Optimal Crane Scheduling

Ben Peterson, Samid Hoda,
John Hooker, Latife Genc Kaya
Carnegie Mellon University

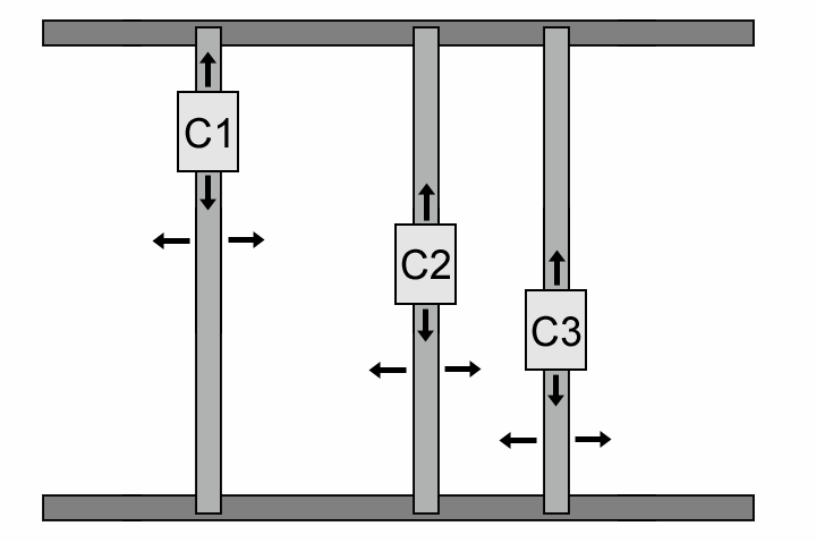
Iiro Harjunkoski

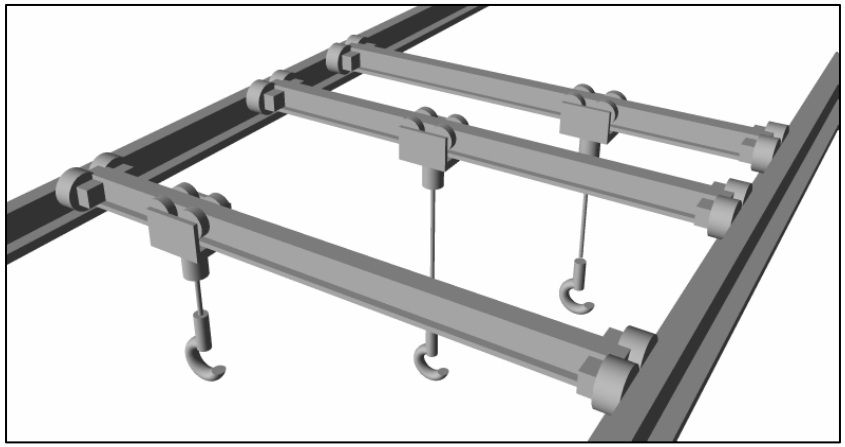
ABB Group

Enterprise-Wide Optimization Meeting
Wednesday, March 12, 2008

Problem Description

Industrial plants use track-mounted cranes to move materials and equipment.





Cranes can move: longitudinally (left and right)
latitudinally (back and forth)

Cranes must not cross paths.

Problem Description

A problem instance consists of:

- A list of jobs (~250) with 1 - 8 tasks each

- Locations and stop times for all tasks

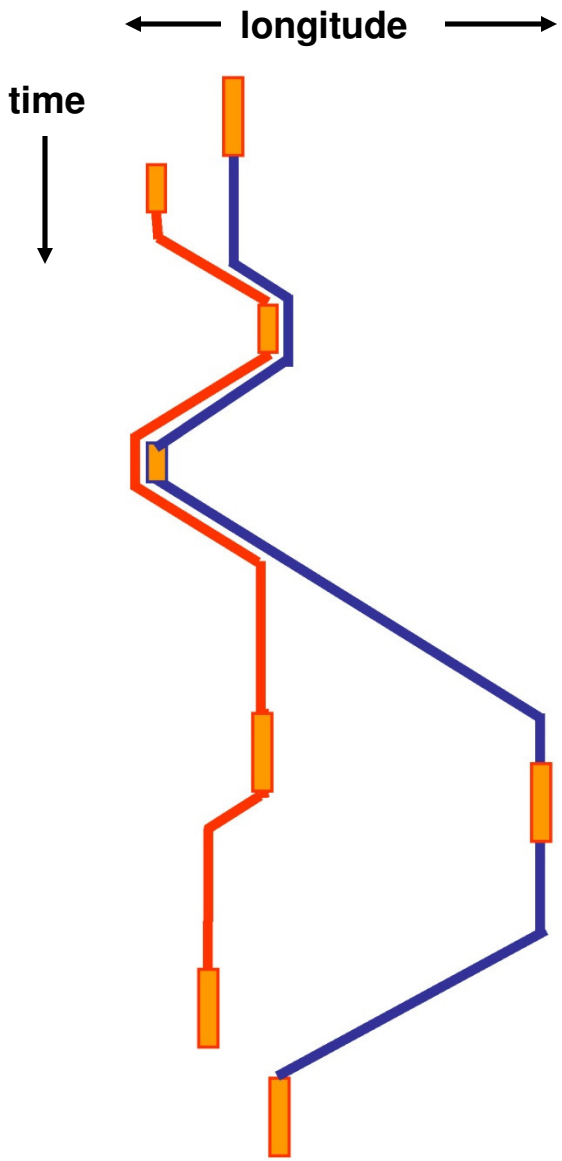
- Time windows and priority values
  for each job

- Various job constraints

Goal:

Design crane trajectories that follow production schedule as closely as possible.

Example crane trajectories



Problem Description

minimize $\sum_{j \in Jobs} priority_j (start_j + finish_j)$

subject to

- Job time windows - $(release_j \leq start_j,\ finish_j \leq deadline_j, \ldots)$

- Job assignment constraints -

- Job sequence constraints - $(finish_{j1} \leq start_{j2})$

- Crane path constraints -

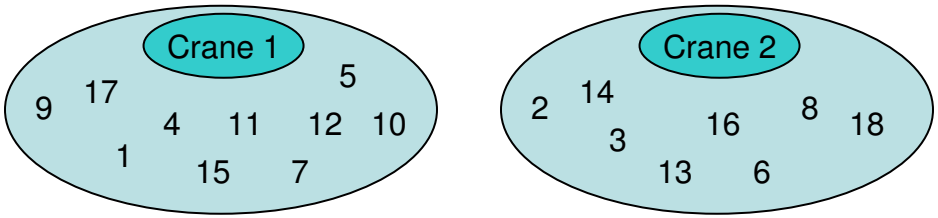
- Velocity constraints -

- Crane interference constraints -

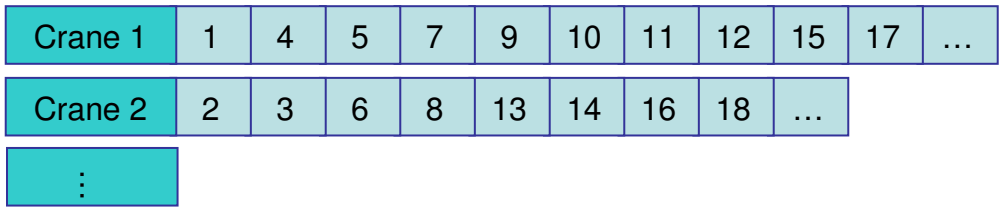
Problem Description

Three problem levels:

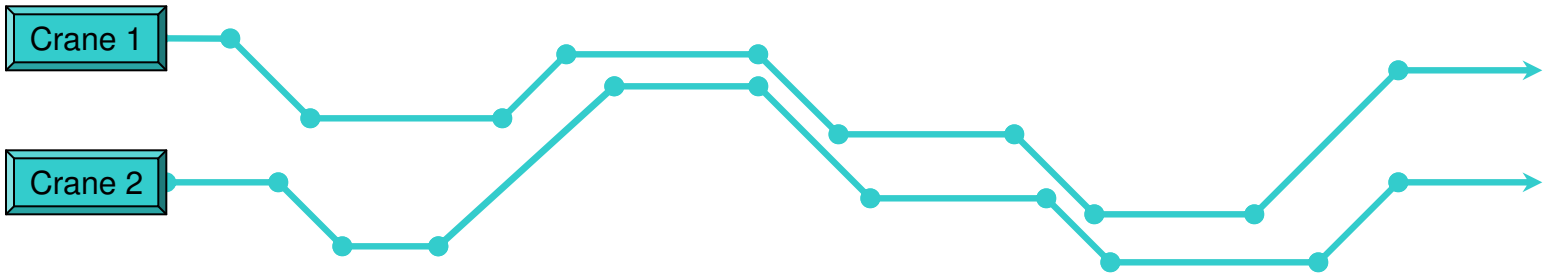
1. Assign jobs to cranes.



2. Sequnce jobs on each crane.



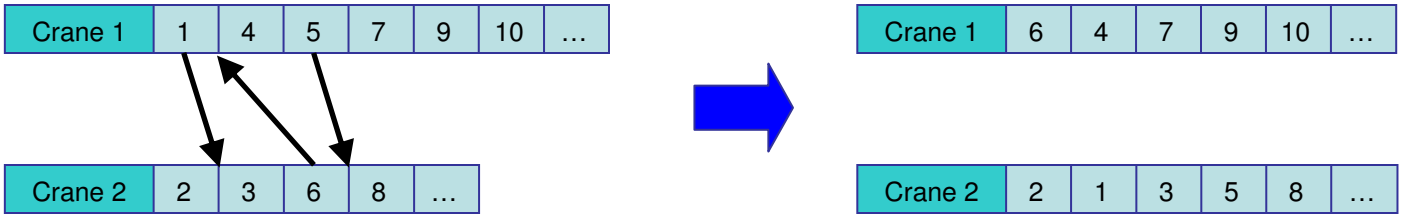
3. Plan crane trajectories.



Algorithm 1: Two-Phase LS/DP

- Phase 1:

Use a local search heuristic to assign and sequence jobs on cranes simultaneously.



Phase 2:

Use dynamic programming to optimize crane trajectories for a given assignment and sequence.

Position

Task time

Current task

$$f_{t+1}\begin{pmatrix} x_{\cdot,t+1} \\ u_{\cdot,t+1} \\ s_{\cdot,t+1} \end{pmatrix} = \min_{\begin{pmatrix} x_{\cdot,t} \\ u_{\cdot,t} \\ s_{\cdot,t} \end{pmatrix} \in S \begin{pmatrix} x_{\cdot,t+1} \\ u_{\cdot,t+1} \\ s_{\cdot,t+1} \end{pmatrix}} \left\{ f_t \begin{pmatrix} x_{\cdot,t} \\ u_{\cdot,t} \\ s_{\cdot,t} \end{pmatrix} + \sum_c g_{ct}\left(s_{ct}, s_{c,t+1}\right) \right\}$$

