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

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

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**.

```prolog
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**.

```prolog
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):

\[
\begin{align*}
\text{all-different}(w[*], d), & \quad d = 1, \ldots, 7 \\
\text{cardinality}(w[*], (A, B, C, D), 5, 6) \\
\text{nvalues}(w[s, *], 1, 2), & \quad s = 1, 2, 3 \\
w[s, d] \in \{A, B, C, D\}, & \quad \text{all } s, d \\
w[s, d] = \text{worker assigned to shift } s \text{ on day } d
\end{align*}
\]

- 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 \(w[s, d]\)

**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):

\[
\sum_{i} x_{isd} = 1, \text{ all } s, d; \quad \sum_{s} x_{isd} \leq 1, \text{ all } i, d
\]

\[
5 \leq \sum_{s,d} x_{isd} \leq 6, \text{ all } i
\]

\[
\sum_{i} y_{is} \leq 2, \text{ all } s; \quad \sum_{d} x_{isd} \leq 7y_{is}, \text{ all } i, s
\]

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

\[
x_{isd} = 1 \text{ if worker } i \text{ assigned to shift } s \text{ on day } d
\]
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.

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

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.

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

  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:

\[
\text{cumulative}((s_1, \ldots, s_n), (p_1, \ldots, p_n), (c_1, \ldots, c_n), C)
\]

- 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:

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

<table>
<thead>
<tr>
<th>Job</th>
<th>( p_j )</th>
<th>( c_j )</th>
<th>([E_j, L_j])</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>[0, 5]</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>[0, 5]</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
<td>[1, 7]</td>
</tr>
</tbody>
</table>

*Domain of \( s_j \) is \([E_j, L_j−p_j]\)
Cumulative scheduling

We can deduce that job 3 must finish last.

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

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

Total energy required = 22
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:

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

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:**

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

Energy available for jobs 1 & 2, which require \(9 + 5 = 14\)
Cumulative scheduling

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

**Additional energy** required by jobs 1 & 2:

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

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
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

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

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

<table>
<thead>
<tr>
<th>CP</th>
<th>Traditional Opt</th>
</tr>
</thead>
</table>
| • 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. | • 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.


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
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

\[
\begin{align*}
\min & \quad f(x, y) \\
(x, y) & \in S \\
x & \in D_x, \quad y \in D_y
\end{align*}
\]

Master problem

\[
\begin{align*}
\min & \quad z \\
z & \geq g_k(x), \text{ all cuts } k \\
x & \in D_x
\end{align*}
\]

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

Subproblem

\[
\begin{align*}
\min & \quad f(\bar{x}, y) \\
(\bar{x}, y) & \in S \\
y & \in D_y
\end{align*}
\]

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

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)
LBBBD example: Home healthcare

**Master problem**

Assign patients to healthcare aides and days of the week

\[
\begin{align*}
\text{max} & \quad \sum_j \delta_j \\
\sum_i x_{ij} &= \delta_j \quad \text{all } j \\
\sum_{i,k} y_{ijk} &= v_j \delta_j \quad \text{all } j \\
y_{ijk} &\leq x_{ij} \quad \text{all } i, j, k
\end{align*}
\]

= 1 if patient \( j \) scheduled

Required number of visits per week

= 1 if patient \( j \) assigned to aide \( i \) on day \( k \)

= 1 if patient \( j \) assigned to aide \( i \)

Spacing constraints on visit days

Benders cuts

Relaxation of subproblem

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

**MILP model**
**LBBD example: Home healthcare**

**Subproblem**

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

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

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

\[
s_{\pi_{k,\nu}} + p_{\pi_{k,\nu}} + t_{\pi_{k,\nu}, \pi_{k,\nu+1}} \leq s_{\pi_{k,\nu+1}}, \text{ all } k, \nu
\]

**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.

$$\sum_{j \in P_{i,k}} (1 - y_{i,j,k}) \geq 1$$

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.

LBBD example: Home healthcare

Computational results

Data from home hospice care firm.

Better results for slightly easier instances in Grenouilleau, Lahrichi, Rousseau (2020)
**LBBD example: Home healthcare**

**Computational results**

Data from Danish home care agency.

