Scheduling Multiple Factory Cranes on a Common Track*

Ben Peterson† Iiro Harjunkoski‡ Samid Hoda§
J. N. Hooker¶

November 2012

Abstract

We describe a heuristic algorithm for scheduling the movement of multiple factory cranes mounted on a common track. The cranes must complete a sequence of tasks at locations along the track without crossing paths, while adhering as closely as possible to a factory production schedule. The algorithm creates a decision tree of possible states of the crane system, which evolves over time as tasks are assigned and sequenced. By identifying and removing inferior states from the tree, the algorithm efficiently generates provably optimal or near-optimal crane schedules, depending on the complexity of the problem instance.

1 Introduction

Factories often employ track-mounted cranes to move materials and equipment from one location to another. Such cranes typically hang from crossbars on which they move laterally while the crossbars themselves move longitudinally along a common track (Fig. 1). This combination allows full three-dimensional movement of the cranes across the factory floor.

*This research was partially supported by grants from ABB Corporate Research and the Pennsylvania Infrastructure Technology Alliance.

†Level 3 Communications, USA, benp84@gmail.com

‡ABB Corporate Research, Germany, Iiro.Harjunkoski@de.abb.com

§Level 3 Communications, USA, samid.hoda@gmail.com

¶Carnegie Mellon University, USA, jh38@andrew.cmu.edu

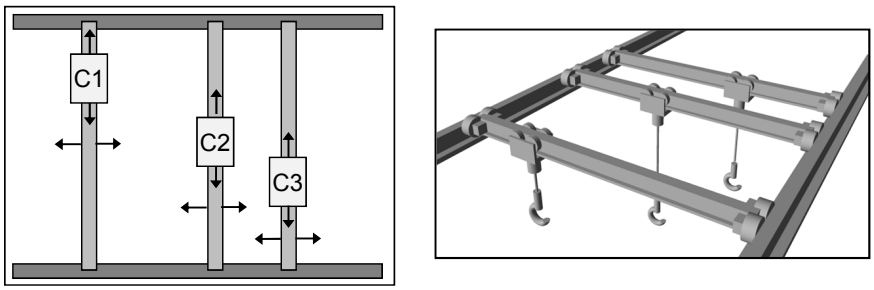

Figure 1: Factory cranes.

Planning the movement of a crane to follow a manufacturing schedule can be a difficult combinatorial problem. For a single-crane factory, the problem consists of sequencing the tasks on the crane in the order that minimizes some function of their completion times. For a multiple-crane factory, there exist the additional problems of choosing a crane for each task and assigning movement priority when the cranes interfere.

Constraints may be imposed on the tasks a crane may perform. Some tasks must be carried out consecutively in a specific order by a single crane. A particular crane may be preassigned to execute a given task. A task may also have a time window in which it must be completed.

Since crane paths must be planned on a scale of seconds for factory schedules on a scale of days, the crane scheduling problem is too complex for general solution methods. This paper presents an efficient specialized algorithm for the problem. Although the number of feasible solutions grows exponentially with the length of the work schedule, our algorithm confines its search for optimal trajectories to a limited solution space using proven dominance rules. In cases where the work schedule is relatively constrained, the algorithm usually returns a provably optimal solution. In cases where the schedule has much more freedom, the solution space of trajectories may be too large for implicit enumeration, and the algorithm returns a near-optimal solution.

Following a review of previous work, we begin with a description of the crane scheduling problem. We then show how to simplify the problem in two stages. We first project out the space-time trajectories of cranes so that the decision variables consist only of the starting times of tasks and their assignment to cranes. We then replace the start times with a task sequence, from which start times can be inferred. At this point we describe a search algorithm and prove that it finds an optimal solution. We show how to reduce

the search space substantially, without sacrificing optimality, by deleting dominated states. Finally, we indicate how to delete additional states when the reduced search space becomes too large, resulting in a solution that may not be optimal. The paper ends with experimental results and conclusions.

2 Previous Research

Crane scheduling has received a great deal of study, but relatively little attention has been given to the factory crane scheduling problem addressed here. The literature focuses primarily on two types of problems: movement of materials from one tank to another in an electroplating or similar process (normally referred to as *hoist scheduling* problems), and loading and unloading of container ships in a port. Both differ significantly from the factory crane problem.

A classification scheme for hoist scheduling problems appears in [14]. These problems typically require a cyclic schedule involving one or more types of parts, where parts of each type must visit specified tanks in a fixed sequence. The most common objective is to minimize cycle time. Even the single-hoist cyclic scheduling problem [1, 13, 18] is NP-complete [7]. Several papers address cyclic two-hoist and multi-hoist problems with exact and heuristic algorithms [8, 24, 3, 6, 9, 10, 11, 15, 19, 20, 21, 22].

Our problem differs from the typical hoist scheduling problem in several respects. The schedule is not cyclic. The problem is given as a set of jobs, each to be performed by one crane, rather than a set of parts to be moved from one tank to another, perhaps by several cranes. A job may consist of several tasks to be performed consecutively in a specified order and perhaps at widely separated locations in the factory. The jobs may all be different. Lastly, we permit lateral crane movements, so that tasks can be distributed over a larger area.

Port cranes are generally classified as quay cranes and yard cranes. Quay cranes may be mounted on a single track, as are factory cranes, but the scheduling problem differs significantly. The cranes load (or unload) containers into ships rather than transferring items from one location on the track to another. A given crane can reach several ships, or several holds in a single ship, either by rotating its arm or perhaps by moving laterally along the track. The problem is to assign cranes to loading (unloading) tasks, and schedule the tasks, so that the cranes do not interfere with each other

[4, 5, 16].

Yard cranes are typically mounted on wheels and can follow certain paths in the dockyard to move containers from one location to another [17, 23]. Port and yard cranes clearly present a different type of scheduling problem than factory cranes.

An early study of factory crane scheduling [12] presents heuristic algorithms that obtain noninterfering solutions only under certain conditions. A worst-case bound is derived for makespan in the two-crane case. However, the method is insufficiently general to address the problem considered here. There is no attempt to apply it to realistic problems, and no computational results are reported.

A dynamic programming algorithm for factory crane scheduling described in [2] evolved from the same project as the present work. It computes optimal movements for two cranes in problems of medium size (e.g., 100-200 tasks) in a minute or so. It assumes that the assignment of jobs to cranes is given in advance. By contrast, the heuristic method presented here is designed to solve the assignment and control problem simultaneously, as well as to solve larger instances with multiple cranes, while possibly sacrificing optimality or proof of optimality.

3 The Crane Scheduling Problem

The crane scheduling problem consists of a list of tasks τ, each of which requires a crane to perform an operation requiring L_τ seconds at a specified location (X_τ, Y_τ), while remaining stationary at that location. For example, the crane might pick up a piece of equipment or empty the contents of a ladle it is carrying. The task must be performed during a specified time window $[R_\tau, D_\tau]$ that is derived from the production schedule of the factory. The task ideally starts at the release time R_τ, but the logistics of crane movements may delay the start. Each task has a nonnegative priority weight to represent its urgency, and the objective is to minimize the weighted sum of delays.

Figure 2 illustrates how one crane may yield to another. In (a), cranes 1 and 2 are assigned tasks 1 and 2, respectively, and there is no interference. The assignment is reversed in (b) and (c), and there is potential interference. In (b), interference is avoided by letting crane 2 yield to crane 1. In (c), crane 1 yields to crane 2.

There are three types of decision variables. One set of variables assigns

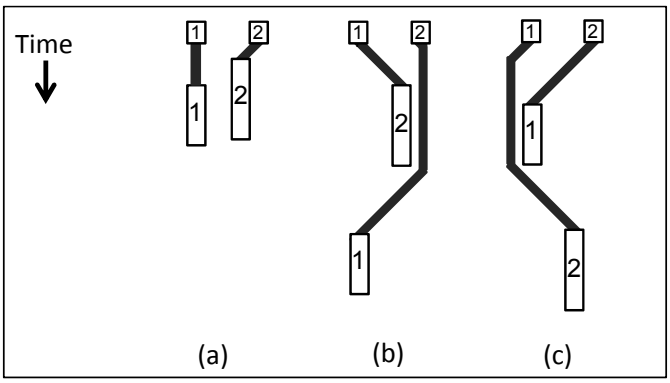

Figure 2: Avoiding interference. (a) No yielding necessary. (b) Crane 2 yields to crane 1. (c) Crane 1 yields to crane 2.

each task to exactly one crane. A second assigns each operation a start time. A third describes space-time trajectories for the cranes. The trajectories must be consistent with the crane assignments, start times, and problem constraints, and they must not allow cranes to interfere with each other. Two types of task precedence constraints may be imposed:

$\tau \prec \tau'$ indicates that the start of task τ' must (eventually) follow the completion of task τ (on any crane)

$\tau \rightarrow \tau'$ indicates that task τ' must immediately follow task τ on the same crane

Production schedules often include short sequences of tasks we call *jobs*. For example, a four-task job might require a crane to pick up a ladle at one location, fill it at a second location, empty it at a third location, and return it to the first location. We use the constraints $\tau_1 \rightarrow \tau_2 \rightarrow \tau_3 \rightarrow \tau_4$ to require that these tasks be executed consecutively by the same crane.

