heuristic search. If the total number of nodes explored to find the optimal solution for a particular problem is N, and the solution depth is d, then b* is the EBF that a uniform tree of depth d would have to contain N+1 nodes. Thus

$$N + 1 = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

Usually the EBF is fairly constant for sufficiently hard problems (Russell and Norvig, 2003). The EBF of a single YC dispatching problem for its assigned zone has been evaluated by Guo et al. (2009). It was confirmed that the EBF value does not change much for different problem sizes x. Thus, given a zone of x jobs, the solution depth d=x. The equation above is used to find N.

With this Space_Partition() algorithm, the best partition plan with the smallest overall total job waiting time among all candidate partition plans will always be found and selected. Candidate partition plans with large total job waiting times could be identified by high LBs. These plans will be at the end part of the priority queue for which full evaluation experiments are unlikely to be explored.

4.2 Time Partition

Space partition algorithm is able to find the best partition plan for the row of yard blocks within a planning window with static non-overlapping zones. However, if the workload partition task at Level 2 of our YC management scheme is solved by our space partition algorithm alone, there are three situations where it will not work or not work well:

- (1) Sometimes there are not enough partition points to carry out the space partition.
- (2) Consider the planning window for Level 2 as a whole, the distribution of the possible partition points does not allow a balanced partition.
- (3) The job arrival pattern may be changing significantly within the planning window for Level 2.

The approach we propose to handle these situations is to divide the planning window for Level 2 into a number of sub-windows. Consider the set of jobs in one planning window as shown in Figure 5 which makes conditions (1) and (2) above true. When we divide the planning window into sub-windows, in each sub-window there will be a smaller set of the jobs and thus more partition points for the space partition algorithm to consider. This means we need to find the most suitable way to divide the planning window for Level 2 into a number of sub-windows of variable lengths. The objective is to get the maximum benefit of different space partition for different sub-windows. We propose and evaluate two algorithms which use the binary division approach.

```

Function Time_Partition1 ( T )
    Breaking T into 2 planning windows of equal time length T1, T2
    TotalJWT_T=Space_Partition (T);
    TotalJWT_T1=Space_Partition (T1);
    TotalJWT_T2=Space_Partition (T2);
    IF (TotalJWT_T1 + TotalJWT_T2 < TotalJWT_T)
        Return Time_Partition1 (T1) + Time_Partition1 (T2)
    ELSE
        Breaking T into 2 planning windows of equal size (number of jobs) T3, T4
        TotalJWT_T3=Space_Partition (T3);
        TotalJWT_T4=Space_Partition (T4);
        IF (TotalJWT_T3 + TotalJWT_T4 < TotalJWT_T)
            Return Time_Partition1 (T3) + Time_Partition1 (T4)
        ELSE
            Return TotalJWT_T
    END Function Time_Partition1 (T)

```

Figure 8: Pseudocode of the Time_Partition algorithm1

Given a planning window T, two possible ways to break the window into 2 windows are to divide window T into 2 sub-windows of equal time length, or to divide window T into 2 sub-windows of equal number of jobs. Our first approach of time partition tries to use both these ways. The algorithm as shown in Figure 8 tries to divide

window T into 2 sub-windows of equal time length T1 and T2 first. If the total vehicle job waiting time is smaller by breaking the window into T1 and T2, consider T1 and T2 as current planning windows and try further, that is, call the algorithm recursively to find the time partitions for T1 and T2. If the total job waiting time does not improve, then try to break window T into 2 sub-windows of equal number of jobs T3 and T4. If the total job waiting time is smaller by breaking the window into T3 and T4, consider T3 and T4 as current planning windows and try further. If both two division methods do not improve the total job waiting time, keep the current planning window T. When evaluating the job waiting time of a planning window, the algorithm calls the space partition algorithm.

```

Function Time_Partition2 ( T )
    Breaking T into 2 planning windows of equal time T1, T2
    TotalJWT_T=Space_Partition (T);
    IF (Number of jobs in T1< Number of YCs) OR (Number of jobs in T2 < Number of YCs)
        Return TotalJWT_T
    ELSE
        TotalJWT_T1=Time_Partition2 (T1);           //recursively divide T1
        TotalJWT_T2=Time_Partition2 (T2);           // recursively divide T2
        IF (TotalJWT_T1 + TotalJWT_T2< TotalJWT_T)
            Return TotalJWT_T1 + TotalJWT_T2
        ELSE
            Return TotalJWT_T                        // no partition, keep window T
    END Function Time_Partition2 (T)

