Integrating Solution Methods through Duality

John Hooker

Microsoft Tractability Workshop
Cambridge, UK, July 2010

Outline

- What makes a problem easy
- Exploiting structure with primal-dual-dual methods
- Simple example
- A closer look at inference duality
- Examples, with computational results

What makes a problem easy

- A problem is **easy** when we **know the answer**.

What makes a problem easy

- A problem is **easy** when we **know the answer**.
  - An **epistemic** concept of tractability.
  - Tractability is relative to the **solution method**.

What makes a problem easy

- A problem is **easy** when we **know the answer**.
  - An **epistemic** concept of tractability.
  - Tractability is relative to the **solution method**.

- Pigeon hole principle
  - Resolution proof – exponential
  - Cutting plane proof – polynomial
  - Counting argument – trivial

What makes a problem easy

- No need to view some problems as **inherently hard**.
  - **"Easy"** problems can be **hard** if we don't exploit structure (e.g., linear programming).
  - **"Hard"** problems can be **solved** if we exploit structure (we do this for a living).

What makes a problem easy

- No need to view some problems as **inherently hard**.
  - **"Easy"** problems can be **hard** if we don't exploit structure (e.g., linear programming).
  - **"Hard"** problems can be **solved** if we exploit structure (we do this for a living).

- A good solution methods **knows** more about the problem.
  - That is, it exploits **problem structure**.

Two ways to exploit structure

- **Inference** and **relaxation**.
  - Used throughout **optimization**
  - Provides a principle for **unifying** and **integrating** solution methods.

Two ways to exploit structure

- **Inference** and **relaxation**.
  - Used throughout **optimization**
  - Provides a principle for **unifying** and **integrating** solution methods.

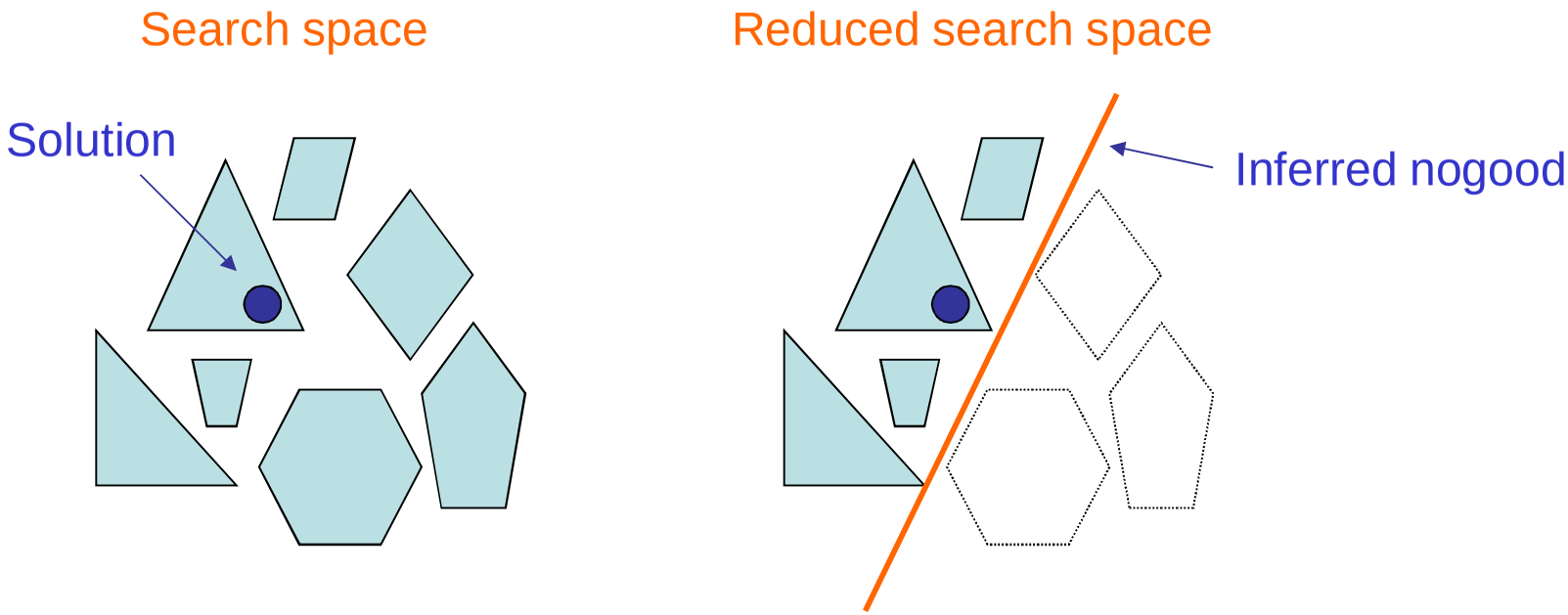
- Well-chosen inference and relaxation techniques can **accelerate solution**.
  - Techniques can be drawn from **existing methods**, which results in **integration**.
  - **New** techniques can be invented for the problem.

We can integrate:

- **MIP**
  - Mixed integer programming.
- **CP**
  - Constraint programming.
- **GO**
  - Global optimization
  - Nonlinear, nonconvex
  - Discrete and/or continuous variables
- **SAT**
  - Propositional satisfiability
- **LS**
  - Local search, metaheuristics

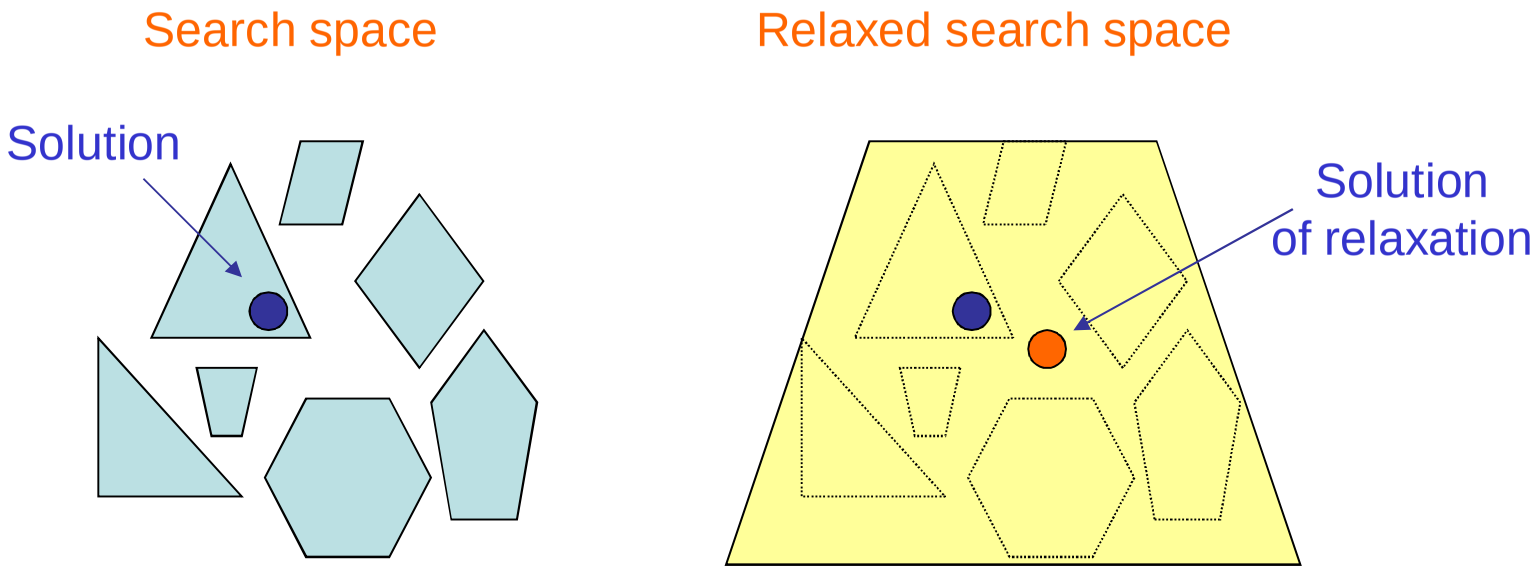
Two ways to exploit structure

- **Inference**
  - Use knowledge of problem structure to reveal **hidden information**.
  - This can exclude **unpromising** areas of the search space..



Two ways to exploit structure

- **Relaxation**
  - Use knowledge of problem structure to design a **larger but simpler** search space.
  - Solution of relaxation may be **near** solution of original problem.



Two dualities

- Duality of search and inference.
  - **Search** looks for a certificate of **feasibility**.
  - The **inference dual** looks for a certificate (proof) of **infeasibility** (or **optimality**).

- Duality of search and relaxation.
  - **Search** enumerates **restrictions** of the problem.
  - The relaxation dual enumerates (parameterized) **relaxations** of the problem.

Primal-dual-dual methods

- Primal methods.
  - Enumerate possible **solutions** (in general, problem restrictions).
- Dual methods.
  - Enumerate **proofs** or **relaxations**.
- Primal-dual methods.
  - Solve the **primal** and a **dual** simultaneously.
- Primal-dual-dual methods.
  - Solve the primal and **both duals** simultaneously.
  - Strategy of **most successful optimization methods**.

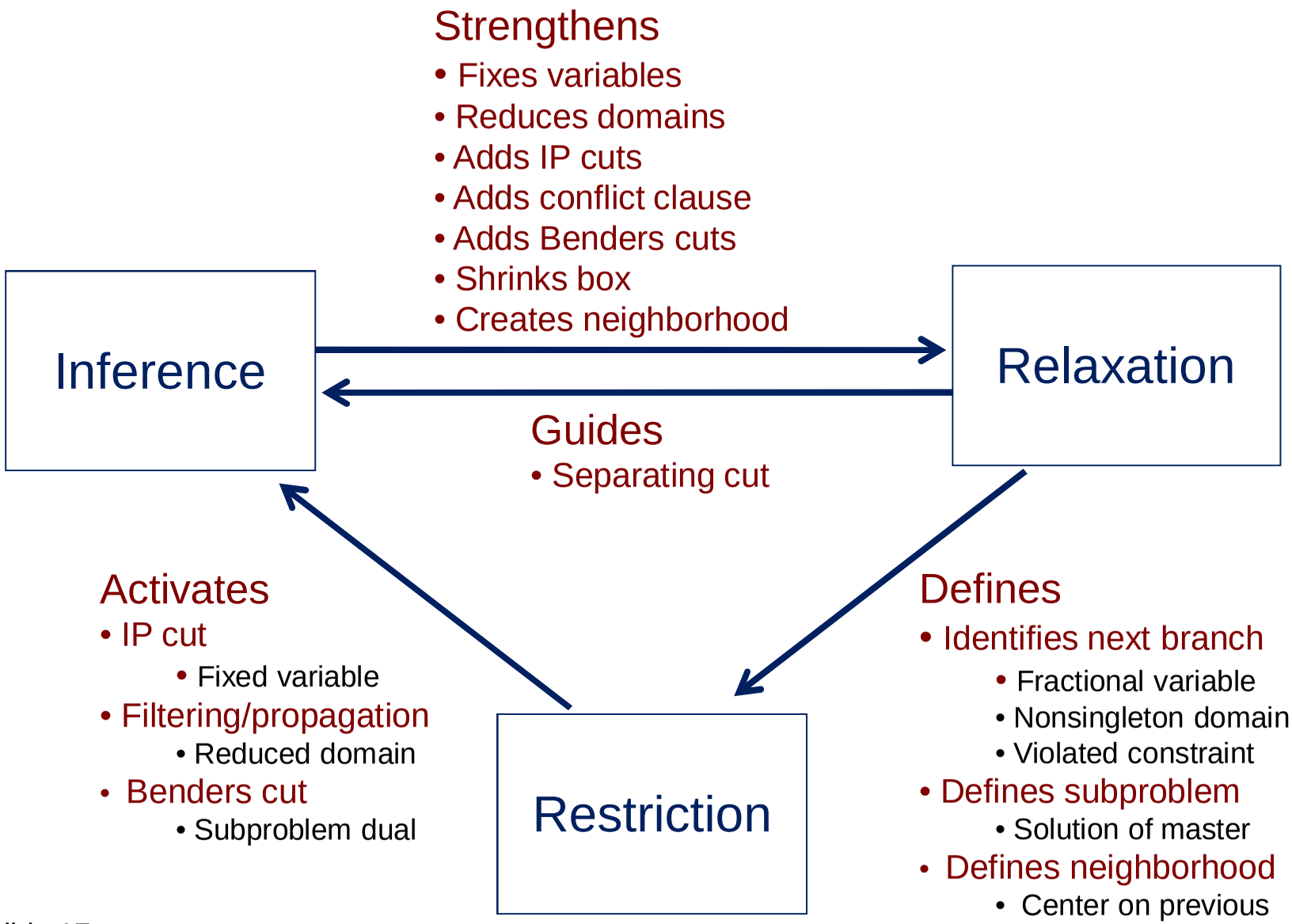
Classical solution methods

- CP solver
  - **Search:** Branching
  - **Inference:** Flltering
  - **Relaxation:** Domain store
- MILP solver
  - **Search:** Branching
  - **Inference:** Cutting planes, presolve, reduced cost variable fixing
  - **Relaxation:** LP
- Benders
  - **Search:** Enumerate subproblems.
  - **Inference:** Benders cuts
  - **Relaxation:** Master problem

Classical solution methods

- Global optimization
  - **Search:** Enumerate boxes
  - **Inference:** Domain reduction, dual-based variable bounding
  - **Relaxation:** Convexification
- SAT
  - **Search:** Branching, dynamic backtracking, etc.
  - **Inference:** Conflict clauses
  - **Relaxation:** Same as restriction
- Local search
  - **Search:** Enumerate neighborhoods.
  - **Inference:** Tabu list, etc.
  - **Relaxation:** Same as restriction

Interaction



Simple example: Freight transfer

- Transport 42 tons of freight using 8 trucks, which come in 4 sizes…

Truck size	Number available	Capacity (tons)	Cost per truck
1	3	7	90
2	3	5	60
3	3	4	50
4	3	3	40

Problem formulation

Number of trucks of type 1

$$\min 90x_1 + 60x_2 + 50x_3 + 40x_4$$

$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$$

$$x_1 + x_2 + x_3 + x_4 \leq 8$$

$$x_i \in \{0,1,2,3\}$$

Truck type	Number available	Capacity (tons)	Cost per truck
1	3	7	90
2	3	5	60
3	3	4	50
4	3	3	40

Inference: Bounds propagation

Standard CP technique

$$\min\ 90x_1 + 60x_2 + 50x_3 + 40x_4$$
$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$$
$$x_1 + x_2 + x_3 + x_4 \leq 8$$
$$x_i \in \{0,1,2,3\}$$

$$x_1 \geq \left\lceil \frac{42 - 5 \cdot 3 - 4 \cdot 3 - 3 \cdot 3}{7} \right\rceil = 1$$

Bounds propagation

Standard CP technique

$$\min 90x_1 + 60x_2 + 50x_3 + 40x_4$$

$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$$

$$x_1 + x_2 + x_3 + x_4 \leq 8$$

$$x_1 \in \{1,2,3\}, \quad x_2, x_3, x_4 \in \{0,1,2,3\}$$

Reduced domain

$$x_1 \geq \left\lceil \frac{42 - 5 \cdot 3 - 4 \cdot 3 - 3 \cdot 3}{7} \right\rceil = 1$$

Bounds propagation

Standard CP technique

$$\min 90x_1 + 60x_2 + 50x_3 + 40x_4$$

$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$$

$$x_1 + x_2 + x_3 + x_4 \leq 8$$

$$x_1 \in \{1,2,3\}, \quad x_2, x_3, x_4 \in \{0,1,2,3\}$$

Reduced domain

$$x_1 \geq \left\lceil \frac{42 - 5 \cdot 3 - 4 \cdot 3 - 3 \cdot 3}{7} \right\rceil = 1$$

In general:

Bounds propagation aims for **bounds consistency**

Domain filtering aims for **hyperarc consistency** (GAC)

Relaxation: Linear programming

$$\min\ 90x_1 + 60x_2 + 50x_3 + 40x_4$$
$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$$
$$x_1 + x_2 + x_3 + x_4 \leq 8$$
$$0 \leq x_i \leq 3, \quad x_1 \geq 1$$

Replace domains with bounds

This is a linear programming problem, which has a **simplified search space** (polyhedron).

Its optimal value provides a **lower bound** on optimal value of original problem.

The optimal solution may be **close to, or equal to,** an optimal solution of the original problem.

