Crane Scheduling with Spatial Constraints

Andrew Lim[1], Brian Rodrigues[2], Fei Xiao[3], and Yi Zhu[1]

[1] Department of Industrial Engineering and Engineering Management

The Hong Kong University of Science and Technology

Clear Water Bay, Kowloon, Hong Kong

[2]School of Business

Singapore Management University

469 Bukit Timah Road, Singapore 259756

[3]Department of Computer Science

National University of Singapore

3 Science Drive 2, Singapore 117543

Abstract

In this work, we examine port crane scheduling with spatial and separation constraints. Although common to most port operations, these constraints have not been previously studied. We assume that cranes cannot cross, there is a minimum distance between cranes and jobs cannot be done simultaneously. The objective is to find a crane-to-job matching which maximizes throughput under these constraints. We provide dynamic programming algorithms, a probabilistic tabu search and a squeaky wheel

optimization heuristic for solution. Experiments show the heuristics perform well compared with optimal solutions obtained by CPLEX for small scale instances where a squeaky wheel optimization with local search approach gives good results within short times.

1 Introduction

The Port of Singapore Authority (PSA) is a large port operator located in Singapore, one of the busiest ports in the world. PSA handles 17.04 million TEU's annually or nine percent of global container traffic in Singapore, the world's largest transshipment hub. PSA is concerned with maximizing throughput at its port due to limited port size, high cargo transshipment volumes and limited physical facilities and equipment [1, 10].

Crane scheduling and work schedules are critical in port management since cranes are at the interface between land and water sections of any port, each with its own traffic lanes, intersections, and vehicle flow control systems. In this multi-channel interface we are likely to find bottlenecks where cranes and other cargo-handling equipment (forklifts, conveyors etc.) converge.

Sabria and Daganzo [5] studied port operations which focused on berthing and cargo-handling systems. In berthing, which is a widely-analyzed port activity, queuing theory has been used widely. Traffic and vehicle-flow scheduling on land in ports has also been well studied [1]. Danganzo [4] studied a static crane scheduling case where cranes could move freely from hold to hold and only one crane is allowed to work on one hold at any one time. The objective was to minimize the aggregate cost of delay. In [13], container handling is modelled as "work" which cranes perform at constant rates and cranes can interrupt work without loss of efficiency. This constituted an "open shop" parallel and identical machines problem, where jobs consist of independent, single-stage and pre-emptable tasks. A branch-and-bound method was used to minimize delay costs for this problem. Crane scheduling has also been studied in the manufacturing environment context (see, for example [11, 12, 6]).

Commonly-found constraints affecting crane operations are absent in studies available on

the subject (see, for example, [13]). Such constraints affect crane work scheduling and need to be factored into operational models. These include the basic requirement that operating cranes do not cross over each other. Also, a minimum separating distance between cranes is necessary since cranes require some spatial flexibility in performing jobs. Finally, there is a need for jobs arriving for stacking at yards to be separated in arrival time to avoid congestion.

We found that operational decision-making at PSA was based largely on experience and simulation techniques [10]. While the latter is of value, analytic models are an advantage and are not limited by experience-generated rules-of-thumbs or simulation [4]. The object of this work is to address the need for such models which take into account common spatial and separation requirements in the scheduling cranes. This work augments Peterkofsky and Daganzo study [13].

2 Problem Description

During the time ships are berthed, various cargo-handling equipment is used to unload cargo, mostly in the form of containers. Different types of cargo require different handling and many ports have bulk, container, dry and liquid-bulk terminals. Cargo that is containerized can be loaded and unloaded in a fewer number of moves by cranes operating directly over ship holds or by crane arms moving over holds or deck areas.

Cargo stacked in yards is moved by cranes onto movers and transported for loading onto ships. "Cargo" here comprises containers of different capacities [1], which, whether in ships or in yards, are parcelled into fixed areas for access to cranes. For example, cargo placed in specific holds or deck sections on ships, or in sections within yards.

In this study, we focus on quay cranes and on cargo processed only from ships. We call cargo from a given area on a ship, a job. This is depicted in Figure 1. More generally, a job is a collection of cargo to be unloaded. Each crane is assigned a "job" and is assumed to be capable of handling all sizes of containers within given jobs. We assume that once each

crane is assigned a job it completes the job and that cranes do not move from one job to another, i.e., there is no crane-to-job variation within the time under consideration. Jobs cannot be shared among cranes, i.e., each job is serviced by only one crane at any time.

Containers are unloaded from ships by quay cranes onto movers or trailers which carry them to assigned yard locations where they are loaded onto stacks by yard cranes. Containers destined for import are set aside, and restacking, if required, is carried out. In the movement of containers, sequencing is crucial because containers are stored in stacks in the ship and on the yard and lanes may be designated to specific trailers at certain times. In addition, the movement of containers involves routing and crane operations where timings may be uncertain. In fact, crane scheduling is one activity among many that determine the movement of containers. Other such activities include berthing, yard storage, ship stowage and vehicle allocation and routing, all of which can be uncertain. Because of the uncertainty present over all activities, it is almost impossible to implement a plan over any length of time. This difficulty is present in scheduling cranes. For example, although a set of jobs may be assigned to a certain crane, it may not be possible for the crane to complete processing a job in this set onto movers once it was known that the route these movers are to take was congested. As another example, although we can specify that jobs bound for the same yard space are not unloaded from ships simultaneously, we cannot expect such containers to be unloaded at a time other than the allotted time interval, since a required resource to complete the job may become unavailable after this time, as for example, if the yard crane becomes unavailable. In view of the dynamically changing environment, a central control devises and maintains a job assignment plan that is periodically updated in order to coordinate operations, including crane scheduling. The system will allocate all jobs and resources periodically.

In the port we studied, a job parcel can include a number of ships and a number of cranes together with jobs. Typically, there can be up to five ships with four to seven cranes per ship and a number of jobs depending on the size and configuration of ships. Jobs have a profit value assigned to them and resources, e.g., cranes, movers, lanes etc., are assigned to each of the jobs depending on their value to the overall operations plan which aims to optimize total throughput. When an assignment plan is updated, the central system reassesses the

current state of operations to regroup and reassign job parcels. Because of this, time is accommodated by constant adjustments of job parcels and assignments based on the current state of all operations. Hence, once jobs and resources are assigned for the time period no update is necessary.

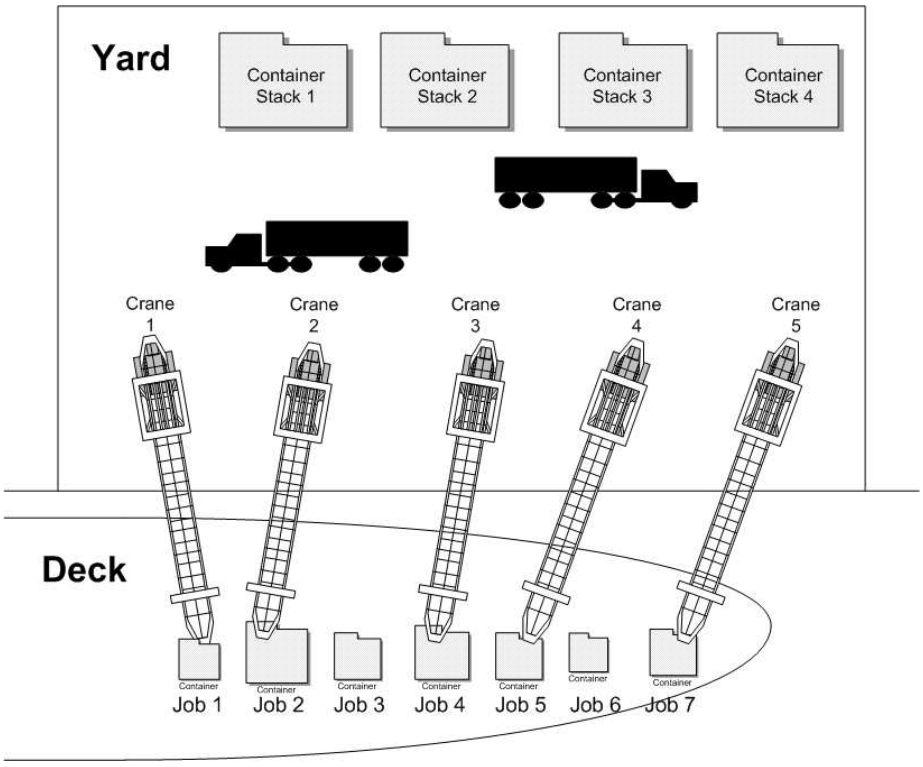

Figure 1: Crane to Job Topology

Jobs come in different sizes, and cranes have different handling capacities. Since we make the assumption that any crane assigned to a job completes it, the throughput or profit, for a given crane-to-job assignment, is a fixed value independent of other crane-to-job assignments.

