Combining Optimization and Constraint Programming

John Hooker
Carnegie Mellon University

Amazon Modeling and Optimization
November 2022

Optimization and constraint programming

- A natural combination…
  - Complementary strengths
  - Deep underlying commonality
  - Gradual integration since mid-1990s
  - Now a fast-moving research area

First CP-AI-OR Workshop
Ferrera, Italy, 1999

- In this talk…
  - Broad overview
  - Examples from 2 very active research streams

Survey paper: JH and W. V. van Hoeve, Constraint programming and operations research, *Constraints* **23** (2018) 172-195. Many references.

In this talk…

- What is constraint programming?
  - Employee scheduling, graph coloring, cumulative scheduling
- Schemes for integration
  - Major research streams
- Snapshots of recent research
  - Logic-based Benders decomposition
    - Home healthcare delivery
    - Multiple machine scheduling
    - Stochastic machine scheduling

  - Decision diagrams
    - Tight job sequencing bounds
    - Stochastic maximum clique

- Software

What is constraint programming?

- Grew out of **logic programming** (e.g., Prolog).
  - Steps in a logic program can be interpreted **procedurally** or **declaratively**.
  - Generalized to **constraint logic programming**.

```
grandmother(X, Y) :- mother(X, Z), parent(Z, Y).
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).

mother(mary, stan).
mother(gwen, alice).
mother(valery, gwen).
father(stan, alice).
```

What is constraint programming?

- Grew out of **logic programming** (e.g., Prolog).
  - Steps in a logic program can be interpreted **procedurally** or **declaratively**.
  - Generalized to **constraint logic programming**.

```
grandmother(X, Y) :- mother(X, Z), parent(Z, Y).
parent(X, Y) :- mother(X, Y).
parent(X, Y) :- father(X, Y).

mother(mary, stan).
mother(gwen, alice).
mother(valery, gwen).
father(stan, alice).
```

- Logical formalism dropped**,** resulting in a **constraint program**.
  - A list of **constraints** that are **processed sequentially**.
  - Unlike an optimization model, which is **purely declarative**.

What is constraint programming?

Example: employee scheduling

Assign 4 workers (A,B,C,D) to 3 shifts over 7 days.

CP model (11 constraints):

$$\text{all-different}\big(w[*,d]\big),\ d = 1,\ldots,7$$
$$\text{cardinality}\big(w[*,*],(A,B,C,D),5,6\big)$$
$$\text{nvalues}\big(w[s,*],1,2\big),\ s = 1,2,3$$
$$w[s,d] \in \{A,B,C,D\},\ \text{all } s,d$$

3 different workers assigned to the 3 shifts each day.

Each worker assigned 5 or 6 days.

At most 2 workers assigned to a shift during the week.

Initial domain of variables *w*[*s*,*d*]

$$w[s,d] = \text{worker assigned to shift } s \text{ on day } d$$

All-different, cardinality and **nvalues** are “global” constraints

What is constraint programming?

Example: employee scheduling

Assign 4 workers (A,B,C,D) to 3 shifts over 7 days.

Integer programming model (72 constraints):

$$\sum_i x_{isd} = 1, \text{ all } s, d; \quad \sum_s x_{isd} \le 1, \text{ all } i, d$$

$$5 \le \sum_{s,d} x_{isd} \le 6, \text{ all } i$$

$$\sum_i y_{is} \le 2, \text{ all } s; \quad \sum_d x_{isd} \le 7 y_{is}, \text{ all } i, s$$

$$x_{ids}, y_{is} \in \{0,1\}, \text{ all } i, d, s$$

$x_{isd} = 1$ if worker i assigned to shift s on day d

What is constraint programming?

- How are constraints processed?
  - Variable domains are **filtered** to remove **inconsistent** values (values that cannot satisfy the constraint).
  - Reduced domains **propagated** (passed on) to next constraint.
  - Cycle through constraints until **no further domain reduction** is possible.

$$\begin{array}{l} \text{all-different}(x, y, z) \\ x, y \in \{\text{A,B}\},\ z \in \{\text{A,B,C}\} \end{array}$$

Filtering reduces domain of *z* to {C}.

In general, matching theory is used to filter all-different.

What is constraint programming?

- How are constraints processed?
  - Variable domains are **filtered** to remove **inconsistent** values (values that cannot satisfy the constraint).
  - Reduced domains **propagated** (passed on) to next constraint.
  - Cycle through constraints until **no further domain reduction** is possible.

$$\text{all-different}(x, y, z)$$
$$x, y \in \{\text{A,B}\},\ z \in \{\text{A,B,C}\}$$

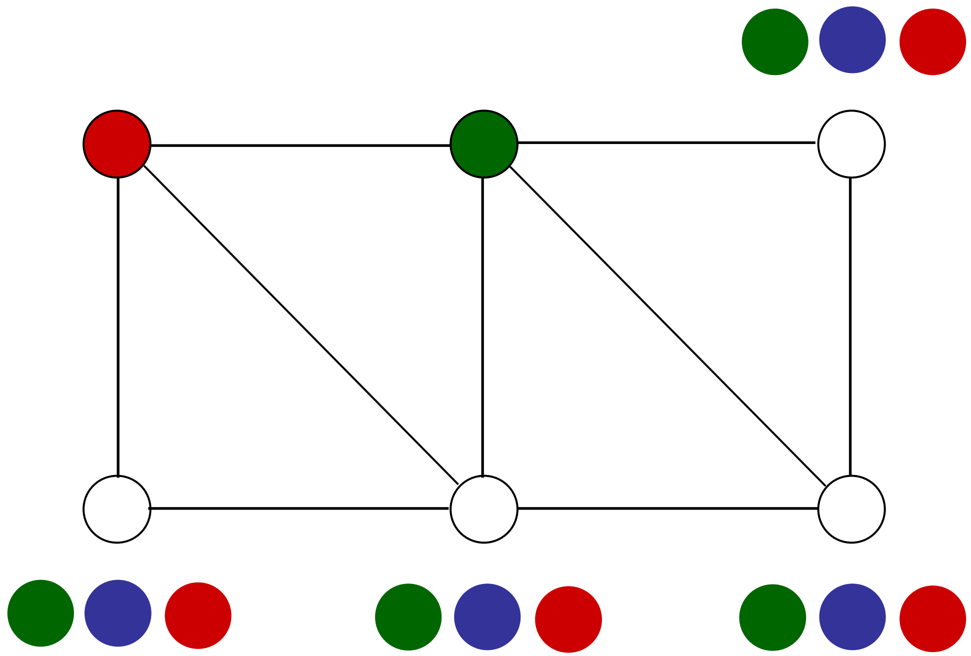
Filtering reduces domain of *z* to {C}.

In general, matching theory is used to filter all-different.

- Then what?
  - If a domain is reduced to empty set, problem is **infeasible**.
  - If all domains are singletons, problem is **solved**.
  - Otherwise, **branch** by splitting a domain (as in IP).

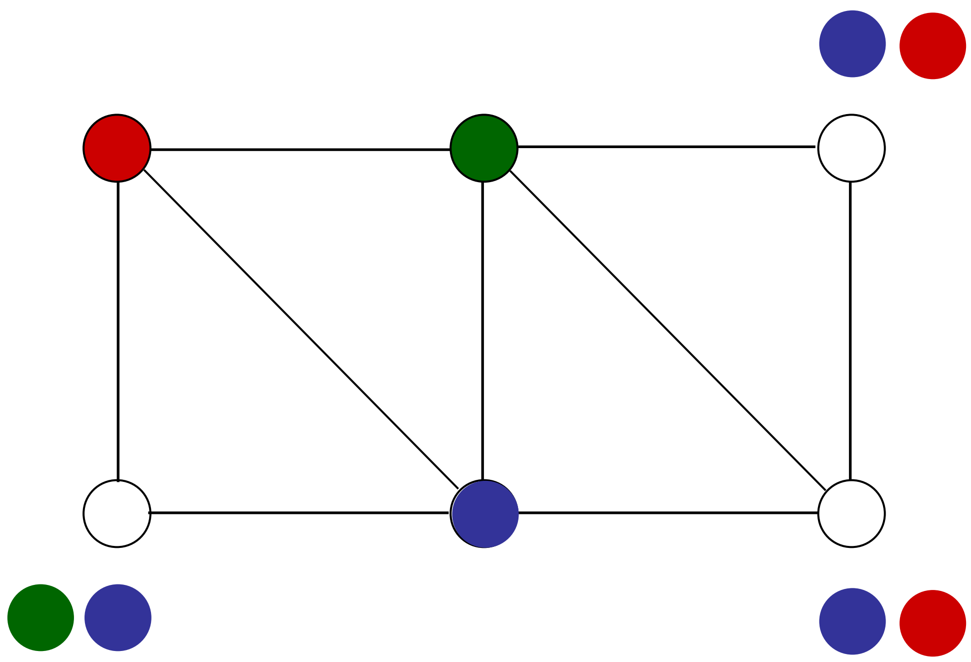
What is constraint programming?

- Example: **graph coloring**
  - Constraints: no 2 adjacent vertices have the same color.
  - Variables: vertex colors. Initial variable domains shown.
  - This instance can be solved by filtering alone.



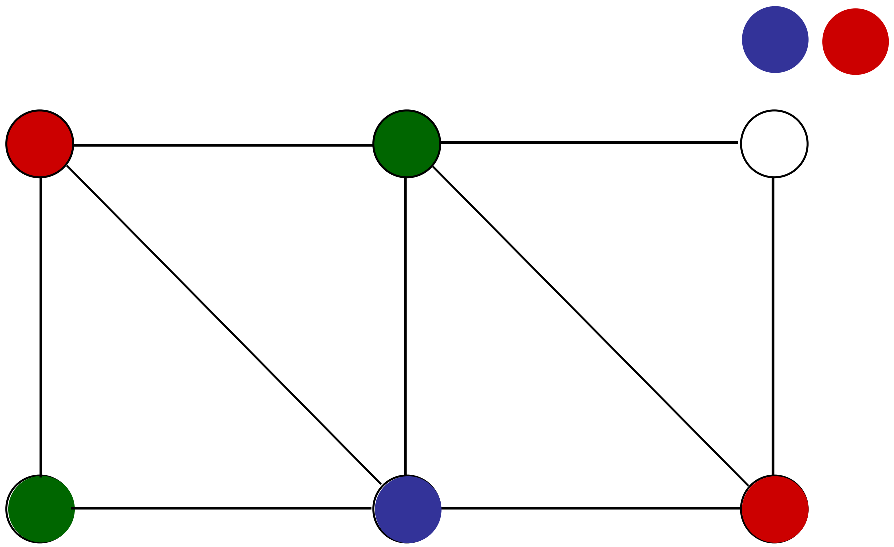
What is constraint programming?

- Example: **graph coloring**
  - Constraints: no 2 vertices have the same color.
  - Variables: vertex colors. Initial variable domains shown.
  - This instance can be solved by filtering alone.



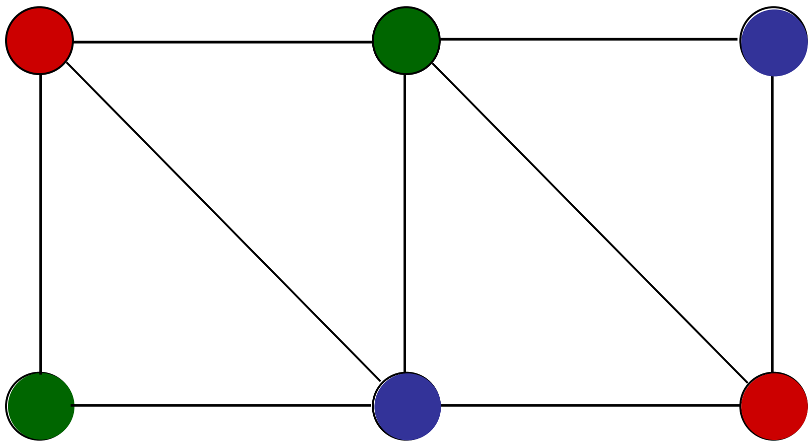
What is constraint programming?

- Example: **graph coloring**
  - Constraints: no 2 vertices have the same color.
  - Variables: vertex colors. Initial variable domains shown.
  - This instance can be solved by filtering alone.



What is constraint programming?

- Example: **graph coloring**
  - Constraints: no 2 vertices have the same color.
  - Variables: vertex colors. Initial variable domains shown.
  - This instance can be solved by filtering alone.



What is constraint programming?

- Example: **cumulative scheduling**
  - Schedule jobs, subject to time windows.
  - Jobs can run simultaneously as long as resource consumption never exceeds *C*.
  - Use the global constraint:

$$\text{cumulative}\Big(\underbrace{(s_1,\ldots,s_n)}_{\text{Job start times (variables)}},\underbrace{(p_1,\ldots,p_n)}_{\text{Job processing times}},\underbrace{(c_1,\ldots,c_n)}_{\text{Job resource requirements}},C\Big)$$

  - Filtered by **edge finding**, originally from optimization literature but now a highly developed technology in CP.

Cumulative scheduling

Consider a problem instance with 3 jobs:

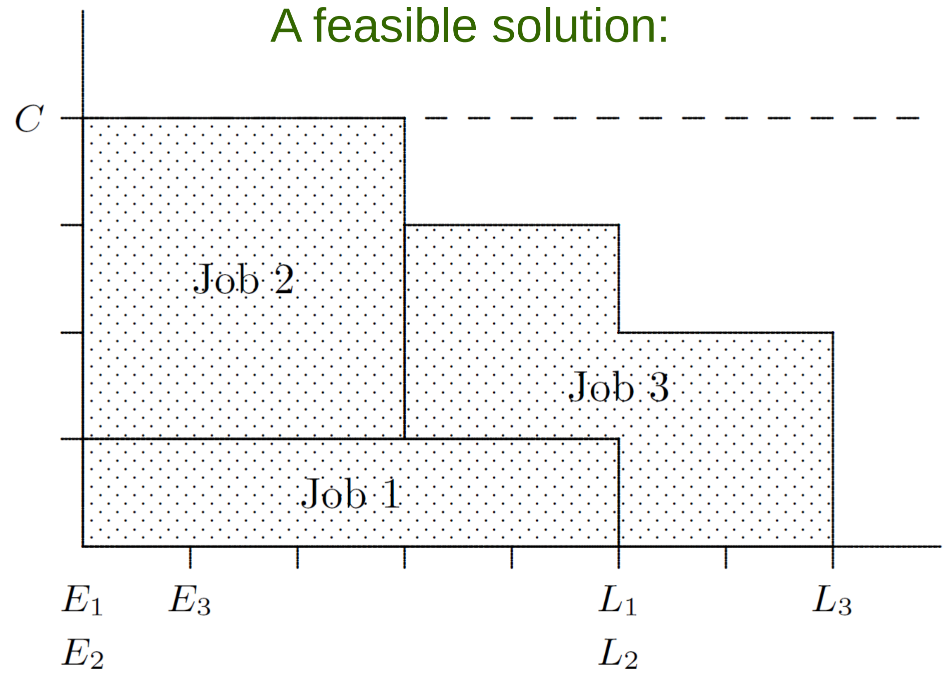
$$\text{cumulative}\big((s_1, s_2, s_3), (p_1, p_2, p_3), (c_1, c_2, c_3), 4\big)$$

Time window*

Job j	p_j	c_j	$[E_j, L_j]$
1	5	1	$[0, 5]$
2	3	3	$[0, 5]$
3	4	2	$[1, 7]$

*Domain of s_j is $[E_j, L_j - p_j]$

A feasible solution:



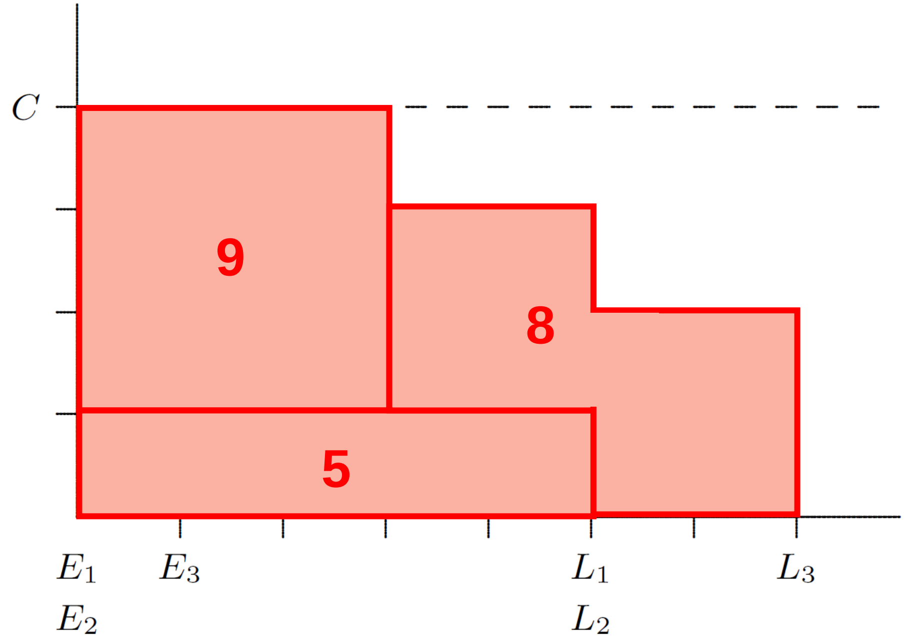
Cumulative scheduling

We can deduce that **job 3 must finish last**.