Inference: cutting planes (valid inequalities)

$$\min\ 90x_1 + 60x_2 + 50x_3 + 40x_4$$
$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$$
$$x_1 + x_2 + x_3 + x_4 \leq 8$$
$$0 \leq x_i \leq 3, \quad x_1 \geq 1$$

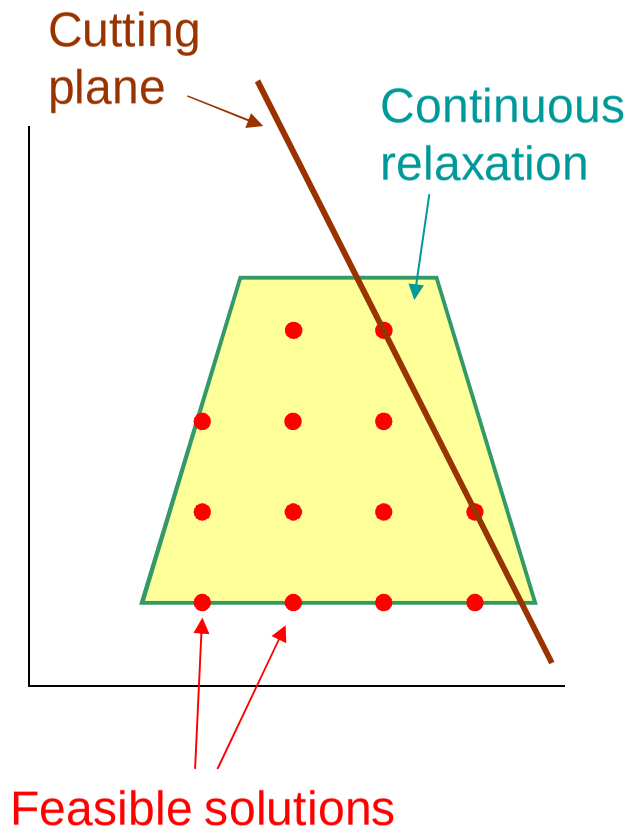
We can create a **tighter** relaxation with the addition of **cutting planes**.

Inference: cutting planes (valid inequalities)

$$\min\ 90x_1 + 60x_2 + 50x_3 + 40x_4$$
$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$$
$$x_1 + x_2 + x_3 + x_4 \leq 8$$
$$0 \leq x_i \leq 3, \quad x_1 \geq 1$$

A cutting plane excludes ("**cut off**") solutions of the continuous relaxation…

but no feasible solutions of original problem.



Inference: cutting planes (valid inequalities)

$$\min\ 90x_1 + 60x_2 + 50x_3 + 40x_4$$

$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$$

$$x_1 + x_2 + x_3 + x_4 \leq 8$$

$$0 \leq x_i \leq 3, \quad x_1 \geq 1$$

{1,2} is a **packing**

…because $7x_1 + 5x_2$ alone cannot satisfy the inequality, even with $x_1 = x_2 = 3$.

Inference: cutting planes (valid inequalities)

$$\min\ 90x_1 + 60x_2 + 50x_3 + 40x_4$$

$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$$

$$x_1 + x_2 + x_3 + x_4 \leq 8$$

$$0 \leq x_i \leq 3, \quad x_1 \geq 1$$

{1,2} is a **packing**

So, $4x_3 + 3x_4 \geq 42 - (7 \cdot 3 + 5 \cdot 3)$

Knapsack cut

which implies

$$x_3 + x_4 \geq \left\lceil \frac{42 - (7 \cdot 3 + 5 \cdot 3)}{\max\{4,3\}} \right\rceil = 2$$

Cutting planes (valid inequalities)

$$\min\ 90x_1 + 60x_2 + 50x_3 + 40x_4$$

$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$$

$$x_1 + x_2 + x_3 + x_4 \leq 8$$

$$0 \leq x_i \leq 3, \quad x_1 \geq 1$$

Maximal Packings	**Knapsack cuts**
{1,2}	$x_3 + x_4 \geq 2$
{1,3}	$x_2 + x_4 \geq 2$
{1,4}	$x_2 + x_3 \geq 3$

Knapsack cuts corresponding to nonmaximal packings can be nonredundant.

Continuous relaxation with cuts

$$\min\ 90x_1 + 60x_2 + 50x_3 + 40x_4$$

$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$$

$$x_1 + x_2 + x_3 + x_4 \leq 8$$

$$0 \leq x_i \leq 3, \quad x_1 \geq 1$$

$$x_3 + x_4 \geq 2$$

$$x_2 + x_4 \geq 2$$

$$x_2 + x_3 \geq 3$$

Knapsack cuts

Optimal value of 523.3 is a lower bound on optimal value of original problem.

Primal-dual-dual method

Search: branching

Inference: Bounds propagation, cutting planes

Relaxation: LP, domain store

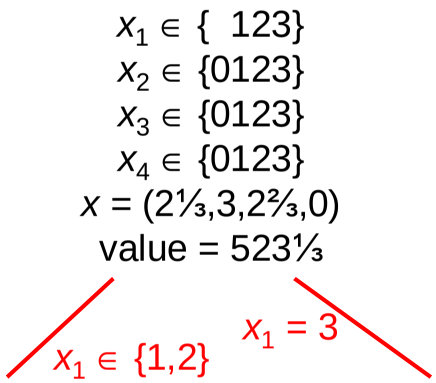
Primal-dual-dual method

$x_1 \in \{\ 123\}$
$x_2 \in \{0123\}$
$x_3 \in \{0123\}$
$x_4 \in \{0123\}$
$x = (2\frac{1}{3},3,2\frac{2}{3},0)$
value = $523\frac{1}{3}$

Propagate bounds and solve relaxation.

Since solution of relaxation is infeasible, branch.

Primal-dual-dual method

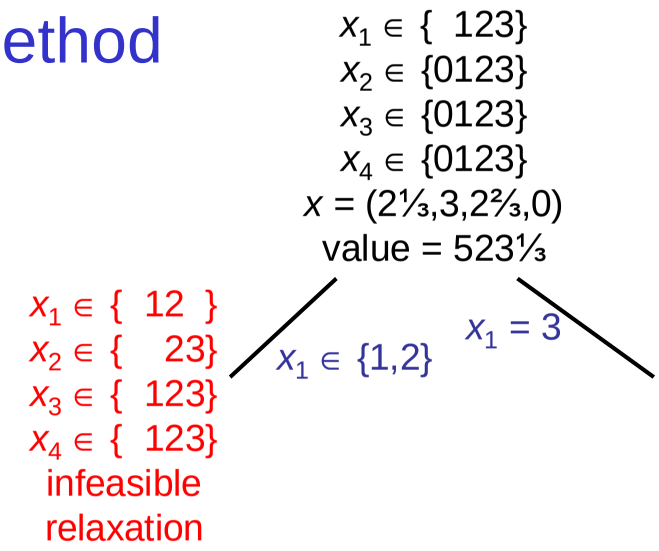


Branch on a variable with nonintegral value in the relaxation.

Primal-dual-dual method

Propagate bounds and solve relaxation.

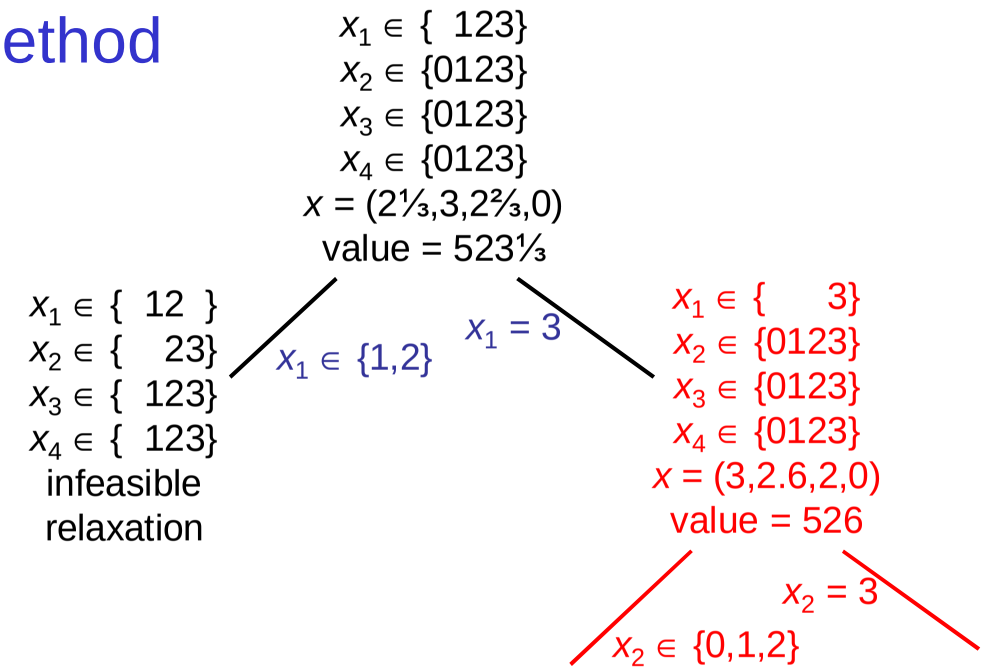
Since relaxation is infeasible, backtrack.



Primal-dual-dual method

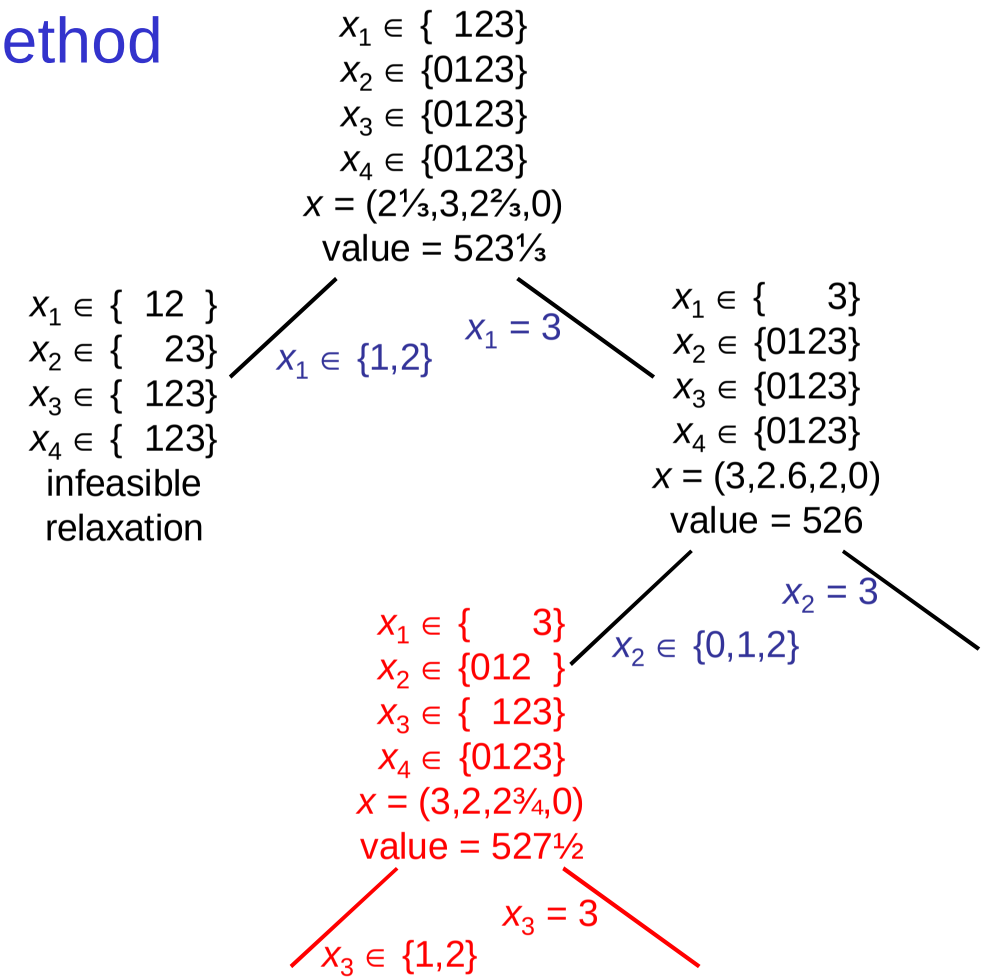
Propagate bounds and solve relaxation.

Branch on nonintegral variable.



Primal-dual-dual method

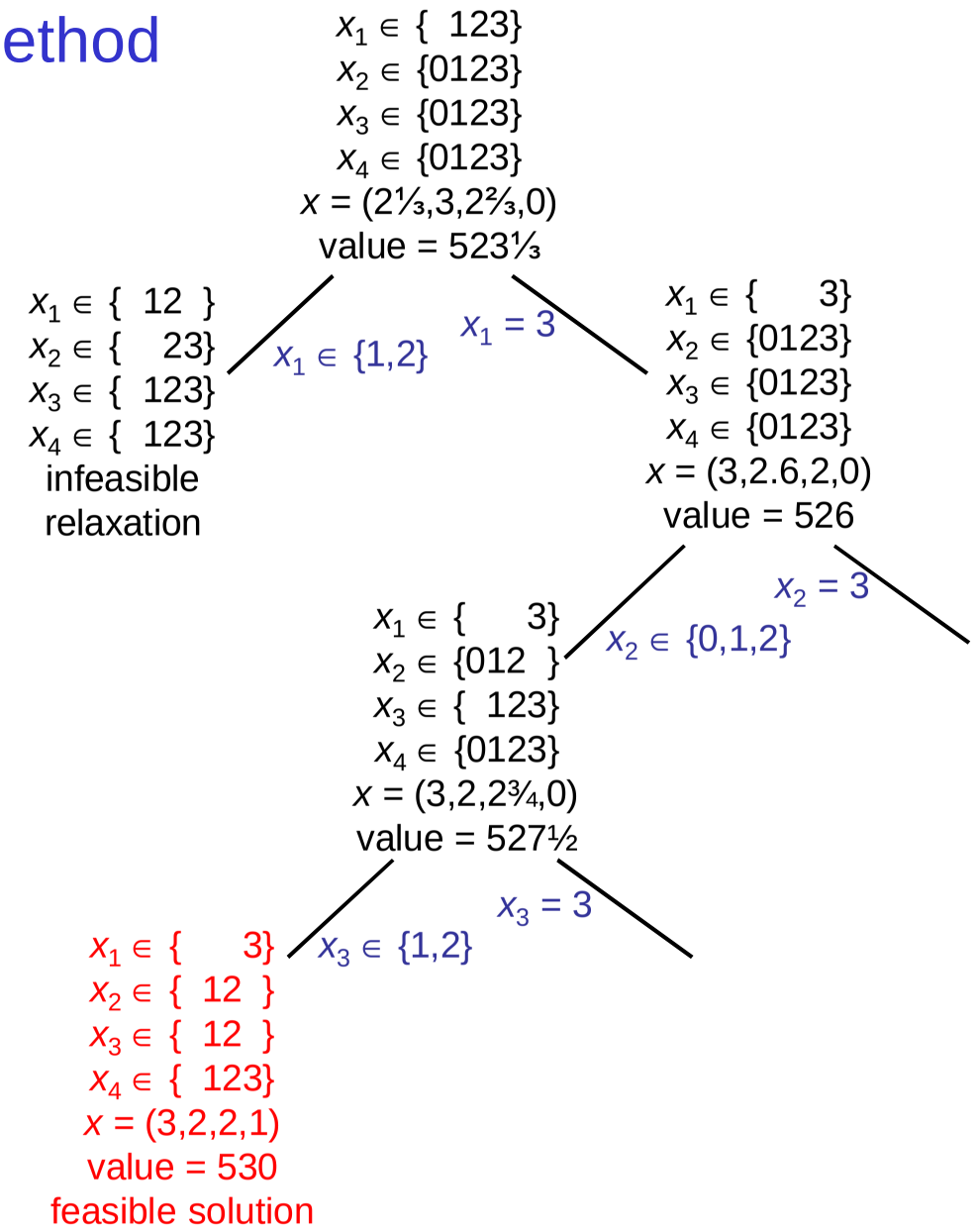
Branch again.