We formulate the crane scheduling problem using the notation in Table 1. Let $\mathbf{s}$ be the tuple of start time variables s_τ. The formulation minimizes

$$f(\mathbf{s}) = \sum_{\tau \in \mathbf{T}} P_\tau (s_\tau - R_\tau) \tag{1}$$

subject to

$$
\begin{array}{l}
\text{Time window and domain constraints:} \\
\quad R_\tau \le s_\tau \le D_\tau - L_\tau, \ \text{ all } \tau \in \mathbf{T} \\
\quad a_\tau \in \mathbf{C}_\tau \text{ and } s_\tau \in \mathbb{R}, \ \text{ all } \tau \in \mathbf{T} \\
\\
\text{Precedence constraints:} \\
\quad s_\tau + L_\tau \le s_{\tau'}, \ \text{ all } \tau, \tau' \in \mathbf{T} \text{ with } \tau \prec \tau' \\
\quad a_\tau = a_{\tau'} \text{ and } s_\tau + L_\tau \le s_{\tau'}, \ \text{ all } \tau, \tau' \in \mathbf{T} \text{ with } \tau \to \tau' \\
\quad s_{\tau''} < s_\tau \text{ or } s_{\tau''} > s_{\tau'}, \\
\qquad\qquad \text{all } \tau, \tau', \tau'' \in \mathbf{T} \text{ with } \tau \to \tau',\ a_{\tau''} = a_\tau,\ \tau'' \ne \tau, \tau'
\end{array}
\tag{2}
$$

as well as trajectory constraints

$$
\begin{array}{l}
\text{Continuity: } \ x_c(t), y_c(t) \in \mathbb{R} \text{ and continuous, } \ \text{all } c \in \mathbf{C}, \ \text{all } t \\
\text{Position: } \ (x_{a_\tau}(t), y_{a_\tau}(t)) = (X_\tau, Y_\tau), \ \ t \in [s_\tau, s_\tau + L_\tau], \ \text{all } \tau \in \mathbf{T} \\
\text{Velocity: } \left\{ \begin{array}{l} x_c(t) - V_x \Delta t \le x_c(t + \Delta t) \le x_c(t) + V_x \Delta t \\ y_c(t) - V_y \Delta t \le y_c(t + \Delta t) \le y_c(t) + V_y \Delta t \end{array} \right\}, \ \text{all } t, \Delta t \ge 0 \\
\text{Spacing: } \ x_{c'}(t) - x_c(t) \ge (c' - c)\Delta X, \ \ \text{all } c, c' \in \mathbf{C} \text{ with } c < c'
\end{array}
\tag{3}
$$

The objective function is the priority-weighted sum of task delays past the desired release times. The time window and domain constraints require observance of release times and deadlines. They also enforce any restrictions on which crane may perform which task. The precedence constraints are of the two types described above. The trajectory constraints require the crane space-time trajectories to satisfy four conditions: they must be continouous, each crane must be at the assigned location of a task while executing the associated operation; each crane must observe speed limits; and cranes must observe the minimum spacing. Initial and/or final positions of a crane may be specified by adding crane-specific zero-length tasks at the desired positions to the beginning and end of the schedule.

4 Projection of Trajectories

Our algorithm operates on a simplified but equivalent form of the crane scheduling problem (1)–(3). Let $\mathbf{a}$ be the tuple of variables a_τ. The simplification consists of two steps: projection of the feasible set onto variables $\mathbf{a}$ and $\mathbf{s}$, and replacement of $\mathbf{s}$ by a tuple $\boldsymbol{\sigma}$ of sequence variables.

Table 1: Notation.

Parameters:	$\mathbf{C}$ =index set of cranes[1]
	$\mathbf{C}_\tau$ =index set of cranes that can perform task τ
	$\mathbf{T}$ =index set of tasks
	P_τ = priority of task τ
	R_τ = release time of task τ
	D_τ = deadline of task τ
	L_τ = duration of task τ
	X_τ = x-position of task τ
	Y_τ = y-position of task τ
	V_x = crane velocity along x-axis
	V_y = crane velocity along y-axis
	ΔX = minimum x-axis gap between adjacent cranes
Variables:	a_τ = assigned crane index for task τ (integer)
	s_τ = start time of task τ (real)
	$x_c(t)$ = x-position of crane c at time t (real)[2]
	$y_c(t)$ = y-position of crane c at time t (real)[2]

[1] cranes are indexed by integers, increasing from left to right

[2] these must be continuous functions of the continuous variable t

We first project the problem onto $\mathbf{a}, \mathbf{s}$ by eliminating the trajectory variables $x_\tau(t), y_\tau(t)$. Once the problem is solved in the projected space, the solution can be lifted to a solution in the full space by specifying appropriate trajectories.

We will show that the projection of the feasible set of (2)–(3) onto $(\mathbf{a}, \mathbf{s})$ is described by (2) plus time gap constraints and interference constraints. The time gap constraints are

$$\left.\begin{array}{l} s_\tau + L_\tau + \max\left\{\dfrac{|X_\tau - X_{\tau'}|}{V_x}, \dfrac{|Y_\tau - Y_{\tau'}|}{V_y}\right\} \le s_{\tau'} \\ \text{or} \\ s_{\tau'} + L_{\tau'} + \max\left\{\dfrac{|X_\tau - X_{\tau'}|}{V_x}, \dfrac{|Y_\tau - Y_{\tau'}|}{V_y}\right\} \le s_\tau \end{array}\right\}, \quad \begin{array}{l} \text{all } \tau, \tau' \in \mathbf{T} \\ \text{with } a_\tau = a_{\tau'} \\ \text{and } \tau \neq \tau' \end{array} \qquad (4)$$

and the interference constraints are

$$\left.\begin{array}{l} X_\tau + (a_{\tau'} - a_\tau)\Delta X \le X_{\tau'} \\ \text{or} \\ s_\tau + L_\tau + \dfrac{|X_\tau + (a_{\tau'} - a_\tau)\Delta X - X_{\tau'}|}{V_x} \le s_{\tau'} \\ \text{or} \\ s_{\tau'} + L_{\tau'} + \dfrac{|X_{\tau'} - (a_{\tau'} - a_\tau)\Delta X - X_\tau|}{V_x} \le s_\tau \end{array}\right\}, \quad \begin{array}{l} \text{all } \tau, \tau' \in \mathbf{T} \\ \text{with } a_\tau < a_{\tau'} \end{array} \tag{5}$$

The time gap constraints ensure that a crane has sufficient time to move between tasks that are assigned to it. The constraint allows time for both x- and y-axis movement. The interference constraints ensure that if two cranes are assigned to two tasks that cannot be performed simultaneously, then one crane must wait for the other to finish before it can travel past the first task and perform the second task.

We suppose that each crane c begins with a dummy task τ_c of zero duration that takes place at time zero and at the starting position of the crane.

Lemma 1 *The projection of the feasible set of (2)–(3) onto the variables* $\mathbf{a}, \mathbf{s}$ *is described by (2), (4) and (5).*

Proof. It suffices to show that, given any solution $\mathbf{a}, \mathbf{s}$ that satisfies (2), (4) and (5), there are crane trajectories $x_c(t), y_c(t)$ that satisfy the trajectory constraints (3). Index the non-dummy tasks $\tau = 1, \ldots, n$ so that $s_1 \le \cdots \le s_n$. We will construct trajectories in stages that correspond to tasks $1, \ldots, n$. In each stage τ we will extend the trajectory of crane $a_\tau = \bar{c}$ enough to perform task τ. That is, we will extend the trajectory from $(X_{\bar{\tau}}, Y_{\bar{\tau}}, s_{\bar{\tau}} + L_{\bar{\tau}})$ to $(X_\tau, Y_\tau, s_\tau + L_\tau)$, where $\bar{\tau}$ is the task that crane $\bar{c}$ performs immediately before τ. We will show that if the existing partial trajectories observe constraints (2)–(3), then this extension of crane $\bar{c}$'s trajectory likewise observes the constraints.

