

# Crane Scheduling with Spatial Constraints: Mathematical Model and Solving Approaches

Yi Zhu and Andrew Lim

*Dept of IEEM, Hong Kong University of Science and Technology  
Clear Water Bay, Kowloon, Hong Kong  
 $\{zhuyi,iealim\}@ust.hk$*

## Abstract

In this paper, we examine crane scheduling for ports. This important component of port operations management is studied when the non-crossing spatial constraint, which is common to crane operations, are considered. We assume that ships can be divided into holds and that cranes can move from hold to hold but jobs are not preemptive, so that only one crane can work on one hold or job to complete it. Our objective is to minimize the latest completion time for all jobs, which is a widely used criteria in practice. We formulate this problem as an integer programming problem with linear objective function and constraints. We provide the proof that this problem is NP-complete and design a branch-and-bound algorithm to obtain optimal solutions. A simulated annealing meta-heuristic with effective neighborhood search is designed to find good solutions in larger size instances. The elaborate experimental results show that the branch-and-bound algorithm runs much faster than CPLEX and the simulated annealing approach can obtain near optimal solutions for instances of various sizes.

## 1 Introduction

The Port of Singapore Authority (PSA), a large technology corporation located in Singapore, is one of the busiest ports in the world. PSA handles 17.04 million TEU's annually or nine percent of global container throughput in Singapore, the world's largest transshipment hub. PSA is concerned with maximizing throughput at its ports in view of pressures derived from limited port size, high cargo transshipment volumes and limited physical facilities and equipment [1][6].

Crane scheduling and work schedules play an important part in port management. Cranes are in the interface between the land and water sides, each with their traffic lanes, intersections, and vehicle flow control systems. It is in this multi-channel interface that we find bottlenecks and where cranes and other cargo-handling equipment (fork lifts, conveyors etc.) come into operation. We find that much of the operational decision-making done at PSA is based on practical experience and simulation (see, for example, [6]). While the latter is invaluable in general, analytic models can provide an enhanced role without the limitations of experience-generated rules-of-thumbs or simulation (see [3]).

Port systems are usually peculiar, making existing models invalidate. Sabria and Aganzo [5] give a bibliography on port operations with the focus on berthing and cargo-handling systems. This bibliography contains some analytic models that have been developed for other ports. Berthing is a widely-analyzed port activity, where queuing theory finds application (see for example, Sabri and Daganzo [5]). On the other hand, traffic and vehicle-flow scheduling on landside is also well studied (see, for example, [1]). In [9], container handling is modeled as “work” which cranes can do at constant rates, and cranes can interrupt their work without loss of efficiency. This constitutes an “open shop” with parallel, identical machines, where jobs consist of independent, single-stage and preemptive tasks. A branch-and-bound method is used to minimize delay costs.

Recently, a set of spatial constraints are studied in crane scheduling problem ([7][8]). The most interesting one is the non-crossing constraint, i.e. crane arms cannot be crossed over each other simultaneously. It is a structural constraint on cranes and crane tracks. In [7] and [8], the problems are modeled as bipartite graph matching. Two simpler problems are solved by dynamic programming algorithms and some heuristic methods are designed to tackle the hardest version.

However, in [7, 8] the job-to-crane assignment is only performed once to maximize the total throughput without taking time into consideration. In practice, the scheduling target is often to complete all the jobs with respect to certain criteria. Thus in this paper, we augment the work by solving the problem that minimizes the latest completion time of all the jobs, which come in different sizes. We assume that once a crane starts to do the job, it could not stop the work until it completes the whole job,i.e., the jobs are non-preemptive.

## 2 Mathematical Formulation

Throughout this paper we study the crane scheduling problem that concerns the job-to-crane assignment in order to minimize the latest completion time. The following assumptions and constraints are considered:

- All jobs have different processing times but the crane rate is constant;
- All jobs are non-preemptive, i.e., once a crane starts to do a job, it must complete it without any pause or shift;
- Any two cranes cannot do two jobs simultaneously if their arms are crossed. Cranes lines up in tracks and jobs are in ships which berth along the wharf. We label the cranes and jobs according to their relative spatial positions. This means cranes, 1, 2, 3, ..., are arranged on a line from left to right (or east to west), jobs, 1, 2, 3, ..., are in the similar manner. For details, please see Figure 1. The non-crossing constraint can be simply expressed as:  
*If job  $i$  is assigned to crane  $k$  and job  $j$  is assigned to crane  $l$  simultaneously, then  $i < j$  if and only if  $k < l$ .*

