Recent Developments in Integrated Methods for Optimization

John Hooker
Carnegie Mellon University

EWO Seminar
August 2012
Premise

• Constraint programming and mathematical programming can work together.
  • There is underlying unity.
  • Result: faster solution and more elegant models.
Premise

• Constraint programming and mathematical programming can work together.
  • There is underlying unity.
  • Result: faster solution and more elegant models.
• Math programming solvers constantly improve…
  • But they would reach a much higher level if this same effort were applied to integrated methods.
  • Solvers are beginning to move in this direction.
Premise

• Constraint programming and mathematical programming can work together.
  • There is underlying unity.
  • Result: faster solution and more elegant models.
• Math programming solvers constantly improve…
  • But they would reach a much higher level if this same effort were applied to integrated methods.
  • Solvers are beginning to move in this direction.
• The same integration principles apply more generally.
  • Global optimization, exact/heuristic methods.
Outline

- What is constraint programming?
- Integrating OR and CP
  - Constraint propagation + relaxation
  - CP-based branch and price
  - Decomposition Methods
  - Software
- Some very recent work

Caveat: This is a high-level overview. Don’t worry about the technical details.
What Is Constraint Programming?

Basic Idea
Applications
Examples
CP and Math Programming
Software
Basic Idea

• Each line of the model is both a constraint and a procedure.
  
  – **Constraint**: often a high-level global constraint
    – Different modeling paradigm than math programming
  
  – **Procedure**: removes infeasible values from variable domains
    – Filtering, domain consistency maintenance
    – Passes reduced domains to next constraint (constraint propagation).
Early commercial successes

- Circuit design (Siemens)
- Container port scheduling (Hong Kong and Singapore)
- Real-time control (Siemens, Xerox)
Applications

• Job shop scheduling
• Assembly line smoothing and balancing
• Cellular frequency assignment
• Nurse scheduling
• Shift planning
• Maintenance planning
• Airline crew rostering and scheduling
• Airport gate allocation and stand planning
Applications

- Production scheduling
  chemicals
  aviation
  oil refining
  steel
  lumber
  photographic plates
  tires

- Transport scheduling (food, nuclear fuel)

- Warehouse management

- Course timetabling
Example: Employee scheduling

• Schedule four nurses in 8-hour shifts.
• A nurse works at most one shift a day, at least 5 days a week.
• Same schedule every week.
• No shift staffed by more than two different nurses in a week.
• A nurse cannot work different shifts on two consecutive days.
• A nurse who works shift 2 or 3 must do so at least two days in a row.
Two ways to view the problem

Assign nurses to shifts

<table>
<thead>
<tr>
<th></th>
<th>Sun</th>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift 1</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Shift 2</td>
<td>C</td>
<td>C</td>
<td>C</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>Shift 3</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>C</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

Assign shifts to nurses

<table>
<thead>
<tr>
<th></th>
<th>Sun</th>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nurse A</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Nurse B</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Nurse C</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>Nurse D</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

0 = day off
Use **both** formulations in the same model!

First, assign nurses to shifts.

Let $w_{sd} =$ nurse assigned to shift $s$ on day $d$

$$\text{alldiff}(w_{1d}, w_{2d}, w_{3d}), \text{ all } d$$

Schedule 3 different nurses on each day
Use **both** formulations in the same model!

First, assign nurses to shifts.

Let $w_{sd} = \text{nurse assigned to shift } s \text{ on day } d$

\[
\text{alldiff}(w_{1d}, w_{2d}, w_{3d}), \text{ all } d
\]

\[
\text{cardinality}(w \mid (A, B, C, D), (5, 5, 5, 5), (6, 6, 6, 6))
\]

Each nurse works at least 5 and at most 6 days a week
Use both formulations in the same model!

First, assign nurses to shifts.

Let $w_{sd} =$ nurse assigned to shift $s$ on day $d$

\[
\text{alldiff}(w_{1d}, w_{2d}, w_{3d}), \quad \text{all } d
\]
\[
\text{cardinality}(w \mid (A, B, C, D), (5,5,5,5), (6,6,6,6))
\]
\[
\text{nvalues}(w_{s,Sun}, \ldots, w_{s,Sat} \mid 1,2), \quad \text{all } s
\]

At least 1 and at most 2 nurses work any given shift.
Remaining constraints are not easily expressed in this notation.

So, assign shifts to nurses.

Let $y_{id}$ = shift assigned to nurse $i$ on day $d$

\[ \text{alldiff} \left( y_{1d}, y_{2d}, y_{3d} \right), \text{ all } d \]

Assign a different nurse to each shift on each day.

Redundant, but speeds solution.
Remaining constraints are not easily expressed in this notation.

So, assign shifts to nurses.

Let $y_{id} =$ shift assigned to nurse $i$ on day $d$

\begin{align*}
\text{alldiff}(y_{1d}, y_{2d}, y_{3d}), & \quad \text{all } d \\
\text{stretch}(y_{i,\text{Sun}}, \ldots, y_{i,\text{Sat}} \mid (2,3), (2,2), (6,6), P), & \quad \text{all } i
\end{align*}

Shift 2 or 3 must be worked at least 2 days in a row.
Remaining constraints are not easily expressed in this notation.

So, assign shifts to nurses.

Let $y_{id}$ = shift assigned to nurse $i$ on day $d$

$$\text{alldiff}(y_{1d}, y_{2d}, y_{3d}), \quad \text{all } d$$

$$\text{stretch}(y_{i,\text{Sun}}, \ldots, y_{i,\text{Sat}} | (2,3),(2,2),(6,6), P), \quad \text{all } i$$

Pattern constraint: $P = \{(s,0),(0,s) | s = 1,2,3\}$

Don’t switch shifts without taking at least one day off.
Connect the $w_{sd}$ variables to the $y_{id}$ variables.