As a starting point for trajectory construction, we define a point trajectory $(x_c(0), y_c(0)) = (X_{\tau_c}, Y_{\tau_c})$ for each dummy task τ_c. The point trajectories trivially satisfy the position and velocity constraints of (3). Because a_{τ_c}, s_{τ_c} satisfy (5) and $s_{\tau_c} = 0$ for each c, they must satisfy the first disjunct of (5). This implies that the point trajectories satisfy the spacing constraints of (3).

We now suppose that the partial trajectories created in stages $1, \ldots, \tau - 1$ satisfy (3), and we extend the trajectory of crane $a_\tau = \bar{c}$ as follows. We first

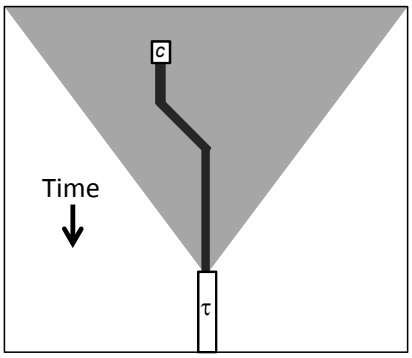

Figure 3: Reachability cone (gray area) for task τ.

define the reachability cone $\mathcal{C}$ to be the set of points from which crane $\bar{c}$ can reach position X_τ at time s_τ (Fig. 3). Thus

$$\mathcal{C} = \{(x, y, t) \mid (s_\tau - t)V_x \geq |x - X_\tau|,\ (s_\tau - t)V_y \geq |y - Y_\tau|\}$$

We next define a non-interference space-time corridor through which we will route crane $\bar{c}$ (Fig. 4). We define the corridor so that all points in it maintain sufficient separation from the other cranes. Thus let $C_L(t)$ be the set of cranes $c < \bar{c}$ (i.e., cranes to the left of $\bar{c}$) for which partial trajectories are defined at time t, and define $C_R(t)$ similarly for cranes to the right. Let $X_L(t)$ be the leftmost position crane $\bar{c}$ can occupy without interfering with a crane to the left, and similarly for $X_R(t)$. Thus

$$X_L(t) = \max_{c \in C_L(t)} \{x_c(t) + (\bar{c} - c)\Delta X\}$$
$$X_R(t) = \min_{c \in C_R(t)} \{x_c(t) - (c - \bar{c})\Delta X\}$$

At each time t, the non-interference corridor spans the interval $[X_L(t), X_R(t)]$. This interval is nonempty because the existing partial trajectories observe the spacing constraints in (3).

We now extend the trajectory of crane $\bar{c}$ as follows. At time $t = s_{\bar{\tau}}$ we have $x_{\bar{c}}(t) = X_{\bar{\tau}}$. At each subsequent time t, the crane moves toward X_τ at velocity V_x, unless it is already at position X_τ, or unless it is constrained by the boundaries of the non-interference corridor. The crane also moves toward Y_τ at velocity V_y until it reaches Y_τ. Thus, for small Δt:

$$x_{\bar{c}}(t + \Delta t) = x_{\bar{c}}(t) + \max\{X_L(t),\ \min\{X_R(t),\ x_{\bar{c}}(t) + V_x(t)\Delta t\}\}$$
$$y_{\bar{c}}(t + \Delta t) = y_{\bar{c}}(t) + V_y(t)\Delta t$$

where

$$V_x(t) = \begin{cases} V_x & \text{if } x_{\bar{c}}(t) < X_\tau \\ -V_x & \text{if } x_{\bar{c}}(t) > X_\tau \\ 0 & \text{if } x_{\bar{c}}(t) = X_\tau \end{cases} \qquad V_y(t) = \begin{cases} V_y & \text{if } y_{\bar{c}}(t) < Y_\tau \\ -V_y & \text{if } y_{\bar{c}}(t) > Y_\tau \\ 0 & \text{if } y_{\bar{c}}(t) = Y_\tau \end{cases}$$

If this trajectory lies inside the reachability cone $\mathcal{C}$, it necessarily reaches location X_τ at time s_τ and therefore satisfies the position constraint of (3). Its construction ensures that it satisfies the velocity constraints of (3). The fact that it belongs to the non-interference corridor implies that it satisfies the spacing constraints of (3). Finally, we complete the trajectory by performing task τ; that is, by setting $x_{\bar{c}}(t) = X_\tau$ for $t \in [s_\tau, s_\tau + L_\tau]$. This final extension clearly satisfies (3).

It therefore suffices to show that the partial trajectory just described lies inside the reachability cone; that is, $(x_{\bar{c}}(t), y_{\bar{c}}(t), t) \in \mathcal{C}$ for $t \in [s_{\bar{\tau}} + L_{\bar{\tau}}, s_\tau]$. The trajectory lies in $\mathcal{C}$ at $t = s_{\bar{\tau}} + L_{\bar{\tau}}$ because $(x_{\bar{c}}(s_{\bar{\tau}}), y_{\bar{\tau}}(s_{\bar{\tau}})) = (X_{\bar{\tau}}, Y_{\bar{\tau}})$, and because $(X_{\bar{\tau}}, Y_{\bar{\tau}})$ and (X_τ, X_τ) must satisfy (4). Subsequently, the trajectory occupies or is moving toward the position (X_τ, Y_τ) except when prevented by a boundary of the non-interference corridor. We therefore need only show that $(X_L(t), y_{\bar{c}}(t), t) \in \mathcal{C}$ when $X_L(t) > X_\tau$ and $(X_R(t), y_{\bar{c}}(t), t) \in \mathcal{C}$ when $X_R(t) < X_\tau$. We show the former, as the argument is analogous for the latter.

Given any $t \in [s_{\bar{\tau}}, s_\tau]$ for which $X_L(t) > X_\tau$, let c' be the crane that determines $X_L(t)$; that is

$$X_L(t) = x_{c'}(t) + (\bar{c} - c')\Delta X \tag{6}$$

Let τ' be the next task performed by crane c'. Then the interference constraints (5) imply

$$(s_\tau - s_{\tau'} - L_{\tau'})\, V_x \geq |X_\tau - (a_\tau - a_{\tau'})\Delta X - X_{\tau'}|$$

which by (6) implies

$$(s_\tau - s_{\tau'} - L_{\tau'})\, V_x \geq |X_\tau - X_L(t)|$$

This and the fact that $t \leq s_{\tau'} + L_{\tau'}$ imply

$$(s_\tau - t)\, V_x \geq |X_\tau - X_L(t)| \tag{7}$$

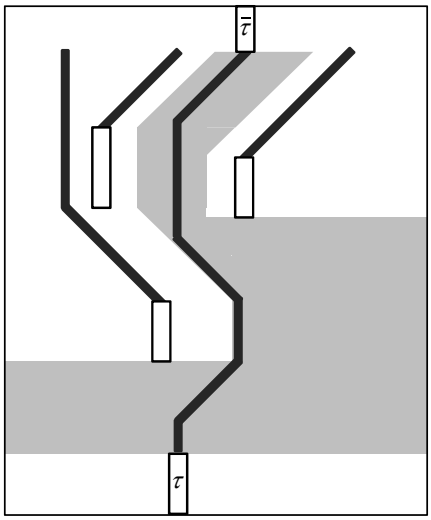

Figure 4: Non-interference corridor for crane $\bar{c}$ (gray area). The crane's extended trajectory from task $\bar{\tau}$ to task τ is shown.

We also have

$$(s_\tau - s_{\bar{\tau}})V_y \geq |Y_\tau - Y_{\bar{\tau}}| \tag{8}$$

from the time gap constraints (4). Also

$$(t - s_{\bar{\tau}})V_y \leq |y_{\bar{c}}(t) - Y_\tau| \tag{9}$$

due to the speed limit V_y. Now

$$\begin{aligned}(s_\tau - t)V_y &= (s_\tau - s_{\bar{\tau}} - (t - s_{\bar{\tau}}))V_y \\ &\geq |Y_\tau - Y_{\bar{\tau}}| - |y_{\bar{c}}(t) - Y_\tau| \geq |Y_\tau - y_{\bar{c}}(t)|\end{aligned} \tag{10}$$

where the first inequality is due to (8)–(9) and the second to the fact that $Y_\tau - Y_{\bar{\tau}}$ and $y_{\bar{c}}(t) - Y_\tau$ have the same sign. Finally, inequalities (7) and (10) imply that $(X_L(t), y_{\bar{c}}(t), t)$ belongs to the reachability cone. □

5 Task Sequencing

The second problem reduction replaces the start time vector $\mathbf{s}$ with a task sequence vector $\boldsymbol{\sigma}$. The motivation for this change is that once the global task sequence is fixed, optimal task start times for that sequence can be found by a greedy choice of the earliest start time for each task in the sequence order.