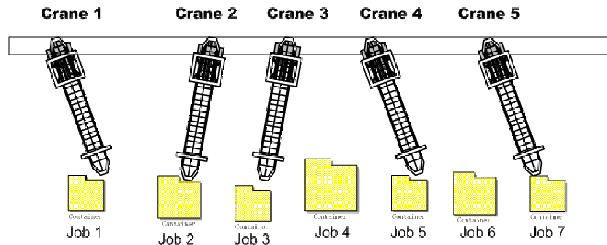


Figure 1: An Illustration of Crane and Job Line

We present the formulation of the problem below:

Input data to the problem:

- $m$ : the number of cranes;
- $n$ : the number of jobs;
- $p_i$ : the processing time of job  $i$  ( $1 \leq i \leq n$ );

The decision variables:

- $c_i$ : an integer variable which represents the completion time of job  $i$  ( $1 \leq i \leq n$ );
- $C_{max}$ : an integer variable which represents the latest completion time among all jobs;
- $x_{ik}$ : a binary variable equal to 1 if and only if job  $i$  is assigned to crane  $k$  ( $1 \leq i \leq n, 1 \leq k \leq m$ );
- $y_{ij}$ : a binary variable equal to 1 if and only if job  $i$  completes no later than job  $j$  starts, i.e.,  $c_i \leq c_j - p_j$  ( $1 \leq i, j \leq n$ );
- $z_{ijkl}$ : a binary variable equal to 1 if and only if job  $i$  is assigned to crane  $k$  and job  $j$  is assigned to crane  $l$ .

The mathematical formulation ( $M$  is a sufficiently large constant number) :

**Minimize:**  $C_{max}$

**Subject to:**

$$C_{max} \geq c_i, \forall 1 \leq i \leq n \quad (1)$$

$$c_i - p_i \geq 0, \forall 1 \leq i \leq n \quad (2)$$

$$\sum_{k=1}^m x_{ik} = 1, \forall 1 \leq i \leq n \quad (3)$$

$$z_{ijkl} \leq x_{ik}, \forall 1 \leq i, j \leq n, \forall 1 \leq k, l \leq m \quad (4)$$

$$z_{ijkl} \leq x_{jl}, \forall 1 \leq i, j \leq n, \forall 1 \leq k, l \leq m \quad (5)$$

$$x_{ik} + x_{jl} - 1 \leq z_{ijkl}, \forall 1 \leq i, j \leq n, \forall 1 \leq k, l \leq m \quad (6)$$

$$c_i - (c_j - p_j) + y_{ij}M > 0, \forall 1 \leq i, j \leq n \quad (7)$$

$$c_i - (c_j - p_j) - (1 - y_{ij})M \leq 0, \forall 1 \leq i, j \leq n \quad (8)$$

$$y_{ij} + y_{ji} \geq z_{iikk}, \forall 1 \leq i, j \leq n, i \neq j, \forall 1 \leq k \leq m \quad (9)$$

$$y_{ij} + y_{ji} \geq z_{ijkl}, \forall 1 \leq i < j \leq n, \forall 1 \leq l < k \leq m \quad (10)$$

In the above formulation, constraints (1) and (2) define the properties of decision variables  $C_{max}$  and  $c_i$ . Constraints (3) ensures that one job must be assigned to exactly one crane. Constraints (4)–(6) define the variable  $z$  (similar definition can be found in [10]). Constraints (7)–(8) define the properties of the variable  $y$ : constraint (7) indicates that  $y_{ij} = 1$  if  $c_i \leq (c_j - p_j)$ , which means  $y_{ij} = 1$  when job  $i$  finishes no later than job  $j$  starts; constraint (8) indicates that  $y_{ij} = 0$  if  $c_i > (c_j - p_j)$ , which means  $y_{ij} = 0$

when job  $i$  finishes after job  $j$  starts. Constraint (9) specifies that the jobs assigned to one crane cannot overlap. And finally, constraint (10) specifies the non-crossing constraint, i.e., if job  $i$  is assigned to crane  $k$  and job  $j$  is assigned to crane  $l$  simultaneously (i.e. their processing durations overlap each other), then  $i < j$  if and only if  $k < l$ .

