Job Sequencing Bounds from Decision Diagrams

John Hooker

CP 2017

Melbourne, Australia

Objective

- Find a general method to **relax dynamic programming models.**
  - **Job sequencing** problems in particular.
  - With **state dependent** processing times.

Objective

- Find a general method to **relax dynamic programming models.**
  - **Job sequencing** problems in particular.
  - With **state dependent** processing times.
- **Why?** To obtain **bounds** on the optimal value.
  - Useful for **heuristics** and **exact methods**

Objective

- Find a general method to **relax dynamic programming models.**
  - **Job sequencing** problems in particular.
  - With **state dependent** processing times.
- **Why?** To obtain **bounds** on the optimal value.
  - Useful for **heuristics** and **exact methods**
- **How?** Using relaxed **decision diagrams**
  - Constructed with **node merger**

Why Dynamic Programming?

- Highly **flexible** modeling.
  - Costs and constraints need not be convex, linear, or even in closed form.
  - Exploits recursive structure.

Why Dynamic Programming?

- Highly **flexible** modeling.
    - Costs and constraints need not be convex, linear, or even in closed form.
    - Exploits recursive structure.
- **Good relaxations** often **unavailable**.
    - Due to the very generality of the model.

Why Dynamic Programming?

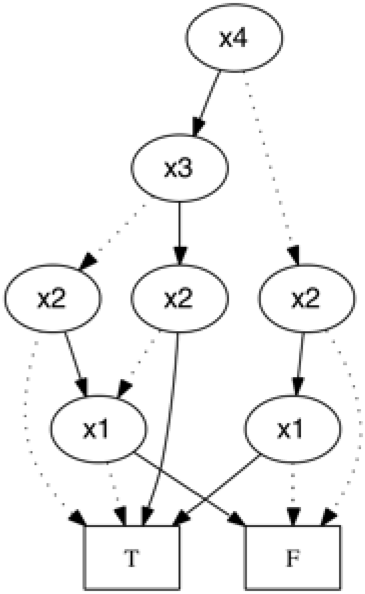
- Highly **flexible** modeling.
  - Costs and constraints need not be convex, linear, or even in closed form.
  - Exploits recursive structure.
- **Good relaxations** often **unavailable**.
  - Due to the very generality of the model.
- Focus on discrete, **deterministic** DP.
  - Extension to **stochastic** DP possible.

Why Decision Diagrams?

- A potentially useful **discrete relaxation**.
    - Obtained by node **splitting** or node **merger**.
    - We focus on node **merger**.
- Can provide relaxations where **none previously existed**.
    - As in job sequencing problems with **state-dependent processing times**..

Decision Diagrams

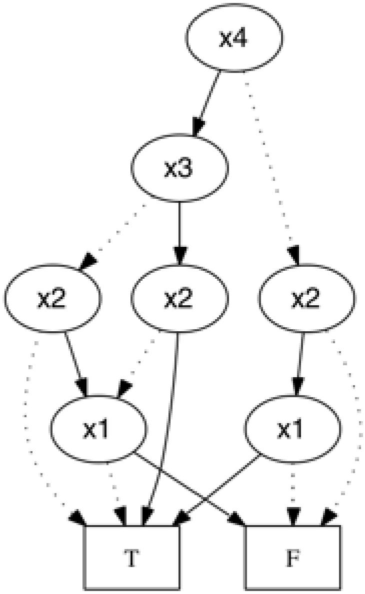
- Graphical encoding of a boolean function
  - Historically used for circuit design & verification
  - **Binary diagrams** easily extended to **multivalued diagrams**.
  - Unique **reduced** diagram for a give variable ordering.



Lee (1959), Bryant (1986)

Decision Diagrams

- Adapt to optimization and constraint programming
  - **Paths** from top to bottom represent **feasible solutions**
  - Path **lengths** represent **costs**.
  - **Shortest** path is **optimal** solution.



Hadžić and JH (2006, 2007)

Job Sequencing

- Problem: sequence jobs with given processing times
  - Minimize **tardiness** subject to **time windows**
  - Processing time may depend on previous jobs
    - For example, some necessary components may have been made for previous jobs

j	r_j	p_j	d_j
1	0	3^*	4
2	1	2	3
3	1	2	5

*2 when job 2 has previously been processed.

Job Sequencing

- Problem: sequence jobs with given processing times
  - Minimize **tardiness** subject to **time windows**
  - Processing time may depend on previous jobs
    - For example, some necessary components may have been made for previous jobs

Release time

j	r_j	p_j	d_j
1	0	3^*	4
2	1	2	3
3	1	2	5

*2 when job 2 has previously been processed.

Job Sequencing

- Problem: sequence jobs with given processing times
  - Minimize **tardiness** subject to **time windows**
  - Processing time may depend on previous jobs
    - For example, some necessary components may have been made for previous jobs

Release time

j	r_j	p_j	d_j
1	0	3^*	4
2	1	2	3
3	1	2	5

Processing time

*2 when job 2 has previously been processed.

Job Sequencing

- Problem: sequence jobs with given processing times
  - Minimize **tardiness** subject to **time windows**
  - Processing time may depend on previous jobs
    - For example, some necessary components may have been made for previous jobs

