A Tabu Search Heuristic for the Quay Crane Scheduling Problem

Marcello Sammarra[§] Jean-François Cordeau[†] Gilbert Laporte[*]
M. Flavia Monaco[§]

21st December 2006

Abstract

This paper proposes a tabu search heuristic for the *Quay Crane Scheduling Problem* (QCSP), the problem of scheduling a fixed number of quay cranes in order to load and unload containers into and from a ship. The optimality criterion considered is the minimum completion time. Precedence and non-simultaneity constraints between tasks are taken into account. The former originate from the different kind of operations that each crane has to perform; the latter are needed in order to avoid interferences between the cranes. The QCSP is decomposed into a routing problem and a scheduling problem. The routing problem is solved by a tabu search heuristic while a local search technique is used to generate the solution of the scheduling problem. This is done by minimising a longest path length in a disjunctive graph. The effectiveness of our algorithm is assessed by comparing it to a branch-and-cut algorithm and to a Greedy Randomised Adaptive Search Procedure (GRASP).

Keywords: container terminal, crane scheduling, disjunctive graph, tabu search

[§]Dipartimento di Elettronica, Informatica e Sistemistica, Università della Calabria, 87036 Rende (CS), Italy. {monaco, m.sammarra}@deis.unical.it

[†]Canada Research Chair in Logistics and Transportation, HEC Montréal, 3000, chemin de la Côte-Sainte-Catherine, Montréal, Canada H3T 2A7. jean-francois.cordeau@hec.ca

[*]Canada Research Chair in Distribution Management, HEC Montréal, 3000, chemin de la Côte-Sainte-Catherine, Montréal, Canada H3T 2A7. gilbert@crt.umontreal.ca

1 Introduction

A container transshipment port can be considered, for our purposes, as an open production system, with a material flow consisting of containers. It is possible to represent a container transshipment port as two subsystems:

- the quayside and the landside which represent the *external interface*;
- the yard, where the containers to be loaded into a ship and those unloaded from a ship are stored, which represents the *storage system*.

Between these two subsystems, a third one, the *cargo handling system*, plays an interconnecting role: a dense network of internal equipments (cranes, straddle carriers and other sorts of specialised vehicles) is provided for container transportation inside the terminal.

It is instructive to provide a short description of container movements. Containers arriving by truck or by rail are moved to appropriate positions in the yard, which is partitioned into *stacks*. In practice it is sufficient to distinguish between areas dedicated to outgoing (export) containers, ingoing (import) containers, empty, and special (perishable or dangerous goods) containers. Outgoing containers are lifted by *yard cranes* and moved by *straddle carriers* to the ship, where they will be loaded by the *quay cranes* [16].

It is clear that optimally managing such a system is very complicated and calls for the solution of several planning problems. For this reason an appropriate solution methodology is to tackle each subproblem separately and try to restrict interactions with other subproblems. In this paper we are concerned with the planning of the quay crane movements to load or unload ships.

The problem was first addressed by Daganzo [3] who provided a mixed integer formulation for the crane scheduling problem. The author considered a fixed number of cranes while assuming that the number of ships to be handled over the planning horizon can be fixed (static problem) or variable (dynamic problem). The problem was solved exactly for small size instances; for the larger cases a heuristic was used. Daganzo and Peterkofsky [4] later provided an exact method to speed up loading and unloading operations or to minimise delay costs. In both problems the authors define a *task* as the loading or unloading of all containers lying in a given area of the yard or the ship (*ship bay*). For this reason the tasks are considered to be preemptable and no precedence relationship is considered between containers stowed in the deck and the hold of a containership, or between loading and unloading operations. Kim and

Park [8] have presented a mixed integer linear model, an exact algorithm and a heuristic for the quay crane scheduling problem. Contrary to the assumptions made in [3] and [4], these authors define a task as a group of homogeneous containers to be handled. For example, these could be containers with the same destination port. Thus in a given ship bay there may be more than one task to be handled, which leads to the imposition of non-preemptable task constraints, precedence relationships between tasks and non-interference constraints between cranes. Moccia et al. [11] have studied the same problem as in [8]; the authors proposed modifications to the model of Kim and Park and developed a branch-and-cut algorithm for its solution. Lim et al. [10] have solved the task-to-crane assignment problem assuming non-interference between cranes; their solution method is based on dynamic programming.

In this paper, we solve the version of the *Quay Crane Scheduling Problem* (QCSP) described by Kim and Park and by Moccia et al. Our aim is to develop a tabu search procedure for the problem as defined by Moccia et al. [11]. The paper is organised as follows: in Section 2 we provide a mathematical model for the problem; Section 3 is devoted to the algorithm, followed by computational results in Section 4, and by conclusions in Section 5.

2 Notation and mathematical model

The model we use for the QCSP is based on that developed in [8], including the modifications reported in [11]. We are given a set of tasks $\Omega = \{1, \ldots, n\}$ and a set of cranes $Q = \{1, \ldots, q\}$. For convenience we introduce two dummy tasks 0 and T to represent, respectively, the first and last tasks performed by each crane; thus the complete set of tasks is $\bar{\Omega} = \Omega \cup \{0, T\}$. We define p_i as the relative processing time of task $i \in \bar{\Omega}$; clearly $p_{\mathbf{0}} = p_T = 0$. Furthermore, $l_i \in \mathbb{Z}^{+}$ represents the location of task $i \in \Omega$ and is expressed as a ship bay number. Precedence relationships between pairs of tasks are expressed by the set $\Phi = \{(i, j) | i, j \in \Omega,\ i \text{ must be completed before } j \text{ starts}\}$, while Ψ is the set of task pairs that cannot be performed at the same time. This means that if $(i, j) \in \Psi$, then either i has to precede j or j has to be completed before j starts; so, clearly, $\Phi \subseteq \Psi$.

