A Hybrid Constraint Programming/Integer Programming Method for Scheduling

John Hooker
Carnegie Mellon University

Univ. de São Paulo
October 2012

Problem

- Solve **planning and scheduling problems** that are intractable with existing technology.
  - Assign jobs to processors.
  - Schedule the jobs on each processor.

Approach

- Combine **integer programming** and **constraint programming**.
  - …through **logic-based Benders decomposition**.
  - Exploit complementary strengths.
    - IP is good for assignment problems
    - CP is good for scheduling.
  - Integrated approach is more effective than IP or CP used separately.

Today's Agenda

- Briefly **introduce CP** and resource-constrained scheduling.
- Apply CP/IP Benders to a **generic planning and scheduling** problem.
  - Assign jobs to processors and schedule them subject to time windows.
    - Allow jobs to run in parallel on a processor, subject to a resource constraint.
  - Report computational results.
- Investigate whether CP/IP Benders works for a **pure scheduling problem**.
  - No obvious decomposition

Motivation

- CP and IP can work together.
  - Complementary strengths.
  - IP good for assignment, CP good for scheduling.
- General-purpose solvers can implement integrated methods.
  - Solvers are moving in this direction.
    - IBM OPL Studio, Mosel, SCIP, SIMPL.
  - Logic-based Benders implemented in SIMPL.

What Is Constraint Programming?

Basic Idea

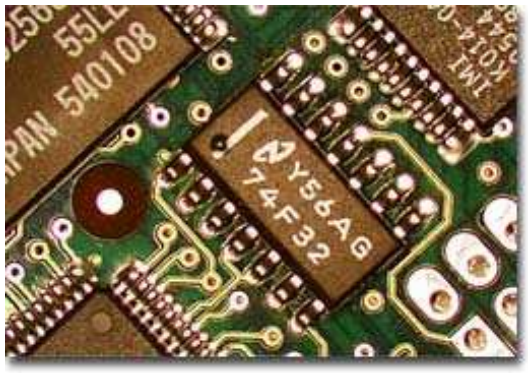
Resource-constrained scheduling

What is Constraint Programming?

- An alternative to optimization methods in operations research.
- Developed in the computer science and artificial intelligence communities.
    - Over the last 20-30 years.
- Particularly successful in scheduling and logistics.

Applications

- 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

Basic Idea

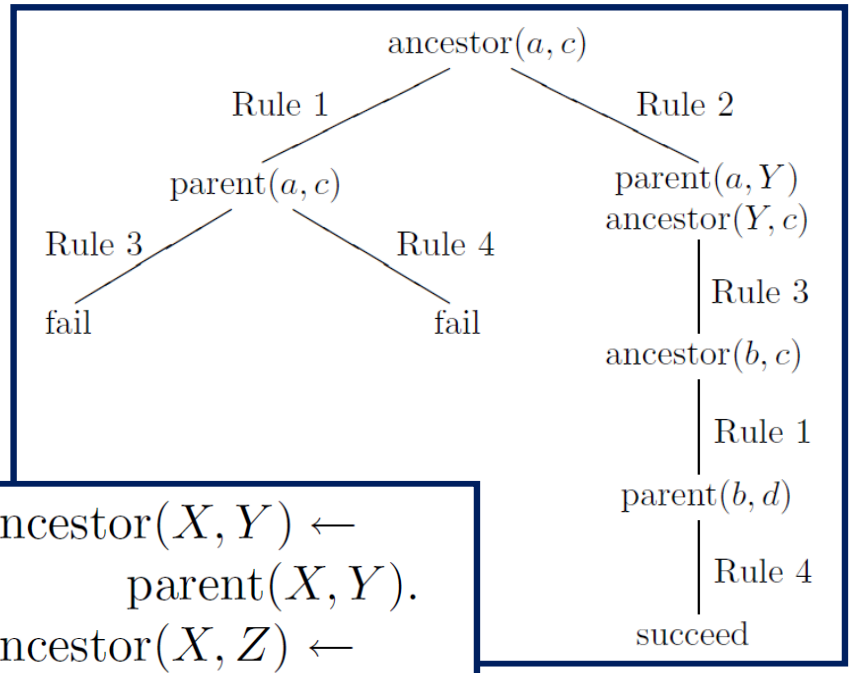
- Each line of the model is both a **constraint** and a **procedure**.
  - **Constraint:** often a high-level global constraint
    - Different modeling paradigm than math programming
  - **Procedure:** removes infeasible values from variable domains
    - Filtering, domain consistency maintenance
 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

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).
- Brief intellectual history…

First step: Logic programming

- Attempt to unify procedural and declarative modeling
  - Procedural: Write the algorithm (CS)
  - Declarative: Write the constraints (OR)
  - Logic programming: Propositions are also procedural goals

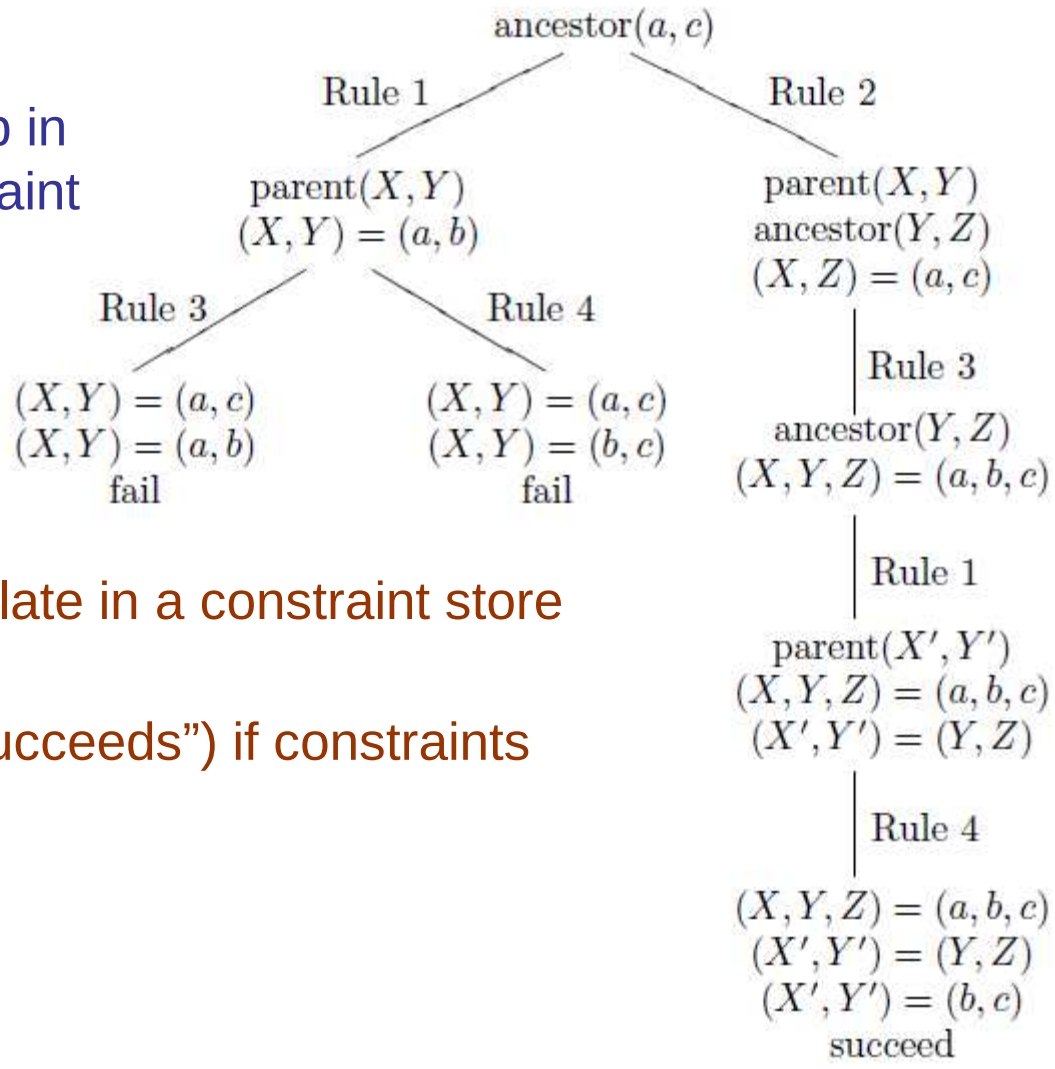


1. ancestor$(X, Y) \leftarrow$ parent(X, Y).
2. ancestor$(X, Z) \leftarrow$ parent(X, Y), ancestor(Y, Z).
3. parent(a, b).
4. parent(b, c).

Example of Prolog

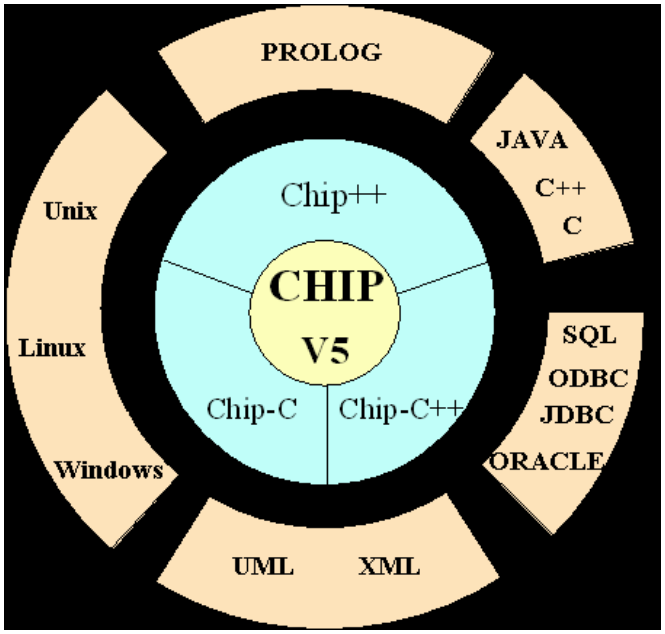
Second step: Constraint logic programming

- Interpret unification step in 1st order logic as constraint solving
  - Extend equality constraints in logic to more general constraints.
  - Constraints accumulate in a constraint store at each leaf node.
  - Node is feasible ("succeeds") if constraints have a solution.



Third step: Constraint programming

- Drop the logic programming framework.
- View each line of the program as specifying both a constraint and a procedure.
  - Constraints are high-level global constraints
  - The procedure removes infeasible values from variable domains (filtering, domain consistency maintenance)
  - Passes reduced domains to next constraint (constraint propagation).



Advantages of CP

- Good at scheduling, logistics
  - …where other optimization methods may fail.
- Adding messy constraints makes the problem easier.
  - The more constraints, the better.
- More powerful modeling language.
  - Simpler models (due to global constraints).
  - Constraints convey problem structure to the solver.