Primal-dual-dual method

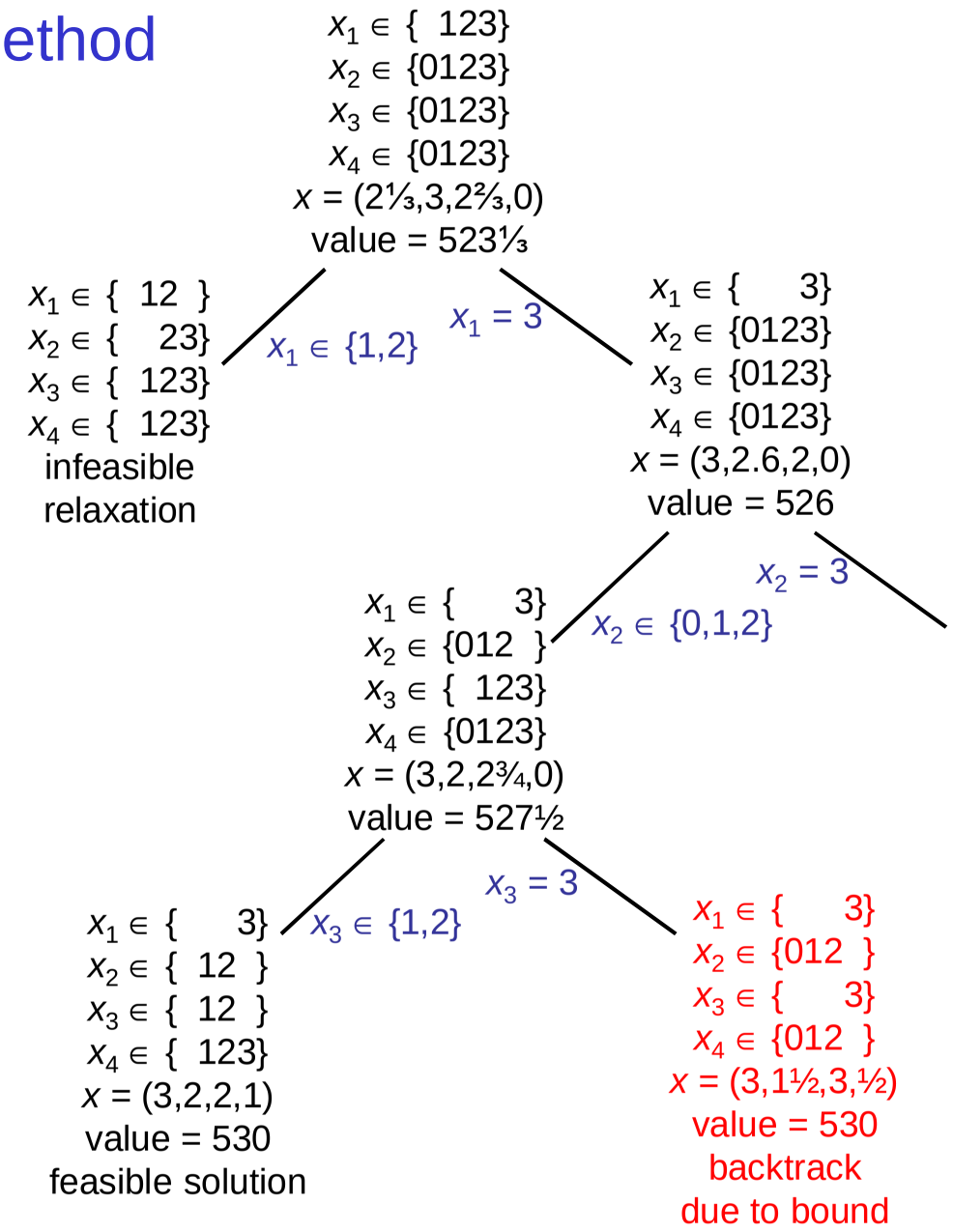
Solution of relaxation is integral and therefore feasible in the original problem.

This becomes the **incumbent solution**.



Primal-dual-dual method

Solution is nonintegral, but we can backtrack because value of relaxation is no better than incumbent solution.



Primal-dual-dual method

Another feasible solution found.

No better than incumbent solution, which is optimal because search has finished.

A closer look at inference duality

- **Formal definition** of inference dual
  - LP, Lagrangean, surrogate duals are special cases

A closer look at inference duality

- **Formal definition** of inference dual
  - LP, Lagrangean, surrogate duals are special cases

- Nogood-based search
  - Solution of inference dual provides **nogoods** to guide search.
  - **Benders cuts** and **conflict clauses** in SAT are examples.

A closer look at inference duality

- **Formal definition** of inference dual
  - LP, Lagrangean, surrogate duals are special cases
- Nogood-based search
  - Solution of inference dual provides **nogoods** to guide search.
  - **Benders cuts** and **conflict clauses** in SAT are examples.
- Example: SAT
  - DPLL
  - DPLL + conflict clauses
  - Partial order dynamic backtracking

A closer look at inference duality

- Recognition of inference dual structure can lead to massive speedups.
  - For example, **logic-based Benders** decomposition
  - As in **machine scheduling** example to follow.

Inference duality

- **All optimization duals** are inference duals
  - Also **relaxation duals**
- Solution of inference dual is **proof** of optimality
  - Primal in **co-NP** when dual is in **NP**
- Postoptimality analysis
  - Result of altering premises of proof

Inference dual

Primal

$$\min f(x)$$
$$x \in S$$

Feasible set

Dual

$$\max v$$
$$x \in S \overset{P}{\Rightarrow} v \geq f(x)$$
$$P \in \mathcal{P}$$

Follows using proof P

Family of proofs (inference method)

- Dual is defined relative to an **inference method**.
- Strong duality applies if inference method is **complete**.

Example: LP dual

Primal

$$\min cx$$
$$Ax \geq b$$
$$x \geq 0$$

Dual

$$\max v \qquad = \max \lambda b$$
$$\left\{ \begin{array}{l} Ax \geq b \\ x \geq 0 \end{array} \right\} \overset{P}{\Rightarrow} cx \geq v \qquad \lambda A \leq c$$
$$\lambda \geq 0$$

$P \in \mathcal{P} \longleftarrow$ Nonnegative linear combination + domination

$Ax \geq b \overset{x \geq 0}{\Rightarrow} cx \geq v$ iff $\lambda Ax \geq \lambda b$ dominates $cx \geq v$

for some $\lambda \geq 0$

$\lambda A \leq c$ and $\lambda b \geq v$

- Inference method is **complete** (assuming feasibility) due to Farkas Lemma.
- So we have **strong duality** (assuming feasibility).

Example: Lagrangean dual

Primal

$$\min f(x)$$
$$g(x) \geq 0$$
$$x \in S$$

Dual

$$\max v$$
$$g(x) \geq b \overset{P}{\Rightarrow} f(x) \geq v$$
$$P \in \mathcal{P} \longleftarrow$$ Nonnegative linear combination + domination

$$= \max_{\lambda \geq 0} \min_{x \in S} \{f(x) - \lambda g(x)\}$$

$$g(x) \geq 0 \overset{x \in S}{\Rightarrow} f(x) \geq v \quad \text{iff} \quad \lambda g(x) \geq 0 \text{ dominates } f(x) - v \geq 0$$

for some $\lambda \geq 0$

$\lambda g(x) \leq f(x) - v$ for all $x \in S$

That is, $v \leq f(x) - \lambda g(x)$ for all $x \in S$

Or $v \leq \min_{x \in S} \{f(x) - \lambda g(x)\}$

- Inference method is **incomplete**

Example: Surrogate dual

Primal

$$\min f(x)$$
$$g(x) \geq 0$$
$$x \in S$$

Dual

$$\max v = \max_{\lambda \geq 0} \min_{x \in S} \{ f(x) \mid \lambda g(x) \geq 0 \}$$

$$g(x) \geq b \overset{P}{\Rightarrow} f(x) \geq v$$

$P \in \mathcal{P}$ ⟵ Nonnegative linear combination + implication

$$g(x) \geq 0 \overset{x \in S}{\Rightarrow} f(x) \geq v \quad \text{iff} \quad \lambda g(x) \geq 0 \text{ implies } f(x) \geq v \text{ for some } \lambda \geq 0$$

Any $x \in S$ with $\lambda g(x) \geq 0$ satisfies $f(x) \geq v$

So, $\min \{ f(x) \mid \lambda g(x) \leq 0, x \in S \} \geq v$

- Inference method is **incomplete**

Nogood-based search

- All search is nogood-based search
  - Each solution examined generates a nogood.
  - Next solution must satisfy current nogood set.

- Nogoods are derived by solving inference dual of the subproblem.
  - Subproblem is normally defined by fixing variables to current values in the search.

Example: Logic-based Benders

Partition variables x,y and search over values of x

$$\min \ f(x,y)$$
$$(x,y) \in S$$

Subproblem results from fixing x

$$\min \ f(\bar{x},y)$$
$$(\bar{x},y) \in S$$

Let proof P be solution of subproblem dual for $x = \bar{x}$

Let $B(P,x)$ be lower bound obtained by P for given x.

Add Benders cut $v \geq B(P,x)$ to **master problem**: $\min v$

Solve master problem for next $\bar{x}$

Benders cuts

Classical Benders

Partition variables x,y and search over values of x

$$\min \ f(x) + cy$$
$$g(x) + Ay \geq b$$
$$x \in D_x, \ y \geq 0$$

Subproblem results from fixing x

$$\min \ f(x^k) + cy$$
$$Ay \geq b - g(x^k) \quad (\lambda)$$
$$y \geq 0$$

Let proof λ be solution of subproblem dual for $x = \bar{x}$

Let $B(\lambda, x) = f(x) + \lambda(b - g(x))$ be bound obtained by λ for given x.

Add Benders cut $v \geq B(\lambda, x)$ to **master problem**: $\min v$

Solve master problem for next $\bar{x}$ Benders cuts

Example: SAT

- Solve SAT by chronological backtracking + unit clause rule = DPLL.
  - Chronological = fixed branching order.

Example: SAT

- Solve SAT by chronological backtracking + unit clause rule = DPLL.
  - Chronological = fixed branching order.

- To get nogood, solve **inference dual** at current node.
  - Solve dual with unit clause rule
  - Nogood identifies branches that create infeasibility.
  - Simplest scheme: nogood rules out path to current leaf node.

Example: SAT

- Solve SAT by chronological backtracking + unit clause rule = DPLL.
  - Chronological = fixed branching order.

- To get nogood, solve **inference dual** at current node.
  - Solve dual with unit clause rule
  - Nogood identifies branches that create infeasibility.
  - Simplest scheme: nogood rules out path to current leaf node.

- Process nogood set with **parallel resolution**
  - Nogood set is a **relaxation** of the problem.

Example: SAT

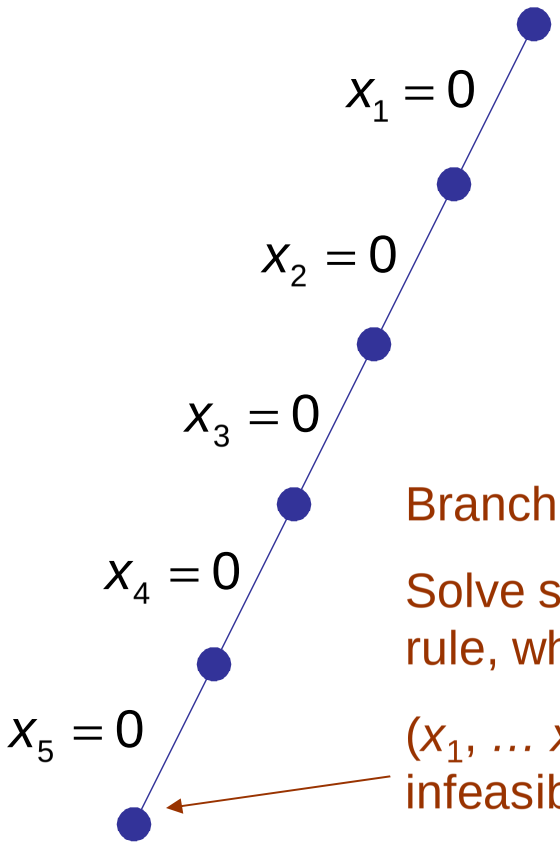
- Solve SAT by chronological backtracking + unit clause rule = DPLL.
  - Chronological = fixed branching order.
- To get nogood, solve **inference dual** at current node.
  - Solve dual with unit clause rule
  - Nogood identifies branches that create infeasibility.
  - Simplest scheme: nogood rules out path to current leaf node.
- Process nogood set with **parallel resolution**
  - Nogood set is a **relaxation** of the problem.
- Solve relaxation without branching
  - Select solution with preference for 0

The problem

Find a satisfying solution.

$$x_1 \vee x_5 \vee x_6$$
$$x_1 \vee x_5 \vee \overline{x}_6$$
$$x_2 \vee \overline{x}_5 \vee x_6$$
$$x_2 \vee \overline{x}_5 \vee \overline{x}_6$$
$$\overline{x}_1 \vee x_3 \vee x_4$$
$$\overline{x}_2 \vee x_3 \vee x_4$$
$$\overline{x}_1 \vee \overline{x}_3$$
$$\overline{x}_1 \vee \overline{x}_4$$
$$\overline{x}_2 \vee \overline{x}_3$$
$$\overline{x}_2 \vee \overline{x}_4$$

DPLL with chronological backtracking

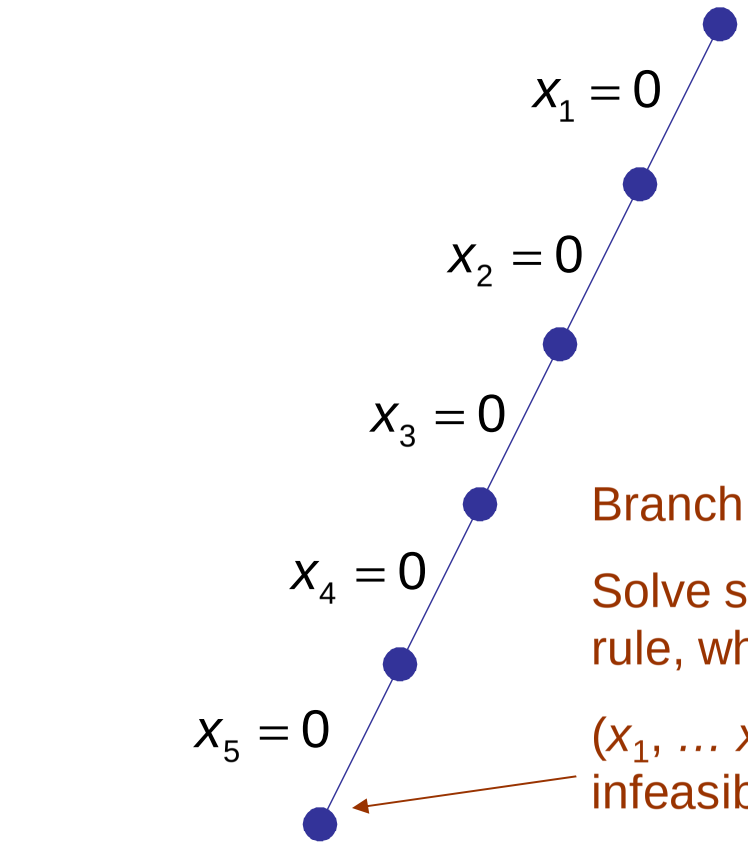


Branch to here.

Solve subproblem with unit clause rule, which proves infeasibility.

(x_1, … x_5) = (0, …, 0) creates the infeasibility.

DPLL with chronological backtracking



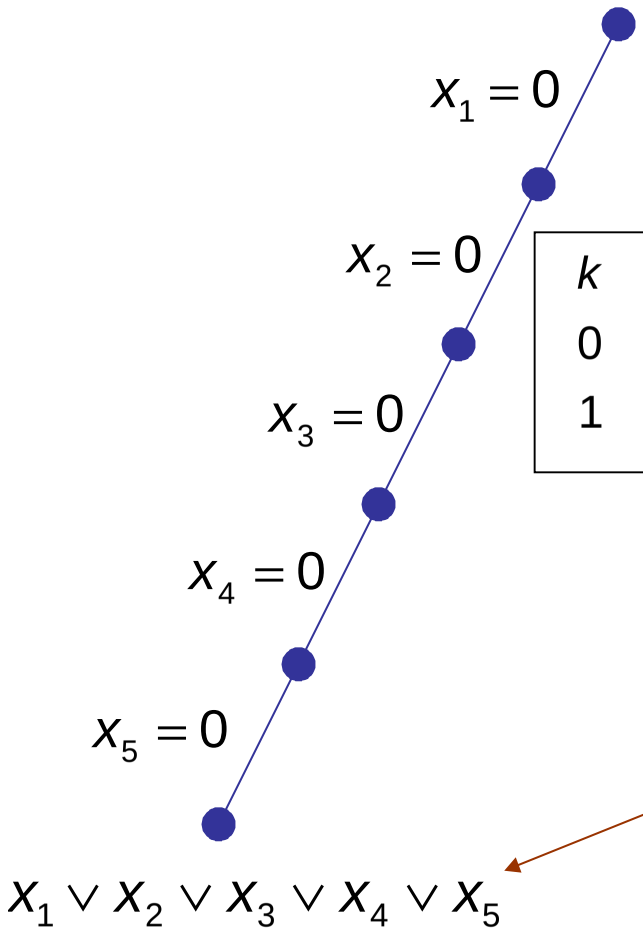
Branch to here.