The total **“energy” (area)** required by all jobs is

$$\boxed{e_3 + e_{\{1,2\}}} > C \cdot (L_{\{1,2\}} - E_{\{1,2,3\}})$$

Total energy required = 22



Cumulative scheduling

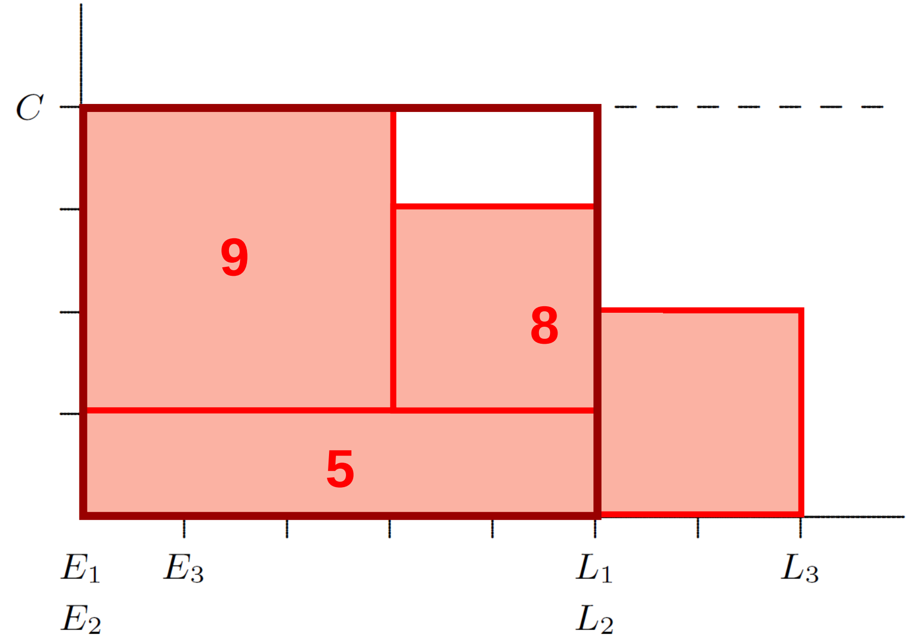
We can deduce that **job 3 must finish last**.

The **available energy if job 3 is not last** is the area between the earliest start time and the deadline of jobs 1 & 2:

$$e_3 + e_{\{1,2\}} > \boxed{C \cdot (L_{\{1,2\}} - E_{\{1,2,3\}})}$$

Total energy required = 22

Area available if job 3 is **not** last = 20



Cumulative scheduling

We can deduce that **job 3 must finish last**.

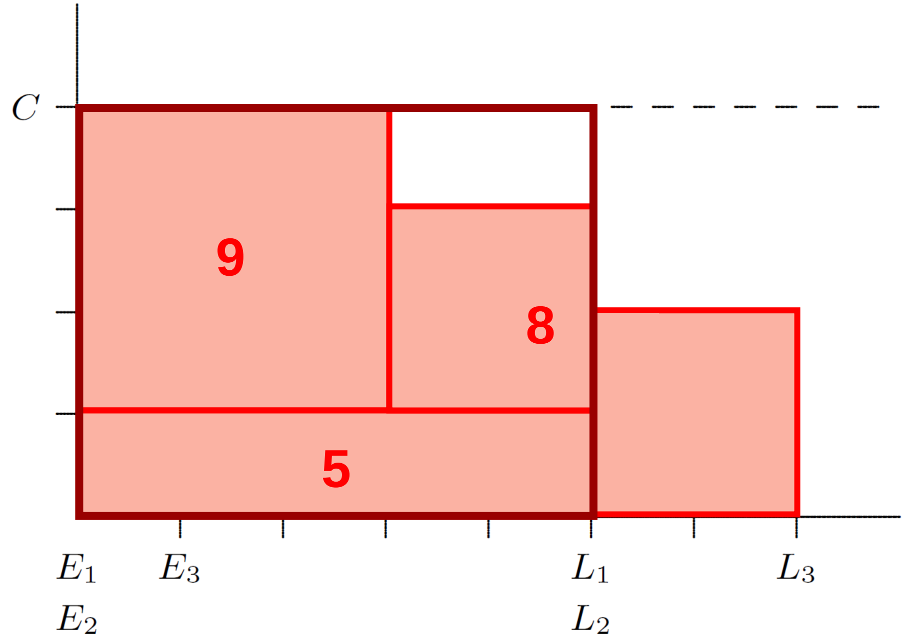
The energy required **exceeds** the available area if job 3 is **not last**:

$$e_3 + e_{\{1,2\}} > C \cdot (L_{\{1,2\}} - E_{\{1,2,3\}})$$

Total energy required = 22

Area available if job 3 is **not** last = 20

Since 22 > 20, job 3 must be **last**

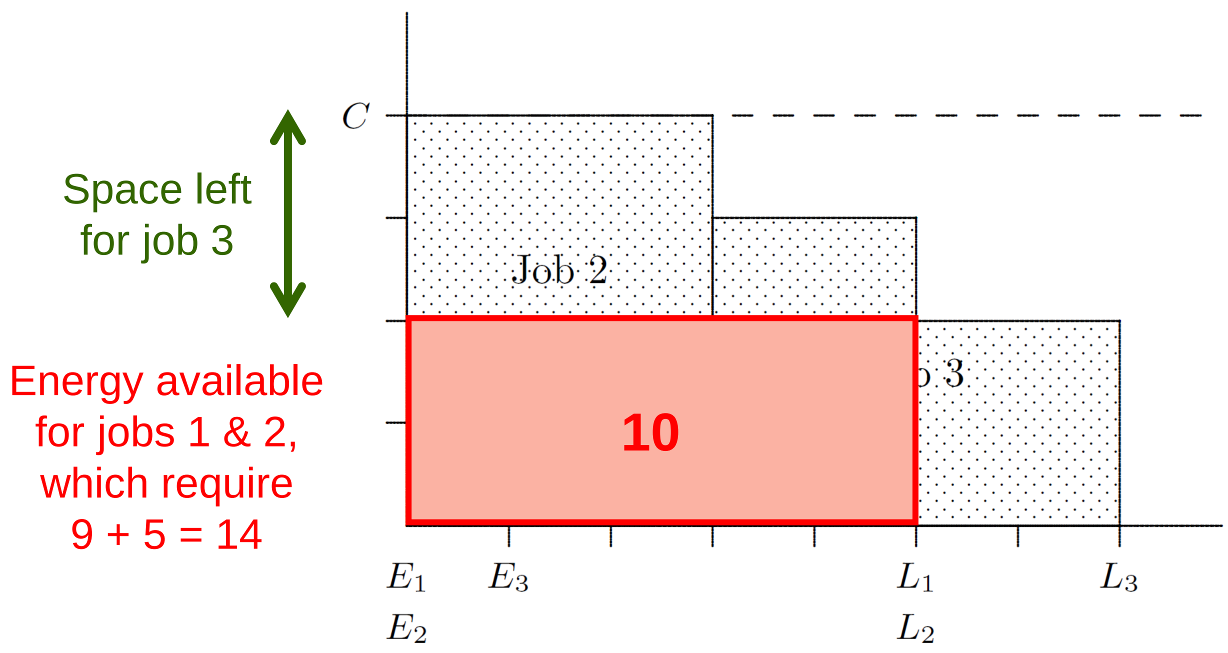


Cumulative scheduling

We now ask **how early can job 3 start?**

Energy available for jobs 1 & 2 **if space is left for job 3 to start anytime**:

$$E_{\{1,2\}} + \frac{e_{\{1,2\}} - \boxed{(C - c_3)(L_{\{1,2\}} - E_{\{1,2\}})}}{c_3}$$



Cumulative scheduling

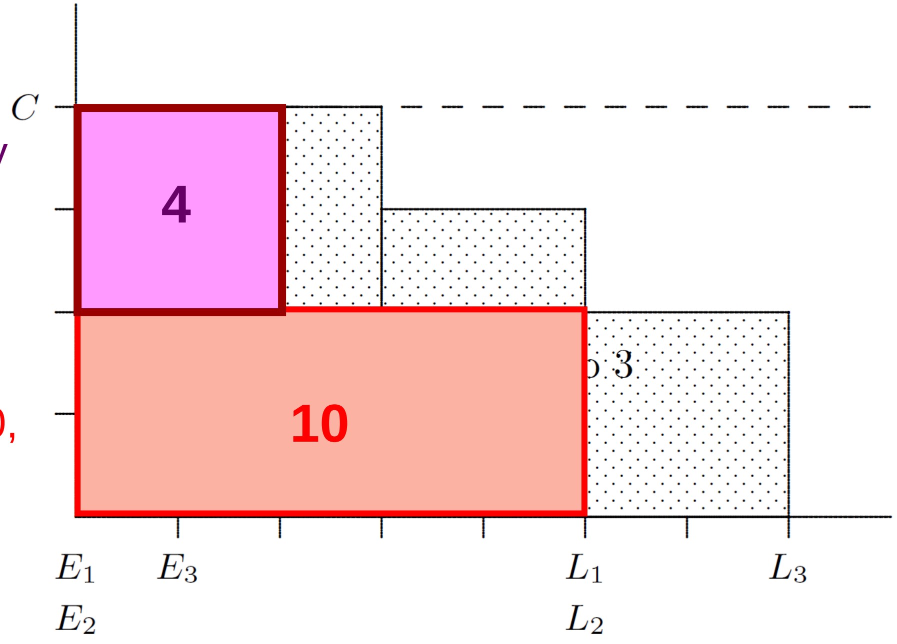
We now ask **how early can job 3 start?**

Additional energy required by jobs 1 & 2:

$$E_{\{1,2\}} + \frac{\boxed{e_{\{1,2\}} - (C - c_3)(L_{\{1,2\}} - E_{\{1,2\}})}}{c_3}$$

Additional energy required by jobs 1 & 2 is 14 – 10 = 4

Energy available for jobs 1 & 2 is 10, but they require 9 + 5 = 14



Cumulative scheduling

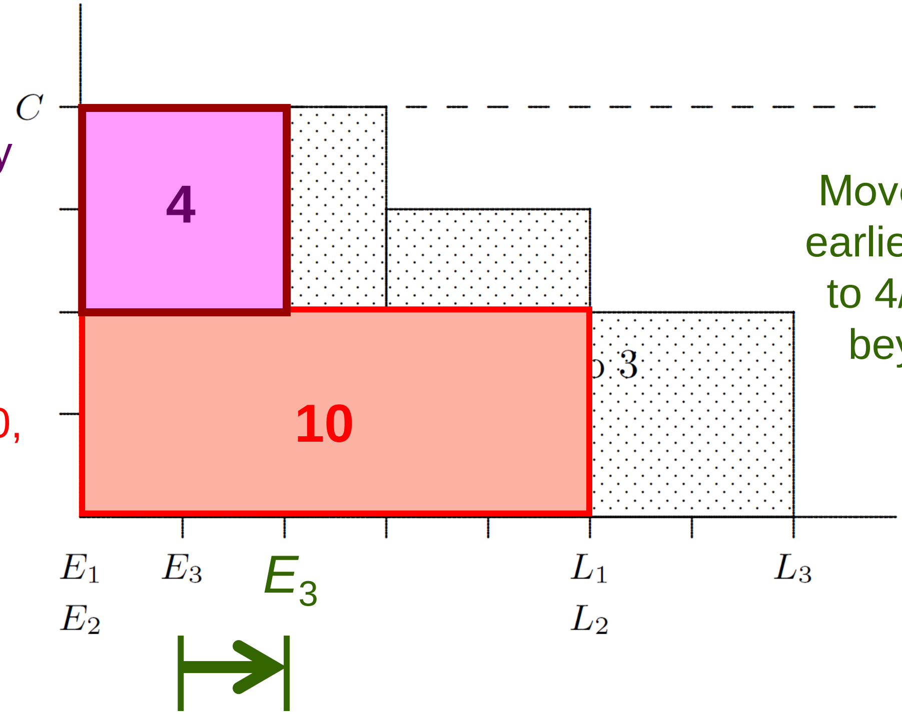
We deduce that job 3 can start **no earlier than time 2**.

We can now **reduce domain** of s_3 from [1,3] to [2,3] by moving up job 3's earliest start time to

$$\boxed{E_{\{1,2\}} + \frac{e_{\{1,2\}} - (C - c_3)(L_{\{1,2\}} - E_{\{1,2\}})}{c_3}}$$

Additional energy required by jobs 1 & 2 is 14 – 10 = 4

Energy available for jobs 1 & 2 is 10, but they require 9 + 5 = 14



Move up job 3's earliest start time to 4/2 = 2 units beyond $E_{\{1,2\}}$

Cumulative scheduling

- Now what?
  - An O(n^2) algorithm finds all applications of the edge finding rule.
  - Apply additional domain reduction rules.
  - If no solution identified, **branch** on which job is first, etc.
- Other domain reduction rules:
  - Extended edge finding.
  - Timetabling.
  - Not-first/not-last rules.
  - Energetic reasoning.

CP & optimization compared

CP

- Deals naturally with discrete variables
  - which need not be numerical
- Good at sequencing/scheduling
  - where MILP has weak relaxations
- Messy constraints OK
  - More constraints make the problem easier.
- Powerful modeling language
  - Global constraints lead to succinct models
  - and convey structure to solver.

Traditional Opt

- Deals naturally with continuous variables
  - using numerical methods
- Good at knapsack constraints, assignments, costs
  - which have tight relaxations
- Focus on optimality bounds
  - due to advanced relaxation technology
- Highly engineered solvers
  - at least for LP, MILP
  - due to decades of development

Schemes for combining CP & optimization

- Optimization-based **filtering** methods.
  - **Network and matching** theory for sequencing constraints.
  - **Dynamic programming** for employee scheduling constraints.
  - **Edge-finding** for disjunctive & cumulative scheduling constraints.
- Constraint **propagation + relaxation**.
  - In a branching context, **reduce domains** with CP and **tighten relaxation** with cutting planes.
  - Each builds on the other.
- CP-based **column generation**.
  - For **branch-and-price** methods.
- Logic-based **Benders decomposition**.
  - Allows CP and optimization **solvers to cooperate**.
- **Decision diagrams**.
  - Combine constraint **propagation** with discrete **relaxation**.

Logic-based Benders decomposition

- Useful when fixing certain variables greatly **simplifies** problem.
  - **Master problem** searches over ways to fix variables.
  - **Subproblem** solves simplified problem that remains.
  - **Benders cut** from subproblem guides next solution of master problem.
- LBBD is an **extension** of classical Benders decomposition.
  - Subproblem can be **any** optimization problem (not just LP).
  - Benders cuts based on **inference dual** (rather than LP dual).
- Frequently used to **combine** math programming and CP.
  - For instance, **MILP** solves master problem, **CP** solves subproblem.