Disadvantages of CP

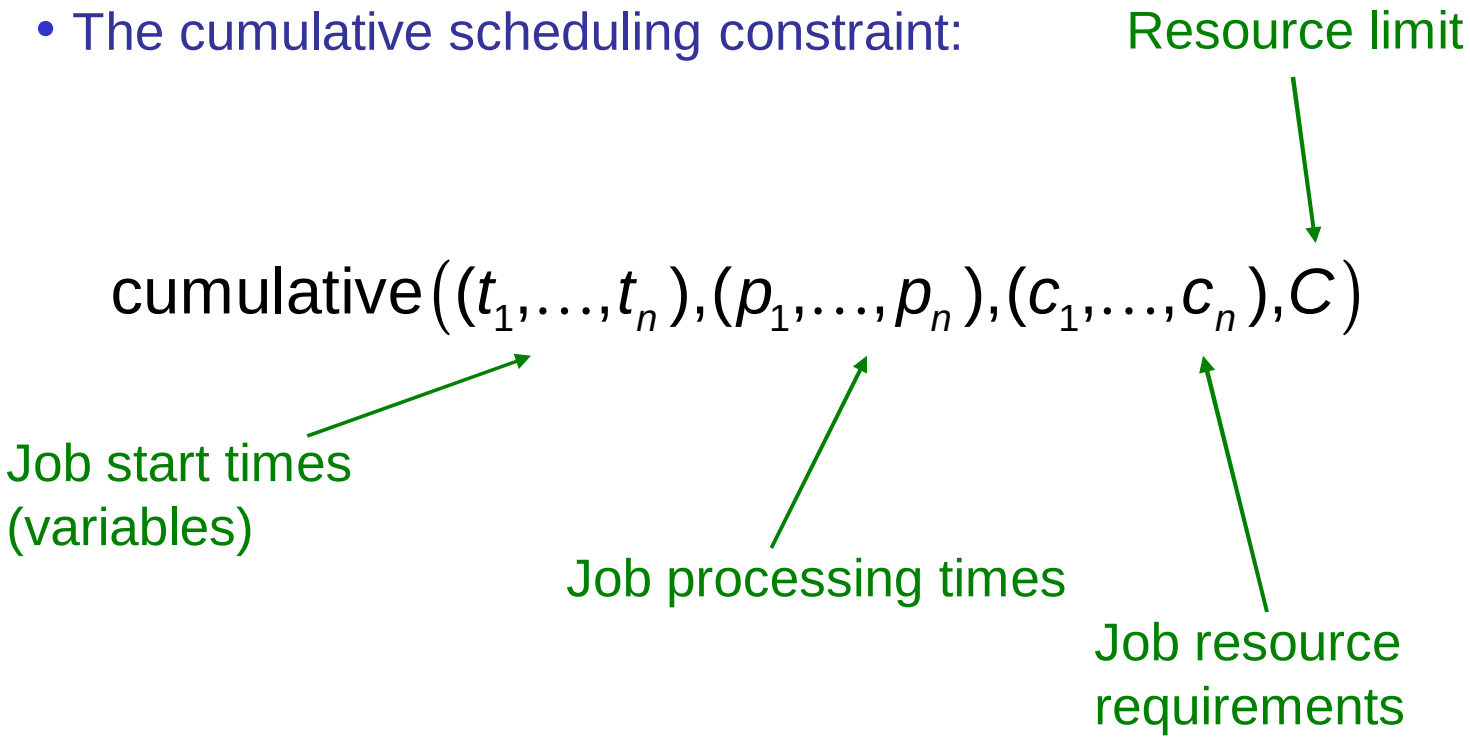
- Less effective for continuous optimization.
  - Relies on interval propagation
- Less robust
  - May blow up past a certain problem size,
  - Lacks relaxation technology
- Software is less highly engineered
  - Younger field

Resource-constrained Scheduling

- One of CP's most successful areas.
  - Schedule jobs in parallel.
  - Subject to time windows and a resource constraint
- Implemented by the **cumulative scheduling** constraint.
  - This is a **global constraint**
  - That is, a constraint that enforces a highly-structured set of more elementary constraints.

Resource-constrained Scheduling

- The cumulative scheduling constraint:



Example: 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.

Precedence constraints

1 → 2,4
2 → 3
3 → 5,7
4 → 5
5 → 6
6 → 8
7 → 8
8 → 9
9 → 10
9 → 14
10 → 11
10 → 12
11 → 13
12 → 13
13 → 15,16
14 → 15
15 → 18
16 → 17
17 → 18
18 → 19
18 → 20,21
19 → 23
20 → 23
21 → 22
22 → 23
23 → 24
24 → 25
25 → 26,30,31,32
26 → 27
27 → 28
28 → 29
30 → 28
31 → 28
32 → 33
33 → 34

Use the cumulative scheduling constraint.

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

Note that there are no integer variables.

Bounds propagation for cumulative scheduling

- **Domain filtering** is a key technology in CP solvers.
  - Remove infeasible values from variable domains.
  - That is, values that cannot occur in any feasible solution.
- **Bounds propagation** is one form of filtering.
  - Tighten bounds on start time of a job.
  - Propagate the bounds to other constraints in the problem.
- **Edge finding** is the most basic technique.

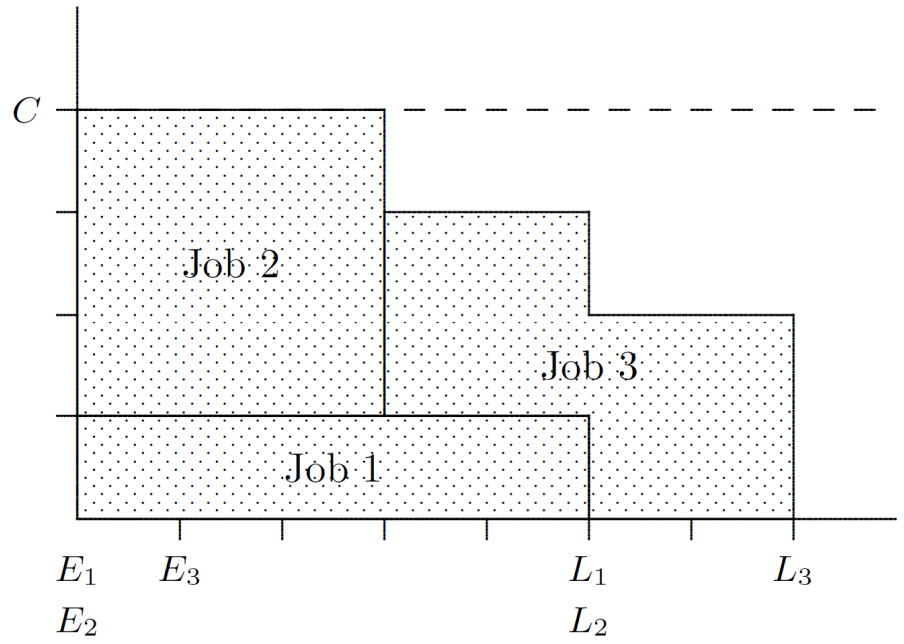
Bounds propagation for cumulative scheduling

Consider a cumulative scheduling constraint:

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

j	p_j	c_j	E_j	L_j
1	5	1	0	5
2	3	3	0	5
3	4	2	1	7

A feasible solution:

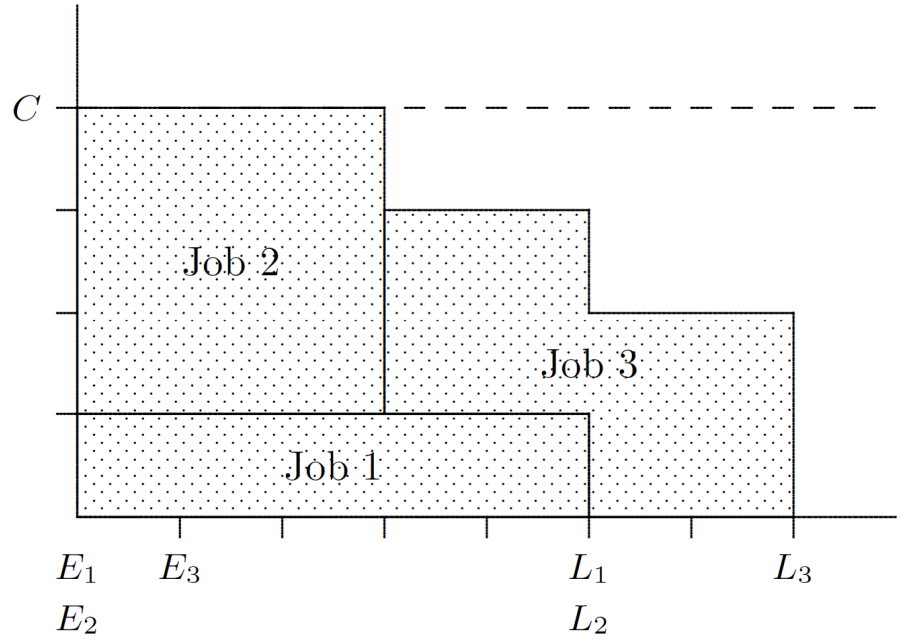


Bounds propagation for cumulative scheduling

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

Suppose that job 3 is not the last to finish.

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



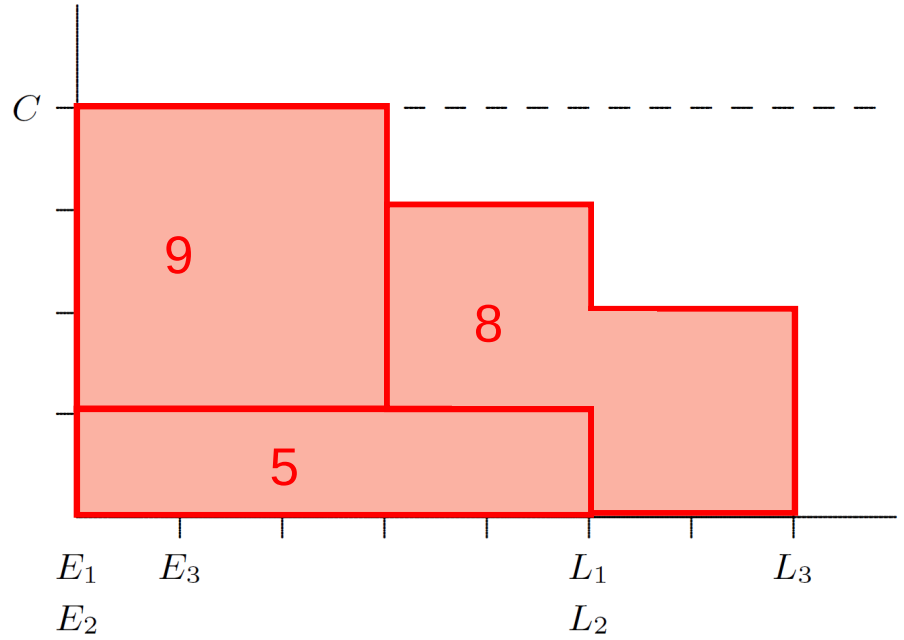
Bounds propagation for cumulative scheduling

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

Suppose that job 3 is not the last to finish.

