Carnegie Mellon University
Research Showcase

Tepper School of Business

10-1-2008

Optimal Movement of Factory Cranes

Ionuț Aron
Cura Capital Management

Latife Genç-Kaya
Carnegie Mellon University

Iiro Harjunkoski
ABB Corporate Research Center

Samid Hoda
Carnegie Mellon University

John N. Hooker
Carnegie Mellon University, john@hooker.tepper.cmu.edu

Follow this and additional works at: http://repository.cmu.edu/tepper

Part of the Economic Policy Commons, and the Industrial Organization Commons

Recommended Citation
Aron, Ionuț; Genç-Kaya, Latife; Harjunkoski, Iiro; Hoda, Samid; and Hooker, John N., "Optimal Movement of Factory Cranes" (2008). *Tepper School of Business.* Paper 143.
http://repository.cmu.edu/tepper/143

This Working Paper is brought to you for free and open access by Research Showcase. It has been accepted for inclusion in Tepper School of Business by an authorized administrator of Research Showcase. For more information, please contact research-showcase@andrew.cmu.edu.

Optimal Movement of Factory Cranes

Ionuț Aron
Cura Capital Management, 1270 Avenue of the Americas
New York, NY 10020, USA, ionut.aron@gmail.com

Latife Genç-Kaya
Tepper School of Business, Carnegie Mellon University
Pittsburgh, PA 15213, USA, lgenc@andrew.cmu.edu

Iiro Harjunkoski
ABB Corporate Research Center, Wallstadter Str. 59
68526 Ladenburg, Germany, Iiro.Harjunkoski@de.abb.com

Samid Hoda
Tepper School of Business, Carnegie Mellon University
Pittsburgh, PA 15213, USA, shoda@andrew.cmu.edu

J. N. Hooker
Tepper School of Business, Carnegie Mellon University
Pittsburgh, PA 15213, USA, john@hooker.tepper.cmu.edu

October 2008

Abstract

We study the problem of finding optimal space-time trajectories for two factory cranes or hoists that move along a single overhead track. Each crane is a assigned a sequence of pickups and deliveries at specified locations that must be performed within given time windows. The cranes must be operated so as not to interfere with each other, although one crane may need to yield to another. The objective is generally to follow a production schedule as closely as possible. We show that only certain types of trajectories need be considered to obtain an optimal solution. This simplifies the operation of the cranes and enhances safety, because the cranes move according to predictable patterns. We present a specialized dynamic programming algorithm that solves the

problem. To control the state space size we use an innovative state space representation based on a cartesian product of intervals of states and an array of two-dimensional circular queues. The algorithm solves medium-sized instances of the problem to optimality and can be used to create benchmarks for tuning heuristic algorithms that solve larger instances.

1 Introduction

Manufacturing facilities frequently rely on track-mounted cranes to move in-process materials or equipment from one location to another. A typical arrangement, and the type studied here, allows one or more hoists to move along a single horizontal track that is normally mounted on the ceiling. Each hoist may be mounted on a crossbar that permits lateral movement as the crossbar itself moves longitudinally along the track. A cable suspended from the crossbar raises and lowers a lifting hook or other device.

When a production schedule for the plant is drawn up, cranes must be available to move materials from one processing unit to another at the desired times. The cranes may also transport cleaning or maintenance equipment. Since the cranes operate on a single track, they must be carefully scheduled so as not to interfere with each other. One crane may be required to yield (move out of the way) to permit another crane to pick up or deliver its load.

The problem is combinatorial in nature because one must not only compute a space-time trajectory for each crane, but must decide which crane yields to another and when. A decision made at one point may create a bottleneck that has unforeseen repercussions much later in the schedule. It is not unusual for production planners to put together a schedule that seems to allow ample time for crane movements, only to find that the crane operators cannot keep up with the schedule. As the cranes lag further and further behind, the production schedule must be adjusted in an ad hoc manner to allow them to catch up.

In this paper we analyze the problem of scheduling two cranes and describe an exact algorithm, based on dynamic programming, to solve it. The problem data consist of time windows, crane assignments, and job sequencing. That is, the problem specifies a release time and deadline for each job, an assignment of each job to a crane, and the order in which the jobs assigned to each crane are to be carried out. Several objectives are possible, but in our experience the primary goal has been to follow the production schedule as closely as possible.

This research is part of a larger project in which both heuristic and exact algorithms have been developed for use in crane scheduling software. The heuristic method makes crane assignment and sequencing decisions as well as computing space-time trajectories, and it is fast enough to accommodate large problems involving several cranes. However, once the assignments and sequencing are given, the heuristic method may fail to find feasible trajectories when they exist and reject good solutions as a result. We therefore found it important to solve the trajectory problem exactly for a given assignment and sequencing, in at least some of the smaller problem instances, as a check on the heuristic method. The exact algorithm has practical value in its own right, because two-crane problems are common in industry, and the algorithm solves instances of respectable size within a minute or so. Nonetheless, we see it as having an equally important role in the creation of benchmarks against which heuristic methods can be tested and tuned for best performance.

We begin by deriving structural results for the two-crane problem that restrict the trajectories that must be considered to certain *canonical* trajectories. This not only makes the problem tractable for dynamic programming by reducing the state space, but it also accelerates the heuristic solution of larger two-crane problems by dramatically reducing the possibilities that must be enumerated. Moreover, the canonical trajectories simplify the operation of the cranes, and enhance safety, by restricting the crane movements to certain predictable patterns. For example, cranes always move at the same speed, never stand at rest except at a pickup or delivery point, and never yield to another crane except when moving alongside that crane (at a safe distance). In addition, the left crane keeps to the left as much as possible, and the right crane to the right.

We then describe a dynamic programming algorithm for the optimal trajectory problem. The state space is large, due to the fine space-time granularity with which the problem must be solved, as well as the necessity of keeping up with which task a crane is performing and how long it has been processing that task. To deal with these complications we introduce a novel state space description that represents many states implicitly as a cartesian product of intervals. The state space is efficiently stored and updated in a data structure that uses an array of two-dimensional circular queues. These enhancements accelerate solution by at least an order of magnitude and allow us to solve problems of realistic size within a reasonable time. The paper concludes with computational results and directions for further research.

2 Previous Work

To our knowledge, no previous work computes space-time trajectories that allow cranes to yield, and none obtains structural results that restrict the types of trajectories that must be considered. The literature on crane scheduling tends to cluster around two types of problems: movement of materials from one vat to another in an electroplating or similar process (typically referred to as *hoist scheduling* problems), and loading and unloading of container ships in a port.

A classification scheme for hoist scheduling problems appears in Manier and Bloch (2003). It is assumed in these problems that each item visits the same vats in the same order, in most cases consecutively. The objective is to minimize cycle time, which is the time lapse between the entry of two consecutive items into the system. Much research in this area deals with the single-hoist cyclic scheduling problem (Phillips and Unger, 1976; Armstrong *et al.*, 1994; Baptiste *et al.*, 1994; Lei and Wang, 1994; Ng, 1996; Ng and Leung, 1997; Chen *et al.*, 1998; Liu *et al.*, 2002). Because there is only one hoist, the space-time trajectory of the hoist is not an issue, so long as it picks up and delivers items at the right time. Even this restricted problem is NP-complete (Lei and Wang, 1989).