Use **channeling constraints**:

$$w_{y_{id}d} = i, \text{ all } i,d$$

$$y_{w_{sd}d} = s, \text{ all } s,d$$

Channeling constraints increase propagation and make the problem easier to solve.
The complete model is:

\[
\text{alldiff}(w_{1d}, w_{2d}, w_{3d}), \quad \text{all } d
\]

\[
\text{cardinality}(w \mid (A, B, C, D), (5, 5, 5, 5), (6, 6, 6, 6))
\]

\[
\text{nvalues}(w_{s, \text{Sun}}, \ldots, w_{s, \text{Sat}} \mid 1, 2), \quad \text{all } s
\]

\[
\text{alldiff}(y_{1d}, y_{2d}, y_{3d}), \quad \text{all } d
\]

\[
\text{stretch}(y_{i, \text{Sun}}, \ldots, y_{i, \text{Sat}} \mid (2, 3), (2, 2), (6, 6), P), \quad \text{all } i
\]

\[
w_{y_{id}d} = i, \quad \text{all } i, d
\]

\[
y_{w_{sd}d} = s, \quad \text{all } s, d
\]
Example: Resource-Constrained Scheduling

- Use the **cumulative scheduling** constraint.
- Total resources consumed by jobs at any one time must not exceed \( L \).

\[
\text{cumulative}((t_1, \ldots, t_n), (p_1, \ldots, p_n), (c_1, \ldots, c_n), C)
\]

- Job start times (variables)
- Job processing times
- Job resource requirements
Ship loading

• The problem
  • Load 34 items on the ship in minimum time (min makespan)
  • Each item $i$ requires $p_i$ minutes and $c_i$ workers.
  • Total of 8 workers available.

• From IBM OPL Studio manual
Precedence constraints

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>→</td>
<td>2,4</td>
</tr>
<tr>
<td>2</td>
<td>→</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>→</td>
<td>5,7</td>
</tr>
<tr>
<td>4</td>
<td>→</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>→</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>→</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>→</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>→</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td>→</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>→</td>
<td>14</td>
</tr>
<tr>
<td>10</td>
<td>→</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>→</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>→</td>
<td>13</td>
</tr>
<tr>
<td>12</td>
<td>→</td>
<td>13</td>
</tr>
<tr>
<td>13</td>
<td>→</td>
<td>15,16</td>
</tr>
<tr>
<td>14</td>
<td>→</td>
<td>15</td>
</tr>
<tr>
<td>15</td>
<td>→</td>
<td>18</td>
</tr>
<tr>
<td>16</td>
<td>→</td>
<td>17</td>
</tr>
<tr>
<td>17</td>
<td>→</td>
<td>18</td>
</tr>
<tr>
<td>18</td>
<td>→</td>
<td>19</td>
</tr>
<tr>
<td>18</td>
<td>→</td>
<td>20,21</td>
</tr>
<tr>
<td>19</td>
<td>→</td>
<td>23</td>
</tr>
<tr>
<td>20</td>
<td>→</td>
<td>23</td>
</tr>
<tr>
<td>21</td>
<td>→</td>
<td>22</td>
</tr>
<tr>
<td>22</td>
<td>→</td>
<td>23</td>
</tr>
<tr>
<td>23</td>
<td>→</td>
<td>24</td>
</tr>
<tr>
<td>24</td>
<td>→</td>
<td>25</td>
</tr>
<tr>
<td>25</td>
<td>→</td>
<td>26,30,31,32</td>
</tr>
<tr>
<td>26</td>
<td>→</td>
<td>27</td>
</tr>
<tr>
<td>27</td>
<td>→</td>
<td>28</td>
</tr>
<tr>
<td>28</td>
<td>→</td>
<td>29</td>
</tr>
<tr>
<td>29</td>
<td>→</td>
<td>30</td>
</tr>
<tr>
<td>30</td>
<td>→</td>
<td>28</td>
</tr>
<tr>
<td>31</td>
<td>→</td>
<td>28</td>
</tr>
<tr>
<td>32</td>
<td>→</td>
<td>33</td>
</tr>
<tr>
<td>33</td>
<td>→</td>
<td>34</td>
</tr>
</tbody>
</table>
Use the cumulative scheduling constraint.

\[
\begin{align*}
\text{min} & \quad z \\
\text{s.t.} & \quad z \geq t_i + p_i, \quad i = 1, \ldots, 34 \\
& \quad \text{cumulative}\left( (t_1, \ldots, t_{34}), (p_1, \ldots, p_{34}), (c_1, \ldots, c_{34}), 8 \right) \\
& \quad t_2 \geq t_1 + 3, \quad t_4 \geq t_1 + 3, \quad \text{etc. (precedence constraints)}
\end{align*}
\]

Note that there are no integer variables. The solver may treat the \( t_i \)'s as integers, but this is an implementation decision.
Edge finding for cumulative scheduling

Consider a cumulative scheduling constraint:

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

A feasible solution:
Edge finding for cumulative scheduling

We can deduce that job 3 must finish after the others finish:  \[ 3 > \{1,2\} \]
Because the total energy required exceeds the area between the earliest release time and the later deadline of jobs 1,2:

\[
e_3 + e_{\{1,2\}} > C \cdot \left( L_{\{1,2\}} - E_{\{1,2,3\}} \right)
\]
Edge finding for cumulative scheduling

We can deduce that job 3 must finish after the others finish: $3 > \{1,2\}$

Because the total energy required exceeds the area between the earliest release time and the later deadline of jobs 1,2:

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

Total energy required = 22
Edge finding for cumulative scheduling

We can deduce that job 3 must finish after the others finish: $3 > \{1,2\}$

Because the total energy required exceeds the area between the earliest release time and the later deadline of jobs 1,2:

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

Total energy required = 22

Area available = 20
Edge finding for cumulative scheduling

We can deduce that job 3 must finish after the others finish: \( 3 > \{1,2\} \)