Regarding the cranes, r^k is the earliest available time of the crane k, and $l^{\mathbf{0}}_k, l^T_k \in \mathbb{Z}^{+}$ are its starting and its final position, respectively, which are expressed by a ship bay number. Denoting by $\hat{t}$ the time needed by a crane to move between two adjacent bays, the travelling time of a crane between two generic bays can be expressed as $t_{ij} = \hat{t}|l_i - l_j|$; likewise $t^k_{\mathbf{0}j} = \hat{t}|l^{\mathbf{0}}_k - l_j|$ and $t^k_{iT} = \hat{t}|l_i - l^T_k|$ are the travelling times taken by crane k to move from its starting position

to bay j and from bay i to its final position, respectively. Finally we recall that the cranes, allocated to handle a given ship, can move along the quay either on rail or on tyres. In both cases they can only translate and therefore they can be ordered, with respect to their starting position for example, from the leftmost, corresponding to the crane located in the lowest ship bay to the rightmost, that is the crane in the highest ship bay.

We also introduce the following variables:

- $x_{ij}^k \ \ (i,j \in \bar{\Omega},\ k \in Q)$ is equal to 1 if and only if tasks i and j are performed consecutively by crane k;
- $z_{ij} \ \ (i,j \in \Omega)$ is equal to 1 if and only if task j starts after the completion time of task i;
- $c_i \ \ (i \in \Omega)$ is the completion time of task i;
- $y^k \ \ (k \in Q)$ is the completion time of crane k;
- w is the makespan.

The model proposed in [11] is the following:

$$\text{minimise } \alpha_{\mathbf{1}} w + \alpha_{\mathbf{2}} \sum_{k \in Q} y^k \tag{1}$$

$$y^k \leq w \qquad (k \in Q) \tag{2}$$

$$\sum_{j \in \mathbf{\Omega}} x_{\mathbf{0}j}^k = 1 \qquad (k \in Q) \tag{3}$$

$$\sum_{j \in \mathbf{\Omega}} x_{jT}^k = 1 \qquad (k \in Q) \tag{4}$$

$$\sum_{k \in Q} \sum_{j \in \mathbf{\Omega}} x_{ij}^k = 1 \qquad (i \in \bar{\Omega}) \tag{5}$$

$$\sum_{j \in \mathbf{\Omega} \cup \{T\}} x_{ij}^k - \sum_{j \in \mathbf{\Omega} \cup \{\mathbf{0}\}} x_{ji}^k = 0 \qquad (i \in \Omega, k \in Q) \tag{6}$$

$$c_i + t_{ij} + p_j - c_j \leq M(1 - x_{ij}^k) \qquad (i,j \in \bar{\Omega}, k \in Q) \tag{7}$$

$$c_i + p_j \leq c_j \qquad ((i,j) \in \Phi) \tag{8}$$

$$c_i + p_j - c_j \leq M(1 - z_{ij}) \qquad (i,j \in \Omega) \tag{9}$$

$$c_i + p_j - c_j + \sum_{k \in Q} \sum_{u \in \mathbf{\Omega} \cup \{\mathbf{0}\}, l_u \neq l_i} \hat{t} x_{uj}^k \leq M(1 - z_{ij}) \qquad (i, j \in \Omega, l_i = l_j) \qquad (10)$$

$$c_j - p_j - c_i \leq M z_{ij} \qquad (i, j \in \Omega) \qquad (11)$$

$$c_j - p_j - c_i - \sum_{k \in Q} \sum_{u \in \mathbf{\Omega} \cup \{\mathbf{0}\}, l_u \neq l_i} \hat{t} x_{uj}^k \leq M z_{ij} \qquad (i, j \in \Omega, l_i = l_j) \qquad (12)$$

$$z_{ij} + z_{ji} = 1 \qquad ((i, j) \in \Psi) \qquad (13)$$

$$\sum_{v=\mathbf{1}}^{k} \sum_{u \in \mathbf{\Omega} \cup \{\mathbf{0}\}} x_{uj}^v - \sum_{v=\mathbf{1}}^{k} \sum_{u \in \mathbf{\Omega} \cup \{T\}} x_{ui}^v \leq M(z_{ij} + z_{ji}) \qquad (i, j \in \Omega, l_i < l_j, k \in Q) \qquad (14)$$

$$c_j + t_{jT}^k - y^k \leq M(1 - x_{jT}^k) \qquad (j \in \Omega, k \in Q) \qquad (15)$$

$$r_k - c_j + t_{\mathbf{0}j}^k + p_j \leq M(1 - x_{\mathbf{0}j}^k) \qquad (j \in \Omega, k \in Q) \qquad (16)$$

$$x_{ij}^k \in \{0, 1\} \qquad (i, j \in \bar{\Omega}, \in Q) \qquad (17)$$

$$z_{ij} \in \{0, 1\} \qquad (i, j \in \Omega) \qquad (18)$$

$$y^k,\ c_i \geq 0 \qquad (i \in \bar{\Omega},\ k \in Q). \qquad (19)$$

In this model the objective function (1) is a linear combination of the makespan (see constraints (2)) and the sum of the crane completion times. As observed in [8, 11], minimising the makespan is the same as minimising the ship completion time, while minimising the completion time of the cranes maximises their productivity (i.e. the number of containers handled per hour), since they are kept idle as little as possible. Therefore by a linear combination of the two objectives it is possible to distinguish, among all schedules yielding the same minimum makespan, those corresponding to a higher productivity. Note that the problem under consideration is intrinsically a multiobjective optimisation problem and the reduction to a single objective problem through a linear combination of the objectives is quite common. In [8, 11] it is assumed that $\alpha_{\mathbf{1}} >> \alpha_{\mathbf{2}}$, since minimising the makespan is considered a primary objective; therefore we can set $\alpha_{\mathbf{2}} = 0$. Constraints (3), (4), (6) are the classical routing constraints. Constraints (5) ensure that each task is performed by one and only one crane. Constraints (7) and (8) calculate the task completion times and simultaneously eliminate subtours. The correct completion times for the last and the first task on each crane are computed through constraints (15) and (16). Constraints (9), (10), (11) and (12) define the z_{ij} variables and prevent crane interference. This occurs whenever the completion time of a task i is equal to the starting handling time of another task j, the tasks lie in the same bay, and are performed

by different cranes. In this case the crane must perform task i safely, which can be enforced by starting task j no sooner than $c_i + \hat{t}$. As noted by Moccia et al. [11], the model developed by Kim and Park does not take into account constraints (10), (11) and (12). This model may therefore yield solutions where interference occurs between cranes. Constraints (13) state that tasks i and j cannot be handled at the same time if $(i, j) \in \Psi$. Finally, constraints (14) avoid both interferences due to the crossing of the cranes' jibs and the overtaking between cranes. Supposing that tasks i and j ($l_i < l_j$) are performed at the same time by the cranes $k_{\mathbf{2}}$ and $k_{\mathbf{1}}$ ($k_{\mathbf{1}} < k_{\mathbf{2}}$), respectively, then the left-hand side of (14) is equal to one, while the right-hand side is zero. The constant M in the model is a large number.