The problem is naturally represented by a bipartite graph matching problem when we take cranes and jobs to be the vertices and define the weights of connecting edges to be crane-to-job throughput. This representation is shown in Figure 2.

This matching problem is interesting because, in practice, a number of spatial constraints arise for cranes and jobs. We first introduce qualitative notions of three particularly common

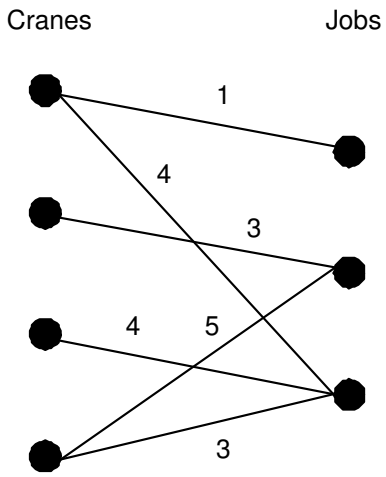

Figure 2: Bipartite Graph Representation

constraints which we call "spatial" constraints since they are related to the relative positions of cranes and jobs. Our objective is to find a crane-to-job assignment scheme which maximizes throughput under these constraints. For reasons given above, we assume that crane-to-job assignments are performed in a given time interval, i.e., there is no temporal component in the problem. Detailed definitions will be given in the relevant sections of this paper.

1. Non-crossing constraint: Cranes cannot cross over each other. This is a structural constraint on cranes and crane tracks.

2. Neighborhood constraint: There is a minimum distance between cranes. This arises, for example, since cranes require flexibility in space to perform jobs and/or for safety reasons. The effect of this constraint is that neighboring jobs may be affected and may not be assignable to other cranes.

3. Job-separation constraint: Certain jobs cannot be done simultaneously. For example, jobs bound for the same yard may require separation in time to avoid trailer congestion in lanes.

In the following sections, we first consider these constraints separately and then simultaneously. In section 3, an $O(mn)$ dynamic programming (DP) algorithm is given to solve the problem with only the Non-crossing constraint where m is the number of cranes and n is the number of jobs. In section 4, we use an $O(m^2n)$ dynamic programming algorithm to achieve an optimal solution for the problem with both the Non-crossing and Neighborhood constraints. In section 5, assuming all three spatial constraints, we show the problem to be NP-complete and give two heuristic approaches to solve the problem — a probabilistic tabu search and a squeaky wheel optimization with local search method. In section 6, we provide experimental results and compare the different approaches.

3 Scheduling with the Non-Crossing Constraint

3.1 The Problem

Throughout this work, $C = \{c_1, c_2, \ldots, c_m\}$ is a set of cranes and $J = \{j_1, j_2, \ldots, j_n\}$ a set of jobs. The order of subscripts assigned to the cranes and jobs represents their spatial (assumed linear) distribution, i.e., the neighbor of j_1 is j_2, the neighbors of j_2 are j_1 and j_3, $\ldots$, and the neighbor of j_n is j_{n-1}, The same holds for the cranes.

An $m \times n$ adjacency matrix, W, is used to represent the relationships between jobs and cranes. For each $W_{x,y} \in W$, the value $W_{x,y}$ represents the throughput when job j_y is assigned to crane c_x where $W_{x,y} = 0$ if job j_y cannot be assigned to crane c_x. The $W_{x,y}$ values arise from the different job sizes and crane capacities.

We seek a solution set, $R = \{(p,q) | 1 \leq p \leq m, 1 \leq q \leq n, W_{p,q} > 0\}$, such that the following constraint is satisfied: For all $(p_1, q_1), (p_2, q_2) \in R$, $p_1 < p_2$ if and only if $q_1 < q_2$. Viewing p's and q's, as subscripts in C and J respectively, we see that any crane-job assignment in R satisfies the Non-crossing Constraint.

The objective is then to find a set R which maximizes $\sum_{(p,q)\in R} W_{p,q}$ subject to the constraints that each job is assigned to at most one crane and each crane is assigned to at most

one job.

3.2 Algorithm Description

We now provide a dynamic programming (DP) approach [3] and describe how to characterize an optimal solution. DP procedures for computing values of solutions in a bottom-up way and for constructing solutions from computed information are omitted since they follow directly and are required only in implementation.

3.2.1 The Structure and Value of an Optimal Solution

We consider the cranes one by one. For each crane c_x, we assign every job j_y $(1 \leq y \leq n)$ to it and compute the total throughput to derive a partial optimal solution $P_{x,y}$ which denotes the optimal value up to the step we assign job j_y to crane c_x. Here, it is not necessary that job j_y is actually assigned to crane c_x, i.e., $(x, y) \in R_{x,y}$ may not hold, where $R_{x,y}$ is the partial solution set corresponding to the partial optimal solution $P_{x,y}$.

The following computes the partial optimal solution, $P_{x,y}$, recursively, for the different cases:

1. If $x = 1$ and $y = 1$, $P_{1,1} := W_{1,1}$
2. If $x = 1$ and $y > 1$, $P_{1,y} := \max\{W_{1,y}, P_{1,y-1}, \}$
3. If $x > 1$ and $y = 1$, $P_{x,1} := \max\{W_{x,1}, P_{x-1,1}\}$
4. If $x > 1$ and $y > 1$, $P_{x,y} := \max\{P_{x,y-1}, P_{x-1,y}, P_{x-1,y-1} + W_{x,y}\}$

(1) is the basic case: If we only consider the first crane and the first job, we will assign this job to the crane if the job can be done by the crane. (2) and (3) are both special cases, i.e., when there is only one node in each part of the bipartite graph. As these are symmetrical, we need consider only (2). For crane c_1 and job j_y, we have two choices: either, assign j_y to c_1, or, assign a job from $\{j_1, \ldots, j_{y-1}\}$ to c_1. This is because at most one job can be assigned

to this first crane. The throughput for the first choice is $W_{1,y}$ while the throughput for the second choice is $P_{1,y-1}$, which represents the maximum throughput if we assign a job among $j_1, j_2, \ldots, j_{y-1}$ to crane c_1. To achieve the cumulative optimal, we choose the larger of these. (4) is the general case in the DP algorithm. Figure 3 illustrates the situation. For c_x and job j_y $(x > 1, y > 1)$, we have three choices:

- Leave job j_y unassigned ((A) in Figure 3). We are reduced to assigning cranes $c_1, c_2, \ldots, c_x$ to jobs $j_1, j_2, \ldots, j_{y-1}$. By induction, the optimal value is then $P_{x,y-1}$;
- Leave crane c_x unassigned ((B) in Figure 3). We are reduced to assigning cranes $c_1, c_2, \ldots, c_{x-1}$ to jobs $j_1, j_2, \ldots, j_y$. By induction, the optimal value is then $P_{x-1,y}$;
- Assign crane c_x to job j_y (or, leave both unassigned if they are not assignable to each other), ((C) in Figure 3, where the bold line indicates actual assignment). In this case, the total throughput is the throughput from this assignment plus the throughput from assigning cranes $c_1, c_2, \ldots, c_{x-1}$ to jobs $j_1, j_2, \ldots, j_{y-1}$. Hence, the value is $P_{x-1,y-1} + W_{x,y}$.

Taking the maximum of these throughput values, the optimal solution is then the final partial optimal solution $P_{m,n}$ obtained.

3.2.2 A Proof of Optimal Substructure

We provide an outline a proof that the problem defined in this section possesses optimal substructures necessary in using DP. An important property for $P_{x,y}$ is: $P_{x,y} \geq P_{x',y'}$ if $x \geq x'$ and $y \geq y'$ (*), which is easily verified since $P_{x,y} \geq P_{x,y-1}$ and $P_{x,y} \geq P_{x-1,y}$. We can now verify the four cases given above by induction:

1. If $x = 1$ and $y = 1$, clearly $P_{1,y} = W_{x,1}$ is the only solution and must be optimal
2. If $x = 1$ and $y > 1$, $P_{1,y} = \max\{W_{1,y}, P_{1,y-1}\}$. Assume there is an optimal solution $P'_{1,y} > P_{1,y}$. Since $x = 1$, we know $P'_{x,y} = W_{1,k}$, where $1 \leq k \leq y$ (job j_k is assigned to the first crane). If $k = y$, then $P'_{1,y} = W_{1,y}$. Since we know $P_{1,y} = \max\{W_{1,y}, P_{1,y-1}\} \geq$