j	r_j	p_j	d_j
1	0	3^*	4
2	1	2	3
3	1	2	5

Release time

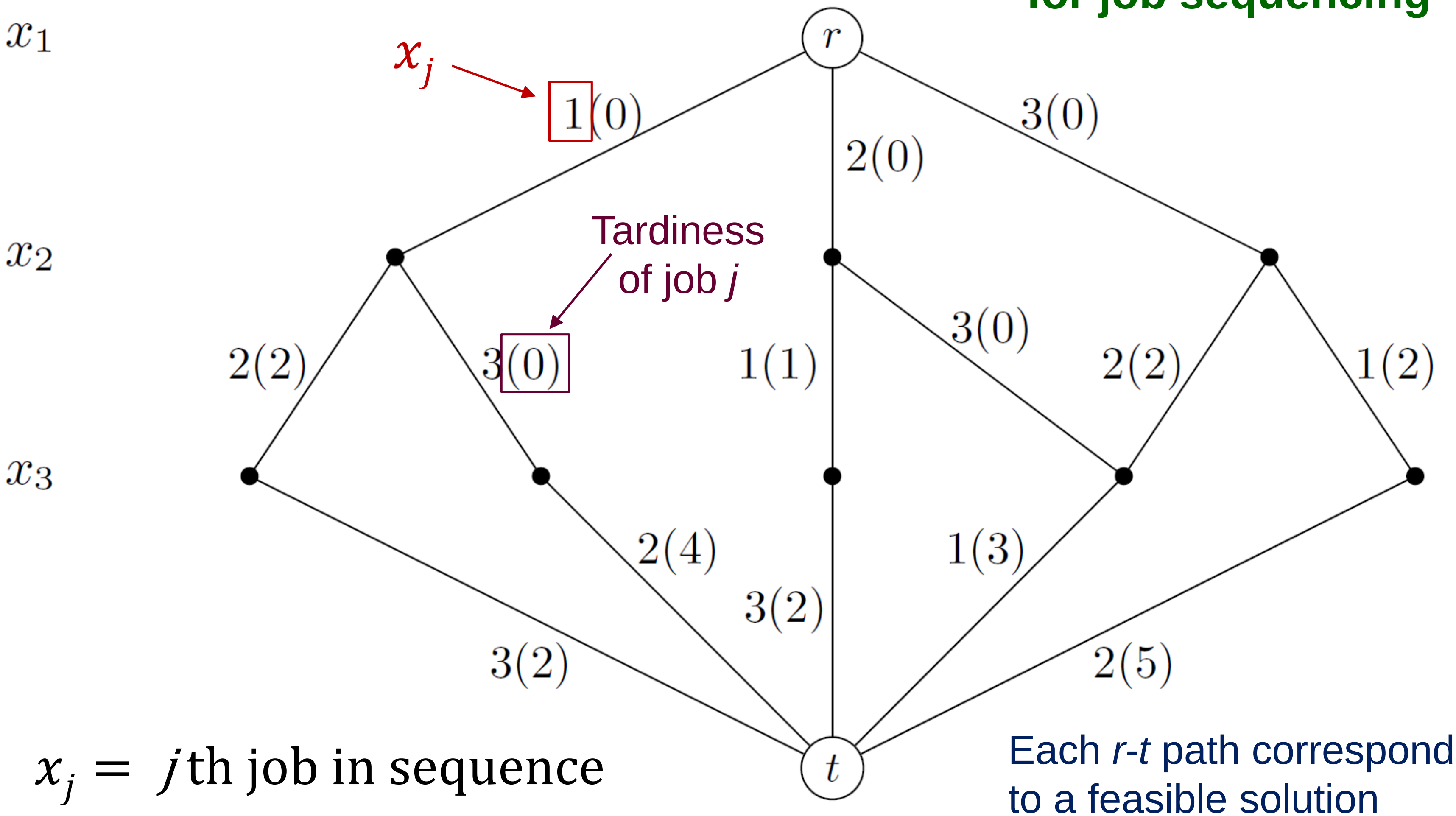
Processing time

Due date

*2 when job 2 has previously been processed.

Job Sequencing

Decision diagram for job sequencing



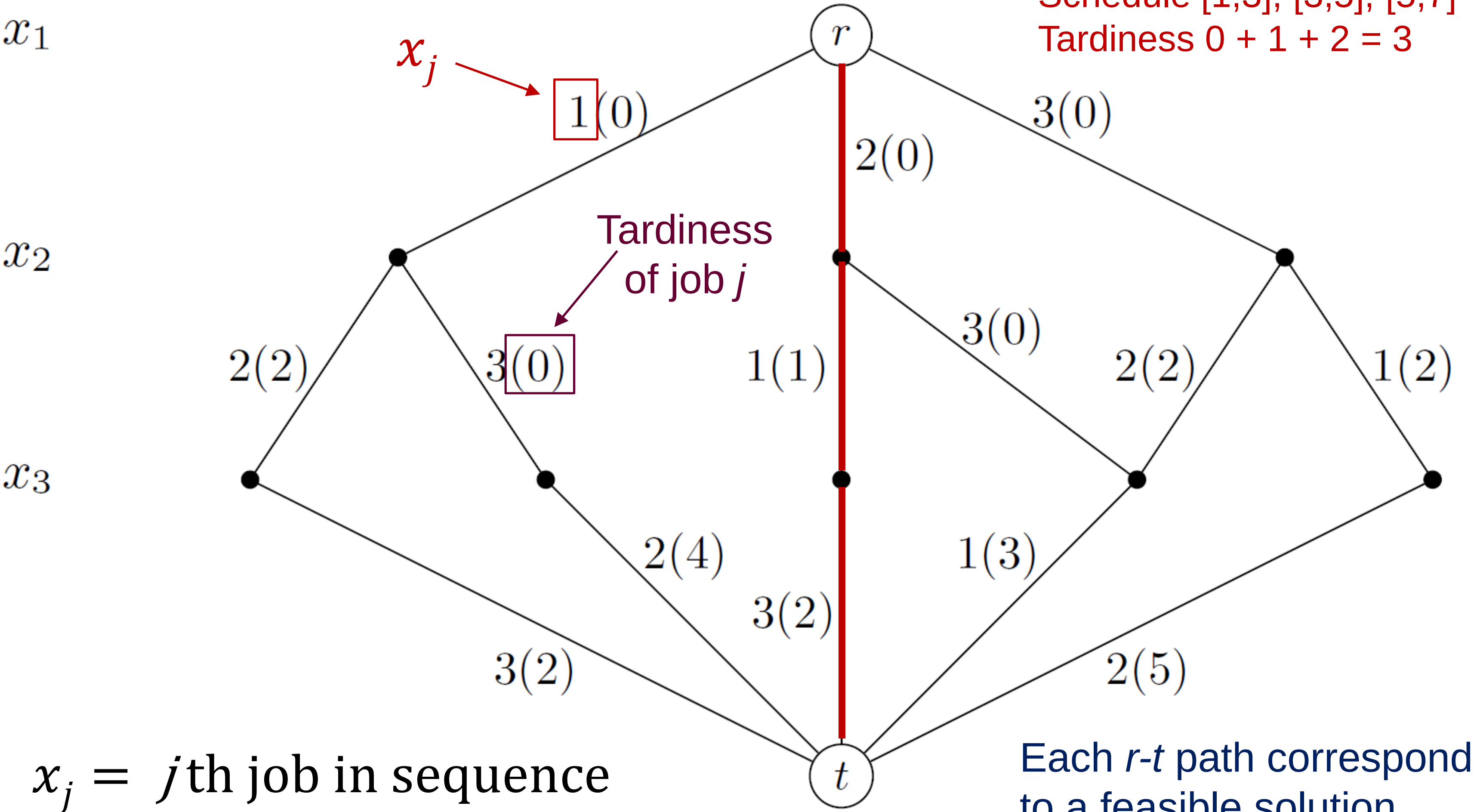
Tardiness of job j

$x_j =$ jth job in sequence

Each r-t path corresponds to a feasible solution

Job Sequencing

An optimal solution:
Sequence 2-1-3
Schedule [1,3], [3,5], [5,7]
Tardiness 0 + 1 + 2 = 3



x_j

Tardiness of job *j*

$x_j =$ *j*th job in sequence

Each *r*-*t* path corresponds to a feasible solution

Building a Decision Diagram

- Our approach:
  - Associate dynamic programming **states** with nodes..
  - …as in a state transition graph.

Dynamic Programming Model

- Our approach:
  - Associate dynamic programming **states** with nodes..
  - …as in a state transition graph.

General recursive model

$$h_j(\boxed{S}) = \min_{x_j \in X_j(S)} \Big\{ c_j(S, x_j) + h_{j+1}\Big((\phi_j(S, x_j))\Big) \Big\}$$

State in stage *j*

Dynamic Programming Model

- Our approach:
  - Associate dynamic programming **states** with nodes..
  - …as in a state transition graph.

General recursive model

$$h_j(\boxed{S}) = \min_{x_j \in \boxed{X_j(S)}} \Big\{ c_j(S, x_j) + h_{j+1}\Big((\phi_j(S, x_j)) \Big) \Big\}$$

State in stage *j*

Set of possible controls

Dynamic Programming Model

- Our approach:
  - Associate dynamic programming **states** with nodes..
  - …as in a state transition graph.