3 Algorithm

The QCSP can be formally defined as a scheduling problem on *parallel uniform machines* with precedence constraints, denoted in the classic three fields notation as $Pm|prec|C_{max}$. The QCSP also possesses non-standard characteristics: ready times on machines and explicit non-simultaneity constraints. It is well known [13] that $Pm|prec|C_{max}$ is strongly NP-hard unless the precedence constraints graph is an *outtree* or an *intree*; other polynomially solvable cases of the above problem arise when the number of machines is one or greater than the number of jobs. Thus the *QCSP* is at least as difficult as $Pm|prec|C_{max}$ since it reduces to this problem when $r^k = 0$, $\forall k \in K$, and $\Psi = \Phi$. The QCSP can also be viewed as a *vehicle routing and scheduling problem* consisting of two subproblems:

- a *routing* problem, which determines the *sequence* of tasks on each machine;
- a *scheduling* problem, which determines, for each *sequence*, the starting handling time (or equivalently the completion time) of the tasks belonging to that route.

In [8] a branch-and-bound algorithm and a Greedy Randomised Adaptive Search Procedure (GRASP) are used to solve the QCSP, while in [11] the authors present an exact branch-and-cut algorithm. The branch-and-bound method of Kim et al., as well as the GRASP, work very well on instances with two cranes and up to ten to 15 tasks, while they fail on instances with three cranes and 20 to 25 tasks. In contrast the branch-and-cut algorithm of Moccia et al. has a higher overall efficiency, and outperforms the branch-and-bound algorithm, both in terms of solution quality and of computation time. Clearly the fast growth of the solution time with the size of the instances remains the major drawback of the branch-and-cut algorithm,

although it is faster than the branch-and-bound algorithm. The heuristic algorithm presented in this paper offers, in contrast, a reasonable compromise in terms of computational burden and solution quality. It results from tackling the QCSP as a vehicle routing problem and it is based on tabu search.

In what follows, we refer to the task sequence handled by a crane k as a *route* σ^k and we denote a schedule by $\sigma = (\sigma^{\mathbf{1}}, \ldots, \sigma^q)$. Moreover given a route σ^k we denote by $\sigma^{k-\mathbf{1}}$ and $\sigma^{k+\mathbf{1}}$, respectively, the left adjacent and the right adjacent route of $\sigma^k \quad \forall k \in Q \setminus \{1, q\}$. The first route has no left adjacent route and the last route has no right adjacent route. The generic task performed by crane k (a vertex in the route σ^k) is denoted by σ_i^k; a path from 0 to T is denoted by $\pi = (\pi_{\mathbf{0}}, \ldots, \pi_T)$. The scheduling and routing subproblems are addressed in Sections 3.1 and 3.2, respectively.

3.1 A local search algorithm for the scheduling subproblem

When feasible values of the x variables are given, i.e., when a given task sequence is known, what remains to be done is to calculate the completion times of the tasks. The latter subproblem can be modeled on a graph $G = (\bar{\Omega}, E)$, where $E = A \cup E_{\mathbf{\Psi}}$, and $A = \bigcup_{k \in Q} A^k \cup A_{\mathbf{\Phi}}$, with

$$A^k = \left\{(i,j) : x_{ij}^k = 1, i, j \in \bar{\Omega}\right\} \ \forall k \in Q, \ A_{\mathbf{\Phi}} = \Phi \setminus \bigcup_{k \in Q} A^k, \ E_{\mathbf{\Psi}} = \Psi \setminus \bigcup_{k \in Q} A^k.$$

For example, consider the following instance:

$\Phi = \{(5,6), (7,8), (9,10)\}$

$\Psi = \{(2,3), (3,4), (4,5), (4,6), (5,6), (5,7), (5,8), (6,7), (6,8), (7,8), (7,9), (7,10), (8,9), (8,10), (9,10)\}$

The graph G corresponding to

$x_{\mathbf{01}}^{\mathbf{1}} = x_{\mathbf{12}}^{\mathbf{2}} = x_{\mathbf{24}}^{\mathbf{1}} = x_{\mathbf{45}}^{\mathbf{1}} = x_{\mathbf{57}}^{\mathbf{1}} = x_{\mathbf{79}}^{\mathbf{1}} = x_{\mathbf{9}T}^{\mathbf{1}} = x_{\mathbf{03}}^{\mathbf{2}} = x_{\mathbf{36}}^{\mathbf{2}} = x_{\mathbf{68}}^{\mathbf{2}} = x_{\mathbf{8\,10}}^{\mathbf{2}} = x_{\mathbf{10}\,T}^{\mathbf{2}} = 1$

is depicted in Figure 1. In this graph:

$A^{\mathbf{1}} = \{(0,1), (1,2), (2,4), (4,5), (5,7), (7,9), (9,T)\}$, $A^{\mathbf{2}} = \{(0,3), (3,6), (6,8), (8,10), (10,T)\}$, $A_{\mathbf{\Phi}} = \Phi$ and $E_{\mathbf{\Psi}} = \{(2,3), (3,4), (4,6), (5,8), (6,7), (7,10), (8,9)\}$.

Graph G consists of $|Q|$ *chains* connected to each other by arcs and edges (undirected arcs). Such a graph is known as a *disjunctive graph.*

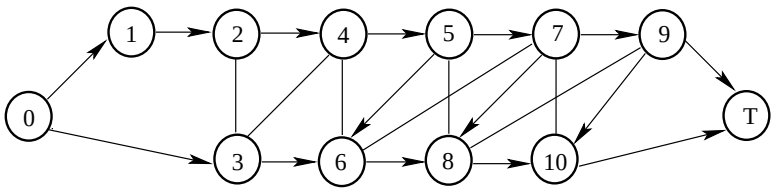

Figure 1: The disjunctive graph obtained from the values of the x variables.