Several papers deal with cyclic two-hoist and multi-hoist problems. One approach partitions the vats into contiguous subsets, assigns a hoist to each subset, and schedules each hoist within its partition (Wei and Wang, 1991; Yang *et al.*, 2001; Zhou and Li, 2008). A better solution can generally be obtained, however, by allowing a vat to be served by more than one hoist. This has been accomplished by careful scheduling of the hoists to avoid collisions, based on a case-by-case analysis of the various ways that they can approach each other (Lei *et al.*, 1993; Varnier *et al.*, 1997; Rodošek and Wallace, 1998; Leung and Zhang, 2003; Che and Chu, 2004; Leung *et al.*, 2004; Liu and Jiang, 2005). None of these studies compute space-time trajectories of the hoists or allow one hoist to yield to another. They avoid collisions by setting departure and arrival times so that no interference is possible when hoists go directly from one vat to the next.

Although we do not address the assignment of tasks to cranes in the present paper, our problem is otherwise more general than hoist scheduling problems in several respects: (a) rather than requiring that every item visit the same sequence of stations, we allow each job to specify an arbitrary subset of tasks in any order; (b) we solve for an optimal space-time trajectory of each crane that allows it to make additional movements in order to yield to the other crane; (c) we accommodate release times and deadlines for the

jobs; and (d) we allow for a variety of objective functions.

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 in several respects. 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 (Daganzo, 1989; Peterkofsky and Daganzo, 1990; Mocchia *et al.*, 2005; Kim and Park, 2004; Lim *et al.*, 2004; Zhu and Lim, 2006).

Yard cranes are typically mounted on wheels and can follow certain paths in the dockyard to move containers from one location to another. Existing solution approaches schedule departure and arrival times for the cranes so that they do not interfere with each other, but the actual space-time trajectories are not examined (Zhang *et al.*, 2002; Ng, 2005).

3 The Optimal Trajectory Problem

In practice, a crane scheduling problem typically consists of a number of *jobs*, each of which specifies several *tasks* to be performed consecutively. For example, a job may require that a crane pick up a ladle at one location, fill the ladle with molten metal at a second location, deliver the metal to a third location, and then return the ladle. Tasks may also involve maintenance and cleaning activities. The same crane must perform all the tasks in a job and must remain stationary at the appropriate location while processing each task.

The location and processing time for each task are given, as are release times and deadlines. We also suppose that each job has been pre-assigned to a certain crane, and the jobs assigned to a crane must be performed in a fixed order. Each job assigned to a given crane must finish before the next job assigned to that crane begins.

In this study we explicitly account only for the longitudinal movements of the crane along the track. We assume that the crane has time to make the necessary lateral and vertical movements as it moves from one task location to another. This results in little loss of generality, because any additional time necessary for lateral or vertical motion can be built into the processing time for the task.

The problem data are:

$$
\begin{aligned}
&R_j = \text{release time of task } j \\
&D_j = \text{deadline for task } j \\
&L_j = \text{processing location (stop) for task } j \\
&P_j = \text{processing time for task } j \\
&c(j) = \text{crane assigned to task } j \\
&v = \text{maximum crane speed} \\
&0, L_{\max} = \text{leftmost and rightmost crane locations} \\
&\Delta = \text{minimum crane separation} \\
&\Delta t = \text{time increment}
\end{aligned}
$$

Note that we refer to the processing location of a task as a *stop*.

If release times $\bar{R}_i$ and deadlines $\bar{D}_i$ are given for each job i rather than each task j, then the task release time R_j is the earliest possible start time for that task:

$$R_j = \bar{R}_i + \sum_{\ell=k}^{j-1} \left(P_\ell + \frac{|L_{\ell+1} - L_\ell|}{v} \right)$$

where k is the first task in job i. Similarly, the task deadline is the latest possible finish time, given the job deadline:

$$D_j = \bar{D}_i - \sum_{\ell=j+1}^{k'} \left(\frac{|L_\ell - L_{\ell-1}|}{v} + P_\ell \right)$$

where k' is the last task in job i.

We suppose for generality that there are cranes $1, \ldots, m$, where crane 1 is the *left crane* and crane m the *right crane*, although we solve the problem only for $m = 2$. $T_{\max}$ is the length of the time horizon. The problem variables are:

$$
\begin{aligned}
&x_{ct} = \text{position of crane } c \text{ at time } t \\
&y_{ct} = \text{task being processed by crane } c \text{ at time } t \text{ (0 if none)} \\
&\tau_j = \text{time at which task } j \text{ starts processing}
\end{aligned}
$$

Task j therefore finishes processing at time $\tau_j + P_j$. We assume that the tasks are indexed so that tasks assigned to a given crane are processed in order of increasing indices.

The problem with n tasks and m cranes may now be stated

$$
\begin{array}{ll}
\min\ f(\tau) & \\
\left.\begin{array}{l} 0 \le x_{ct} \le L_{\max} \\ x_{ct} - v\Delta t \le x_{c,t+\Delta t} \le x_{ct} + v\Delta t \\ y_{ct} > 0 \Rightarrow x_{ct} = L_{y_{ct}} \end{array}\right\} \text{all } c,t & \begin{array}{l} (a) \\ (b) \\ (c) \end{array} \\
x_{ct} \le x_{c+1,t} - \Delta,\ c = 1,\ldots,m-1,\ \text{all } t & (d) \\
\left.\begin{array}{l} R_j \le \tau_j \le D_j - P_j,\ \text{all } j \\ y_{c(j)t} = j,\ t = \tau_j,\ldots,\tau_j + P_j - \Delta t \end{array}\right\} \text{all } j & \begin{array}{l} (e) \\ (f) \end{array} \\
\{c(j) = c(j'),\ j < j'\} \Rightarrow \tau_j < \tau_{j'},\ \text{all } j, j' & (g) \\
y_{ct} \in \{0,\ldots,n\},\ \text{all } c,t &
\end{array}
\tag{1}
$$

Constraint (a) requires that the cranes stay on the track, and (b) that their speed be within the maximum. Constraint (c) implies that a crane must be at the right location when it is processing a task. Constraint (d) makes sure the cranes do not interfere with each other. Constraint (e) enforces the time windows, and (f) ensures that processing continues for the required amount of time once it starts. Constraint (g) requires that the tasks assigned to a crane be processed in the right order.