Survey paper: JH, Logic-based Benders decomposition for large-scale optimization, in *Large-Scale Optimization Applied to Supply Chain and Smart Manufacturing,* Springer (2019)

Forthcoming book: JH, *Logic-based Benders Decomposition: Theory and Applications*, Springer (2023)

Some LBBD applications

- Planning and scheduling:
  - Machine allocation and scheduling
  - Steel production scheduling
  - Chemical batch processing (BASF, etc.)
  - Auto assembly line management (Peugeot-Citroën)
  - Allocation and scheduling of multicore processors (IBM, Toshiba, Sony)
  - Edge-cloud computing
  - Container port management
  - Electric vehicle ride sharing

Some LBBD applications

- Planning and scheduling:
  - Lock scheduling
  - Shift scheduling
  - Flow shop scheduling
  - Hospital scheduling
  - Covid vaccine delivery
  - Mass Covid testing
  - Optimal control of dynamical systems
  - Sports scheduling
  - Underground mine scheduling
  - Multiperiod distribution network logistics

Some LBBD applications

- Routing and scheduling
  - Multiple vehicle routing
  - Drone-assisted parcel delivery
  - Home health care
  - Food distribution
  - Automated guided vehicles in flexible manufacturing
  - Traffic diversion around blocked routes
  - Concrete delivery
  - Train dispatching

Some LBBD applications

- Planning and scheduling:
  - Allocation of frequency spectrum (U.S. FCC)
  - Wireless local area network design
  - Facility location-allocation
  - Stochastic facility location and fleet management
  - Wind turbine maintenance
  - Queuing design and control

29

Some LBBD applications

- Other:
  - Logical inference (SAT solvers essentially use Benders)
  - Logic circuit verification
  - Warehouse robot control
  - Shelf space allocation
  - Bicycle sharing
  - Service restoration in a network
  - Infrastructure resilience planning
  - Supply chain management
  - Space packing
  - Part assembly planning

Logic-based Benders decomposition

- Solves problem of the form

$$\begin{aligned}&\min\ f(\mathbf{x},\mathbf{y})\\&(\mathbf{x},\mathbf{y})\in S\\&\mathbf{x}\in D_{\mathbf{x}},\ \mathbf{y}\in D_{\mathbf{y}}\end{aligned}$$

Master problem

$$\begin{aligned}&\min\ z\\&z\ge g_k(\mathbf{x}),\ \text{all cuts } k\\&\mathbf{x}\in D_{\mathbf{x}}\end{aligned}$$

Minimize cost z subject to bounds given by Benders cuts, obtained from values of x attempted in previous iterations k.

Trial value $\bar{x}$ that solves master →

← Benders cut $z \geq g_k(x)$

Subproblem

$$\begin{aligned}&\min\ f(\bar{\mathbf{x}},\mathbf{y})\\&(\bar{\mathbf{x}},\mathbf{y})\in S\\&\mathbf{y}\in D_{\mathbf{y}}\end{aligned}$$

Obtain proof of optimality (solution of inference dual). Use same proof to deduce cost bounds for other assignments, yielding Benders cut.

JH (2000), JH & Ottosson (2003)

LBBD example: Home healthcare

- Caregiver assignment and routing
  - Focus on regular hospice care
  - Qualifications matched to patient needs
  - Time windows, breaks, etc., observed
  - Weekly schedule
- Rolling time horizon
  - New patients every week.
  - Minimal schedule change for existing patients.
- Efficient staff utilization
  - Maximize number of patients served by given staff level.
  - Optimality important, due to cost of taking on staff.

Heching, JH, Kimura (2019)

LBBD example: Home healthcare

Master problem

Assign patients to healthcare aides and days of the week

$$\max \sum_j \delta_j$$

= 1 if patient *j* scheduled

$$\sum_i x_{ij} = \boxed{\delta_j}, \quad \text{all } j$$

Required number of visits per week

$$\sum_{i,k} \boxed{y_{ijk}} = \boxed{v_j} \delta_j, \quad \text{all } j$$

= 1 if patient *j* assigned to aide *I* on day *k*

$$y_{ijk} \le \boxed{x_{ij}}, \quad \text{all } i, j, k$$

Spacing constraints on visit days

Benders cuts

Relaxation of subproblem

$$\delta_j, x_{ij}, y_{ijk} \in \{0, 1\}$$

= 1 if patient *j* assigned to aide *i*

MILP model

LBBD example: Home healthcare

Subproblem

Sequence and schedule visits for each healthcare aide *j* separately.

*n*th patient in sequence

Patients assigned to aide *i*

$$\text{all-different}\{\boxed{\pi_{k\nu}} \mid \nu = 1,\ldots,|\boxed{P_i}|\}$$

Start time

$$[s_j, s_j + p_j] \subseteq [r_j, d_j]$$

$$\boxed{s_{\pi_{k\nu}}} + \boxed{p_{\pi_{k\nu}}} + \boxed{t_{\pi_{k\nu}\pi_{k,\nu+1}}} \leq s_{\pi_{k,\nu+1}}, \quad \text{all } k,\ \nu$$

Visit duration

Travel time

CP model

(or use interval variables)

LBBD example: Home healthcare

Benders cuts

If no feasible schedule for aide *j*, generate a cut requiring that at least one patient be assigned to another aide.

$$\sum_{j \in \boxed{\bar{P}_{ik}}} (1 - y_{ijk}) \geq 1$$

Reduced set of patients whose assignment to aide *i* on day *k* creates infeasibility, obtained by re-solving subproblem with fewer aides. This excludes many assignments that cannot be feasible.

Branch and check

Variant of LBBD that generates Benders cuts during branch-and-bound solution of master problem. Master problem solved only once.

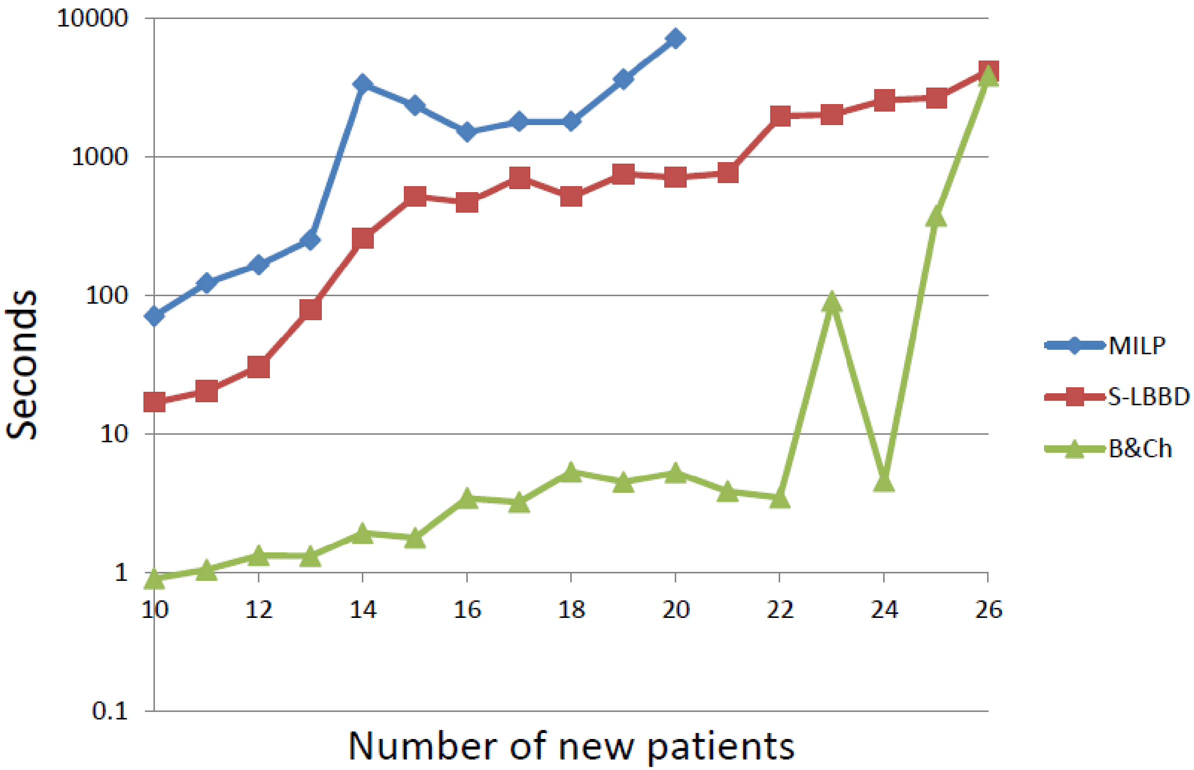
JH (2000), Thorsteinsson (2001)

LBBD example: Home healthcare

Computational results

Heching, JH, Kimura (2019)

Data from home hospice care firm.



Better results for slightly easier instances in Grenouilleau, Lahrichi, Rousseau (2020)

LBBD example: Home healthcare

Computational results

Heching, JH, Kimura (2019)

Data from Danish home care agency.

Instance	Patients	Crews	Weighted objective			Covering objective		
			MILP	LBBD	B&Ch	MILP	LBBD	B&Ch
hh	30	15	*	3.16	**1.41**	*	**23.3**	441
ll1	30	8	*	1.74	**0.43**	*	108	**1.41**
ll2	30	7	2868	1.56	**0.32**	*	**1.38**	6.45
ll3	30	6	1398	2.16	**0.30**	*	**3.07**	5.98

*Computation time exceeded one hour.

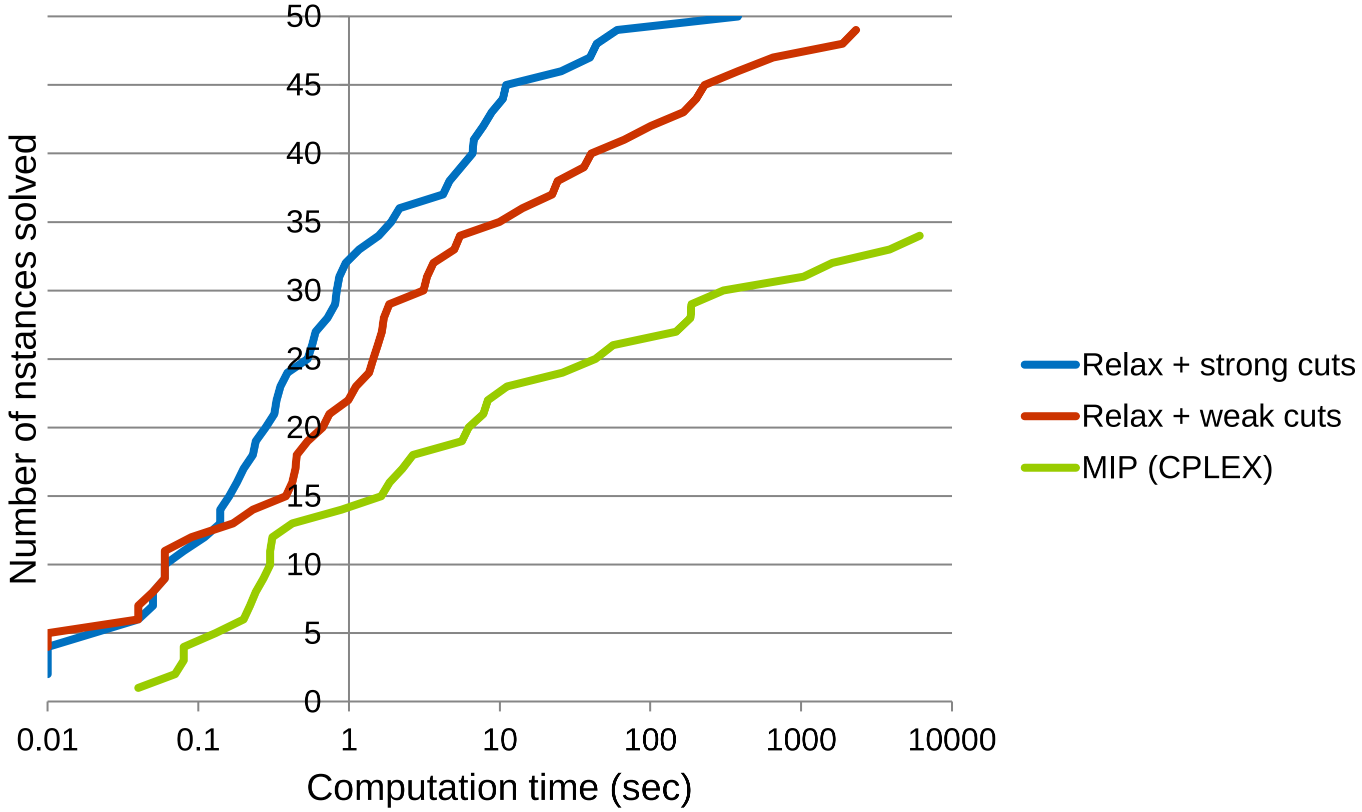
LBBD example: Multiple machine scheduling

- Master problem
  - Use **MILP** to assign tasks to (nonidentical) machines.
  - Minimize makespan, etc.
- Subproblem
  - Schedule tasks on each machine, subject to time windows.
  - Use **CP** (cumulative scheduling) for each machine.
  - Minimize makespan, etc.
- Benders cuts
  - Use **analytical cuts** based on structure of subproblem.

JH (2007)

LBBD example: Multiple machine scheduling

Performance profile for 50 problem instances



Ciré, Coban, JH (2015)

LBBD example: Stochastic machine scheduling

- **Random** processing times
  - Represented by multiple scenarios.
  - Processing times revealed after machine assignment but before scheduling on each machine.
  - Solve subproblem by CP
- Previous state of the art
  - Integer L-shaped method.
  - Classical Benders cuts based on LP relaxation of MILP subproblem.
  - Weak “integer cuts” to ensure convergence.

LBBD example: Stochastic machine scheduling

Computation time

10 jobs, 2 machines, processing times drawn from uniform distribution

Each time (seconds) is average over 3 instances

Scenarios	*Integer L-shaped*	*Branch & Check*
1	127	1
5	839	2
10	2317	3
50	> 3600	17
100	> 3600	37
500	> 3600	279

Elçi and JH (2022)

Decision diagrams

- Binary decision diagrams
  - Graphical representation of Boolean function.
  - Traditionally used for logic circuit verification, product configuration, etc.
  - Can be generalized to **multivalued** DDs.

Lee (1959), Akers (1978)

Bryant (1986)

Survey paper: M.P. Castro, A.A. Cire, J.C. Beck, Decision diagrams for discrete optimization: A survey of recent advances, *INFORMS Journal on Computing* **34** (2022)

Decision diagrams

- Binary decision diagrams
  - Graphical representation of Boolean function.
  - Traditionally used for logic circuit verification, product configuration, etc.
  - Can be generalized to **multivalued** DDs.

Lee (1959), Akers (1978)

Bryant (1986)

- Constraint programming applications
  - Representation and **filtering** of global constraints (e.g. table constraint).
  - **Relaxed DDs** provide data structure for **constraint propagation**.

Andersen, Hadžić, JH, Tiedemann (2007)

Hadžić, JH, O'Sullivan, Tiedemann (2008)

Survey paper: M.P. Castro, A.A. Cire, J.C. Beck, Decision diagrams for discrete optimization: A survey of recent advances, *INFORMS Journal on Computing* **34** (2022)

Decision diagrams

- Binary decision diagrams
  - Graphical representation of Boolean function. Lee (1959), Akers (1978)
  - Traditionally used for logic circuit verification, product configuration, etc.
  - Can be generalized to **multivalued** DDs. Bryant (1986)

- Constraint programming applications
  - Representation and **filtering** of global constraints (e.g. table constraint).
  - **Relaxed DDs** provide data structure for **constraint propagation**.