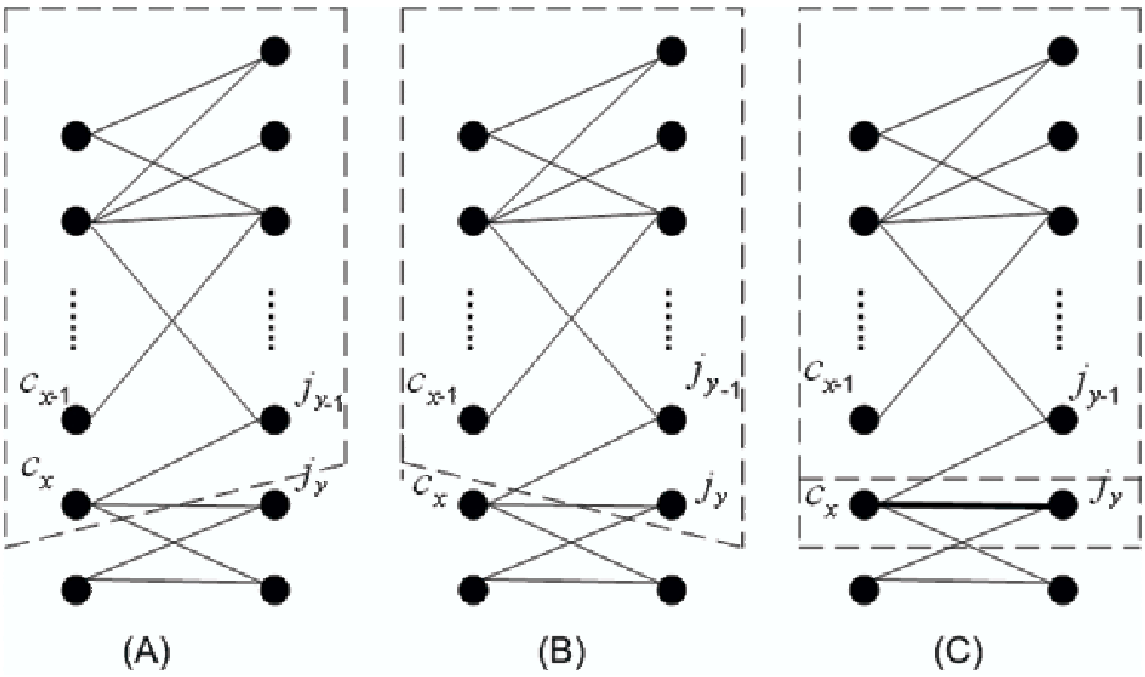

Figure 3: Illustration of DP Induction

$W_{1,y}$, $P_{1,y} \geq P'_{1,y}$ and this contradicts the assumption $P'_{1,y} > P_{1,y}$. If $1 \leq k \leq y-1$, we know that $P'_{1,y} = W_{1,k} = P_{1,k}$. Since $y - 1 \geq k$, $P_{1,y-1} \geq P_{1,k}$ (by (*)), we have $P_{1,y-1} \geq P'_{1,y}$. Since $P_{1,y} = \max\{W_{1,y}, P_{1,y-1}\} \geq P_{1,y-1}$, we have $P_{1,y} \geq P'_{1,y}$, which contradicts the assumption $P'_{1,y} > P_{1,y}$. Hence $P_{1,y}$ is the optimal solution

3. If $x > 1$ and $y = 1$, $P_{x,y} = \max\{P_{x-1,1}, W_{x,1}\}$. The proof is symmetric to (2) above

4. If $x > 1$ and $y > 1$, $P_{x,y} = \max\{P_{x,y-1}, P_{x-1,y}, P_{x-1,y-1} + W_{x,y}\}$. We assume there is an optimal solution $P'_{x,y} > P_{x,y}$ and the solution set corresponding to $P'_{x,y}$ is $R'_{x,y} = \{(c_{a_1}, j_{b_1}), (c_{a_2}, j_{b_2}), \ldots, (c_{a_k}, j_{b_k})\}$ which is ordered, i.e., $a_1 \leq a_2 \leq \ldots \leq a_k$ and $b_1 \leq b_2 \leq \ldots \leq b_k$, by virtue of the Non-crossing constraint. There are now two possibilities for $P'_{x,y}$:

   (a) If $(x, y) \notin R'_{x,y}$, then $P'_{x,y} \leq P_{a_k,b_k}$. By (*), we know $P_{x,y} \geq P_{a_k,b_k}$ since $x \geq a_k, y \geq b_k$. So $P_{x,y} \geq P'_{x,y}$, which contradicts the assumption $P'_{x,y} > P_{x,y}$. Hence $P_{x,y}$ is the optimal solution

(b) If $(x, y) \in R'_{x,y}$, then $P'_{x,y} \leq P_{a_{k-1},b_{k-1}} + W_{x,y}$. By (*), we know $P_{x-1,y-1} \geq P_{a_{k-1},b_{k-1}}$ since $x - 1 \geq a_{k-1}, y - 1 \geq b_{k-1}$. So $P'_{x,y} \leq P_{x-1,y-1} + W_{x,y}$. Because $P_{x,y} \geq P_{x-1,y-1} + W_{x,y}$, we get $P_{x,y} \geq P'_{x,y}$, which contradicts our assumption $P'_{x,y} > P_{x,y}$. Hence, $P_{x,y}$ is the optimal solution.

We can conclude that $P_{x,y}$ is the optimal solution for all (x, y), $1 \leq x \leq m, 1 \leq y \leq n$,

3.3 The Time Complexity of the Algorithm

The computation for every partial solution $P_{x,y}$ is in constant time, so the time complexity for this algorithm is $O(mn)$.

4 Scheduling with the Neighborhood Constraint

4.1 The Problem

In this problem, both the Non-crossing constraint and the Neighborhood constraint are considered. In addition to the Non-crossing constraint, we use the set $S = \{s_1, s_2, \ldots, s_m\}$ to represent the Neighborhood constraint associated with the cranes. Here $s_x = k$ if crane c_x performs job j_y and job j_z $(a \leq z \leq b, z \neq y)$ cannot be worked on by any other crane, where $a = \max\{1, y - k\}$ and $b = \min\{y + k, m\}$. In other words, if crane c_x performs job j_y, the job "interval" centered at y with length $2k + 1$ is affected by crane c_x when $s_x = k$.

We seek a solution set $R = \{(p, q) | 1 \leq p \leq m, 1 \leq q \leq n, W_{p,q} > 0\}$ satisfying:

1. For all $(p_1, q_1), (p_2, q_2) \in R$, $p_1 < p_2$ if and only if $q_1 < q_2$ (Non-crossing constraint)
2. For all $(p, q) \in R$, if $1 \leq p' \leq m$ and $p' \neq p$, and $a \leq q' \leq b$, where $a = \max\{1, q - s_p\}$ and $b = \min\{q + s_p, n\}$, then $(p', q') \notin R$ (Neighborhood constraint)

Our objective is to find R that maximizes the total weight $\sum_{(p,q)\in R} W_{p,q}$ where each job is assigned to at most one crane and each crane is assigned to at most one job.

4.2 Algorithm Description

We follow the approach in section 3.2 here.

4.2.1 The Structure and Value of an Optimal Solution

We continue to consider the cranes one by one. For each crane c_x, we attempt to assign every job $j_y(1 \leq y \leq n)$ to it and compute the total throughput up to this step to give a partial optimal solution $P_{x,y}$.

Here, the partial optimal solution $P_{x,y}$ is cumulative and the edge inclusion $(x, y) \in R_{x,y}$ may not hold. However, different from the definition used in the previous section, crane x must be assigned some job j $(1 \leq j \leq y)$ for $P_{x,y}$, i.e., there must be an edge $(x, j) \in R_{x,y}$, where $(1 \leq j \leq y)$; if there is no job in the interval $[1, y]$ that can be assigned to crane x, then $P_{x,y} = 0$. Now, we define the value of the partial optimal solution $P_{x,y}$ for the different cases:

1. If $x = 1$ and $y = 1$, $P_{1,1} := W_{1,1}$
2. If $x = 1$ and $y > 1$, $P_{1,y} := \max\{W_{1,y}, P_{1,y-1}\}$
3. If $x > 1$ and $y = 1$, $P_{x,1} := W_{x,1}$
4. If $x > 1$ and $y > 1$, $P_{x,y} := \max\{P_{x,y-1}, P_{i,c} + W_{x,y}\}, 1 \leq i < x, c = y - \max\{s_x, s_i\} - 1$.