We assume that the objective $f(\tau)$ is a function only of the task start times, because this is sufficient for practical application and allows us to prove the structural results below. Generally one is interested in conforming to the production schedule as closely as possible. For instance, one might minimize the lapse between the release time R_k and the start time τ_k of the first task k in a job, or the lapse between the earliest finish time $R_{k'} + P_{k'}$ and the completion time $\tau_{k'} + P_{k'}$ of the last task k' in a job, or some combination of these. We used the more general objective

$$
f(\tau) = \sum_j \alpha_j \left(\tau_j - R_j\right) \tag{2}
$$

but normally set α_j to a positive value only when task j is the first or last task of a job. One might also be concerned that the cranes make no unnecessary movements. We can incorporate this into the objective function only if it has the form $f(x, \tau)$, but there is no need to do so. By restricting the cranes to the canonical trajectories defined by the structural results below, we avoid unnecessary movements.

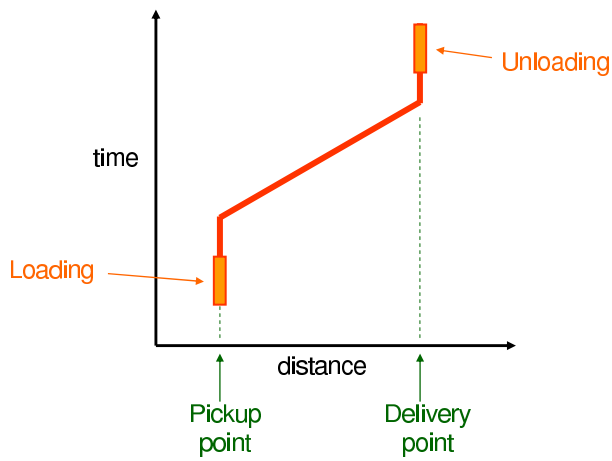

Figure 1: Sample space-time trajectory for one task. The shaded vertical bars denote processing, which in this case consists of loading and unloading.

4 Canonical Trajectories

Optimal control of the cranes is much easier to calculate when it is recognized that only certain trajectories need be considered, namely those we call canonical trajectories. We will show that when there are two cranes, some pair of canonical trajectories is optimal.

Let a *processing schedule* for a given crane consist of the vector τ of task start times. We define the *extremal* trajectory for the left crane, with respect to a given processing schedule, to be one that observes the processing schedule and that, while not processing a task, always follows the leftmost trajectory that never moves in the direction away from the next stop. For example, the trajectory in Figure 1 is not extremal because the crane moves to the right sooner than necessary.

More precisely, if the next stop (processing location) is to the right of the current stop, then the left crane follows the canonical trajectory if it leaves the current stop as late as possible so as to arrive at next stop just as processing starts (Fig. 2a). If the next stop is to the left of the current one, the crane leaves the current stop as early as possible (Fig. 2b). Thus at any time the crane is either stationary or moving at maximum speed. The extremal trajectory for the right crane follows the rightmost trajectory: it leaves the current stop as late as possible if moving to the left, and as early

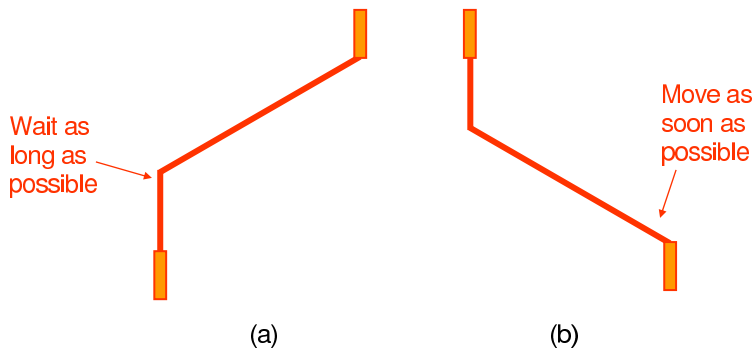

Figure 2: Extremal trajectory for the left crane (a) when the destination is to the right of the origin, and (b) when the destination is to the left of the origin.

as possible if moving to the right.

A trajectory for the left crane is *canonical* with respect to the right crane if at each moment it is the rightmost of (a) the extremal trajectory for the left crane and (b) the trajectory that runs parallel to and just to the left of the right crane's trajectory (Fig. 3). More precisely, trajectory x_1' is canonical for the left crane, with respect to trajectory x_2 for the right crane, if the extremal trajectory $\bar{x}_1$ for the left crane satisfies $x_1'(t) = \min\{\bar{x}_1(t), x_2(t) - \Delta\}$ at each time t. A trajectory for the right crane is canonical with respect to the left crane if it is the leftmost of the extremal trajectory for the right crane and the left crane's trajectory. That is, $x_2'(t)$ is canonical if $x_2'(t) = \max\{\bar{x}_2(t), x_1(t)+\Delta\}$, where $\bar{x}_2(t)$ is the extremal trajectory. Finally, a pair of trajectories is canonical if the trajectories are canonical with respect to each other.

Theorem 1 *If the two-crane problem (1) has an optimal solution, then some optimal pair of trajectories is canonical.*

Proof. The idea of the proof is to replace the left crane's optimal trajectory with a canonical trajectory with respect to the right crane's optimal trajectory. Then assign the right crane a canonical trajectory with respect to the left crane's new trajectory, and finally assign the left crane a canonical trajectory with respect to the right crane's new trajectory. At this point it is shown that the trajectories are canonical with respect to each other. Since

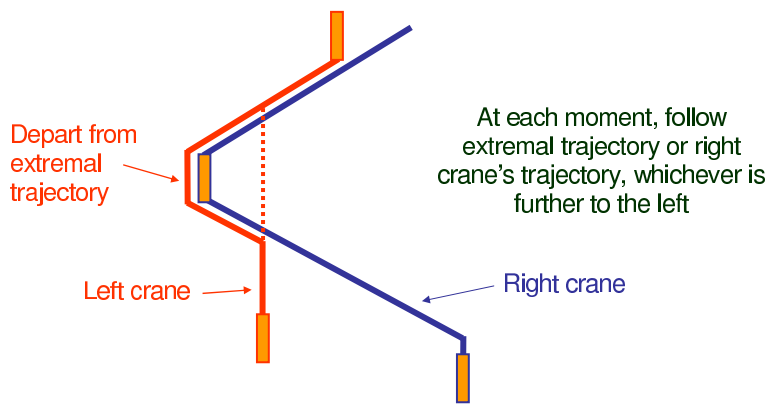

Figure 3: Canonical trajectory for the left crane (leftmost solid line).

these replacements never change the objective function value, the canonical trajectories are optimal, and the theorem follows.

Thus let $x^* = (x_1^*, x_2^*)$ be a pair of optimal trajectories for a two-crane problem. Let $\bar{x}_1$, $\bar{x}_2$ be extremal trajectories for the left and right cranes with respect to the processing schedules in the optimal trajectories.