3.1.1 Disjunctive graph representation

The concept of disjunctive graph was introduced by Roy and Soussmann [15] for the job shop scheduling problem $Jm||C_{max}$. In this context two represention schemes are available. In both representations the vertices represent operations of a single task. The difference between the two representations lies in the meaning of a chain. In the first representation a chain is a task. Therefore, vertices belonging to the same chain are connected by arcs, while those belonging to different chains are connected by edges. In the second representation a chain is the set of operations performed by a machine. Therefore, vertices belonging to the same chain are connected by edges while vertices lying on different chains are connected by arcs, as the correct sequence of operations for each task must be kept. In both representations every feasible schedule for the job shop problem corresponds to an *edge orientation* yielding an acyclic directed graph; conversely any orientation yielding an acyclic direct graph corresponds to a feasible schedule [14]. Moreover the makespan of a feasible schedule corresponding to an acyclic orientation of the edges is determined by a *longest path* from a dummy source vertex 0 to a dummy sink vertex T. Thus, the minimisation of the makespan in a job shop environment reduces to finding a feasible orientation minimising the longest path length [13].

The same idea can be applied to the QCSP, although the structure of the underlying disjunctive graph is quite different from that of the job shop problem. Actually, while the chains of the job shop graph are only connected by edges or by arcs, in our case both edges and arcs can exist between chains. This is not a drawback since this structure can be profitably exploited to reduce the number of edges. To illustrate, consider vertices 4 and 6 connected by an edge; because there exists a direct path between the nodes 4 and 6, any feasible orientation will include arc $(4, 6)$ but not arc $(6, 4)$, for otherwise a cycle between nodes 4, 5 and 6 would result; the same reasoning can be applied to edges $(5, 8)$ and $(7, 10)$. The resulting graph is depicted in Figure 2. When a feasible orientation can be determined for the edges, as just shown, computing the makespan reduces to determining a longest path from 0 to T on a weighted

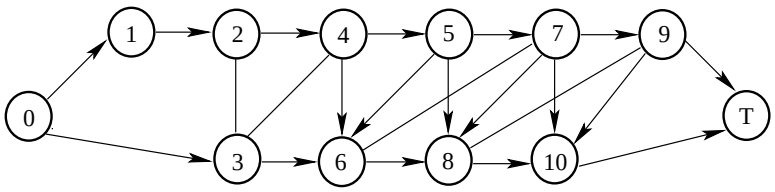

Figure 2: The reduced disjunctive graph.

directed graph, were the arc weights are given by:

$$\begin{aligned}
&\tau_{\mathbf{0}j} = r^k_{\mathbf{0}} + t^k_{\mathbf{0}\,j} + p_j && \forall k \in Q,\ x^k_{\mathbf{0}\,j} = 1,\\
&\tau_{iT} = t^k_{i\,T} && \forall k \in Q,\ x^k_{i\,T} = 1,\\
&\tau_{ij} = t_{ij} + p_j && \forall k \in Q,\ (i,j) \in A^k,\\
&\tau_{ij} = p_j && \forall (i,j) \in A_{\mathbf{\Phi}} \cup \bar{E}_{\mathbf{\Psi}}.
\end{aligned}$$

The longest path can be computed through a label setting algorithm derived from the general Bellman-Ford algorithm [1].

3.1.2 Local search heuristic

When it is not possible to direct all the edges, as in the example just provided, we use the following simple local search heuristic. Let $U = \{u_{\mathbf{1}}, \ldots, u_\lambda\}$ be the set of undirected edges, $F = \{f_{\mathbf{1}}, \ldots, f_\lambda\}$ a random orientation of U, and F^i $(i = 1, \ldots, \lambda)$ the orientation obtained from F by reversing the orientation of the arc f_i.

1. **Initialisation**

   a) $best = \infty$

   b) compute a longest path in the graph $G = (\bar{\Omega}, A, F)$; let *cost* be the solution value

   c) if $cost < best$ then $best = cost$.

2. **Local Search**

   a) for $i = 1, \ldots, \lambda$

      – compute a longest path in the graph $G = (\bar{\Omega}, A, F^i)$

      – if $cost < best$ then $best = cost$

   b) return *best* and the corresponding orientation.

The local search starts from F and explores its immediate neighbourhood. It is possible that either F or some of the orientations F^i give a cyclic graph. In this case we simply discard these orientations, since they represent infeasible solutions. The approximation introduced by the local search procedure, which solves only $|U| + 1$ longest path problems instead of $2^{|U|}$, is very drastic. Motivations for this approach arise from the following considerations: first, it is suitable to overestimate the optimal value rather than using a lower bound on the makespan; second, the computation of a good upper bound must be carried out quickly. The ideas of the solution approach presented in this section are integrated within a tabu search heuristic to be described in Section 3.2.

3.2 A tabu search algorithm for the routing subproblem

We now propose a tabu search algorithm for the routing subproblem. Tabu search [6, 7, 5] is basically a local search heuristic. Therefore, at each iteration h, it moves from the current solution $x^{(h)}$ to the best solution $x^{(h+1)}$ in a subset $N(x^{(h)})$ of the neighbourhood of $x^{(h)}$. In order to avoid being trapped in a local optimum, tabu search may perform moves that deteriorate the current solution, with the hope of eventually identifying a better one. Moreover, to prevent the search from revisiting previously visited solutions and thus to return to the same local optimum, the tabu search keeps in memory the most recent moves, declaring those as forbidden, or *tabu*, for a given number of iterations. However the tabu status of a move may be revoked if this move yields a new incumbent (best known solution). By performing a wider exploration of the solution space, tabu search allows the identification of much better solutions than classical local search algorithms do.

The basic ingredient of a tabu search heuristic is the neighbourhood function Δ which enables moves between different feasible solutions. Our neighbourhood function transforms a feasible schedule σ into the set of schedules $\Delta(\sigma)$ obtained by performing one of the following moves:

- *swapping* adjacent tasks;
- *inserting* a task, currently assigned to route k, into an adjacent route.