(1) is the basic case and (2) and (3) are the special cases. (2) has been explained in the previous section. (3) is different. Since we require for $P_{x,y}$ that crane c_x must be assigned job j_y, we have no choice but to assign this job to the current crane when there is only job available. The induction step in (4) is somewhat complex. Figure 4 illustrates the induction. When we consider crane c_x and job j_y, we either assign one of the jobs in $j_1, ..., j_{y-1}$ to crane c_x ((A) in Figure 4) or just j_y to c_x ((B) in Figure 4, where the bold lines indicate the actual assignments). In the first case, we keep job j_y unassigned, so we are reduced to assigning cranes $c_1, c_2, \ldots, c_x$ to jobs $j_1, j_2, \ldots, j_y$; hence, we obtain $P_{x,y-1}$. In the second case, since we assigned job j_y to crane c_x, the Neighborhood constraint for c_x must be considered. Also, we

must consider the Neighborhood constraint for the cranes $c_1, c_2, \ldots, c_{x-1}$ which are assigned to jobs. The Non-crossing constraint simplifies the computation leaving us only to check the neighborhood constraint for the "largest label" crane assigned and is the reason the c value is needed in the formula.

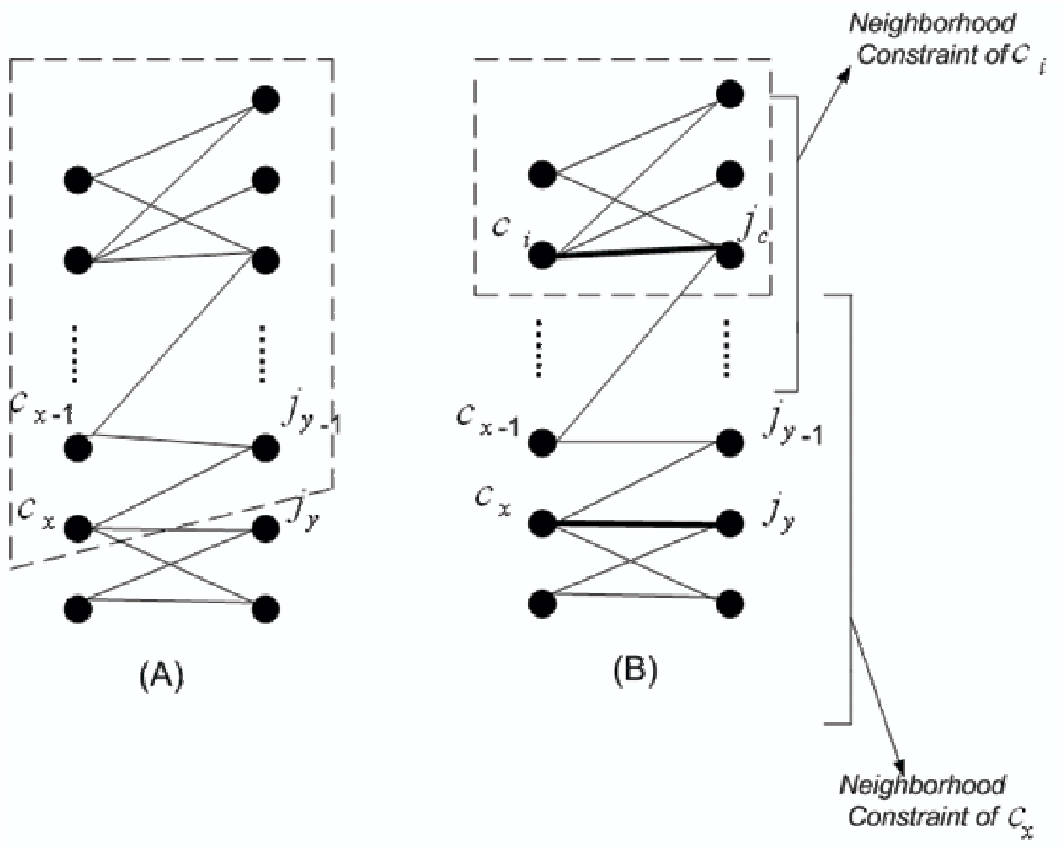

Figure 4: Illustration of DP induction

The final optimal is the maximum value of all partial optimal solutions obtained, i.e., it is $\max\{P_{x,y}\}$ over all (x, y), $1 \leq x \leq m, 1 \leq y \leq n$.

4.2.2 A Proof of Optimal Substructure

Similar to the earlier problem, we show that this problem has an optimal substructure. As before, we use the fact that $P_{x,y} \geq P_{x,y'}$ if $y \geq y'$ (**). This is easy to verify since a partial

optimal solution is cumulative for each crane x. As before, we have the following cases:

1. If $x = 1$ and $y = 1$, clearly $P_{1,1} = W_{1,1}$ is the optimal solution

2. If $x = 1$ and $y > 1$, only one crane is involved, so the Neighborhood constraint does not take the effect. The proof is then as given in the previous section

3. If $x > 1$ and $y = 1$, since crane c_x has to be assigned a job, and there is only one job, clearly $P_{x,1} = W_{x,1}$

4. If $x > 1$ and $y > 1$, $P_{x,y} = \max\{P_{x,y-1}, P_{i,c} + W_{x,y}\}, 1 \leq i < x, c = y - \max\{s_x, s_i\} - 1$. We assume there is an optimal solution $P'_{x,y} > P_{x,y}$ and the solution set corresponding to $P'_{x,y}$ is $R'_{x,y} = \{(c_{a_1}, j_{b_1}), (c_{a_2}, j_{b_2}), \ldots, (c_{a_k}, j_{b_k})\}$. Here, we can take this set to be ordered, i.e., $a_1 \leq a_2 \leq \ldots \leq a_k$ and $b_1 \leq b_2 \leq \ldots \leq b_k$, by virtue of the Non-crossing constraint. Noting $a_k = x$ from the definition above, there are now two possibilities for $P'_{x,y}$:

   (a) If $(x, y) \notin R'_{x,y}$, then $P'_{x,y} \leq P_{a_k,b_k}$, since P_{a_k,b_k} is the partial optimal solution ((A) in Figure 4). Hence, $P'_{x,y} \leq P_{x,b_k}$, since $a_k = x$. From property (**), we know $P_{x,y-1} \geq P_{x,b_k}$ since $y - 1 \geq b_k$ (job j_y is unassigned). So $P_{x,y-1} \geq P'_{x,y}$. By the recursive definition, $P_{x,y} \geq P_{x,y-1}$. Hence, we get $P_{x,y} \geq P'_{x,y}$, which contradicts the assumption $P'_{x,y} > P_{x,y}$. So $P_{x,y}$ is the optimal solution in this case

   (b) If $(x, y) \in R'_{x,y}$, then $P'_{x,y} \leq P_{a_{k-1},b_{k-1}} + W_{x,y}$, since $P_{a_{k-1},b_{k-1}}$ is the partial optimal solution. Obviously $1 \leq a_{k-1} < x$, so we let $i = a_{k-1}$ and $c = y - \max\{s_x, s_i\} - 1$ ((B) in Figure 4). We know $b_{k-1} \leq c$ since the cranes c_x and c_i are both assigned jobs and their Neighborhood constraints are in effect. From (**), we get $P_{i,c} \geq P'_{i,b_{k-1}}$because of $c \geq b_{k-1}$, i.e., $P_{a_{k-1},c} + W_{x,y} \geq P_{a_{k-1},b_{k-1}} + W_{x,y}$, and so $P_{a_{k-1},c} + W_{x,y} \geq P'_{x,y}$. From the definition $P_{x,y} = \max\{P_{x,y-1}, P_{i,c} + W_{x,y}\}$, we know $P_{x,y} \geq P'_{x,y}$, which contradicts the assumption $P'_{x,y} > P_{x,y}$. Hence, $P_{x,y}$ must be the optimal solution in this case

We conclude that $P_{x,y}$ is the optimal solution for all (x, y), $1 \leq x \leq m, 1 \leq y \leq n$.

4.3 The Time Complexity of the Algorithm

Since, for each partial solution $P_{x,y}$, we take the maximum value of the partial solutions $P_{i,c}, 1 \leq i < x$, the time complexity is $O(m^2 n)$.

5 Scheduling with the Job Separation Constraint

5.1 The Problem

We can now study the third and most general spatial constraint — the Job-separation constraint. An $n \times n$ matrix D represents this constraint: $D_{p,q} = 1(1 \leq i \leq n, 1 \leq j \leq n)$, when job j_p and j_q cannot be done simultaneously. Otherwise, the elements of D are 0. Note that D is symmetric. We seek a solution set $R = \{(p,q) | 1 \leq p \leq m, 1 \leq q \leq n, W_{p,q} > 0\}$, for which the following three conditions are satisfied:

1. For all $(p_1, q_1), (p_2, q_2) \in R$, $p_1 < p_2$ if and only if $q_1 < q_2$ (Non-crossing constraint)
2. For all $(p,q) \in R$, if $1 \leq p' \leq m$ and $p' \neq p$, and $a \leq q' \leq b$, where $a = \max\{1, q - s_p\}$ and $b = \max\{q + s_p, n\}$, then $(p', q') \notin R$ (Neighborhood constraint)
3. For all $(p,q) \in R$, if $1 \leq p' \leq m$, and $D_{q,q'} = 1$, then $(p', q') \notin R$ (Job-separation constraint)

The objective is to find a set R which maximizes total weight $\sum_{(p,q) \in R} W_{p,q}$ where each job is assigned to at most one crane and each crane is assigned to at most one job.