Transitions that satisfy constraints

Computes penalty

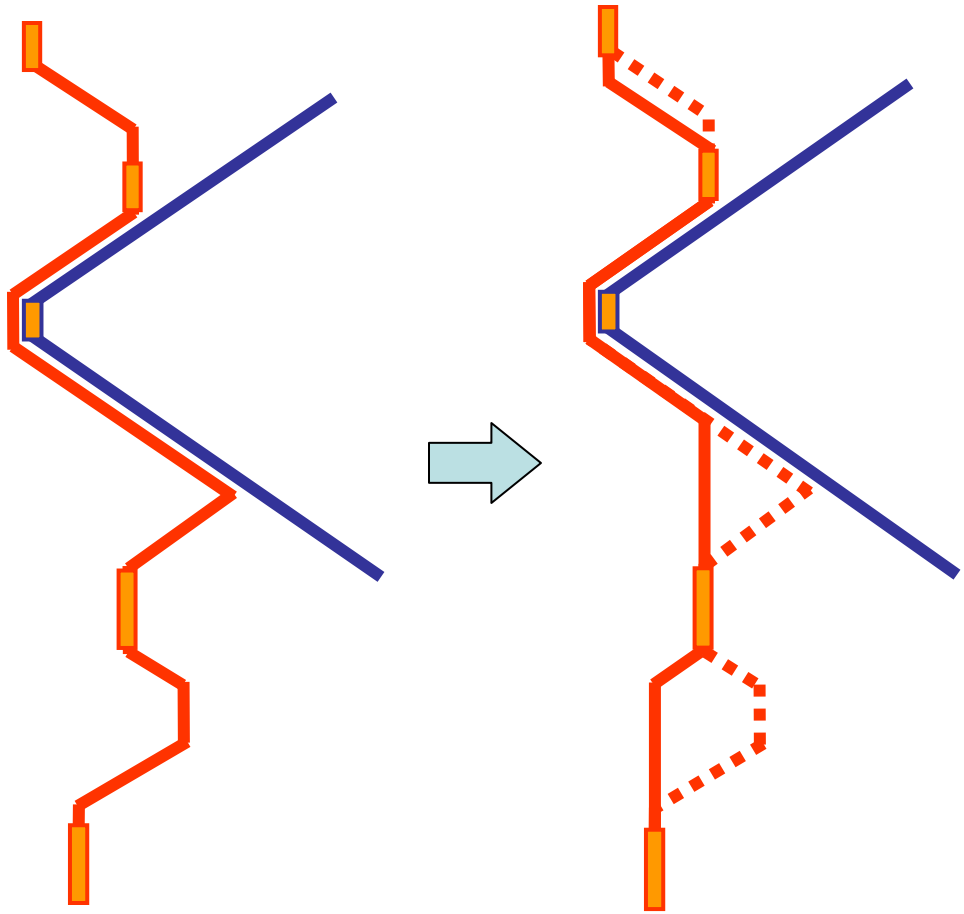
Algorithm 1: Two-Phase LS/DP

Improvement 1:

Canonical paths

- Standard movement rules shrink size of state space

- Canonical crane paths minimize movement and interference



Algorithm 1: Two-Phase LS/DP

Improvement 2:

Processing intervals

$$\begin{pmatrix} x_{1,t} & x_{2,t} \\ u_{1,t} & u_{2,t} \\ s_{1,t} & s_{2,t} \end{pmatrix} \Rightarrow \begin{pmatrix} x_{1,t} & x_{2,t} \\ [u_{1,t,MIN}, u_{1,t,MAX}] & [u_{2,t,MIN}, u_{2,t,MAX}] \\ s_{1,t} & s_{2,t} \end{pmatrix}$$