Note that in this integer programming model, all the objective function and constraints are linear. Standard integer programming solvers, such as CPLEX, can be used to get optimal solutions for smaller instances. However, the number of inequalities, such as constraints (9) and (10) is large even for instances with moderate size, which have  $O(n^2m^2)$  number of inequalities. Hence the running time of branch-and-bound algorithm of the IP solver can be quite long, as verified in the experimental results section.

### 3 NP-completeness Proof

We prove the decision version of the crane scheduling problem is *NP-complete*:

- **Input:** the number of cranes  $m$ , the number of jobs  $n$ , and the jobs processing time  $p_i$  ( $1 \leq i \leq n$ ), a positive integer  $C$ ;
- **Question:** Is there a job-to-crane assignment, so that if job  $i$  is assigned to crane  $k$  and job  $j$  is assigned to crane  $l$  simultaneously, then  $i < j$  if and only if  $k < l$  (the non-crossing constraint is satisfied), and  $\text{Max}\{c_i\} = C?$  ( $1 \leq i \leq n$ ,  $c_i$  denotes the completion time of job  $i$ )

**Proof:** To prove a problem is *NP-complete*, we have to show it is in *NP* and also *NP-hard* [2]. It is trivial to show that crane scheduling problem is in *NP*, as checking whether a solution has latest completion time  $k$  can be done in  $O(n)$  time and checking the non-crossing constraint can be accomplished in  $O(n^2)$  time.

To prove crane scheduling problem is *NP-hard*, we show that the set partition problem, which is known as *NP-complete*, is reducible to crane scheduling problem. The set partition problem is described as follows:

- **Input:** An integer set  $S = \{s_1, s_2, \dots, s_n\}$
- **Question:** Can  $S$  be partitioned into two disjoint subsets  $S_1$  and  $S_2$ , such that  $\sum_{s_i \in S_1} s_i = \sum_{s_i \in S_2} s_i = hs$ , where  $hs = 1/2 \sum_{s_i \in S} s_i$ ?

The reduction algorithm takes an instance  $S$  of the set partition problem. The constructed crane scheduling problem instance has 2 cranes and  $n + 2$  jobs; the processing time of both job 1 and job  $(n + 2)$  is  $hs$ ; and the

processing time of job  $(i + 1)$  has the processing time  $s_i$ , where  $1 \leq i \leq n$ . For example, if the set  $S$  is  $\{1, 3, 4, 6\}$ , then the crane scheduling problem has 6 jobs, with the processing time 7, 1, 3, 4, 6 and 7 respectively. Obviously, this transformation algorithm runs in polynomial time.

To complete the proof, we show that this transformation is indeed a reduction: the set  $S$  can be partitioned to two equal subsets if and only if all the  $n + 2$  jobs could be completed in  $2hs$  time.

Suppose set  $S$  can be partitioned to two equal sets  $S_1$  and  $S_2$ , then perform the crane-to-job assignments in two steps:

- **Step 1:** we assign job 1 to crane 1. Since crane 1 needs time  $hs$  to complete job 1, we can assign jobs  $j_1, j_2, \dots, j_k$  to crane 2, where  $j_i - 1 \in S_1, 1 \leq i \leq k$ . Since the sum of processing time of all the jobs  $j_1, j_2, \dots, j_k$  is  $hs$ , and the non-crossing constraint is not considered (as crane 1 is doing the first job, and the labels of jobs done by crane 2 are all greater than 1), crane 2 can complete the jobs  $j_1, j_2, \dots, j_k$  in  $hs$  time.
- **Step 2:** we assign job  $n + 2$  to crane 2. Similarly, we assign jobs  $j'_1, j'_2, \dots, j'_{k'}$  to crane 1, where  $j'_i - 1 \in S_2, 1 \leq i \leq k'$ . Also, the non-crossing constraint is not considered, as crane 2 is doing the last job, and the labels of jobs done by crane 1 are all less than  $n + 2$ , crane 1 can complete the jobs  $j'_1, j'_2, \dots, j'_{k'}$  in  $hs$  time.