Heching, JH, Kimura (2019)

<table>
<thead>
<tr>
<th>Instance</th>
<th>Patients</th>
<th>Crews</th>
<th>Weighted objective</th>
<th>Covering objective</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>MILP</td>
<td>LBBD</td>
</tr>
<tr>
<td>hh</td>
<td>30</td>
<td>15</td>
<td>*</td>
<td>3.16</td>
</tr>
<tr>
<td>ll1</td>
<td>30</td>
<td>8</td>
<td>*</td>
<td>1.74</td>
</tr>
<tr>
<td>ll2</td>
<td>30</td>
<td>7</td>
<td>2868</td>
<td>1.56</td>
</tr>
<tr>
<td>ll3</td>
<td>30</td>
<td>6</td>
<td>1398</td>
<td>2.16</td>
</tr>
</tbody>
</table>

*Computation time exceeded one hour.*
LBBB 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)
LBBDD example: Multiple machine scheduling

Performance profile for 50 problem instances

Ciré, Coban, JH (2015)
LBBB 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.
LBBM example: Stochastic machine scheduling

Computation time

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

Each time (seconds) is average over 3 instances

<table>
<thead>
<tr>
<th>Scenarios</th>
<th>Integer L-shaped</th>
<th>Branch &amp; Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>127</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>839</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>2317</td>
<td>3</td>
</tr>
<tr>
<td>50</td>
<td>&gt; 3600</td>
<td>17</td>
</tr>
<tr>
<td>100</td>
<td>&gt; 3600</td>
<td>37</td>
</tr>
<tr>
<td>500</td>
<td>&gt; 3600</td>
<td>279</td>
</tr>
</tbody>
</table>

Elçi and JH (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)]

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.

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

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.

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

- **A new perspective on optimization**
  - DDs can perform all functions of an optimization solver...

---

Decision diagrams

Modeling with recursive formulations

Relaxation with relaxed diagrams

Search with a novel branch-and-bound method

Primal heuristics with restricted diagrams

Postoptimality analysis with sound diagrams

Constraint propagation through a relaxed diagram

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.

<table>
<thead>
<tr>
<th>Job</th>
<th>Release time</th>
<th>Processing time</th>
<th>Due date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>
DD example: Job sequencing bounds

Decision diagram for job sequencing

Each \( r-t \) path corresponds to a feasible solution
**DD example: Job sequencing bounds**

Decision diagram for job sequencing

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

An optimal solution:
Sequence 2-3-1
Tardiness $0 + 0 + 4 = 4$

$x_j = j$th job in sequence

<table>
<thead>
<tr>
<th>$j$</th>
<th>$r_j$</th>
<th>$p_j$</th>
<th>$d_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>
Interpret DD as dynamic programming
state transition graph

DD example: Job sequencing bounds

State variable:
- jobs scheduled so far
- finish time of last job

Cost to go
- Minimum tardiness

<table>
<thead>
<tr>
<th>j</th>
<th>r_j</th>
<th>p_j</th>
<th>d_j</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>
DD example: Job sequencing bounds

Interpret DD as dynamic programming state transition graph

State variable:
- jobs scheduled so far
- finish time of last job

Cost to go
- Minimum tardiness

State transitions:
- $x_j$ for each job $j$

Table:

<table>
<thead>
<tr>
<th>$j$</th>
<th>$r_j$</th>
<th>$p_j$</th>
<th>$d_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>
DD example: Job sequencing bounds

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

Relax DD by **merging** some nodes

<table>
<thead>
<tr>
<th>j</th>
<th>r_j</th>
<th>p_j</th>
<th>d_j</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Example: merge these nodes

Andersen, Hadžić, JH, Tiedemann (2007)

Ciré and van Hoeve (2013)

Bergman, Ciré, van Hoeve, JH (2013)
DD example: Job sequencing bounds

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

<table>
<thead>
<tr>
<th>$j$</th>
<th>$r_j$</th>
<th>$p_j$</th>
<th>$d_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Relax DD by **merging** some nodes

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

State variable: Jobs scheduled along all paths from root

Andersen, Hadžić, JH, Tiedemann (2007)

Ciré and van Hoeve (2013)

Bergman, Ciré, van Hoeve, JH (2013)
Exact DD grows exponentially. Obtain **lower bound** on tardiness from smaller **relaxed DD**

Relax DD by **merging** some nodes

**DD example: Job sequencing bounds**

Andersen, Hadžić, JH, Tiedemann (2007)

Ciré and van Hoeve (2013)

Bergman, Ciré, van Hoeve, JH (2013)

<table>
<thead>
<tr>
<th>j</th>
<th>r_j</th>
<th>p_j</th>
<th>d_j</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>
DD example: Job sequencing bounds

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).

<table>
<thead>
<tr>
<th>$j$</th>
<th>$r_j$</th>
<th>$p_j$</th>
<th>$d_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Shortest path yields a lower bound of 2 on optimal value of 4.
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 \texttt{tardiness}.*
  – Min tardiness + \texttt{earliness}.*
  – Min tardiness with \texttt{time-dependent} processing times.
  – Min tardiness with \texttt{state-dependent} processing times.
  – TSP \textbf{without} time windows.
  – TSP \textbf{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