We therefore associate a greedy solution $(\mathbf{a}, \mathbf{s})$ with each pair $(\mathbf{a}, \boldsymbol{\sigma})$. We first set $s_{\sigma_1} = R_{\sigma_1}$. Then for $k = 2, \ldots, n$, we set s_{σ_k} to the smallest feasible value that is greater than or equal to $s_{\sigma_{k-1}}$.

To make this more precise, reformulate the constraints (2) and (4)–(5) as follows. For a fixed $\mathbf{a}$, a solution $\mathbf{s}$ satisfies these constraints if each s_τ satisfies $s_\tau \in [R_\tau, D_\tau]$ and

$$s_\tau \geq B_\tau(\tau', s_{\tau'}), \quad \text{all } \tau' \neq \tau \text{ with } s_{\tau'} \leq s_\tau \tag{11}$$

Here $B_\tau(\tau', s_{\tau'}) = \max\{B^1_\tau(\tau', s_{\tau'}), B^2_\tau(\tau' s_{\tau'})\}$, where

$$B^1_\tau(\tau', s_{\tau'}) = \begin{cases} s_{\tau'} + L_{\tau'} & \text{if } \tau' \prec \tau \\ -\infty & \text{otherwise} \end{cases}$$

$$B^2_\tau(\tau', s_{\tau'}) = \begin{cases} s_{\tau'} + L_{\tau'} + \max\left\{ \dfrac{|X_{\tau'} - X_\tau|}{V_x}, \dfrac{|Y_{\tau'} - Y_\tau|}{V_y} \right\} & \text{if } a_{\tau'} = a_\tau \\ s_{\tau'} + L_{\tau'} + \dfrac{|X_{\tau'} + (a_{\tau'} - a_\tau)\Delta X - X_\tau|}{V_x} & \text{if } a_\tau < a_{\tau'} \text{ and } \tau \not\ll \tau' \\ s_{\tau'} + L_{\tau'} + \dfrac{|X_{\tau'} + (a_\tau - a_{\tau'})\Delta X - X_\tau|}{V_x} & \text{if } a_{\tau'} < a_\tau \text{ and } \tau' \not\ll \tau \\ -\infty & \text{otherwise} \end{cases}$$

and where $\tau \ll \tau'$ means that τ and τ' satisfy the crane spacing constraint $X_\tau + (a_{\tau'} - a_\tau)\Delta X \leq X_{\tau'}$. The greedy solution sets $s_{\sigma_1} = R_{\sigma_1}$ and

$$s_{\sigma_k} = \max\left\{ s_{\sigma_{k-1}}, \max_{i<k} \{B_{\sigma_k}(\sigma_i, s_{\sigma_i})\} \right\}, \quad k = 2, \ldots, n$$

The greedy solution is clearly feasible.

Lemma 2 *If the projected crane scheduling problem has an optimal solution, then a greedy solution corresponding to some pair* $(\mathbf{a}, \boldsymbol{\sigma})$ *is optimal.*

Proof. Consider any optimal solution $(\bar{\mathbf{a}}, \bar{\mathbf{s}})$ of the projected problem. Without loss of generality, index the tasks so that $\bar{s}_1 \leq \cdots \leq \bar{s}_n$, and let $\boldsymbol{\sigma} = (1, \ldots, n)$. We claim that the greedy solution $(\bar{\mathbf{a}}, \mathbf{g})$ corresponding to $(\bar{\mathbf{a}}, \boldsymbol{\sigma})$ is optimal. It suffices to show that $\mathbf{g} \leq \bar{\mathbf{s}}$, because $(\bar{\mathbf{a}}, \bar{\mathbf{s}})$ is optimal and each objective function coefficient $P_\tau \geq 0$. We show this by induction. First, $g_1 \leq \bar{s}_1$ because $g_1 = R_1$. Now suppose that $g_i \leq \bar{s}_i$ for $i = 1, \ldots k-1$, and

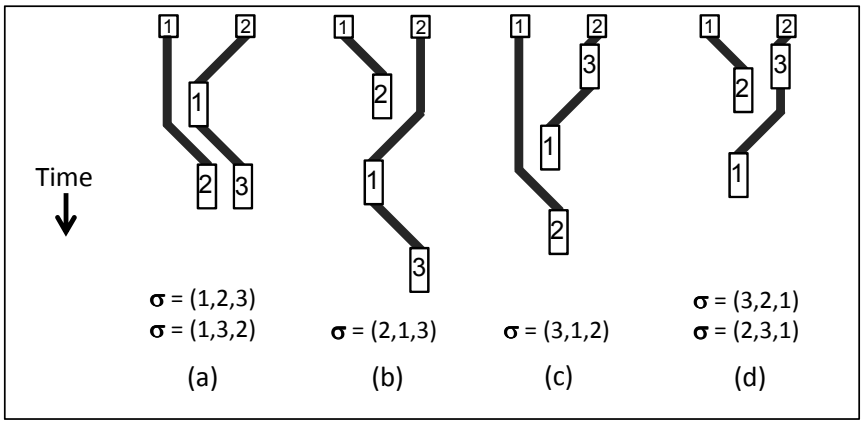

Figure 5: Illustration of canonical and non-canonical sequences.

show that $g_k \leq \bar{s}_k$. By definition of the greedy solution, g_k is the smallest value that satisfies $g_k \geq g_{k-1}$ and $g_k \geq B_k(i, g_i)$ for $i = 1, \ldots, k-1$. Note that $B_\tau(\tau', s_{\tau'})$ is a monotone nondecreasing function of $s_{\tau'}$. This and the fact that each $g_i \leq \bar{s}_i$ imply that no value of $\bar{s}_k$ smaller than g_k can satisfy both $\bar{s}_k \geq g_{k-1}(= \bar{s}_{k-1})$ and

$$\bar{s}_k \geq B_k(i, \bar{s}_i), \quad i = 1, \ldots, k-1 \tag{12}$$

But $\bar{s}_k \geq \bar{s}_{k-1}$ is given, and $\bar{s}_k$ must satisfy (12) because $\bar{\mathbf{s}}$ is feasible. Thus $g_k \leq \bar{s}_k$, and the lemma follows. □

The greedy schedule for a sequence $\boldsymbol{\sigma}$ need not result in start times $\mathbf{s}$ that have the ordering $\boldsymbol{\sigma}$. Consider, for example, a 3-task problem in which crane A is assigned to task 2 and crane B is assigned to tasks 1 and 3; that is, $\mathbf{a} = (\mathrm{B}, \mathrm{A}, \mathrm{B})$. The six possible sequences $\boldsymbol{\sigma}$ result in four solutions as shown in Fig. 5, assuming that time windows do not restrict the solutions. Note in part (d) of the figure that the sequence $\boldsymbol{\sigma} = (2, 3, 1)$ results in start times that follow a different order. This is because the relative order of two tasks makes no difference in the resulting vector $\mathbf{s}$ when the cranes assigned to them do not spatially interfere. Reversing the order of tasks 2 and 3 makes no difference in the resulting schedule, as illustrated in parts (a) and (d) of the figure.