We can update the release time of job 3 to

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

Energy available for jobs 1,2 if space is left for job 3 to start anytime = 10
Edge finding for cumulative scheduling

We can deduce that job 3 must finish after the others finish: \(3 > \{1,2\}\)

We can update the release time of job 3 to

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

Energy available for jobs 1,2 if space is left for job 3 to start anytime = 10

Excess energy required by jobs 1,2 = 4
Edge finding for cumulative scheduling

We can deduce that job 3 must finish after the others finish: \( 3 > \{1,2\} \)

We can update the release time of job 3 to

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

Energy available for jobs 1,2 if space is left for job 3 to start anytime = 10

Excess energy required by jobs 1,2 = 4

Move up job 3 release time 4/2 = 2 units beyond \( E_{\{1,2\}} \)
Edge finding for cumulative scheduling

In general, if \( e_{J \cup \{k\}} > C \cdot \left( L_J - E_{J \cup \{k\}} \right) \)
then \( k > J \), and update \( E_k \) to

\[
\max_{J' \subset J} \left\{ E_{J'} + \frac{e_{J'} - (C - c_k)(L_{J'} - E_{J'})}{c_k} \right\}
\]

In general, if \( e_{J \cup \{k\}} > C \cdot \left( L_{J \cup \{k\}} - E_J \right) \)
then \( k < J \), and update \( L_k \) to

\[
\min_{J' \subset J} \left\{ L_{J'} - \frac{e_{J'} - (C - c_k)(L_{J'} - E_{J'})}{c_k} \right\}
\]
Edge finding for cumulative scheduling

There is an $O(n^2)$ algorithm that finds all applications of the edge finding rules.
Other propagation rules for cumulative scheduling

- Extended edge finding.
- Timetabling.
- Not-first/not-last rules.
- Energetic reasoning.
## CP and Mathematical Programming

### Comparison

<table>
<thead>
<tr>
<th>CP</th>
<th>MP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic processing</td>
<td>Numerical calculation</td>
</tr>
<tr>
<td>Inference (filtering, constraint propagation)</td>
<td>Relaxation</td>
</tr>
<tr>
<td>High-level modeling (global constraints)</td>
<td>Atomistic modeling (linear inequalities)</td>
</tr>
<tr>
<td>Branching</td>
<td>Branching</td>
</tr>
<tr>
<td>Constraint-based processing</td>
<td>“Independence” of model and algorithm</td>
</tr>
</tbody>
</table>
Software for Constraint Programming

- **ECLiPSe (NICTA), open source**
  - Early CP (and hybrid) solver, still maintained
- **CHIP (Cosytect), commercial**
  - State-of-the-art solver
- **OPL CP Optimizer (IBM), commercial**
  - State-of-the-art solver, originally developed by ILOG
- **Gecode (Schulte & Tack), free download**
  - State-of-the-art toolkit for building CP solvers
- **Frontline MIP/CP solver (Frontline Systems), commercial**
  - Add-in for Excel spreadsheets
- **G12 (NICTA), under development**
  - Major CP and hybrid system
- **Google OR-tools (Google), open source**
  - Includes CP solver
Integrating OR and CP

Complementary strengths
How to integrate
Constraint propagation + relaxation
CP-based branch and price
Decomposition methods
Complementary Strengths

- **CP:**
  - Inference methods
  - Modeling
  - Exploits local structure
  - Good at scheduling

- **OR:**
  - Relaxation methods
  - Duality theory
  - More robust
  - Good with continuous variables

Let’s bring them together!
How to Integrate

• Constraint propagation + relaxation
  – Propagation reduces search space.
  – Relaxation bounds prune the search
• CP-based column generation
  – In branch-and-price methods
  – CP accommodates complex constraints on columns
• Decomposition methods
  – Distinguish master problem and subproblem
  – MILP solves one, CP the other.
• Use CP-style modeling
Constraint propagation + relaxation

• Combine MIP-style bounding with CP-style propagation.
• At each node of search tree:
  – Solve linear relaxation to get bound, as in MIP.
  – Filter variable domains and propagate constraints, as in CP.
Search tree for an IP problem

\[
\text{min } 90x_1 + 60x_2 + 50x_3 + 40x_4 \\
7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42 \\
x_1 + x_2 + x_3 + x_4 \leq 8 \\
x_i \in \{1, 2, 3\}, \quad x_i \geq 1
\]

\[
x_3 + x_4 \geq 2 \\
x_2 + x_4 \geq 2 \\
x_2 + x_3 \geq 3
\]

Knapsack cuts

feasible solution

\[
x = (3, 2, 3, 4, 0) \\
\text{value } = 530
\]

feasible solution

\[
x = (3, 1, 2, 3, 1) \\
\text{value } = 530
\]

backtrack due to bound

infeasible

\[
x = (2 \frac{1}{3}, 3, 2 \frac{2}{3}, 0) \\
\text{value } = 523 \frac{1}{3}
\]
Search tree for an IP problem

\[
\begin{align*}
\text{min } & 90x_1 + 60x_2 + 50x_3 + 40x_4 \\
& 7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42 \\
& x_1 + x_2 + x_3 + x_4 \leq 8 \\
& x_i \in \{1,2,3\}, \quad x_i \geq 1
\end{align*}
\]

Knapsack cuts

\[x_3 + x_4 \geq 2\]
\[x_2 + x_4 \geq 2\]
\[x_2 + x_3 \geq 3\]

Prune tree due to propagation alone.