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

Total energy required = 22



Bounds propagation for cumulative scheduling

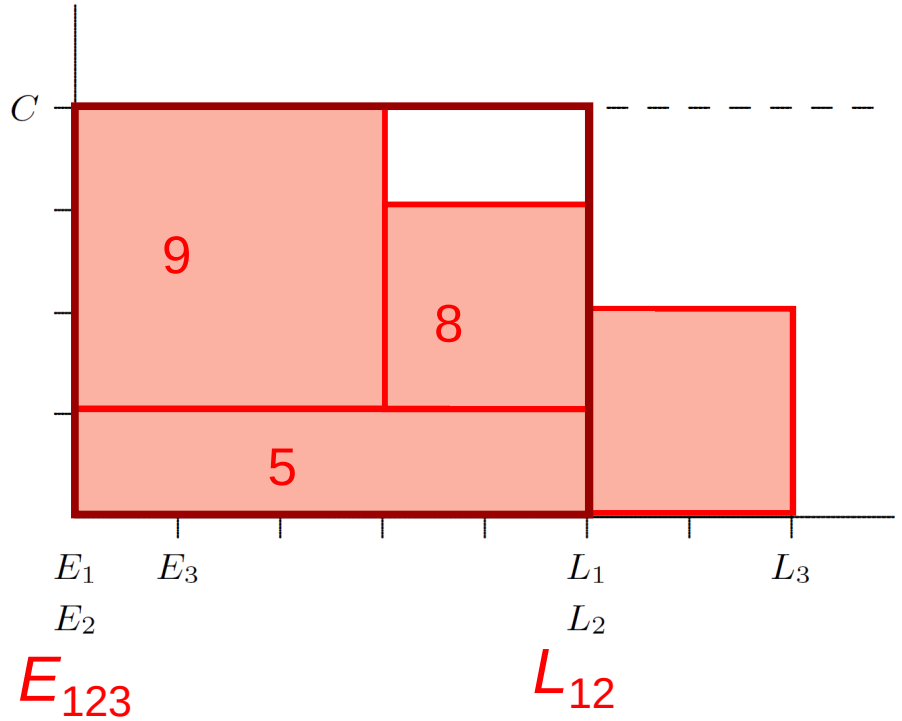
We can deduce that job 3 must finish after the others finish: $3 > \{1,2\}$

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

Energy available = 20



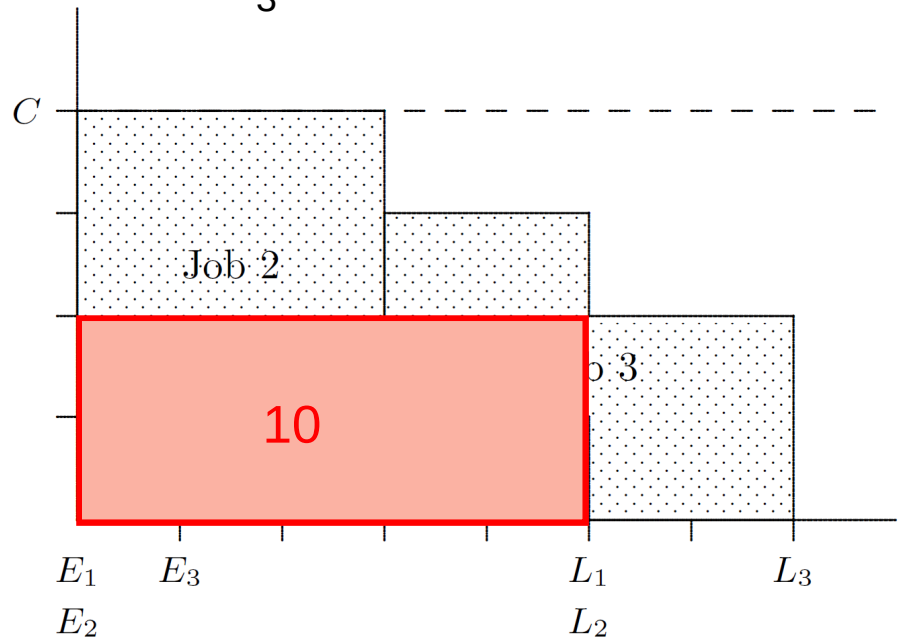
Bounds propagation for cumulative scheduling

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



Bounds propagation for cumulative scheduling

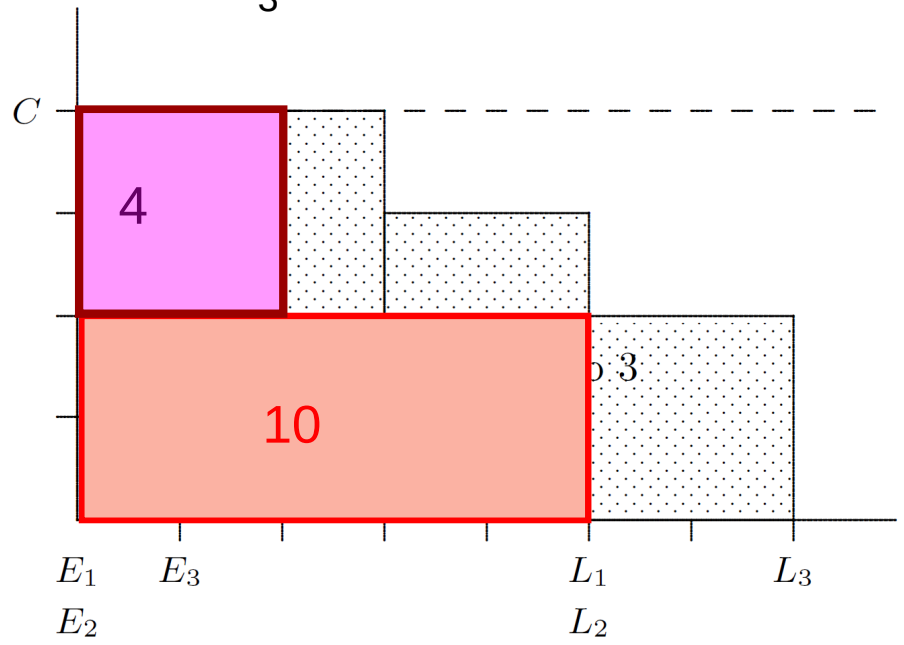
We can deduce that job 3 must finish after the others finish: $3 > \{1,2\}$

We can update the release time of job 3 to

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

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



Bounds propagation for cumulative scheduling

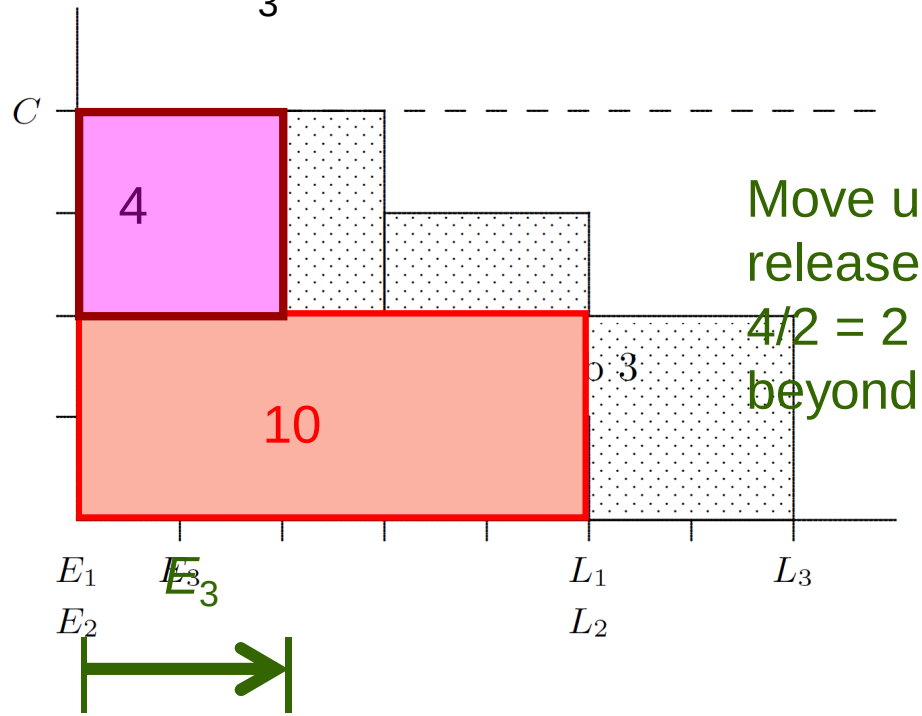
We can deduce that job 3 must finish after the others finish: $3 > \{1,2\}$

We can update the release time of job 3 to

$$E_{\{1,2\}} + \frac{e_{\{12\}} - (C - c_3)(L_{\{1,2\}} - E_{\{1,2\}})}{c_3}$$

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 $E_{\{1,2\}}$

Bounds propagation 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

$$\max_{\substack{J'\subset J \\ e_{J'}-(C-c_k)(L_{J'}-E_{J'})>0}} \left\{ E_{J'} + \frac{e_{J'} - (C-c_k)(L_{J'}-E_{J'})}{c_k} \right\}$$

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

$$\min_{\substack{J'\subset J \\ e_{J'}-(C-c_k)(L_{J'}-E_{J'})>0}} \left\{ L_{J'} - \frac{e_{J'} - (C-c_k)(L_{J'}-E_{J'})}{c_k} \right\}$$

Bounds propagation 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

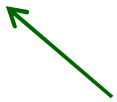
- Extended edge finding.
- Timetabling.
- Not-first/not-last rules.
- Energetic reasoning.

Schemes for CP/IP Integration

- 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

Schemes for CP/IP Integration

- 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



Today's topic

Planning and Scheduling

A small example

Benders cuts

Computational results

A Small Example

- Assign 5 jobs to 2 processors (A and B), and schedule the machines assigned to each machine within time windows.
  - The objective is to minimize makespan.

Time lapse between start of first job and end of last job.

- Assign the jobs in the **master problem**, to be solved by IP.
- Schedule the jobs in the **subproblem**, to be solved by CP.

A Small Example

Job Data

Job j	*Release time* r_j	*Dead-line* d_j	*Processing time* p_{Aj}	p_{Bj}
1	0	9	1	5
2	0	9	3	6
3	2	7	3	7
4	2	9	4	6
5	4	7	2	5

Processsor A

Processor B

Example

Assign 5 jobs to 2 processors.

Schedule jobs assigned to each processor without overlap.

A Small Example

Job Data

Job j	*Release time* r_j	*Dead-line* d_j	*Processing time* p_{Aj}	p_{Bj}
1	0	9	1	5
2	0	9	3	6
3	2	7	3	7
4	2	9	4	6
5	4	7	2	5

Machine A

Machine B

Once jobs are assigned, we can minimize overall makespan by minimizing makespan on each processor individually.

So the subproblem decouples.

A Small Example

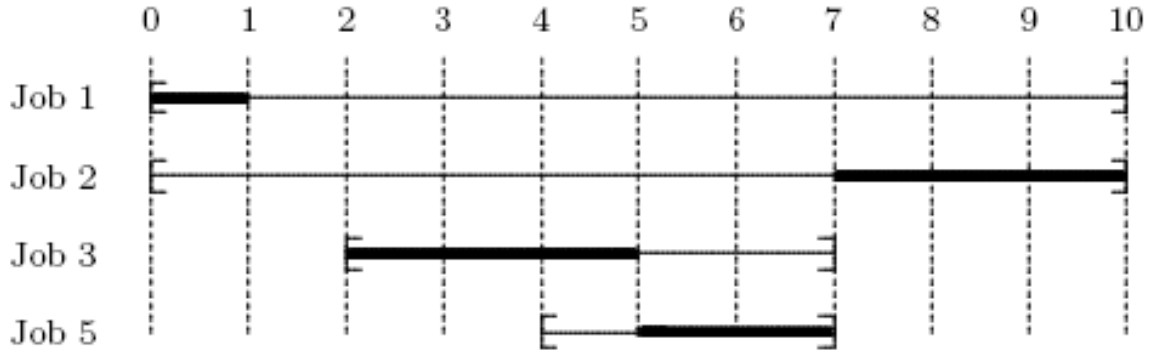
Job Data

Job j	*Release time* r_j	*Dead-line* d_j	*Processing time* p_{Aj}	p_{Bj}
1	0	9	1	5
2	0	9	3	6
3	2	7	3	7
4	2	9	4	6
5	4	7	2	5

Once jobs are assigned, we can minimize overall makespan by minimizing makespan on each processor individually.