General recursive model

$$h_j(\boxed{S}) = \min_{x_j \in \boxed{X_j(S)}} \left\{ \boxed{c_j(S, x_j)} + h_{j+1}\Big((\phi_j(S, x_j))\Big) \right\}$$

State in stage *j*

Set of possible controls

Immediate cost

Dynamic Programming Model

- Our approach:
  - Associate dynamic programming **states** with nodes..
  - …as in a state transition graph.

General recursive model

State transition function

$$h_j(\boxed{S}) = \min_{x_j \in \boxed{X_j(S)}} \left\{ \boxed{c_j(S, x_j)} + h_{j+1}\Big(\big(\boxed{\phi_j}(S, x_j)\big)\Big) \right\}$$

State in stage *j*

Set of possible controls

Immediate cost

Dynamic Programming Model

- Our approach:
  - Associate dynamic programming **states** with nodes..
  - …as in a state transition graph.

General recursive model

State transition function

$$h_j(\boxed{S}) = \min_{x_j \in \boxed{X_j(S)}} \left\{ \boxed{c_j(S, x_j)} + \boxed{h_{j+1}((\boxed{\phi_j}(S, x_j))} \right\}$$

State in stage *j*

Set of possible controls

Immediate cost

Cost to go

DP Model for Job Sequencing

Set of jobs scheduled so far

Initial state = $(\emptyset, 0)$

State: $S = (\boxed{V,}\boxed{f})$

Finish time of last job scheduled

$$h_j(\boxed{S}) = \min_{x_j \in \boxed{X_j(S)}} \left\{ \boxed{c_j(S, x_j)} + \boxed{h_{j+1}\big((\phi_j(S, x_j))\big) } \right\}$$

State in stage *j*

Set of possible controls

Immediate cost

Cost to go

DP Model for Job Sequencing

Set of jobs scheduled so far

State: $S = (\boxed{V,}\boxed{f})$

Finish time of last job scheduled

Initial state = $(\emptyset, 0)$

Controls: $x_j(V, f) = \{1, \ldots, n\} \setminus V$

$$h_j(\boxed{S}) = \min_{x_j \in \boxed{X_j(S)}} \left\{ \boxed{c_j(S, x_j)} + \boxed{h_{j+1}((\phi_j(S, x_j)) } \right\}$$

State in stage *j*

Set of possible controls

Immediate cost

Cost to go

DP Model for Job Sequencing

Set of jobs scheduled so far

Initial state = $(\emptyset, 0)$

State: $S = (\boxed{V,}\boxed{f})$

Finish time of last job scheduled

Controls: $x_j(V, f) = \{1, \ldots, n\} \setminus V$

Immediate cost: $c_j((V, f), x_j) = \left(\max\{r_{x_j}, f\} + \boxed{p_{x_j}(V)} - d_{x_j}\right)^+$

State-dependent processing time

$$h_j(\boxed{S}) = \min_{x_j \in \boxed{X_j(S)}} \left\{ \boxed{c_j(S, x_j)} + \boxed{h_{j+1}((\phi_j(S, x_j)) \}} \right.$$

State in stage *j*

Set of possible controls

Immediate cost

Cost to go

DP Model for Job Sequencing

Set of jobs scheduled so far

Initial state = $(\emptyset, 0)$

State: $S = (\boxed{V,}\boxed{f})$

Finish time of last job scheduled

Controls: $x_j(V, f) = \{1, \ldots, n\} \setminus V$

Immediate cost: $c_j\big((V, f), x_j\big) = \big(\max\{r_{x_j}, f\} + p_{x_j}(V) - d_{x_j}\big)^+$

Transition: $\phi_j\big((V, f), x_j\big) = \big(V \cup \{x_j\}, \max\{r_{x_j}, f\} + p_{x_j}(V)\big)$

$$h_j(\boxed{S}) = \min_{x_j \in \boxed{X_j(S)}} \Big\{\boxed{c_j(S, x_j)} + \boxed{h_{j+1}\big((\phi_j(S, x_j)\big)\Big\}}$$

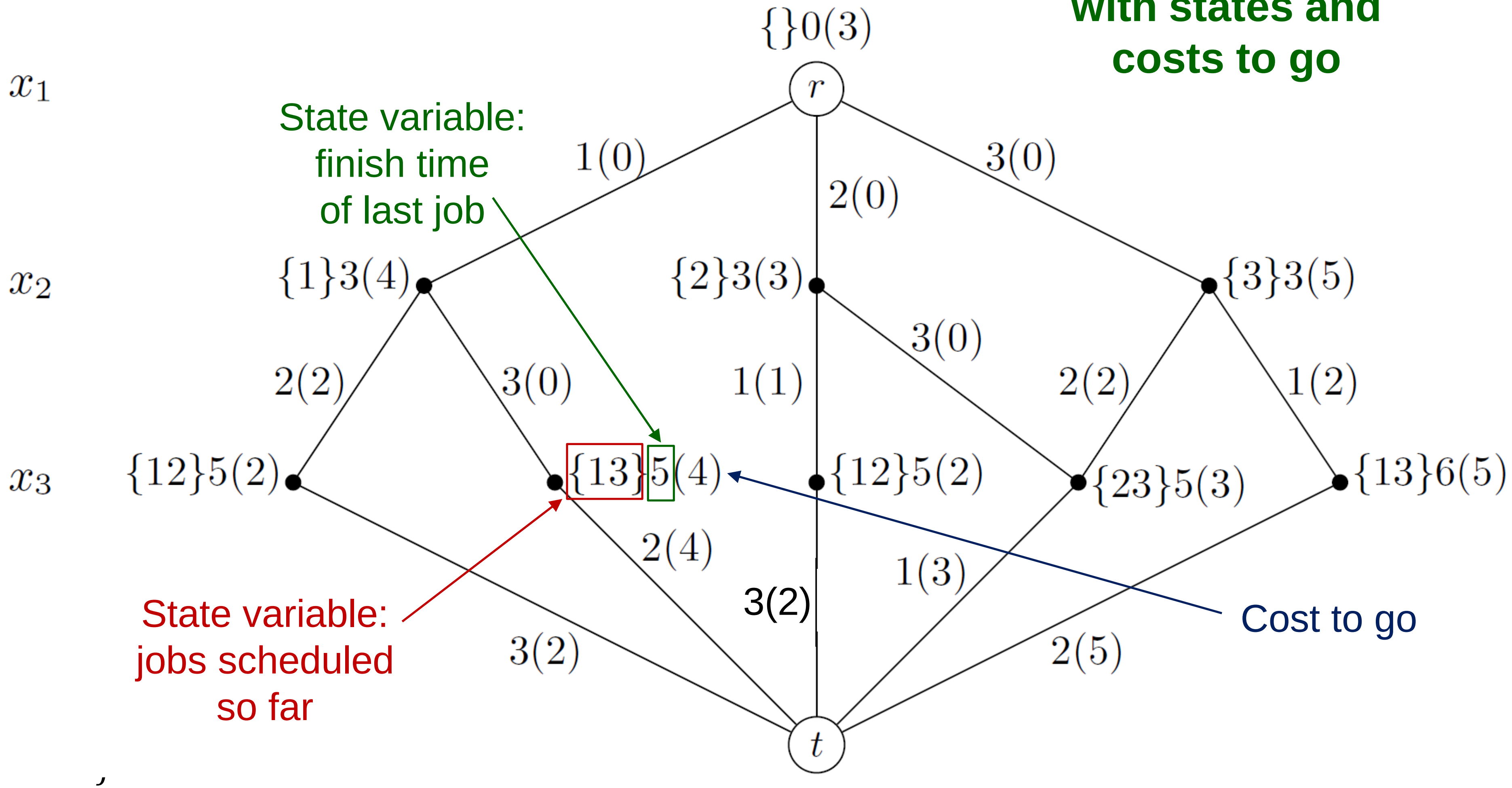
State in stage *j*

Set of possible controls

Immediate cost

Cost to go

Job Sequencing Diagram



Decision diagram with states and costs to go

State variable: finish time of last job

State variable: jobs scheduled so far

Cost to go

Relaxed Decision Diagram

- Definition
  - Every *r*-*t* path of the original diagram appears in the relaxed diagram with equal or smaller cost.
  - So a relaxed diagram **may represent some infeasible solutions**.
- Motivation
  - **Shortest path** in the relaxed diagram provides a **lower bound** on the optimal value.