In general, the task sequence $\boldsymbol{\sigma}$ overspecifies the solution when some crane assignments do not interfere with each other. For this reason we define a *canonical* task sequence $\boldsymbol{\sigma}$ to be one that reflects the order of the resulting

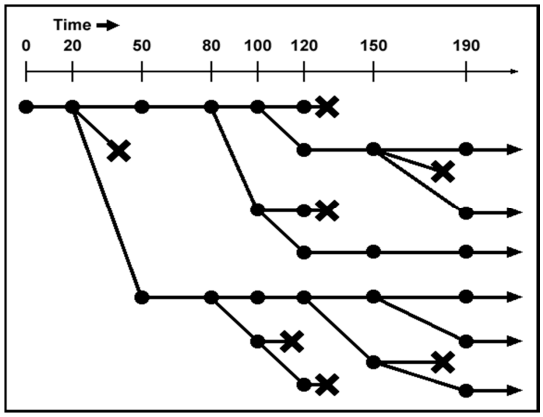

Figure 6: Illustration of crane simulation. Each node of the solution tree represents a state (partial schedule). Dominated states are pruned, as indicated by crossouts.

greedy start times, with ties broken by the crane assignments. That is, $\boldsymbol{\sigma}$ is canonical if $s_{\sigma_i} \leq s_{\sigma_{i+1}}$ for $i = 1, \ldots, n-1$, and $a_{\sigma_i} < a_{\sigma_{i+1}}$ when $s_{\sigma_i} = s_{\sigma_{i+1}}$, where $\mathbf{s}$ consists of the greedy start times for $\boldsymbol{\sigma}$. Thus (1,3,2) and (2,3,1) are noncanonical in Fig. 5.

6 Crane Simulation

The solution algorithm consists of two main operations: crane simulation and state pruning. Crane simulation is a method of building solutions by assigning values to decision variables in chronological order. It assigns and sequences tasks on cranes as they become available in time, based on their time windows and precedence conditions. The simulation branches on each of these decision variables, generating a tree of partially defined solutions (Fig. 6). Because the solution tree can grow rapidly, we frequently remove partial solutions that are dominated by others.

Each partial solution in the solution tree is described by its *state*. The state consists of a partial task sequence $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_k)$, the assigned cranes $\mathbf{a}$, the resulting greedy start times $\mathbf{s}$, a *nogood list*, and a *wakeup time*. The nogood list contains crane-task pairs that have already been enumerated for the current position in the task sequence. The wakeup time is a lower bound on the time that the next task can start and is determined by the other state variables. The algorithm maintains a clock, and a state need be examined

only when the clock time advances to the state's wakeup time.

The algorithm is initialized with a single state in which all state variables are empty and the clock is set to zero. At each iteration, it selects a state S that is currently awake. It determines which crane-task pairs (c, τ) can be appended to the task sequence in S. A new state is created for each such pair (c, τ), representing a new branch in the solution tree. This is accomplished by making a copy S_τ of S, adding τ to the sequence $\boldsymbol{\sigma}$ in S_τ, and setting $a_\tau = c$ in S_τ. The clock is then advanced to the maximum of the current clock time and the earliest wakeup time of current states, and the next iteration is performed. The state S is left in the tree indefinitely, because it may be possible to add other tasks to it at a later clock time.

A state is deleted if it becomes infeasible, which occurs when no task can be scheduled for the next position in the task sequence. The algorithm terminates when all remaining states have a wakeup time of ∞, indicating that all the tasks have been scheduled in each state. A state with the minimum objective function value over all remaining states describes an optimal solution $(\mathbf{a}, \boldsymbol{\sigma})$.

Figure 7 describes the processing of each state in more detail. The wakeup time $W(\boldsymbol{\sigma}, \mathbf{s}, t)$ for a state is the earliest finish time of tasks in process, or the time at which next task will be released, whichever comes first. That is,

$$W(\boldsymbol{\sigma}, \mathbf{s}, t) = \min \left\{ \min_{\tau \in T_1} \{s_\tau + L_\tau\}, \min_{\tau \in T_2} \{R_\tau\} \right\}$$

where t is the clock time, T_1 contains the tasks τ appearing in $\boldsymbol{\sigma}$ for which $t \in [s_\tau, s_\tau + L_\tau]$, and T_2 contains the tasks for which $t < R_\tau$. Thus if all cranes are idle and all tasks have been released, $W(\boldsymbol{\sigma}, \mathbf{s}, t) = \infty$.

An illustration of crane simulation appears in Fig. 8. There are three cranes and two tasks. In state A, task 1 has been assigned to crane 2, starting at time 30. The state has wakeup time 45 because the next task is released at time 45, which occurs before the completion time (90) of the task currently in process. State A is processed when the clock reaches time 45. The state is copied onto state B, in which task 2 is assigned to crane 1; that is, the pair $(c, \tau) = (1, 2)$ is added to the schedule. Task 2 can be performed simultaneously with task 1 and therefore starts at time 45. Another copy is made onto state C, in which task 2 is assigned to crane 3; that is, $(c, \tau) = (3, 2)$ is added to the schedule. Crane 3 must wait at position $(x, y) = (30, 0)$ until task 1 is finished at time 90 before it starts the 20-second

Let t be the current clock time.
For each crane-task pair (c, τ):
  Check whether (c, τ) can be assigned to position $k+1$ in the task sequence $\boldsymbol{\sigma}$ by checking the following conditions:
    Crane c is not busy (i.e., all its previously assigned tasks are complete at t).
    Time t is within task τ's time window $[R_\tau, D_\tau]$.
    Adding (c, τ) to $(\mathbf{a}, \boldsymbol{\sigma})$ satisfies the precedence constraints in (2).
    The nogood list $\mathcal{N}$ does not contain (c, τ).
  If (c, τ) satisfies these conditions, then:
    Add (c, τ) to the nogood list $\mathcal{N}$.
    Update the wakeup time to $W(\boldsymbol{\sigma}, \mathbf{s}, t)$.
    Compute the greedy start time s_τ for task τ.
    If $s_\tau < s_{\sigma_k}$, or $s_\tau = s_{\sigma_k}$ and $c < a_{\sigma_k}$, then:
      Do not add (c, τ) to $(\mathbf{a}, \boldsymbol{\sigma})$, because the resulting sequence is not canonical.
    Else:
      Make a copy of state $(\boldsymbol{\sigma}, \mathbf{a}, \mathbf{s}, \text{wakeup}, \mathcal{N})$, and modify the copy as follows:
      Add (c, τ) to $(\mathbf{a}, \boldsymbol{\sigma})$ by setting $\sigma_{k+1} = \tau$ and $a_\tau = c$.
      Remove all nogoods from $\mathcal{N}$.
      Change the wakeup time to $W(\boldsymbol{\sigma}, \mathbf{s}, t)$.

Figure 7: Algorithm for processing a state $(\boldsymbol{\sigma}, \mathbf{a}, \mathbf{s}, \mathcal{N}, \text{wakeup})$, where $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_k)$.

move to position $(x, y) = (10, 0)$ to perform task 2. Task 2 therefore begins at time 110.

State A is now modified as follows. The pairs $(c, \tau) = (1, 2), (3, 2)$ are added to the nogood list. The wakeup time is updated to 90, because the completion time of the current scheduled task (90) is less than the earliest release time (∞) of tasks not yet released (all tasks have been released). In state B, the wakeup time is 85, because task 2 finishes first at time 85. In state C, the wakeup time is 90, because task 1 finishes first.

Because every incomplete state remains in the solution tree indefinitely (or until it becomes infeasible), there is an opportunity to try all possible assignments $\mathbf{a}$ and task sequences $\boldsymbol{\sigma}$ as the clock advances. The algorithm creates a greedy schedule $\mathbf{s}$ for each feasible pair $(\mathbf{a}, \boldsymbol{\sigma})$ with a canonical sequence $\boldsymbol{\sigma}$. Due to Lemmas 1 and 2, this suffices to find an optimal solution, and we have the following.

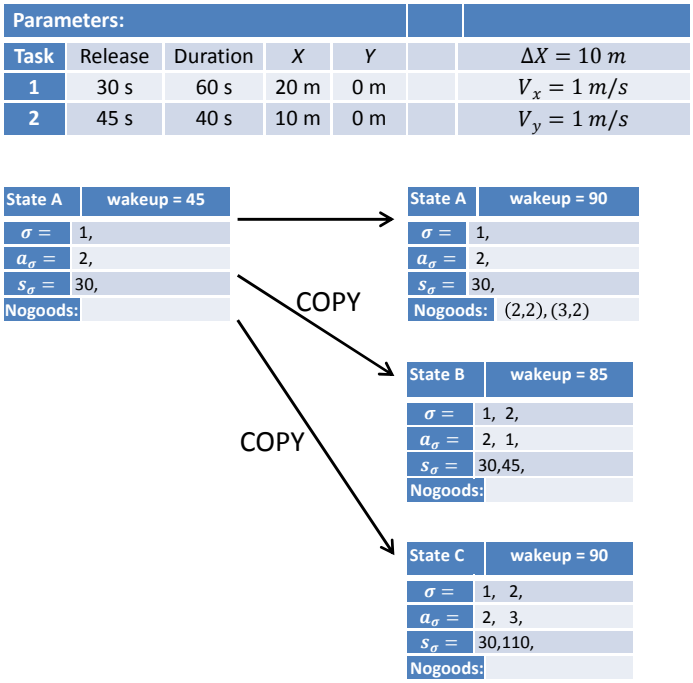

Figure 8: Example of crane simulation. Two copies of state A are created when the clock reaches the state's wakeup time of 45 s.