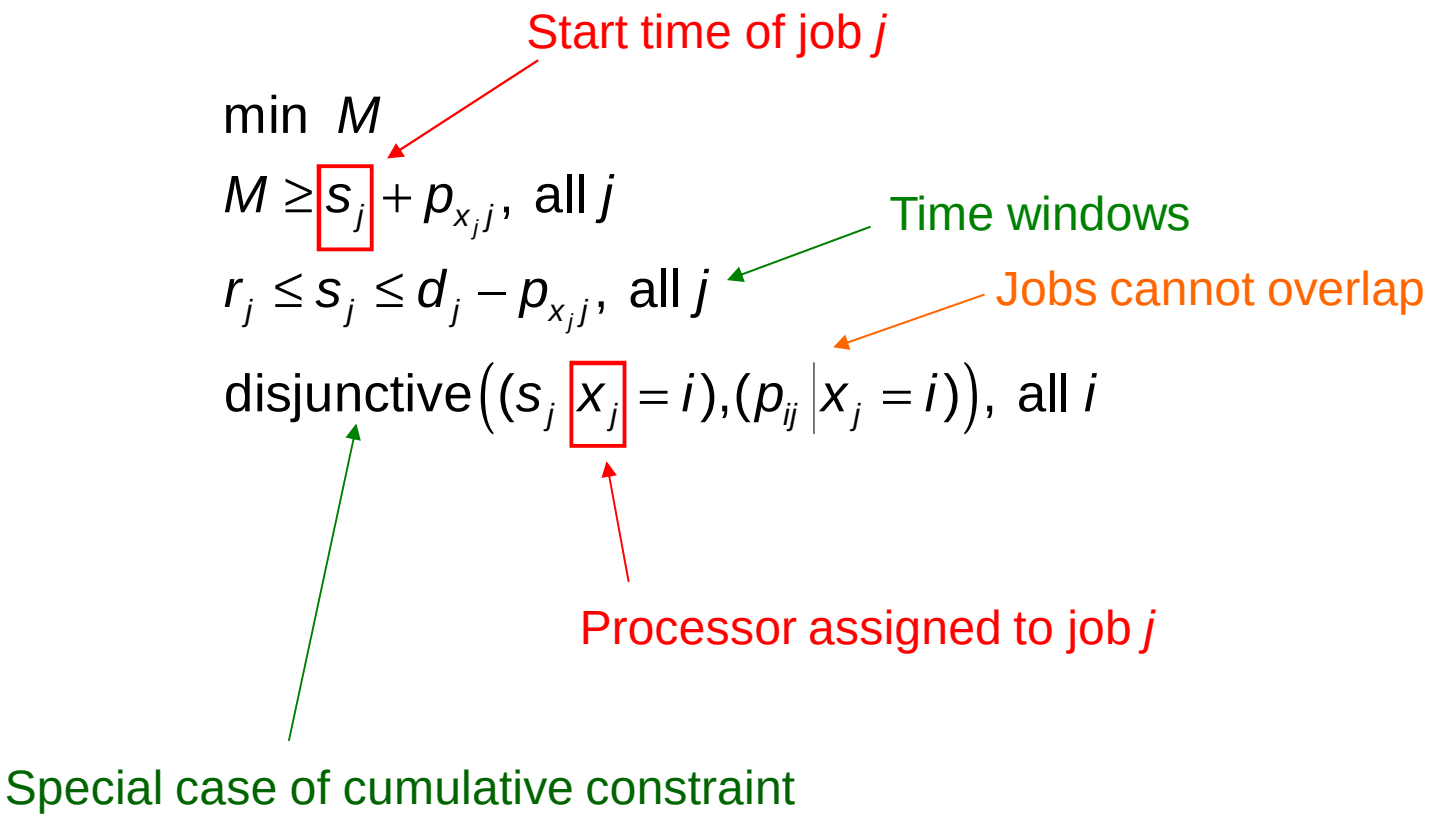
So the subproblem decouples.

Minimum makespan schedule for jobs 1, 2, 3, 5 on processor A



A Small Example

The model:



A Small Example

The model:

Start time of job j

$$\min \; M$$

$$M \geq s_j + p_{x_j j}, \text{ all } j$$

Time windows

$$r_j \leq s_j \leq d_j - p_{x_j j}, \text{ all } j$$

Jobs cannot overlap

$$\text{disjunctive}\left((s_j \,|\, x_j = i), (p_{ij} \,|\, x_j = i)\right), \text{ all } i$$

For a fixed assignment $\bar{x}$ the subproblem on each processor i is

$$\min \; M$$

$$M \geq s_j + p_{\bar{x}_j j}, \text{ all } j \text{ with } \bar{x}_j = i$$

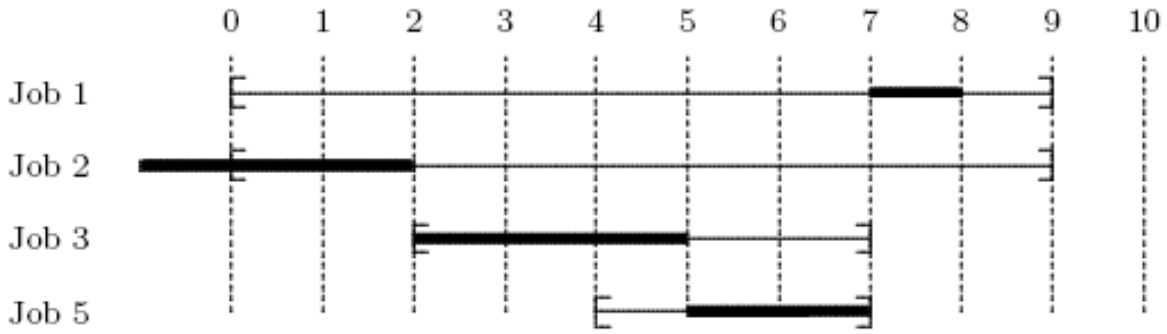
$$r_j \leq s_j \leq d_j - p_{\bar{x}_j j}, \text{ all } j \text{ with } \bar{x}_j = i$$

$$\text{disjunctive}\left((s_j \,|\, \bar{x}_j = i), (p_{ij} \,|\, \bar{x}_j = i)\right)$$

Benders cuts

Suppose we assign jobs 1,2,3,5 to processor A in iteration *k*.

We can prove that 10 is the optimal makespan by proving that the schedule is infeasible with makespan 9.



Edge finding derives infeasibility by reasoning only with jobs 2,3,5. So these jobs alone create a minimum makespan of 10.

So we have a simple "nogood" cut

$$M \geq B_{k+1}(x) = \begin{cases} 10 & \text{if } x_2 = x_3 = x_4 = A \\ 0 & \text{otherwise} \end{cases}$$

Benders cuts

We want the master problem to be an IP, which is good for assignment problems.

So we write the Benders cut

$$M \geq B_{k+1}(x) = \begin{cases} 10 & \text{if } x_2 = x_3 = x_4 = A \\ 0 & \text{otherwise} \end{cases}$$

using 0-1 variables:

$$M \geq 10\left(x_{A2} + x_{A3} + x_{A5} - 2\right)$$

$$M \geq 0$$

= 1 if job 5 is assigned to processor A

Master problem

The master problem is an MILP:

$$\min \; M$$

$$\sum_{j=1}^{5} p_{Aj} x_{Aj} \le 10, \text{ etc.}$$

$$\sum_{j=1}^{5} p_{Bj} x_{Bj} \le 10, \text{ etc.}$$

$$M \ge \sum_{j=1}^{5} p_{ij} x_{ij}, \;\; v \ge 2 + \sum_{j=3}^{5} p_{ij} x_{ij}, \text{ etc.}, \;\; i = A, B$$

$$M \ge 10(x_{A2} + x_{A3} + x_{A5} - 2)$$

$$M \ge 8 x_{B4}$$

$$x_{ij} \in \{0,1\}$$

Constraints derived from time windows

Constraints derived from release times

Benders cut from processor A

Benders cut from processor B

Stronger Benders cuts

If all release times are the same, we can strengthen the Benders cuts.

We are now using the cut

$$v \geq M_{ik}\left(\sum_{j \in J_{ik}} x_{ij} - |J_{ik}| + 1\right)$$

Min makespan on processor *i* in iteration *k*

Set of jobs assigned to processor *i* in iteration *k*

A stronger cut provides a useful bound even if only some of the jobs in J_{ik} are assigned to processor *i*:

$$v \geq M_{ik} - \sum_{j \in J_{ik}} (1 - x_{ij}) p_{ij}$$

Stronger Benders cuts

To strengthen cuts further, the subproblem is re-solved after removing jobs one at a time from J_{ik}.

$$v \geq M_{ik}\left(\sum_{j \in J_{ik}} x_{ij} - |J_{ik}| + 1\right)$$

Min makespan on processor *i* in iteration *k*

Set of jobs assigned to processor *i* in iteration *k*

If removing a job has no effect on min makespan, it is left out of J_{ik}.

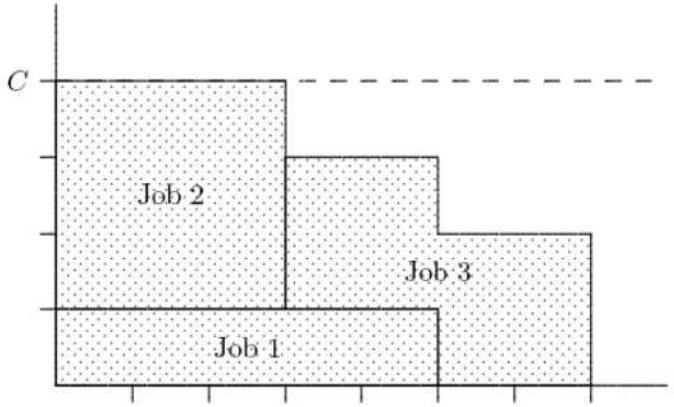
Stronger Benders cuts

Min makespan Benders cut for **cumulative scheduling** subproblem, if release times are equal.

$$M \geq M_i^* - \left(\sum_{j \in J_i} p_{ij}(1 - x_{ij}) + \max_{j \in J_i}\{d_j\} - \min_{j \in J_i}\{d_j\} \right)$$

Jobs currently assigned to processor *i*

Minimum makespan on processor *i* for jobs currently assigned



Min cost problem

There is a cost for assigning each job to each processor.

Subproblem is a feasibility problem – Can these jobs be assigned to the processor?

Benders cuts are very simple:

$$\sum_{j \in J_i} (1 - x_{ij}) \geq 1$$

← Don't assign these jobs to processor *i* again

Cuts are iteratively strengthened as before.

Computational Results

IP model is solved by CPLEX 11.

We are now updating results using CPLEX 12 in both IP and Benders methods. Also faster CP solver for subproblems.

Results – Min cost problem

Long processing times – average of 5 instances

Jobs	Proces-sors	MILP (CPLEX 11) Nodes	MILP (CPLEX 11) Sec.	Benders Iter.	Benders Cuts	Benders Sec.
3	2	1	0.00	2	1	0.00
7	3	1	0.00	13	16	0.12
12	3	3,351	6.6	26	35	0.73
15	5	2,779	8.8	20	29	0.83
20	5	33,321	882	13	82	5.4
22	5	352,309	10,563	69	98	9.6

Results – Min cost problem

Short processing times – average of 5 instances

Jobs	Proces-sors	MILP (CPLEX 11) Nodes	MILP (CPLEX 11) Sec.	Benders Iter.	Benders Cuts	Benders Sec.
3	2	1	0.01	1	0	0.00
7	3	1	0.02	1	0	0.00
12	3	499	0.98	1	0	0.01
15	5	529	2.6	2	1	0.06
20	5	250,047	369	6	5	0.28
22	5	> 27.5 mil.	> 48 hr	9	12	0.42
25	5	> 5.4 mil.	> 19 hr*	17	21	1.09

*out of memory

Results – Min makespan problem

Average of 5 instances

3 processors

Jobs	MILP Sec.	Benders Sec.
10	3.9	0.23
12	12	0.38
14	524	1.4
16	1716+	7.6
28	4619+	30
20		8.7
22		2012+

4 processors

Jobs	MILP Sec.	Benders Sec.
10	1	0.19
12	5	0.43
14	24	0.82
16	35	1.0
28	3931+	4.4
20		28
22		945

+Some instances exceeded limit of 2 hours

Results – Min cost and makespan

Benders method – Larger instances (average of 5)

Jobs	Processors	Min cost Sec.	Min makespan Sec.
10	2	0.1	0.2
15	3	0.3	1.6
20	4	3.2	32
25	5	3.3	28
30	6	1.4	65
35	7	8.0	767
40	8	157	5944+
45	9	95	5762+
50	10	19	

+Some instances exceeded limit of 2 hours

Min Tardiness Problem

- The min tardiness problem cuts are slightly different.

$$T_i \geq T_i^* \left(1 - \sum_{j \in J_i} (1 - y_{ij})\right)$$

Similar to makespan nogood cuts

$$T_i \geq T_i(J_i \setminus Z_i) \left(1 - \sum_{j \in J_i \setminus Z_i} (1 - y_{ij})\right)$$

$T_i(S)$ = min tardiness on processor i when it runs jobs in set S

Set of jobs that can be removed from processor i, one at a time with replacement, without changing the min tardiness.