Solve subproblem with unit clause rule, which proves infeasibility.

(x_1, … x_5) = (0, …, 0) creates the infeasibility.

Generate nogood.

DPLL with chronological backtracking

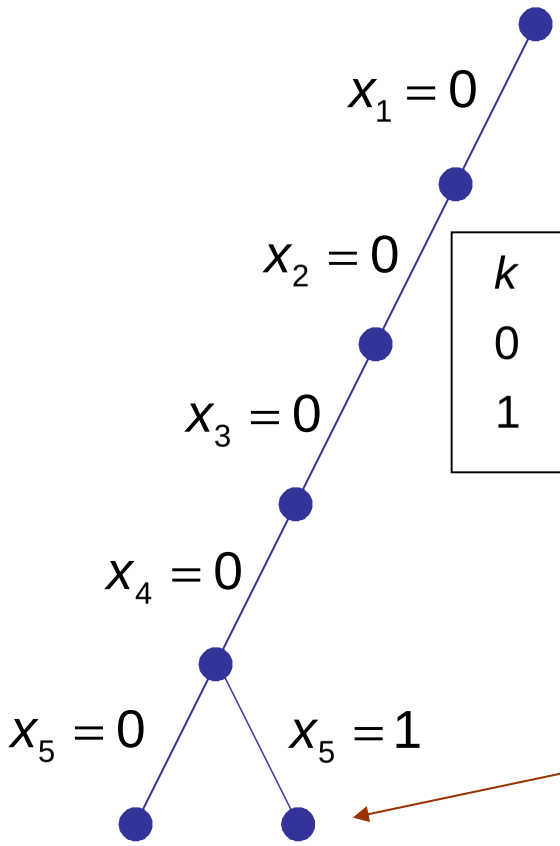


Consists of processed nogoods

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$
1	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$		

Conflict clause appears as nogood induced by solution of R_k.

DPLL with chronological backtracking

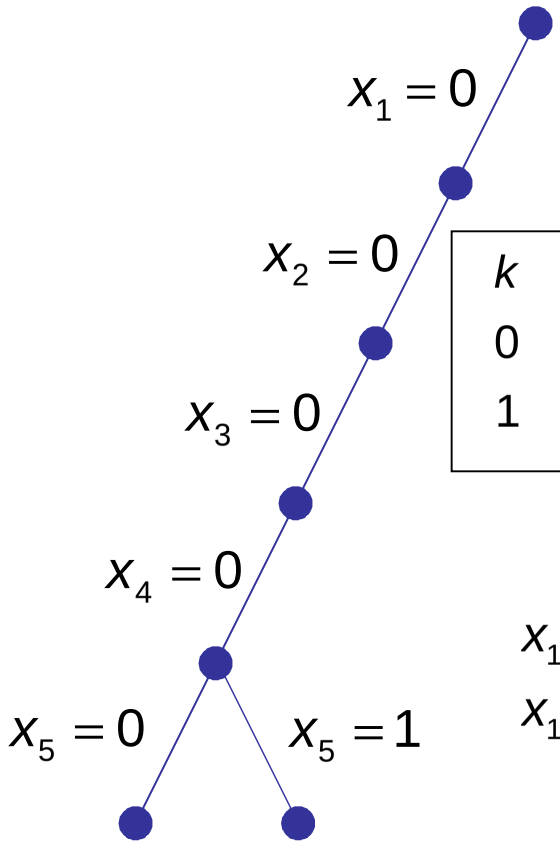


Consists of processed nogoods

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$
1	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$	$(0,0,0,0,1,\cdot)$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \overline{x}_5$

Go to solution that solves relaxation, with priority to 0

DPLL with chronological backtracking



Consists of processed nogoods

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$
1	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$	$(0,0,0,0,1,\cdot)$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \overline{x}_5$

$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$

$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \overline{x}_5$

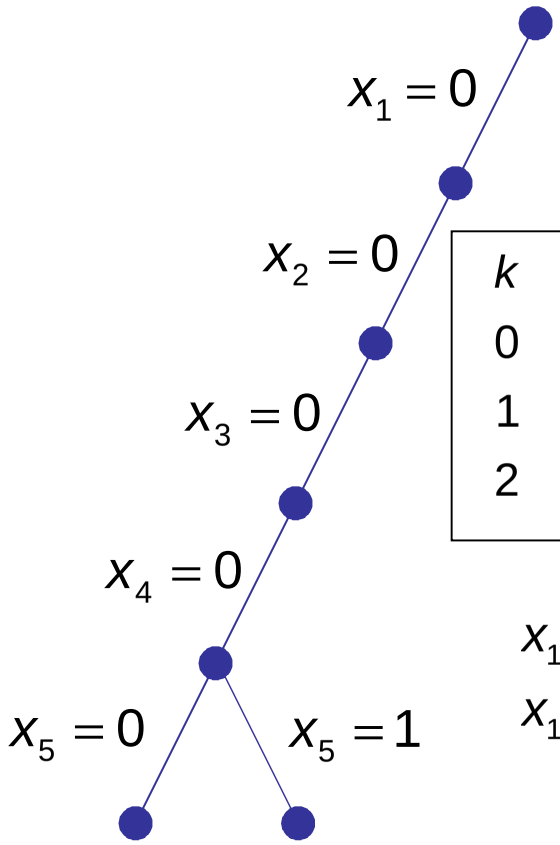
Process nogood set with **parallel resolution**

$x_1 \vee x_2 \vee x_3 \vee x_4$ **parallel-absorbs**

$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$

$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \overline{x}_5$

DPLL with chronological backtracking



Consists of processed nogoods

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$
1	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$	$(0,0,0,0,1,\cdot)$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \overline{x}_5$
2	$x_1 \vee x_2 \vee x_3 \vee x_4$		

$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$

$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \overline{x}_5$

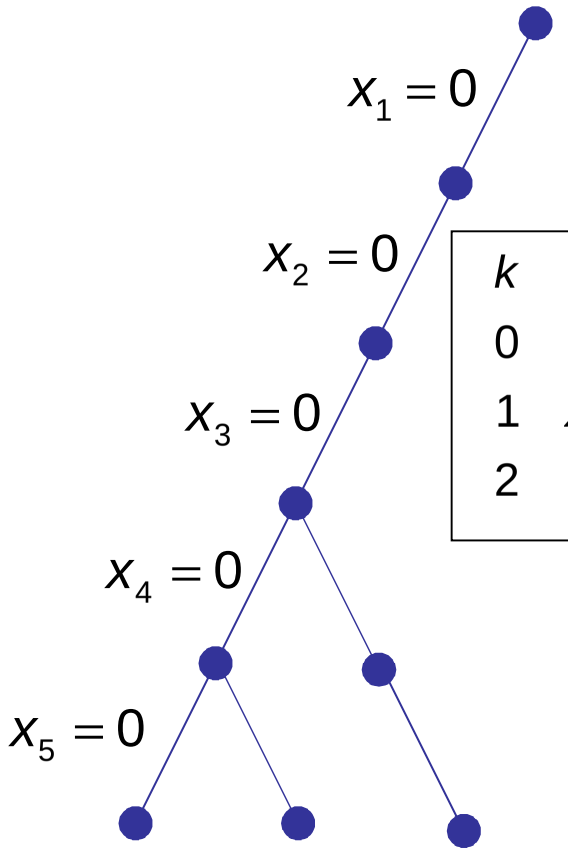
Process nogood set with **parallel resolution**

$x_1 \vee x_2 \vee x_3 \vee x_4$ **parallel-absorbs**

$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$

$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \overline{x}_5$

DPLL with chronological backtracking



Consists of processed nogoods

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$
1	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$	$(0,0,0,0,1,\cdot)$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \overline{x}_5$
2	$x_1 \vee x_2 \vee x_3 \vee x_4$	$(0,0,0,1,0,\cdot)$	

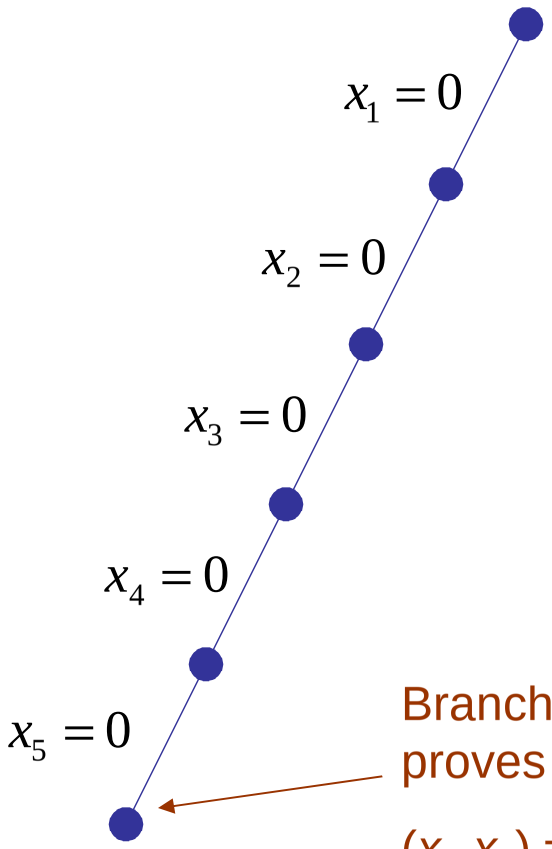
Solve relaxation again, continue.

So backtracking is nogood-based search with parallel resolution

Example: SAT + conflict clauses

- Usae stronger nogoods = conflict clauses.
  - Nogoods rule out only branches that play a role in unit clause refutation.

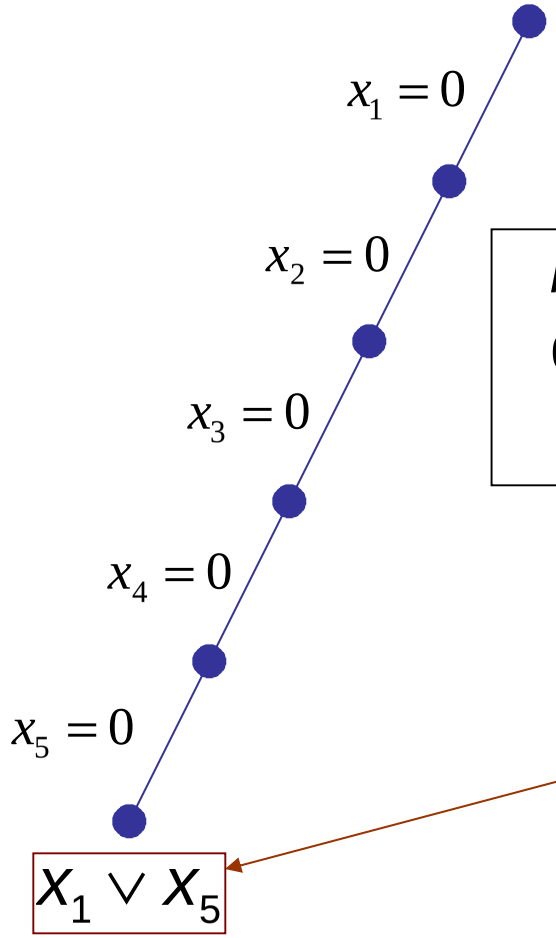
DPLL with conflict clauses



Branch to here. Unit clause rule proves infeasibility.

(x_1,x_5) = (0,0) is only premise of unit clause proof.

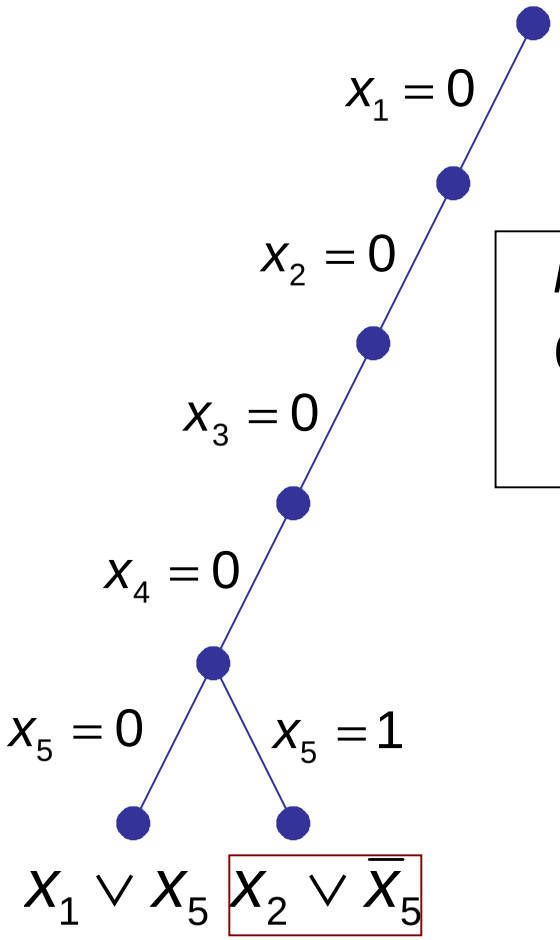
DPLL with conflict clauses



k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_1 \vee x_5$
1	$x_1 \vee x_5$		

Conflict clause appears as nogood induced by solution of R_k.

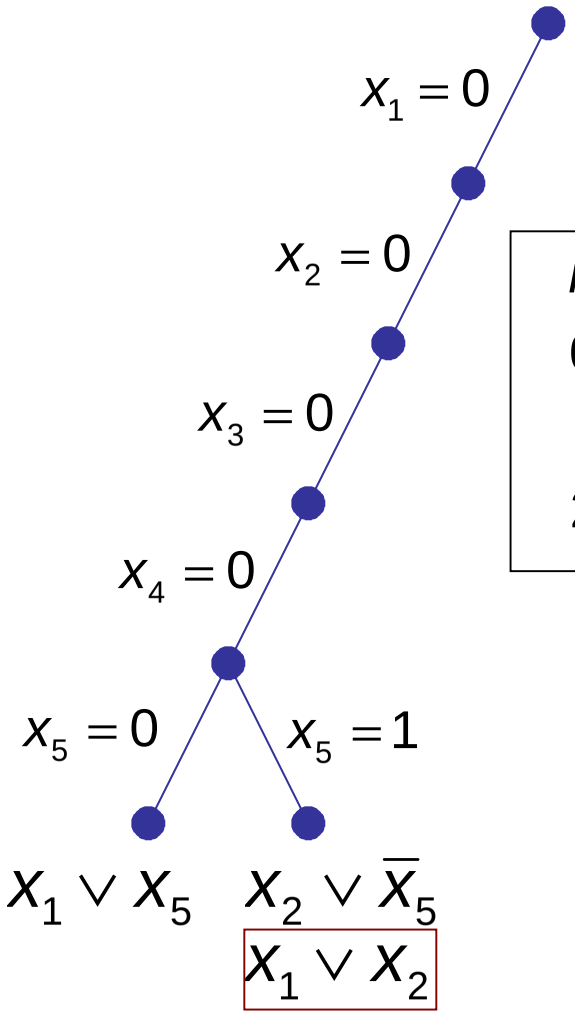
DPLL with conflict clauses



Consists of processed nogoods

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_1 \vee x_5$
1	$x_1 \vee x_5$	$(0,0,0,0,1,\cdot)$	$x_2 \vee \overline{x}_5$

DPLL with conflict clauses



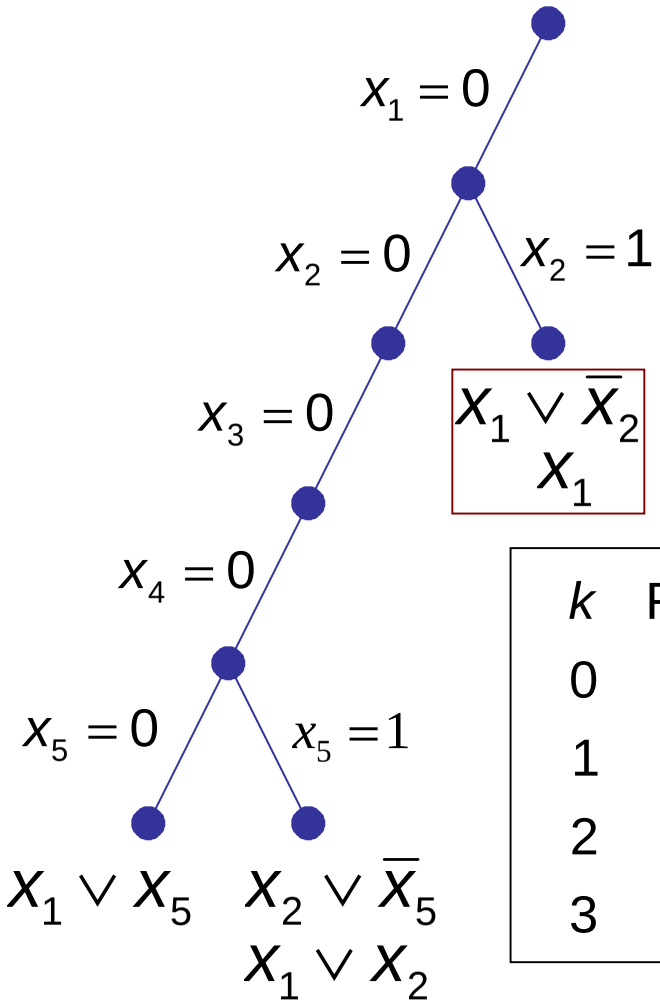
Consists of processed nogoods

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_1 \vee x_5$
1	$x_1 \vee x_5$	$(0,0,0,0,1,\cdot)$	$x_2 \vee \bar{x}_5$
2	$x_1 \vee x_2$		