- Cranes spend most of time processing tasks

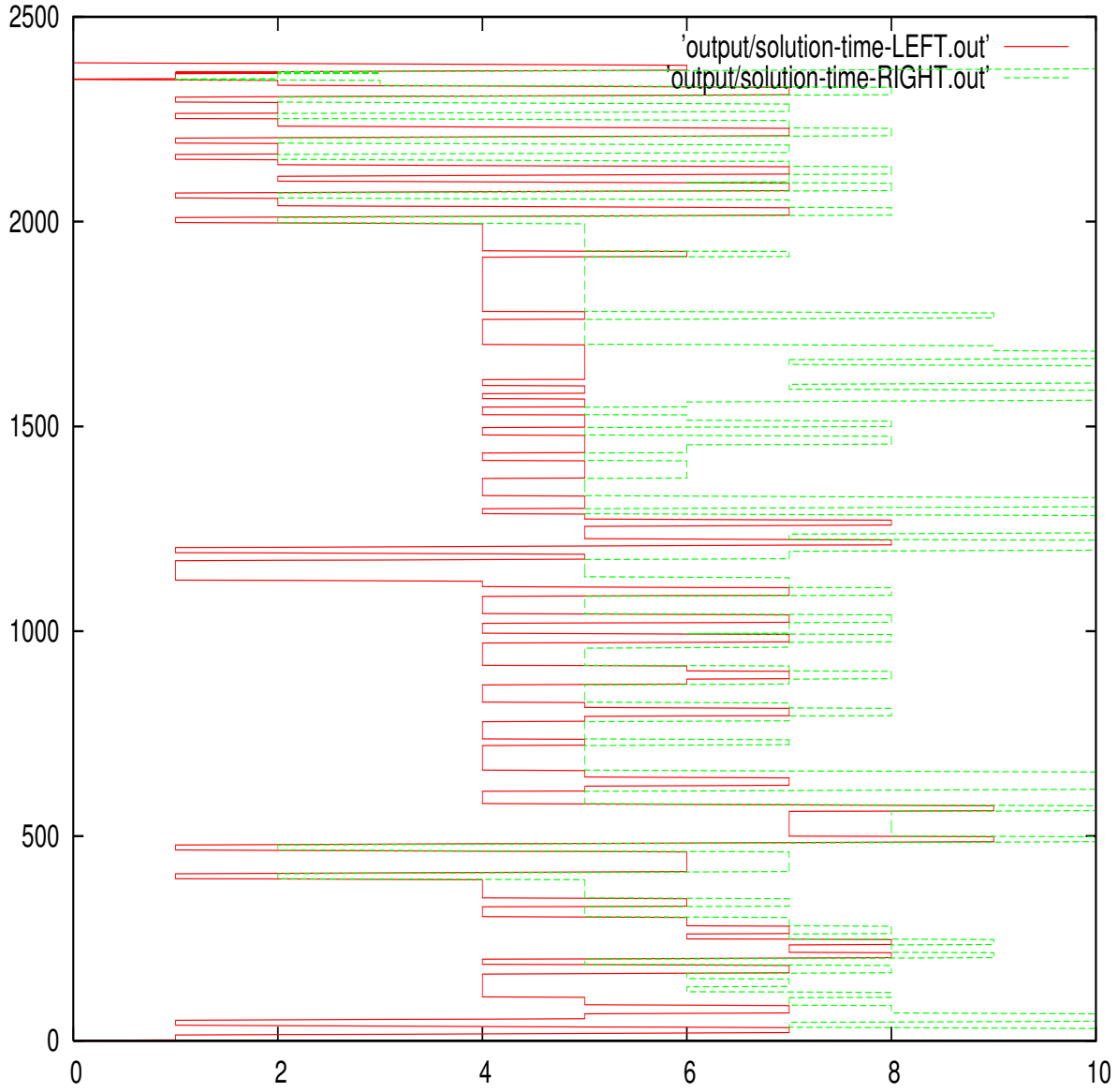
- Store intervals of possible processing times instead of storing as separate states

- Costs can be updated quickly

u_{1min} u_{2min}	$u_{1min}+1$ u_{2min}	...	...	$u_{1max}-1$ u_{2min}	u_{1max} u_{2min}
u_{1min} $u_{2min}+1$	$u_{1min}+1$ $u_{2min}+1$	...	...	$u_{1max}-1$ $u_{2min}+1$	u_{1max} $u_{2min}+1$
⋮	⋮	⋱	⋱	⋮	⋮
⋮	⋮	⋱	⋱	⋮	⋮
u_{1min} $u_{2max}-1$	$u_{1min}+1$ $u_{2max}-1$	...	...	$u_{1max}-1$ $u_{2max}-1$	u_{1max} $u_{2max}-1$
u_{1min} u_{2max}	$u_{1min}+1$ u_{2max}	...	...	$u_{1max}-1$ u_{2max}	u_{1max} u_{2max}

Algorithm 1: Computational Results

Solution to a 60-job problem



Algorithm 1: Computational Results

Effect of New Data Structure

Jobs	Time window (min)	Avg # states		Peak # states		Time (s)	
		Old	New	Old	New	Old	New
10	25	3,224	139	9,477	465	158	20
20	35	3,200	144	22,204	927	826	86
30	35	3,204	216	22,204	940	1,438	150