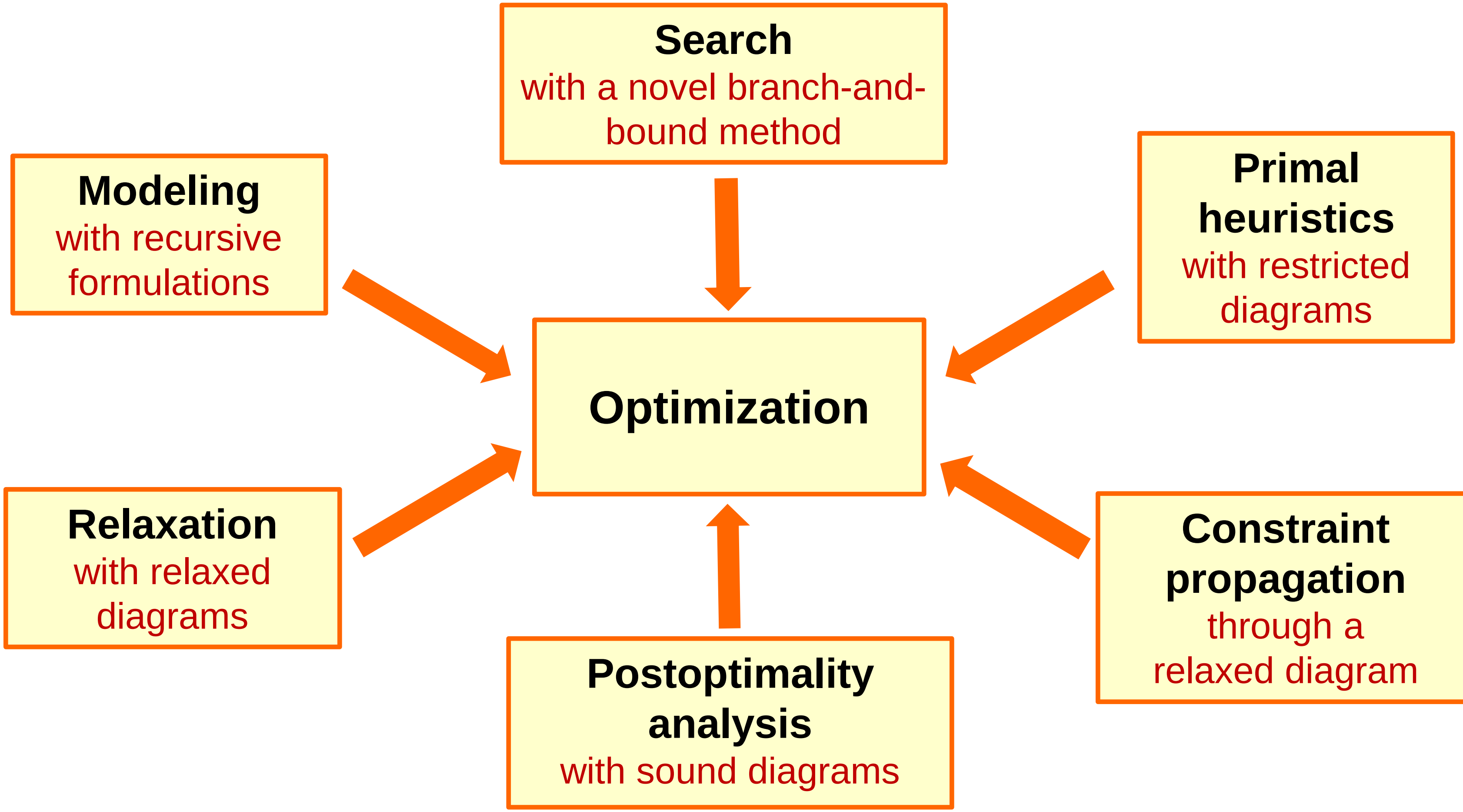
Andersen, Hadžić, JH, Tiedemann (2007)

Hadžić, JH, O'Sullivan, Tiedemann (2008)

- A new perspective on optimization Hadžić and JH (2006, 2007)
  - DDs can perform all functions of an optimization solver...

Survey paper: M.P. Castro, A.A. Cire, J.C. Beck, Decision diagrams for discrete optimization: A survey of recent advances, *INFORMS Journal on Computing* **34** (2022)

Decision diagrams



Book: D. Bergman, A. A. Cire, W. J. van Hoeve, JH,
Decision Diagrams for Optimization, Springer (2016)

DD example: Job sequencing bounds

- Sequence jobs
  - Release times and due dates.
  - Minimize total tardiness.
  - Problems often **too hard** to solve to proven optimality.
- Find a tight bound on min tardiness
  - To evaluate **heuristic** solutions.
  - Use **DDs** and **Lagrangian relaxation** on **dynamic programming** model.

Job

j	r_j	p_j	d_j
1	0	3	5
2	1	2	3
3	1	2	5

Release time

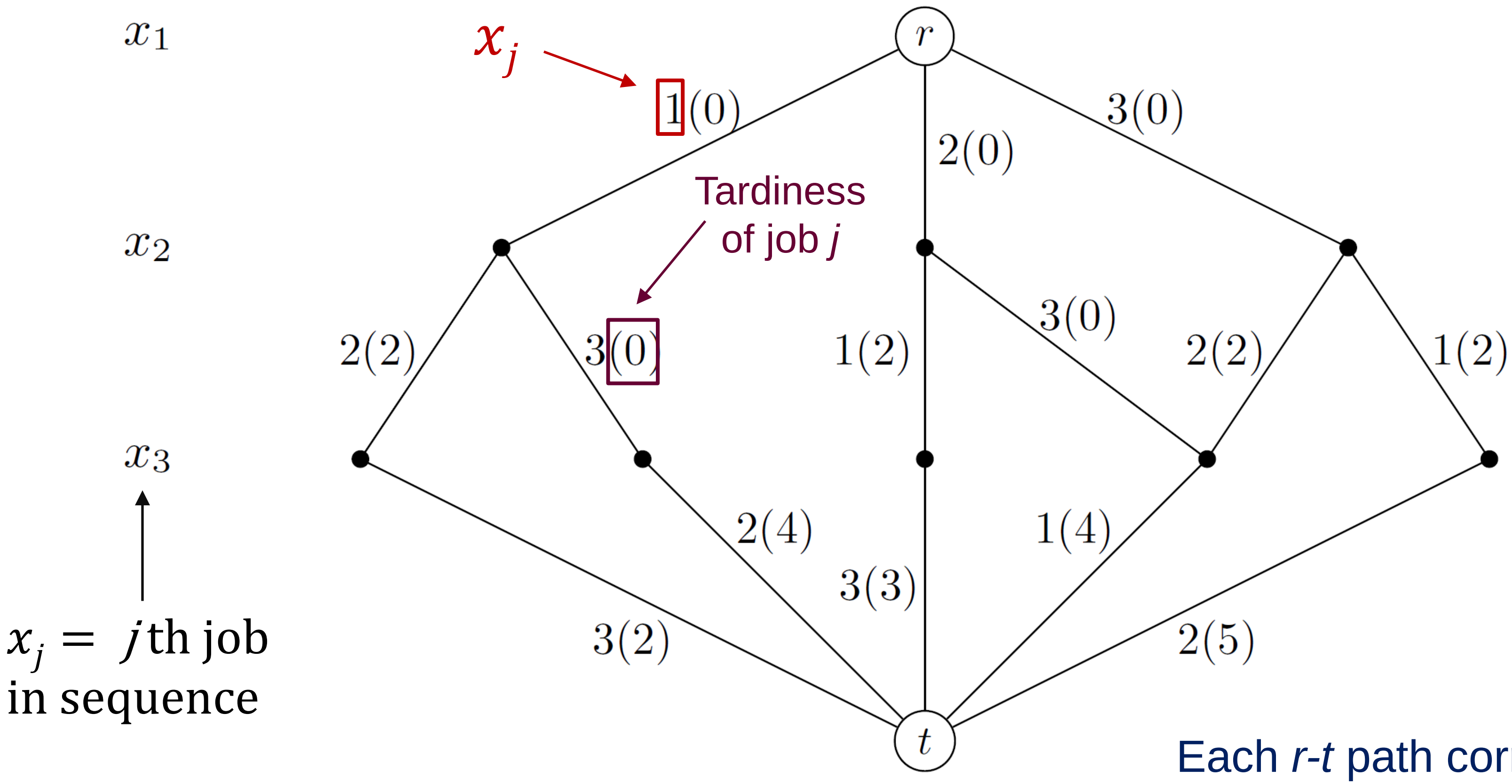
Due date

Processing time

DD example: Job sequencing bounds

j	r_j	p_j	d_j
1	0	3	5
2	1	2	3
3	1	2	5

Decision diagram for job sequencing



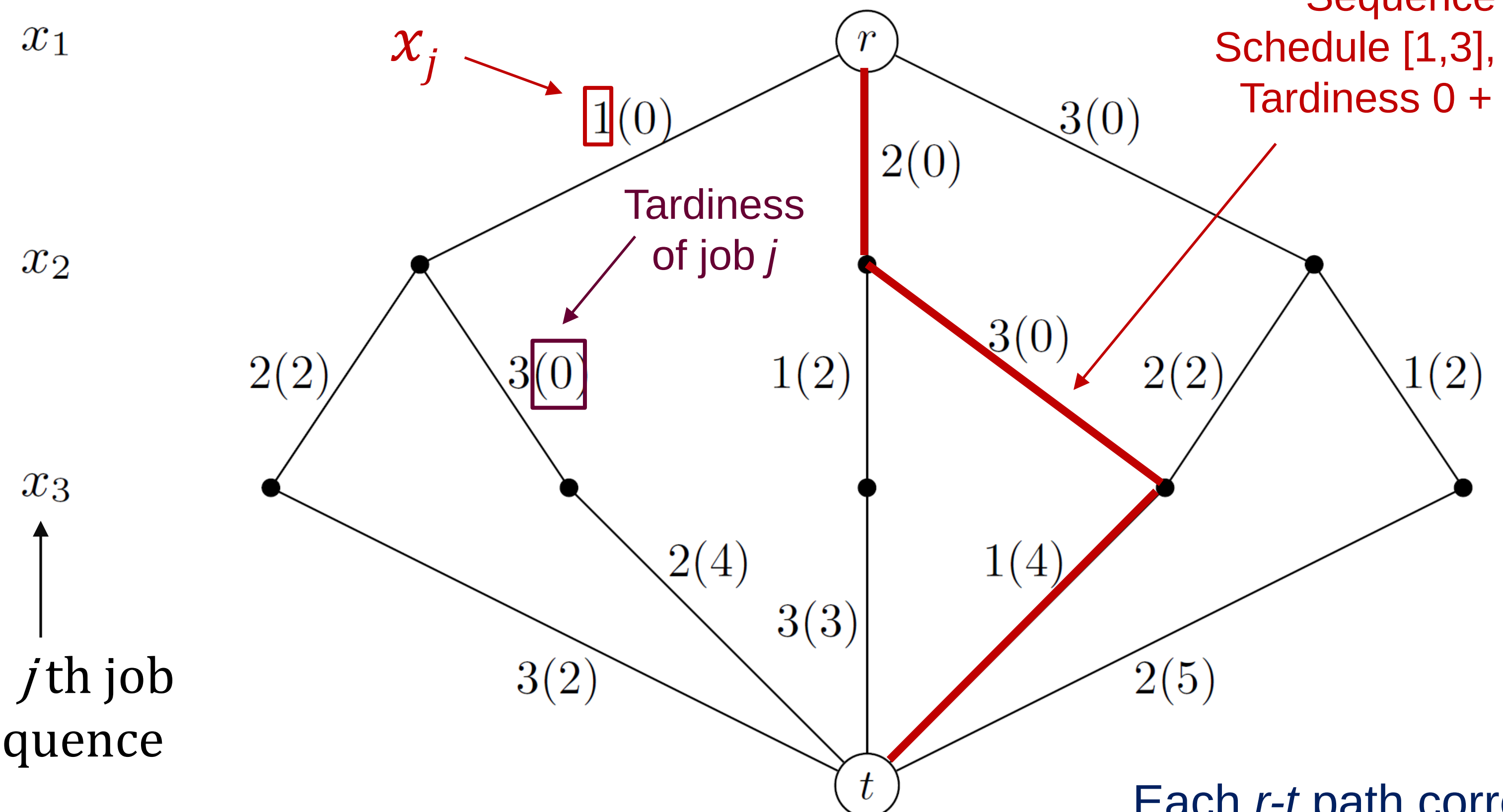
Each r-t path corresponds to a feasible solution

DD example: Job sequencing bounds

j	r_j	p_j	d_j
1	0	3	5
2	1	2	3
3	1	2	5

Decision diagram for job sequencing

An optimal solution:
Sequence 2-3-1
Schedule [1,3], [3,5], [5,7]
Tardiness 0 + 0 + 4 = 4