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

*Best known solution

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

*Best known solution

Time = about 40 minutes per instance

JH (2019)
DD example: Job sequencing bounds

Min tardiness + earliness, 50 jobs

<table>
<thead>
<tr>
<th>Instance</th>
<th>Target</th>
<th>Bound</th>
<th>Gap</th>
<th>Percent gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>39250</td>
<td>39250</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>2</td>
<td>29043</td>
<td>29043</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>3</td>
<td>33180</td>
<td>33180</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>4</td>
<td>25856</td>
<td>25847</td>
<td>9</td>
<td>0.03%</td>
</tr>
<tr>
<td>5</td>
<td>31456</td>
<td>31439</td>
<td>17</td>
<td>0.05%</td>
</tr>
<tr>
<td>6</td>
<td>33452</td>
<td>33444</td>
<td>8</td>
<td>0.02%</td>
</tr>
<tr>
<td>7</td>
<td>42234</td>
<td>42228</td>
<td>6</td>
<td>0.01%</td>
</tr>
<tr>
<td>8</td>
<td>42218</td>
<td>42203</td>
<td>15</td>
<td>0.04%</td>
</tr>
<tr>
<td>9</td>
<td>33222</td>
<td>33218</td>
<td>4</td>
<td>0.01%</td>
</tr>
<tr>
<td>10</td>
<td>31492</td>
<td>31481</td>
<td>11</td>
<td>0.03%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Instance</th>
<th>Target</th>
<th>Bound</th>
<th>Gap</th>
<th>Percent gap</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12754</td>
<td>12752</td>
<td>2</td>
<td>0.02%</td>
</tr>
<tr>
<td>2</td>
<td>8468</td>
<td>8463</td>
<td>5</td>
<td>0.06%</td>
</tr>
<tr>
<td>3</td>
<td>9935</td>
<td>9935</td>
<td>0</td>
<td>0%</td>
</tr>
<tr>
<td>4</td>
<td>7373</td>
<td>7335</td>
<td>38</td>
<td>0.52%</td>
</tr>
<tr>
<td>5</td>
<td>8947</td>
<td>8938</td>
<td>9</td>
<td>0.10%</td>
</tr>
<tr>
<td>6</td>
<td>10221</td>
<td>10213</td>
<td>8</td>
<td>0.08%</td>
</tr>
<tr>
<td>7</td>
<td>12002</td>
<td>11981</td>
<td>21</td>
<td>0.17%</td>
</tr>
<tr>
<td>8</td>
<td>11154</td>
<td>11141</td>
<td>13</td>
<td>0.12%</td>
</tr>
<tr>
<td>9</td>
<td>10968</td>
<td>10965</td>
<td>3</td>
<td>0.03%</td>
</tr>
<tr>
<td>10</td>
<td>9652</td>
<td>9650</td>
<td>3</td>
<td>0.03%</td>
</tr>
</tbody>
</table>

Time = about 8 minutes per instance

JH (2019)
### DD example: Job sequencing bounds

Min tardiness + earliness, 100 jobs

<table>
<thead>
<tr>
<th>Instance</th>
<th>((h_1, h_2) = (0.1, 0.2))</th>
<th></th>
<th>((h_1, h_2) = (0.2, 0.5))</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Target</td>
<td>Bound</td>
<td>Gap</td>
<td>Percent gap</td>
</tr>
<tr>
<td>100 jobs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>139573</td>
<td>139556</td>
<td>17</td>
<td>0.01%</td>
</tr>
<tr>
<td>2</td>
<td>120484</td>
<td>120465</td>
<td>19</td>
<td>0.02%</td>
</tr>
<tr>
<td>3</td>
<td>124325</td>
<td>124289</td>
<td>36</td>
<td>0.03%</td>
</tr>
<tr>
<td>4</td>
<td>122901</td>
<td>122876</td>
<td>25</td>
<td>0.02%</td>
</tr>
<tr>
<td>5</td>
<td>119115</td>
<td>119101</td>
<td>14</td>
<td>0.01%</td>
</tr>
<tr>
<td>6</td>
<td>133545</td>
<td>133536</td>
<td>9</td>
<td>0.007%</td>
</tr>
<tr>
<td>7</td>
<td>129849</td>
<td>129830</td>
<td>19</td>
<td>0.01%</td>
</tr>
<tr>
<td>8</td>
<td>153965</td>
<td>153958</td>
<td>7</td>
<td>0.005%</td>
</tr>
<tr>
<td>9</td>
<td>111474</td>
<td>111466</td>
<td>8</td>
<td>0.007%</td>
</tr>
<tr>
<td>10</td>
<td>112799</td>
<td>112792</td>
<td>7</td>
<td>0.006%</td>
</tr>
</tbody>
</table>

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.

A maximum clique

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

Max expected clique size → \( 2.056 \)

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\}

State = \{vertices currently in clique\}

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

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

Last point of each series is optimal
Stochastic DD example: Max clique

Random instances (solved to optimality)
Stochastic DD example: Max clique

2 DIMACS instances (solved to optimality)
Stochastic DD example: Max clique

DIMACS instances (not solved to optimality)

- 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