Consider the canonical trajectory x_1' for the left crane with respect to x_2^*, which is given by $x_1'(t) = \min\{\bar{x}_1(t), x_2^*(t) - \Delta\}$. We claim that (x_1', x_2^*) is optimal. First note that it has the same objective function value as x^*, since x_1' has the same processing schedule as x_1^*. Furthermore, it is feasible because the cranes do not interfere with each other, and the speed of the left crane is never greater than v. The cranes do not interfere with each other because $x_1'(t) \leq x_2^*(t) - \Delta$ for all t, due to $x_1'(t) \leq x_1^*(t)$ and $x_1^*(t) \leq x_2^*(t) - \Delta$. To show that the speed of the left crane is never more than v it suffices to show that the average speed in the left-to-right direction between any pair of time points t_1, t_2 is never more than v, and similarly for the average speed in the right-to-left direction. The former is

$$\begin{aligned}\frac{x_1'(t_2) - x_1'(t_1)}{t_2 - t_1} &= \frac{\min\{\bar{x}_1(t_2), x_2^*(t_2) - \Delta\} - \min\{\bar{x}_1(t_1), x_1^*(t_1) - \Delta\}}{t_2 - t_1} \\ &\leq \max\left\{\frac{\bar{x}_1(t_2) - \bar{x}_1(t_1)}{t_2 - t_1}, \frac{x_2^*(t_2) - x_2^*(t_1)}{t_2 - t_1}\right\} \leq v\end{aligned}$$

where the first inequality is due to the fact that

$$\min\{a, b\} - \min\{c, d\} \leq \max\{a - c, b - d\}$$

for any a, b, c, d, and the second inequality due to the fact that $\bar{x}_1$ and x_2^* are feasible trajectories. The speed in the right-to-left direction is similarly bounded.

Now consider the canonical trajectory x_2' for the right crane with respect to x_1', given by $x_2'(t) = \max\{\bar{x}_2(t), x_1'(t) + \Delta\}$. It can be shown as above that (x_1', x_2') is optimal.

Finally, let x_1'' be the canonical trajectory for the left crane with respect to x_2', given by $x_1''(t) = \min\{\bar{x}_1(t), x_2'(t) - \Delta\}$. Again (x_1'', x_2') is optimal. Since x_1'' is canonical with respect to x_2', to prove the theorem it suffices to show that x_2' is canonical with respect to x_1''; that is, $\max\{\bar{x}_2(t), x_1''(t)+\Delta\} = x_2'(t)$ for all t. To show this we consider four cases for each time t.

Case 1: $\bar{x}_1(t) + \Delta \leq \bar{x}_2(t)$. We first show that

$$(x_1''(t), x_2'(t)) = (\bar{x}_1(t), \bar{x}_2(t)) \tag{3}$$

by considering the subcases (a) $x_2^*(t) \leq \bar{x}_1(t)$ and (b) $\bar{x}_1(t) < x_2^*(t)$. In subcase (a),

$$x_1'(t) = \min\{\bar{x}_1(t), x_2^*(t) - \Delta\} = x_2^*(t) - \Delta$$

which implies

$$x_2'(t) = \max\{\bar{x}_2(t), x_1'(t) + \Delta\} = \max\{\bar{x}_2(t), x_2^*(t)\} = \bar{x}_2(t)$$

and

$$x_1''(t) = \min\{\bar{x}_1, x_2'(t) - \Delta\} = \min\{\bar{x}_1, \bar{x}_2(t) - \Delta\} = \bar{x}_1(t)$$

In subcase (b), $x_1'(t) = \bar{x}_1(t)$, which implies $x_2'(t) = \max\{\bar{x}_2(t), \bar{x}_1(t)+\Delta\} = \bar{x}_2(t)$ and again $x_1''(t) = \bar{x}_1$. Now from (3) we have

$$\max\{\bar{x}_2(t), x_1''(t) + \Delta\} = \max\{\bar{x}_2(t), \bar{x}_1(t) + \Delta\} = \bar{x}_2(t) = x_2'(t)$$

as claimed.

The remaining cases suppose $\bar{x}_2(t) < \bar{x}_1(t) + \Delta$ and consider the situations in which $x_2^*(t)$ is less than or equal to $\bar{x}_2(t)$, between $\bar{x}_2(t)$ and $\bar{x}_1(t) + \Delta$, and greater than $\bar{x}_1(t) + \Delta$.

Case 2: $x_2^*(t) \leq \bar{x}_2(t) < \bar{x}_1(t)+\Delta$. It can be checked that $(x_1''(t), x_2'(t)) = (\bar{x}_2(t) - \Delta, \bar{x}_2(t))$ and $\max\{\bar{x}_2(t), x_1''(t) + \Delta\} = \max\{\bar{x}_2(t), \bar{x}_2(t)\} = \bar{x}_2(t) = x_2'(t)$, as claimed.

Case 3: $\bar{x}_2(t) < x_2^*(t) \leq \bar{x}_1(t)+\Delta$. Here $(x_1''(t), x_2'(t)) = (x_2^*(t)-\Delta, x_2^*(t))$ and $\max\{\bar{x}_2(t), x_1''(t) + \Delta\} = \max\{\bar{x}_2(t), x_2^*(t)\} = x_2^*(t) = x_2'(t)$.

Case 4: $\bar{x}_2(t) < \bar{x}_1(t)+\Delta < x_1^*(t)$. Here $(x_1''(t), x_2'(t)) = (\bar{x}_1(t), \bar{x}_1(t)+\Delta)$ and $\max\{\bar{x}_2(t), x_1''(t) + \Delta\} = \max\{\bar{x}_2(t), \bar{x}_1(t) + \Delta\} = \bar{x}_1(t) + \Delta = x_2'(t)$. This completes the proof.

The properties of canonical trajectories allow us to consider a very restricted subset of trajectories when computing the optimum.

Corollary 2 *If the two-crane problem has an optimal solution, then there is an optimal solution with the following characteristics:*

(a) While not processing a task, the left (right) crane is never to the right (left) of both the previous and the next stop.

(b) While not processing a task, the left (right) crane is moving in a direction toward its next stop if it is to the right (left) of the previous or next stop.

(c) A crane never moves in the direction away from its next stop unless it is adjacent to the other crane at all times during such motion.

(d) While not processing a task, the left (right) crane can be stationary only if it is (i) at the previous or the next stop, whichever is further to the left (right), or (ii) adjacent to the other crane.

Proof.

(a) If crane 1 (the left crane) is to the right of both its previous and next stop at some time t, then $x_1(t) > \bar{x}_1(t)$. This is impossible in a canonical trajectory, in which $x_1(t) = \min\{\bar{x}_1(t), x_2(t) - \Delta\}$. The argument is similar for crane 2.

(b) Suppose crane 1 is to the right of its previous stop. Due to (a), it is not to the right of its next stop, which must therefore be to the right of the previous stop. We cannot have $x_1(t) > \bar{x}_1(t)$ as in (a), and we cannot have $x_1(t) < \bar{x}_1(t)$, since this means the crane cannot reach its next stop in time. So crane 1 is on its canonical trajectory, which means that it is moving toward its next stop. The argument is similar if crane is to the right of the next stop.