Filtered domains

feasible solution

Relaxation bound

backtrack due to bound

feasible solution

value = 530
Constraint propagation + relaxation

- **Example: Relaxation of cumulative constraint**
  - Use discrete-time MILP model
    
    $$z_t = z_{t-1} + \sum_j c_j x_{jt} - \sum_j c_j x_{j, t-p_j}, \text{ all } t$$
    
    $$\sum_t x_{jt} = 1, \text{ all } j$$
    
    $$z_t \leq C, \text{ all } t$$
    
    $$0 \leq x_{jt} \leq 1, \text{ all } j, t$$

    - Add this to linear relaxation of problem while propagating the cumulative constraint.
      - 0-1 variables $x_{jt}$ appear only in the relaxation
Constraint propagation + relaxation

- **Example**: Relaxation of cumulative constraint
- Or use valid inequalities in original variables $t_j$

\[
\sum_{j \in J} t_j \geq kr_{j} \quad + \quad \frac{1}{C} \sum_{i=1}^{k} (k - i + 1)p_{ji}c_{ji} - \sum_{i=1}^{k} p_{ji},
\]

Earliest release time in set $J$

\[
\sum_{j \in J} t_j \leq kr_{j} - \frac{1}{C} \sum_{i=1}^{k} (k - i + 1)p_{ji}c_{ji},
\]

where $J = \{i_1, \ldots, i_k\}$ is any subset of jobs.
Constraint propagation + relaxation

- **Example:** Piecewise linear functions
  - Use convex hull relaxation in original variables.

- No need for 0-1 variables
Constraint propagation + relaxation

- Example: Piecewise linear functions
  - Branch on original variables & tighten relaxation.

Solution value of $x$ in current linear relaxation
## Computational Advantage of Integrating CP and OR

Using CP + relaxation from OR

<table>
<thead>
<tr>
<th>Problem</th>
<th>Speedup</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lesson timetabling (LP + red. cost var. fixing)</td>
<td>2 to 50 times faster than CP</td>
<td>Focacci, Lodi, Milano (1999)</td>
</tr>
<tr>
<td>Piecewise linear costs (LP relaxation)</td>
<td>2 to 120 times faster than MILP</td>
<td>Refalo (1999), Yunes, Aron &amp; Hooker (2010)</td>
</tr>
<tr>
<td>Flow shop scheduling, etc. (LP relaxation + cuts)</td>
<td>4 to 150 times faster than MILP.</td>
<td>Hooker &amp; Osorio (1999)</td>
</tr>
</tbody>
</table>
Computational Advantage of Integrating CP and OR

Using CP + relaxation from OR

<table>
<thead>
<tr>
<th>Problem</th>
<th>Speedup</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product configuration (LP relaxation + cuts)</td>
<td>30 to 40 times faster than CP, MILP</td>
<td>Thorsteinsson &amp; Ottosson (2001)</td>
</tr>
<tr>
<td>Automatic recording (Lagrangean relaxation)</td>
<td>1 to 10 times faster than CP, MILP</td>
<td>Sellmann &amp; Fahle (2001)</td>
</tr>
<tr>
<td>Stable set problem (semidefinite relaxation)</td>
<td>Better than CP in less time</td>
<td>Van Hoeve (2001)</td>
</tr>
</tbody>
</table>
# Computational Advantage of Integrating CP and OR

Using CP + relaxation from OR

<table>
<thead>
<tr>
<th>Problem</th>
<th>Speedup</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structural design (nonlinear)</td>
<td>Up to 600 times faster than MILP.</td>
<td>Bollapragada, Ghattas &amp; Hooker (2001)</td>
</tr>
<tr>
<td>(LP quasi-relaxation + logic cuts)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radiation therapy planning</td>
<td>10 times faster than CP, MILP</td>
<td>Cambazard, O’Mahony and O’Sullivan (2010)</td>
</tr>
<tr>
<td>(Lagrangean relaxation)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Applications of Integrated CP and OR

### Using CP + relaxation from OR

<table>
<thead>
<tr>
<th>Application</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orthogonal Latin squares</td>
<td>Appa, Magos &amp; Mourtos (2002)</td>
</tr>
<tr>
<td>Truss structure design</td>
<td>Bollapragada, Gattas &amp; Hooker (2001)</td>
</tr>
<tr>
<td>Chemical processing network design</td>
<td>Grossmann et al. (1994), Hooker &amp; Osorio (1999)</td>
</tr>
<tr>
<td>Multiple machine scheduling</td>
<td>Bockmayr &amp; Pisaruk (2003)</td>
</tr>
<tr>
<td>Shuttle transit routing</td>
<td>Quadrifoglio, Dessouky &amp; Ordóñez (2008)</td>
</tr>
<tr>
<td>Boat party scheduling</td>
<td>Hooker &amp; Osorio (1999)</td>
</tr>
</tbody>
</table>
# Applications of Integrated CP and OR

Using CP + relaxation from OR

<table>
<thead>
<tr>
<th>Application</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multidimensional knapsack problem</td>
<td>Osorio &amp; Glover (2001)</td>
</tr>
<tr>
<td>Factory retrofit planning</td>
<td>Sawaya &amp; Grossmann (2005)</td>
</tr>
<tr>
<td>Strip packing</td>
<td>Sawaya &amp; Grossmann (2005)</td>
</tr>
<tr>
<td>Transport &amp; production problems with piecewise linear costs</td>
<td>Refalo (1999), Ottosson et al. (1999)</td>
</tr>
<tr>
<td>TSP with time windows</td>
<td>Milano &amp; van Hoeve (2002)</td>
</tr>
<tr>
<td>Product configuration</td>
<td>Milano &amp; van Hoeve (2002)</td>
</tr>
</tbody>
</table>
### Applications of Integrated CP and OR

Using CP + relaxation from OR

<table>
<thead>
<tr>
<th>Application</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network design</td>
<td>Cronholm &amp; Ajili (2004)</td>
</tr>
<tr>
<td>Automatic digital recording</td>
<td>Sellmann &amp; Fahle (2001)</td>
</tr>
<tr>
<td>Traveling tournament problem</td>
<td>Benoist, Laburthe &amp; Rottembourg (2001)</td>
</tr>
<tr>
<td>Resource-constrained shortest path problem</td>
<td>Gellermann, Sellmann &amp; Wright (2005)</td>
</tr>
<tr>
<td>Radiation therapy planning</td>
<td>Cambazard, O’Mahony &amp; O’Sullivan (2010)</td>
</tr>
<tr>
<td>CP domain filtering</td>
<td>Khemmoudj, Bennaceur &amp; Nagih (2005)</td>
</tr>
</tbody>
</table>
CP-Based Branch and Price