Obviously the running time of any tabu search algorithm depends on the size of the neighbourhood, since the objective function must be evaluated for each neighbour. In our case the evaluation of the objective function is not immediate because it requires the solution of an

NP-hard problem. We have to consider a small, but at the same time good, neighbourhood structure.

3.2.1 The swapping move

Consider a schedule $\sigma = (\sigma^{\mathbf{1}}, \ldots, \sigma^{q})$. If we perform all possible swaps on σ, we obtain a neighbourhood whose size is bounded from above by the number of tasks. However some swaps lead to infeasible solutions because the resulting schedule may violate a precedence constraint; other swaps may yield makespans that are worse than the current value. We now explain the swapping move. The value of the makespan equals the value of a longest path π in G; a longest path determines a set of *critical tasks*, i.e., tasks whose starting time cannot be postponed without increasing the makespan. Tasks lying on the critical path can be grouped into *blocks* defined as the maximum number of tasks lying on the critical path and handled by the same crane. For the example reported in the Section 3.1.1, suppose that, for a given orientation, the longest path is $\pi = (0\ ,1\ ,2\ ,3\ ,4\ ,5\ ,6\ ,7\ ,9\ ,T)$ (see Figure 3). There are five blocks:

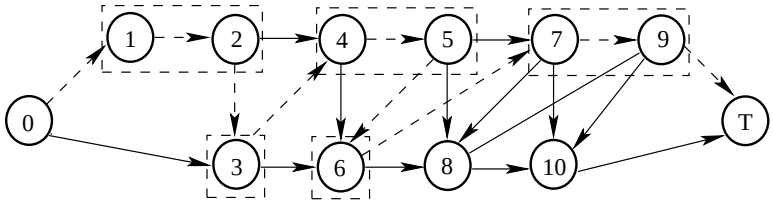

Figure 3: Graph showing blocks and the critical path identified by dashed arcs.

$P^{\mathbf{1}} = [1, 2]$, $P^{\mathbf{2}} = [3]$, $P^{\mathbf{3}} = [4, 5]$, $P^{\mathbf{4}} = [6]$, $P^{\mathbf{5}} = [7, 9]$. If tasks 8 and 10 are swapped on $\sigma^{\mathbf{2}}$, the path π still remains in the graph resulting after the swap. This amounts to saying that swapping tasks not belonging to the longest path does not immediately improve the makespan. This is why Van Laahroven et al. [9] have proposed swapping only those tasks belonging to the critical path while Nowicki et al. [12] suggested swapping only the first and the last two tasks of each block, leading to an important neighbourhood reduction. Our swapping rule is quite different: for each block $P^h = \pi_{h_1}, \ldots, \pi_{h_u}, u \geq 2$, and for each pair of consecutive tasks in P^h, if $\Phi(\pi_{h_i}, \pi_{h_{i+1}}) = 0$ then swap tasks π_{h_i} and $\pi_{h_{i+1}}$. We do not allow the swapping of the last task of block P^h with the first one of block $P^{h\mathbf{+1}}$ because this swap corresponds to an insertion move.

3.2.2 The insertion move

While it is easy to evaluate swapping moves that do not improve the makespan, the same cannot be said of insertion moves. However, we found that the best insertions are those that limit overcrossings between cranes. Thus we allow the movement of tasks from their currently assigned crane to an adjacent one. More precisely we evaluate the following insertion moves:

1. insertion of a task i from route σ^k as the first one in route σ^{k+1}, $k = 1, \ldots, q-1$;
2. insertion of a task i from route σ^k as the last one in route σ^{k+1}, $k = 1, \ldots, q-1$;
3. insertion of a task i from route σ^k as the first one in route σ^{k-1}, $k = 2, \ldots, q$;
4. insertion of a task i from route σ^k as the last one in route σ^{k-1}, $k = 2, \ldots, q$.

If the insertion of task i in the first position of an adjacent route is infeasible, meaning that in the route there is at least a task j such that $(j, i) \in \Phi$, we then insert task i immediately after the last task j such that $(j, i) \in \Phi$; also if the insertion move on the last position of an adiacent route is infeasible, we then insert i immediately before the first task j for which $(i, j) \in \Phi$.

3.2.3 The search process

To describe how our tabu search algorithm works, it is necessary to introduce additional notation. A schedule σ is represented by a set of attributes

$$B(\sigma) = \{(i, j, k) | i \text{ and } j \text{ are consecutive tasks handled by crane } k\}.$$

As in [2] the transition from a schedule σ to another schedule $\bar{\sigma}$ can be carried out by modifying $B(\sigma)$ according to the neighbourhood function described in Section 3.2.3. If an attribute (i, j, k) is removed from $B(\sigma)$, then (i, j, k) is declared tabu and cannot be inserted in $B(\sigma)$ before θ iterations. This means that a move is considered tabu if at least one of the attributes describing the schedule obtained by that move is tabu. Nevertheless a tabu move will be performed if at least one attribute satisfies the *aspiration criterion*, i.e., the move yields a new incumbent. The aspiration level of an attribute (i, j, k) is denoted by α_{ij}^k. The number of iterations for which an attribute remains forbidden is denoted by β_{ij}^k, while the number of times (*frequency*) an attribute has been added to a solution is denoted by δ_{ij}^k. Therefore these define the memory mechanism of the tabu search algorithm. More precisely the $\beta' s$ represent the *short term*

memory which prevent the search from visiting solutions just visited; the $\delta's$ represent the *long term* memory, and are used to lead the search into unexplored region of the solution space (*diversification*). In order to reach this objective, non-improving moves are penalised by a term equal to the sum of the frequencies of all the attributes added to the current solution, multiplied by a constant factor γ.

In what follows the neighbourhood of a schedule σ will be denoted by $N(\sigma)$, while the set of schedules obtained by performing non-tabu moves will be denoted by $M(\sigma)$. Finally we recall that w is the makespan of a schedule σ, while w^* and σ^* represent, respectively, the incumbent optimal value of the makespan and the schedule for which the value is reached, λ is an iteration counter and λ_{max} is the maximum number of iterations performed by our algorithm.