Hence all the  $n + 2$  jobs can be completed in  $2hs$  time if the set  $S$  can be partitioned to two equal subsets.

Conversely, suppose all the  $n + 2$  jobs can be done in  $2hs$  time, then both cranes are fully utilized as the sum of the processing time of all the jobs is  $4hs$ . Then we can conclude that job 1 and job  $n + 2$  must be done by crane 1 and crane 2 respectively (if job 1 is done by crane 2 or job  $n + 2$  is done by crane 1, the non-crossing constraint takes effect and the cranes could not be fully utilized). Assume jobs  $j_1, j_2, \dots, j_k$  are assigned to crane 1 in addition to job 1, jobs  $j'_1, j'_2, \dots, j'_{k'}$  are assigned to crane 2 in addition to job  $n + 2$ , then  $\sum_{i=1}^k p_{j_i} = \sum_{i=1}^{k'} p_{j'_i}$ , which means, the set  $S$  can be partitioned to two equal subsets  $S_1 = \{s_{j_1-1}, s_{j_2-1}, \dots, s_{j_k-1}\}$  and  $S_2 = \{s_{j'_1-1}, s_{j'_2-1}, \dots, s_{j'_{k'}-1}\}$ .

Hence the set  $S$  can be partitioned to two equal subsets if all the  $n + 2$  jobs can be completed in  $2hs$  time.

Thus the crane scheduling problem is *NP-hard* and it is *NP-complete* as it is in *NP* which is proven in the beginning of this section.

## 4 The Branch-and-Bound Algorithm

In this section, we shall discuss our branch-and-bound algorithm to solve the crane scheduling problem, as it is *NP-Complete*. Our algorithm can solve the problem optimally. In this section and next section, the following notations are used:

- $m$ : the number of cranes;
- $n$ : the number of jobs;
- $p_i$ : the processing time of job  $i$ ;
- $a_i$ : the assigned crane of job  $i$ ;
- $c_i$ : the completion time of job  $i$ ;

In the branch-and-bound search procedure, we make a decision of one job-to-crane assignment in each step (thus  $n$  steps are needed). So in every step, the  $n$  jobs can be divided into two disjoint sets: one set  $S_a$  that contains the jobs that have been assigned to cranes, and the other set  $S_u$  that contains the jobs that have not been assigned to cranes.

In each step, for each unassigned job  $i$  with respect to every crane  $k$ , we define the *Earliest Assignment Time*,  $EAT(i, k)$ , which indicates the earliest possible time if job  $i$  is assigned to crane  $k$  in this step. This value helps us to determine the two very important issues in branch-and-bound algorithm: the branching strategy and the bounding functions.

### 4.1 Branching Strategy

A naive branching method is to try each job-to-crane assignment ( $i \leftarrow k$ ),  $\forall i \in S_u, \forall 1 \leq k \leq m$ . However, as we observe, if the two job-to-crane assignments do not constrain each other, performing which assignment first does not matter. For example, assume the assignments ( $i \leftarrow k$ ) and ( $j \leftarrow l$ ) do not constrain each other, then we can perform ( $i \leftarrow k$ ) in the current step and perform ( $j \leftarrow l$ ) in the next step. Similarly, we can also perform the assignment of ( $j \leftarrow l$ ) followed by ( $i \leftarrow k$ ). Hence, we can enforce some job-to-crane assignment order to eliminate to speedup our search. The following two propositions can be used to reduce branches:

**Proposition 1.** Branch ( $i \leftarrow k$ ) can be eliminated if  $\exists j \in S_u, EAT(j, k) + p_j \leq EAT(i, k)$ .

**Proposition 2.** Branch ( $i \leftarrow k$ ) can be eliminated if  $\exists j \in S_a, i < j$  and  $k < a_j$ .

The first proposition enforces the order for the assignments pair  $(i \leftarrow k)$  and  $(j \leftarrow k)$ , given that the two assignments do not overlap each other in one crane. And the second proposition enforces the order for the assignments pair  $(i \leftarrow k)$  and  $(j \leftarrow a_j)$ , as long as they do not constrain each other by the non-crossing constraint.

With these two propositions, the number of branches can be reduced significantly thus making the algorithm more efficient.