Andersen, Hadžić, JH, Tiedemanmn (2007)

Building a Relaxed Diagram

- Node splitting
  - Start with a diagram that represents **all** solutions (feasible and infeasible) and **refine** it.

Andersen, Hadžić, JH, Tiedemanmn (2007)

Ciré and van Hoeve (2013)

Building a Relaxed Diagram

- Node splitting
  - Start with a diagram that represents **all** solutions (feasible and infeasible) and **refine** it.
- Node merger – examined here
  - **Merge** some nodes in the **exact** diagram.
  - …to make the diagram **smaller** while excluding no feasible solutions and introducing some infeasible low-cost solutions.

Andersen, Hadžić, JH, Tiedemanmn (2007)

Ciré and van Hoeve (2013)

Bergman, Ciré, van Hoeve, JH (2013)

Node Merger

- Don't begin with exact diagram
  - It is too large
- Merge nodes as the diagram is constructed
  - In particular, nodes that are not likely to lie on an optimal path.
  - Combine states of the merged nodes in a way that yields a valid relaxation.
  - This often requires **additional state variables**.
  - As in the job sequencing case.

Bergman, Ciré, van Hoeve, JH (2013, 2016)

Relaxed DP Model

Set of jobs scheduled in **all** feasible solutions so far

$$S = (\boxed{V,}\boxed{U,}\boxed{f)}$$

Initial state = $(\emptyset, \emptyset, 0)$

New state variable: set of jobs scheduled in **some** feasible solution so far

Earliest possible finish time of immediately previous job

$$h_j(S) = \min_{x_j \in X_j(S)} \left\{ c_j(S, x_j) + h_{j+1}\Big((\phi_j(S, x_j))\Big) \right\}$$

Relaxed DP Model

Set of jobs scheduled in **all** feasible solutions so far

$$S = (\boxed{V},\boxed{U},\boxed{f})$$

Initial state = $(\emptyset, \emptyset, 0)$

New state variable: set of jobs scheduled in **some** feasible solution so far

Earliest possible finish time of immediately previous job

Transition:

$$\phi_j\big((V,U,f),x_j\big) = \big(V \cup \{x_j\}, U \cup \{x_j\}, \max\{r_{x_j}, f\} + \boxed{p_{x_j}(U)}\big)$$

Processing time depends on *U*, not *V*
(state variable *V* can be dropped if desired)

$$h_j(S) = \min_{x_j \in X_j(S)} \Big\{ c_j(S, x_j) + h_{j+1}\big((\phi_j(S, x_j))\big) \Big\}$$

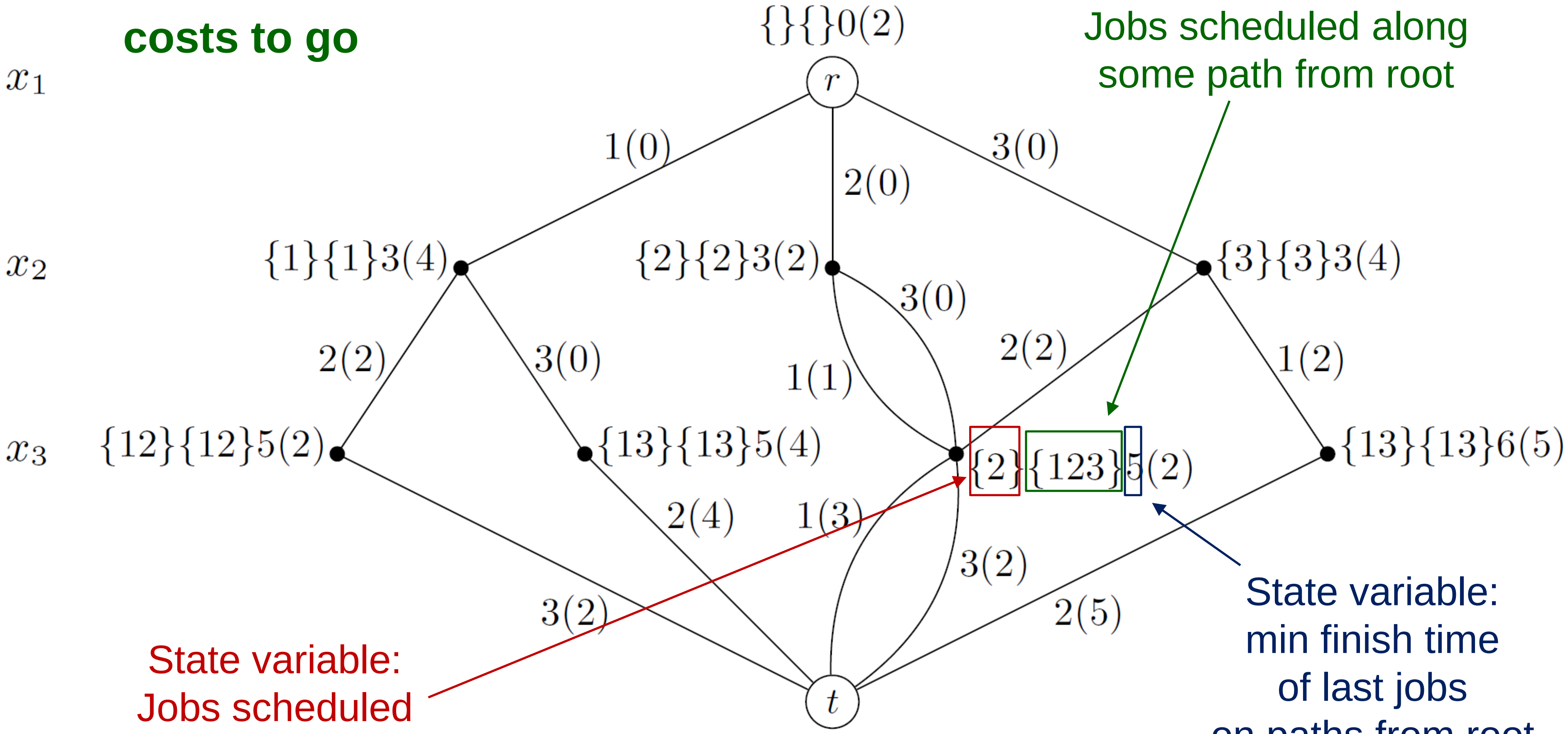
Node Merger in Relaxation

- Merge nodes as the diagram is constructed
  - States *S*, *T* merge to form state $S \oplus T$
- Merger operation must yield valid relaxation
  - In state-dependent job sequencing,

$$(V, U, f) \oplus (V', U', f') = \big(V \cap V', U \cup U', \min\{f, f'\}\big)$$