3.2.4 Overall description of the tabu search heuristic

Our tabu search algorithm for the QCSP can now be described:

1. Compute a starting solution σ; let w be the value of the makespan.
2. $\sigma^* = \sigma$, $w^* = w$.
3. $\forall (i,j,k)$ $\delta_{ij}^k = 0$, $\beta_{ij}^k = 0$, $\alpha_{ij}^k = \infty$.
4. $\forall (i,j,k) \in B(\sigma)$ $\alpha_{ij}^k = w^*$.
5. For $\lambda = 1, \ldots, \lambda_{max}$
   a) $M(\sigma) = \emptyset$.
   b) $\forall \bar{\sigma} \in N(\sigma), \forall (i,j,k) \in B(\bar{\sigma}) \setminus B(\sigma)$ such that $\beta_{ij}^k < \lambda$ or $\bar{w} < \alpha_{ij}^k$, $M(\sigma) = M(\sigma) \cup \{\bar{\sigma}\}$.
   c) $\forall \bar{\sigma} \in M(\sigma)$, if $\bar{w} \geq w$, then $g(\bar{\sigma}) = \bar{w} + \gamma w \sum_{(i,j,k) \in B(\bar{\sigma}) \setminus B(\sigma)} \delta_{ij}^k / \lambda$, else $g(\bar{\sigma}) = \bar{w}$.
   d) Solve $\min_{\bar{\sigma} \in M(\sigma)} g(\bar{\sigma})$, and let $\hat{\sigma}$ be the optimal value.
   e) $\forall (i,j,k) \in B(\sigma) \setminus B(\hat{\sigma})$ set $\beta_{ij}^k = \lambda + \theta$.
   f) $\forall (i,j,k) \in B(\hat{\sigma}) \setminus B(\sigma)$ set $\delta_{ij}^k = \delta_{ij}^k + 1$.
   g) If $\hat{w} < w^*$, then $\sigma^* = \hat{\sigma}$ $w^* = \hat{w}$.
   h) $\forall (i,j,k) \in B(\hat{\sigma})$ set $\alpha_{ij}^k = \min\left\{\alpha_{ij}^k, \hat{w}\right\}$.

i) Set $\sigma = \hat{\sigma}$ and $w = \hat{w}$.

We now provide a brief description of the main loop (5).

- Steps (5.a) and (5.b) calculate the set $M(\sigma)$ of all schedules in the neighbourhood of current solution σ, whose added attribute are either non-tabu or satisfy the aspiration criterion;
- Step (5.c) implements the diversification process;
- Step (5.d) returns the best feasible schedule $\hat{\sigma} \in M(\sigma)$;
- Steps (5.e) and (5.f) update the short term memory and the long term memory, respectively: each attribute removed from those describing the current solution σ is declared tabu for the next θ iterations; the frequency of each attribute added to describe $\hat{\sigma}$ is increased by one unit;
- Step (5.g) keeps track of the best solution found through the search;
- Step (5.h) updates the aspiration level of the attributes describing the new identified solution;
- Step (5.i) makes the search proceed to a new solution.

4 Computational results

The tabu search algorithm described in Section 3.2 was coded in C++ and run on a PC equipped with a P4 2.66GHz processor and 256Mb of RAM. It uses three parameters: the maximum number of iterations allowed λ_{max}, the length of the tabu list θ, and the coefficient γ of the diversification process in Step 5.c. The fine tuning of the parameters was carried out by running the algorithm with various values of λ_{max}, θ and γ. Regarding γ, one observes that this parameter should depend on problem size. After some experimentation we set $\gamma = \hat{\gamma}\sqrt{|\Omega||Q|}$, with $\hat{\gamma}$ equal to 0.15, and then $\lambda_{max} = 5000$. With these values, we let θ vary in the interval $[5, 15]$.

While small values of θ, around 7, are suitable for small and medium size instances, for the larger instances this setting is unsuitable. We observed that for large size instances, the number of iterations required to find the best solution decreases with θ, while for small size

instances the number of iterations is not affected by θ. Thus we fixed $\theta = 12$. Finally we recall that the performance of tabu search, like the performance of any local search heuristic, is affected by the starting solution, although in tabu search this influence is limited by the diversification mechanism. We have run the algorithm with two different starting solutions. A first one, *S-TASKS*, is constructed by assigning to each crane the same number of tasks. The second starting solution, *S-LOAD*, is obtained by assigning tasks to cranes so that each crane has the same load in terms of total processing time. We found that the best results were reached by starting the search from the *S-LOAD* solution, and by setting $\theta = 12$ and $\hat{\gamma} = 0.02$.

The algorithm was tested on the instances used by [8] and [11]. The sizes of the instances are the following:

- instances from $k13$ to $k22$ have 10 tasks and 2 cranes (set A);
- instances from $k23$ to $k32$ have 15 tasks and 2 cranes (set B);
- instances from $k33$ to $k42$ have 20 tasks and 3 cranes (set C);
- instances from $k43$ to $k49$ have 25 tasks and 3 cranes (set D).

Finally, we recall that Moccia et al. [11] have used a 2.5 GHz P4 machine equipped with 512 Mb of RAM for their numerical experiments, while the GRASP described in [8] was run on a 466MHz PII machine equipped with 64Mb of RAM. The comparison results between our algorithm, the branch-and-cut algorithm and the GRASP are shown in what follows.

4.1 Tabu search vs branch-and-cut

Here we compare the results provided by our tabu search algorithm with those reported in [11]. As noted by Moccia et al., the instances of sets A and B can been solved exactly by CPLEX, while on instances of sets C and D CPLEX does not always return the optimal solution within the time limit, which has been fixed by the authors to two hours. We therefore compare tabu search with CPLEX on instances of sets A and B (Table 1), and with the branch-and-cut algorithm on instances belonging to the sets C and D (Table 2). In both cases we report the best solution value reached by the algorithms ("Best"), the total computation time ("Time") taken by the algorithms, and the time ("T_S") taken by the tabu search algorithm to find the best solution. All times are expressed in minutes. For instances of sets C and D we report also the best lower bound ("LB") returned by the the branch-and-cut algorithm.

