Journal of Zhejiang University SCIENCE A
ISSN 1009-3095
http://www.zju.edu.cn/jzus
E-mail: jzus@zju.edu.cn

An integrated approach for modeling and solving the scheduling problem of container handling systems

CHEN Lu (陈 璐)†[1,2,3], XI Li-feng (奚立峰)[1], CAI Jian-guo (蔡建国)[1], BOSTEL Nathalie[2,4], DEJAX Pierre†[2,3]

([1]*School of Mechanical Engineering, Shanghai Jiao Tong University, Shanghai 200030, China*)
([2]*IRCCyN–Nantes Cybernetic and Communication Research Institute, BP 92 101, 44321 Nantes Cedex 03, France*)
([3]*Department of Automatic Control and Industrial Engineering, Ecole des Mines de Nantes, The Chantrerie, F-44070 Nantes Cedex 03, France*)
([4]*IUT of Saint Nazaire of Nantes University, GLT Department, 58 Michel Ange, 44606 Saint Nazaire Cedex, France*)
†E-mail: chenlu@sjtu.edu.cn; pierre.dejax@emn.fr

Received Dec. 23, 2004; revision accepted Apr. 13, 2005

Abstract: An integrated model is presented to schedule the container handling system. The objective is to improve the cooperation between different types of equipments, and to increase the productivity of the terminal. The problem is formulated as a Hybrid Flow Shop Scheduling problem with precedence constraint, setup times and blocking (HFSS-B). A tabu search algorithm is proposed to solve this problem. The quality and efficiency of the proposed algorithm is analyzed from the computational point of view.

Key words: Container terminal, Scheduling, Tabu search, Hybrid flow shop, Makespan
doi:10.1631/jzus.2006.A0234 **Document code:** A **CLC number:** TP29

INTRODUCTION

Worldwide container trade has grown 9.5% per year last decade and will continue to do so at an 8% growth rate in the coming years (James *et al.*, 1997), so container terminals have to handle containers more efficiently. Fig.1 shows a typical container terminal layout composed of "Quayside area" and "Storage yard".

Many optimization problems associated with a container terminal have been extensively studied in the past few years. Vis and De Koster (2003) gave a comprehensive review of literature. Kim and Park (2004) treated quay cranes (QCs) scheduling as an *m*-parallel machine scheduling problem. Kim and Kim (1999a; 1999b; 1999c) tried to minimize the total travel distance of a yard crane. Bose *et al*.(2000) dealt with dispatching strategies for straddle carriers. Meersmans (2002) provided models and algorithms for scheduling container handling equipment in an integrated way in an automated container terminal but considered only loading operations.

In view of the strong interdependence of various equipments, we developed an integrated model to: (1) address the scheduling of various equipments at the same time so as to achieve a higher level of coordination, (2) consider both loading and unloading operations simultaneously, (3) minimize the makespan in serving loading and unloading ships in a given time period.

PROBLEM DESCRIPTION

As soon as a berth is assigned for a ship, QCs are designated for discharging import containers and loading export containers. At the storage yard, yard cranes (YCs) stack containers in block. And yard vehicles (YVs) are used for transporting containers between the quay side and the yard side, serving as prime movers of both the quay crane and yard crane.

Basic notations

(1) Jobs: Each container is associated with a job, which is defined as a complete loading or discharging

process. There are two main kinds of jobs: loading jobs and unloading jobs.

(2) Machines: There are three different sets of machines: QC, YC and YVs.

(3) Operations: Each job consists of the following three operations: Transfer operation of a container from/onto the ship, which is done by QCs; Transfer operation within the storage yard is done by YCs; Transfer operation between the QCs and YCs is implemented by YVs.