The problem can be formulated as an Integer Program (IP) described in Appendix A. This IP model will be used in the CPLEX solver as described in section 6.

5.2 Proof of NP-completeness

To show that this problem is NP-complete, we use the Independent Set problem which is defined as follows: Given a graph $G = (V, E)$ and a positive integer $k \leq |V|$, is there a

$V' \subseteq V$ such that for all $u, v \in V'$, the edge (u, v) is not in E and $|V'| \geq k$?

In order to prove that this problem is NP-hard, we transform an arbitrary instance of the Independent Set problem to the problem in polynomial time. Assuming there are n nodes in the graph $G = (V, E)$ of the Independent Set problem, we construct the model with n cranes and n jobs where the only edges are $(1, 1), (2, 2), \ldots, (n, n)$, all with weight equal to 1. The Job-separation constraint matrix D is defined as follows: For all $(x, y) \in E, D_{x,y} = 1$, otherwise $D_{x,y} = 0, 1 \leq x, y \leq n$. The transformation is illustrated in Figure 5 and can be achieved in polynomial time.

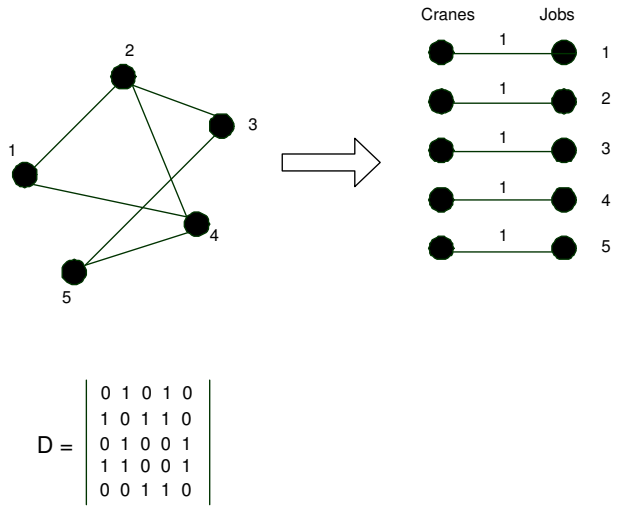

Figure 5: Transformation from the Independent Set Problem

Now we show that the Independent Set problem has a solution of size k if and only if the problem has a solution with total profit k. First, if there are k independent nodes in graph G, there must be k jobs that do not, pairwise, conflict. Since we constructed n parallel edges with weight 1, the Non-crossing constraint and Neighborhood constraint do not have any effect here. Hence, we can use k cranes to do the k jobs without violating the Job-separation constraint with total profit k. If we now assume that there is a solution in this problem with profit k, there must be k jobs selected without violating the Job-separation constraint. There must be k nodes that are not connected by any edges and therefore a set of nodes of size k.

We can verify solutions by checking crane-job assignments one by one for violation of the three constraints. Clearly, this can be done in polynomial time, so the problem is in the class NP. Since the problem has been shown to be NP-hard, it is NP-complete and, unless P = NP, there are no polynomial algorithms to solve it optimally. It would be useful therefore to develop heuristic solutions for the problem, which we do in the following sections.

5.3 A Probabilistic Tabu Search Approach

Tabu Search (TS) is a search procedure that iterates from one solution to another by moves in a neighborhood space with the help of an adaptive memory. Probabilistic Tabu Search (PTS) is a variant of TS, which places emphasis on randomization when compared with basic TS [7]. The basic approach is to create move evaluations that include references to the tabu status and other relevant biases from TS strategies using penalties, modifying underlying decision criterion and selecting the next move among those neighborhood moves with different probabilities which are based on different evaluation values [7]. In this section, we describe how it can be employed for the crane scheduling problem.

5.3.1 Neighborhood Structure

From an initial feasible solution obtained by a greedy method or a random crane-job assignment, the graph representation becomes almost edge "saturated", i.e. we can hardly add an edge without violating the Non-crossing, Neighborhood and Job-separation constraints. We can however delete an edge from the current solution and try to add other edges until it is "saturated" again. Deleting the edge which connects crane c and job j allows some cranes and jobs to become assignable. Obviously, these can only come from cranes and jobs which are neighbors to c and j, respectively, which do not violate the Non-crossing constraint w.r.t. all current assignments (discounting the c to j assignment). Jobs selected must also satisfy the Neighborhood and Job-separation constraints. After deleting the edge connecting c to j, we consider each neighbor of c from these feasible neighbors together with c, one by one. For each crane, we assign a probability p_1 for it to be selected for a job. For each selected crane,

we have two types of assignments: one is a greedy assignment which selects a compatible job with the largest weight; the other is a random assignment which randomly picks one job from all the compatible jobs. Which scheme is chosen depends on yet another probability, p_2.

5.3.2 Tabu Search Memory

TS memory structures guide the search process. There are two kinds of memory structures. One is "short-term memory", which can prevent the search from being trapped in a local optimum and the other is "long-term memory", which provides diversification and intensification.

Short-term memory restricts the composition of new solutions generated. If an edge is deleted in a move, we forbid its addition in the next few moves; similarly, if an edge is added in a move, we forbid its deletion in the next few moves. Such a mechanism prevents the search from revisiting local optima in the short term and reduces the chance of cycling in the long term.

How long a restriction is in effect depends on a tabu tenure parameter, which identifies the number of iterations a particular restriction remains in force [7]. We implemented short-term memory using a recency-based memory structure as follows. Let iter denote the current iteration number, tabu_add(x, y) and tabu_delete(x, y) denote future iteration values that forbid a reversal of the moves on adding edge (x, y) or deleting edge (x, y). Furthermore, let tabu_add_tenure and tabu_delete_tenure be the values of tabu tenure for these two moves. When the TS restriction is imposed, we update the recency memory by:

$$\text{tabu\_add}(x, y) = \text{iter} + \text{tabu\_add\_tenure}$$
$$\text{tabu\_delete}(x, y) = \text{iter} + \text{tabu\_delete\_tenure}$$

We assign positive penalties to edges in tabu status, which means they are forbidden by the recency memory.

A TS restriction is overridden by aspiration if the outcome of the move under consideration is sufficiently desirable. This can be achieved by deducting a large number from the

total penalty.

For Long-term TS memory, we count the number of times that a specific edge is to be deleted by updating the frequency counter, $f_{x,y} := f_{x,y} + 1$ when an edge (x, y) is deleted in a current move. In evaluation, we give a positive penalty to all edges in the current solution according to this frequency memory. At the same time, we also count the number of times that a specific edge exists in the solution by updating the frequency counter, $f'_{x,y} := f'_{x,y} + 1$ when an edge (x, y) is in the current solution. In contrast to the transition measure, we will assign a negative penalty in the evaluation based on this residency measure. This will encourage moves that keep the edges which frequently remain in past moves and have a relatively higher potential to provide good solutions.

5.3.3 Move Evaluation

After finding all neighborhood candidate moves, we evaluate these moves so that they are ready for selection. The resulting evaluation value is in two parts: the total profit and the penalties. For our crane scheduling problem, the penalties comprise: (1) Short-term memory penalties which include tabu status penalties as well as aspiration satisfaction "penalties" (2) Long-term memory penalties which include transition measure and residence measure penalties and (3) Penalties for other biases. The details for computing the value of a move can be found in Appendix B.

5.3.4 Probabilistic Move Selection

After evaluating all candidate moves, we select one move to proceed. The fundamental idea of move selection is to choose the best move, i.e., choose the move with the highest evaluation. However, we found that this greedy selection strategy is strongly biased. We therefore made adjustments using probabilities. The strategy (see also [15]) for a probabilistic move selection is given as follows:

1. Generate the candidate list and evaluate candidate moves which have described above

2. Select the move from the candidate list with the highest evaluation value

3. Accept the move with probability p and exit; otherwise, go to (4). Here, p is a parameter and is set to 0.3 in the algorithm

4. Remove the move from the candidate list. If the list is empty, accept the first move of the original candidate list and exit. Otherwise, go to (2).

It is easily shown that the probability of choosing one of the best k moves is $1-(1-p)^k$, which is large even if p is not large. For example, if p is 0.25, the probability of choosing the best 5 moves is 0.763, the best 10 moves is 0.944. We can therefore choose relatively high evaluation moves while avoiding favoring those with highest evaluations always.