## 4.2 Lower Bounds

In the algorithm, we consider the following two lower bounds:

- **Lower Bound 1:** Let  $CL_k$  denote the latest job completion time on crane  $k$  in the current step, i.e.,  $CL_k = \text{Max}\{c_i\}, i \in S_a, a_i = k$ , and  $C' = \text{Max}\{c_i\}, i \in S_a$  denotes the current latest completion time, then the following inequality holds:

$$C' + \text{Max}\{0, \lceil (\sum_{i \in S_u} p_i - \sum_{k=1}^m (C' - CL_k)) / |S_u| \rceil\} \leq OPT$$

Intuitively, the lower bound makes the assumption that all the remaining unassigned jobs are preemptive and the non-crossing constraint is ignored too. Then we just sum up the processing time of all the jobs and distribute it to all the cranes uniformly.

- **Lower Bound 2:** The second lower bound considers the jobs are non-preemptive, and each job is assigned to the crane that can complete it earliest, i.e., with the least  $EAT$  value. It can be expressed as follows:

$$\text{Max}_{i=1}^n \{\text{Min}_{k=1}^m \{EAT(i, k)\} + p_i\} \leq OPT$$

Both lower bounds can be computed efficiently ( $O(n)$  for the first bound and  $O(mn)$  for the second bound). We calculate the two lower bounds in each step of search to cut the branches.

This branch-and-bound algorithm is observed to be efficient as compared with the CPLEX solver in our experiments. The details are presented in the experimental results section.

## 5 A Simulated Annealing Heuristic Algorithm

In this section, we propose a simulated annealing (SA) heuristic algorithm which is based on the graph-searching based neighborhood moves to solve

the large scale crane scheduling problems in port logistics. We discuss the neighborhood moves and the SA algorithm in the following section.

### 5.1 Neighborhood Moves

Designing the neighborhood moves is an important component for many meta-heuristic methods, such as simulated annealing or tabu search. It is easy and flexible to design neighborhood moves for some *NP-hard* problems, as they have large solution space so that feasible solutions are easy to obtain by various neighborhood moves. However, our crane scheduling problem does not fall into this category. Consider a simple 3-crane and 10-job instance as an example (shown in Figure 2), we can not perform any classical neighborhood moves, such as *insertion*, *1-exchange*, or *2-exchange* moves, which are proposed in [10] to solve the Airport Gate Assignment Problem that has the similar data structures. Even the enhanced *interval exchange* move which is proposed in [4] cannot be used because of the non-crossing constraint.

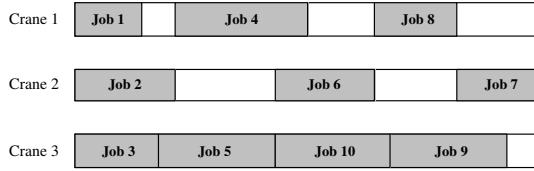


Figure 2: A Simple Problem Instance

In this paper, we develop a new neighborhood move which is based on the graph search. As we know, the objective of the neighborhood search is to find better solutions by reassigning or shifting a subset jobs to cranes. As the optimality criterion is to minimize the latest completion time  $C_{max}$ , the jobs that complete right at the time  $C_{max}$  would be the “trouble makers” (there may be one job or several jobs), i.e., they have to be shifted forward otherwise the objective value will not be improved. However, only shifting or reassigning those jobs is not possible. Using Figure 2 as an example, *Job 7* cannot be shifted forward because of the non-crossing constraint with *Job 8*; *Job 6* cannot be shifted forward because of the non crossing constraint with *Job 5*; *Job 9* cannot be shifted forward because it cannot overlap with *Job 10*, etc. Hence, shifting a job *a* forward may require cascading changes: we have to shift job *b* beforehand if *a* is constrained by *b*; then job *c* and job *d* must be shifted before *b* if *b* is constrained by *c* and *d*, ..., until reaching the

jobs that are not constrained by others. Thus, an effective neighborhood move must reassign jobs that constrain the movement of the jobs with completion time  $C_{max}$ .