Job Sequencing Relaxed Diagram

Relaxed decision diagram with states and costs to go



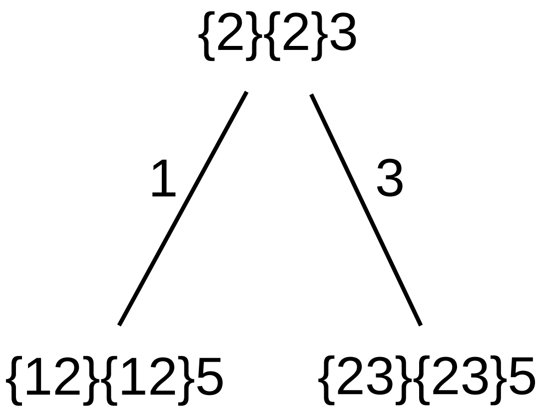
State variable:
Jobs scheduled along
some path from root

State variable:
min finish time
of last jobs
on paths from root

State variable:
Jobs scheduled
along all paths from root

Job Sequencing Node Merger

Without merger



{2}{2}3

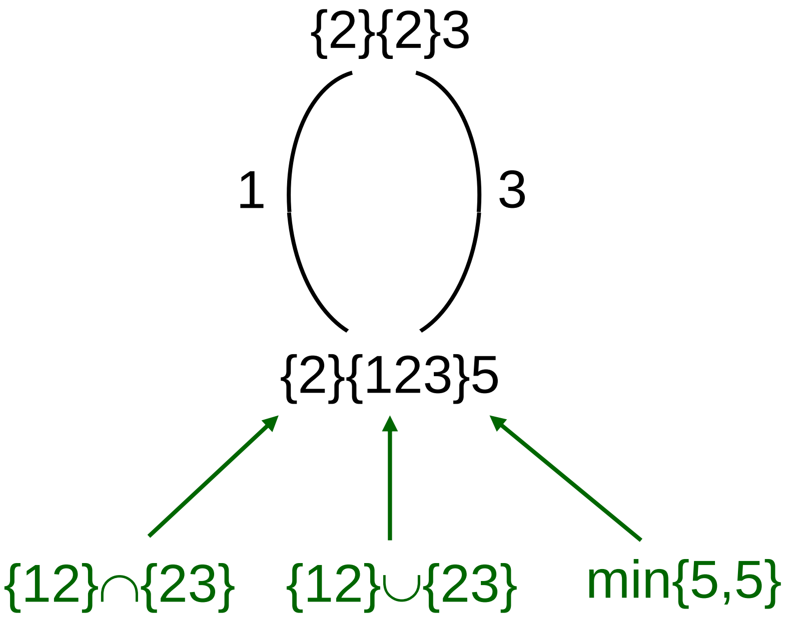
1

3

{12}{12}5

{23}{23}5

With merger



{2}{2}3

1

3

{2}{123}5

{12}∩{23}

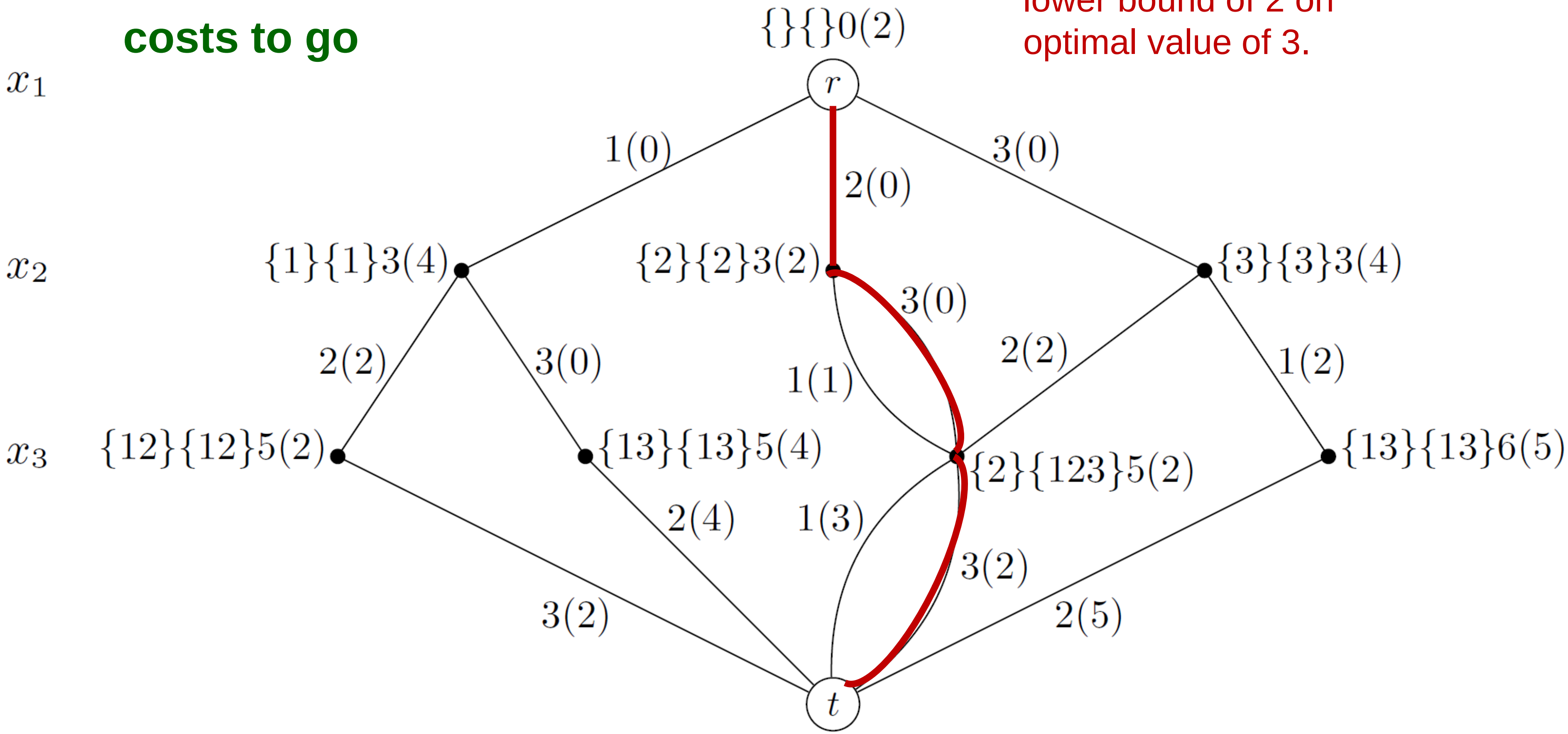
{12}∪{23}

min{5,5}

Job Sequencing Relaxed Diagram

Relaxed decision diagram with states and costs to go

Shortest path yields a lower bound of 2 on optimal value of 3.