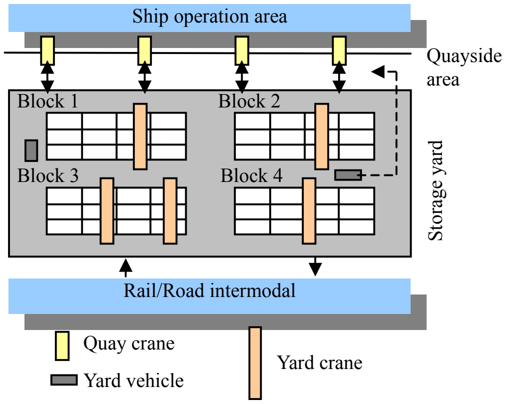

Fig.1 A typical container terminal layout

Problem definition

We assume that no YC is shared between inbound containers and outbound containers. Since for one ship, loading operations will be scheduled after all unloading operations. We could treat our problem as a Hybrid Flow Shop Scheduling (HFSS) problem by adding dummy cranes for QCs. When QC_d has loading operations following unloading operations in its sequence, we add a dummy crane QC'_d for each QC_d, $d=1, \ldots, D$. Then, we move all the loading operations from QC_d to QC'_d. As a result, each QC_d is decomposed into two: one for unloading operation and the other for loading operations. Therefore, using the triplet $\alpha|\beta|\gamma$ notation for scheduling problems (Pinedo, 2002), we define our problem as follows:

Definition 1 We use $HF, unrelated|prec, s_{ik}, block|C_{\max}$ to denote the problem as a 3-stage Hybrid Flow Shop Scheduling Problem with precedence constraint, setup times and Blocking (HFSS-B), where each stage is made up of unrelated parallel machines. The objective is to minimize the makespan.

The restrictions of our scheduling problem are:

(1) Unrelated parallel machines: Since the travel distance of each QC and YC is limited in order to avoid potential collision, the whole set of QCs and YCs are divided into several subsets, each of which has some parallel machines.

(2) Job precedence constraints: For unloading operations, containers on a deck must be discharged before containers in the hold of the same ship-bay; for loading operations, containers in a hold must precede the containers on the deck of the same ship-bay.

(3) Sequence dependent setup times: In a container terminal, empty travel times arise when a crane or a vehicle travels between two containers. These empty travel times may be regarded as sequence-dependent setup times.

(4) Blocking: If a job shop has a limited buffer or no buffer at all between two successive machines, then it may happen that when the buffer is full the upstream machine is not allowed to release a completed job. This phenomenon is known as “blocking”, and will happen in a container terminal. The transfer of containers between different equipment, i.e., from QCs to YVs, from YVs to YCs, can only take place on condition that the downstream equipment is ready.

TABU SEARCH ALGORITHM

The HFSS problem can be seen as an extension of the classical flowshop (Negenman, 2001; Nowicki and Smutnicki, 1998; Low, 2005). A computational experiment showed that the computational time is excessive for practical use. For example, a relaxation problem with unit setup times, one QC, one YC, one YV and ten containers required more than 4 h to solve completely, using Xpress-MP, a specialized software for integer and linear program.

Tabu search (TS) is a meta-heuristic method developed by Glover (1989; 1990) for large combinatorial optimization tasks. Critical path-based local search algorithms (Van Laarhoven *et al.*, 1992; Dell’Amico and Trubian, 1993; Taillard, 1994) had been proved to be extremely efficient in solving classical flow shop scheduling problem. Unfortunately, these methods cannot be applied directly to scheduling problem with blocking constraints. In our research, we propose a completely new neighborhood structure to deal with the scheduling problem with blocking constraints. Based on this neighborhood structure, a tabu search algorithm was designed and implemented.

Solution representation

We use vector representation to represent a solution. The following notations are used: P denotes a complete schedule; M denotes the whole set of machines; j is an index of stage, j=1, 2, 3; k is an index of machine, $k\in M$; $M(j)$ denotes the set of machines at stage j, j=1, 2, 3; m_j is the number of machines in $M(j)$, $m_j=|M(j)|$.