5.4 SWO with Local Search Approach

"Squeaky Wheel" Optimization (SWO) is a general approach to optimization and consists of a Construct-Analyze-Prioritize cycle at its core [2][8]. As illustrated in Figure 6, a solution is constructed by the Constructor using a greedy algorithm. The Analyzer will assign a numerical "blame" value to the problem elements that contribute to shortcomings in the current solution. The Prioritizer will modify sequences of problem elements and elements that received blame are moved to the front of the sequence. The higher the blame, the further the element is moved. The Constructor deals with problem elements according to the modified sequence in the next iteration. The cycle repeats until a termination condition is satisfied.

The SWO algorithm has been effective in job scheduling and graph coloring problems and outperforms TS and integer programming [8] in some applications. Although SWO strives to avoid getting trapped in local optima by changing the sequence in the Prioritizer, blame values can trap SWO in small cycles. In SWO, there are tasks that must be handled badly locally in order to achieve a good overall solution. However, the Analyzer always assigns high blames to these tasks to force the Constructor handle them first in the next iteration which can be a disadvantage.

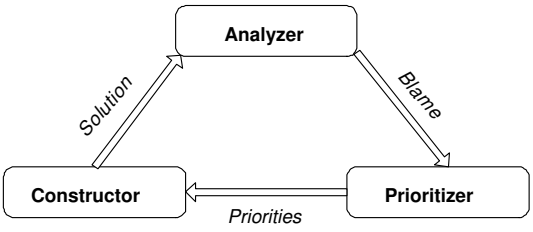

Figure 6: The Construct-Analyze-Prioritize cycle

To overcome these weaknesses, we developed an enhanced SWO framework. In this framework, we first obtain a greedy solution from the Constructor, perform a local search on the solution, and then try to find a better solution in the local solution space. The SWO cycle is augmented as Figure 7 shows:

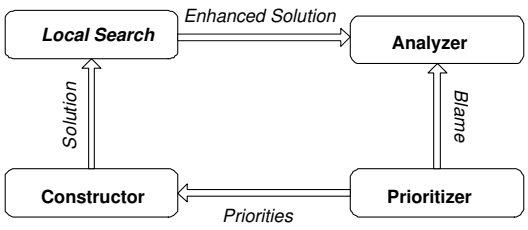

Figure 7: The Construct-Local Search-Analyze-Prioritize cycle

Local search can improve the SWO heuristic in the following ways:

- Unlike the traditional heuristic search methods such as TS and Simulated Annealing, SWO is sensitive to the sequence of problem elements rather than the objective function. Omitting the objective function may cause the final solution to be inferior. However, local search compensates this by causing moves to neighbors with higher objective function values.

- Because of the range of local search solutions, we can select neighbors based on different criteria so that the chance of getting trapped in a cycle is reduced greatly. We can also combine randomization techniques in local search.

- In many cases, local search can identify tasks that can be sacrificed to obtain a higher evaluation solutions. .

To illustrate the SWO with local search heuristic when applied to the crane scheduling problem, the following describes the four components of the heuristic, i.e., the Constructor, Local Search, the Analyzer and the Prioritizer.

The Constructor: The Constructor generates a solution using a greedy algorithm which assigns priorities to problem elements. We regard the cranes as the problem elements and try to assign a job to a crane one at a time in the order they occur in the priority sequence. Instead of assigning a job to a crane greedily each time, we use an alternative strategy which involves more randomness. For each crane, we have a selection threshold probability p_1 and for each selected crane, we have two means of assignment, one is a greedy assignment which selects a compatible job with the largest weight; the other is random assignment which randomly picks one job from compatible jobs. Here "compatible job" means we assign the job to the crane satisfying all the constraints. Which scheme is chosen depends on another probability p_2. The detailed algorithm can be found in Appendix C.

Local Search: In the local search component, we perform neighborhood search and enhance it by using meta-heuristic techniques [9, 14]. For neighborhood search, we continue to use the same scheme described in section 5.3.1. Here, after creating neighbors of the current greedy solution, we choose the one with the highest profit as the next solution and move to it. This will continue until a local optimum is reached or the number of steps stipulated is exceeded.

The Analyzer: The Analyzer assigns blame to each crane. How large the blame value is depends on how the current assignment affects the solution. In this problem, the blame value of a crane depends on how much the assignment contributes to the total profit, and also, how many jobs are "banned" by the assignment because of the spatial constraints. In the blame value calculation, both of the factors should be considered. The detailed calculation is described in Appendix D.

The Prioritizer: Once blame has been assigned, the Prioritizer modifies the previous

sequence of cranes. Cranes with higher blame values will move forward in the priority sequence while ones with smaller blame values will remain at the back of the sequence. Troublesome cranes will then be handled first by the Constructor in the next iteration.

6 Experimental Results

We implemented the four different algorithms – Hill Climbing with 100-restarts (HC), Probabilistic Tabu Search (PTS), Squeaky Wheel Optimization (SWO) and SWO with Local Search (SWO+L) – using C++ and ran on a P4 2.52 GHZ CPU with 1GB memory. The parameters used were: PTS - max iterations: 20000, tabu_add_tenure: 10, tabu_delete_tenure: 10, Delete Penalty Unit (DPU): 100, Add Penalty Unit (APU): 50, Aspiration Threshold (AST): 300, Transition Penalty Unit (TPU): 10, Residence Penalty Unit (RPU): 10, α: 2, β: 4, γ: 4, K: 50 (see Appendix B) Candidate selection probability (p): 0.3, SWO/SWO+L - max iterations: 2000, Restarts: 10, hill climbing iterations: 10. These were a result of parameter tuning obtained from running extensive experiments on small test cases.

For test sizes, we provide results here to reflect actual situations at the port. As stated, a job parcel can include a number of jobs to be processed in a given time interval, and these jobs can come from a number of ships and involve a number of cranes. Typically, depending on the size of the ship, between 4 to 7 cranes are assigned to a ship. The number of jobs required by any one ship can vary depending on the size and configuration of each ship and on how jobs are defined. A ship with four holds and a number of deck areas could well have nine different jobs, for example. Since, typically, a job parcel will not exceed five ships, we tested the algorithms on data representing not more than thirty cranes and forty jobs in the first instance. These results were compared with those from ILOG CPLEX 7.1 IP solver applied to the IP model for the problem given in Appendix A.

We ran 110 instances in group 1 but could only obtain results for the smallest 20 instances from the solver. The remaining 90 larger instances caused CPLEX to be out of memory. Detailed results using CPLEX and our heuristics for the 20 cases are listed in Table 1.

All running times were measured in seconds and we found that both CPLEX and heuristic algorithms were fast. All algorithms achieved optimal solutions in most of the instances and SWO+L obtained optimal solution for all 20 cases.

We generated a second group 2 of 20 instances whose sizes were between 10×30 and 30×40. These instances were used to observe how running time increased with size as well as to compare the quality of solutions. Results are given in Table 2. Here, "—" in the table indicates that CPLEX ran out of memory. The "gap" columns shows the difference between a heuristic solution and the optimal solution as a percentage of the latter.

CPLEX found 17 optimal solutions out of 20 instances. However, the running times for CPLEX increased significantly when the instances became larger, and it ran out of memory for the last three cases. Figure 8 plots the time increments for the first 10 instances (instances nos. 1-10). On the other hand, the four heuristic algorithms continued to obtain solutions in very short times although HC, PTS and SWO did not perform very well on larger cases. SWO+L, however, continued to perform well in time and quality of solutions. For group 2, it obtained 14 out of 17 optimal solutions and found good solutions for the last three cases for which CPLEX ran out of memory. These were achieved in very short times, all under five seconds.