Conditions for a Valid Relaxation

- There are two **jointly sufficient conditions** for obtaining a relaxed diagram from node merger.
    - A condition on the **transition function**
    - And a condition on the **merger operation** $S \oplus T$

Conditions for a Valid Relaxation

- There are two **jointly sufficient conditions** for obtaining a relaxed diagram from node merger.
  - A condition on the **transition function**
  - And a condition on the **merger operation** $S \oplus T$
- First we need a definition: state S' **relaxes** state S in the same stage if
  - Every control feasible in S is feasible in S'

$$X_j(S) \subseteq X_j(S')$$

  - The immediate cost of a control feasible in S is no greater in S'.

$$c_j(S, x_j) \geq c_j(S', x_j), \quad \text{all } x_j \in X_j(S)$$

Conditions for a Valid Relaxation

Theorem. The merger of states *S* and *T* in layer *j* of diagram *D* yields a relaxation of *D* if:

- S′ relaxes S implies that $\phi_j(S', x_j)$ relaxes $\phi_j(S, x_j)$ for any control x_j feasible in *S*.

- $S \oplus T$ relaxes both *S* and *T*.

Proof by induction.

This generalizes to stochastic decision diagrams, where the conditions are much more complicated.

Conditions for a Valid Relaxation

Check whether node merger for **job sequencing** satisfies the conditions.

- S′ relaxes S implies that $\phi_j(S', x_j)$ relaxes $\phi_j(S, x_j)$ for any control x_j feasible in *S*.

- $S \oplus T$ relaxes both *S* and *T*.

Conditions for a Valid Relaxation

Check whether node merger for **job sequencing** satisfies the conditions.

- S′ relaxes S implies that $\phi_j(S', x_j)$ relaxes $\phi_j(S, x_j)$ for any control x_j feasible in S.

(V', U', f') relaxes (V, U, f) when $\begin{array}{l} V' \subseteq V \\ U' \supseteq U \\ f' \leq f \end{array}$ Now…

$\phi_j\big((V', U', f'), x_j\big)$ relaxes $\phi_j\big((V, U, f), x_j\big)$ because

$$\begin{array}{l} V' \cup \{x_j\} \subseteq V \cup \{x_j\} \\ U' \cup \{x_j\} \supseteq U \cup \{x_j\} \\ \min\{r_{x_j}, f'\} + p_{x_j}(U') \leq \min\{r_{x_j}, f\} + p_{x_j}(U) \end{array}$$

Conditions for a Valid Relaxation

Check whether node merger for **job sequencing** satisfies the conditions.

- $S \oplus T$ relaxes both *S* and *T*.

$$(V, U, f) \oplus (V', U', f) \text{ relaxes } (V, U, f)$$

because

$$\begin{aligned} & V \cap V' \subseteq V \\ & U \cup U' \supseteq U \\ & \min\{f, f'\} \leq f \end{aligned}$$

…and similarly for (V', U', f)

Merger Heuristics

- Goal:
  - Merge nodes that are **not likely to lie on short paths**.
  - This reduces the likelihood of creating a superoptimal path, which would weaken the bound.
  - Keep merging nodes until desired width is obtained.

Merger Heuristics

- Goal:
  - Merge nodes that are **not likely to lie on short paths**.
  - This reduces the likelihood of creating a superoptimal path, which would weaken the bound.
  - Keep merging nodes until desired width is obtained.
- Underlying idea
  - Preserve accuracy in the region of the diagram that is likely to contain the best solutions.
  - Analogous to using smaller finite elements in models of the atmosphere in regions with more activity.

Merger Heuristics

- Finish time heuristic
  - Merge nodes whose **last finish time states are large**.
  - Paths through these nodes are likely to be longer.
  - Information readily available in the state description.

Merger Heuristics

- Finish time heuristic
  - Merge nodes whose **last finish time states are large**.
  - Paths through these nodes are likely to be longer.
  - Information readily available in the state description.
- Shortest path heuristic
  - Merge nodes whose **shortest-path distances from root are large.**
  - This requires recursively computing shortest paths to root as we go along

Merger Heuristics

- Finish time heuristic
  - Merge nodes whose **last finish time states are large**.
  - Paths through these nodes are likely to be longer.
  - Information readily available in the state description.
- Shortest path heuristic
  - Merge nodes whose **shortest-path distances from root are large.**
  - This requires recursively computing shortest paths to root as we go along

– Random heuristic

  - Randomly choose nodes for merger..

Experimental Design

- Question 1
  - What is the relationship between the width of the relaxed diagram and the quality of the bound?
- Question 2
  - Which node merger heuristic is best?

Experimental Design

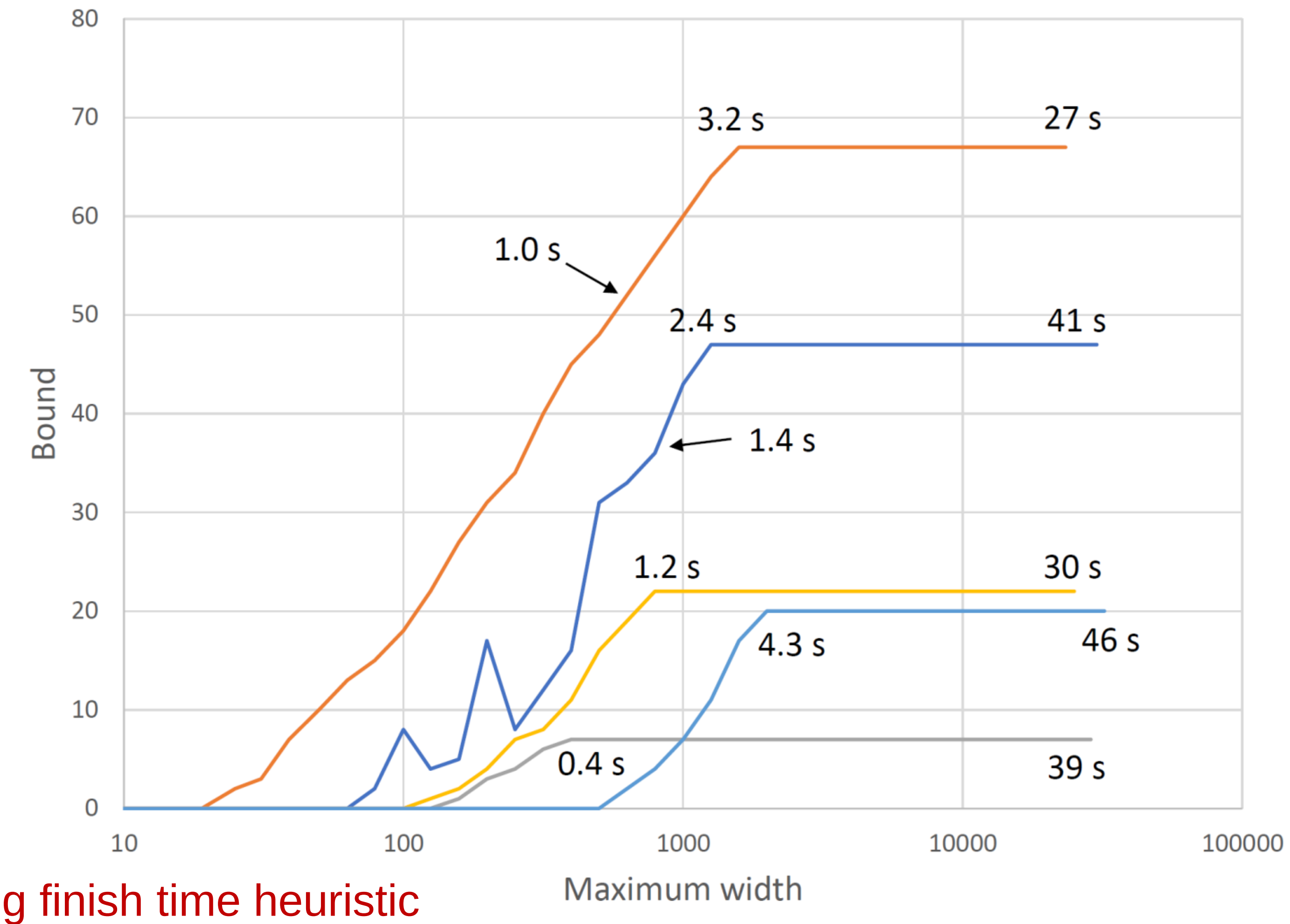
- Problem instances
  - State-dependent processing times.
  - In particular, processing times depend on state variable U
- Benchmark instances?
  - Apparently, none exist for **state-dependent** processing times.
- Random instances?
  - Must generate random instances.