To represent the relationship between jobs clearer, and to ease our search, we define the *Job Constraining Graph (JCG)*, which is a directed graph depicting constraining relationships among jobs. Each node in *JCG* corresponds to one job; the directed edge  $(i, j)$  exists if and only if job  $i$  is constrained by job  $j$ , i.e., job  $i$  could not start before job  $j$  completes when the non-crossing constraint (or overlapping in one crane) is taken into account. Figure 3 shows the *JCG* for the example in Figure 2.

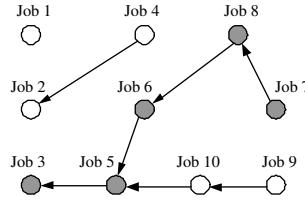


Figure 3: A Job Constraining Graph Example

A set of properties for *JCG* is given below:

- Each solution corresponds to one *JCG*, hence different solutions have different *JCGs*, even those solutions are from one problem instance;
- *JCG* is a directed acyclic graph (DAG);
- Each node in *JCG* with out-degree 0 (i.e. they are not constrained by any other jobs) corresponds to a job starting at time 0;
- All the “trouble makers” (i.e. they complete at time  $C_{max}$ ) correspond to nodes in *JCG* with in-degree 0; but the converse may not hold (see job 1, 4 and 9 in the example).

With the aid of *JCG*, it is easy to detect the subset of jobs that constrain the “trouble makers”. We can perform breath first search from the “trouble maker” nodes along the directed edges all the way to the nodes with no outgoing edges. As shown in Figure 3, the gray nodes are the nodes we identify by using the breath first search from the “trouble maker” node *job 7*.

After obtaining the subset of nodes after the breath first search on  $JCG$ , we remove them from the original solution and insert them back after re-ordering. The insertion follows the greedy fashion: for each job, we insert it to the earliest possible place among all the cranes. During the insertion, all the jobs that are not removed stay fixed. Of course, different solutions could be obtained if the insertion order varies. In implementation, we try to perform the insertion several times and the one with best cost is selected.

Our neighborhood moves guarantee the feasibility of the obtained neighborhood solutions, which is not easy to achieve by using commonly used neighborhood moves. Also, the solutions are diverse as we perform stochastic reordering.

## 5.2 Simulated Annealing Algorithm Framework

We can now describe our SA approach. In each iteration of SA, the above mentioned neighborhood search is performed and one move is generated. Each neighborhood move is accepted if it improves the objective function value. Other possible solutions are also accepted according to a probabilistic criterion. Such probabilities are based on the annealing process and they are obtained as a function of system temperature.

The SA heuristic framework that we employ is as follows:

1. Get an initial solution  $x_{now}$ . Set  $x_{best} = x_{now}$ ;
2. Set the annealing temperature  $T$ ;
3. Perform the neighborhood search and generate a move, calculate the delta cost  $\Delta$ ;
4. Decide whether to perform the neighborhood move generated, with the probability,  $p_o = a * \exp(-\Delta/(k * T))$ , where constants  $a$  and  $k$  determine the accept rate. If the move is performed and the cost is smaller than  $x_{best}$ , update  $x_{best}$ ;
5. Decrease  $T$  by a cooling rate factor  $d$ ;  $T = T * d$ . If  $T$  does not meet the termination criteria, go to Step 3;

The SA approach we apply here is standard SA with simple geometric cooling schedule is used. Quality solutions can be obtained by this SA algorithm with graph-searching based neighborhood search, as supported by the experimental results which are presented in the next section.

## 6 Experimental Results

We conduct a series of experiments to test the correctness and efficiency of our Branch-and-Bound (B&B) and SA algorithms. Both algorithms are coded in Java and run in Pentium IV 1.6GHz machine. As comparison, we use CPLEX 7.1.0, which is installed in Pentium IV 2.5GHz machine, to solve the formulation presented in Section 2. The initial temperature  $T$  of SA is 100, and the cooling rate  $d$  is 0.999. The SA process terminates if  $T$  drops to  $10^{-5}$  or the maximum number of iterations exceeds  $10^4$ .

We design several categories of instances to test various aspects of our algorithms.