```

Figure 9: Pseudocode of the Time Partition algorithm2

The second time partition algorithm is shown in Figure 9. Given the current planning window T and let the two sub-windows of T with equal planning time be T1 and T2, the best window size(s) can be expressed as follows:

$$Time_Partition2(T) = \begin{cases} Space_Partition(T) & , \text{if } |T1| < M \text{ or } |T2| < M \\ \min\{Space_Partition(T), Time_Partition2(T1) + Time_Partition2(T2)\} & \end{cases}$$

where M is the number of YCs serving the row of yard blocks. The algorithm tries to divide T into 2 sub-windows, T1 and T2, of equal planning time. If any of the sub-windows has a number of jobs smaller than M , then T should not be divided. In this case, in T1 or T2, each YC may have less than one job on average and this would lead to idling YCs. When both T1 and T2 have at least M jobs, the best partition is either no partition (keep window T) or the best partition for T1 followed by the best partition for T2, whichever gives the lower total job waiting time. The

resulting time partition will be a series of planning time windows of variable time lengths which returns the smallest total job waiting time of all combinations.

With the help of time partition approaches, YC management could achieve dynamic workload balance among YCs. From the view of the original planning window in between 2 rounds of YC distribution at Level 1, YC workload partition is dynamic since space partition changes from one sub-planning time window to the next. Having small planning time sub-windows has the advantage of flexibility to match changing job arrival pattern. On the other hand, when the job arrival pattern permits, a large planning time sub-window provides more opportunities to optimize the job serving sequence than a small one. Our two time partition algorithms try to keep the planning window as large as possible but will return a sequence of smaller windows when the expected job waiting time is smaller working with the smaller windows.

5. PERFORMANCE EVALUATION

Evaluation of the proposed YC workload partition algorithms are done by comparing performance among five algorithms:

- Space Only Partition Algorithm (SOP)
- Time Partition Algorithm 1 (TP1)
- Time Partition Algorithm 2 (TP2)
- Ng's algorithm with matching constraints (Ng's)
- Basic Algorithm

The Basic algorithm is designed to represent the basic workload partition scheme. It selects the partition plan which allocates whole yard blocks to YCs while trying to minimize the difference in number of jobs among them. When the number of YCs is larger than the number of yard blocks in a row, yard blocks would be sorted in descending order by the number of jobs in each block. Extra YCs would be assigned to each block accordingly provided that there is a safety separation of 8 slots in the block. This scheme is quite commonly practiced in terminals and is employed as a benchmark scheme.

Ng's algorithm (Ng, 2005) is discussed in Section 2. As said there, it has a more accurate estimate of workload by a heuristic than by the number of jobs. Their experiments have shown that the performance of the algorithm is on average 7.3% above a generated lower bound. However, safety constraint of separating two YCs by a minimum

distance is not considered by them and this may degrade the performance in real operations. We present our implementation of Ng's algorithm in Section 5.1.

Comparison of the CPU times used by Ng's algorithm, SOP, TP1 and TP2 is also presented to show that our time and space partition approach is efficient enough for practical problems.

5.1 Ng's Work with Matching Constraints

Ng's algorithm (Ng 2005) is implemented with the additional constraint that two YCs are separated by at least 8 slots or by an inter block gap at all times. The original version allows two YCs to work together in two slots next to each other at the same time. The algorithm involves two phases. In phase 1, the static non-overlapping partition with estimated smallest total job waiting time is selected using dynamic programming approach. The estimation of job waiting times is done by applying the greedy first-complete-first-serve algorithm to find job sequences for individual YCs. In phase 2, neighboring YCs try to pass jobs to each other if it improves YC performance from the partition in phase 1. The main logic of phase 2 is shown in pseudocode in Figure 10.