Experimental Design

- Optimal solutions
  - Must have optimal value to assess quality of bound.
  - Because instances frequently have min tardiness = zero.
- How to obtain optimal solutions?
  - Dynamic programming is the only practical method for **state-dependent** processing times.
  - Due to state-space explosion, instances with more than 15 jobs are very hard to solve.
  - Solution with DP is equivalent to generating an **exact** diagram.
  - Solve instances with 12 and 14 jobs.

Computational Results

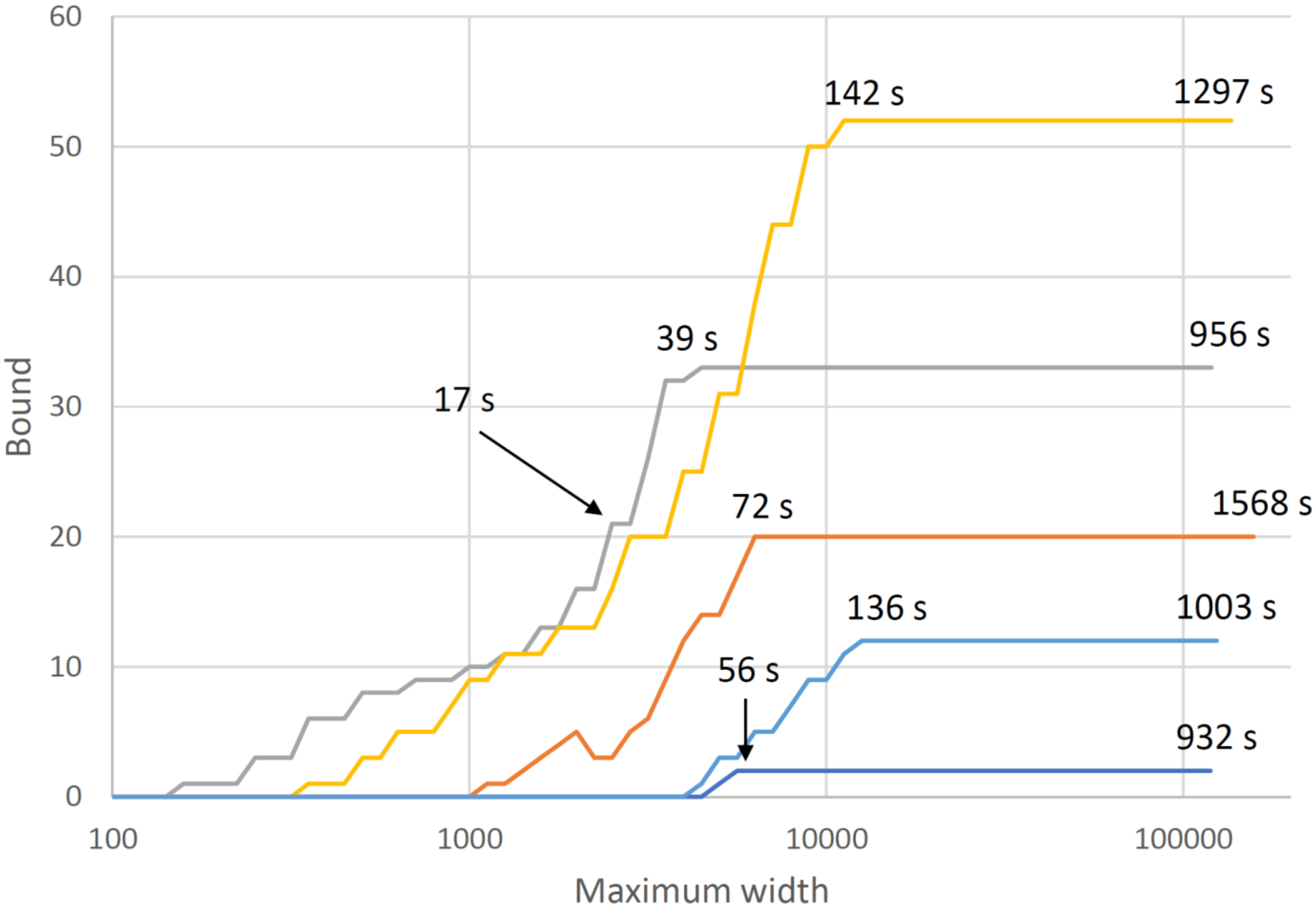
12 jobs



Using finish time heuristic

Computational Results

14 jobs



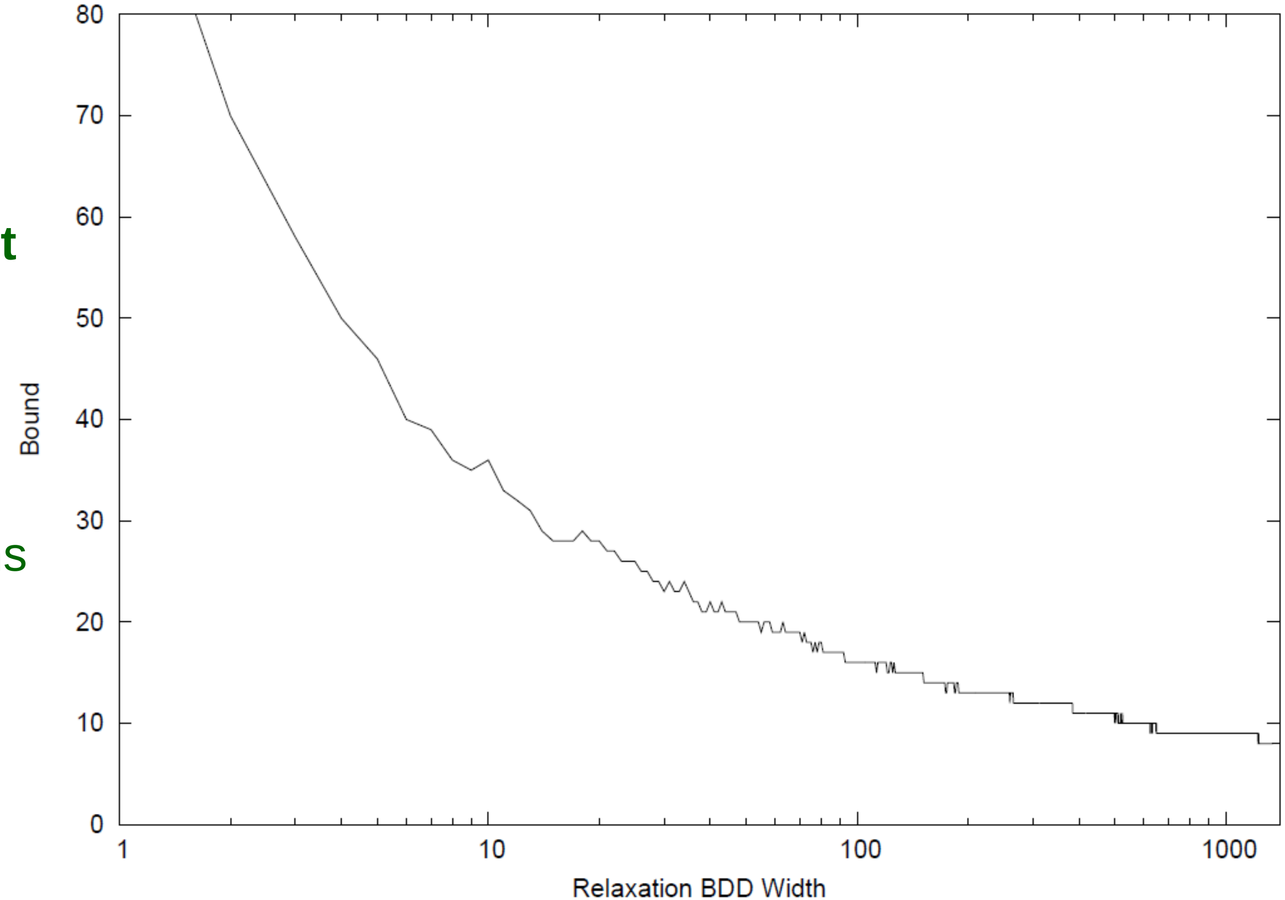
Using finish time heuristic

Computational Results

- Consistent results
  - …across instances. May extend to larger instances.
- Early convergence to optimality
  - For 12 jobs, optimal value achieved when relaxed diagram is 1/32 to 1/15 width of exact diagram.
  - For 14 jobs, optimal value achieved when relaxed diagram is 1/26 to 1/10 width of exact diagram.
- …rather than asymptotic improvement
  - As in studies on other types of problems.

Computational Results

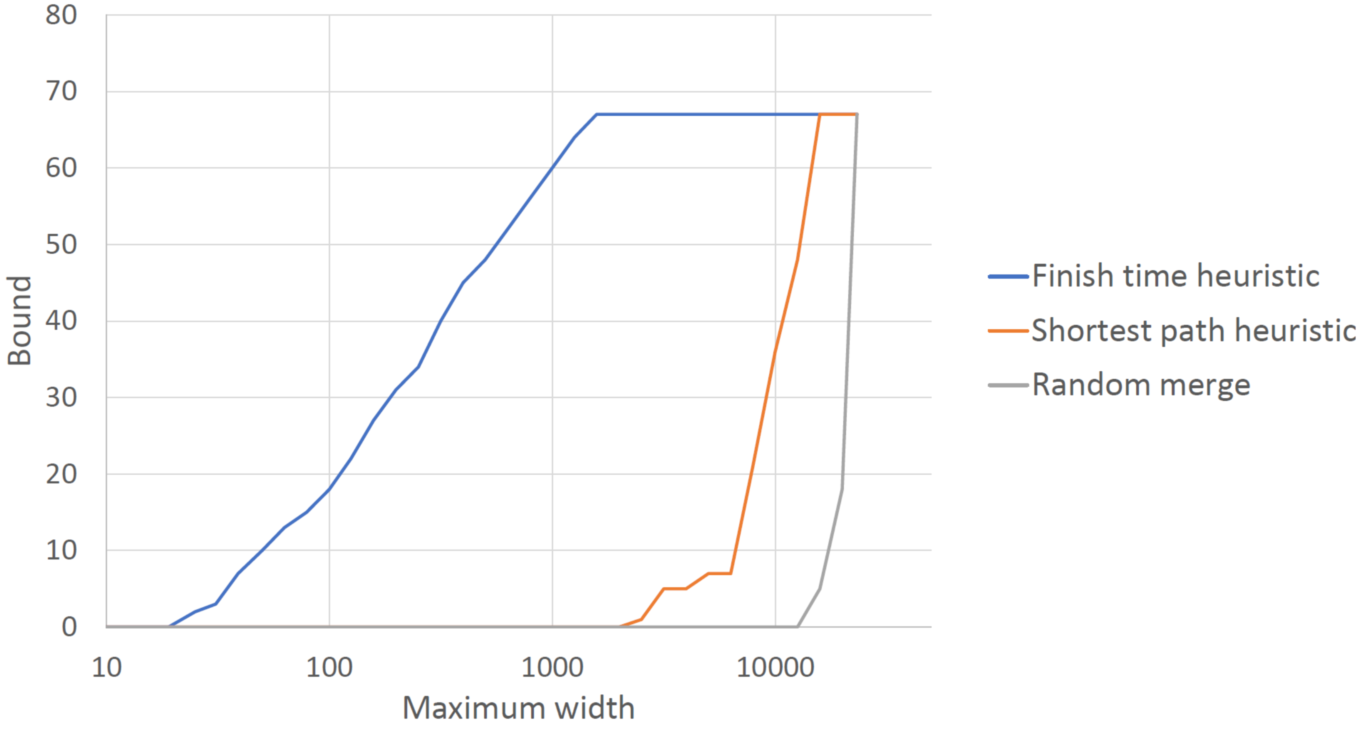
- Asymptotic convergence to optimality
  - Here, **max independent set** problem.
  - We want an **upper** bound
  - Typical of **other** types of problems



Bergman, Ciré, van Hoeve, JH (2013)

Computational Results