Results – Min tardiness problems

3 processors – Individual instances

Jobs	MILP Sec.	Benders Sec.
10	4.7	2.6
	6.4	1.6
	6.4	1.6
	32	4.1
	33	22
12	0.7	0.2
	0.6	0.1
	0.7	0.2
	15	2.4
	25	12

Jobs	MILP Sec.	Benders Sec.
14	7.0	6.1
	34	3.7
	45	19
	73	40
	>7200	3296
16	19	1.4
	46	2.1
	52	4.2
	1105	156
	3424	765

Results – Min tardiness problems

3 processors – Individual instances

Jobs	MILP Sec.	Benders Sec.
18	187	2.8
	15	5.3
	46	49
	256	47
	>7200	1203
20	105	18
	4141	23
	39	29
	1442	332
	>7200	>7200

Jobs	MILP Sec.	Benders Sec.
22	6.3	19
	584	37
	>7200	>7200
	>7200	>7200
	>7200	>7200
24	10	324
	>7200	94
	>7200	110
	>7200	>7200
	>7200	>7200

Summary of results

- Benders is much faster for min cost and min makespan problems.
- Benders is somewhat faster for min tardiness problem.
    - Better cuts are needed.
- Updated results are similar so far.

Inference Dual

- In general, Benders cuts are obtained by solving the **inference dual** of the subproblem.
  - The dual solution is a **proof** of optimality.
  - LP dual is a special case, where the proof is encoded by dual multipliers.

Inference Dual

- In general, Benders cuts are obtained by solving the **inference dual** of the subproblem.
  - The dual solution is a **proof** of optimality.
  - LP dual is a special case, where the proof is encoded by dual multipliers.
- The Benders cut states conditions on the master problem variables under which the proof **remains valid**.
  - Classical Benders cut is a special case.

A Pure Scheduling Problem

Segmented problem

Unsegmented problem

Single-processor Scheduling

- Apply logic-based Benders to **single-processor scheduling** with long time horizons and many jobs.
  - The classic **one-machine scheduling** problem.

Single-processor Scheduling

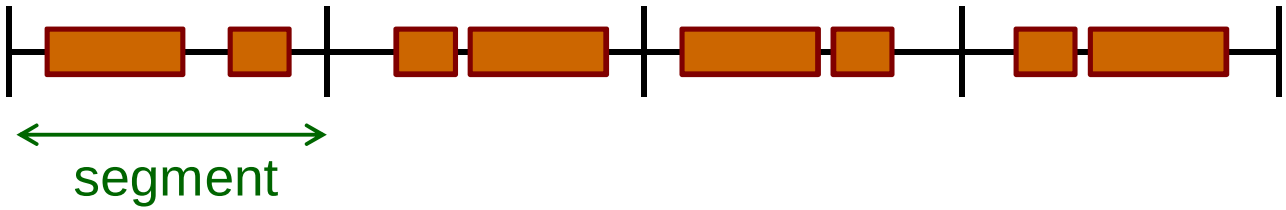
- Apply logic-based Benders to **single-processor scheduling** with long time horizons and many jobs.
  - The classic **one-machine scheduling** problem.
- The problem does not naturally decompose.
  - But we decompose it by assigning jobs to **segments** of the time horizon.

Single-processor Scheduling

- Apply logic-based Benders to **single-processor scheduling** with long time horizons and many jobs.
  - The classic **one-machine scheduling** problem.
- The problem does not naturally decompose.
  - But we decompose it by assigning jobs to **segments** of the time horizon.
- Two versions:
  - **Segmented problem** – Jobs cannot cross segment boundaries (e.g., weekends).
  - **Unsegmented problem** – Jobs can cross segment boundaries.

Segmented problem

- Benders approach is very similar to that for the planning and scheduling problem.
  - Assign jobs to time segments rather than processors.
  - Benders cuts are the same.



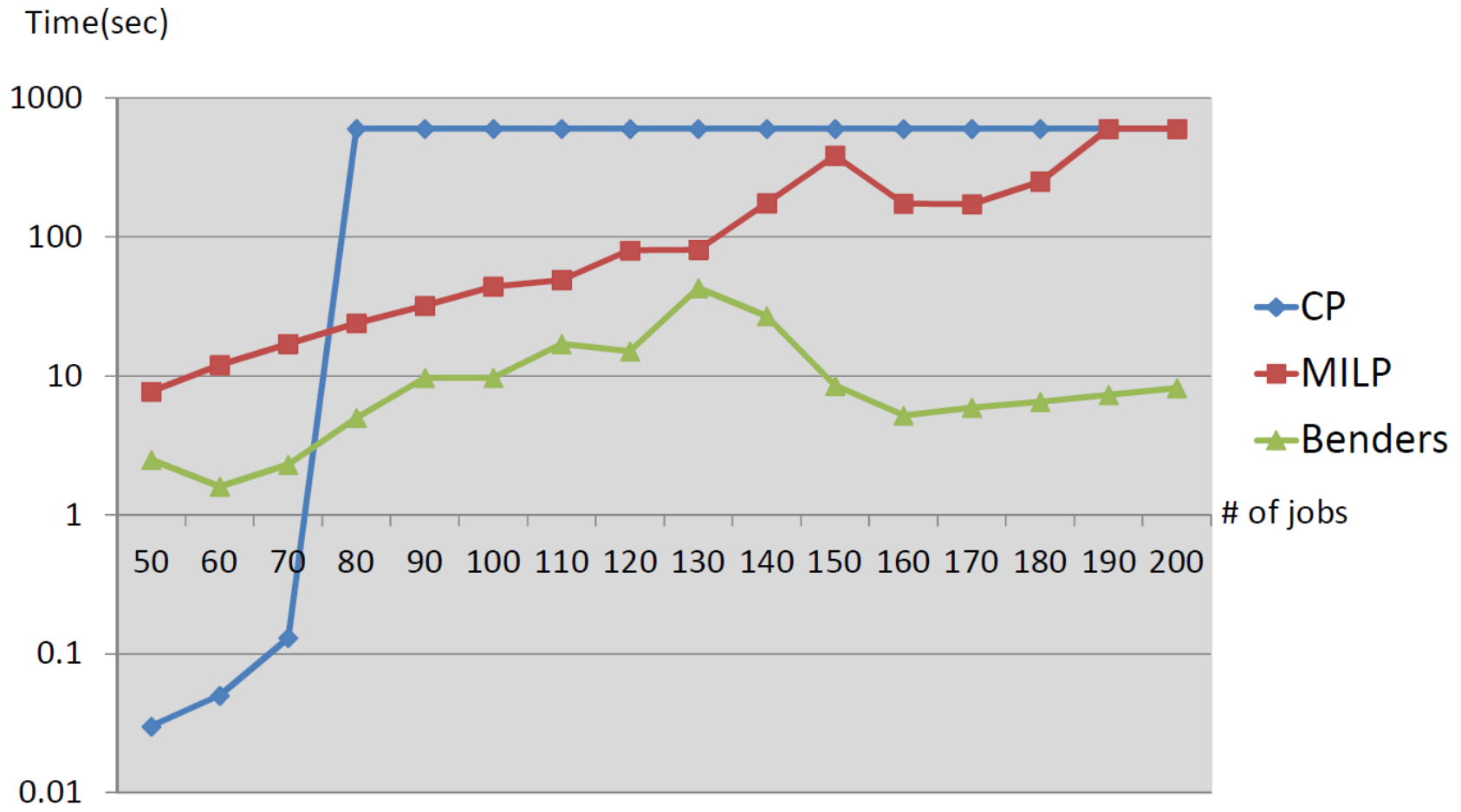
Jobs do not overlap segment boundaries

Segmented problem

- Experiments use most recent versions of CP and IP solvers.
  - IBM OPL Studio 6.1
  - CPLEX 12

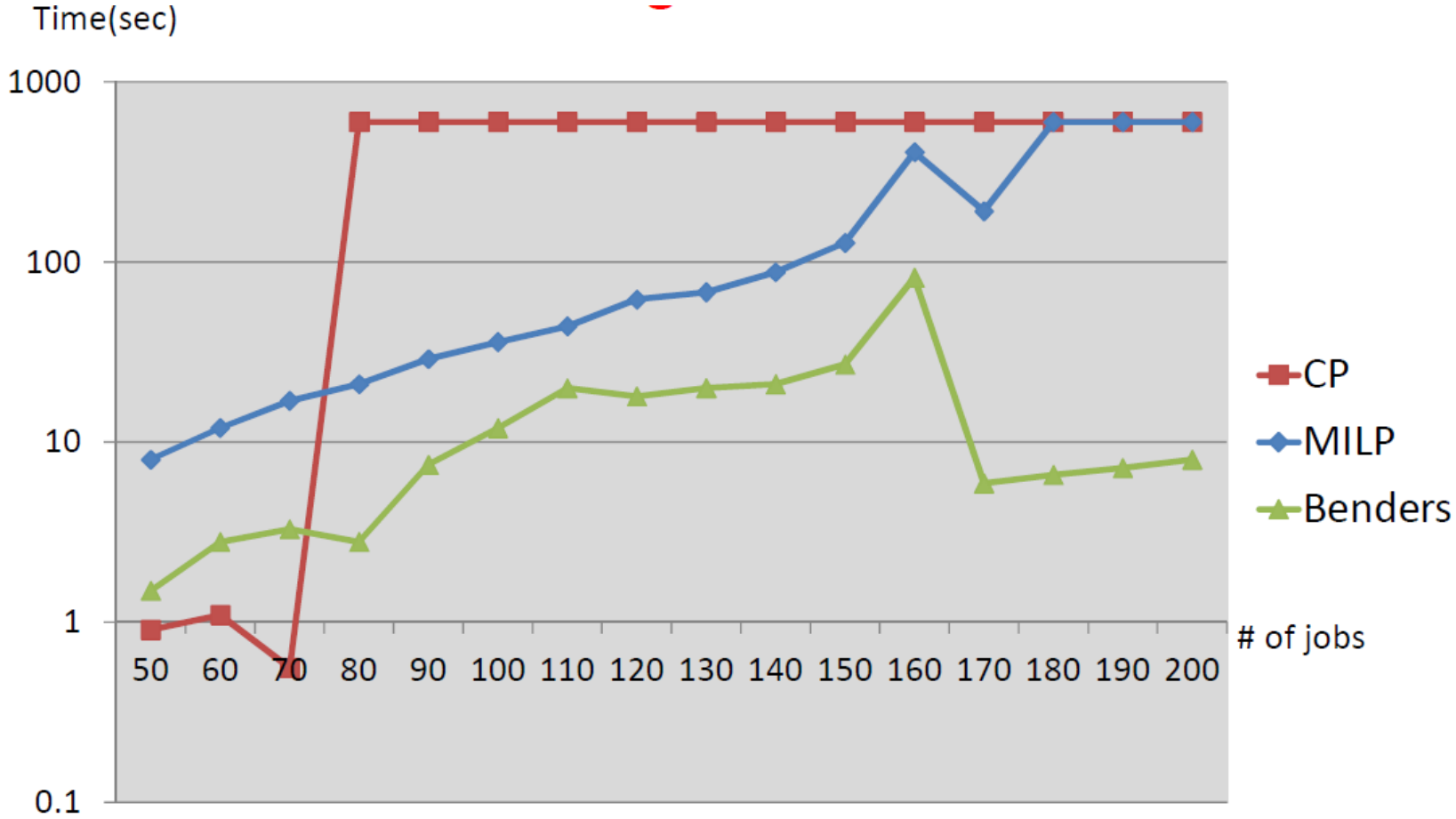
Segmented problem computational results

Feasibility – Wide time windows (individual instances)