Computation Time for one DP

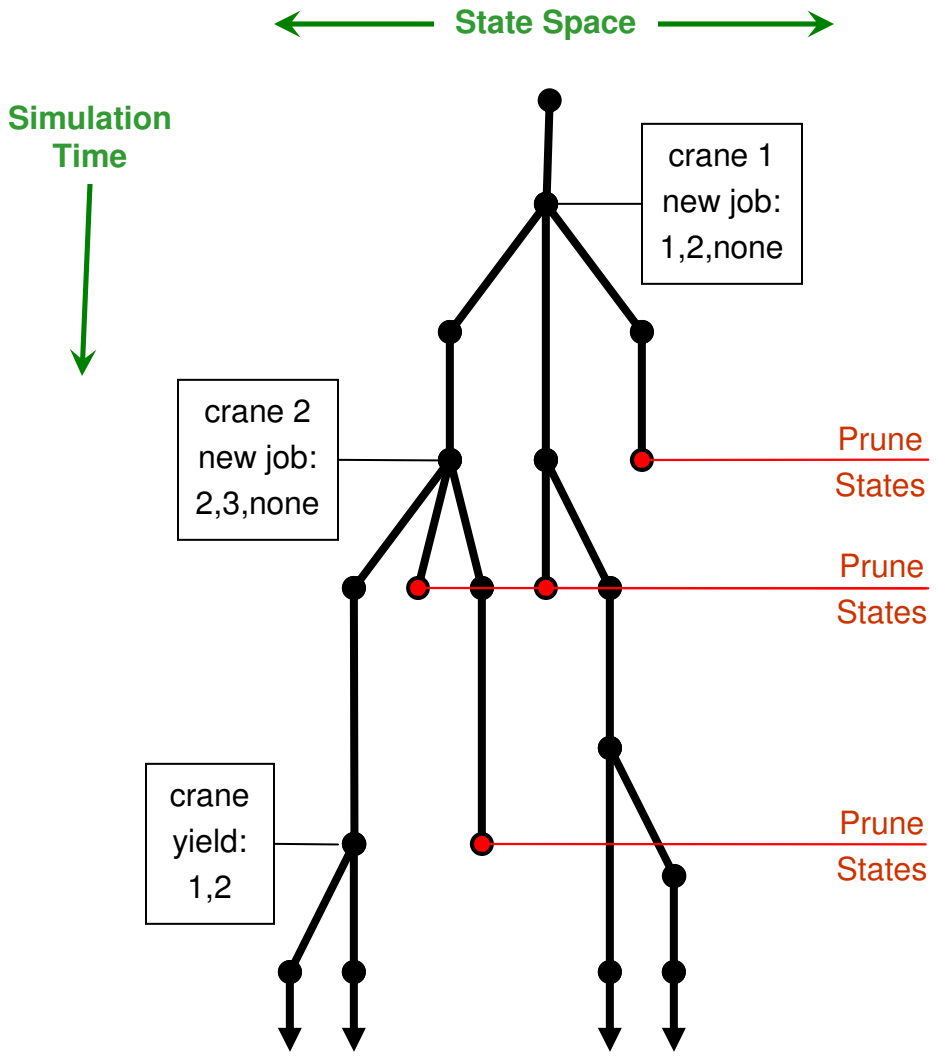
Jobs	Time window (min)	Computation Time per DP (s)
10	40	6.8
20	40	7.5
30	40	15.8
40	40	16.7
50	40	18.8
60	55	48.1

Algorithm 2: One-Phase DP w/ State Cuts

Two main functions:

#1: Crane Simulation:
Simulates crane activity
Splits states at decision points

#2: State Pruning:
Eliminates inferior states

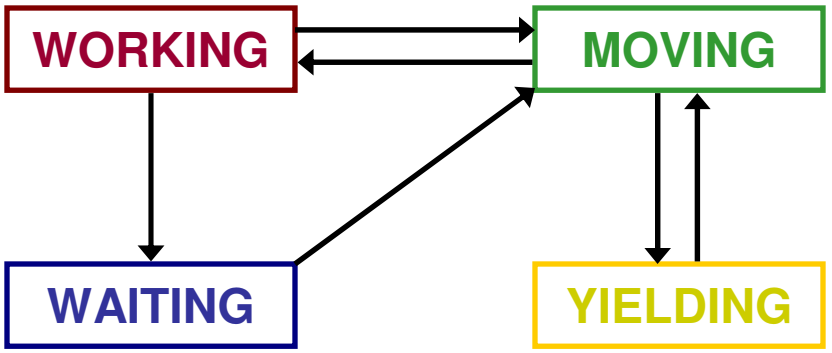


Algorithm 2: One-Phase DP w/ State Cuts

#1: Crane Simulation:

4 crane actions:

- ***working*** on task
- ***moving*** toward task
- ***yielding*** to other crane
- ***waiting*** for new job



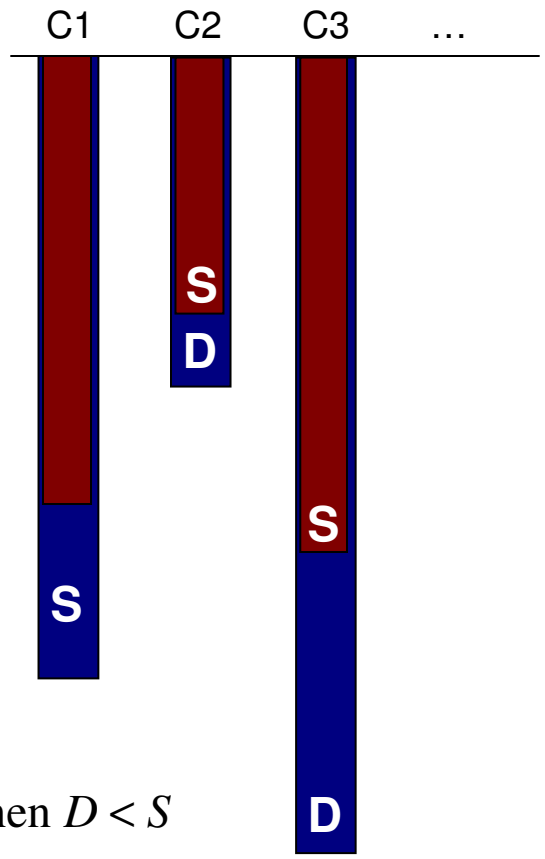
Simulation operates on
next action to complete.

Cranes will ***work*** and ***move*** until:

- Time to pick a job; state splits for each job choice
- Cranes interfere; state splits for each yielding choice

Algorithm 2: One-Phase DP w/ State Cuts

#2: State Pruning:



Exact pruning by state domination

State **D** dominates state **S** if

1) $D[objective] \leq S[objective]$

2) $D[\,jobs\ complete] \supseteq S[\,jobs\ complete]$

3) For all cranes c,

$D_c[\,job] = S_c[\,job]$ OR $D_c[action] = waiting$

AND

$D_c[\,job\ \ progress] \geq S_c[\,job\ \ progress]$

4) If $D[all$ vars $except\ path] = S[all$ vars $except\ path]$ then $D < S$

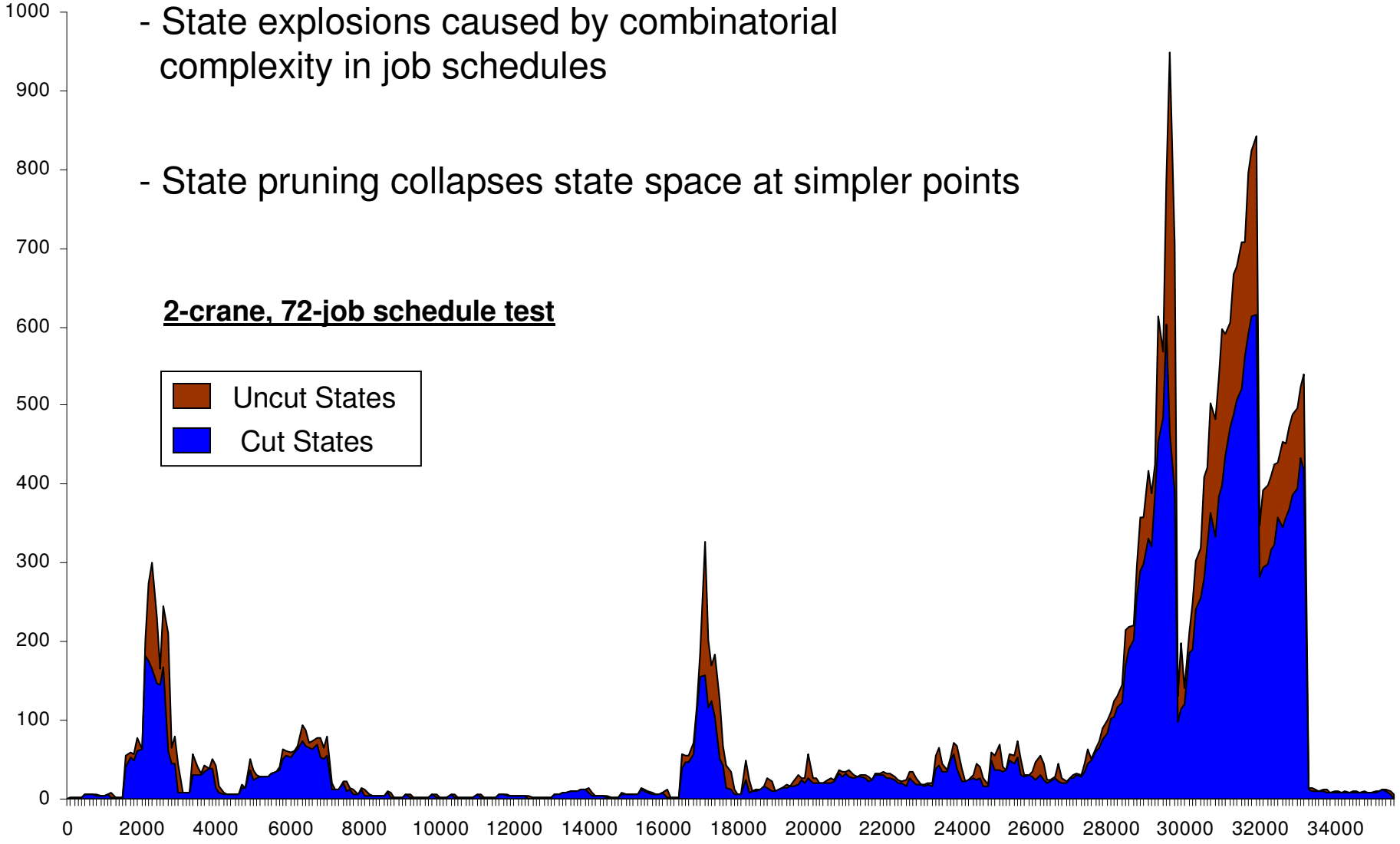
Inexact pruning by state buffer limit

- Drop highest-objective states.

Algorithm 2: Computational Results

- State explosions caused by combinatorial complexity in job schedules

- State pruning collapses state space at simpler points



Algorithm 2: Computational Results

- Frequent state space collapse makes runtime essentially linear with respect to the number of jobs

# Jobs	10	20	30	40	50	60
Runtime (sec)	0.422	0.687	0.734	1.688	1.750	1.890

- Most 2-crane problems solve in seconds

Algorithm 2: Richer Set of Constraints

- Algorithm 2 is able to deal with many more scheduling constraints:

  Task precedence

  Job precedence

  Crane specialization

  etc.

- Implemented using a system of binary flags

  Jobs and tasks can be restricted to start only when certain flags have been set

- Efficiently handles groups of identical jobs

Conclusions

- New algorithm meets runtime goals.

- Handles additional cranes if necessary.

- Is still sensitive to schedule complexity.

Future Work

- Incorporation of production schedule info

- Adaptation to other crane problems, e.g. cranes in ports.

- Possible application to other scheduling problems.