Theorem 3 *Crane simulation finds an optimal solution of the crane scheduling problem (1)–(3) if one exists.*

7 State Pruning

Periodically during the crane simulation, the solution tree is examined to identify noncompletable, suboptimal or redundant states that can be deleted.

Noncompletable states are identified by flagging a state whenever its wakeup time passes the latest start time $D_\tau - L_\tau$ of a task τ. If the task has not been scheduled in the state, the schedule cannot be completed, and the state is deleted.

We also delete states that are dominated by others. Suppose state S contains the partial solution $(\mathbf{a}, \mathbf{s})$, and state S' contains the partial solution $(\mathbf{a}', \mathbf{s}')$. Then S is dominated by S' if the following conditions are satisfied:

(a) Every task scheduled in S is also scheduled in S'.

(b) Any task τ that can be subsequently sequenced in S by assigning it crane $a_\tau = c$ and start time s_τ either (a) is part of a job whose first task already appears in S' or (b) can be added to S' by assigning it the same crane $a'_\tau = c$ and a start time $s'_\tau \leq s_\tau$.

(c) The accumulated objective value for the tasks that are assigned in S' is no greater in S' than in S. Let T be the set of tasks scheduled in S, and T' the set of tasks scheduled in S'. The accumulated objective value in S' is

$$f' = \sum_{\tau \in T'} P_\tau(s'_\tau - R_\tau)$$

and the accumulated objective value in S is

$$f = \sum_{\tau \in T'} P_\tau(t_\tau - R_\tau)$$

where $t_\tau = s_\tau$ if $\tau \in T$, and otherwise t_τ is the wakeup time for S. We require that $f' \leq f$.

Theorem 4 *Crane simulation obtains an optimal solution, if one exists, when dominated states are deleted.*

Proof. It suffices to show that whenever a state S is deleted because it is dominated by another state S', some completion of S' is at least as good as an optimal completion of S. Suppose, then, that S and S' correspond to partial solutions $(\mathbf{a}, \mathbf{s})$ and $(\mathbf{a}', \mathbf{s}')$ as defined above. Let $(\bar{\mathbf{a}}, \bar{\mathbf{s}})$ be an optimal completion of $(\mathbf{a}, \mathbf{s})$. That is, $(\bar{\mathbf{a}}, \bar{\mathbf{s}})$ is an optimal solution of the projected crane scheduling problem subject to $\bar{a}_{\sigma_i} = a_{\sigma_i}$ and $\bar{s}_{\sigma_i} = s_{\sigma_i}$ for $i = 1, \ldots, k$. It suffices to exhibit a feasible completion $(\bar{\mathbf{a}}', \bar{\mathbf{s}}')$ of $(\mathbf{a}', \mathbf{s}')$ whose cost is no higher than that of $(\bar{\mathbf{a}}, \bar{\mathbf{s}})$. We do this by keeping the schedule for tasks already scheduled in S', and scheduling other tasks as in the optimal completion of S. That is, we define $\bar{a}'_\tau = a'_\tau$ and $\bar{s}'_\tau = s'_\tau$ when τ is scheduled in S', and $\bar{a}'_\tau = \bar{a}_\tau$ and $\bar{s}'_\tau = \bar{s}_\tau$ otherwise.

We first show by induction that $(\bar{\mathbf{a}}', \bar{\mathbf{s}}')$ is feasible. Let τ be the task unscheduled in S' that starts earliest in $(\bar{\mathbf{a}}, \bar{\mathbf{s}})$. Due to dominance condition (a), τ is not already scheduled in S. Thus τ can be scheduled next on crane $\bar{a}_\tau$ with start time $\bar{s}_\tau$ in S. Dominance condition (b) implies that τ can be feasibly scheduled in S' with the same crane assignment and start time. Now let S_+ and S'_+ be the states that result when τ is scheduled in this fashion in

states S and S', respectively. We claim that dominance conditions (a) and (b) hold for S_+ and S'_+. Condition (a) obviously holds. Condition (b) holds because task τ is scheduled at the same time and on the same crane in S_+ and S'_+. Any task that can be scheduled next in S_+ can be identically scheduled in S'_+ without violating feasibility with respect to τ, and therefore without violating feasibility with respect to any task in S'_+. Now that conditions (a) and (b) hold for S_+ and S'_+, we can repeat the argument for a second task that is unscheduled in S'. It follows by induction that $(\bar{\mathbf{a}}', \bar{\mathbf{s}}')$ is feasible.

We now show that the cost of $(\bar{\mathbf{a}}', \bar{\mathbf{s}}')$ is no higher than that of $(\bar{\mathbf{a}}, \bar{\mathbf{s}})$. The cost of $(\bar{\mathbf{a}}', \bar{\mathbf{s}}')$ is

$$C' = \sum_{\tau \in T'} P_\tau(s'_\tau - R_\tau) + \sum_{\tau \notin T'} P_\tau(\bar{s}_\tau - R_\tau) = f' + \sum_{\tau \notin T'} P_\tau(\bar{s}_\tau - R_\tau) \tag{13}$$

The cost of $(\bar{\mathbf{a}}, \bar{\mathbf{s}})$ is

$$\begin{aligned} C &= \sum_{\tau \in T} P_\tau(s_\tau - R_\tau) + \sum_{\tau \in T' \setminus T} P_\tau(\bar{s}_\tau - R_\tau) + \sum_{\tau \notin T'} P_\tau(\bar{s}_\tau - R_\tau) \\ &\geq \sum_{\tau \in T} P_\tau(s_\tau - R_\tau) + \sum_{\tau \in T' \setminus T} P_\tau(w - R_\tau) + \sum_{\tau \notin T'} P_\tau(\bar{s}_\tau - R_\tau) \end{aligned} \tag{14}$$

where w is the wakeup time for state S. The inequality follows from the fact that any task added to S must be scheduled at or after w. From (14) we have

$$C \geq f + \sum_{\tau \notin T'} P_\tau(\bar{s}_\tau - R_\tau) \tag{15}$$

Dominance condition (c) ensures that $f' \leq f$. This, along with (13) and (15), imply that $C' \leq C$, as desired. □

8 State Chopping

If state pruning procedure is not sufficiently effective, crane simulation may generate too many states to hold in memory. We therefore apply a *state chopping* procedure when the number of states exceeds a certain user-defined threshold. State chopping deletes less promising states without proving that they are dominated. Thus when state chopping is turned on, an optimal solution is no longer guaranteed.

The chopping heuristic sorts the states according to the number of tasks completed (high to low), followed by the accumulated objective value (low to high). It deletes states from the front of the list until it is sufficiently short. Although this voids the optimality guarantee, the resulting solutions tend to be very good and even optimal in practice.

9 Experimental Results

We tested the algorithm on two sets of instances, one based on random generation, one consisting of six realistic 2-crane problems obtained from industry. The random instances were also designed to resemble industry problems. We generated instances of 4 different sizes for 1, 2, 3 and 4 cranes, with 5 instances of each size. The tasks have uniformly distributed processing times and time window durations, along with priorities. They are grouped randomly into jobs, and precedence constraints are also generated. We then constructed a heuristic solution by assigning and sequencing the tasks on cranes and computing greedy start times. We used the resulting start times as the given release times for the tasks, so that the optimal value of each instance is exactly zero.

We imposed gradually increasing bounds on the number of states, to observe the effect on solution quality. The algorithm checks the number of states after each pruning, and if the number exceeds the bound, excess states are chopped as described in Section 8. The peak number of states between chopping operations may exceed the bound, sometimes by a factor of two or three. The solver can always prove optimality with a sufficiently large bound on the state space, although we did not reach this bound for the larger instances.

Table 2 displays the solution quality for random instances as the state space bound increases by powers of two. Each number in the table is the percent (averaged over five instances) by which the solution value exceeds the optimal or best known value. As expected, the solution improves as the state space grows, in many cases culminating in a proof of optimality. Table 3 shows the corresponding average computation times in seconds. The boldface figures correspond to state space sizes for which optimality was proved. Note that it generally takes much less time to obtain a value that is within, say, 1% of the best known value than to obtain the best known value. Figs. 9–11 display some representative trajectories produced by the algorithm.