$x_1 \vee x_5$
$x_2 \vee \bar{x}_5$ parallel-resolve to yield $x_1 \vee x_2$

$x_1 \vee x_2$ parallel-absorbs $x_1 \vee x_5$, $x_2 \vee \bar{x}_5$

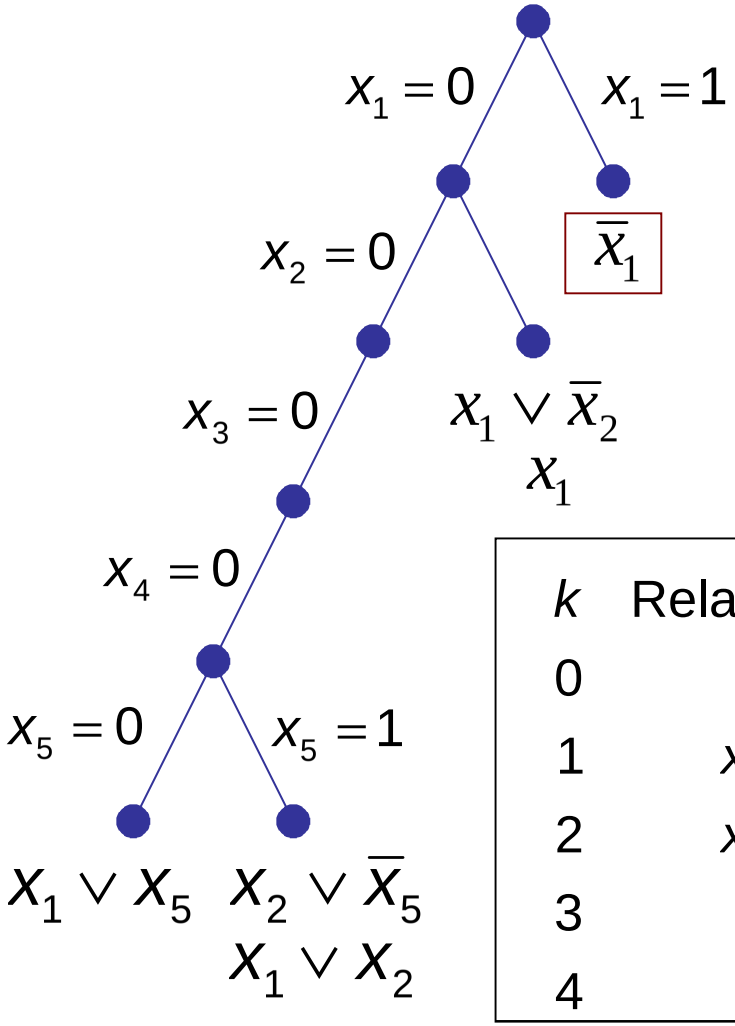
DPLL with conflict clauses



k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_1 \vee x_5$
1	$x_1 \vee x_5$	$(0,0,0,0,1,\cdot)$	$x_2 \vee \overline{x}_5$
2	$x_1 \vee x_2$	$(0,1,\cdot,\cdot,\cdot,\cdot)$	$x_1 \vee \overline{x}_2$
3	x_1		

$x_1 \vee x_2$
$x_1 \vee \overline{x}_2$ parallel-resolve to yield x_1

DPLL with conflict clauses



k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_1 \vee x_5$
1	$x_1 \vee x_5$	$(0,0,0,0,1,\cdot)$	$x_2 \vee \overline{x}_5$
2	$x_1 \vee x_2$	$(0,1,\cdot,\cdot,\cdot,\cdot)$	$x_1 \vee \overline{x}_2$
3	x_1	$(1,\cdot,\cdot,\cdot,\cdot,\cdot)$	$\overline{x}_1$
4	$\varnothing$		

Search terminates

Example: SAT + partial order dynamic backtracking

- Solve relaxation by selecting a solution that **conforms** to nogoods.
  - Conform = takes opposite sign than in nogoods.
  - More freedom than in branching.

Partial Order Dynamic Backtracking

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_5 \vee x_1$
1	$x_1 \vee x_5$		

Arbitrarily choose one variable to be last

Partial Order Dynamic Backtracking

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_5 \vee x_1$
1	$x_1 \vee x_5$		

Other variables are penultimate

Arbitrarily choose one variable to be last

Partial Order Dynamic Backtracking

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_5 \vee x_1$
1	$x_1 \vee x_5$	$(1,\cdot,\cdot,\cdot,0,\cdot)$	
2			

Since x_5 is penultimate in at least one nogood, it must conform to nogoods.

It must take value opposite its sign in the nogoods.

x_5 will have the same sign in all nogoods where it is penultimate.

This allows more freedom than chronological backtracking.

Partial Order Dynamic Backtracking

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_5 \vee x_1$
1	$x_1 \vee x_5$	$(1,\cdot,\cdot,\cdot,0,\cdot)$	$x_5 \vee \overline{x}_1$
2			

Choice of last variable is arbitrary but must be consistent with partial order implied by previous choices.

Since x_5 is penultimate in at least one nogood, it must conform to nogoods.

It must take value opposite its sign in the nogoods.

x_5 will have the same sign in all nogoods where it is penultimate.

This allows more freedom than chronological backtracking.

Partial Order Dynamic Backtracking

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_5 \vee x_1$
1	$x_1 \vee x_5$	$(1,\cdot,\cdot,\cdot,0,\cdot)$	$x_5 \vee \overline{x}_1$
2	x_5		

Choice of last variable is arbitrary but must be consistent with partial order implied by previous choices.

$x_5 \vee x_1$
$x_5 \vee \overline{x}_1$

Parallel-resolve to yield x_5

Since x_5 is penultimate in at least one nogood, it must conform to nogoods.

It must take value opposite its sign in the nogoods.

x_5 will have the same sign in all nogoods where it is penultimate.

This allows more freedom than chronological backtracking.

Partial Order Dynamic Backtracking

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_5 \vee x_1$
1	$x_1 \vee x_5$	$(1,\cdot,\cdot,\cdot,0,\cdot)$	$x_5 \vee \overline{x}_1$
2	x_5	$(\cdot,0,\cdot,\cdot,1,\cdot)$	$\overline{x}_5 \vee x_2$
3	x_5 $\overline{x}_5 \vee x_2$		

x_5 does not parallel-resolve with $\overline{x}_5 \vee x_2$

because x_5 is not last in both clauses

Partial Order Dynamic Backtracking

k	Relaxation R_k	Solution of R_k	Nogoods
0		$(0,0,0,0,0,\cdot)$	$x_5 \vee x_1$
1	$x_1 \vee x_5$	$(1,\cdot,\cdot,\cdot,0,\cdot)$	$x_5 \vee \overline{x}_1$
2	x_5	$(\cdot,0,\cdot,\cdot,1,\cdot)$	$\overline{x}_5 \vee x_2$
3	$\{x_5, \overline{x}_5 \vee x_2\}$	$(\cdot,1,\cdot,\cdot,1,\cdot)$	$\overline{x}_2$
4	$\varnothing$		

Search terminates

Must conform

Examples, with computational results

- Production planning.
  - Semicontinuous piecewise linear functions
- Product configuration.
  - Variable indices
- Machine scheduling.
  - Logic-based Benders
- Truss structure design.
  - Global optimization.

- Solved by SIMPL, a prototype integrated solver.

Production Planning

Maximize profit, which is a piecewise linear function of output.

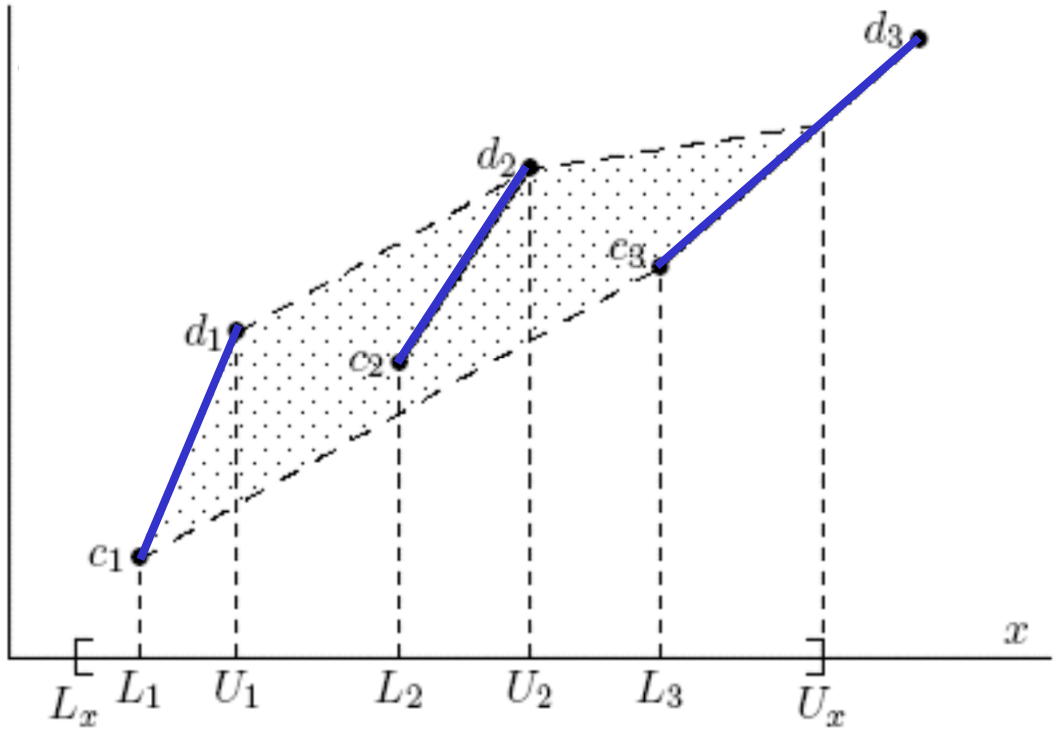
$$\max \sum_i f_i(x_i)$$

$$\sum_i x_i \le C$$

Each f_i is a piecewise linear semicontinuous function

Production Planning

Semicontinuous piecewise linear function $f(x)$



Production Planning

Integrated approach

- *Search:* **Branch** on variables
- *Relaxation:* Use a specialized **convex hull relaxation** for piecewise linear functions (no need for 0-1 model).
- *Inference:* **Bounds propagation**.

Original hand-coded method: Ottosson, Thorsteinsson and JNH 1999.

Production Planning

Integrated model

$$\max \sum_i u_i$$

$$\sum_i x_i \le C$$

$$\text{piecewise}(x_i, u_i, L_i, U_i, c_i, d_i), \quad \text{all } i$$

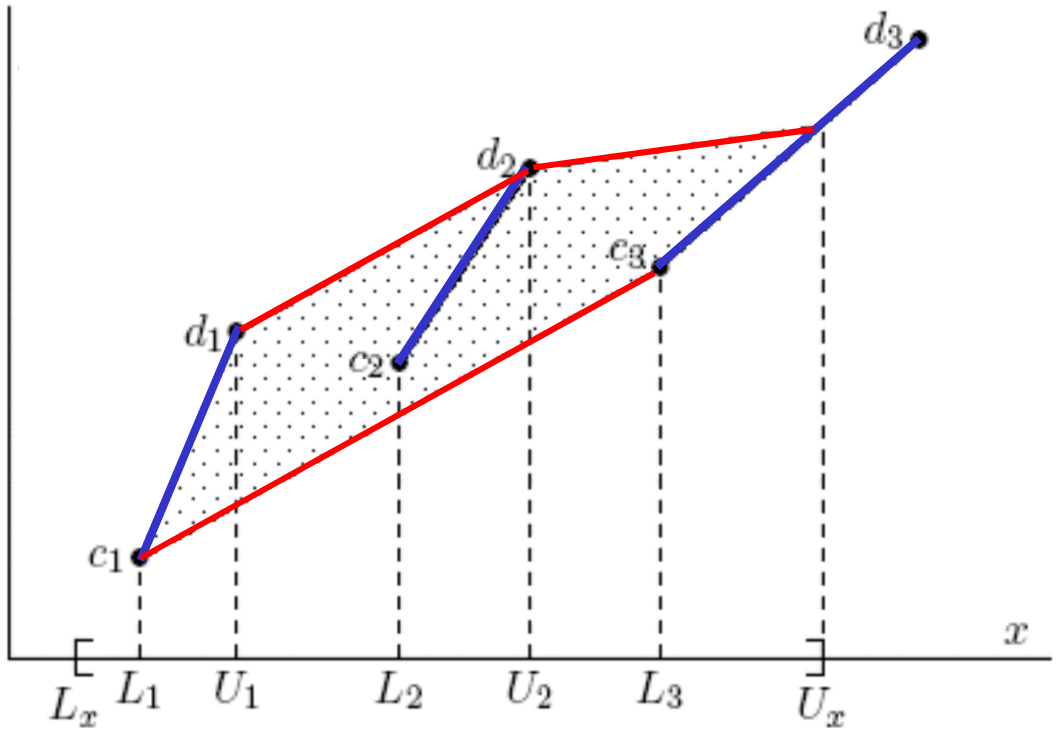
Metaconstraint

(global constraint in CP)

Production Planning

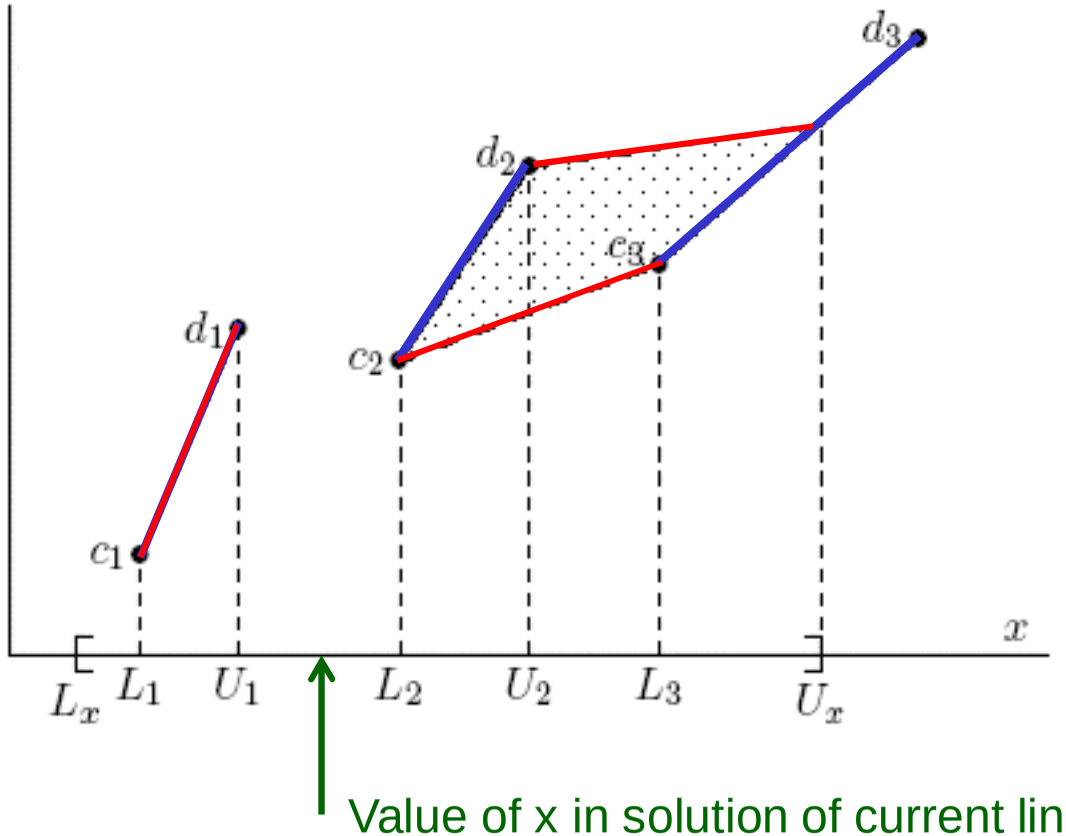
Semicontinuous piecewise linear function $f(x)$

Tight linear relaxation



Production Planning

Semicontinuous piecewise linear function *f*(*x*)



Tighter relaxation after branching

Value of x in solution of current linear relaxation

Production Planning

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of u[i]
03. CONSTRAINTS
04.   capacity means {
05.     sum i of x[i] <= C
06.     relaxation = { lp, cp } }
07.   piecewisectr means {
08.     piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
09.     relaxation = { lp, cp } }
10. SEARCH
11.   type = { bb:bestdive }
12.   branching = { piecewisectr:most }
```

Production Planning

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of u[i]
03. CONSTRAINTS
04.   capacity means {
05.     sum i of x[i] <= C
06.     relaxation = { lp, cp } }
07.   piecewisectr means {
08.     piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
09.     relaxation = { lp, cp } }
10. SEARCH
11.   type = { bb:bestdive }
12.   branching = { piecewisectr:most }
```

Recognized as a linear system.