Segmented problem computational results

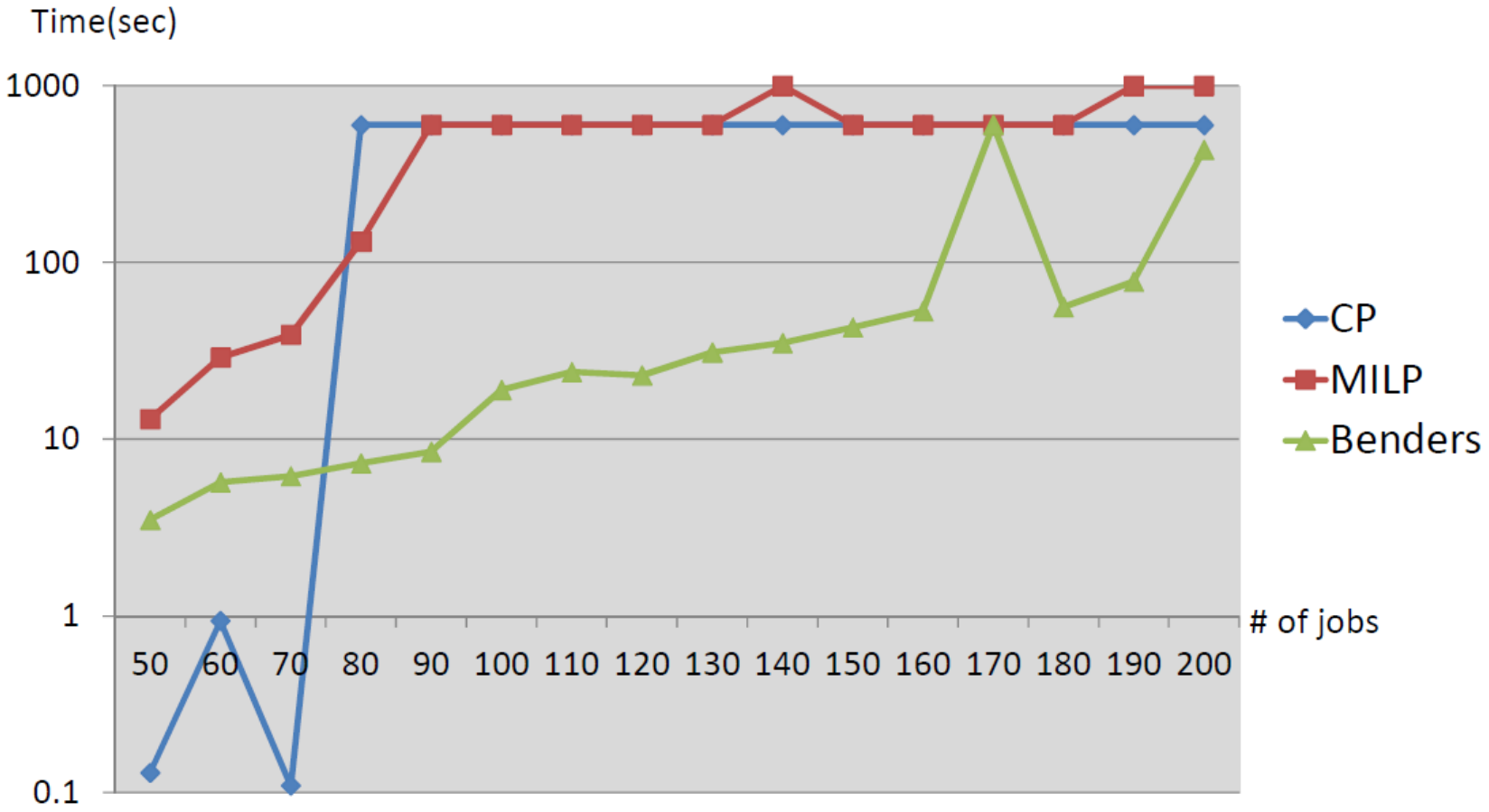
Feasibility – Tight time windows (individual instances)



Slide 68

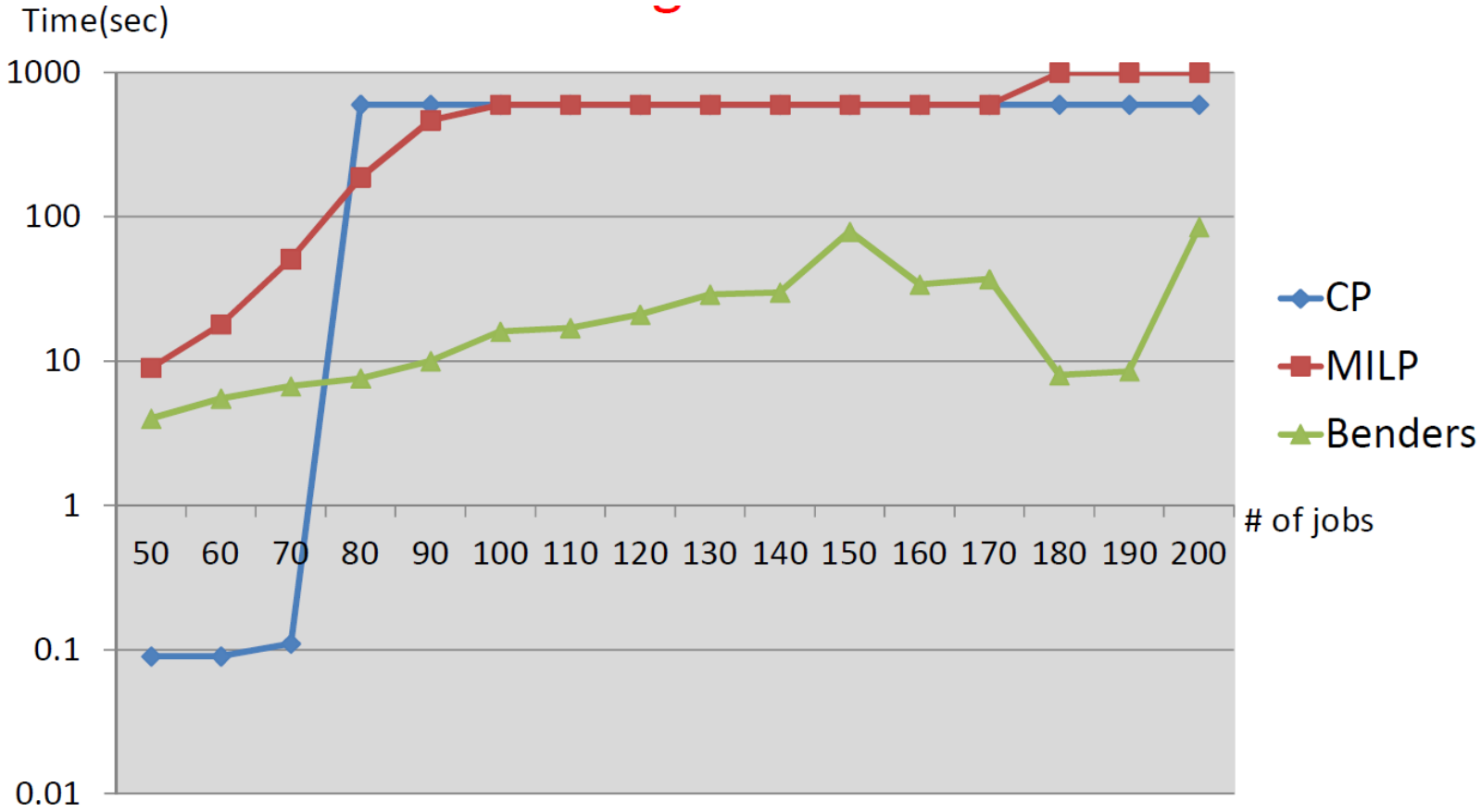
Segmented problem computational results

Min makespan – Wide time windows (individual instances)



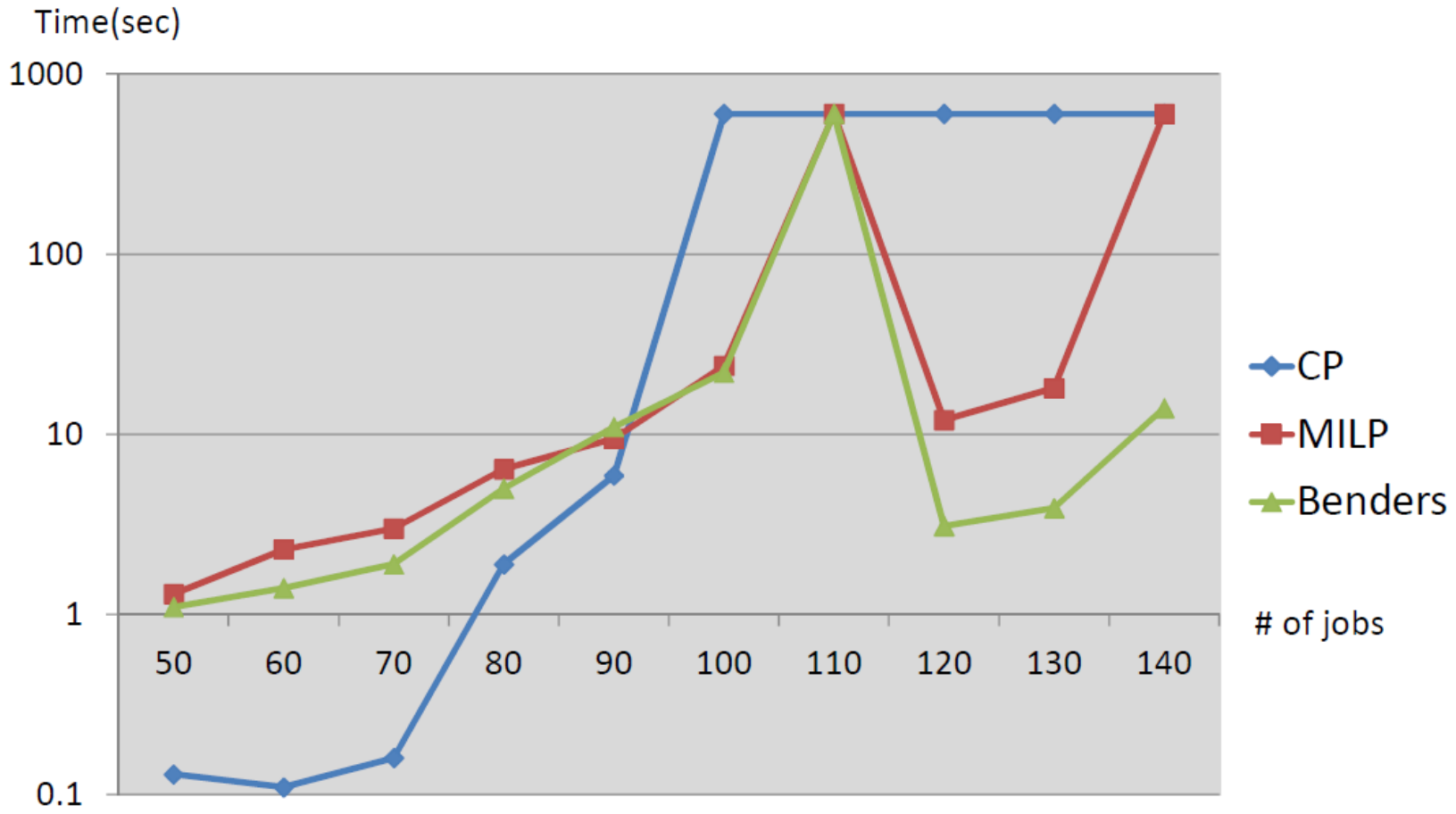
Segmented problem computational results

Min makespan – Tight time windows (individual instances)