Comparison of node merger heuristics on a 12-job instance



Computational Results

- **Finish time heuristic** is vastly superior to others.
  - **Shortest path heuristic** fails because shortest distance to root is usually zero in the upper part of the diagram.
  - This means the heuristic provides no guidance until it is too late.
  - **Random heuristic** provides useless bound.
  - This confirms the importance of a good heuristic.
  - …and potential for further improvement.

Future Work

- Problem: diagrams of a **fixed size** lose their ability to generate bounds as instances scale up.
  - Bound does not rise above zero until relaxed diagram width is 1/1000 to 1/25 that of exact diagram

Future Work

- Problem: diagrams of a **fixed size** lose their ability to generate bounds as instances scale up.
  - Bound does not rise above zero until relaxed diagram width is 1/1000 to 1/25 that of exact diagram
- This suggests a combination with other bounding techniques
  - …that can yield a nonzero bound in smaller relaxed diagrams.
  - Such as **Lagrangean relaxation** obtained by modifying costs in the diagram..

Bergman, Ciré, van Hoeve (2015)

Future Work

- Bounds for **stochastic dynamic programming**
  - From **stochastic diagrams**.
  - Node merger can again provide a valid relaxation.
  - A theoretical result is available.
  - Awaiting good merger heuristics and computational tests.