Production Planning

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of u[i]
03. CONSTRAINTS
04.   capacity means {
05.     sum i of x[i] <= C
06.     relaxation = { lp, cp } }
07.   piecewisectr means {
08.     piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
09.     relaxation = { lp, cp } }
10. SEARCH
11.   type = { bb:bestdive }
12.   branching = { piecewisectr:most }
```

Is its own LP relaxation.

CP relaxation propagates bounds.

Production Planning

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of u[i]
03. CONSTRAINTS
04.   capacity means {
05.     sum i of x[i] <= C
06.     relaxation = { lp, cp } }
07.   piecewisectr means {
08.     piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
09.     relaxation = { lp, cp } }
10. SEARCH
11.   type = { bb:bestdive }
12.   branching = { piecewisectr:most }
```

Piecewise linear metaconstraint.

Production Planning

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of u[i]
03. CONSTRAINTS
04.   capacity means {
05.     sum i of x[i] <= C
06.     relaxation = { lp, cp } }
07.   piecewisectr means {
08.     piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
09.     relaxation = { lp, cp } }
10. SEARCH
11.   type = { bb:bestdive }
12.   branching = { piecewisectr:most }
```

LP relaxation is convex hull.

CP relaxation propagates bounds.

Production Planning

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of u[i]
03. CONSTRAINTS
04.   capacity means {
05.     sum i of x[i] <= C
06.     relaxation = { lp, cp } }
07.   piecewisectr means {
08.     piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
09.     relaxation = { lp, cp } }
10. SEARCH
11.   type = { bb:bestdive }
12.   branching = { piecewisectr:most }
```

Branch-and-bound search.

Dive to leaf node from node with best lower bound.

Production Planning

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of u[i]
03. CONSTRAINTS
04.   capacity means {
05.     sum i of x[i] <= C
06.     relaxation = { lp, cp } }
07.   piecewisectr means {
08.     piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
09.     relaxation = { lp, cp } }
10. SEARCH
11.   type = { bb:bestdive }
12.   branching = { piecewisectr:most }
```

Branch on piecewise constraint with greatest violation.

Production Planning

Computational Results (seconds)

Hand-coded integrated method was comparable to CPLEX 9

No. Products	MILP CPLEX 9	MILP CPLEX 11	SIMPL
50	9.5	2.4	0.43
60	55	1.8	1.1
70	99	2.6	0.82
80	61	4.6	1.25
90	422	6.2	1.7
100	4458	4.4	2.8

Production Planning

CPLEX has become orders of magnitude faster,
but still slower than SIMPL

No. Products	MILP CPLEX 9	MILP CPLEX 11	SIMPL
50	9.5	2.4	0.43
60	55	1.8	1.1
70	99	2.6	0.82
80	61	4.6	1.25
90	422	6.2	1.7
100	4458	4.4	2.8

SIMPL's advantage grows with the problem size

Seconds

No. Products	MILP CPLEX 9	MILP CPLEX 11	SIMPL
300	82	376	19
300	701	372	19
600	3515	4509	39
600	214	9416	131

SIMPL's advantage grows with the problem size

Seconds

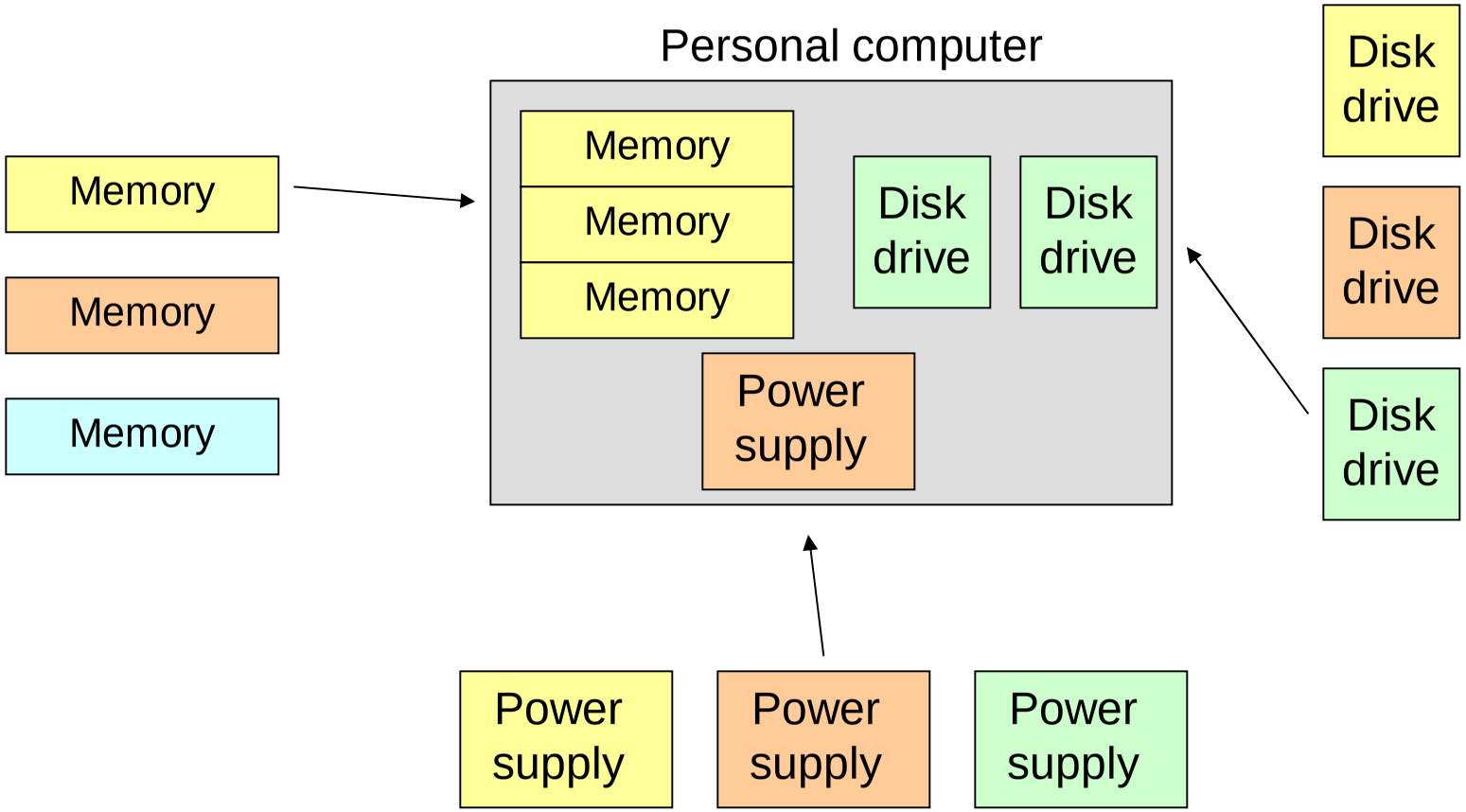
No. Products	MILP CPLEX 9	MILP CPLEX 11	SIMPL
300	82	376	19
300	701	372	19
600	3515	4509	39
600	214	9416	131

Nodes

No. Products	MILP CPLEX 9	MILP CPLEX 11	SIMPL
300	10,164	101,756	73
300	43,242	128,333	58
600	363,740	646,907	74
600	7,732	1,297,071	214

Product configuration

Choose what type of each component, and how many



Product configuration

Integrated approach

- *Search:* B**ranch** on variables.
- *Relaxation:* Generate a **specialized linear relaxation** for CP-based global constraints in the model.
- *Inference:* Apply specialized **filtering** algorithms to global constraints, and generate **knapsack cuts**.

Product configuration

Unit cost of producing attribute *j*

Amount of attribute *j* produced (< 0 if consumed): *memory, heat, power, weight, etc.*

$$\min \sum_j c_j v_j$$

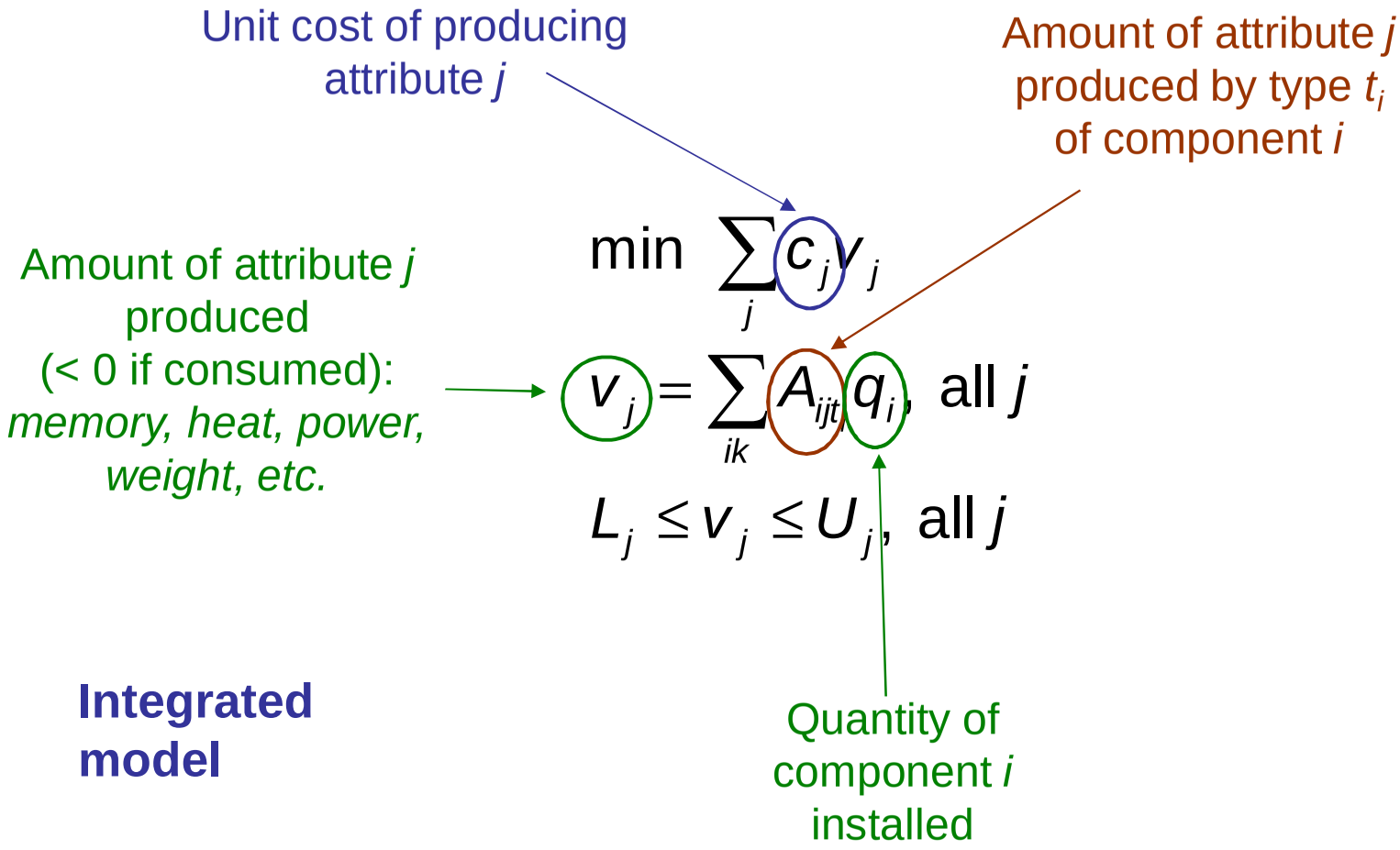
$$v_j = \sum_{ik} A_{ijt_i} q_i, \text{ all } j$$

$$L_j \le v_j \le U_j, \text{ all } j$$

Integrated model

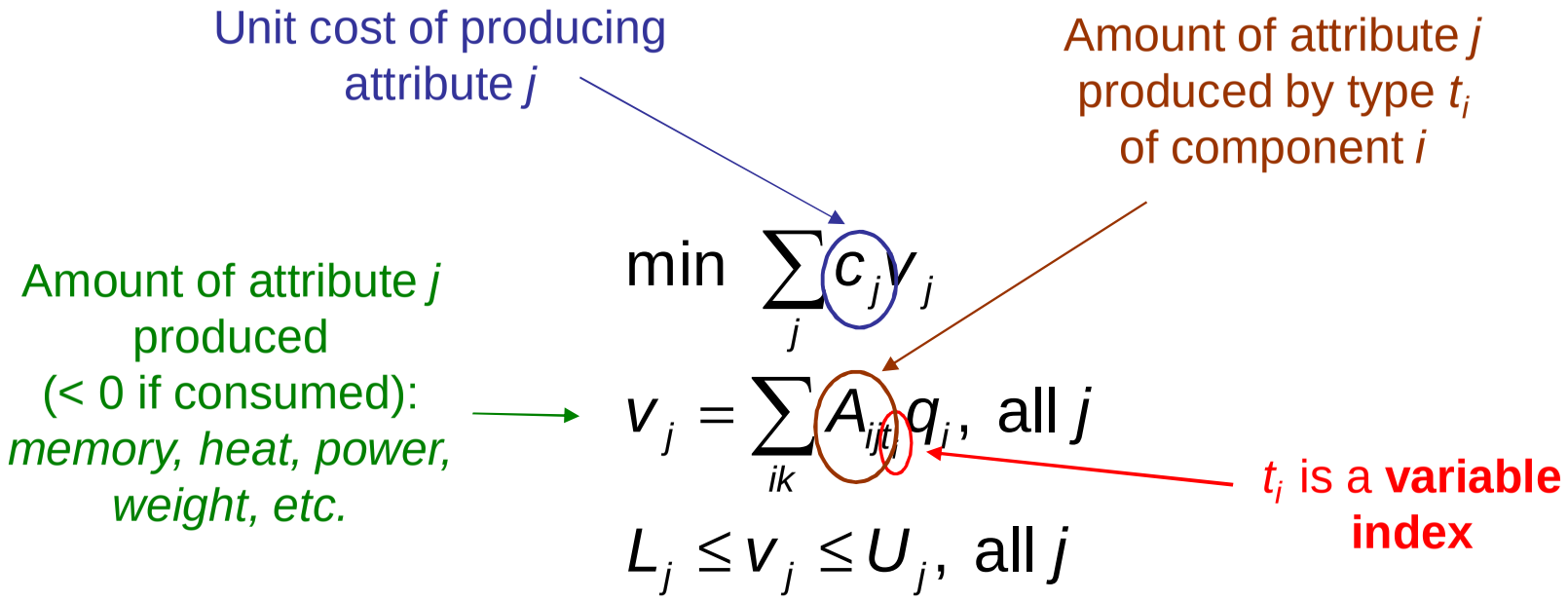
Quantity of component *i* installed

Product configuration



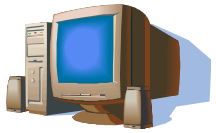
Integrated model

Product configuration



Integrated model

Product configuration



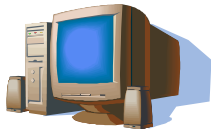
Linear inequality
metaconstraint

$$\min \sum_j c_j v_j$$

$$v_j = \sum_{ik} q_i A_{ijt_i}, \text{ all } j$$

$$L_j \le v_j \le U_j, \text{ all } j$$

Product configuration



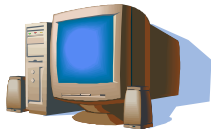
$$\min \sum_j c_j v_j$$

Indexed linear metaconstraint

$$v_j = \sum_{ik} q_i A_{ijt_i}, \text{ all } j$$

$$L_j \le v_j \le U_j, \text{ all } j$$

Product configuration



Propagation

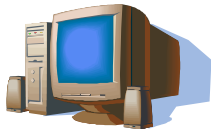
$$\min \sum_j c_j v_j$$

$$v_j = \sum_{ik} q_i A_{ijt_i}, \text{ all } j$$

$$\boxed{L_j \le v_j \le U_j, \text{ all } j}$$

This is propagated in the usual way

Product configuration



Propagation

$$v_j = \sum_i z_i, \text{ all } j$$

$$\text{element}\left(t_i, (q_i, A_{ij1}, \ldots, q_i A_{ijn}), z_i\right), \text{ all } i, j$$

$$\min \sum_j c_j v_j$$

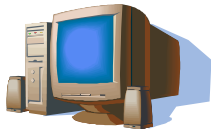
$$v_j = \sum_{ik} q_i A_{ijt_i}, \text{ all } j$$

This is rewritten as

$$L_j \le v_j \le U_j, \text{ all } j$$

This is propagated in the usual way

Product configuration



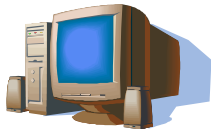
Propagation

$$v_j = \sum_i z_i, \text{ all } j$$

$$\text{element}\left(t_i, (q_i, A_{ij1}, \ldots, q_i A_{ijn}), z_i\right), \text{ all } i, j$$

This is propagated by
(a) using specialized **filters** for *element* constraints of this form…

Product configuration



Propagation

$$v_j = \sum_i z_i, \text{ all } j$$

$$\text{element}\left(t_i, (q_i, A_{ij1}, \ldots, q_i A_{ijn}), z_i\right), \text{ all } i, j$$

This is propagated by
(a) using specialized **filters** for *element* constraints of this form,
(b) adding **knapsack cuts** for the valid inequalities:

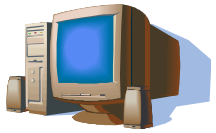
$$\sum_i \max_{k \in D_{t_i}} \{A_{ijk}\} q_i \geq \underline{v}_j, \text{ all } j$$

$$\sum_i \min_{k \in D_{t_i}} \{A_{ijk}\} q_i \leq \overline{v}_j, \text{ all } j$$

and (c) propagating the knapsack cuts.

$[\underline{v}_j, \overline{v}_j]$ is current domain of v_j

Product configuration



Relaxation

$$\min \sum_j c_j v_j$$

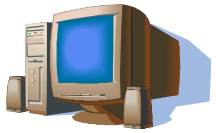
$$v_j = \sum_{ik} q_i A_{ijt_i}, \text{ all } j$$

$$\boxed{L_j \le v_j \le U_j, \text{ all } j}$$

This is relaxed as

$$\underline{v}_j \le v_j \le \bar{v}_j$$

Product configuration



Relaxation

$$v_j = \sum_i z_i, \text{ all } j$$

$$\text{element}\left(t_i, (q_i, A_{ij1}, \ldots, q_i A_{ijn}), z_i\right), \text{ all } i, j$$

$$\min \sum_j c_j v_j$$

$$v_j = \sum_{ik} q_i A_{ijt_i}, \text{ all } j$$

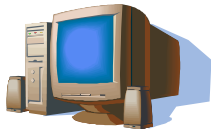
$$L_j \le v_j \le U_j, \text{ all } j$$

This is relaxed by relaxing *this* and adding the knapsack cuts.

This is relaxed as

$$\underline{v}_j \le v_j \le \overline{v}_j$$

Product configuration



Relaxation

$$v_j = \sum_i z_i, \text{ all } j$$

$$\text{element}\left(t_i, (q_i, A_{ij1}, \ldots, q_i A_{ijn}), z_i\right), \text{ all } i, j$$

This is relaxed by writing each *element* constraint as a **disjunction** of linear systems and writing a **convex hull** relaxation of the disjunction:

$$z_i = \sum_{k \in D_{t_i}} A_{ijk} q_{ik}, \quad q_i = \sum_{k \in D_{t_i}} q_{ik}$$

Product configuration

SIMPL model