As the constraints are tight in this problem, that is, for a given crane there were only a few jobs that were compatible, solutions neighborhood search can generate are restricted. HC is easily trapped at local optima and the quality of results confirm this. Similarly, PTS is based on the neighborhood search and is easily trapped in regions of the search space which can explain its poor performance. On the other hand, the SWO and SWO+L use both the solution space and priority space where a small change in the priority space can cause large changes in the solution space. SWO+L is the best of the four algorithms. The difference in performance between SWO and SWO+L is significant, and is attributable entirely to the local search component which is able to generate a diverse range of solutions. This is significant when compared to the purely greedy solution created by the Constructor used in SWO. Further, since all elements are assigned equal blame, there is a higher chance that local optima can be avoided. SWO+L , however, has the longest run times among the heuristics

Instance	Size	CPLEX		HC		PTS		SWO		SWO+L	
No.	$(m \times n)$	obj.	time	obj.	time	obj.	time	obj.	time	obj.	time
1	10 × 10	220	0	220	0	217	0	220	1	220	1
2	10 × 10	172	0	171	0	172	0	172	0	172	0
3	10 × 10	183	0	183	0	183	1	183	0	183	1
4	10 × 10	260	0	213	0	260	0	260	0	260	0
5	10 × 10	102	0	102	0	102	0	102	1	102	1
6	10 × 10	109	0	109	0	109	0	109	0	109	0
7	10 × 10	152	0	152	0	152	1	152	0	152	1
8	10 × 10	150	0	150	0	150	0	150	0	150	0
9	10 × 10	183	0	183	0	183	0	183	1	183	1
10	10 × 10	156	0	156	0	156	0	156	0	156	0
11	10 × 30	352	3	352	0	352	0	352	0	352	1
12	10 × 30	318	3	288	0	318	0	318	1	318	1
13	10 × 30	436	3	429	0	433	0	434	0	436	1
14	10 × 30	485	3	485	0	485	1	485	0	485	1
15	10 × 30	544	3	501	0	544	0	544	1	544	1
16	10 × 30	215	3	215	0	215	0	215	0	215	0
17	10 × 30	615	3	615	0	615	0	615	0	615	1
18	10 × 30	298	3	263	0	298	1	297	1	298	1
19	10 × 30	653	3	653	0	653	0	618	0	653	1
20	10 × 30	505	3	505	0	505	0	505	0	505	1
No. of Optimal		20		14		17		17		20	

Table 1: Heuristic Results Compared with CPLEX in Group 1

Instance	Size	CPLEX		HC			PTS			SWO			SWO+L		
No.	$(m \times n)$	obj.	time	obj.	time	gap(%)	obj.	time	gap(%)	obj.	time	gap(%)	obj.	time	gap(%)
1	12 × 30	495	6	495	0	0.00	495	1	0.00	495	1	0.00	495	1	0.00
2	14 × 30	505	9	505	1	0.00	505	0	0.00	505	0	0.00	505	2	0.00
3	16 × 30	625	13	540	0	13.60	604	1	3.36	619	1	0.96	625	1	0.00
4	18 × 30	432	16	425	0	1.62	432	0	0.00	386	1	10.65	432	3	0.00
5	20 × 30	745	18	745	0	0.00	745	1	0.00	722	0	3.09	745	1	0.00
6	22 × 30	722	31	722	0	0.00	722	0	0.00	722	1	0.00	722	2	0.00
7	24 × 30	1180	41	862	0	26.95	1152	1	2.37	1152	1	2.37	1180	3	0.00
8	26 × 30	665	78	429	0	35.49	567	1	14.74	608	0	8.57	665	4	0.00
9	28 × 30	439	172	425	0	3.18	425	1	3.18	435	2	0.91	439	5	0.00
10	30 × 30	995	225	937	1	5.82	995	1	0.00	897	1	9.85	995	5	0.00
11	12 × 40	374	10	374	0	0.00	374	0	0.00	368	0	1.60	374	2	0.00
12	14 × 40	339	15	339	0	0.00	339	0	0.00	327	1	3.54	339	1	0.00
13	16 × 40	523	20	519	0	0.76	523	1	0.00	519	1	0.76	523	3	0.00
14	18 × 40	597	52	595	0	0.34	595	0	0.34	587	1	1.68	595	3	0.34
15	20 × 40	1382	91	923	0	33.21	923	1	33.21	1307	0	5.43	1382	2	0.00
16	22 × 40	917	220	747	1	18.54	917	1	0.00	875	1	4.58	904	4	1.42
17	24 × 40	1104	223	899	0	18.57	829	1	24.91	1030	1	6.70	1098	3	0.54
18	26 × 40	—	—	713	0	—	695	0	—	719	1	—	768	4	—
19	28 × 40	—	—	766	1	—	690	1	—	766	1	—	766	4	—
20	30 × 40	—	—	822	0	—	1080	1	—	962	2	—	1107	5	—
No. of Optimal		17		7			10			3			14 (+3)		

Table 2: Heuristic Results Compared with CPLEX in Group 2

due to its local search component.

In any computation study, it is desirable to scale data up to test the effectiveness of algorithms. In view of this, we ran a second set of tests where the number of cranes ranged up to one hundred and the jobs up to two thousand. Results for these were similar to those obtained for small cases where the SWO+L algorithm outperformed the other heuristics.

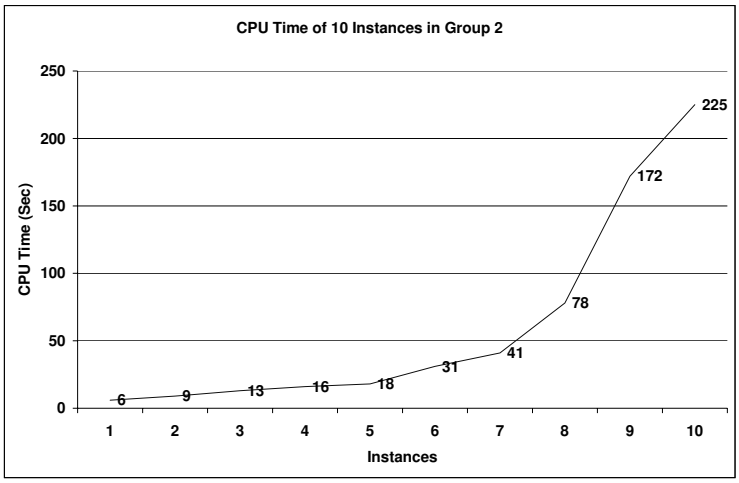

Figure 8: CPU times for CPLEX

7 Conclusion

In this work, we studied a new crane scheduling model which included commonly-found spatial constraints. For the Non-crossing and Neighborhood constraint problems, we proposed dynamic programming algorithms. For the more difficult Job Separation constraint problem, we showed it to be NP-complete and used PTS and SWO with Local Search. We proposed a new framework which improves the SWO technique by including a local search component. Experiments were conducted on industry-size test cases as well as large cases.

For the smaller test cases, the quality and speed of the heuristics was compared with results obtained from CPLEX. These showed that SWO with Local Search performed best and can give good results for both small and large problems in very short running times which allows its use in practical situations.

In practice, more complex situations could occur and it would be interesting and useful to study crane scheduling with spatial and time considerations although this would be the subject of further research.

Acknowledgment: The authors wish to thank the anonymous reviewer for his invaluable comments and suggestions in improving this work. The first and fourth named authors acknowledge support in part from the Logistics and Supply Chain Institute, Hong Kong University of Science and Technology.

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] David P. Clements, James M. Crawford, David E. Joslin, Geoge L. Nemhauser, Markus E. Puttlitz, and Martin W.P. Savelsbergh. Heurstic optimization: A hybrid ai/or approach. In *Proceedings of the Workshop on Industrial Constraint-Directed Scheduling*, 1997.

[3] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[4] Carlos F. Daganzo. The crane scheduling problem. *Transportation Research*, 23B(3):159–175, 1989.

[5] Sabria F. and Carlos F. Daganzo. Queuing systems with scheduled arrivals and established service order. *Transportation Research*, 23B(3):159–175, 1989.

[6] Y. Ge and Y. Yih. Crane scheduling with time windows in circuit board production lines. *International Journal of Production Research*, 33(5):1187–1199, 1995.

[7] Fred Glover and Manuel Laguna. *Tabu Search*. Kluwer Acadamic Publishers, 1997.

[8] David E. Joslin and David P. Clements. "squeaky wheel" optimization. In *Proceedings of AAAI98*, pages 340–346, 1998.

[9] G. Murty Katta. *Operations Research, Deterministic Optimization Models*. Prentice Hall, 1995.

[10] 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.

[11] Hirofumi Matsuo, Jen S. Shang, and Robert S. Sullivan. A knowledge-based system for stacker crane control in a manufacturing environment. *IEEE Transcations on Systems, Man and Cybernetics*, 19(5):932–945, September/October 1989.

[12] Hirofumi Matsuo, Jen S. Shang, and Robert S. Sullivan. A crane scheduling problem in a computer-integrated manufacturing environment. *Management Science*, 37(5):587–606, May 1991.

[13] 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.

[14] V.J.Rayward-Smith, I.H.Osman, C.R.Reeves, and G.D.Simth. *Modern Heuristic Search Methods*. John Wiley & Sons Ltd., 1996.

[15] Jiefeng Xu, Steve Y. Chiu, and Fred Glover. Optimizing a ring-based private line telecommunication network using tabu search. *Management Science*, 45(3):330–345, March 1999.

Appendix A: An Integer Program Model of the Problem with Job-separation Constraint

We use notations previously defined. In addition, we define the binary variables $g_{pq} = 1$ when job q is assigned to crane p, and $g_{pq} = 0$ otherwise and the binary variables h_{pqr} for the Neighborhood constraint, where $h_{pqr} = 1$ when the assignment (p, q) (assign job q to crane p) forbids job r to be done by other cranes, and $h_{pqr} = 0$ otherwise. M represents a sufficiently large number.