Total job completion time (TCT) = total job waiting time (TJWT) + total job arrival time + total job serving time

The objective of minimizing TCT is equivalent to minimizing overall total job waiting time (TotalJWT) since total job arrival time, total job servicing time and the total number of jobs are known.

```

Function Ng_Phase2 ( )
TotalJWT_Best is collected from Ng_Phase1
  FOR each pair of neighboring YCs in the Row
    FOR each YC in the pair
      FOR each job in the job set of the YC in ascending order of arrival times
        Assign the job to the other YC
        Determine the service sequence for current YC and the other YC
        IF TotalJWT improves from TotalJWT_Best And No YC Interference occur
          Keep the assignment of job and update TotalJWT_Best
        ELSE
          Cancel the assignment of job
END Function Ng_Phase2 ( )

```

Figure 10: Pseudocode of the Ng's algorithm – phase 2

5.2 Experimental Design

Experiments using parameter settings like terminal layout, YC gantry speed, vehicle arrival patterns and YC handling rate from real world terminal models were carried out. The linear gantry speed of the YC is 7.8km/hour. As reshuffling operation of containers in the yard is commonly done in the lull periods of yard operations, we assume containers to be retrieved are already on the top of the slots and containers to be stored in yard will be placed on top of their slot locations. The YC process time is thus assumed to be 120s for each container job. The usage of constant YC process time is highly consistent with many existing works such as Lee et al. (2007), Jung and Kim (2006) and Lee et al. (2006). The simulation model is programmed using C++ language under Visual C++ 6.0 compiler on Pentium Core2 Quad CPU Q9450 and 3GB RAM.

In our experiments, there are 5 yard blocks in a row. Each yard block has 37 yard slots. Inter-block distances between two neighboring yard blocks in a row are about 18 meters which are about 3 yard slots in length. We take into consideration that YCs may need to slow down to wait for vehicles to stop before crossing from one block to next. So inter-block gantry time between two neighboring blocks in a row is set to be equivalent to YC gantry time of 10 yard slots. We tested scenarios where 3, 5 or 7 YCs are allocated to the row respectively for the planning period of one hour.

Table 1: Means of job inter arrival times under different scenarios sets (seconds)

sec	Set A Row	Set B		Set C			Set D							
		T1	T2	Blk 1	Blk 2,3	Blk 4,5	T1				T2			
		Row	Row				Blk 1	Blk 2,3	Blk 4	Blk 5	Blk 1	Blk 2,3	Blk 4	Blk 5
3YC	150	100	300	400	600	400	171	150	171	300	286	500	286	1000
5YC	90	60	180	240	360	240	239	187.5	239	375	471	825	471	1650
7YC	65	45	135	180	270	180	400	300	400	600	857	1500	857	3000

It is not the focus of this paper to study how to predict vehicle arrivals by real time data driven simulation or other methods. So we generate vehicle arrival times. The inter-arrival times follow an exponential distribution density function for a mixture of storing/retrieval jobs for multiple vessels. The locations of job arrivals are uniformly distributed. Workload of the entire row is adjusted by changing the mean inter-arrival time so that the

numbers of job arrivals are proportional to the number of YCs in the row. It reflects the fact that Level 1 of YC operation management will assign a suitable number of YCs to a row in proportion to the expected workload.

As shown in Table 1, four job arrival patterns are tested in four workload scenarios. In workload scenario Set A, the same job arrival rate applies across the entire row for the entire planning window. This represents a relatively static and uniform job arrival pattern across a row. In real world applications, job arrival pattern is likely to be more dynamic and less uniform. Workload scenarios B, C and D are designed to investigate such situations. In workload scenario Set B, different time periods within the planning window experience different job arrival rates. In the experiments, we have two time periods within the planning window. Both periods are set to 30 minutes. Set B represents the case of heavier workload for the first half of the planning window and lighter workload for the second half of the planning window. Workload scenario Set C represents the case of different job arrival rates for different areas of the row. This is commonly seen in terminals that some clusters might be busy serving vessels while others might see less job arrivals in some time periods. Workload scenario Set D represents the case of different job arrival rates not only for different areas of the row but also for different time periods within the planning window. It is most likely the case in real world terminals because (1) with the progress of a loading (unloading) process of a vessel, the locations in the yard to be accessed by vehicles will change; (2) the storage yard is serving different sets of vessels in different time periods so it is a mixed set of jobs of different types. Data in Table 1 show the mean inter-arrival times to the zones of interest under different workload scenarios. For example the mean inter-arrival time for the entire row is 150s in Scenarios Set A with 3 YCs; the mean inter-arrival time for the area of BLK2+BLK3 is 600s in Scenario Set C with 3 YCs.