		CPLEX [11]		Tabu Search		
Instance	Set	Best	Time	Best	T_S	Time
k13	A	453	7.75	453	0.00	1.58
k14	A	546	0.01	546	0.00	1.65
k15	A	513	0.02	513	0.00	1.47
k16	A	312	0.09	312	0.00	1.46
k17	A	453	0.04	453	0.02	1.46
k18	A	375	0.01	375	0.01	1.50
k19	A	543	1.96	543	0.56	1.54
k20	A	399	0.04	399	0.00	1.54
k21	A	465	0.01	465	0.00	1.49
k22	A	537	0.18	537	0.08	1.48
k23	B	576	0.06	582	1.17	5.88
k24	B	666	0.96	669	0.00	5.83
k25	B	738	0.59	741	0.16	5.80
k26	B	639	0.14	639	2.67	5.79
k27	B	657	0.05	660	0.85	5.72
k28	B	531	0.21	531	0.44	5.93
k29	B	807	0.15	810	0.01	5.74
k30	B	891	0.22	891	3.60	5.88
k31	B	570	85.56	570	0.04	5.94
k32	B	591	1.12	591	0.59	6.10
Average values		563.10	4.96	564.00	0.51	3.69

Table 1: Comparison between tabu search and CPLEX.

The analysis of Tables 1 and 2 shows that our algorithm works very well on data sets A and D, i.e., for the smallest and the largest instances tested. Indeed tabu search solves all instances of set A, and four out of seven instances of set D. For the medium size instances (data sets B and C) our method solves seven out of 20 problems and its performance deteriorates on instances of the set C. However, when an optimal solution is not reached, the cost of the solution found by tabu search is very close to the cost of the solution reported in [11]. Actually the largest gap is approximately 2.8% (instance $k48$).

We have also analysed the behaviour of our algorithm on the unsolved instances, i.e., those instances for which the branch-and-cut algorithm has not been able to find an optimal solution within the fixed time limit. To this aim we have let the tabu search to run for two hours. The results are presented in Table 3. We note that on five out of nine instances ($k42$, $k44$, $k45$, $k48$, $k49$), an improvement of the solution quality has been obtained with respect to the previous tabu search runs. The improvement is particularly significant on instances $k42$, $k48$, and $k49$, where the cost of the solution reached by the tabu search algorithm is also smaller than or

equal to the cost returned by the branch-and-cut. Regarding the remaining four instances, for which the tabu search has not been able to improve the previously determined solution, we note that on instance $k38$ both algorithms returned the same solution. Since the corresponding gap returned by the branch-and-cut is 0.7% [11], this solution is near-optimal. For the instance $k43$ the solution returned by the tabu search is already much better than that identified by the branch-and-cut and it is probable that no further improvement can be achieved.

Regarding computation times it is clear that our heuristic is disadvantaged with respect to the branch-and-cut algorithm because it uses a stopping criterion that is not based on the proof of the optimality of the solution found. Even so, the performance of our algorithm is comparable to that of the exact algorithms; in fact for medium and large instances, data sets C and D, respectively, in only four cases (problems $k33$, $k34$, $k37$, $k47$) is branch-and-cut faster than tabu search.

		Branch-and-Cut [11]			Tabu Search		
Instance	Set	LB	Best	Time	Best	T_S	Time
k33	C	603.00	603	10.60	603	3.38	20.88
k34	C	717.00	717	14.86	735	17.71	22.28
k35	C	684.00	684	42.04	690	14.90	22.26
k36	C	678.00	678	86.06	681	0.01	21.58
k37	C	510.00	510	21.20	519	19.03	22.31
k38	C	613.67	618	120.00	618	0.88	21.53
k39	C	508.38	513	120.00	519	1.62	21.45
k40	C	564.00	564	67.10	567	0.02	22.24
k41	C	585.06	588	120.00	594	7.27	21.59
k42	C	560.31	570	120.00	576	0.79	21.39
k43	D	859.32	897	120.00	879	47.49	49.42
k44	D	820.35	822	120.00	834	0.23	49.08
k45	D	824.88	840	120.00	852	4.11	49.62
k46	D	690.00	690	90.40	690	24.57	48.27
k47	D	792.00	792	27.00	792	0.01	46.48
k48	D	628.87	645	120.00	663	39.87	49.84
k49	D	879.22	927	120.00	912	1.90	48.06
Average values			685.76	84.66	689.65	10.81	32.84

Table 2: Comparison between tabu search and branch-and-cut.

4.2 Tabu search vs GRASP

Table 4 summarises the comparison between our tabu search algorithm and the GRASP developed by Kim and Park in [8]. In order to provide a fair comparison of the algorithms, we

		Branch-and-Cut [11]	Tabu Search	
Instance	**Set**	**Best**	**Best**	T_S
k38	**C**	**618**	**618**	**0.88**
k39	**C**	**513**	**519**	**1.62**
k41	**C**	**588**	**594**	**7.26**
k42	**C**	**570**	**570**	**80.79**
k43	**D**	**897**	**879**	**47.49**

		GRASP [8]		Tabu Search		
Instance	**Set**	**Best**	T_C	**Best**	T_S	It
k13	A	453	0.0093	453	0.0000	0
k14	A	546	0.0057	546	0.0007	2
k15	A	516	0.0037	513	0.0000	0
k16	A	321	0.0050	312	0.0002	1
k17	A	456	0.0077	456	0.0002	1
k18	A	375	0.0103	375	0.0101	42
k19	A	552	0.0073	552	0.0000	0
k20	A	480	0.0073	399	0.0036	12
k21	A	465	0.0053	465	0.0005	2
k22	A	720	0.0087	555	0.0070	23
k23	B	591	0.0217	585	0.0007	1
k24	B	675	0.0313	669	0.0046	4
k25	B	741	0.0293	753	0.0252	29
k26	B	651	0.0420	651	0.0010	1
k27	B	687	0.0173	669	0.0020	2
k28	B	549	0.0277	537	0.0033	3
k29	B	819	0.0313	810	0.0104	9
k30	B	906	0.0323	921	0.0057	5
k31	B	570	0.0190	609	0.0178	16
k32	B	597	0.0403	594	0.0231	19
k33	C	666	0.0367	666	0.0293	11
k34	C	762	0.0393	741	0.0218	5
k35	C	699	0.0423	825	0.0354	18
k36	C	708	0.0793	681	0.0078	2
k37	C	540	0.0930	540	0.0849	22
k38	C	660	0.0533	624	0.0501	14
k39	C	579	0.0780	522	0.0213	5
k40	C	597	0.0493	567	0.0174	4
k41	C	642	0.0897	600	0.0724	17
k42	C	666	0.0713	606	0.0036	1
k43	D	942	0.0970	921	0.0481	5
k44	D	858	0.2063	858	0.0768	8
k45	D	873	0.1330	876	0.0283	3
k46	D	735	0.2223	702	0.1156	12
k47	D	807	0.1457	792	0.0088	1
k48	D	669	0.1023	666	0.0093	1
k49	D	972	0.1523	939	0.1135	12
Average values		649.86	0.0555	636.49	0.0233	8.46

Table 4: Comparison between tabu search and GRASP.

iterations. Therefore, with only a slight increase in computation time, tabu search yields very good quality solutions.