- Same as traditional branch and price…
  - Except that CP generates the columns.
- Introduce an integer variable for each combinatorial possibility.
  - Removes most or all of symmetry.
  - Filter variable domains and propagate constraints, as in CP.
- At each node of MIP search tree:
  - Solve linear relaxation using column generation.
  - Generate improving columns with CP (pricing problem).
  - Start with these columns at child nodes.
Airline Crew Scheduling

Assign crew members to flights to minimize cost while covering the flights and observing complex work rules.

A roster is the sequence of flights assigned to a single crew member.

The gap between two consecutive flights in a roster must be from 2 to 3 hours.

Total flight time for a roster must be between 6 and 10 hours.

The possible rosters are:

(1,3,5), (1,4,6), (2,3,5), (2,4,6)
Airline Crew Scheduling

There are 2 crew members, and the possible rosters are:

\[(1,3,5), (1,4,6), (2,3,5), (2,4,6)\]

The LP relaxation of the problem is:

\[
\min z
\]

\[
\begin{bmatrix}
10 & 12 & 7 & 13 & 9 & 11 & 6 & 12 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
x_{11} \\
x_{12} \\
x_{13} \\
x_{14} \\
x_{21} \\
x_{22} \\
x_{23} \\
x_{24} \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
1 \\
\end{bmatrix}
\]

\[x_{ik} \geq 0, \text{ all } i, k\]

Cost of assigning crew member 1 to roster 2

\[= 1 \text{ if we assign crew member 1 to roster 2, } = 0 \text{ otherwise.}\]

Each crew member is assigned to exactly 1 roster.

Each flight is assigned at least 1 crew member.
Airline Crew Scheduling

There are 2 crew members, and the possible rosters are:

1           2          3           4
(1,3,5), (1,4,6), (2,3,5), (2,4,6)

The LP relaxation of the problem is:

\[
\min z = \begin{cases} 
1 & \text{if we assign crew member 1 to roster 2,} \\
0 & \text{otherwise.}
\end{cases}
\]

Cost of assigning crew member 1 to roster 2

Each crew member is assigned to exactly 1 roster.

Each flight is assigned at least 1 crew member.

Rosters that cover flight 1.

EWO Seminar
August 2012  Slide 56
Airline Crew Scheduling

There are 2 crew members, and the possible rosters are:

1          2          3          4
(1,3,5), (1,4,6), (2,3,5), (2,4,6)

The LP relaxation of the problem is:

Cost $c_{12}$ of assigning crew member 1 to roster 2

Each crew member is assigned to exactly 1 roster.

Each flight is assigned at least 1 crew member.

In a real problem, there can be millions of rosters.
Airline Crew Scheduling

We start by solving the problem with a subset of the columns:

\[
\begin{align*}
\min z & \\
\begin{bmatrix}
10 & 13 & 9 & 12 \\
1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
\end{bmatrix} & \begin{bmatrix} x_{11} \\ x_{14} \\ x_{21} \\ x_{24} \end{bmatrix} = \begin{bmatrix} z \end{bmatrix} \\
\begin{bmatrix}
(10) \\
(9) \\
(0) \\
(0) \\
(0) \\
(0) \\
(0) \\
(0) \\
(3) \\
\end{bmatrix}
\end{align*}
\]

Optimal dual solution

\[u_1, u_2, v_1, v_2, v_3, v_4, v_5, v_6\]
Airline Crew Scheduling

We start by solving the problem with a subset of the columns:

\[
\begin{bmatrix}
10 & 13 & 9 & 12 \\
1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
\end{bmatrix}\begin{bmatrix}
x_{11} \\
x_{14} \\
x_{21} \\
x_{24} \\
\end{bmatrix} = \begin{bmatrix}
z \\
1 \\
1 \\
1 \\
1 \\
1 \\
\end{bmatrix}\begin{bmatrix}
(10) \\
(9) \\
(0) \\
(0) \\
(0) \\
(3) \\
\end{bmatrix}
\]

\[x_{ik} \geq 0, \text{ all } i, k\]
Airline Crew Scheduling

We start by solving the problem with a subset of the columns:

\[
\min z = \begin{bmatrix} 10 & 13 & 9 & 12 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{14} \\ x_{21} \\ x_{24} \end{bmatrix} = \begin{bmatrix} z \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}
\]

The reduced cost of an excluded roster \( k \) for crew member \( i \) is

\[
c_{ik} - u_i - \sum_{j \text{ in roster } k} v_j
\]

Dual variables

\[
\begin{align*}
u_1 \\
v_1 \\
u_2 \\
v_2 \\
v_3 \\
v_3 \\
v_4 \\
v_4 \\
v_5 \\
v_5 \\
v_6 \\
v_6
\end{align*}
\]

We will formulate the pricing problem as a shortest path problem.
Pricing problem

Crew member 1

Crew member 2

EWO Seminar
August 2012  Slide 61
Pricing problem

Each s-t path corresponds to a roster, provided the flight time is within bounds.
Pricing problem

Cost of flight 3 if it immediately follows flight 1, offset by dual multiplier for flight 1

EWO Seminar
August 2012  Slide 63
Pricing problem

Cost of transferring from home to flight 1, offset by dual multiplier for crew member 1

Dual multiplier omitted to break symmetry
Pricing problem

Length of a path is reduced cost of the corresponding roster.