(c) From (a) and (b), at a given time t crane 1 can be moving in the direction opposite its next stop only if it is at or to the left of both the previous and next stops. This means that it will be to the left of both at time $t + \Delta t$, so that $x_1(t + \Delta t) < \bar{x}_1(t + \Delta t)$. But since

$$x_1(t + \Delta t) = \min\{\bar{x}_1(t + \Delta t), x_2(t + \Delta t) - \Delta\}$$

this means $x_1(t+\Delta t) = x_2(t+\Delta t) - \Delta$, and crane 1 is adjacent to the other crane. Since crane 1 is moving left between t and $t+\Delta t$, it must be adjacent to the other crane at time t as well.

(d) From (a) and (b), a stationary crane 1 must be at or to the left both the previous and the next stop. If it is at one of them, then (i) applies. If it is to the left of both, then $x_1(t) < \bar{x}_1(t)$, which again implies that $x_1(t) = x_2(t) - \Delta$, and (ii) holds.

5 Dynamic Programming Recursion

The optimal control problem for the cranes is not simply a matter of computing an optimal space-time trajectory. It is complicated by three factors: (a) each crane must perform tasks in a certain order; (b) each task must be performed at a certain location for a certain amount of time; and (c) the cranes must not interfere with each other. We chose to solve the problem with dynamic programming because it has the flexibility to deal with these additional constraints while preserving optimality (up to the precision allowed by the space and time granularity). The drawback is a potentially exploding state space, but we will show how to keep it under control for problems of reasonable size. To simplify notation, we assume from here out that $\Delta t = 1$.

There are three state variables for each crane. Two of them are x_{ct} and y_{ct} as defined in model (1), and the third is

$$u_{ct} = \begin{cases} \text{amount of time crane } c \text{ will have been processing at time } t+1 \\ (0 \text{ if the crane is neither processing nor starts processing at time } t) \end{cases}$$

In principle the recursion is straightforward, although a practical implementation requires careful management of state transitions and data structures. Let $x_t = (x_{1t}, x_{2t})$, and similarly for y_t and u_t. Also let $z_t = (x_t, y_t, u_t)$. It is convenient to use a forward recursion:

$$g_{t+1}(z_{t+1}) = \min_{z_t \in S^{-1}(z_{t+1})} \{h(t, y_t, u_t) + g_t(z_t)\} \tag{4}$$

where $g_t(z_t)$ is the cost of an optimal trajectory between the initial state and state z_t at time t, $h(t, y_t, u_t)$ is the cost incurred at time t, and $S^{-1}(z_{t+1})$ is the set of states at time t from which the system can move to state z_{t+1} at time $t+1$. Given the cost function (2), the cost $h(t, y_t, u_t)$ is $\sum_c h_c(t, y_t, u_t)$, where

$$h_c(t, y_t, u_t) = \begin{cases} \alpha_{y_{ct}}(t - R_{y_{ct}}) & \text{if } u_{ct} = 1 \\ 0 & \text{otherwise} \end{cases}$$

The boundary condition is

$$g_0(z_0) = 0$$

when z_0 is the initial state. The optimal cost is $g_{T_{\max}}(z_{T_{\max}})$, where $z_{T_{\max}}$ is the desired terminal state.

For each state z_{t+1} the recursion (4) computes the minimum $g_{t+1}(z_{t+1})$ and the state $z_t = s_{t+1}^{-1}(z_{t+1})$ that achieves the minimum. Thus $s_{t+1}^{-1}(z_{t+1})$ points to the state that would precede z_{t+1} in the optimal trajectory if z_{t+1} were in the optimal trajectory. For a basic recursion, the cost table $g_{t+1}(\cdot)$ is stored in memory until $g_{t+2}(\cdot)$ is computed, and then released (this is modified in the next section). Thus only two consecutive cost tables need be stored in memory at any one time. The table $s_{t+1}^{-1}(\cdot)$ of pointers is stored offline. Then if z_T is the final state, we can retrace the optimal solution in reverse order by reading the tables $s_{t+1}^{-1}(\cdot)$ into memory one at a time and setting $z_t = s_{t+1}^{-1}(z_{t+1})$ for $t = N-1, N-2, \ldots, 0$.

6 Reduction of the State Space

We can substantially reduce the size of the state space if we observe that in practical problems, the cranes spend much more time processing than moving. The typical processing time for a state ranges from two to five minutes (sometimes much longer), while the typical transit time to the next location is well under a minute. Furthermore, the state variables representing location and task assignment (x_{ct} and y_{ct}) cannot change while the crane is processing.

These facts suggests that the processing time state variable u_{ct} should be replaced by an *interval* $U_{ct} = [u_{ct}^{lo}, u_{ct}^{hi}] = \{u_{ct}^{lo}, u_{ct}^{lo}+1, \ldots, u_{ct}^{hi}\}$ of consecutive processing times. A single "state" $(x_t, u_t, U_{ct}) = (x_t, u_t, (U_{1t}, U_{2t}))$ now represents a set of states, namely the Cartesian product

$$\{(x_t, y_t, (i, j)) \mid i \in U_{1t},\, j \in U_{2t}\}$$

The possible state transitions for either crane c are shown in Table 1. The transitions in the table are feasible only if they satisfy other constraints in the problem, including those based on time windows, the physical length of the track, and interactions with the other crane. The transitions can be explained, line by line, as follows:

1. Because the processing time interval is the singleton $[0, 0]$, the crane can be in motion and can in particular move to either adjacent location. When it arrives at the next location, the currently assigned task can

start processing if the crane is in the correct position, in which case the state interval is $U_{ct} = [0, 1]$ to represent two possible states: one in which the task does not start processing at time $t + 1$, and one in which it does (the interval is $[1, 1]$ if the deadline forces the task to start processing at $t + 1$). If the crane is in the wrong location for the task, the state remains $[0, 0]$.

2. None of the states in the interval $[0, u_2]$ allow processing to finish at time $t+1$. So all of the processing time states advance by one—except possibly the zero state, in which processing has not yet started and can be delayed yet again if the deadline permits it.

3. The last state in the interval $[0, P_{y_{ct}}]$ allows processing to finish at time $t + 1$. This state splits off from the interval and assumes one of the processing state intervals in line 1. The other states evolve as in line 2.

4. Because the task is underway in all states, all processing times advance by one.

5. This is similar to line 3 except that there is no zero state.

There is no need to store a pointer $s^{-1}_{t+1}(x_t, y_t, (i, j))$ for every state $(x_t, y_t, (i, j))$ in (x_t, y_t, U_t). This is because when $u_{ct} \geq 2$, the state of crane c preceding (x_{ct}, y_{ct}, u_{ct}) must be $(x_{ct}, y_{ct}, u_{ct} - 1)$. Thus we store $s^{-1}_{t+1}(x_t, y_t, (i, j))$ only when $i \leq 1$ or $j \leq 1$.