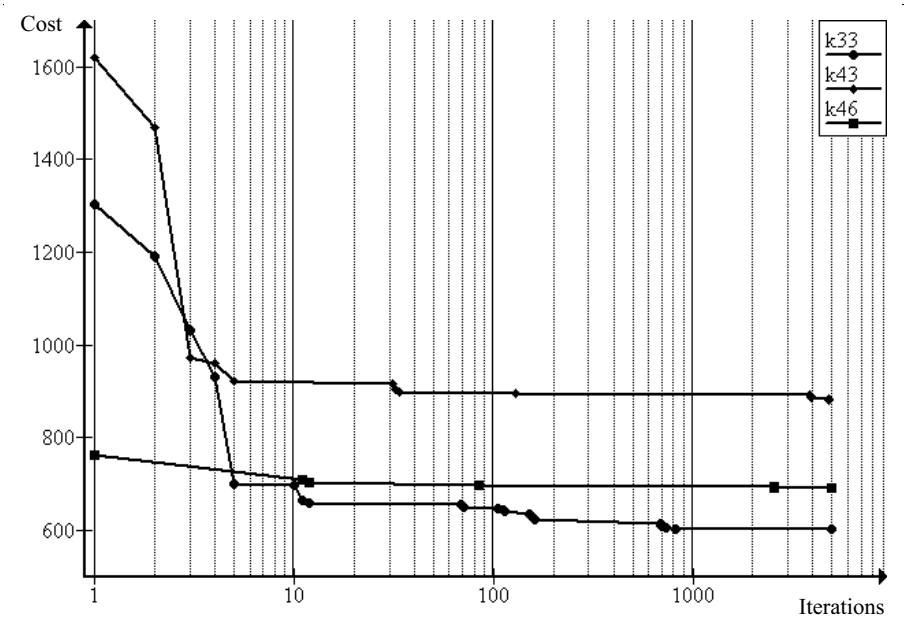

Figure 4: Solution cost vs. number of iterations on three instances.

5 Conclusions

We have described a model and an algorithm for the Quay Crane Scheduling Problem. We have proved that the QCSP can be viewed as a vehicle routing problem which can be decomposed into a routing problem and a scheduling problem. The scheduling problem is NP-hard because it reduces to a longest path problem in a disjunctive graph. We have developed a tabu search algorithm for the routing problem and we have embedded into it a local search technique for the scheduling problem. Our results have been compared with those provided by a GRASP and by a branch-and-cut algorithm. We have seen that our algorithm outperforms the GRASP. Compared with the branch-and-cut, our algorithm provides a good compromise between solution quality and computation time. Our tabu search is capable of identifying the optimal solution on several instances. When an optimum is not found, the solution obtained is almost optimal.

Acknowledgements

This work was partially done when the first author was visiting the Center for Research on Transportation, Université de Montréal. We are grateful to Luigi Moccia for his helpful and accurate comments. Thanks are also due to the Canadian Natural Sciences and Engineering Research Council (grants 227827-04 and 39682-05) and to MIUR (grant 11584-2002 297/1999) for their financial support. We thank the referees for their valuable comments.

References

[1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows - Theory, Algorithms and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] J.-F. Cordeau, M. Gendreau, and G. Laporte. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks*, 30:105–119, 1997.

[3] C. F. Daganzo. The crane scheduling problem. *Transportation Research - B*, 23:159–175, 1989.

[4] C. F. Daganzo and R. I. Peterkofsky. A branch and bound solution method for the crane scheduling problem. *Transportation Research - B*, 24:159–172, 1990.

[5] M. Gendreau and J.-Y. Potvin. Tabu search. In E. K. Burke and G. Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Springer-Verlag, 2005.

[6] F. Glover and M. Laguna. Tabu search. In C.R. Reeves, editor, *Modern Heuristics Techniques for Combinatorial Problems*. Blackwell, Oxford, 1993.

[7] F. Glover and M. Laguna. *Tabu Search*. Kluwer, Boston, 1998.

[8] K.H. Kim and Y.M. Park. A crane scheduling method for port container terminals. *European Journal of Operational Research*, 156:752–768, 2004.

[9] P. J. M. Van Laarhoven, E. H. L. Aarts, and J. K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40:113–125, 1992.

[10] A. Lim, B. Rodrigues, F. Xiao, and Y. Zhu. Crane scheduling with spatial constraints. *Naval Research Logistics*, 51:386–406, 2004.

[11] L. Moccia, J.-F. Cordeau, M. Gaudioso, and G. Laporte. A branch-and-cut algorithm for the quay crane scheduling problem in a container terminal. *Naval Research Logistics*, 53: 45–59, 2006.

[12] E. Nowicki and C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42:797–813, 1996.

[13] M. Pinedo. *Scheduling - Theory, Algorithms and Systems*. Prentice Hall, Englewood Cliffs, NJ, 1995.

[14] M. Pinedo and M. Singer. A shifting bottelneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics*, 46:1–17, 1999.

[15] B. Roy and B. Soussmann. Les problèmes d'ordonnancement avec constraintes disjonctives. Note D.S. 9 bis, SEMA, Paris, 1964.

[16] D. Steenken, S. Voss, and R. Stahlbock. Container terminal operation and operations research - a classification and literature review. *Operations Research Spectrum*, 26:3–49, 2004.