A schedule P can be decomposed into subsets P_{jk}, $P=\bigcup_{\substack{j=1,2,3\\k\in M}} P_{jk}$. P_{jk} denotes a sequence of operations involving machine k in stage j. Let $n_{jk}=|P_{jk}|$, P_{jk} can be defined by a vector $\boldsymbol{\pi}_{jk}=(\pi_{jk}(1),\ldots,\pi_{jk}(n_{jk}))$, where $\pi_{jk}(i)$ denotes the element in position i in $\boldsymbol{\pi}_{jk}$. We can represent a solution by m-tuple $\boldsymbol{\pi}=(\boldsymbol{\pi}_{11},\ \ldots,\ \boldsymbol{\pi}_{jm_j})$. Further, if we distinguish the machine in different stages, and use $\boldsymbol{\pi}_j=(\boldsymbol{\pi}_{j1},\ \ldots,\ \boldsymbol{\pi}_{jm_j})$ to represent the processing sequence on the machines in stage j. Then, we can define a solution in the following way:

$$\boldsymbol{\pi}=\bigcup_{j=1}^{3}\boldsymbol{\pi}_j=\bigcup_{j=1}^{3}(\boldsymbol{\pi}_{j1},\ldots,\boldsymbol{\pi}_{jm_j}). \qquad (1)$$

Initial solution generation

In this section, a Multiple Insertion Heuristic (MIH) is developed for obtaining an initial solution. It is based on the insertion heuristic for the Vehicle Routing Problem (VRP) (Vigo and Toth, 2002). The main idea of MIH is to schedule all containers on each stage in turn. When scheduling the current container, we consider the possibility of inserting it into every possible position in the partial container sequence; and find the most feasible insertion place for the current container using a priority criterion. Because our scheduling problem has no intermediate storage, those containers with assigned tasks finished earlier will be delivered to the next stage as soon as possible. So First Come First Served (FCFS) rule is applied when building the container sequences in the stages other than the first stage. Jobs' precedence constraints are considered before inserting a container and applying the FCFS rule, which is to ensure that a feasible schedule can be obtained.

The heuristics for generating the initial schedule can be fully described as follows:

Step 1: Arrange containers in numerical order from 1 to n in stage j=1.

Step 2: The priority of a container follows numerical order. A container i is inserted at all possible positions in the machine sequence developed so far. Extra time c_i for each insertion is calculated.

Step 3: Place container i at the position with the minimum extra time (MET) c_i^*.

Step 4: For the remaining stages, the priority of a container follows FCFS rule. A container is inserted in the position incurring minimum extra time.

To insert container i between container l and k, the extra time is defined as

$$c_i=c(l,i,k)=s_{il}+s_{ik}-s_{lk},\ \ i,\ l,\ k=1,\ \ldots,\ n, \qquad (2)$$

where s_{ik} means the setup time between container i and k.

It is noted that, besides the MET, the insertion priority can be non-delay (ND) machine and minimum completion time (MCT). Therefore, three priority policies can be applied to implement the initial solution heuristics. Thus, we have MIH_MET, MIH_ND, MIH_MCT. The overall complexity of MIH_MET, MIH_ND, MIH_MCT is $O(n^2)$, $O(n^2)$ and $O(n^3)$ respectively (Chen *et al.*, 2004).

Neighborhood structure

By implementing insertion move, our neighborhood structure N_{Ist} restricts the search to a subset of the solution space. A k-insertion of operation $v\in O$ in machine $k\in M$ is performed in two steps: (1) Delete v from its current machine sequence $\boldsymbol{\pi}_{ja}$; (2) Assign v to machine k and choose the position of v in $\boldsymbol{\pi}_{jk}$. Note that if machine $a\neq k$, then the operation is re-assigned to another machine; if $a=k$, then the operation is re-sequenced on the same machine.

We restrict the search to the collection of solutions associated with the vector $\boldsymbol{\pi}_1$. Each vector $\boldsymbol{\pi}_1$ represents the container processing orders in the first stage. When building a complete schedule, the containers are firstly sequenced in the non-decreasing order of their completion time at the previous stage, and then assigned to the First Available Machines (FAM) at the current stage.

Neighborhood N_{Ist} In the first stage (j=1) of solution $\boldsymbol{\pi}$, x is a position of vector $\boldsymbol{\pi}_{1a}$. A neighbor of $\boldsymbol{\pi}$ is obtained by performing a b-insertion of the operation $\boldsymbol{\pi}_{1a}(x)$, a, $b\in M(1)$ (compatible with precedence constraints). The job schedules for the subsequent stages (j=2, 3) are obtained by using FCFS and FAM.

Since the job sequence in Stage I is obtained through insertion move, and job sequences for subsequent stages (j=2, 3) are obtained by using FCFS and FAM, the complexity of neighborhood solution calculation is $O(n)$.