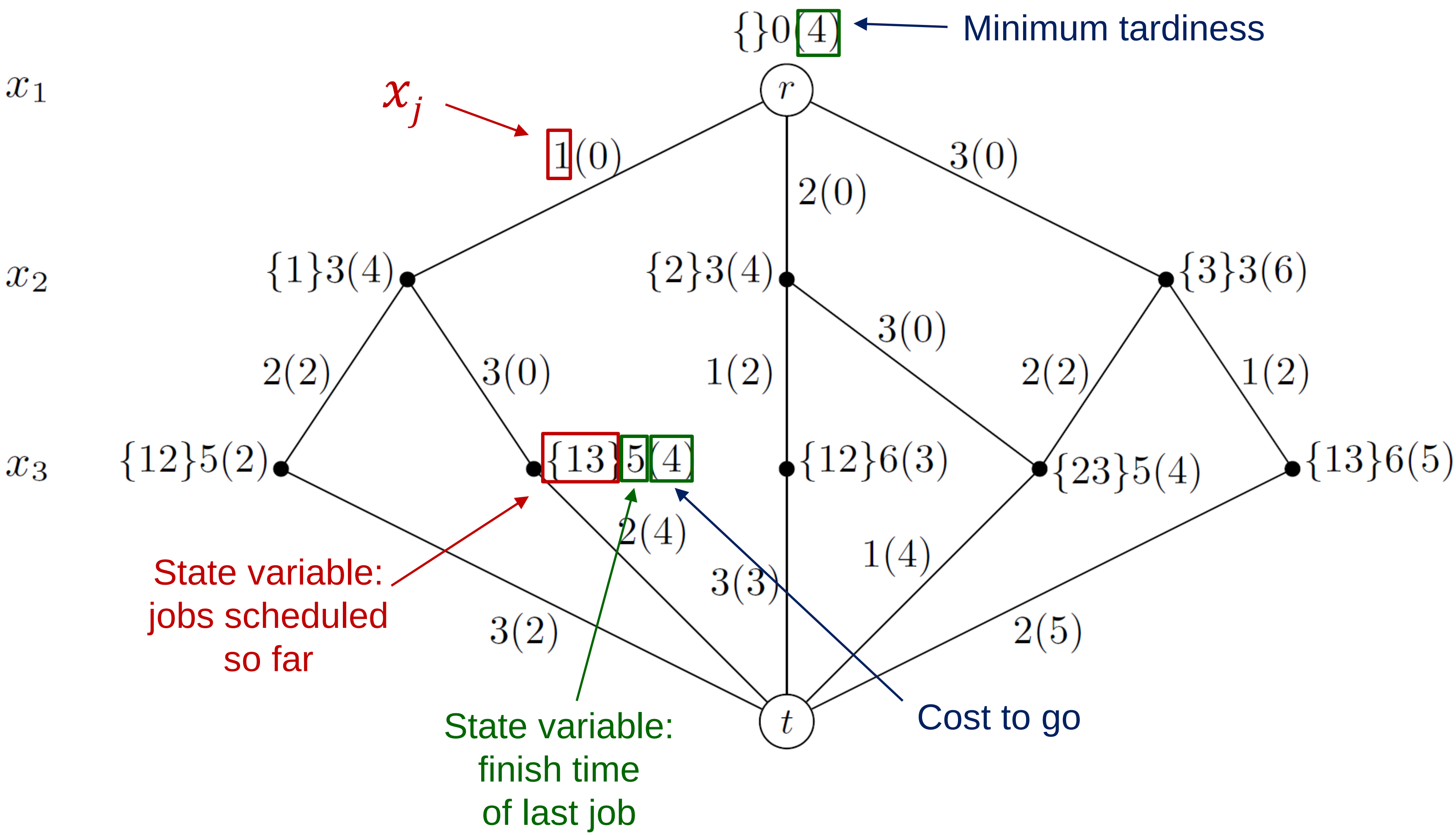
x_j = jth job in sequence

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

DD example: Job sequencing bounds

j	r_j	p_j	d_j
1	0	3	5
2	1	2	3
3	1	2	5

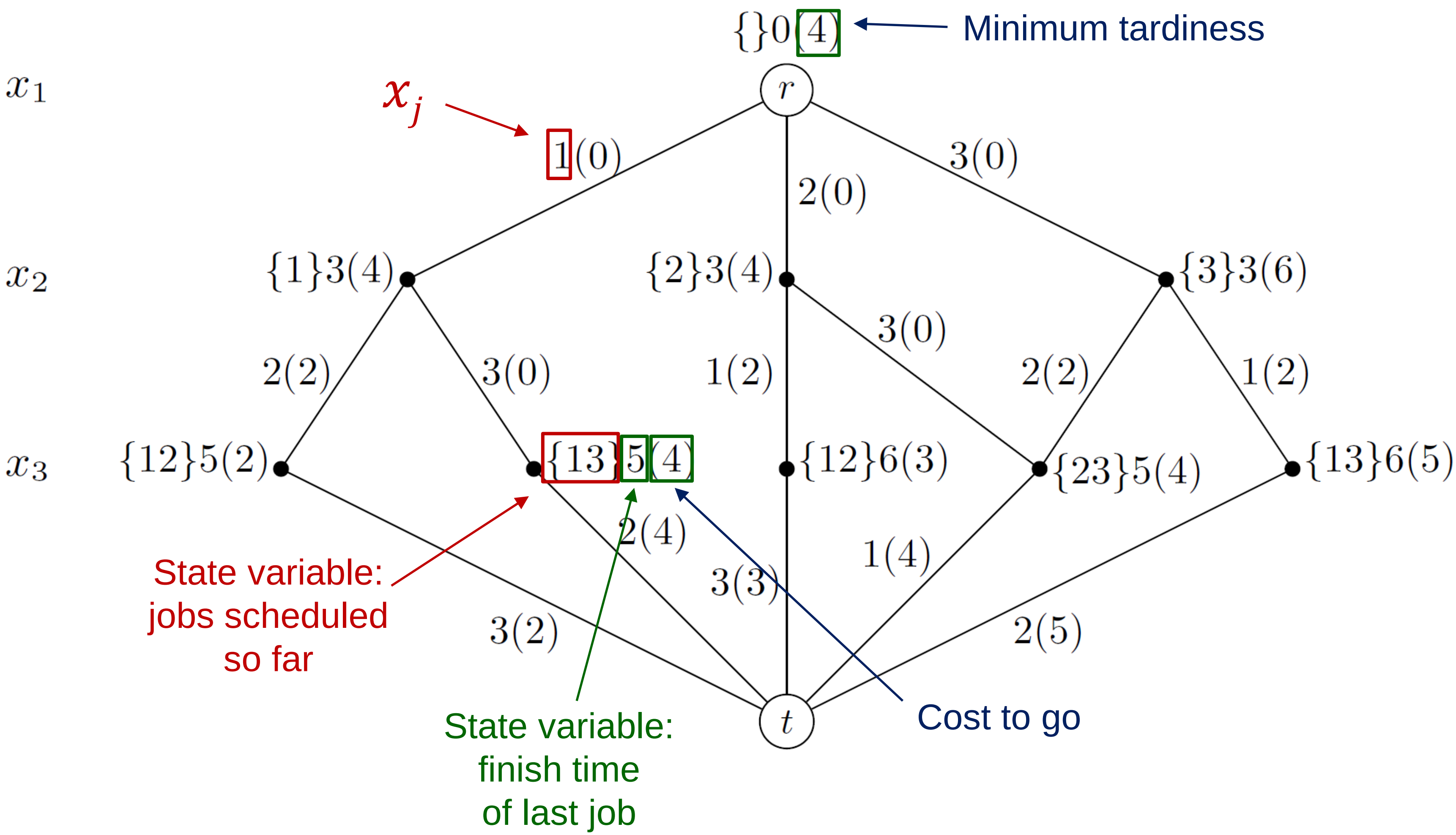
Interpret DD as dynamic programming
state transition graph



DD example: Job sequencing bounds

j	r_j	p_j	d_j
1	0	3	5
2	1	2	3
3	1	2	5

Interpret DD as dynamic programming
state transition graph

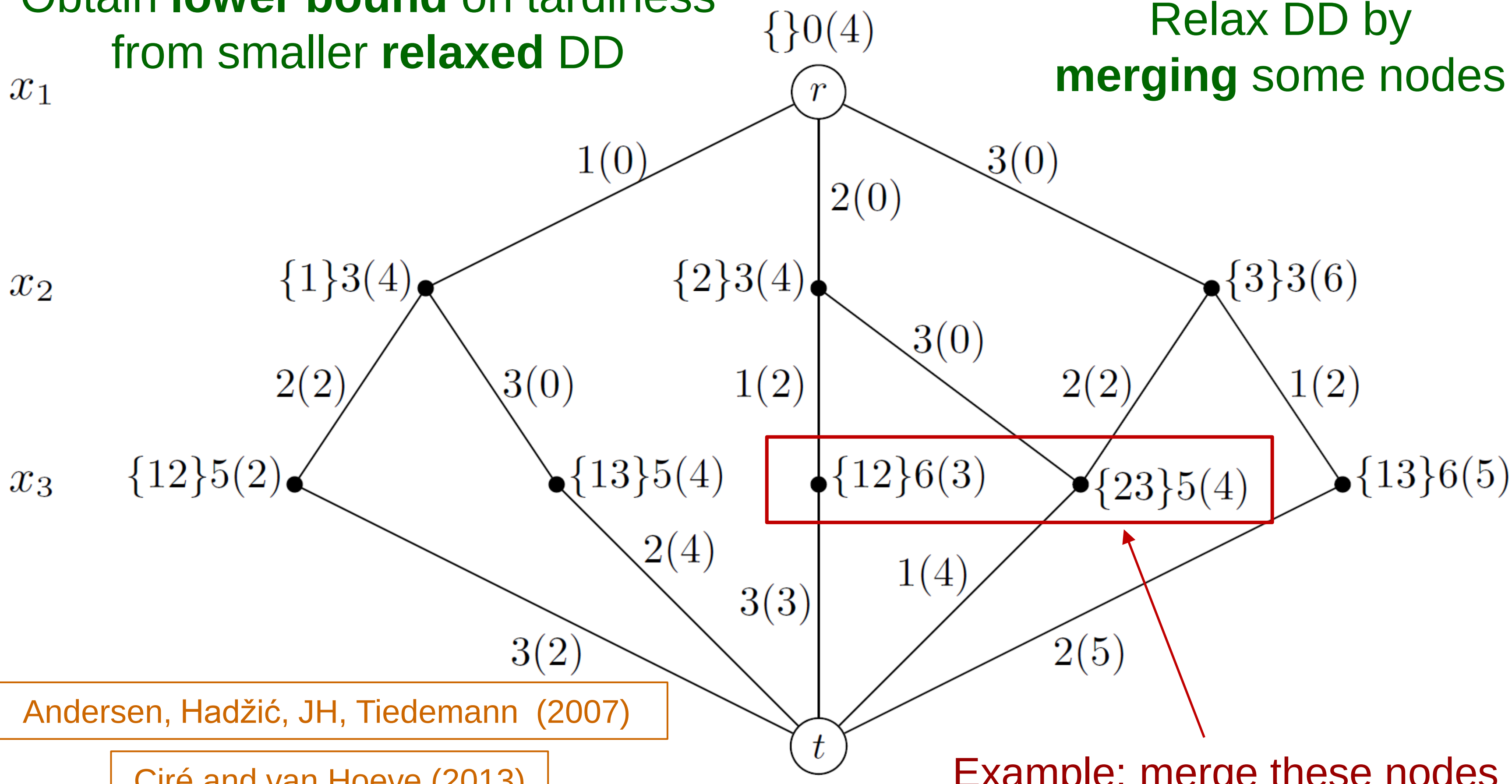


DD example: Job sequencing bounds

j	r_j	p_j	d_j
1	0	3	5
2	1	2	3
3	1	2	5

Exact DD grows exponentially.
Obtain **lower bound** on tardiness from smaller **relaxed** DD

Relax DD by **merging** some nodes



Andersen, Hadžić, JH, Tiedemann (2007)

Ciré and van Hoeve (2013)

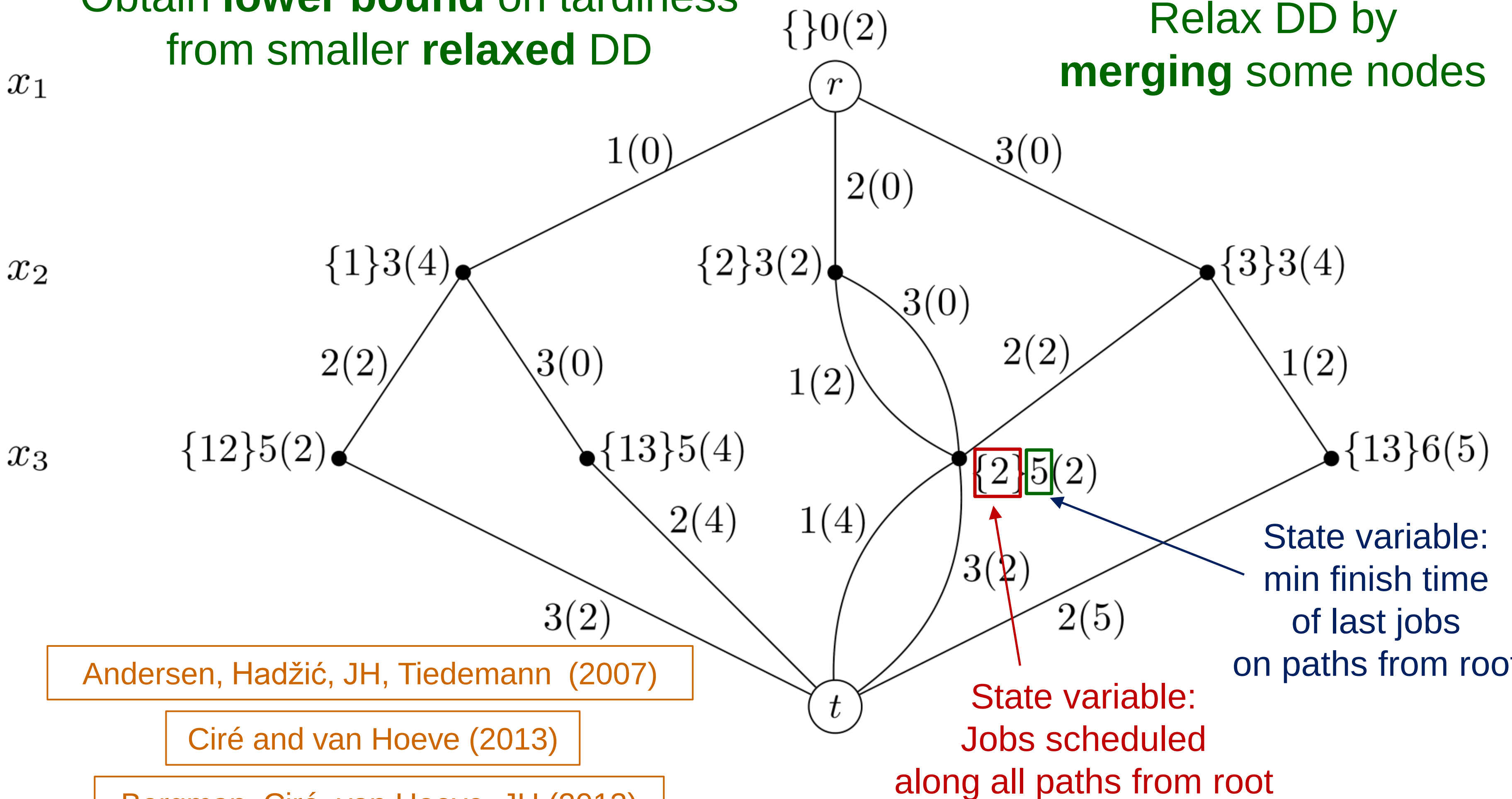
Bergman, Ciré, van Hoeve, JH (2013)

DD example: Job sequencing bounds

j	r_j	p_j	d_j
1	0	3	5
2	1	2	3
3	1	2	5

Exact DD grows exponentially.
Obtain **lower bound** on tardiness
from smaller **relaxed** DD

Relax DD by
merging some nodes



Andersen, Hadžić, JH, Tiedemann (2007)

Ciré and van Hoeve (2013)

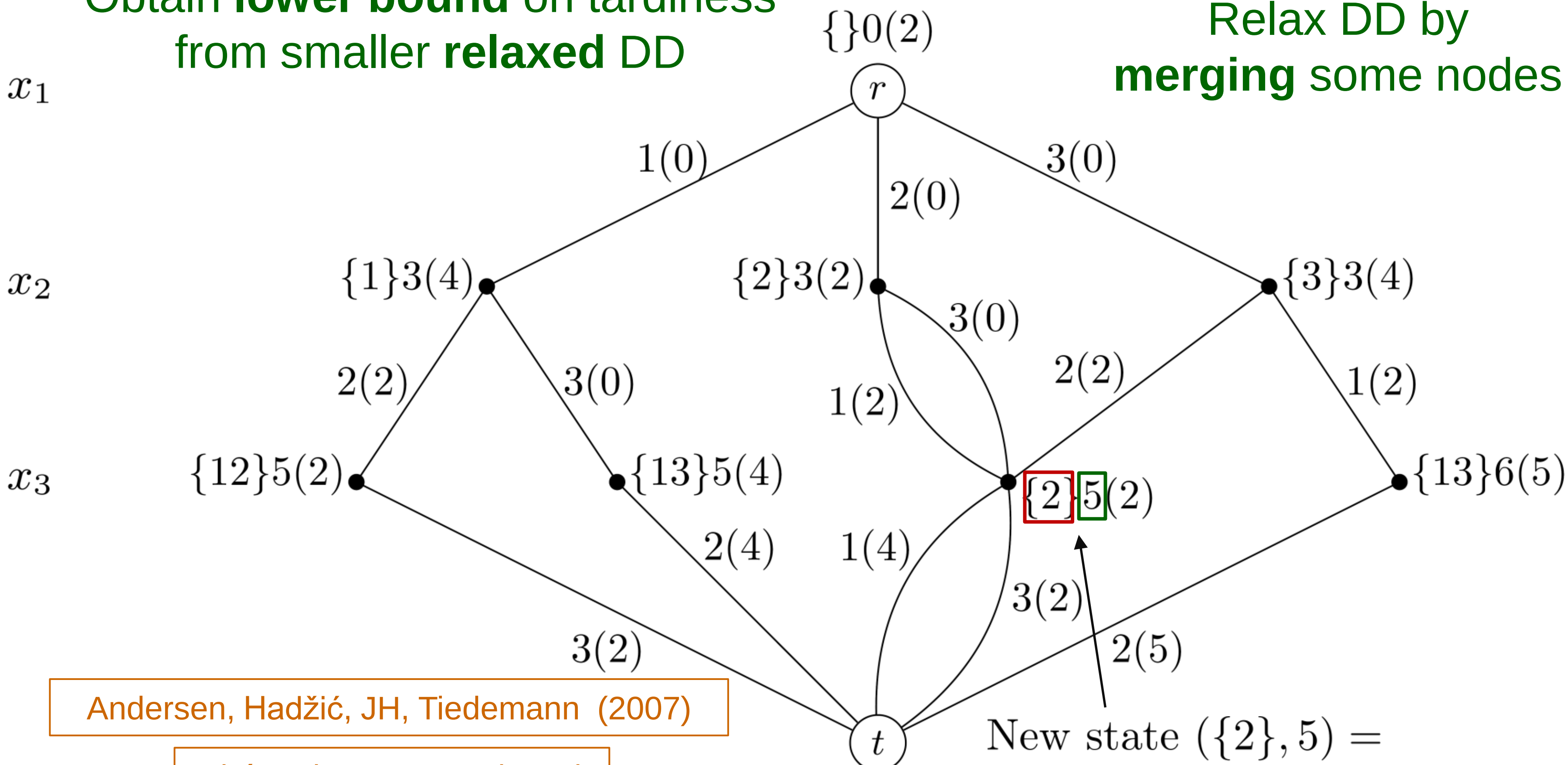
Bergman, Ciré, van Hoeve, JH (2013)