The IP model can be written as follows:

$$\textbf{maximize} \sum_{p=1}^{m} \sum_{q=1}^{n} W_{pq} g_{pq}$$

$$g_{pq} \leq W_{pq}, 1 \leq p \leq m, 1 \leq q \leq n \tag{1}$$

$$\sum_{p=1}^{m} g_{pq} \leq 1, 1 \leq q \leq n \tag{2}$$

$$\sum_{q=1}^{n} g_{pq} \leq 1, 1 \leq p \leq m \tag{3}$$

$$g_{pq} + g_{p'q'} \leq 1, 1 \leq p < p' \leq m, 1 \leq q' \leq q \leq n \tag{4}$$

$$h_{pqr} M > (r - q + s_p).(q + s_p - r), 1 \leq p \leq m, 1 \leq q, r \leq n \tag{5}$$

$$(h_{pqr} - 1)M \leq (r - q + s_p).(q + s_p - r), 1 \leq p \leq m, 1 \leq q, r \leq n \tag{6}$$

$$g_{pq} + g_{p'q'} + h_{pqq'} \leq 2, 1 \leq p, p' \leq m, 1 \leq q, q' \leq n, q \neq q' \tag{7}$$

$$g_{pq} + g_{p'q'} + h_{p'q'q} \leq 2, 1 \leq p, p' \leq m, 1 \leq q, q' \leq n, q \neq q' \tag{8}$$

$$g_{pq} + g_{p'q'} + D_{qq'} \leq 2, 1 \leq p, p' \leq m, 1 \leq q, q' \leq n \tag{9}$$

In the IP, constraint (1) ensures that only the provided edges can be selected; constraint (2) ensures that one job can be done by at most one crane; constraint (3) ensures that any crane can perform at most one job; and constraint (4) is the Non-crossing constraint, i.e., (p, q) and (p', q') cannot hold simultaneously, if they cross over each other. Constraints (5) and (6) define the properties of variables h_{pqr}. We know $h_{pqr} = 1$ if and only if $r \in$

$[q - s_p, q + s_p]$, so constraint (5) ensures $y_{pqr} = 1$ when $r \in [q - s_p, q + s_p]$ and constraint (6) ensures $y_{pqr} = 0$ when $r \notin [q - s_p, q + s_p]$. Constraints (7) and (8) define the Neighborhood constraint: if job q is assigned to crane p and job q' is assigned to crane p' simultaneously, job q' cannot be blocked by the assignment (p, q) (constraint (7)) and job q cannot be blocked by the assignment (p', q') (constraint (8)). Finally, the Job-separation constraint is defined by constraint (9).

Appendix B: The Details of Move Evaluation in Probabilistic Tabu Search

This appendix discusses details for evaluating a move in the probabilistic tabu search, studied in the section 5.3.3. The three types of penalties are short-term memory penalties, long-term memory penalties, and penalties for other bias.

Short-term Memory Penalties. As mentioned, the delete and add tabu status can be converted to penalties by using the following values:

$$SD_{x,y} = DPU * (tabu\_add(x, y) - iter)$$

if (x, y) is deleted in the current move and where DPU denotes the scaling factor Delete Penalty Unit, and

$$SA_{x,y} = APU * (tabu\_delete(x, y) - iter)$$

if (x, y) is added in the current move and where APU denotes the scaling factor Add Penalty Unit.

If a move is admissible by aspiration we deduct a large integer, called the Aspiration Threshold (AST) from the penalty so as to encourage this move.

To sum up, the total Short-term Memory Penalty (SMP) can be given by:

$$SMP = \sum_{deleted(x,y)} SD_{x,y} + \sum_{added(x,y)} SA_{x,y} - [AST]$$

Long-term Memory Penalties. We consider long-term memory using two measures — transition measure and residence measure. We evaluate the two measures in very similar ways but note that the effects are opposite — transition measure gives a positive penalty while residence measure gives a negative penalty. Let $f_{x,y}$ denotes the frequency that the edge (x, y) is deleted, and $f'_{x,y}$ denotes the frequency that the edge (x, y) is added. If there is an edge (x, y) in the current evaluated solution, the transition and residence measure penalties can be calculated as follows.

$$LT_{x,y} = f_{x,y} * TPU/iter$$

$$LR_{x,y} = f'_{x,y} * RPU/iter$$

where TPU denotes for Transition Penalty Unit and RPU denotes for Residence Penalty Unit.

Hence the total Long-term Memory Penalty (LMP) can be taken as

$$LMP = \sum_{(x,y)} (LT_{x,y} + LR_{x,y})$$

Note that the residence penalty is negative since the residence measure gives a negative penalty to encourage moves.

Penalties for Other Biases. In addition to the short-term and long-term memory, we consider other metrics which can measure moves. Considering the three spatial constraints at hand, firstly, because of the Non-crossing constraint, if job y is assigned to crane x, all the jobs that are greater than y cannot be carried out by cranes that are less than x, and vice versa. So we expect the crane and job to be as close as possible. The distance between the crane and job can taken as $|x - y|$. Taking the problem size into account, we can express a Non-Crossing Penalty (NCP) as follows.

$$NCP_{x,y} = \alpha * |x - y|/(m + n)$$

where m and n are number of cranes and jobs respectively, and α is a scaling factor.

Secondly, we consider the Neighborhood constraint. We prefer to choose the cranes that have smaller values of Neighborhood constraint, i.e., they will block out jobs. This penalty, the Neighborhood Penalty (NHP), is given by:

$$NHP_x = \beta * S_x/n$$

where x is the crane we choose to assign jobs and S_x is the Neighborhood constraint value, and β is a scaling factor.

Thirdly, similar to the above, we prefer to choose the jobs with smaller numbers of conflicting jobs, i.e., choose those jobs that will block fewer other jobs. This penalty, the Job-Separation Penalty (JSP), is given by:

$$JSP_y = \gamma * CJLength_y/n$$

where y is the crane we choose to assign jobs and $CJLength_y$ is number of jobs that conflict with job y, and γ is a scaling factor.

To sum up, the total penalty for the spatial constraints biases can be given by

$$BSP = \sum_{(x,y)} NCP_{x,y} + \sum_x NHP_x + \sum_y JSP_y$$

Finally, the total penalty for a given move evaluation is the sum of the Short-term Memory Penalty, the Long-term Memory Penalty and the penalty for biases, which is given by:

$$total\_penalty = SMP + LMP + BSP$$

Hence, in our evaluation, the final value is the total profit less the total penalty. Because of different instance sizes, we normalize the penalty by dividing the total penalty by a suitable multiple of the number of edges e. The formula to evaluate a given move is

$$move\_value = total\_profit - total\_penalty/(K * e)$$

where K is a scaling factor.

Appendix C: The Greedy Algorithm in the SWO Constructor

The procedure of constructing a greedy solution in the SWO constructor is described by Algorithm 1.

Algorithm 1 Construction of a Greedy Algorithm of SWO

for each crane c_i in $1 \leq i \leq m$ **do**

randomly generate a real number p in $(0, 1)$

if $p < p_1$ **then**

randomly generate a real number q in $(0, 1)$

if $q < p_2$ **then**

select a compatible job from $\{j_1, \ldots, j_n\}$ with the largest weight

else

randomly pick one job from all the compatible jobs in $\{j_1, \ldots, j_n\}$

end if

end if

end for

Appendix D: Blame Values Calculation in the Analyzer Component of SWO

The Neighborhood and Job-separation constraints are crane specific, i.e., for a certain crane, if it is assigned a job, the same number of jobs will be affected, regardless of which job it is

assigned. Hence, we can omit this factor when we assign blame to the crane.

However, how much the Non-crossing constraint affects our overall solution depends on the "distance" between the crane and the job. If job y is assigned to crane x, all jobs that are greater than y cannot be done by the cranes that are less than x, and vice versa. So we expect the cranes and jobs are as "close" as possible. The "distance" between the crane and job can be taken as $|x - y|$. Generally, for a given crane x, if job y is assigned to it, then a unit throughput weight is defined as $UW_{x,y} = W_{x,y}/(|x - y| + 1)$. (We use $|x - y| + 1$ to avoid dividing by zero.) Furthermore, we let $MUW_x = \max\{UW_{x,y}\}$, $1 \leq y \leq n$, be the maximum unit weight for crane x among all assignments. After obtaining a solution from the Constructor enhanced by local search, the blame of a crane x can be calculated as follows.

- If no job is assigned to crane x, the blame $B_x = MUW_x$;
- If job y is assigned to crane x, the blame $B_x = MUW_x - UW_{x,y}$;

This blame calculation scheme addresses both profit and constraint aspects of the problem. It is useful as it identifies which assignments to cranes are troublesome.