However, we must store the cost $g_{t+1}(x_t, y_t, (i, j))$ for every (i, j), because it is potentially different for every (i, j). Fortunately, it is not necessary to update this entire table at each time period, because most of the costs evolve in a predictable fashion. If $i, j \geq 2$, then

$$g_{t+1}(x_y, y_t, (i, j)) = g_t(x_t, y_t, (i - 1, j - 1))$$

So for each pair of tasks (y, y') we maintain a two-dimensional circular queue $Q_{yy'}(\cdot, \cdot)$ in which the cost

$$g_{t+1}((L_y, L_{y'}), (y, y'), (i, j)) \tag{5}$$

for $i, j \geq 2$ is stored at location

$$Q_{yy'}((t + i - 2) \bmod M, (t + j - 2) \bmod M)$$

where M is the size of the array $Q_{yy'}(\cdot, \cdot)$ (i.e., the longest possible processing time). In each period we insert the cost (5) into Q only for pairs (i, j) in

Table 1: Possible state transitions for crane c using an interval-valued state variable for processing time.

State at time t	*State at time t + 1*
1. $(x_{ct}, y_{ct}, [0,0])$	$(x', y_{ct}, [0,0])^1$ or $(x', y_{ct}, [0,1])^{1,2}$ or $(x', y_{ct}, [1,1])^{1,2,3}$
2. $(x_{ct}, y_{ct}, [0,u_2])^4$	$(x_{ct}, y_{ct}, [0,u_2+1])$ or $(x_{ct}, y_{ct}, [1,u_2+1])^{2,4}$
3. $(x_{ct}, y_{ct}, [0,P_{y_{ct}}])$	$(x_{ct}, y_{ct}, [0,P_{y_{ct}}])$ or $(x_{ct}, y_{ct}, [1,P_{y_{ct}}])^3$ or $(x_{ct}, y', [0,0])^5$ or $(x_{ct}, y', [0,1])^{2,5}$ or $(x_{ct}, y', [1,1])^{2,3,5}$
4. $(x_{ct}, y_{ct}, [u_1,u_2])^{4,6}$	$(x_{ct}, y_{ct}, [u_1+1,u_2+1])$
5. $(x_{ct}, y_{ct}, [u_1,P_{y_{ct}}])^6$	$(x_{ct}, y_{ct}, [u_1+1,P_{y_{ct}}])$ or $(x_{ct}, y', [0,0])^5$ or $(x_{ct}, y', [0,1])^{2,5}$ or $(x_{ct}, y', [1,1])^{2,3,5}$

[1]The next location x' is $x_{ct}-1$, x_{ct}, or $x_{ct}+1$.

[2]This transition is possible only if task y_{ct} processes at location x'.

[3]This transition is possible only if task y_{ct} can start no later than time $t+1$.

[4]Here $0 < u_2 < P_{y_{ct}}$.

[5]Task y' is the task that follows task y_{ct} on crane c.

[6]Here $u_1 > 0$.

which $i = 2$ or $j = 2$; the costs for other pairs with $i, j \geq 2$ were computed in previous periods. Thus only one row and one column of the Q array are altered in each time period, which substantially reduces computation time. When $i \leq 1$ or $j \leq 1$, the cost (5) is stored as a table entry $g_{t+1}(x_t, y_t, (i,j))$ that is updated at every time period, as with pointers.

The array $Q_{yy'}(\cdot,\cdot)$ is created when the state $((L_y, L_{y'}), (y, y'), (i,j))$ is first encountered with $i, j \geq 2$. The array is kept in memory over multiple periods until it is no longer updated, at which time it is deleted.

7 Experimental results

We report computational tests on a representative problem that is based on an actual industry scheduling situation. There are 60 jobs, each of which contains from two to eight tasks. We obtain smaller instances by scheduling only some of the jobs, namely the first ten (in order of release time), the first twenty, and so forth. Results on other problems we have examined are

Table 2: Computational results for subsets of the 60-job problem.

Jobs	Time window (mins)	Computation time (sec)
10	40	6.8
20	40	7.6
30	40	15.8
40	40	16.7
50	40	18.8
60	95	48.1

similar.

Release times were obtained from the production schedule, but no deadlines were given. We initially set the deadline of each job to be 40 minutes after each release time, with the expectation that these may have to be relaxed to obtain a feasible solution.

We divided the 108.5-meter track into ten equal segments, so that each distance unit represents 10.85 meters. Each crane can traverse the length of the track in about one minute. Because we want the crane to move one distance unit for each time unit, we set the time unit at six seconds. The 60-job schedule requires about four hours to complete, which means that the dynamic programming procedure has about $T_{\max} = 2400$ time stages.

Table 2 shows computation times obtained on a desktop PC running Windows XP with a Pentium D processor 820 (2.8 GHz). The assignment and sequencing of jobs used in each instance is the best one that was obtained by a heuristic procedure. Feasible solutions were found for all the instances except the full 60-job problem. To obtain a feasible solution of this problem, we were obliged to enlarge the time windows from 40 to 95 minutes by postponing the deadlines. This illustrates the combinatorial nature of the problem, because the addition of only ten jobs created new bottlenecks that delayed at least one job nearly 95 minutes beyond its release time. Wider time windows result in a larger state space and thus greater computation time. Nonetheless, the 60-job problem with 95-minute windows was solved in well under a minute.

The optimal trajectories for selected instances appear in Figs. 4–6. The horizontal axis represents distance along the track in 10.85-meter units. The vertical axis represents time in 6-second units. Thus the schedule for the

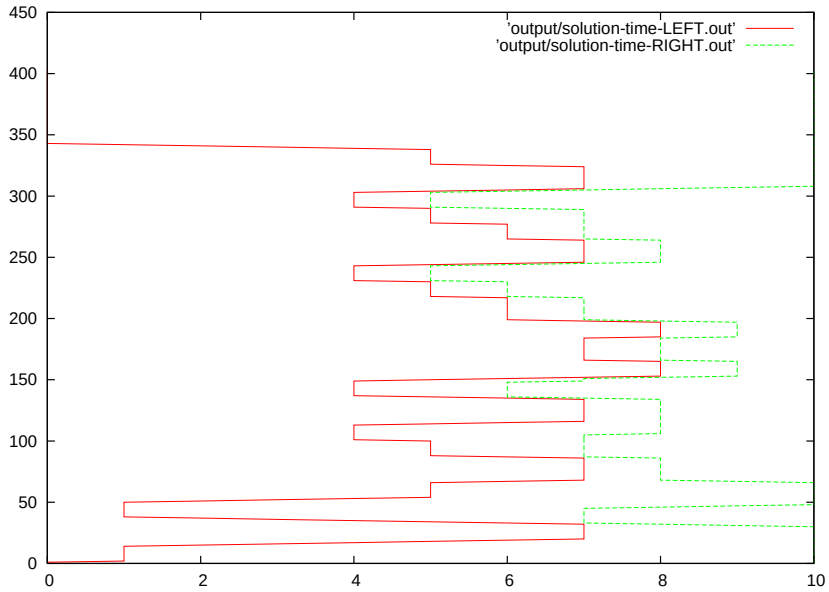

Figure 4: Optimal solution of the 10-job instance.