### 1. Random Instances with Small Sizes

We create 10 instances with small sizes. The processing time of jobs are randomly generated in the interval [10, 40]. We run the CPLEX solver, branch-and-bound and SA algorithms using the 10 instances and the results are shown in Table 1. We observe that our B&B algorithm runs much faster than CPLEX solver, though both are exact method; and SA can obtain optimal solutions in short time. We believe that the performance of CPLEX is not good because it employs the general branch-and-bound method while our B&B is specific to this problem; also, there are too many constraints in the formulation that slow down the computation (instances 5 and 10, indicated by \*, make the program ran out of memory, hence, only the feasible solutions, which may not be optimal, are indicated).

### 2. Special Designed Instances with Small Sizes

We designed 10 instances with small sizes. In each of the instances, the processing time of one job is very large (500–1000) and others are all small (10–40). Hence the optimal value is just that large processing time (if there are more than two cranes), as we can complete all the other jobs while the job with large processing time is being handled. Table 2 shows the test results. Our B&B obtains all the solutions instantly taking the advantage of our bounding functions, while CPLEX still consumes much time. SA takes longer time than B&B as it does the iterative improvement, but the solutions it obtains are all optimal.

### 3. Random Instances with Medium Sizes

10 medium sizes instances are generated to compare the B&B algorithm and SA, as CPLEX cannot handle such test data. The pro-

cessing time is generated in the interval [10, 50]. Table 3 reports the solution. We can see that SA is still able to obtain optimal solutions, and its running time is superior to the B&B method.

#### 4. Random Instances with Large Sizes

To test the performance our SA algorithm thoroughly, we conduct experiments using groups of large instances. We select three  $m$  values (the number of cranes): 5, 10 and 15; for each  $m$  value, we choose the  $n$  values (the number of jobs) as 5–10 multiple of  $m$  respectively; and for each  $m$  and  $n$  values combination, 10 instances are generated. In total, the 180 instances which are divided into 18 groups. The processing time for each job generated randomly from the interval [50, 150].

We compare our SA results with the lower bounds which are calculated using the two bounding functions mentioned in section 3. Table 4 shows the summarized results. Note that all the values in the table are the average of results among 10 instances in those groups. The excessive percentage ( $exp$ ) is calculated as follows:  $exp = (V_{SA} - V_{LB})/V_{LB} * 100\%$ .

All the results are very close to their lower bounds, especially for the first few groups. We believe the solutions are very near to optimal. The other interesting observation is that for all three groups, given the same number of cranes, the  $exp$  percentage becomes lesser when the number of jobs increases.

## 7 Conclusion

The non-crossing constraint, which is an important and practical spatial constraint, is considered in the crane scheduling problem and studied in this paper. We showed that the problem is NP-complete and provided an integer programming model for the problem. We devised a branch-and-bound algorithm to solve the instances of moderate sizes and showed the our approach outperforms the CPLEX solver for this problem using the mathematical model. We also devised a new graph-search based neighborhood search and used it in the simulated annealing framework to tackle large instances of the problem. The results obtained by our simulated annealing algorithm are optimal or near optimal.

Instance No.	Size ( $n \times m$ )	CPLEX		B&B		SA	
		value	CPU(sec)	value	CPU(sec)	value	CPU(sec)
1	$8 \times 3$	71	105.45	71	0.063	71	8.687
2	$9 \times 3$	86	84.36	86	0.250	86	10.688
3	$10 \times 3$	85	783.70	85	1.250	85	13.203
4	$11 \times 3$	95	2316.38	95	5.047	95	13.765
5	$12 \times 3$	94*	19869.20	94	41.828	94	16.656
6	$8 \times 4$	62	17.55	62	0.047	62	8.203
7	$9 \times 4$	63	59.22	63	0.094	63	10.484
8	$10 \times 4$	66	3305.31	66	0.500	66	11.641
9	$11 \times 4$	60	7955.19	60	3.078	60	14.719
10	$12 \times 4$	86*	5330.89	81	12.875	81	16.219

Table 1: Results on Random Instances with Small Sizes

## References

- [1] Ebru K. Bish, Thin-Yin Leong, Chung-Lun Li, Jonanthan W.C.Ng, and David Simchi-Levi. Analysis of a new vehicle scheduling and location problem. *Naval Research Logistics*, 48(5):1002–1024, 2001.
- [2] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.
- [3] Carlos F. Daganzo. The crane scheduling problem. *Transportation Research*, 23B(3):159–175, 1989.
- [4] Haoning Ding, Andrew Lim, Brian Rodrigues, and Yi Zhu. New heuristics for the over-constrained airport gate assignment problem. *Journal of the Operational Research Society*, to appear, 2002.
- [5] Sabria F. and Carlos F. Daganzo. Queuing systems with scheduled arrivals and established service order. *Transportation Research*, 23B(3):159–175, 1989.
- [6] Peng-Hong Koh, Jimmy L.K. Goh, Hak-Soon Ng, and Hwei-Chiat Ng. Using simulation to preview plans of a container port operations. In *Proceedings of the 1994 Winter Simulation Conference*, pages 1109–1115, 1994.