A heavier workload scenario is also generated to test the performance of the algorithms by increasing the workload by 25% from the workload shown in Table 1.

5.3 Results Analysis

5.3.1 Performance: Job Waiting Time

We evaluate the YC workload partition algorithms by the average job waiting time (Avg_JWT). Avg_JWT resulted from workload partitioning for 3, 5 and 7 YCs over a row of 5 yard blocks for workload scenarios: Set A, Set B, Set C, and Set D are presented in Figures 11-14 respectively. Results presented in all figures show the percentage performance improvement of partition algorithm: SOP, TP1, TP2 and Ng's over the Basic algorithm.

Across all scenarios shown in Table 1, all four algorithms SOP, TP1, TP2 and Ng's improve from the basic algorithm in terms of average job waiting time. It shows that firstly, simply balancing workload by balancing the number of jobs among YCs as in Algorithm Basic will not work well. Secondly, it also shows that considering more partition points than using inter-block gaps only improves the performance. All four algorithms SOP, TP1, TP2 and Ng's consider more candidate partition points than Algorithm Basic. By assigning inter-block gantry time between two neighboring blocks to be equivalent to YC gantry time of 10 yard slots, the experiments have considered possible delays due to inter-block gantry. The results confirm that it is beneficial to have more candidate partition points.

Algorithms TP1 and TP2 are both better than SOP. This confirms that the problems with SOP discussed in Section 4.2 really affect the partition results badly. Space partition alone will not produce balanced workload among YCs in those situations mentioned there. This is also why Ng's algorithm with its swapping of jobs among neighbouring YCs is better than SOP. This result shows the effectiveness of doing time partition in our scheme.

Algorithm TP1 has no conclusive performance in comparison with Ng's. The difference between TP1 and TP2 is the strategy to carry out time partitioning. TP1 tries to partition a planning window by time length or number of jobs but the partition only occurs if adopting the two sub-windows improves the performance. TP2 gets the best partition in a process of repeatedly dividing the planning window by two. Results show that TP1 does not consider enough candidate time partition points as TP2 does and therefore performs worse than TP2.

Algorithm TP2 performs best in all tested cases. It improves over Algorithm Basic by 48% on average and improves over Ng's by 11.4% on average. Our time and space partitioning scheme outperforms Ng's approach for three reasons. Firstly, in the second phase of Ng's work, each job is considered in isolation whether it is beneficial to assign it to the neighboring YC. So a job will remain with the YC assigned in phase 1 if moving it (alone) to the neighboring YC does not improve YC performance. It may happen that moving a few jobs physically close to each other and with close arrival times to the neighboring YC is a better choice. This better choice will be discovered by our time partition and space partition algorithms. Secondly, it needs to be pointed out that the Ng's work is implemented here with the additional constraint of at least 8-slots separation between two neighboring YCs. Thus it is possible that the performance of Ng's work here is worse than in his original paper where two YCs could work side by side at the same time. We insist on the 8-slot separation so that vehicles served by the two YCs will not block each other at all times. Thirdly, computing the optimal YC dispatching sequences in the YC zones in our scheme is also a contributing factor compared with the use of the greedy first-complete-first-serve heuristic in Ng's

work. The results also show that finding suitable planning windows and the best zone partition for each planning window in algorithm TP2 is able to handle all scenarios better than other algorithms. These include the relative constant workload scenario A, different workload in different time periods within the planning window as in scenario B, different workload for different areas of the row as in scenario C and the combination of B and C as in scenario D. Set D represents the most dynamically changing job arrival pattern among all workload scenarios, which is usually the case in real terminals.

When the workloads are increased by 25% over the ones in Table 1, Avg_JWT resulted from workload partitioning for 3, 5 and 7 YCs over a row of 5 yard blocks for workload scenarios: Set A, Set B, Set C, and Set D are presented in Figures 15-18 respectively. Similar to previous results, algorithm TP2 always produces the smallest average job waiting time in all scenarios, with at least 35%, up to 59% improvement over the Basic Algorithm.