Tabu list and search strategy

A tabu list is applied to trace the evolution of the search to prevent cycling. In our case, once a job j is moved away from position x, we add both job and position to the tabu list. We do not allow a move that involves returning job j to position x for a certain number of iterations. Aspiration criterion is used to override the tabu status of a move. In our algorithm, we allow a tabu move if such a move leads to a neighborhood solution that has a smaller $C_{\max}$ (makespan) than that of the best solution obtained so far. We adopt the best-fit strategy in our search. Given a schedule (a job processing order) $\boldsymbol{\pi}$, we evaluate each neighborhood solution of $\boldsymbol{\pi}$ which are not currently forbidden in the tabu list, and then move to the best non-tabu neighbor.

Algorithm

The above considerations lead us to the following TS algorithm. We start with an initial processing order $\boldsymbol{\pi}$ and a primary empty tabu list T. At each iteration, we apply the neighborhood search strategy to find out the set of moves $OP(\boldsymbol{\pi})$. We select a best non-tabu move op', which determines neighborhood $\boldsymbol{\pi}'=\pi_{op'}$. Thereafter we create the new tabu list T. The processing order $\boldsymbol{\pi}'$ is set to be the primal for the next iteration. The best found $C^*_{\max}$ and associated processing order $\boldsymbol{\pi}^*$ are updated. The algorithm terminates if there is no improvement to the best solution obtained after a certain number of iterations (*NonImpIter*), or if the total number of iterations reaches a predetermined value *MaxIter*.

Step 1: Initialization: $\boldsymbol{\pi}^*=\boldsymbol{\pi}$, $C^*_{\max}=C_{\max}(\boldsymbol{\pi}^*)$, $T=\varnothing$, $Iter=NIter=0$.

Step 2: Set $Iter:=Iter+1$, $NIter:=NIter+1$. Find the move $op'\in OP(\boldsymbol{\pi})$, the neighborhood $\boldsymbol{\pi}'=\boldsymbol{\pi}_{op'}$ and the modified tabu list T. Set $\boldsymbol{\pi}:=\boldsymbol{\pi}'$.

Step 3: If $C_{\max}(\boldsymbol{\pi})<C_{\max}$, then set $\boldsymbol{\pi}^*:=\boldsymbol{\pi}$, $C^*_{\max}=C_{\max}(\boldsymbol{\pi})$, $NIter:=0$ and go to Step 2.

Step 4: If ($Iter\le MaxIter$) & ($NIter\le NonImpIter$), then go to Step 2; otherwise stop.

Counters *Iter* and *NIter* trace the number of total iterations and the number of iterations performed without improvement, respectively.

COMPUTATIONAL EXPERIMENTS

Problem settings

Our computational study is based on the real situation we got from the Port of Shanghai. The performance of the algorithm is compared with the SH lower bound (Santos *et al*., 1995). SH lower bound is a global lower bound for makespan problem. All experimental tests were implemented on a personal computer with a Centrino 1300 MHz, 256 MB RAM.

Initial seed selection

Average Percentage Deviation (*APD*) is used to evaluate the performance of the algorithms,

$$APD=100\times\frac{\text{heuristic solution}-\text{lower bound}}{\text{lower bound}}. \tag{3}$$

The *APD* is computed for 20 test problems in each environment. The performances of three priority policies, MIH_MET, MIH_ND and MIH_MCT, are evaluated. The average performance is summarized in Table 1, where it is observed that the MIH_ND performs best in terms of the makespan as the problem size (n means the number of containers) increases.

Proposed TA performance

Two proposed tabu search algorithms are compared with each other for both solution quality and efficiency. These two different algorithms are defined as:

TA1: tabu search algorithm with MIH_ND initial solution heuristics;

TA2: tabu search algorithm with MIH_MET initial solution heuristics.