DD example: Job sequencing bounds

j	r_j	p_j	d_j
1	0	3	5
2	1	2	3
3	1	2	5

Exact DD grows exponentially.
Obtain **lower bound** on tardiness
from smaller **relaxed** DD

Relax DD by
merging some nodes



New state $(\{2\}, 5) =$
$= (\{1, 2\} \cap \{2, 3\}, \min\{6, 5\})$

Andersen, Hadžić, JH, Tiedemann (2007)

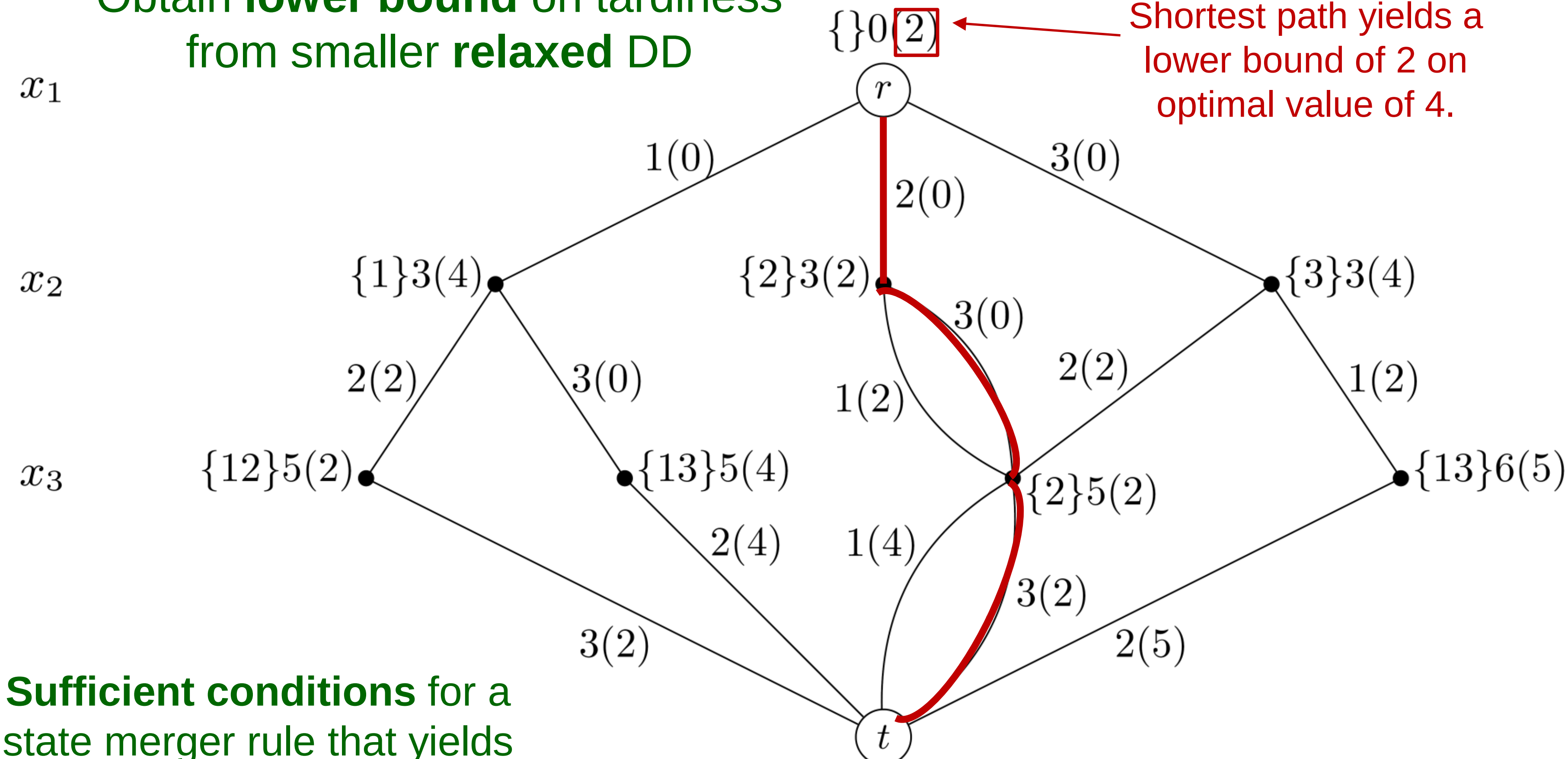
Ciré and van Hoeve (2013)

Bergman, Ciré, van Hoeve, JH (2013)

DD example: Job sequencing bounds

j	r_j	p_j	d_j
1	0	3	5
2	1	2	3
3	1	2	5

Exact DD grows exponentially.
Obtain **lower bound** on tardiness from smaller **relaxed** DD

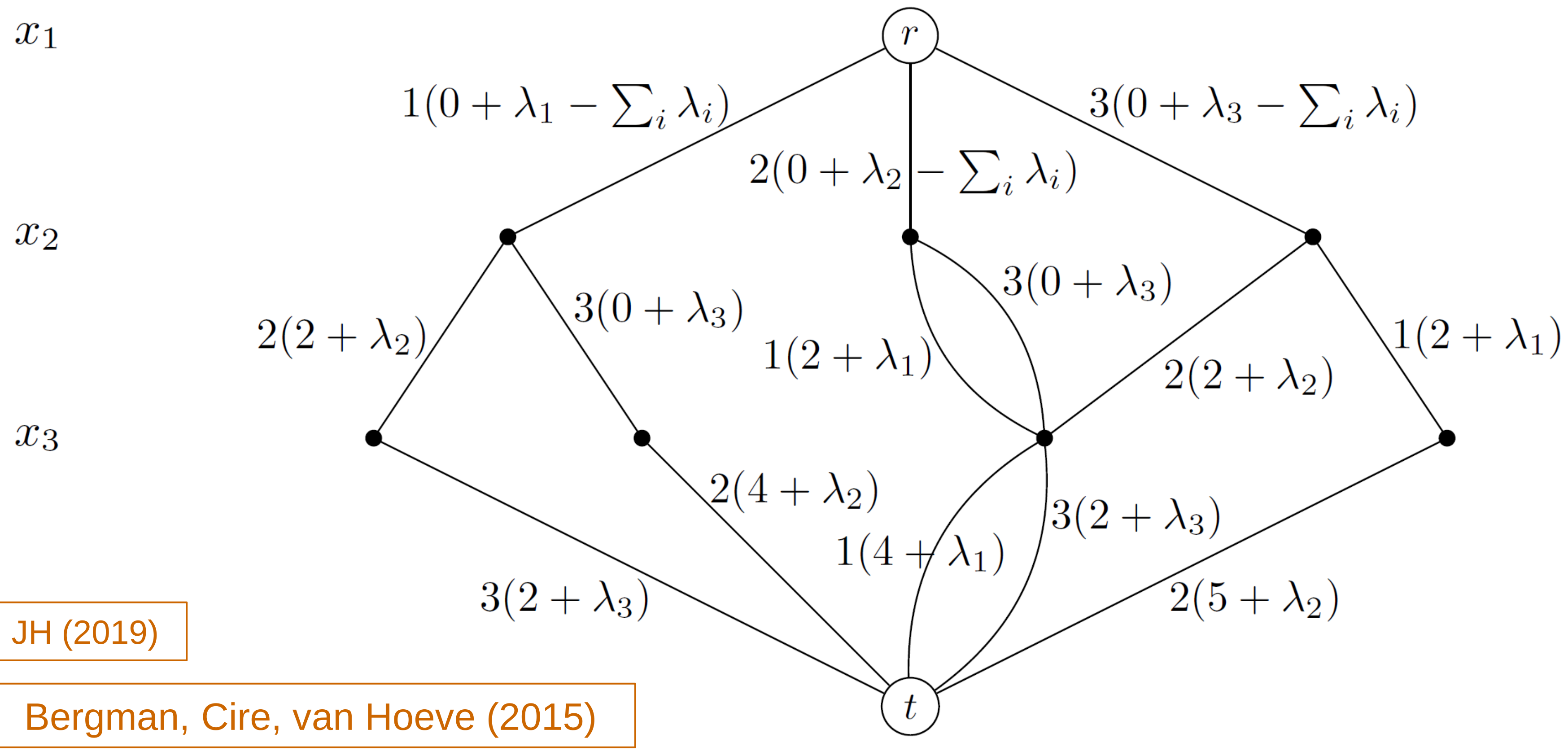


Sufficient conditions for a state merger rule that yields a valid relaxed DD given in JH (2017)

DD example: Job sequencing bounds

We can tighten bound by including **Lagrange penalties** on infeasible paths.

Path length now includes total Lagrange penalty



JH (2019)

Bergman, Cire, van Hoeve (2015)

DD example: Job sequencing bounds

Theorem. Lagrangian relaxation can be implemented in a relaxed DD **if** nodes are merged **only when** their states **agree** on the values of the state variables on which the arc costs and Lagrangian arc penalties depend.

- Applies to **dynamic programming in general**.
- Useful when immediate cost and penalty functions depend on **only a few state variables**.

JH (2019)

DD example: Job sequencing bounds

- For which problems is Lagrangian + DD relaxation practical, based on the theorem?
  - Min **tardiness**.* 
  - Min tardiness **+ earliness**.* 
  - Min tardiness with **time-dependent** processing times. 
  - Min tardiness with **state-dependent** processing times. 
  - TSP **without** time windows. 
  - TSP **with** time windows. 

* Computational tests to follow…

DD example: Job sequencing bounds

Computational tests.

- Min **tardiness**
  - Crauwells-Potts-Wassenhove instances.
  - Provably optimal solutions **known** for **most** instances.
  - Compare DD bound with known **optimal** values.
- Min **tardiness + earliness**
  - Biskup-Feldman instances.
  - Provably optimal solutions previously **unknown** for **all** instances.
  - Compare DD bound with **best** solutions known.

DD example: Job sequencing bounds

Min tardiness, 50 jobs

	50 jobs			
Instance	Target	Bound	Gap	Percent gap
1	2134	2100	34	1.59%
2	1996	1864	132	6.61%
3	2583	2552	31	1.20%
4	2691	2673	18	0.67%
5	1518	1342	176	11.59%
6	26276	26054	222	0.84%
7	11403	11128	275	2.41%
8	8499	8490	9	0.11%
9	9884	9507	377	3.81%
10	10655	10594	61	0.57%
11	*43504	43472	32	0.07%
12	*36378	36303	75	0.21%
13	45383	45310	73	0.16%

*Best known solution

	50 jobs			
Instance	Target	Bound	Gap	Percent gap
14	*51785	51702	83	0.16%
15	38934	38910	47	0.12%
16	87902	87512	390	0.44%
17	84260	84066	194	0.23%
18	104795	104633	162	0.15%
19	*89299	89163	136	0.15%
20	72316	72222	94	0.13%
21	214546	214476	70	0.03%
22	150800	150800	0	0%
23	224025	223922	103	0.05%
24	116015	115990	25	0.02%
25	240179	240172	7	0.003%

*Best known solution

JH (2019)

Time = about 40 minutes per instance

DD example: Job sequencing bounds

Min tardiness + earliness, 50 jobs

Instance	$(h_1, h_2) = (0.1, 0.2)$ Target	Bound	Gap	Percent gap
50 jobs				
1	39250	39250	0	0%
2	29043	29043	0	0%
3	33180	33180	0	0%
4	25856	25847	9	0.03%
5	31456	31439	17	0.05%
6	33452	33444	8	0.02%
7	42234	42228	6	0.01%
8	42218	42203	15	0.04%
9	33222	33218	4	0.01%
10	31492	31481	11	0.03%

Instance	$(h_1, h_2) = (0.2, 0.5)$ Target	Bound	Gap	Percent gap
50 jobs				
1	12754	12752	2	0.02%
2	8468	8463	5	0.06%
3	9935	9935	0	0%
4	7373	7335	38	0.52%
5	8947	8938	9	0.10%
6	10221	10213	8	0.08%
7	12002	11981	21	0.17%
8	11154	11141	13	0.12%
9	10968	10965	3	0.03%
10	9652	9650	3	0.03%

Time = about 8 minutes per instance

JH (2019)

DD example: Job sequencing bounds

Min tardiness + earliness, 100 jobs

Instance	$(h_1, h_2) = (0.1, 0.2)$ Target	Bound	Gap	Percent gap
100 jobs				
1	139573	139556	17	0.01%
2	120484	120465	19	0.02%
3	124325	124289	36	0.03%
4	122901	122876	25	0.02%
5	119115	119101	14	0.01%
6	133545	133536	9	0.007%
7	129849	129830	19	0.01%
8	153965	153958	7	0.005%
9	111474	111466	8	0.007%
10	112799	112792	7	0.006%

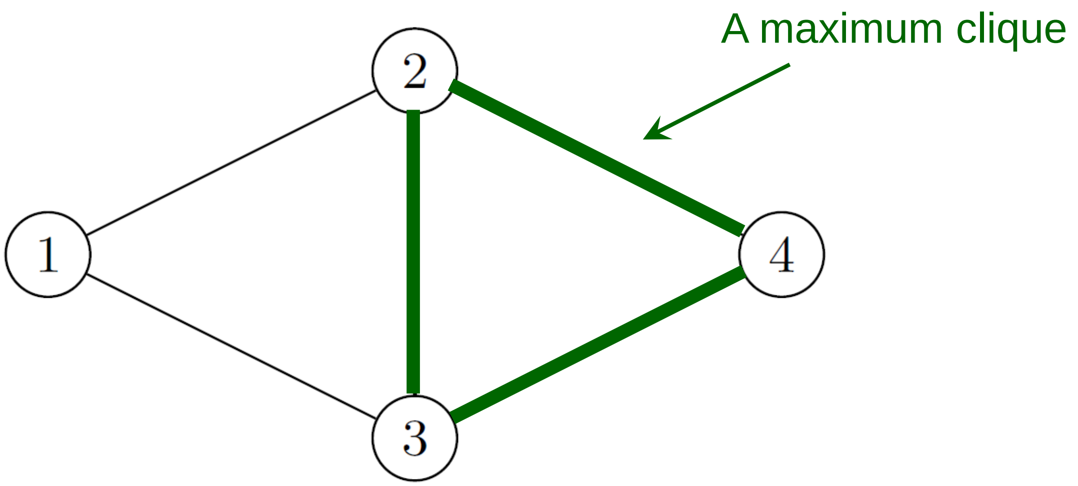
Instance	$(h_1, h_2) = (0.2, 0.5)$ Target	Bound	Gap	Percent gap
100 jobs				
1	39495	39467	28	0.07%
2	35293	35266	27	0.08%
3	38174	38150	24	0.06%
4	35498	35467	31	0.09%
5	34860	34826	34	0.10%
6	35146	35123	23	0.07%
7	39336	39303	33	0.08%
8	44963	44927	36	0.08%
9	31270	31231	39	0.12%
10	34068	34048	20	0.06%

Time = about 65 minutes per instance

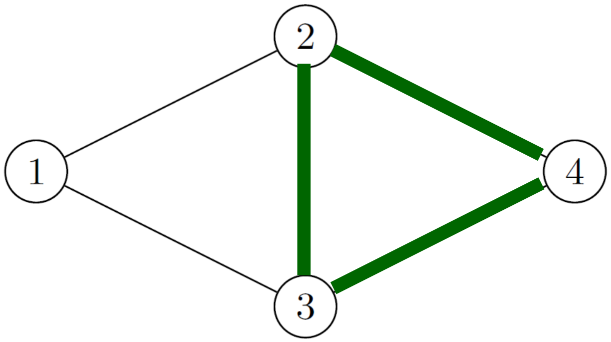
JH (2019)