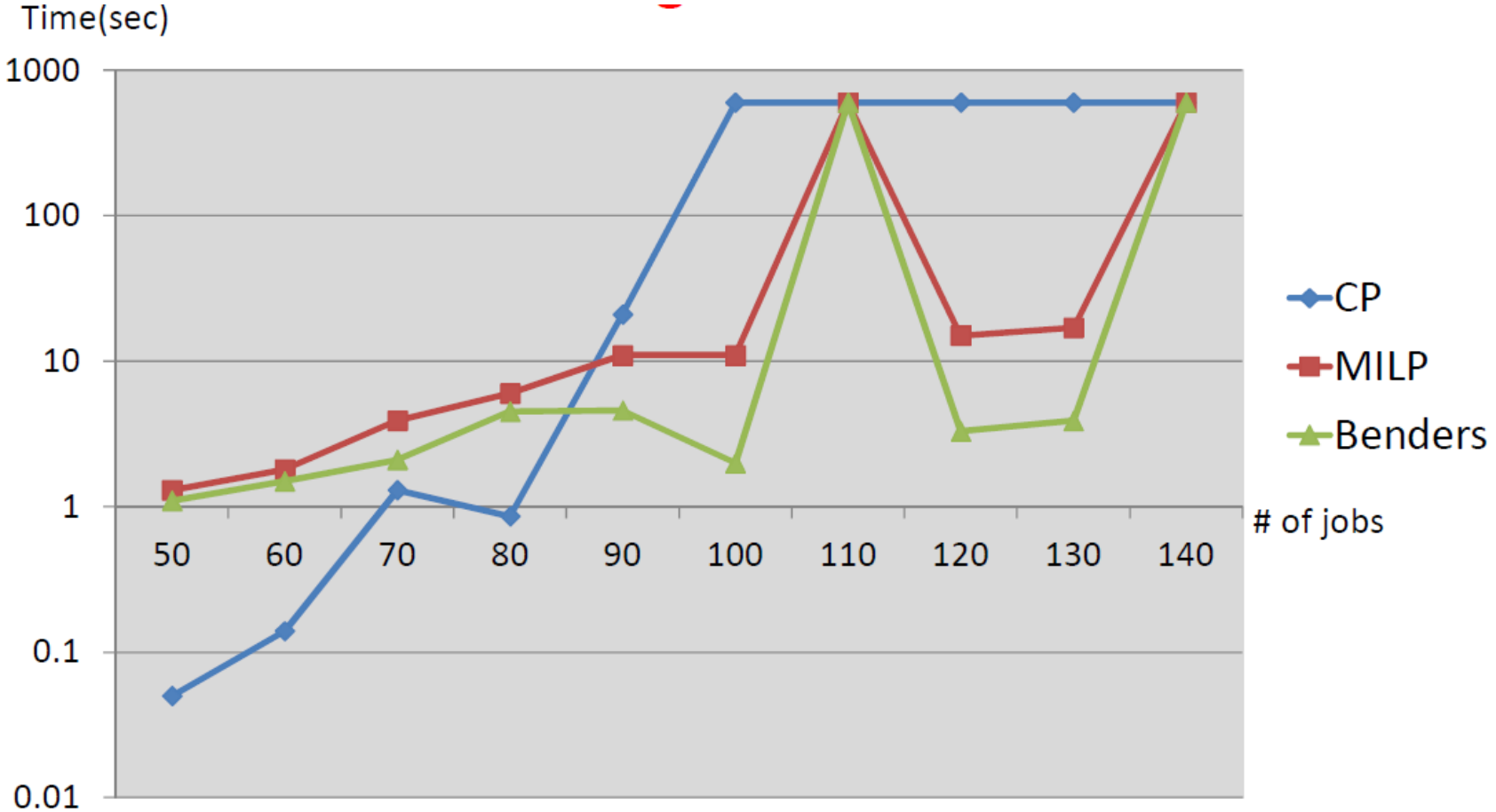
Segmented problem computational results

Min tardiness – Wide time windows (individual instances)



Segmented problem computational results

Min tardiness – Tight time windows (individual instances)



Segmented problem

Computational results – 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.

	Feasibility			Makespan			Tardiness		
Jobs	CP	MILP	Bndrs	CP	MILP	Bndrs	CP	MILP	Bndrs
60	0.1	14	1.9	60	7.7	6.4	0.1	16	3.0
80	181*	45	2.7	420*	147	11	63*	471*	20
100	199*	58	4.3	600*	600	17	547*	177*	11
120	272*	137	4.8	600*	600	39	600*	217*	2.9
140	306*	260*	6.8	600*	432*†	33	600*	373*	5.0
160	314*	301*	8.0	600*	359*	14			
180	600*	350*†	4.8	600*	557*†	5.3			
200	600*	†	5.8	600*	600*†	6.6			

*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.

Segmented problem

Computational results – wide time windows

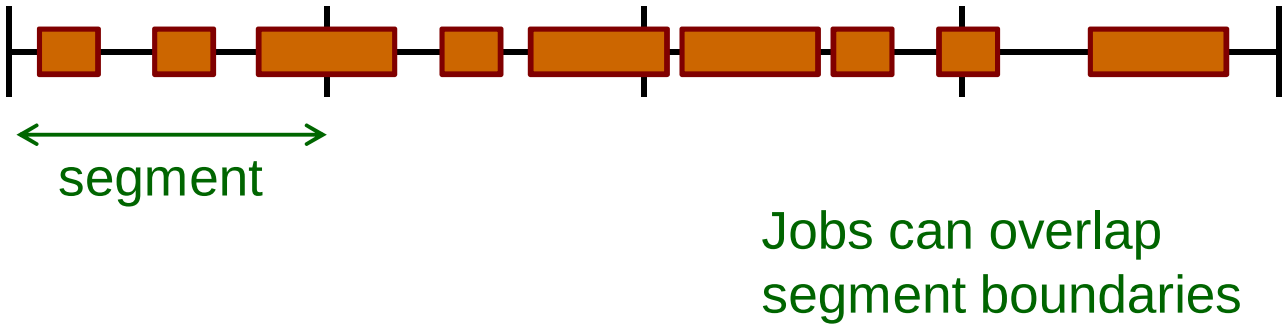
Table 5: Average computation times in seconds for the segmented problem with wide time windows. The number of segments is 10% the number of jobs. Ten instances of each size are solved.

	Feasibility			Makespan			Tardiness		
Jobs	CP	MILP	Bndrs	CP	MILP	Bndrs	CP	MILP	Bndrs
60	0.05	12	1.9	0.2	16	5.8	0.2	8.0	2.3
80	0.28	22	2.5	180*	59	9.0	1.5	94	3.7
100	0.14	37	3.8	360*	403*	14	79*	594*	85*
120	0.13	61	5.0	540*	600*	25	600*	251*	183*
140	61*	175	7.0	600*	600*	107	600*	160*	4.3
160	540*	216*	4.8	600*	562*	157			
180	600*	375*†	4.5	600*	535*	10			
200	600*	†	5.5	600*	560*	6.9			

*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.

Unsegmented problem

- Master problem is more complicated.
  - Jobs can overlap two or more segments.
  - Master problem variables must keep track of this.
- Benders cuts more sophisticated.



Unsegmented problem

- Master problem:

$$\sum_{i \in I} y_{ij} \geq 1, \quad j \in J$$

$$y_{ij} = y_{ij0} + y_{ij1} + y_{ij2} + y_{ij3}, \quad i \in I, j \in J$$

$$\sum_{j \in J} y_{ij1} \leq 1, \quad \sum_{j \in J} y_{ij2} \leq 1, \quad \sum_{j \in J} y_{ij3} \leq 1, \quad i \in I$$

$$y_{ij1} \leq y_{i-1,j,2} + y_{i-1,j,3}, \quad i \in I, i > 1, \ j \in J$$

$$y_{ij2} \leq y_{i+1,j,1} + y_{i+1,j,3}, \quad i \in I, i < n, \ j \in J$$

$$y_{ij3} \leq y_{i-1,j,3} + y_{i-1,j,2}, \quad i \in I, i > 1, \ j \in J$$

$$y_{ij3} \leq y_{i+1,j,3} + y_{i+1,j,1}, \quad i \in I, i < n, \ j \in J$$

$$\sum_{i \in I} y_{ij0} \leq 1, \quad \sum_{i \in I} y_{ij1} \leq 1, \quad \sum_{i \in I} y_{ij2} \leq 1, \quad j \in J$$

$$y_{1j1} = y_{1j3} = y_{nj2} = y_{nj3} = 0, \quad j \in J$$

$$\sum_{i \in I} y_{ij3} \leq \left\lfloor \frac{p_j}{a_{i+1} - a_i} \right\rfloor, \quad j \in J$$

$$y_{ij}, y_{ij0}, y_{ij1}, y_{ij2}, y_{ij3} \in \{0, 1\}, \quad i \in I, j \in J$$

$$x_{ij1} \leq p_j y_{ij1}, \quad x_{ij2} \leq p_j y_{ij2}$$

$$x_{ij} = p_j y_{ij0} + x_{ij1} + x_{ij2} + (a_{i+1} - a_i) y_{ij3}$$

$$x_{ij1}, x_{ij2} \geq 0$$

y_{ijk} variables keep track of whether job j starts, finishes, or runs entirely in segment i.

x_{ijk} variables keep track of how long a partial job j runs in segment i.

Unsegmented problem

- Cuts for min cost problem.
- Subproblem is a feasibility problem. We generate a cut if it is infeasible.

Case 1: No partial jobs in segment i. Use simple nogood cut

$$\sum_{j \in J_{io}} (1 - y_{ij0}) \geq 1$$

Unsegmented problem

- Cuts for min cost problem.
- Subproblem is a feasibility problem. We generate a cut if it is infeasible.

Case 2: There is a partial job j_1 only at the start of segment i. Maximize the time job j_1 can run in this segment, rather than fixing it to the time in solution of master problem

Case 2a: This modified problem is still infeasible. Use nogood cut

$$\sum_{j \in J_{io}} (1 - y_{ij0}) \geq 1$$

Unsegmented problem

- Cuts for min cost problem.
- Subproblem is a feasibility problem. We generate a cut if it is infeasible.

Case 2: There is a partial job j_1 only at the start of segment i. Maximize the time job j_1 can run in this segment, rather than fixing it to the time in solution of master problem

Case 2b: Max time is 0. Must remove job j_1 or another job.

$$(1-y_{ij_1 1}) + \sum_{j \in J_{io}} (1-y_{ij0}) \geq 1$$

Unsegmented problem

- Cuts for min cost problem.
- Subproblem is a feasibility problem. We generate a cut if it is infeasible.