The experimental results are summarized in Table 2, which includes the APD of the makespan obtained from the tabu search algorithms and their computation times. We observed that for small sized problems, both TA1 and TA2 could obtain an optimal solution

Table 1 Performance comparison of three priority policies

Problem size				Performance in *APD**			CPU time (s)		
n	QC	YC	YV	MIH_MET	MIH_ND	MIH_MCT	MIH_MET	MIH_ND	MIH_MCT
10	2	4	4	5.35(0)	8.66(5)	7.68(2)	0.290	0.382	0.301
10	2	4	6	0.50(3)	7.52(6)	4.40(3)	0.291	0.371	0.300
20	2	4	6	14.55(0)	4.98(2)	5.12(1)	0.340	0.400	0.321
20	2	4	8	13.78(0)	2.56(2)	2.67(1)	0.310	0.401	0.321
20	2	6	8	16.45(0)	4.18(2)	4.17(0)	0.310	0.391	0.321
40	2	4	8	16.21(0)	5.78(0)	6.87(0)	0.471	0.561	0.500
40	2	6	8	19.88(0)	8.65(0)	10.32(0)	0.480	0.561	0.501
40	2	6	10	21.58(0)	8.83(0)	9.76(0)	0.481	0.561	0.501
50	2	6	8	23.12(0)	12.22(0)	13.65(0)	0.641	0.731	0.661
50	2	6	10	25.49(0)	12.45(0)	12.56(0)	0.651	0.731	0.661
50	4	6	10	70.88(0)	13.64(0)	12.75(0)	0.641	0.731	0.661
50	4	8	10	75.11(0)	11.75(0)	12.34(0)	0.641	0.721	0.671
60	4	8	12	82.76(0)	11.31(0)	12.09(0)	0.882	0.962	0.932
60	4	8	15	63.90(0)	12.36(0)	12.45(0)	0.882	0.971	0.911
60	4	10	12	87.11(0)	13.09(0)	15.74(0)	0.881	0.971	0.911
60	4	10	15	88.35(0)	15.12(0)	14.36(0)	0.881	0.972	0.901
80	4	10	15	89.85(0)	15.65(0)	16.76(0)	1.672	1.752	1.703
80	4	10	20	90.09(0)	16.32(0)	17.85(0)	1.672	1.752	1.712
100	4	10	20	86.17(0)	15.73(0)	14.98(0)	2.975	3.075	3.034
100	4	12	24	91.54(0)	17.36(0)	18.76(0)	2.984	3.075	3.024

*Number of optimal solutions found in the 20 iterations

Table 2 Performance comparison of TA1 and TA2

Problem size				Performance in *APD**		CPU time (s)	
n	QC	YC	YV	TA1	TA2	TA1	TA2
10	2	4	4	0.71(15)	2.19(5)	26.0	41.8
10	2	4	6	1.30(12)	0.33(12)	32.7	39.0
20	2	4	6	0.15(12)	0.68(2)	34.0	37.0
20	2	4	8	1.10(5)	0.35(8)	39.5	35.8
20	2	6	8	0.45(4)	2.13(0)	34.0	35.3
40	2	4	8	0.52(0)	2.60(0)	43.0	61.7
40	2	6	8	0.38(0)	0.27(0)	46.7	59.0
40	2	6	10	0.33(2)	0.22(0)	48.0	44.2
50	2	6	8	0.48(0)	0.30(0)	95.2	95.6
50	2	6	10	0.44(0)	0.30(0)	94.0	93.7
50	4	6	10	9.39(0)	9.92(0)	131.7	157.0
50	4	8	10	8.09(0)	>20	113.3	165.0
60	4	8	12	8.81(0)	>20	197.7	257.7
60	4	8	15	8.36(0)	>20	179.0	262.5
60	4	10	12	7.58(0)	>20	216.5	278.0
60	4	10	15	8.63(0)	>20	112.5	140.7
80	4	10	15	8.43(0)	>20	272.3	461.0
80	4	10	20	8.26(0)	>20	269.0	905.3
100	4	10	20	8.72(0)	>20	496.2	1506.0
100	4	12	24	8.58(0)	>20	563.0	1872.1

* Number of optimal solutions found in the 20 iterations

in most of the 20 iterations. When the job number and the machine number increase, the performance of TA1 is much better than TA2 in terms of both solution quality and computational efficiency. It reveals the fact that initial solution heuristics affect the performance of a designed algorithm.