Stochastic DD example: Max clique

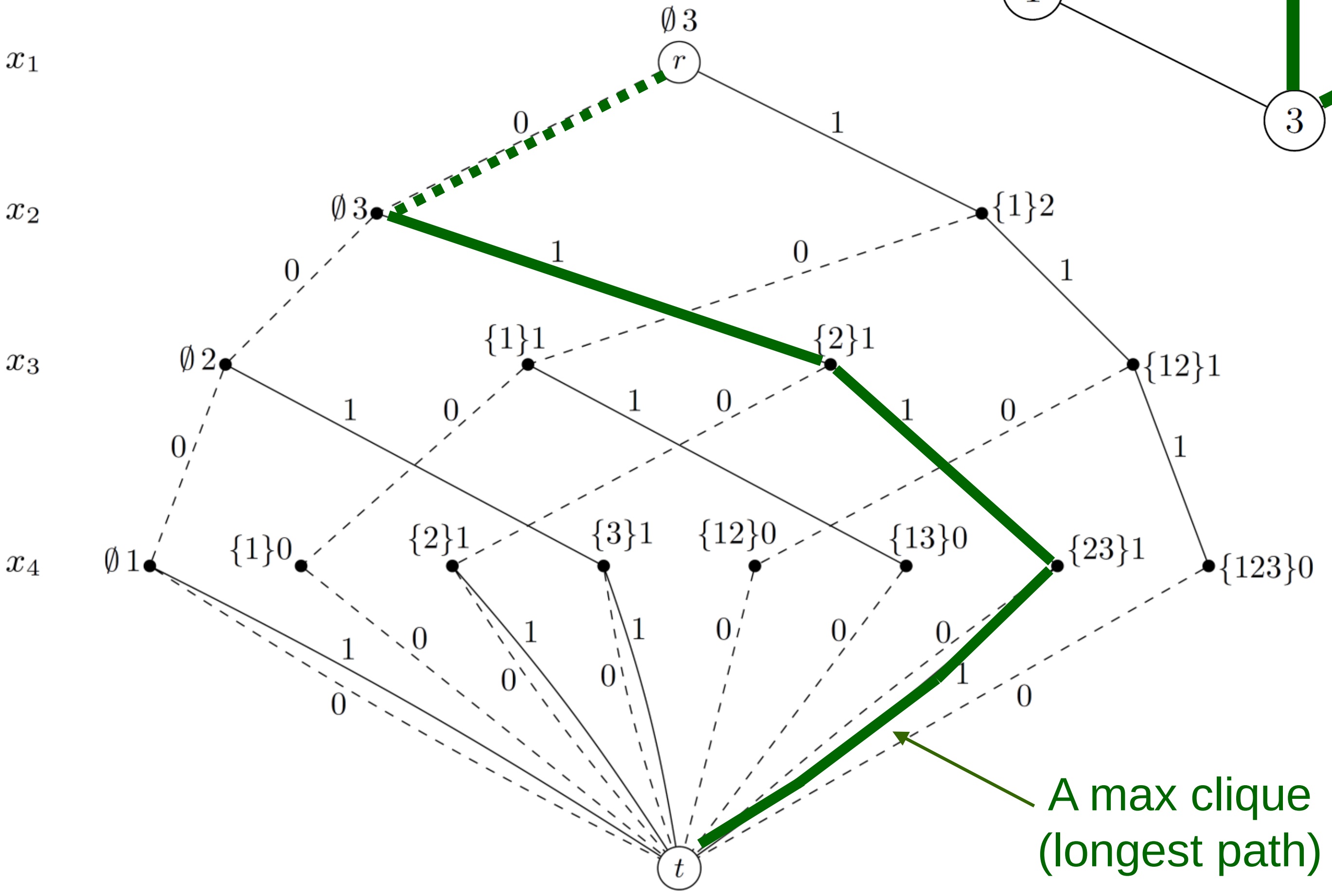
- Find clique in a graph with max expected size
  - Each edge occurs with probability 0.6.
  - Even small instances are intractable for exact solution.
- Find bound on max expected clique size
  - For solving stochastic **dynamic programming** models.
  - Requires relaxed **stochastic** DDs. JH (2022)



Stochastic DD example: Max clique



DD for deterministic problem



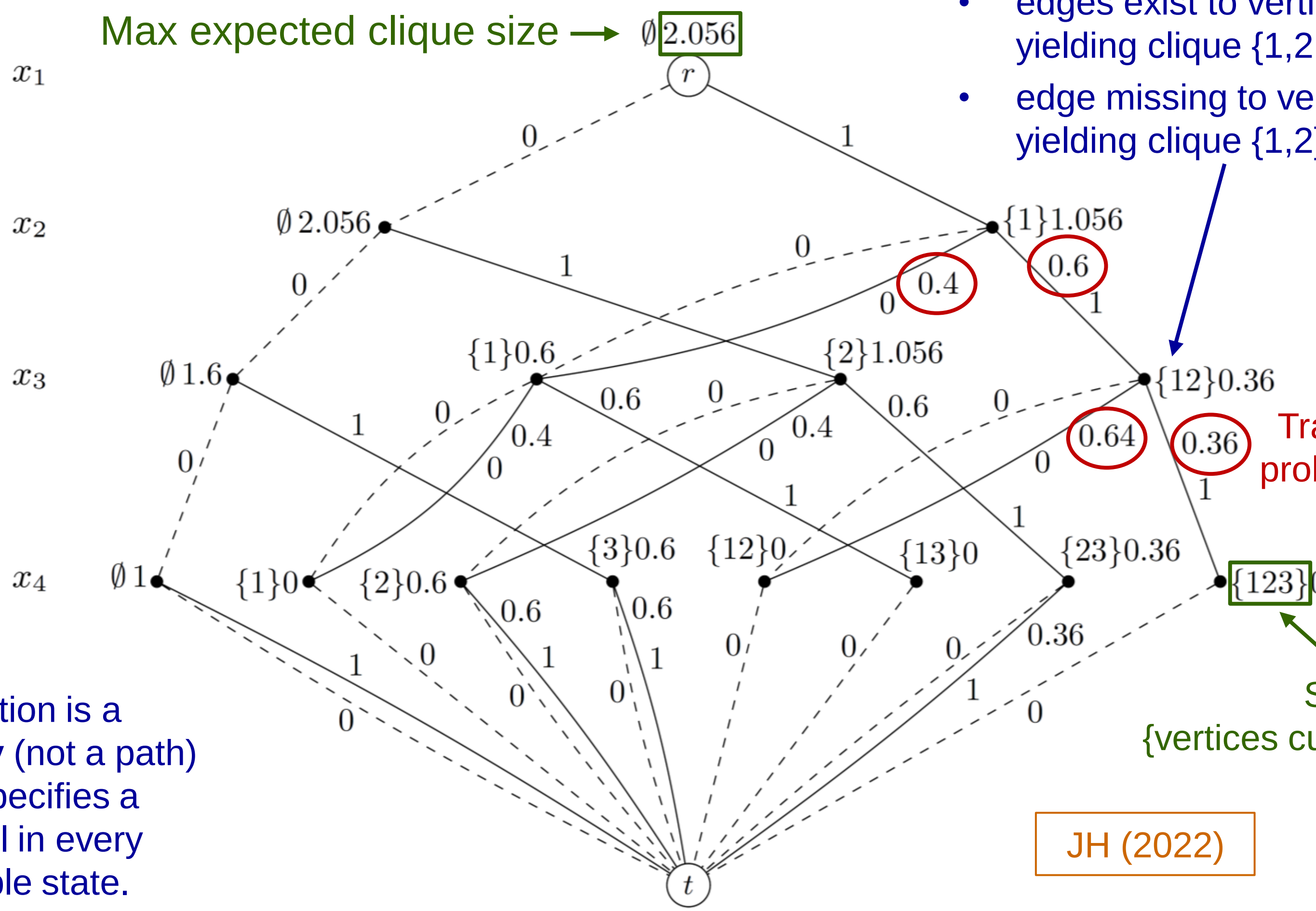
A max clique
(longest path)

Stochastic DD example: Max clique

DD for stochastic problem

Control x_3 = 1 (put vertex 3 in clique) has 2 possible outcomes:

- edges exist to vertices 1 and 2, yielding clique {1,2,3}
- edge missing to vertex 1 or 2, yielding clique {1,2}

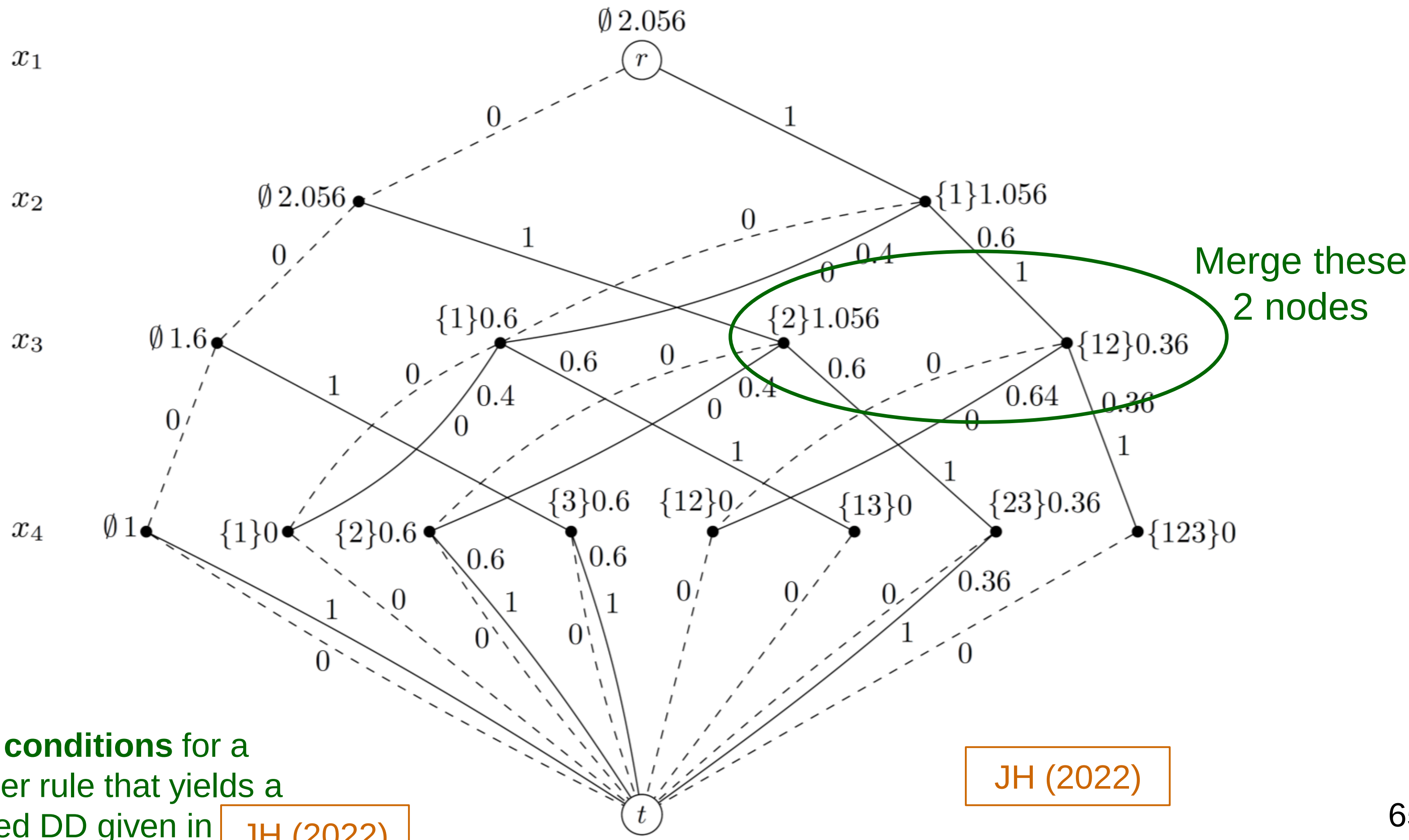


A solution is a **policy** (not a path) that specifies a control in every possible state.

JH (2022)

Stochastic DD example: Max clique

Relax DD by merging nodes



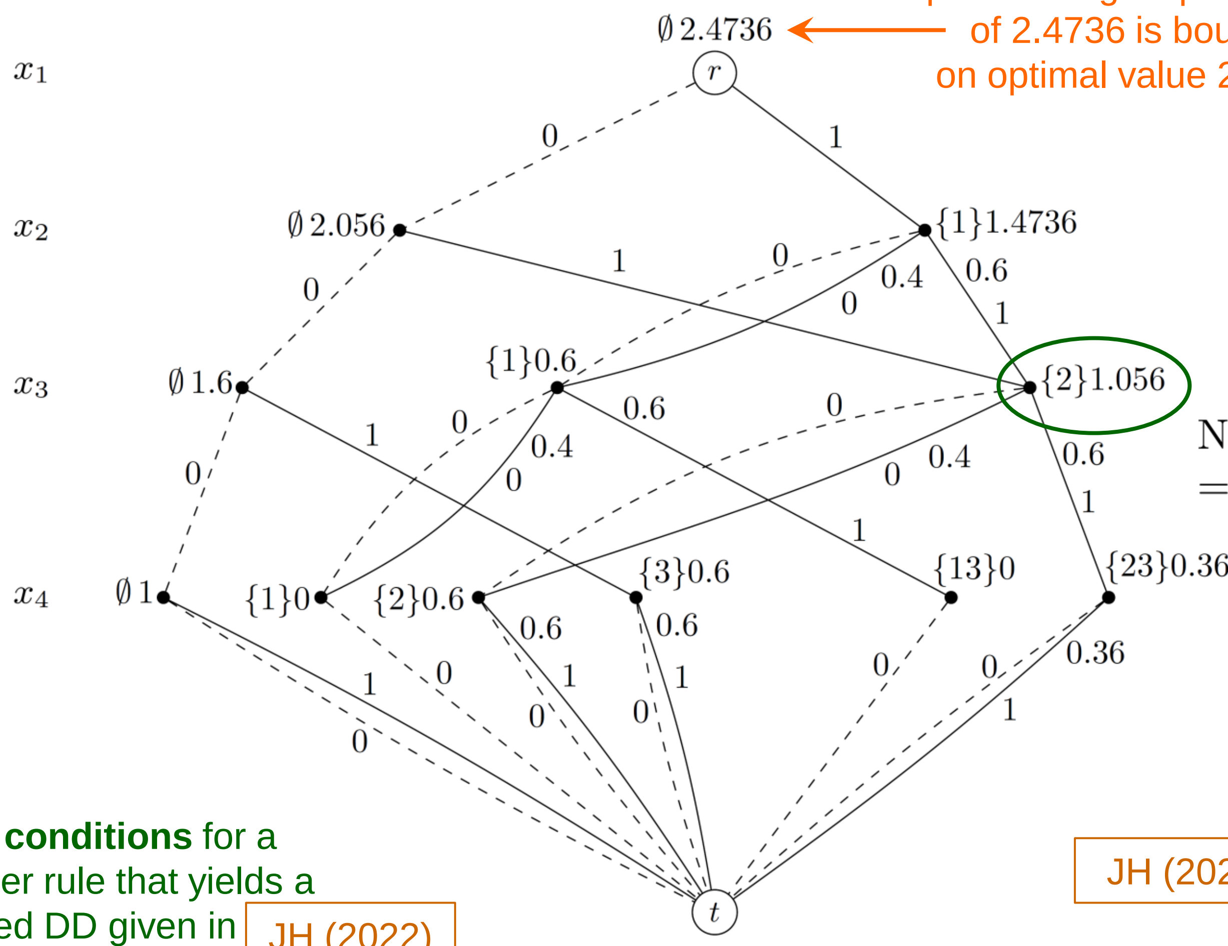
Sufficient conditions for a state merger rule that yields a valid relaxed DD given in JH (2022)

JH (2022)

Stochastic DD example: Max clique

Relax DD by merging nodes

Expected longest path length of 2.4736 is bound on optimal value 2.056



Result of merger

New state $\{2\} = \{2\} \cap \{1,2\}$

Sufficient conditions for a state merger rule that yields a valid relaxed DD given in JH (2022)