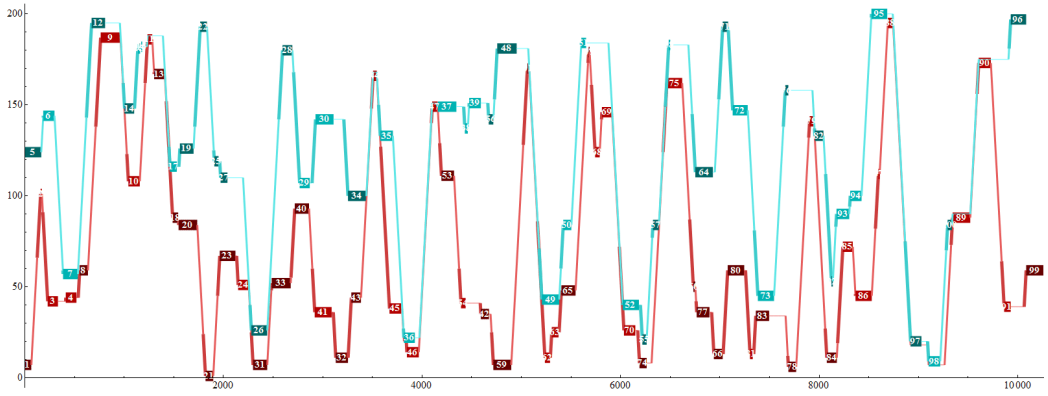

Figure 9: Solution trajectory for 2 cranes and 100 tasks, optimality proved. The horizontal axis represents time.

When an optimal solution is proved, the algorithm discovers the solution in a much smaller state space than is required to prove optimality. As the state space grows, the solution values reach a stable point that is eventually proved to be optimal, and the values never depart from a plateau after reaching it. For instances in which optimality is not proved, the solution values likewise reach a plateau, but the state space does not grow large enough to prove optimality. We might reasonably assume that the stable value is optimal, based on the behavior of the algorithm when it proves optimality. In Table 3, the times shown in boxes correspond to state space sizes for which a stable value was obtained *for all 5 instances*. This indicates that if these instances are representative, one can ordinarily reach a stable value within the time shown, or less.

Results for the instances obtained from industry appear in Tables 4 and 5. The instances are listed in increasing difficulty. Optimality was proved for none of the instances, but a stable value was reached for all but the last one. Aside from this instance, a value within about 1% of the best known value was reached in times ranging from 10 seconds to 20 minutes.

The primary source of complexity in the algorithm is the manner in which it assigns multiple cranes to multiple tasks near the same clock time. It generates a state for each possible combination of crane-task assignments and sequences for those assignments. Especially in problems with more than two cranes, the number of such combinations can be quite large, multiplying the number of states substantially.

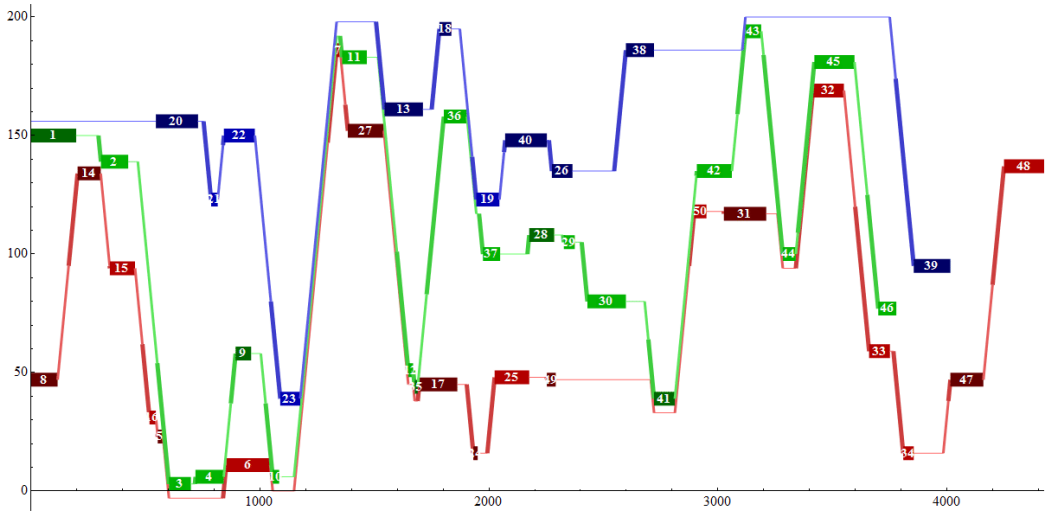

Figure 10: Solution trajectory for 3 cranes and 50 tasks, optimality likely but not proved.

10 Conclusions

We presented an algorithm that solves the factory crane scheduling problem by creating a decision tree of possible states and pruning the tree using a dominance relation between states. It yields optimal or serviceable solutions for problems of realistic size involving up to 4 cranes and 200 tasks, where a serviceable solution is one that is within 1% of the apparent optimum, or within a few percent for the hardest instances. It allows the user to improve a suboptimal solution to the extent desired by raising the ceiling on the state space size and investing more computation time.

A key to the success of the algorithm is a dominance check that prunes inferior partial solutions long before they are extended to complete solutions. This often prevents what would otherwise become an exponential explosion of solution states.

There is room for improvement in the algorithm, however, particularly in the state pruning procedure. When the state space is large, the time required to compare state pairs can become quite large. We hypothesize that streamlining the pruning procedure provides the greatest opportunity for improving the algorithm's efficiency.

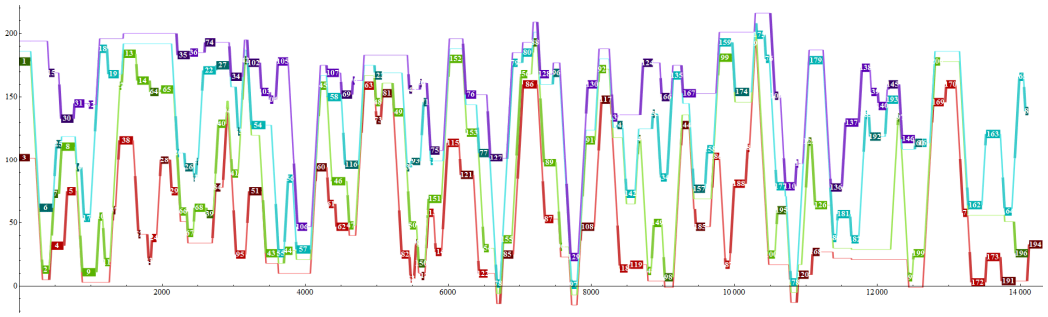

Figure 11: Solution trajectory for 4 cranes and 200 tasks, optimality not proved.

References

[1] Armstrong, R., Lei, L., Gu, S.: A bounding scheme for deriving the minimal cycle time of a single-transporter N-stage process with time-window constraints. European Journal of Operational Research **78**, 130–140 (1994)

[2] Aron, I., Genç-Kaya, L., Harjunkoski, I., Hoda, S., Hooker, J.N.: Factory crane scheduling by dynamic programming. In: R.K. Wood, R.F. Dell (eds.) Operations Research, Computing and Homeland Defense, pp. 93–107. INFORMS (2011)

[3] Che, A., Chu, C.: Single-track multi-hoist scheduling problem: A collision-free resolution based on a branch-and-bound approach. International Journal of Production Research **42**, 2435–2456 (2004)

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

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

[6] Lei, L., Armstrong, R., Gu, S.: Minimizing the fleet size with dependent time-window and single-track constraints. Operations Research Letters **14**, 91–98 (1993)

[7] Lei, L., Wang, T.J.: A proof: The cyclic hoist scheduling problem is NP-complete. Working paper, Rutgers University (1989)

[8] Lei, L., Wang, T.J.: The minimum common-cycle algorithm for cycle scheduling of two material handling hoists with time window constraints. Management Science **37**, 1629–1639 (1991)

[9] Leung, J., Levner, E.: An efficient algorithm for multi-hoist cyclic scheduling with fixed processing times. Operations Research Letters **34**, 465–472 (2006)

[10] Leung, J., Zhang, G.: Optimal cyclic scheduling for printed circuit board production lines with multiple hoists and general processing sequence. IEEE Transactions on Robotics and Automation **19**, 480–484 (2003)