CONCLUSION

In this work, an integrated model was developed to schedule various kinds of handling equipment in a maritime container terminal. The objective is to improve the cooperation among different types of equipments, and to increase the productivity of the terminal. A TS based algorithm was developed to solve the addressed problem. The computational results showed that the proposed algorithm TA1 outperforms TA2, with respect to solution quality, especially when the job size is increasing. The running time for TA1 is acceptable. In addition, from the experimental results one can also observe that good initial solution heuristics is important for this kind of scheduling problem and helpful for further improvement of the solution.

References

Bose, J., Reiners, T., Steenken, D., 2000. Vehicle Dispatching at Seaport Container Terminal Using Evolutionary Algorithm. Proceedings of the 33rd Hawaii International Conference on System Science, p.1-10.

Chen, L., Bostel, N., Dejax, P., Cai, J., Xi, L., 2004. Heuristiques Pour le Problème D'ordonnancement Intégré des Équipements de Manutention de Conteneurs dans un Terminal Maritime. Proceedings de la 5e conférence francophone de modélisation et simulation, France, p.1089-1095 (in French).

Dell'Amico, M., Trubian, M., 1993. Applying tabu search to the job-shop scheduling problem. *Annals of Operations Research*, **41**(3):231-252. [doi:10.1007/BF02023076]

Glover, F., 1989. Tabu search part I. *ORSA Journal on Computing*, **1**:190-206.

Glover, F., 1990. Tabu search part II. *ORSA Journal on Computing*, **2**:4-32.

James, A.P., Howard, J.M., Basilotto, J.P., Harbottle, H., 1997. Megaports and Load Centers of the Future with the Port of Houston as the Baseline Port. Research Report, Texas A&M University.

Kim, K.H., Kim, K.Y., 1999a. Routing straddle carriers for the loading operation of containers using a beam search algorithm. *Computers & Industrial Engineering*, **36**(1):109-136. [doi:10.1016/S0360-8352(99)00005-4]

Kim, K.H., Kim, K.Y., 1999b. A routing algorithm for a single straddle carrier to load export containers onto a containership. *International Journal of Production Economics*, **59**(1-3):425-433. [doi:10.1016/S0925-5273(98)00108-X]

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

Kim, K.H., Park, Y., 2004. A crane scheduling method for port container terminals. *European Journal of Operational Research*, **156**(3):752-768. [doi:10.1016/S0377-2217(03)00133-4]

Low, C.Y., 2005. Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines. *Computers & Operations Research*, **32**(8):2013-2026. [doi:10.1016/j.cor.2004.01.003]

Meersmans, P.J.M., 2002. Optimization of Container Handling Systems. Ph.D Thesis, Tinbergen Institute 271, Erasmus University Rotterdam.

Negenman, E.G., 2001. Local search algorithms for the multiprocessor flow shop scheduling problem. *European Journal of Operational Research*, **128**(1):147-158. [doi:10.1016/S0377-2217(99)00354-9]

Nowicki, E., Smutnicki, C., 1998. The flow shop with parallel machines: a tabu search approach. *European Journal of Operational Research*, **106**(2-3):226-253. [doi:10.1016/S0377-2217(97)00260-9]

Pinedo, M., 2002. Scheduling Theory, Algorithms, and Systems, 2nd Edition. Prentice Hall, New Jersey.

Santos, D.L., Hunsucker, J.L., Deal, D.E., 1995. Global lower bounds for flow shops with multiple processors. *European Journal of Operational Research*, **80**(1):112-120. [doi:10.1016/0377-2217(93)E0326-S]

Taillard, E.D., 1994. Parallel taboo search techniques for the job-shop scheduling problem. *ORSA Journal on Computing*, **6**(2):108-117.

Van Laarhoven, P.J.M., Aarts, E.H.L., Lenstra, J.K., 1992. Job shop scheduling by simulated annealing. *Operations Research*, **40**(1):113-125.

Vigo, D., Toth, P., 2002. The Vehicle Routing Problem. Philadelphia, USA.

Vis, I.F.A., De Koster, R., 2003. Transshipment of containers at a container terminal: An overview. *European Journal of Operational Research*, **147**(1):1-16. [doi:10.1016/S0377-2217(02)00293-X]