Case 2: There is a partial job j_1 only at the start of segment i. Maximize the time job j_1 can run in this segment, rather than fixing it to the time in solution of master problem

Case 2c: Max time > 0. Then time is either less than given by master, or job j_1 is dropped. Use this cut:

$$\alpha_i + \sum_{j \in J_{io}} (1 - y_{ij0}) \geq 1$$

$$x_{ij_1} \leq x_{ij_1} + p_{j_1}(1 - \alpha_i) \qquad \text{where } \alpha_i \in \{0,1\}$$

Unsegmented problem

- Cuts for min cost problem.
- Subproblem is a feasibility problem. We generate a cut if it is infeasible.

Case 3: There is a partial job j_2 only at the end of segment i. Cuts are similar to Case 2.

Unsegmented problem

- Cuts for min cost problem.
- Subproblem is a feasibility problem. We generate a cut if it is infeasible.

Case 4: There are partial job j_1 at the start and j_2 at the end of segment i. Maximize the **sum** x_i^* of the times they can run in this segment.

Case 4b: $x_i^* = 0$. Use the cuts

$$(1-y_{ij_1 1}) + \sum_{j \in J_{io}} (1-y_{ij\,0}) \geq 1$$

$$(1-y_{ij_2 2}) + \sum_{j \in J_{io}} (1-y_{ij\,0}) \geq 1$$

Unsegmented problem

- Cuts for min cost problem.
- Subproblem is a feasibility problem. We generate a cut if it is infeasible.

Case 4: There are partial job j_1 at the start and j_2 at the end of segment i. Maximize the **sum** x_i^* of the times they can run in this segment.

Case 4c: $x_i^* > 0$. Use the cuts

$$\gamma_i + (1 - y_{ij_1 1}) + (1 - y_{ij_2 2}) + \sum_{j \in J_{io}} (1 - y_{ij0}) \geq 1$$

$$x_{ij_1} + x_{ij_2} \leq x_i^* + (p_{j_1} + p_{j_2})(1 - \gamma_i) \qquad \text{where } \gamma_i \in \{0,1\}$$

Unsegmented problem

- Cuts for **min makespan** problem.
- Subproblem is an optimization problem.

Case 1: There are no partial jobs in segment *i*. Use the cuts

$$M \geq M_i^* - \sum_{j \in J'_{i0}} p_j (1 - y_{ij0}) - w_i - M_i^* \sum_{j \in J''_{i0}} (1 - y_{ij0}) - M_i^* q_i$$

$$q_i \leq 1 - y_{ij0}, \quad j \in J_{i0}$$

$$w_i \leq \left(\max_{j \in J'_{i0}} \{d_j\} - \min_{j \in J'_{i0}} \{d_j\} \right) \sum_{j \in J'_{i0}} (1 - y_{ij0})$$

$$w_i \leq \left(\max_{j \in J'_{i0}} \{d_j\} - \min_{j \in J'_{i0}} \{d_j\} \right)$$

Unsegmented problem

- Cuts for **min makespan** problem.
- Subproblem is an optimization problem.

Case 2: There is a partial job at the start of segment *i*. Solve a series of problems to generate the cuts

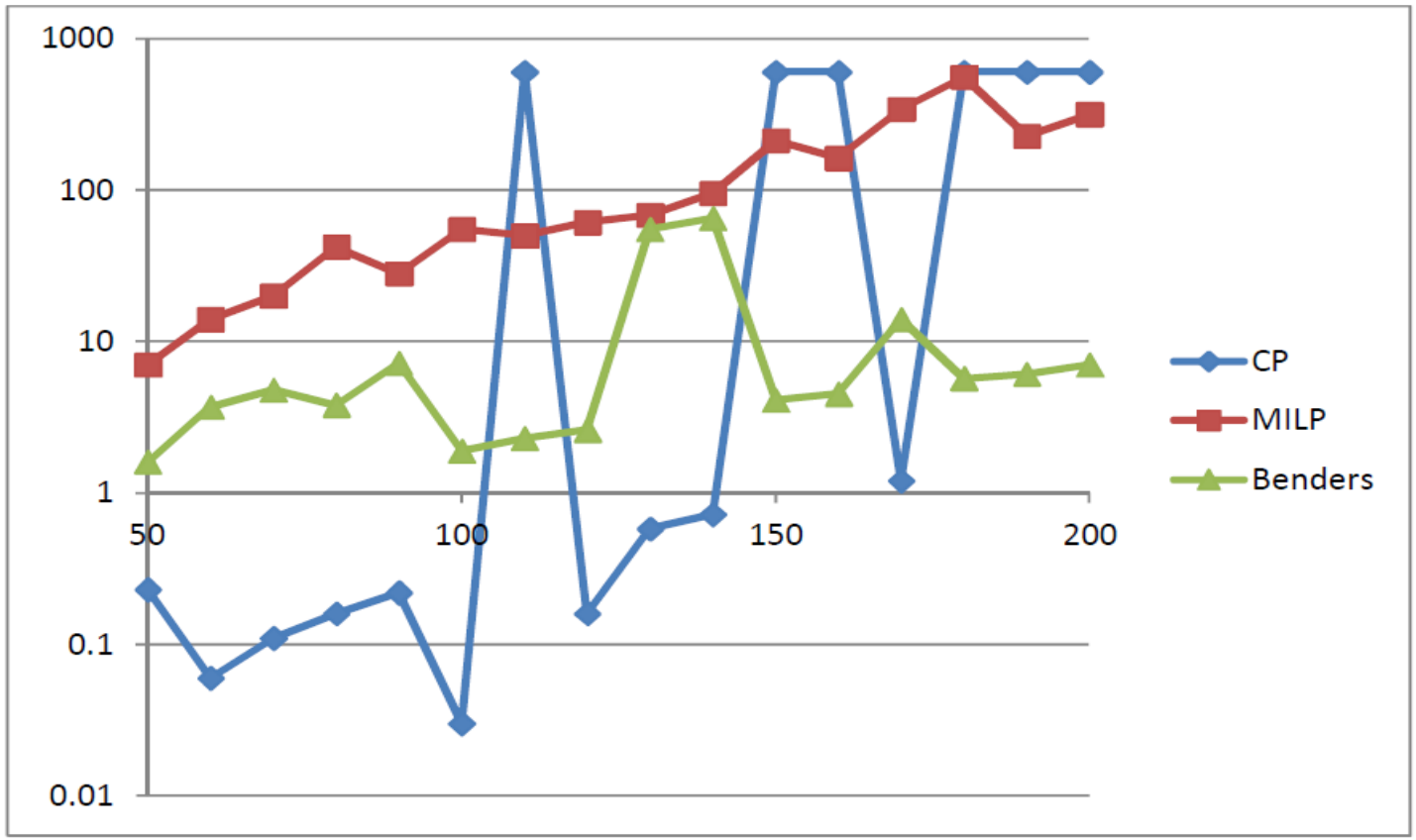
$$M \geq M_i^* (1-\eta_i) - M_i^* \sum_{j \in J''_{i0}} (1-y_{ij0})$$

$$\bar{x}_{ij_1} - x_{ij_1} + \Delta_i \leq p_{\min} + \left(\bar{x}_{ij_1} + \Delta_i - p_{\min}\right)\eta_i - \varepsilon$$

$$\bar{x}_{ij_1} - x_{ij_1} + \Delta_i \geq p_{\min} - \left(p_{j_1} - \bar{x}_{ij_1} + p_{\min} - \Delta_i\right)(1-\eta_i)$$

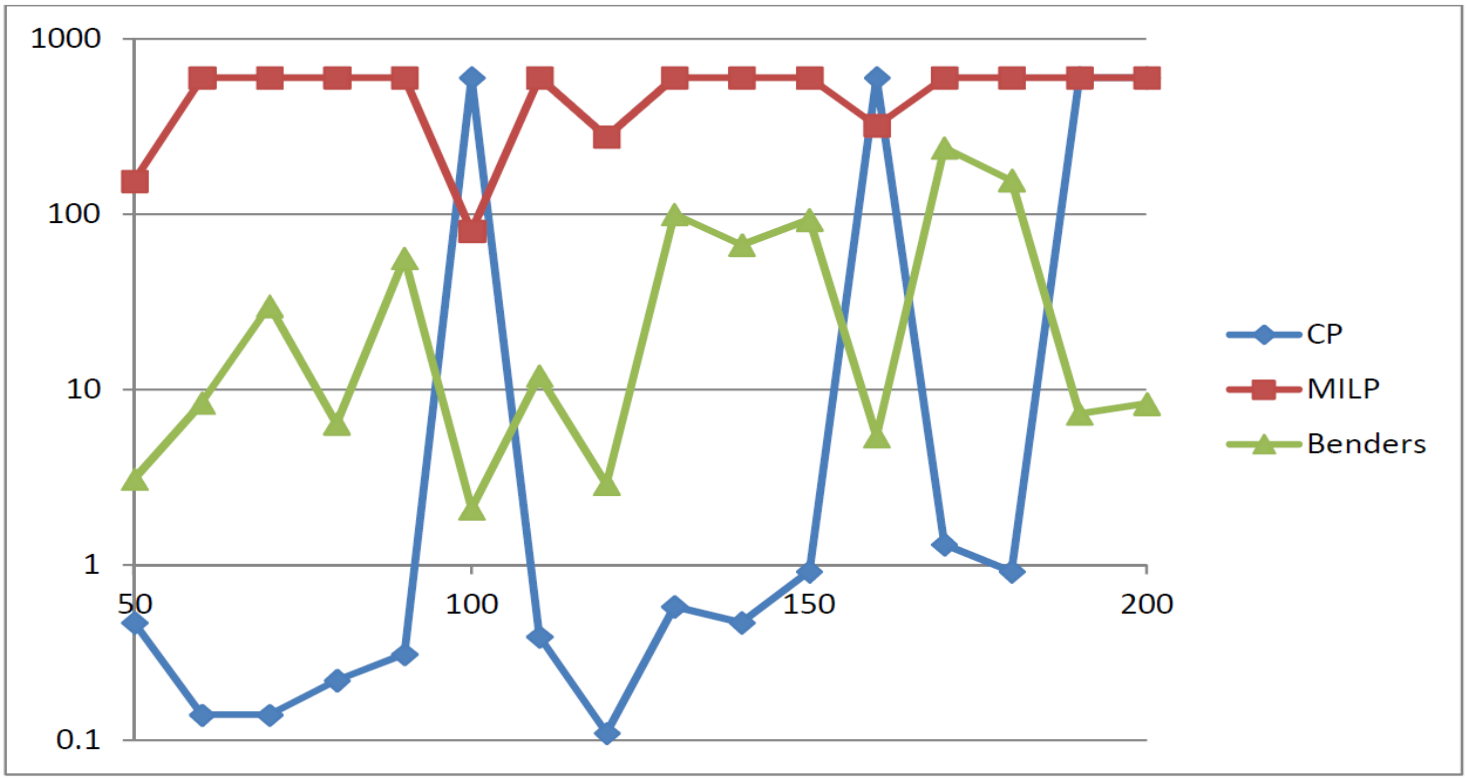
Unsegmented problem computational results

Feasibility -- individual instances



Unsegmented problem computational results

Min makespan – individual instances



Unsegmented problem

Computational results

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,

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

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

Unsegmented problem

Computational results

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

*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).

Summary of results

- Segmented problems:
  - Benders is much faster for min cost and min makespan problems.
  - Benders is somewhat faster for min tardiness problem.

Summary of results

- Segmented problems:
  - Benders is much faster for min cost and min makespan problems.
  - Benders is somewhat faster for min tardiness problem.
- Unsegmented problems:
  - Benders and CP can work together.
  - Let CP run for 1 second.
  - If it fails to solve the problem, it will probably blow up. Switch to Benders for reasonably fast solution.

Obrigado!

Vocês têm perguntas?