EWO Seminar
August 2012  Slide 65
Pricing problem

Arc lengths using dual solution of LP relaxation
Pricing problem

Solution of shortest path problems

After $x_{12}$ and $x_{23}$ are added to the problem, no remaining variable has negative reduced cost.
Pricing problem

The shortest path problem cannot be solved by traditional shortest path algorithms, due to the bounds on total duration of flights.

It can be solved by CP:

\[
\min \max \text{Path}(X_i, z_i, G), \text{ all flights } i
\]

Set of flights assigned to crew member \( i \)

Path length

Graph

Path global constraint

Setsum global constraint

\[
T_{\text{min}} \leq \sum_{j \in X_i} (f_j - s_j) \leq T_{\text{max}}
\]

\( X_i \subset \{\text{flights}\}, \ z_i < 0, \text{ all } i \)

Duration of flight \( j \)
Process Scheduling

Assign batches to processing-unit/start-time combinations while observing time constraints.

A schedule is the sequence of unit/start-time combinations assigned to a batch.

The time gap between two consecutive processes in a schedule is bounded.

Total time for a schedule is bounded.
The integer programming model is:

\[ \min \sum_{ij} c_{ij} x_{ij} \]

\[ \sum_{j} x_{ij} = 1, \text{ all } i \quad (u_i) \]

\[ \sum_{ij} a_{jk} x_{ij} \leq 1, \text{ all } k \quad (v_k) \]

\[ x_{ij} \in \{0,1\}, \text{ all } i, j \]

- Cost of assigning batch \( i \) to schedule \( j \)
- Each batch is assigned to exactly 1 schedule.
- Each unit/start-time is assigned at most one batch.
- \( = 1 \) if we assign batch \( i \) to schedule \( j \)
- \( = 1 \) if schedule \( j \) includes unit/start-time \( k \)
Pricing problem

Each s-t path corresponds to a schedule, provided the total processing time is within bounds.

Batch 1

Unit/start-time combination

Batch 2

Everything else is the same as before, except network generation.
Computational Advantage of Integrating CP and OR
Using CP-based Branch and Price

<table>
<thead>
<tr>
<th>Problem</th>
<th>Speedup</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Urban transit crew scheduling</td>
<td>Schedules 210 trips, vs. 120 for traditional branch and price</td>
<td>Yunes, Moura &amp; de Souza (1999)</td>
</tr>
<tr>
<td>Airline crew rostering</td>
<td>Incorporates complicated work rules</td>
<td>Fahle et al. (2002)</td>
</tr>
<tr>
<td>Traveling tournament scheduling</td>
<td>First to solve 8-team instance</td>
<td>Easton, Nemhauser &amp; Trick (2002)</td>
</tr>
</tbody>
</table>
# Applications of Integrated CP and OR

## Using CP-based branch and price

<table>
<thead>
<tr>
<th>Application</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sellmann et al. (2002)</td>
</tr>
<tr>
<td>Aircraft scheduling</td>
<td>Grönqvist (2003)</td>
</tr>
<tr>
<td>Bus crew scheduling</td>
<td>Yunes, Moura &amp; de Souza (2005)</td>
</tr>
<tr>
<td>Network design</td>
<td>Chabrier (2003)</td>
</tr>
<tr>
<td>Employee timetabling</td>
<td>Demassey, Pesant &amp; Rousseau (2005)</td>
</tr>
<tr>
<td>Physician scheduling</td>
<td>Gendron, Lebbah &amp; Pesant (2005)</td>
</tr>
<tr>
<td>Radiation therapy planning</td>
<td>Cambazard, O’Mahony &amp; O’Sullivan (2010)</td>
</tr>
</tbody>
</table>
Decomposition Methods

- Can be applied to planning and scheduling problems.
  - MILP solves planning (master) problem.
  - CP solves scheduling subproblem.
- Link master and subproblem with logic-based Benders cuts.
Facility assignment and scheduling

Assign jobs to facilities, and schedule them, to minimize makespan.

Master problem assigns jobs, using MILP.

Subproblem decomposes into a separate scheduling problem for each facility, solved by CP.

\[
\begin{align*}
\text{min} & \quad M \\
M & \geq t_j + p_{x_{ij}}, \quad \text{all } j \\
r_j & \leq t_j \leq d_j - p_{x_{ij}}, \quad \text{all } j \\
\text{cumulative} \left( (t_j | x_j = i), (p_{ij} | x_j = i), (c_{ij} | x_j = i) \right), \quad \text{all } i
\end{align*}
\]
Facility assignment and scheduling

Subproblem for each facility $i$, given an assignment $x$ from master

$$\min \ M$$
$$M \geq t_j + p_{x_{ij}}, \text{ all } j$$
$$r_j \leq t_j \leq d_j - p_{x_{ij}}, \text{ all } j$$
$$\text{cumulative}((t_j|x_j = i),(p_{ij}|x_j = i),(c_{ij}|x_j = i))$$

Sample Benders cut (all release times the same):

$$M \geq M_{ik} \left( \sum_{j \in J_{ik}} p_{ij} (1 - y_{ij}) + \max_{j \in J_{ik}} \{d_j\} - \min_{j \in J_{ik}} \{d_j\} \right)$$

- Min makespan on facility $i$ in iteration $k$
- $1$ if job $j$ assigned to facility $i$ ($x_j = i$)
- Deadline for job $j$
- Set of jobs assigned to facility $i$ in iteration $k$
Facility assignment and scheduling

The master problem is

\[
\min M \\
M \geq M_{ik} \left( \sum_{j \in J_{ik}} p_{ij} (1 - y_{ij}) + \max_{j \in J_{ik}} \{d_j\} - \min_{j \in J_{ik}} \{d_j\} \right), \quad \text{all } i,k
\]

Relaxation of subproblem
\[y_{ij} \in \{0,1\}\]
## Computational Advantage of Integrating CP and OR