JH (2022)

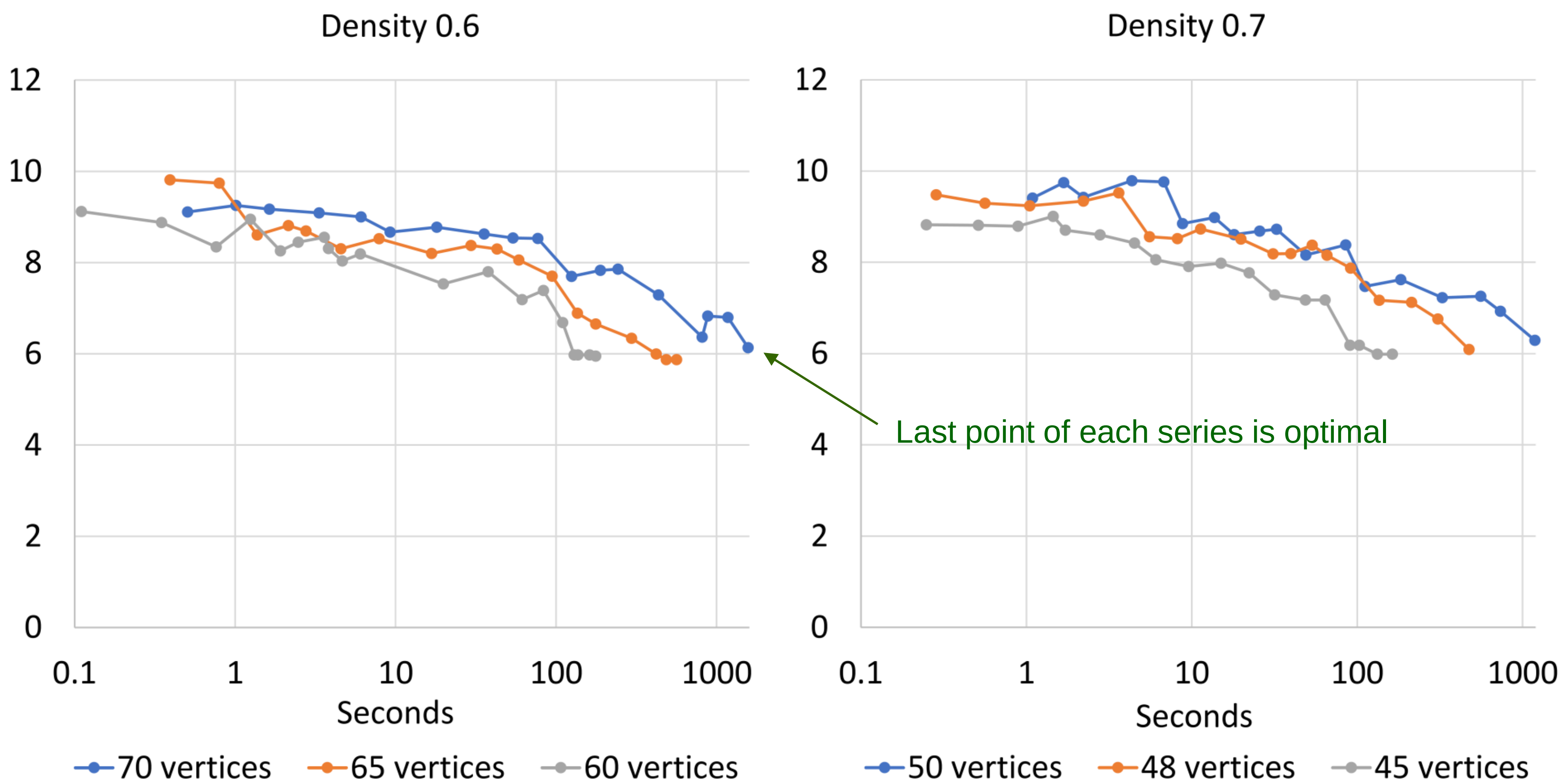
Stochastic DD example: Max clique

Computational tests.

- Basic issue
  - Need **exact** (or **very good**) solution to judge quality of bound.
  - Nearly all nontrivial instances are **intractable**.
- Random instances
  - Choose parameters that allow solution to proven optimality.
  - Measure **quality of bound** against time required to process DDs of **increasing width**.
- DIMACS instances + edge probabilities
  - **Only 2** could be solved to optimality, one requiring 24 hours.
  - Take others up to 1000 seconds.
- Results
  - Bound quality **degrades slowly** as exact DD is relaxed.
  - Gap varies roughly with **logarithm** of time investment

Stochastic DD example: Max clique

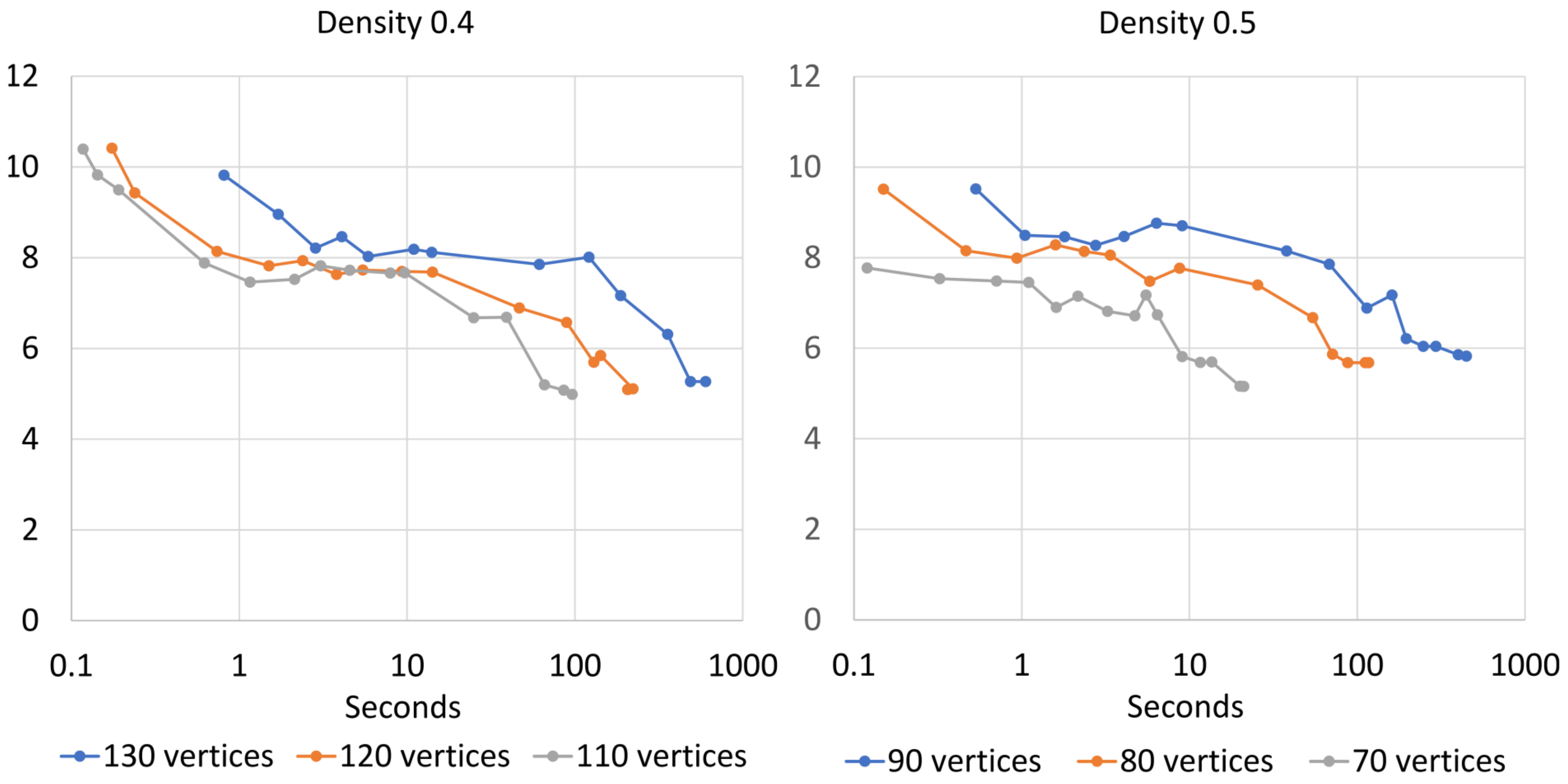
Random instances (solved to optimality)



JH (2022)

Stochastic DD example: Max clique

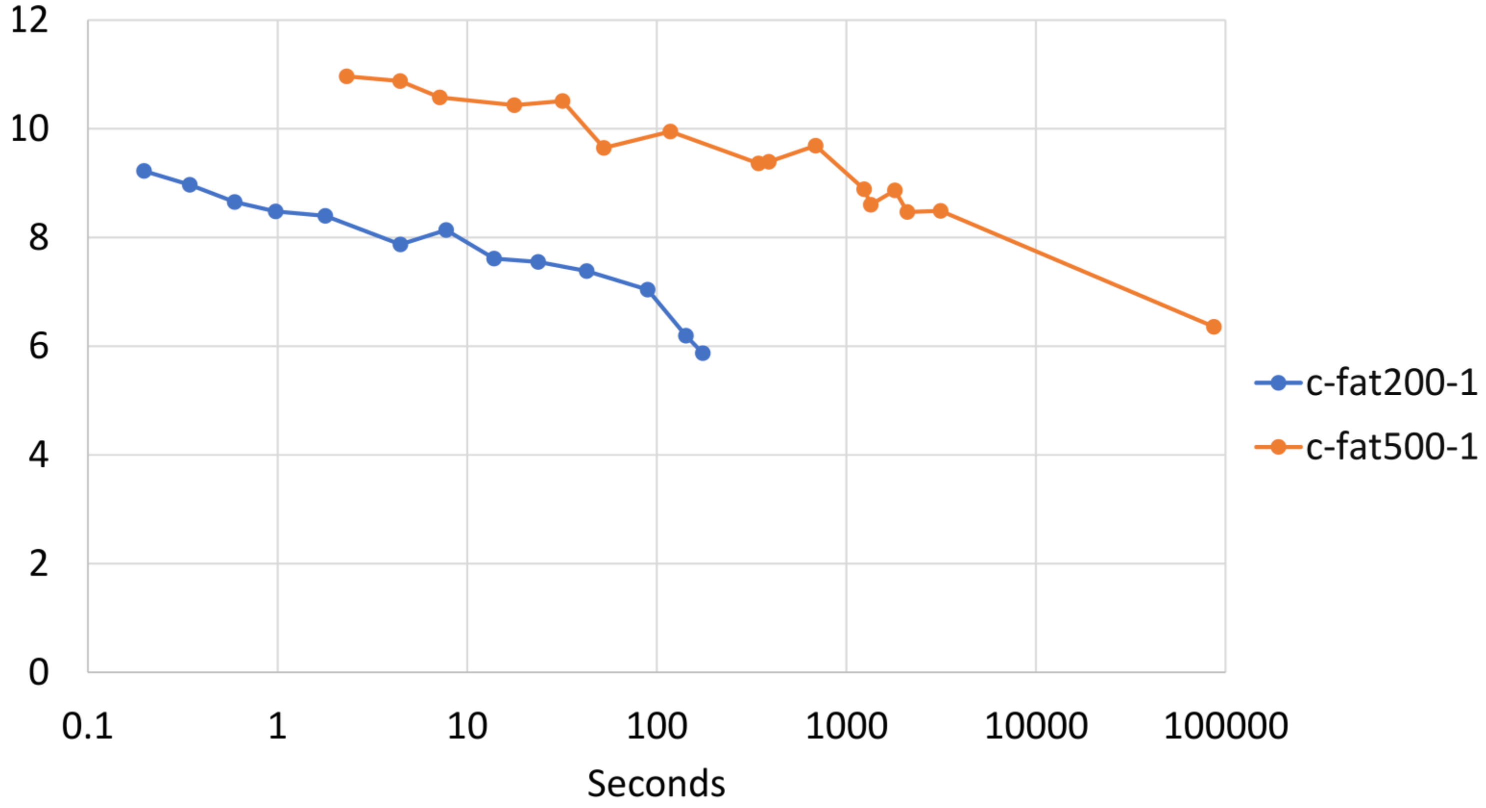
Random instances (solved to optimality)



JH (2022)

Stochastic DD example: Max clique

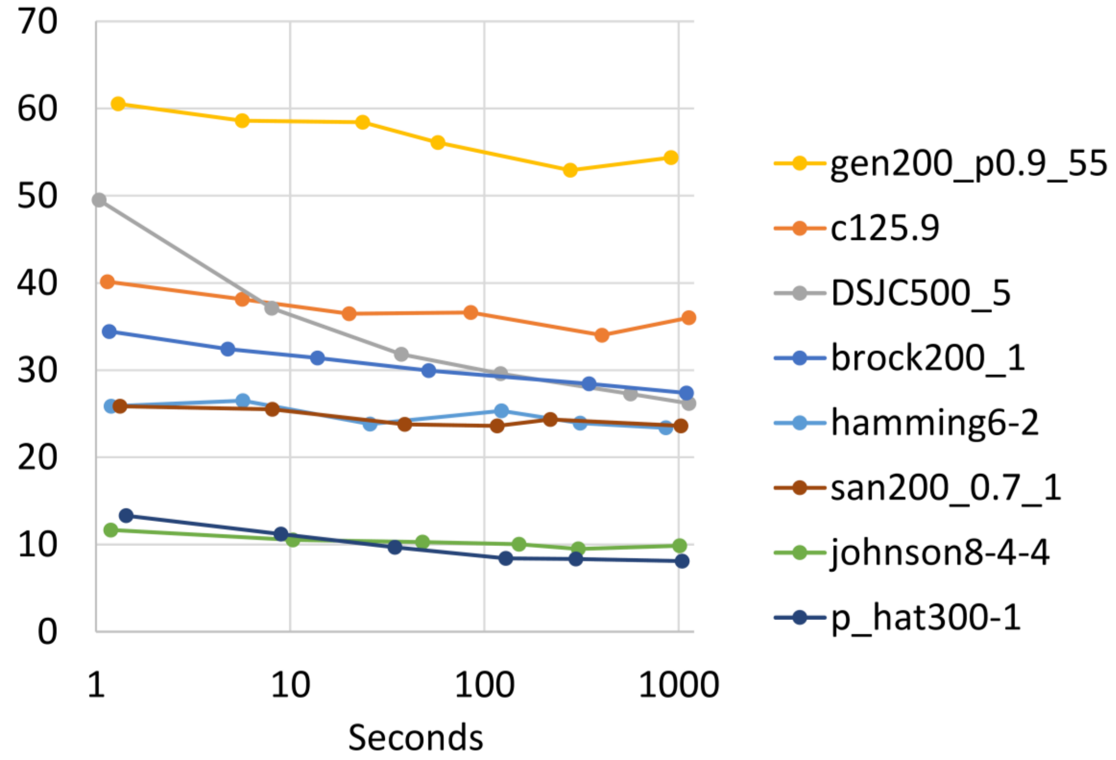
2 DIMACS instances (solved to optimality)



JH (2022)

Stochastic DD example: Max clique

DIMACS instances (not solved to optimality)



JH (2022)

- Conclusion
  - Bound quality **degrades slowly** as exact DD is relaxed.
  - Gap varies roughly with **logarithm** of time investment

Software

- General CP/opt integration
  - **IBM ILOG CPLEX** Optimizer
  - **MiniZinc** modeling language (open source) for cooperating solvers
  - **SCIP** (open source)
  - **BARON** (global optimization)
- Constraint programming solvers
  - **IBM ILOG CPLEX** Optimizer
  - **Gecode** (open source)
  - **Chuffed** (open source)
  - **Google OR Tools** CP solver and CP-SAT solver (open source)
- Logic-based Benders
  - Automatic LBBD in **MiniZinc** (open source)
  - **Nutmeg** (branch and check, open source)
- Decision diagrams
  - **DDO** (open source)
  - **Haddock** (CP + DDs, open source)
  - **Hop** (developed by nextmv for logistics)


THE
END