60-job problem spans about 2300 time units, or 230 minutes. The space-time trajectory of the left crane appears as a solid line, and as a dashed line for the right crane. The left crane begins and ends at the leftmost position, and analogously for the right crane. Note that the cranes are at rest most of the time. The trajectories are canonical trajectories as defined above, which ensures a certain consistency in the way the two cranes interact. In the 60-job instance, the left crane finishes before the right crane, which may indicate a poor allocation of jobs to cranes.

Figures 7–9 track the evolution of state space size over time. The horizontal axis corresponds to time stages, which again are separated by six seconds. The number of time stages exceeds the duration of the optimal trajectory, because trajectories with longer durations are considered in the solution process. The vertical axis is the number of states at each time stage. The state space size remains quite reasonable, never exceeding 2000 states, even though the theoretical maximum is astronomical.

We found computation time to be sensitive to the width of the time windows. Typically, only a few time windows must be wide to allow a feasible solution, because only a few jobs must be delayed so that others may be completed on time. Yet it is difficult or impossible to predict which are the critical jobs. It is therefore necessary to be able to solve problems in

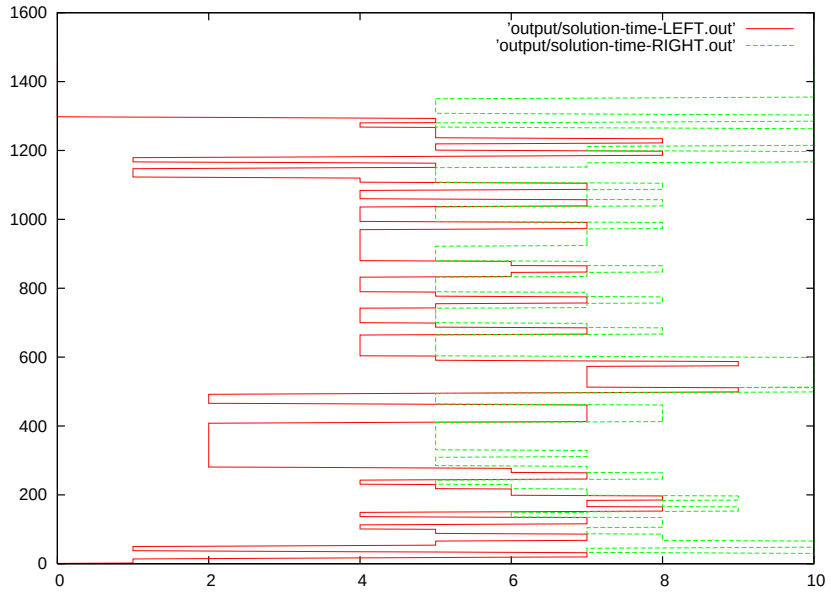

Figure 5: Optimal solution of the 30-job instance.

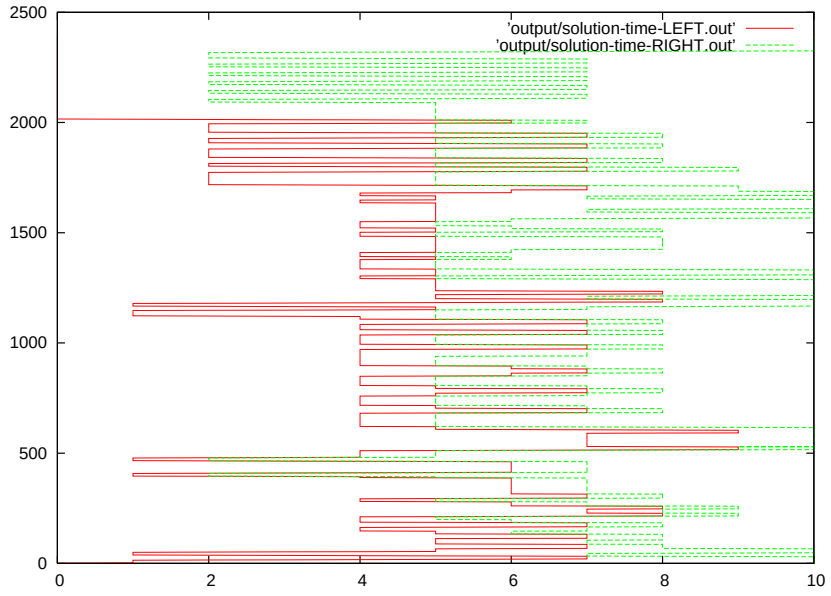

Figure 6: Optimal solution of the 60-job instance.

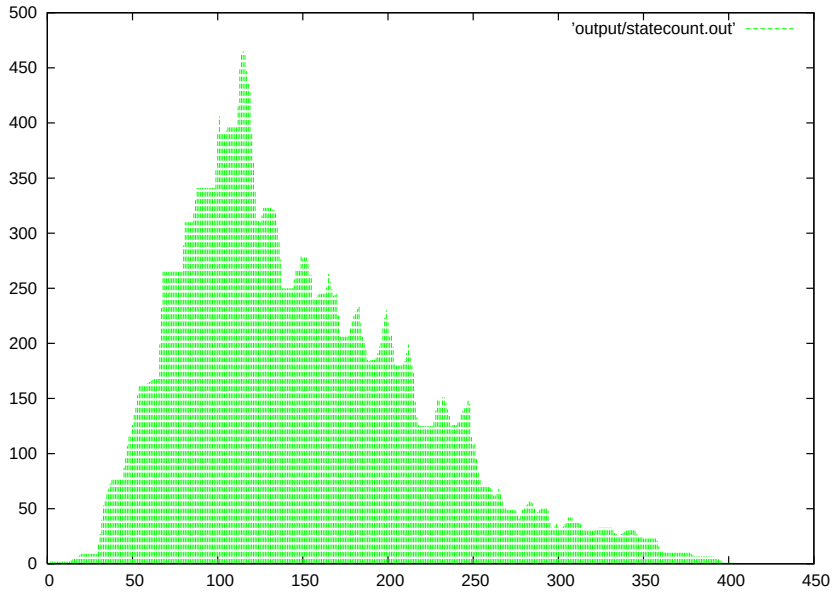

Figure 7: Evolution of the state space size for the 10-job instance. The horizontal axis is the time stage, and the vertical axis the number of states.