```
01. OBJECTIVE
02.   minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04.   usage means {
05.     v[j] = sum i of q[i]*a[i][j][t[i]] forall j
06.     relaxation = { lp, cp }
07.     inference = { knapsack } }
08.   quantities means {
09.     q[1] >= 1 => q[2] = 0
10.     relaxation = { lp, cp } }
15. SEARCH
16.   type = { bb:bestdive }
17.   branching = { quantities, t:most, q:least:triple, types:most }
18.   inference = { q:redcost }
```

Recognized as indexed linear system

Product configuration

SIMPL model

```
01. OBJECTIVE
02.   minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04.   usage means {
05.     v[j] = sum i of q[i]*a[i][j][t[i]] forall j
06.     relaxation = { lp, cp }
07.     inference = { knapsack } }
08.   quantities means {
09.     q[1] >= 1 => q[2] = 0
10.     relaxation = { lp, cp } }
15. SEARCH
16.   type = { bb:bestdive }
17.   branching = { quantities, t:most, q:least:triple, types:most }
18.   inference = { q:redcost }
```

LP relaxation is convex hull of disjunction.

CP relaxation propagates bounds.

Product configuration

SIMPL model

```
01. OBJECTIVE
02.   minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04.   usage means {
05.     v[j] = sum i of q[i]*a[i][j][t[i]] forall j
06.     relaxation = { lp, cp }
07.     inference = { knapsack } }
08.   quantities means {
09.     q[1] >= 1 => q[2] = 0
10.     relaxation = { lp, cp } }
15. SEARCH
16.   type = { bb:bestdive }
17.   branching = { quantities, t:most, q:least:triple, types:most }
18.   inference = { q:redcost }
```

Generate knapsack cuts from associated valid inequalities.

Product configuration

SIMPL model

```
01. OBJECTIVE
02.   minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04.   usage means {
05.     v[j] = sum i of q[i]*a[i][j][t[i]] forall j
06.     relaxation = { lp, cp }
07.     inference = { knapsack } }
08.   quantities means {
09.     q[1] >= 1 => q[2] = 0
10.     relaxation = { lp, cp } }
15. SEARCH
16.   type = { bb:bestdive }
17.   branching = { quantities, t:most, q:least:triple, types:most }
18.   inference = { q:redcost }
```

Logical constraint on quantities

Product configuration

SIMPL model

```
01. OBJECTIVE
02.   minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04.   usage means {
05.     v[j] = sum i of q[i]*a[i][j][t[i]] forall j
06.     relaxation = { lp, cp }
07.     inference = { knapsack } }
08.   quantities means {
09.     q[1] >= 1 => q[2] = 0
10.     relaxation = { lp, cp } }
15. SEARCH
16.   type = { bb:bestdive }
17.   branching = { quantities, t:most, q:least:triple, types:most }
18.   inference = { q:redcost }
```

First branch on violated logical constraint on q_i variables

Product configuration

```
01. OBJECTIVE
02.   minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04.   usage means {
05.     v[j] = sum i of q[i]*a[i][j][t[i]] forall j
06.     relaxation = { lp, cp }
07.     inference = { knapsack } }
08.   quantities means {
09.     q[1] >= 1 => q[2] = 0
10.     relaxation = { lp, cp } }
15. SEARCH
16.   type = { bb:bestdive }
17.   branching = { quantities, t:most, q:least:triple, types:most }
18.   inference = { q:redcost }
```

SIMPL model

Then branch on most violated t_i in-domain constraint.

Violated when domain of t_i is not a singleton, or two or more associated q_{ik}s are positive.

Product configuration

SIMPL model

```
01. OBJECTIVE
02.   minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04.   usage means {
05.     v[j] = sum i of q[i]*a[i][j][t[i]] forall j
06.     relaxation = { lp, cp }
07.     inference = { knapsack } }
08.   quantities means {
09.     q[1] >= 1 => q[2] = 0
10.     relaxation = { lp, cp } }
15. SEARCH
16.   type = { bb:bestdive }
17.   branching = { quantities, t:most, q:least:triple, types:most }
18.   inference = { q:redcost }
```

Then branch on least violated q_i in-domain constraint.

Create three branches: q_i = nearest integer q_i', $q_i < q_i'$, $q_i > q_i'$

Product configuration

SIMPL model

```
01. OBJECTIVE
02.   minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04.   usage means {
05.     v[j] = sum i of q[i]*a[i][j][t[i]] forall j
06.     relaxation = { lp, cp }
07.     inference = { knapsack } }
08.   quantities means {
09.     q[1] >= 1 => q[2] = 0
10.     relaxation = { lp, cp } }
15. SEARCH
16.   type = { bb:bestdive }
17.   branching = { quantities, t:most, q:least:triple, types:most }
18.   inference = { q:redcost }
```

Then branch on most violated logical constraint on t_i variables (omitted)

Product configuration

SIMPL model

```
01. OBJECTIVE
02.   minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04.   usage means {
05.     v[j] = sum i of q[i]*a[i][j][t[i]] forall j
06.     relaxation = { lp, cp }
07.     inference = { knapsack } }
08.   quantities means {
09.     q[1] >= 1 => q[2] = 0
10.     relaxation = { lp, cp } }
15. SEARCH
16.   type = { bb:bestdive }
17.   branching = { quantities, t:most, q:least:triple, types:most }
18.   inference = { q:redcost }
```

Reduced-cost variable fixing for q_i's

Product configuration

Computational results

SIMPL matches hand-coded integrated method, which was orders of magnitude faster than CPLEX.

Again, CPLEX has become much faster, now somewhat faster than SIMPL.

MILP (CPLEX 11)		SIMPL	
Nodes	Sec.	Nodes	Sec.
1	0.07	56	0.49
1	0.10	32	0.25
31	0.68	186	1.67
1	0.02	28	0.24
1	0.12	32	0.33
1	0.07	9	0.09
10	0.18	35	0.30
1	0.10	32	0.25
1	0.05	28	0.22
1	0.12	14	0.13

Machine scheduling

- Assign jobs to machines, and schedule the machines assigned to each machine within time windows.
- The objective is to minimize **processing cost**.

Machine scheduling

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

Example

Assign 5 jobs to 2 machines.

Schedule jobs assigned to each machine without overlap.

Machine scheduling

Integrated model

$$\min \sum_j c_{x_j j}$$

Start time of job *j*

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

Time windows

Jobs cannot overlap

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

Machine assigned to job *j*

Machine scheduling

Integrated approach

• *Search:* Enumerate **subproblems** (defined by assigning jobs to machines)

• *Relaxation:* Enumerate **master problems** (which assign jobs to machines)

• *Inference:* Generate **nogoods** (logic-based Benders cuts), which are added to master problem.

Machine scheduling

Integrated approach

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

Original hand-coded method: Jain and Grossman 2001.

Machine scheduling

Integrated approach

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

The subproblem decouples into a separate scheduling problem on each machine.

In this problem, the subproblem is a feasibility problem.

Original hand-coded method: Jain and Grossman 2001.

Machine scheduling

Integrated approach

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

The subproblem decouples into a separate scheduling problem on each machine.

In this problem, the subproblem is a feasibility problem.

- Solve **inference dual** of subproblem to generate **nogoods** (logic-based Benders cuts), which are added to master problem.

Original hand-coded method: Jain and Grossman 2001.

Machine scheduling

Integrated model

$$\min \sum_j c_{x_j j}$$

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

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

Indexed linear metaconstraint

Machine scheduling

Integrated model

$$\min \sum_j c_{x_j j}$$

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

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

Indexed linear metaconstraint

Disjunctive scheduling metaconstraint

Machine scheduling

Integrated 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 machine i is

$$\min \ M$$

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

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

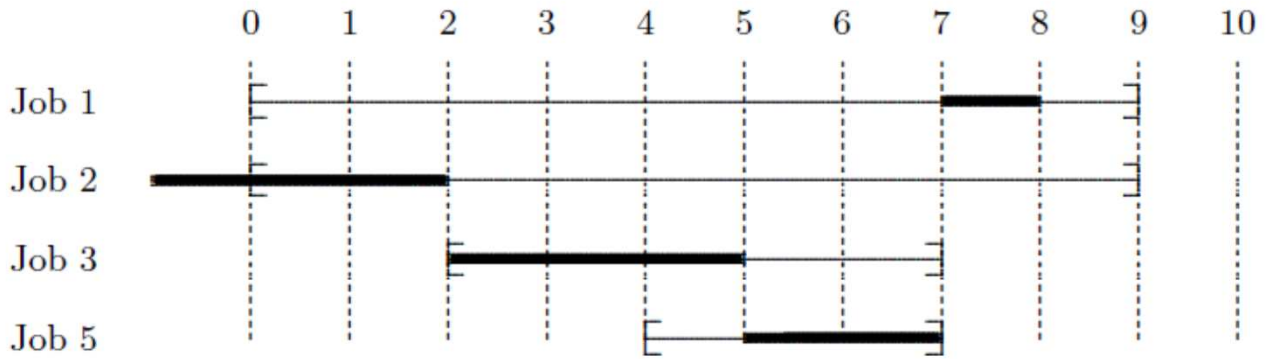
$$\text{disjunctive}\left((s_j \,|\, \bar{x}_j = i),(p_{ij} \,|\, \bar{x}_j = i)\right)$$

Machine scheduling

Logic-based Benders approach

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

We can prove that there is no feasible schedule.



Edge finding derives infeasibility by reasoning only with jobs 2,3,5.
So these jobs alone create infeasibility.

So we have a Benders cut $\neg(x_2 = x_3 = x_5 = A)$

Machine scheduling

Logic-based Benders approach

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

So we write the Benders cut $\neg(x_2 = x_3 = x_5 = A)$

Using 0-1 variables: $x_{A2} + x_{A3} + \boxed{x_{A5}} \leq 2$

= 1 if job 5 is assigned to machine A

Machine scheduling

The master problem is a **relaxation**, formulated as an MILP:

$$\min \sum_{ij} c_{ij} x_{ij}$$

$$\sum_{j=1}^{5} p_{Aj} x_{Aj} \leq 9, \text{ etc.}$$

$$\sum_{j=1}^{5} p_{Bj} x_{Bj} \leq 9, \text{ etc.}$$

Constraints derived from time windows

$$x_{A2} + x_{A3} + x_{A5} \leq 2$$

Benders cut from machine A

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

Machine scheduling

The master problem is a **relaxation**, formulated as an MILP:

$$\min \sum_{ij} c_{ij} x_{ij}$$

$$\sum_{j=1}^{5} p_{Aj} x_{Aj} \leq 9, \text{ etc.}$$

$$\sum_{j=1}^{5} p_{Bj} x_{Bj} \leq 9, \text{ etc.}$$

Constraints derived from time windows

$$x_{A2} + x_{A3} + x_{A5} \leq 2$$

Benders cut from machine A

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

Benders cuts have been developed for min **makespan** and min **tardiness** (subproblem is an optimization problem)

Also for **cumulative** scheduling.

Machine scheduling

SIMPL model

```
01. OBJECTIVE
02.   min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04.   assign means {
05.     sum i of x[i][j] = 1 forall j;
06.     relaxation = { ip:master } }
07.   xy means {
08.     x[i][j] = 1 <=> y[j] = i forall i, j;
09.     relaxation = { cp } }
10.   tbounds means {
11.     r[j] <= t[j] forall j;
12.     t[j] <= d[j] - p[y[j]][j] forall j;
13.     relaxation = { ip:master, cp } }
14.   machinecap means {
15.     cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.     relaxation = { cp:subproblem, ip:master }
17.     inference = { feasibility } }
18. SEARCH
19.   type = { benders }
```

Machine assignment constraint

Machine scheduling

```
01. OBJECTIVE
02.   min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04.   assign means {
05.     sum i of x[i][j] = 1 forall j;
06.     relaxation = { ip:master } }
07.   xy means {
08.     x[i][j] = 1 <=> y[j] = i forall i, j;
09.     relaxation = { cp } }
10.   tbounds means {
11.     r[j] <= t[j] forall j;
12.     t[j] <= d[j] - p[y[j]][j] forall j;
13.     relaxation = { ip:master, cp } }
14.   machinecap means {
15.     cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.     relaxation = { cp:subproblem, ip:master }
17.     inference = { feasibility } }
18. SEARCH
19.   type = { benders }
```

SIMPL model

MILP relaxation of the constraint (which is the constraint itself) goes into master problem

Machine scheduling

SIMPL model