Percentage improvements of Algorithm TP2 over Algorithm Ng's are summarized in Table 10 under various settings. Data confirm that algorithm TP2 outperforms Ng's in all tested cases. The average percentage improvements for 3, 5 and 7 YCs are 9.6%, 9.2% and 17.5% respectively. When the ratio of YC/Blocks in a Row is greater than one, the performance differences between the two algorithms increase. It suggests that when more than one YCs work in one yard block, TP2 handles better the interference between YCs including prevention of collision and slowdowns than Ng's algorithm.

5.3.2 Performance: Computational Time

Another important performance indicator is the computational time of YC workload partition algorithms. Computational times resulted from various workload partitioning algorithms for 3, 5 and 7 YCs in a row of 5 yard blocks for workload scenarios: Set A, Set B, Set C, and Set D are presented in Table 2-5 respectively. When the workloads are increased by 25% over the ones in Table 1, computational times resulted from various algorithms for the same settings are presented in Table 6-9 respectively. Results presented in all tables show 95% confidence intervals of computational times in units of seconds.

Table 2: Computational time of the algorithms (Set A).

sec	7 YCs	5 YCs	3 YCs
SOP	17.6±14.8	8.3±4.6	2.0±1.7
TP1	22.2±16.3	9.6±4.5	2.1±1.4
TP2	21.8±16	9.4±4.5	2.1±1.4
Ng's	<1.0	<1.0	<1.0

Figure 11: Percentage improvement of average job waiting time of Algorithm TP1, TP2, Ng's over Algorithm Basic (Set A).

Table 3: Computational time of the algorithms (Set B).

sec	7 YCs	5 YCs	3 YCs
SOP	15.1±1.6	6.3±3.2	1.8±1.7
TP1	30.4±25.6	7.9±4.3	2.3±1.9
TP2	30.7±26.1	8.4±5.0	2.3±1.9
Ng's	<1.0	<1.0	<1.0

Figure 12: Percentage improvement of average job waiting time of Algorithm TP1, TP2, Ng's over Algorithm Basic (Set B).

Table 4: Computational time of the algorithms (Set C).

sec	7 YCs	5 YCs	3 YCs
SOP	21.6±14.4	7.2±7.0	2.5±2.1
TP1	30.7±18.9	10.2±8.6	2.6±2.1
TP2	30.1±18.4	10.0±8.5	2.6±2.1
Ng's	<1.0	<1.0	<1.0

Figure 13: Percentage improvement of average job waiting time of Algorithm TP1, TP2, Ng's over Algorithm Basic (Set C).

Table 5: Computational time of the algorithms (Set D).

sec	7 YCs	5 YCs	3 YCs
SOP	22.6±15.4	6.4±5.5	3.3±2.2
TP1	39.8±28.2	6.8±5.9	4.0±2.6
TP2	38.5±27.2	7.1±6.4	4.0±2.7
Ng's	<1.0	<1.0	<1.0

Figure 14: Percentage improvement of average job waiting time of Algorithm TP1, TP2, Ng's over Algorithm Basic (Set D).

Table 6: Computational time of the algorithms (Set A).

sec	7 YCs	5 YCs	3 YCs
SOP	47.5±30.8	38.4±23.4	18.4±17.2
TP1	85.2±42.0	48.7±34.3	20.0±18.4
TP2	72.6±35.8	48.0±34.1	19.9±18.4
Ng's	<1.0	<1.0	<1.0

Figure 15: Percentage improvement of average job waiting time of Algorithm TP1, TP2, Ng's over Algorithm Basic (Set A).

Table 7: Computational time of the algorithms (Set B).

sec	7 YCs	5 YCs	3 YCs
SOP	63.2±39.4	44.0±37.5	28.4±25.0
TP1	106.3±52.2	82.5±57.4	35.2±28.8
TP2	106.0±52.3	80.3±55.6	34.9±28.7
Ng's	<1.0	<1.0	<1.0

Figure 16: Percentage improvement of average job waiting time of Algorithm TP1, TP2, Ng's over Algorithm Basic (Set B).