Using CP/MILP Benders methods

<table>
<thead>
<tr>
<th>Problem</th>
<th>Speedup</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min-cost planning &amp; scheduling</td>
<td>20 to 1000 times faster than CP, MILP</td>
<td>Jain &amp; Grossmann (2001), Thorsteinsson (2001)</td>
</tr>
<tr>
<td>Min-cost planning &amp; scheduling</td>
<td>Solved some MIP-intractable instances in &lt;1 sec</td>
<td>Yunes, Aron &amp; Hooker (2010)</td>
</tr>
<tr>
<td>Polypropylene batch scheduling at BASF</td>
<td>Solved previously insoluble problem in 10 min</td>
<td>Timpe (2002)</td>
</tr>
</tbody>
</table>
## Computational Advantage of Integrating CP and OR

Using CP/MILP Benders methods

<table>
<thead>
<tr>
<th><strong>Problem</strong></th>
<th><strong>Speedup</strong></th>
<th><strong>Source</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Call center scheduling</td>
<td>Solved twice as many instances as traditional Benders</td>
<td>Benoist, Gaudin, Rottembourg (2002)</td>
</tr>
<tr>
<td>Min-cost, min-makespan planning &amp; cumulative scheduling</td>
<td>100-1000 times faster than CP, MILP</td>
<td>Hooker (2004)</td>
</tr>
<tr>
<td>Min tardiness planning &amp; cumulative scheduling</td>
<td>10-1000 times faster than CP, MILP</td>
<td>Hooker (2005)</td>
</tr>
</tbody>
</table>
Computational Advantage of Integrating CP and OR

Using CP/MILP Benders methods

<table>
<thead>
<tr>
<th>Problem</th>
<th>Speedup</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sports scheduling</td>
<td>Several orders of magnitude speedup vs MILP</td>
<td>Rasmussen &amp; Trick (2007)</td>
</tr>
<tr>
<td>Single-facility scheduling</td>
<td>Schedules several times as many jobs as MILP. Faster and/or more robust than CP.</td>
<td>Coban &amp; Hooker (2012)</td>
</tr>
</tbody>
</table>
# Applications of Integrated CP and OR

Using CP/MILP Benders methods

<table>
<thead>
<tr>
<th><strong>Application</strong></th>
<th><strong>Reference</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic circuit verification</td>
<td>Hooker &amp; Yan (1995)</td>
</tr>
<tr>
<td>Steel production scheduling</td>
<td>Harjunkoski &amp; Grossmann (2001)</td>
</tr>
<tr>
<td>Computer processor scheduling</td>
<td>Cambazard et al. (2004), Benini et al. (2005, 2008),</td>
</tr>
</tbody>
</table>
## Applications of Integrated CP and OR

**Using CP/MILP Benders methods**

<table>
<thead>
<tr>
<th>Application</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Location-allocation</td>
<td>Fazel-Zarandi &amp; Beck (2009)</td>
</tr>
<tr>
<td>Transport network design</td>
<td>Peterson &amp; Trick (2009)</td>
</tr>
<tr>
<td>Queuing design &amp; control</td>
<td>Terekhov, Beck &amp; Brown (2007)</td>
</tr>
</tbody>
</table>
Software for Integrated Methods

• **ECLiPSe (NICTA), open source**
  - Exchanges information between ECLiPSE solver, Xpress-MP

• **IBM OPL Studio (IBM), commercial**
  - Combines CPLEX and ILOG CP Optimizer with script language

• **Mosel (FICO), commercial**
  - Combines Xpress-MP, Xpress-Kalis with low-level modeling

• **SIMPL (CMU), free download**
  - Full integration with high-level modeling (prototype)

• **SCIP (ZIB), free download, commercial users asked to buy license**
  - Combines MILP and CP-based propagation

• **G12 (NICTA), under development**
  - Converts generic model to one that invokes cooperating solvers

• **BARON (global optimization), commercial**
  - Combines convexification and interval propagation
Some Very Recent Work

Benders for scheduling
Cutting planes from CP model
BDDs as constraint store
BDDs for relaxation bounds
Recent work – Benders for Scheduling

Joint work with Elvin Coban.

Apply logic-based Benders to single-facility scheduling with long time horizons and many jobs.

Decompose the problem by assigning jobs to segments of time horizon.

Segmented problem – Jobs cannot cross segment boundaries (e.g., weekends).

Unsegmented problem – Jobs can cross segment boundaries.
Recent work – Benders for Scheduling

Segmented instances, tight time windows

Table 4: Computation times in seconds for the segmented problem with tight time windows. The number of segments is 10% the number of jobs. Ten instances of each size are solved.

<table>
<thead>
<tr>
<th>Jobs</th>
<th>Feasibility</th>
<th>Makespan</th>
<th>Tardiness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CP</td>
<td>MILP</td>
<td>Bndrs</td>
</tr>
<tr>
<td>60</td>
<td>0.1</td>
<td>14</td>
<td>1.9</td>
</tr>
<tr>
<td>80</td>
<td>181*</td>
<td>45</td>
<td>2.7</td>
</tr>
<tr>
<td>100</td>
<td>199*</td>
<td>58</td>
<td>4.3</td>
</tr>
<tr>
<td>120</td>
<td>272*</td>
<td>137</td>
<td>4.8</td>
</tr>
<tr>
<td>140</td>
<td>306*</td>
<td>260*</td>
<td>6.8</td>
</tr>
<tr>
<td>160</td>
<td>314*</td>
<td>301*</td>
<td>8.0</td>
</tr>
<tr>
<td>180</td>
<td>600*</td>
<td>350*†</td>
<td>4.8</td>
</tr>
<tr>
<td>200</td>
<td>600*</td>
<td>†</td>
<td>5.8</td>
</tr>
</tbody>
</table>

*Solution terminated at 600 seconds for some or all instances.
†MILP solver ran out of memory for some or all instances, which are omitted from the average solution time.
Recent work – Benders for Scheduling

Segmented instances, wide time windows

Table 5: Average computation times in seconds for the segmented problem with wide time windows. The number of segments is 10% the number of jobs. Ten instances of each size are solved.