Instance No.	Size ( $n \times m$ )	CPLEX		B&B		SA	
		value	CPU(sec)	value	CPU(sec)	value	CPU(sec)
1	16 × 3	500	33.23	500	0.031	500	23.563
2	17 × 3	400	165.63	400	0.015	400	34.062
3	18 × 3	675	119.77	675	0.015	675	30.109
4	19 × 3	1375	263.69	1375	0.031	1375	32.266
5	20 × 3	1643	127.03	1643	0.015	1643	54.266
6	16 × 4	1700	68.11	1700	0.016	1700	28.531
7	17 × 4	1200	543.23	1200	0.031	1200	70.969
8	18 × 4	1500	2977.53	1500	0.016	1500	53.687
9	19 × 4	1976	102.52	1976	0.047	1976	79.953
10	20 × 4	2700	165.36	2700	0.015	2700	72.719

Table 2: Results on Special Designed Instances with Small Sizes

Instance No.	Size ( $n \times m$ )	B&B		SA	
		value	CPU(sec)	value	CPU(sec)
1	13 × 3	122	271.313	122	17.218
2	13 × 3	113	250.547	113	17.500
3	13 × 4	98	109.078	98	21.485
4	13 × 4	94	174.172	94	20.235
5	14 × 3	136	1728.578	136	19.453
6	14 × 3	130	2132.062	130	19.719
7	14 × 4	112	939.265	112	20.641
8	14 × 4	106	1010.578	106	23.172
9	15 × 3	141	14134.297	141	24.422
10	15 × 4	102	3390.250	102	25.531

Table 3: Results on Random Instances with Medium Sizes

Group No.	Size ( $n \times m$ )	value	exp (%)	CPU (sec)	Group No.	Size ( $n \times m$ )	value	exp (%)	CPU (sec)
1	$25 \times 5$	494	1.63	708	10	$80 \times 10$	814	1.70	9292
2	$30 \times 5$	594	0.92	1015	11	$90 \times 10$	922	1.75	12912
3	$35 \times 5$	698	0.72	1470	12	$100 \times 10$	1022	1.33	16616
4	$40 \times 5$	814	0.44	2007	13	$75 \times 15$	537	6.27	8284
5	$45 \times 5$	926	0.54	2653	14	$90 \times 15$	637	6.88	12186
6	$50 \times 5$	980	0.52	3404	15	$105 \times 15$	724	4.05	17620
7	$50 \times 10$	524	3.03	3185	16	$120 \times 15$	832	3.71	25741
8	$60 \times 10$	613	2.10	4995	17	$135 \times 15$	928	2.81	34239
9	$70 \times 10$	720	2.10	7009	18	$150 \times 15$	1016	1.74	43080

Table 4: Results on Random Instances with Large Sizes

- [7] Andrew Lim, Brian Rodrigues, Fei Xiao, and Yi Zhu. Crane scheduling using tabu search. In *Proceedings of International Conference on Tools with Artificial Intelligence, (ICTAI 2002, Washington, USA)*, pages 146–153, 2002.
- [8] Andrew Lim, Brian Rodrigues, and Yi Zhu. Crane scheduling using swo with local search. In *Proceedings of 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL 2002, Singapore)*, 2002.
- [9] Roy I. Peterkofsky and Carlos F. Daganzo. A branch and bound solution method for the crane scheduling problem. *Transportation Research*, 24B(3):159–172, 1990.
- [10] Jiefeng Xu and Glenn Bailey. The airport gate assignment problem: Mathematical model and a tabu search algorithm. In *Proceedings of 34th Hawaii International Conference on System Sciences*, 2001.