Table 8: Computational time of the algorithms (Set C).

sec	7 YCs	5 YCs	3 YCs
SOP	61.8±43.8	51.7±34.1	37.6±32.4
TP1	103.1±80.3	79.5±55.5	39.3±33.3
TP2	100.3±79.6	78.2±54.8	38.9±32.9
Ng's	<1.0	<1.0	<1.0

Figure 17: Percentage improvement of average job waiting time of Algorithm TP1, TP2, Ng's over Algorithm Basic (Set C).

Table 9: Computational time of the algorithms (Set D).

sec	7 YCs	5 YCs	3 YCs
SOP	101.2±43.9	62.1±41.9	28.0±20.2
TP1	200.5±72.2	80.4±56.9	38±33.5
TP2	189.9±67.5	79.5±56.2	37.9±33.5
Ng's	<1.0	<1.0	<1.0

Figure 18: Percentage improvement of average job waiting time of Algorithm TP1, TP2, Ng's over Algorithm Basic (Set D).

Table 10: Percentage Improvements of Algorithm TP2 over Algorithm Ng's

	Normal Workload				Heavier Workload (125%)			
	Set A	Set B	Set C	Set D	Set A	Set B	Set C	Set D
7 YC	16.0%	17.8%	16.2%	14.7%	20.2%	23.6%	16.4%	14.9%
5 YC	8.6%	9.4%	8.1%	7.2%	9.1%	7.9%	11.4%	12.0%
3 YC	9.1%	10.4%	8.5%	11.0%	10.1%	9.6%	10.6%	7.7%

Across all scenarios shown in Table 1, Ng's algorithm is very fast in generating solutions by taking less than one second. The main reason is the use of the greedy first-complete-first-serve heuristic for individual YC dispatching. Algorithms TP1 and TP2 take more time than SOP (space only partition) as they need to partition the time and evaluate sub-planning windows. Among four workload scenarios, algorithms take slightly more times in scenario D where both time and space distributions of jobs vary. With respect to the workloads in Table 1, there is 95% confidence that algorithm TP2 will finish in 65.7 ($=38.5+27.2$) seconds.

When the workload is 25% heavier, all algorithms need more time to generate workload partition plans. Data in Table 6-9 show that computational time increases under the same scenario with the increase in the number of YCs. This reflects the fact that more time is needed to compute the partition of a heavier workload since the number of YCs is proportional to the incoming workload. Even in the worst case at scenario set D with 7 YCs, the TP2 algorithm is able to finish within 5 minutes ($> 189.9+67.5=257.4$ seconds) with 95% confidence.

In conclusion, Algorithm TP2 is able to improve YC workload partition performance in reasonable computational times through combined simulation optimization.

6. CONCLUSIONS AND FUTURE WORK

This paper proposes a new dynamic YC workload partition scheme. The scheme combines simulation and optimization to minimize the average vehicle job waiting time at the yard side. It involves both time partition and space partition algorithms to handle the changing job arrival patterns in the container storage yard. The time partition algorithm divides the planning window into a number of sub-planning windows of variable lengths. The

space partition algorithm divides a row of yard blocks into a number of non-overlapping zones for individual YCs to take charge. As a result, the working zones of YCs changes dynamically.

Our experimental results show that the time partition algorithm is very effective and performs much better than using space partition only to manage YC workload. The proposed algorithm TP2 which also involves space partition makes substantial improvements to job waiting times over the basic partition scheme and Ng's algorithm in all tested job arrival scenarios. Although the CPU time used by our algorithms are much more than Ng's algorithm, less than 5 minutes is required for the planning of one hour's operations in all tested scenarios. This means the CPU times of our algorithms are not an issue.

Future work includes the study of robustness of the algorithms under job arrival time noises. In our current YC management system, the inputs to the optimization module like vehicle arrival times are assumed accurately predictable by the simulation module. However, this information will likely involve uncertainties in reality. More thorough study can be done on how much uncertainties can be tolerated and whether the algorithms can be made more robust. Methodologies other than by simulation for the predictions of vehicle arrival times based on their current positions and other current status will also be worth looking into. Another problem is how to distribute a number of YCs into a number of rows of yard blocks. This is the Level 1 function in our hierarchical YC management scheme.

7. ACKNOWLEDGEMENT

We would like to thank the anonymous reviewers and the Associate Editor for their valuable comments.

8. REFERENCES

AXS-Alphaliner. Data on July, 2010.