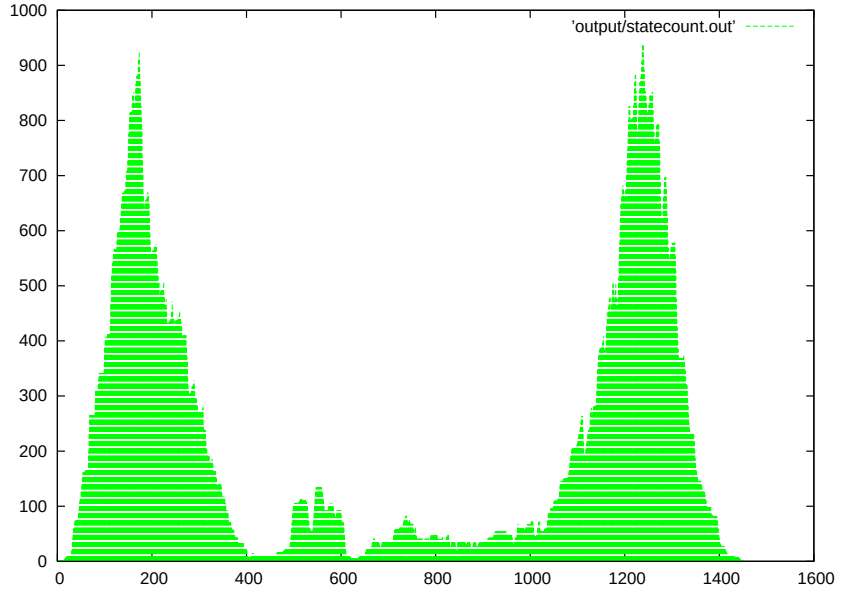

Figure 8: Evolution of the state space size for the 30-job instance.

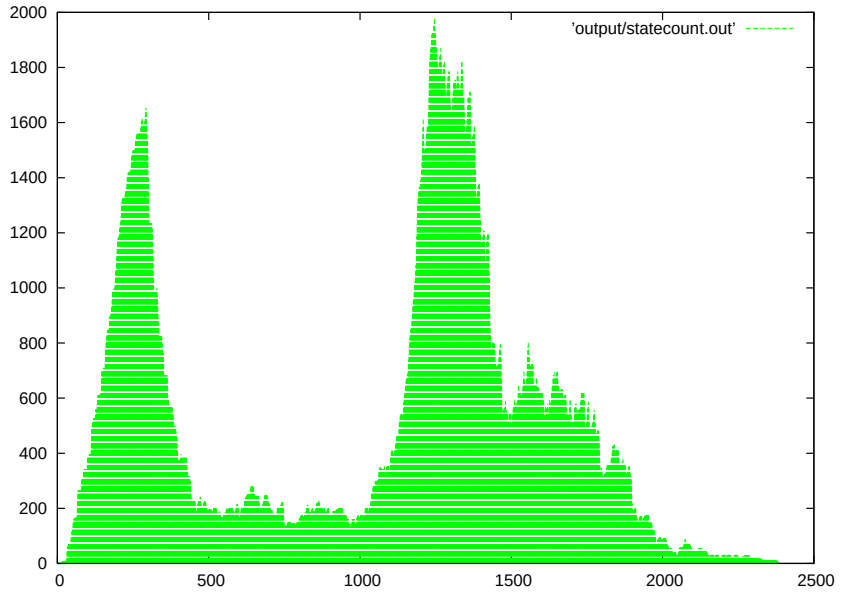

Figure 9: Evolution of the state space size for the 60-job instance.

which all of the time windows are wide, perhaps on the order of 90 minutes as in the 60-job instance. It was to accommodate wide time windows that we developed the state space reduction techniques of Section 6.

Table 3 reveals the critical importance of these techniques. For each of the three problem instances, the table shows the average time and state space size required to compute the optimal trajectories for ten different job assignments and sequencings. The assignments and sequencings were those obtained in ten iterations of a heuristic method. Without the state space reduction technique, the dynamic programming algorithm could scale up to only 30 jobs, and even then only for narrow time windows. The width of the time windows is reduced in these experiments to make the problem easier to solve, while still maintaining feasibility. The 30-job instance has a feasible solution with 35-minute time windows, but larger instances require wider time windows to achieve feasibility, and this causes the state space to explode. However, the table shows that the state space reduction technique reduces the number of states by a factor of about 20, and the computation time by a factor of ten. It is this state space reduction that makes the full 60-job problem tractable.

Table 3: Effect of state space reduction on state space size and computation time. Each instance is solved for 10 different jobs assignments and sequencings. "Before" and "after" refer to results before and after state space reduction, respectively.

Jobs	Time window (min)	Avg number of states		Peak number of states		Average time (sec)	
		Before	After	Before	After	Before	After
10	25	3224	139	9477	465	15.8	2.0
20	35	3200	144	22,204	927	82.6	8.6
30	35	3204	216	22,204	940	143.8	15.0

8 Conclusions and Future Research

We presented a specialized dynamic programming algorithm that computes optimal space-time trajectories for two interacting factory cranes. The state space is economically represented in such a way that medium-sized problems can be solved to optimality. The technique is useful both for solving a significant number of practical problems and as a benchmarking and calibration tool for heuristic methods that solve larger problems. Unlike other methods, it specifies precisely how cranes can yield to one another to minimize delay in carrying out a production schedule.

We also proved structural theorems to show that only certain types of trajectories need be considered to obtain an optimal solution. This not only accelerates solution of the problem, but it permits easier and safer operation of the cranes.

An obvious direction for future research is to attempt to generalize the structural results to three or more cranes. This would allow heuristic methods that are capable of solving large, multi-crane problems to examine fewer trajectories. Another useful research program would be a systematic empirical comparison of heuristic methods with the exact algorithm described here to determine how best to design and tune a heuristic algorithm.

References

Armstrong, R., Lei, L., and Gu, S. (1994). 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.

Baptiste, P., Legeard, B., Manier, M.-A., and Varnier, C. (1994). A scheduling problem optimisation solved with constraint logic programming. In *Second International Conference on the Practical Application of Prolog*, pages 47–66, London.

Che, A. and Chu, C. (2004). 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.

Chen, H., Chu, C., and Proth, J.-M. (1998). Cyclic scheduling of a hoist with time window constraints. *IEEE Transactions on Robotics and Automation*, **14**, 144–152.

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

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

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

Lei, L. and Wang, T. J. (1994). Determining optimal cyclic hoist schedules in a single-hoist electroplating line. *IIE Transactions*, **26**, 25–33.

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

Leung, J. and Zhang, G. (2003). 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.

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

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

Liu, J. and Jiang, Y. (2005). An efficient optimal solution to the two-hoist no-wait cyclic scheduling problem. *Operations Research*, **53**, 313–327.

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

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

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

Ng, W. C. (1996). A branch and bound algorithm for hoist scheduling of a circuit board production line. *International Journal of Flexible Manufacturing Systems*, **8**, 45–65.

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

Ng, W. C. and Leung, J. (1997). Determining the optimal move times for a given cyclic schedule of a material handling hoist. *Computers and Industrial Engineering*, **32**, 595–606.

Peterkofsky, R. I. and Daganzo, C. F. (1990). A branch and bound solution method for the crane scheduling problem. *Transportation Research Part B*, **24**, 159–172.

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

Rodošek, R. and Wallace, M. (1998). A generic model and hybrid algorithm for hoist scheduling problems. In M. Maher and J.-F. Puget, editors, *Principle and Practice of Constraint Programming (CP 1998)*, volume 1520, Pisa. Springer.

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

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

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

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

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

Zhu, Y. and Lim, A. (2006). Crane scheduling with non-crossing constraint. *Journal of the Operational Research Society*, **57**, 1464–1471.