[11] Leung, J.M.Y., Zhang, G., Yang, X., Mak, R., Lam, K.: Optimal cyclic multi-hoist scheduling: A mixed integer programming approach. Operations Research **52**, 965–976 (2004)

[12] Lieberman, R.W., Turksen, I.B.: Two operation crane scheduling problems. IIE Transactions **14**, 147–155 (1982)

[13] Liu, J., Jiang, Y., Zhou, Z.: Cyclic scheduling of a single hoist in extended electroplating lines: A comprehensive integer programming solution. IIE Transactions **34**, 905–914 (2002)

[14] Manier, M.A., Bloch, C.: A classification for hoist schedling problems. International Journal of Flexible Manufacturing Systems **15**, 37–55 (2003)

[15] Manier, M.A., Varnier, C., Baptiste, P.: Constraint-base model for the cyclic multi-hoists scheduling problem. Production Planning and Control **11**, 244–257 (2000)

[16] Mocchia, L., Cordeau, J.F., Gaudioso, M., Laporte, G.: A branch-and-cut algorithm for the quay crane scheduling problem in a container terminal. Naval Research Logistics **53**, 45–59 (2005)

[17] Ng, W.C.: Crane scheduling in container yards with inter-crane interference. European Journal of Operational Research **164**, 64–78 (2005)

[18] Phillips, L.W., Unger, P.S.: Mathematical programming solution of a hoist scheduling problem. AIIE Transactions **8**, 219–321 (1976)

[19] Riera, D., Yorke-Smith, N.: An improved hybrid model for the generic hoist scheduling problem. Annals of Operations Research **115**, 173–191 (2002)

[20] Rodošek, R., Wallace, M.: A generic model and hybrid algorithm for hoist scheduling problems. In: M. Maher, J.F. Puget (eds.) Principle and Practice of Constraint Programming (CP 1998), vol. 1520. Springer, Pisa (1998)

[21] Varnier, C., Bachelu, A., Baptiste, P.: Resolution of the cyclic multi-hoists scheduling problem with overlapping partitions. INFOR **35**, 309–324 (1997)

[22] Yang, G., Ju, D.P., Zheng, W.M., Lam, K.: Solving multiple hoist scheduling problems by use of simulated annealing. Transportation Research Part B **36**, 537–555 (2001)

[23] Zhang, C., Wan, Y.W., Liu, J., Linn, R.J.: Dynamic crane deployment in container storage yards. Ruan Jian Xue Bao (Journal of Software) **12**, 11–17 (2002)

[24] Zhou, Z., Li, L.: A solution for cyclic scheduling of multi-hoists without overlapping. Annals of Operations Research **(online)** (2008)

Table 2: Quality of solution for various limits on the number of states, for random instances. Each figure is the average percent by which the solution value exceeds the optimal or best known value. Boldface figures indicate that optimality was proved. Five instances of each size were solved.

Cranes		Maximum number of states after pruning													
	Tasks	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}
1	20	**0***													
	50	0	0	**0***											
	100	0	0	0	**0***										
	200	0	0	0	0	**0***									
2	20	7.5	4.6	0.2	0.1	0	0	**0***							
	50	13	6.2	4.6	0.7	0	0	0	0	0	0	**0***			
	100	19	8.6	5.3	3.1	2.5	2.0	0.2	0.07	0	0	0	0	0	**0****
	200	21	14	8.3	7.1	3.9	2.5	1.3	1.1	0.6	0.2	0.2	0	0	0
3	20	24	16	2.1	0.9	0.9	0	0	0	**0**	**0*****				
	50	47	31	22	18	18	8.5	7.7	1.1	0.9	0.2	0	0	0	0
	100	62	32	38	17	13	7.0	4.2	3.5	3.2	2.4	1.4	0.6	0.3	0
	200	57[a]	40	34	19	21	16	12	6.5	5.3	4.6	3.2	2.5	0.4	0
4	20	111[b]	32	21	13	5.7	1.3	1.8	0.08	0.08	0.08	0	**0**	**0**	**0**†
	50	122	56	36	31	21	21	15	11	7.6	3.7	1.9	1.9	0.9	0
	100	75	64	52	43	28	25	18	11	8.7	6.9	5.2	3.3	1.9	1.1††
	200	102	66	42	38	30	19	15	10	12	7.8	6.9	3.1	1.1	1.1††

[a,b]No feasible solution found for 1 (a) or 2 (b) of the 5 instances.

*Optimality proved for all 5 instances.

**Optimality proved for 3 of the 5 instances.

***Optimality proved for 3 instances using 2^{12} states and 2 instances using 2^{13} states.

†Optimality proved for 2 instances using 2^{15} states, 1 using 2^{16} states, and 2 using 2^{17} states.

††This number is greater than zero because some instances obtained their best solution using fewer than 2^{17} states and a slightly worse solution using 2^{17} states.

Table 3: Average computation times (in seconds) for various limits on the number of states, for random instances. Boldface figures indicate the time required to prove optimality (specifics are in Table 2). Figures in boxes correspond to the state space size at which the optimal or best known solution was obtained for all 5 instances.

Cranes		Maximum number of states after pruning													
	Tasks	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}
1	20	**0.05**													
	50	0.05	0.05	**0.05**											
	100	0.06	0.1	0.06	**0.05**										
	200	0.06	0.07	0.1	0.07	**0.08**									
2	20	0.05	0.05	0.06	0.06	0.06	0.06	**0.06**							
	50	0.06	0.07	0.09	0.1	0.2	0.3	0.6	0.9	1.3	1.6	**1.8**			
	100	0.1	0.1	0.2	0.4	0.6	1.0	2.0	3.6	6.3	11	19	32	47	**69**
	200	0.3	0.5	0.8	1.4	2.5	4.6	9.8	20	35	65	114	206	380	1041
3	20	0.06	0.06	0.07	0.09	0.1	0.2	0.4	0.6	**0.8**	**1.2**				
	50	0.1	0.1	0.2	0.2	0.6	1.0	2.6	5.1	9.7	19	36	67	116	485
	100	0.2	0.3	0.6	1.0	1.9	3.6	8.9	19	38	75	145	291	565	1717
	200	0.9	1.7	3.1	5.5	10	19	45	99	195	398	757	1591	3110	6122
4	20	0.2	0.2	0.2	0.3	0.6	1.1	2.2	3.9	6.9	12	19	**30**	**43**	**51**
	50	0.6	0.4	0.8	1.4	2.7	4.3	9.4	20	38	77	151	320	664	1436
	100	1.0	1.8	4.2	6.0	11	17	42	86	166	342	680	1458	2993	6828
	200	2.7	5.6	11	20	30	61	112	229	427	948	1798	3598	9017	17503

Table 4: Quality of solution for various limits on the number of states for 6 instances obtained from industry. All instances schedule 30 jobs with 60 tasks on 2 cranes. Each figure is the percent by which the solution value exceeds the optimal or best known value. An ∞ indicates that no feasible solution was found.

Maximum number of states after pruning													
2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}
∞	∞	47	16	1.6	1.6	0.0	0	0	0	0	0	0	0
∞	49	32	14	16	11	6.4	0	0	0	0	0	0	0
46	40	33	12	6.9	5.5	0.3	0.1	0.1	0.1	0	0	0	0
∞	49	5.8	17	2.7	1.6	1.0	0.7	0.7	0.2	0	0	0	0
64	∞	38	15	5.0	11	2.7	2.6	1.8	1.8	1.6	1.1	0	0
∞	50	30	31	24	15	9.5	7.9	5.3	4.9	4.9	2.8	0.5	0.0

Table 5: Computation times (in seconds) for various limits on the number of states for 6 instances obtained from industry. Figures in boxes correspond to the state space size at which the best known solution was obtained. Optimality was not proved for any instances.

Maximum number of states after pruning													
2^4	2^5	2^6	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}	2^{15}	2^{16}	2^{17}
1.1	1.5	2.4	2.4	2.6	4.8	9.9	17	49	130	377	1730	5762	15299
0.9	1.8	1.5	2.2	3.3	5.2	7.1	19	51	142	401	1261	4847	11977
2.5	2.2	2.5	2.7	3.1	4.9	7.2	15	38	84	202	475	1183	2315
0.8	1.9	1.6	2.7	3.5	6.4	11	31	95	291	992	4288	15303	45865
1.4	0.9	2.1	2.3	3.3	5.4	12	20	58	152	375	1203	4324	12708
1.1	2.1	1.8	2.4	2.9	4.7	8.9	21	53	139	420	1290	4715	13692