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).

R.K. Cheung, C-L. Li and W. Lin, "Interblock Crane Deployment at Container Terminals", *Transportation Science*, 36(1): 79-93. (2002).

R. Dechter and J. Pearl, "Generalized Best-First Search Strategies and the Optimality of A*", *JACM*, 32(3): 505-536. (1985).

- G. Froyland, T. Koch, N. Megow, E. Duane and H. Wren, "Optimization the Landside Operation of a Container Terminal", *OR Spectrum*, 30: 53–75. (2008).
- X. Guo and S.Y. Huang, "Performing A* Search for Yard Crane Dispatching in Container Terminals", In *Proceedings of the International Conference on Tools with Artificial Intelligence*, (1): 263-267. (2008).
- X. Guo, S.Y. Huang, W.J. Hsu, M.Y.H. Low, "A Simulation Based Hybrid Algorithm for Yard Crane Dispatching in Container Terminals". In *Proceedings of the Winter Simulation Conference*, Dec, 2009.
- X. Guo, S.Y. Huang, W.J. Hsu, M.Y.H. Low, "Dynamic Yard Crane Dispatching in Container Terminals with Predicted Vehicle Arrival Information", *Advanced Engineering Informatics*, 25(3): 472-484. (2011).
- S.H. Jung and K.H. Kim, "Load Scheduling for Multiple Quay Cranes in Port Container Terminals", *Journal of Intelligent Manufacturing*, 17: 479–492. (2006).
- K.H. Kim, J.S. Kang, K.R. Ryu, "A Beam Search Algorithm for the Load Sequencing of Outbound Containers in Port Container Terminals". *OR Spectrum*, 26: 93–116. (2004).
- K.H. Kim and K.Y. Kim, "An Optimal Routing Algorithm for a Transfer Crane in Port Container Terminals", *Transportation Science*, 33 (1): 17–33. (1999).
- K.Y. Kim and K.H. Kim, "Heuristic Algorithms for Routing Yard-Side Equipment for Minimizing Loading Times in Container Terminals", *Naval Research Logistics*, 50: 498–514. (2003).
- H.Y. Lao and Y. Zhao, "Integrated Scheduling of Handling Equipment at Automated Container Terminals", *International Journal of Production Economics*, 112: 665–682. (2008).
- D-H. Lee, Z. Cao, Q. Meng, "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. (2007).
- L.H. Lee, E.P. Chew, K.C. Tan, Y.B. Han, 2006. "An Optimization Model for Storage Yard Management in Transshipment Hubs", *OR Spectrum*, 28: 539-561. (2006).
- A. Narasimhan and U.S. Palekar, "Analysis and Algorithm for the Transtainer Routing Problem in Container Port Operation", *Transportation Science*, 36 (1): 63–78. (2002).
- W.C. Ng, "Crane Scheduling in Container Yards with Intercrane Interference", *European Journal of Operational Research*, 164: 64–78. (2005).

- M.E.H. Petering, Y. Wu, W. Li, M. Goh, R.d. Souza, "Development and Simulation Analysis of Real-Time Yard Crane Control Systems for Seaport Container Transshipment Terminals", *OR Spectrum*, 31: 801-835. (2009).
- S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. 2nd Edition. Pearson Education, Inc., Upper Saddle River, NJ. (2003).
- R. Stahlbock, S. Voß, "Operations Research at Container Terminals: A Literature Update", *OR Spectrum*, 30(1): 1-52, (2007).
- D. Steenken, S. Voß, R. Stahlbock, "Container Terminal Operation and Operations Research – A Classification and Literature Review", *OR Spectrum*, 26: 3-49 (2004).
- I.F.A. Vis, R. De Koster, "Transshipment of Containers at a Container Terminal: An Overview", *European Journal of Operational Research*, 147(1): 1-16. (2003).
- Q. Zeng and Z. Yang, "Integrating Simulation and Optimization to Schedule Loading Operations in Container Terminals", *Computers & Operations Research*, 36(6): 1935–1944. (2009).
- C. Zhang, Y-W. Wan, J. Liu, R.J. Linn, "Dynamic Crane Deployment in Container Storage Yard". *Transportation Research Part B: Methodological*, 36(6): 537-555. (2002).