<table>
<thead>
<tr>
<th>Jobs</th>
<th>Feasibility</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CP</td>
<td>MILP</td>
<td>Bndrs</td>
<td>CP</td>
<td>MILP</td>
<td>Bndrs</td>
<td>CP</td>
<td>MILP</td>
</tr>
<tr>
<td>60</td>
<td>0.05</td>
<td>12</td>
<td>1.9</td>
<td>0.2</td>
<td>16</td>
<td>5.8</td>
<td>0.2</td>
<td>8.0</td>
</tr>
<tr>
<td>80</td>
<td>0.28</td>
<td>22</td>
<td>2.5</td>
<td>180*</td>
<td>59</td>
<td>9.0</td>
<td>1.5</td>
<td>94</td>
</tr>
<tr>
<td>100</td>
<td>0.14</td>
<td>37</td>
<td>3.8</td>
<td>360*</td>
<td>403*</td>
<td>14</td>
<td>79*</td>
<td>594*</td>
</tr>
<tr>
<td>120</td>
<td>0.13</td>
<td>61</td>
<td>5.0</td>
<td>540*</td>
<td>600*</td>
<td>25</td>
<td>600*</td>
<td>251*</td>
</tr>
<tr>
<td>140</td>
<td>61*</td>
<td>175</td>
<td>7.0</td>
<td>600*</td>
<td>600*</td>
<td>107</td>
<td>600*</td>
<td>160*</td>
</tr>
<tr>
<td>160</td>
<td>540*</td>
<td>216*</td>
<td>4.8</td>
<td>600*</td>
<td>562*</td>
<td>157</td>
<td></td>
<td></td>
</tr>
<tr>
<td>180</td>
<td>600*</td>
<td>375*†</td>
<td>4.5</td>
<td>600*</td>
<td>535*</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>200</td>
<td>600*</td>
<td>†</td>
<td>5.5</td>
<td>600*</td>
<td>560*</td>
<td>6.9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Solution terminated at 600 seconds for some or all instances.
†MILP solver ran out of memory for some or all instances, which are omitted from the average solution time.
Recent work – Benders for Scheduling

Unsegmented instances

Table 6: Average computation times in seconds for the unsegmented problem. The number of segments is 10% the number of jobs. Ten instances of each size are solved.

<table>
<thead>
<tr>
<th>Jobs</th>
<th>Feasibility</th>
<th></th>
<th></th>
<th>Makespan</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CP</td>
<td>MILP</td>
<td>Bndrs</td>
<td>CP</td>
<td>MILP</td>
<td>Bndrs</td>
</tr>
<tr>
<td>60</td>
<td>0.10</td>
<td>11</td>
<td>2.8</td>
<td>0.2</td>
<td>24</td>
<td>5.1</td>
</tr>
<tr>
<td>80</td>
<td>0.14</td>
<td>21</td>
<td>3.7</td>
<td>0.7</td>
<td>376*</td>
<td>8.7</td>
</tr>
<tr>
<td>100</td>
<td>0.25</td>
<td>35</td>
<td>7.0</td>
<td>1.1</td>
<td>600*</td>
<td>21</td>
</tr>
<tr>
<td>120</td>
<td>0.43</td>
<td>57</td>
<td>23</td>
<td>0.4</td>
<td>600*</td>
<td>93</td>
</tr>
<tr>
<td>140</td>
<td>0.72</td>
<td>97</td>
<td>65</td>
<td>1.2</td>
<td>600*</td>
<td>115</td>
</tr>
<tr>
<td>160</td>
<td>420*</td>
<td>188</td>
<td>9.0</td>
<td>241*</td>
<td>549*</td>
<td>67</td>
</tr>
<tr>
<td>180</td>
<td>123*</td>
<td>307*</td>
<td>79</td>
<td>61*</td>
<td>600*</td>
<td>168</td>
</tr>
<tr>
<td>200</td>
<td>180*</td>
<td>410*</td>
<td>29</td>
<td>180*</td>
<td>587*</td>
<td>21</td>
</tr>
</tbody>
</table>

*Solution terminated at 600 seconds for some or all instances.

CP solves it quickly (< 1 sec) or blows up, in which case Benders solves it in 6 seconds (average).

So: try CP for 1 sec, then switch to Benders.
Recent work – Cutting Planes from CP Model

Joint work with David Bergman.

Polyhedral analysis of overlapping all-different constraints (equivalent to graph coloring).

Used in many scheduling problems, sudoku puzzles, etc. etc.

Derive cutting planes from CP alldiff formulation and map them into 0-1 model.

Provides tighter bounds than all CPLEX cuts in a small fraction of the time (e.g., 1%).
Recent work – BDDs as Constraint Store

Joint work with Henrik Andersen, David Bergman, Andre Cire, Tarik Hadzic, Willem van Hoeve, Barry O’Sullivan, Peter Tiedemann

Replace variable domains in CP with relaxed binary decision diagrams (BDDs).

BDDs have long been used for circuit design, configuration, etc.

We use them to represent relaxation of feasible set.

Replace domain filtering with BDD-based propagation.

Reduces search tree for multiple alldiffs from 1 million nodes to 1 node, time speedup factor of 100. Speedups on other problems.

Now being incorporated into Google CP solver.
Recent work – BDDs for Relaxation Bounding

Joint work with David Bergman, Andre Cire, Willem van Hoeve

Replace LP relaxation with a relaxed **binary decision diagram** (BDD).

Shortest path in BDD provides a lower bound on optimal value.

For most instances of independent set problem, we get tighter bounds than full cutting plane technology in CPLEX.

Bound is normally obtained in very small fraction of the time.
Further Reading