```
01. OBJECTIVE
02.   min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04.   assign means {
05.     sum i of x[i][j] = 1 forall j;
06.     relaxation = { ip:master } }
07.   xy means {
08.     x[i][j] = 1 <=> y[j] = i forall i, j;
09.     relaxation = { cp } }
10.   tbounds means {
11.     r[j] <= t[j] forall j;
12.     t[j] <= d[j] - p[y[j]][j] forall j;
13.     relaxation = { ip:master, cp } }
14.   machinecap means {
15.     cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.     relaxation = { cp:subproblem, ip:master }
17.     inference = { feasibility } }
18. SEARCH
19.   type = { benders }
```

Definition of x_{ij} variables for MILP master problem

Machine scheduling

SIMPL model

```
01. OBJECTIVE
02.   min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04.   assign means {
05.     sum i of x[i][j] = 1 forall j;
06.     relaxation = { ip:master } }
07.   xy means {
08.     x[i][j] = 1 <=> y[j] = i forall i, j;
09.     relaxation = { cp } }
10.   tbounds means {
11.     r[j] <= t[j] forall j;
12.     t[j] <= d[j] - p[y[j]][j] forall j;
13.     relaxation = { ip:master, cp } }
14.   machinecap means {
15.     cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.     relaxation = { cp:subproblem, ip:master }
17.     inference = { feasibility } }
18. SEARCH
19.   type = { benders }
```

CP-based propagation

Machine scheduling

```
01. OBJECTIVE
02.   min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04.   assign means {
05.     sum i of x[i][j] = 1 forall j;
06.     relaxation = { ip:master } }
07.   xy means {
08.     x[i][j] = 1 <=> y[j] = i forall i, j;
09.     relaxation = { cp } }
10.   tbounds means {
11.     r[j] <= t[j] forall j;
12.     t[j] <= d[j] - p[y[j]][j] forall j;
13.     relaxation = { ip:master, cp } }
14.   machinecap means {
15.     cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.     relaxation = { cp:subproblem, ip:master }
17.     inference = { feasibility } }
18. SEARCH
19.   type = { benders }
```

SIMPL model

Time window constraints recognized as indexed linear system

Machine scheduling

```
01. OBJECTIVE
02.   min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04.   assign means {
05.     sum i of x[i][j] = 1 forall j;
06.     relaxation = { ip:master } }
07.   xy means {
08.     x[i][j] = 1 <=> y[j] = i forall i, j;
09.     relaxation = { cp } }
10.   tbounds means {
11.     r[j] <= t[j] forall j;
12.     t[j] <= d[j] - p[y[j]][j] forall j;
13.     relaxation = { ip:master, cp } }
14.   machinecap means {
15.     cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.     relaxation = { cp:subproblem, ip:master }
17.     inference = { feasibility } }
18. SEARCH
19.   type = { benders }
```

SIMPL model

MILP formulation goes into master problem

CP-based propagation

Machine scheduling

```
01. OBJECTIVE
02.   min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04.   assign means {
05.     sum i of x[i][j] = 1 forall j;
06.     relaxation = { ip:master } }
07.   xy means {
08.     x[i][j] = 1 <=> y[j] = i forall i, j;
09.     relaxation = { cp } }
10.   tbounds means {
11.     r[j] <= t[j] forall j;
12.     t[j] <= d[j] - p[y[j]][j] forall j;
13.     relaxation = { ip:master, cp } }
14.   machinecap means {
15.     cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.     relaxation = { cp:subproblem, ip:master }
17.     inference = { feasibility } }
18. SEARCH
19.   type = { benders }
```

SIMPL model

Disjunctive scheduling constraint written as special case of cumulative scheduling constraint (resource consumption = 1, capacity = 1)

Machine scheduling

```
01. OBJECTIVE
02.   min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04.   assign means {
05.     sum i of x[i][j] = 1 forall j;
06.     relaxation = { ip:master } }
07.   xy means {
08.     x[i][j] = 1 <=> y[j] = i forall i, j;
09.     relaxation = { cp } }
10.   tbounds means {
11.     r[j] <= t[j] forall j;
12.     t[j] <= d[j] - p[y[j]][j] forall j;
13.     relaxation = { ip:master, cp } }
14.   machinecap means {
15.     cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.     relaxation = { cp:subproblem, ip:master }
17.     inference = { feasibility } }
18. SEARCH
19.   type = { benders }
```

SIMPL model

The CP problem goes into the Benders subproblem.

A relaxation of the constraint goes into the master

Machine scheduling

SIMPL model

```
01. OBJECTIVE
02.   min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04.   assign means {
05.     sum i of x[i][j] = 1 forall j;
06.     relaxation = { ip:master } }
07.   xy means {
08.     x[i][j] = 1 <=> y[j] = i forall i, j;
09.     relaxation = { cp } }
10.   tbounds means {
11.     r[j] <= t[j] forall j;
12.     t[j] <= d[j] - p[y[j]][j] forall j;
13.     relaxation = { ip:master, cp } }
14.   machinecap means {
15.     cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.     relaxation = { cp:subproblem, ip:master }
17.     inference = { feasibility } }
18. SEARCH
19.   type = { benders }
```

Generate logic-based Benders cuts when the subproblem is infeasible

Machine scheduling

```
01. OBJECTIVE
02.   min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04.   assign means {
05.     sum i of x[i][j] = 1 forall j;
06.     relaxation = { ip:master } }
07.   xy means {
08.     x[i][j] = 1 <=> y[j] = i forall i, j;
09.     relaxation = { cp } }
10.   tbounds means {
11.     r[j] <= t[j] forall j;
12.     t[j] <= d[j] - p[y[j]][j] forall j;
13.     relaxation = { ip:master, cp } }
14.   machinecap means {
15.     cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.     relaxation = { cp:subproblem, ip:master }
17.     inference = { feasibility } }
18. SEARCH
19.   type = { benders }
```

SIMPL model

Benders-based search, where problem restrictions are Benders subproblems and problem relaxations are master problems.

Machine scheduling

Computational results – Long processing times

Jobs	Machines	MILP (CPLEX 11) Nodes	MILP (CPLEX 11) Sec.	SIMPL Benders Iter.	SIMPL Benders Cuts	SIMPL 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

SIMPL results are similar to original hand-coded results.

Machine scheduling

Computational results – Short processing times

Jobs	Machines	MILP (CPLEX 11) Nodes	MILP (CPLEX 11) Sec.	SIMPL Benders Iter.	SIMPL Benders Cuts	SIMPL 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

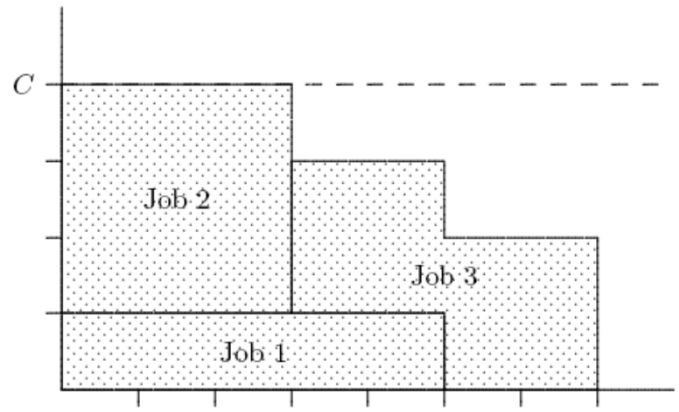
Machine scheduling

Benders cut for minimum **makespan**, with a **cumulative scheduling** subproblem

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

Jobs currently assigned to machine *i*

Minimum makespan on machine *i* for jobs currently assigned



Logic-based Benders Decomposition

- 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.

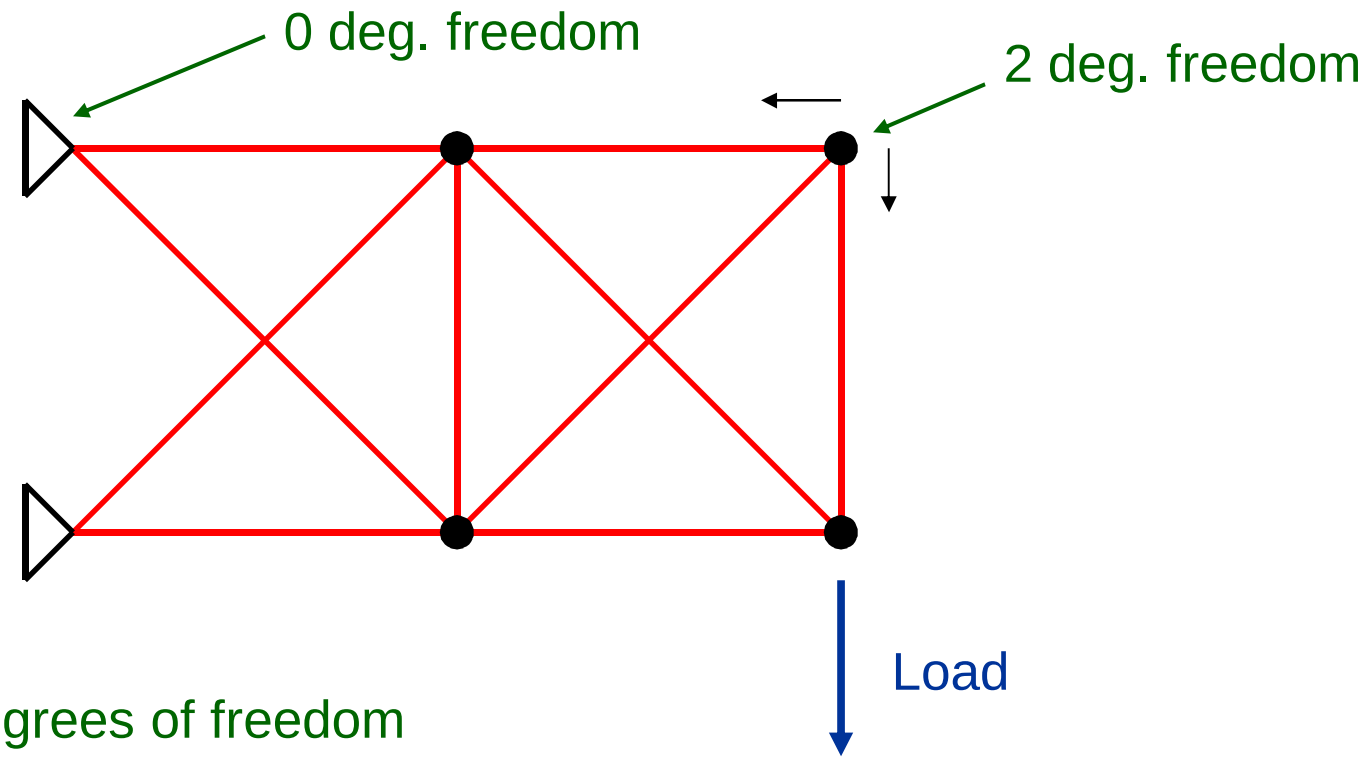
Logic-based Benders Decomposition

- 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.

Truss Structure Design

Select size of each bar (possibly zero) to support the load while minimizing weight.

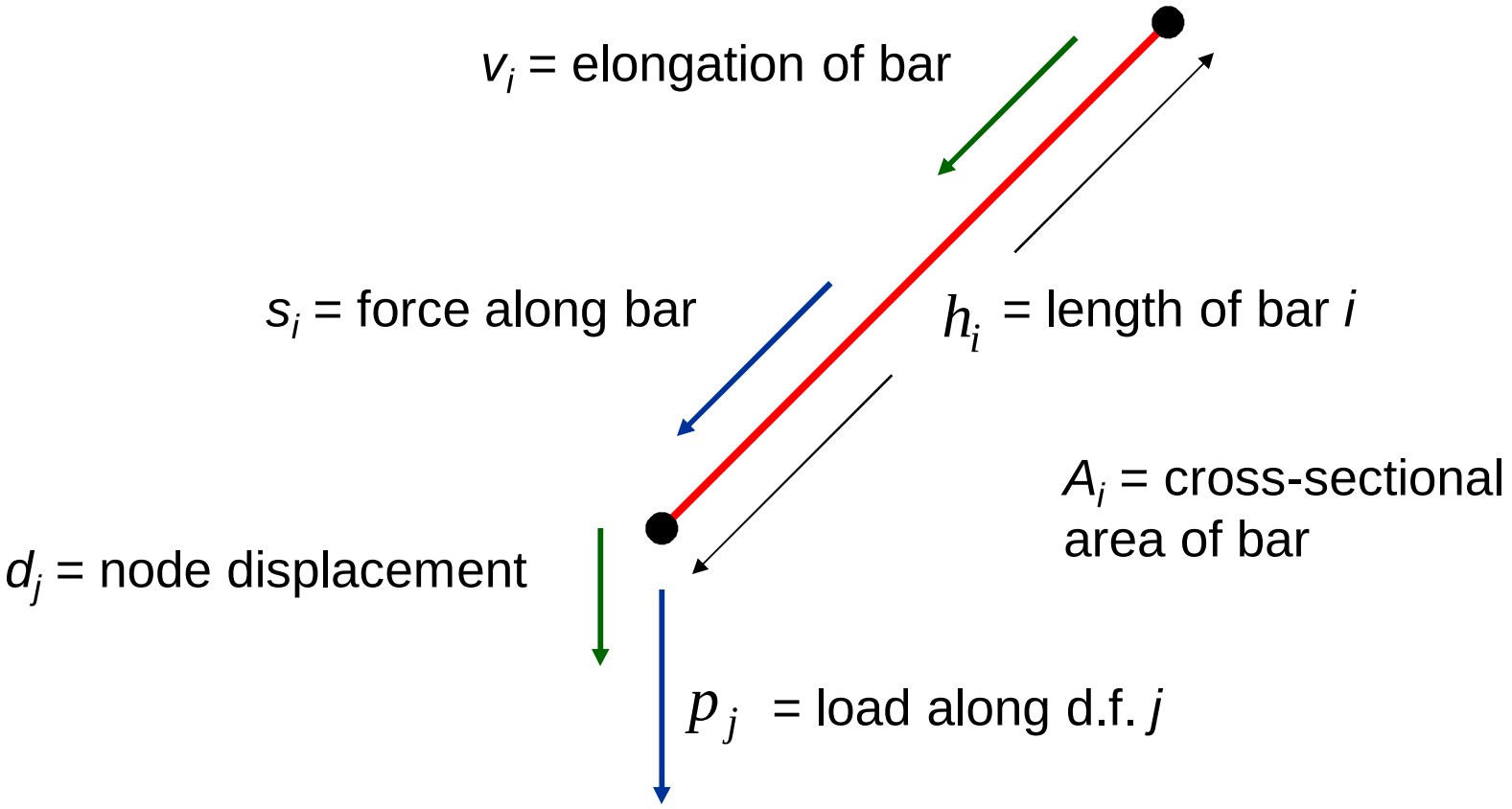
10-bar cantilever truss



Total 8 degrees of freedom

Truss Structure Design

Notation



Truss Structure Design

$$\min \quad \sum_i h_i A_i$$ } Minimize total weight

$$s.t. \quad \sum_i \cos\theta_{ij} s_i = p_j, \text{ all } j$$ } Equilibrium

$$\sum_j \cos\theta_{ij} d_j = v_i, \text{ all } i$$ } Compatibility

nonlinear →

$$\frac{E_i}{h_i} A_i v_i = s_i, \text{ all } i$$ } Hooke's law

$$v_i^L \le v_i \le v_i^U, \text{ all } i$$ } Elongation bounds

$$d_j^L \le d_j \le d_j^U, \text{ all } j$$ } Displacement bounds

$$\bigvee_k (A_i = A_{ik})$$ } Logical disjunction

Area must be one of several discrete values A_{ik}

Constraints can be imposed for multiple loading conditions

Truss Structure Design

Introducing new variables linearizes the problem but makes it much larger.

MILP model

$$\min \quad \sum_i h_i \sum_k A_{ik} y_{ik}$$

$$s.t. \quad \sum_i \cos\theta_{ij} s_i = p_j, \text{ all } j$$

$$\sum_j \cos\theta_{ij} d_j = \sum_k v_{ik}, \text{ all } i$$

$$\frac{E_i}{h_i} \sum_k A_{ik} v_{ik} = s_i, \text{ all } i$$

$$v_i^L \le v_i \le v_i^U, \text{ all } i$$

$$d_j^L \le d_j \le d_j^U, \text{ all } j$$

$$\sum_k y_{ik} = 1, \text{ all } i$$

0-1 variables indicating size of bar *i*

Elongation variable disaggregated by bar size

Hooke's law becomes linear

Truss Structure Design

Integrated approach

• Search: Branch by **splitting** the range of areas A_i (no need for 0-1 variables).

• Relaxation: Generate a **quasi-relaxation**, which is linear and much smaller than MILP model.

• Inference: Use **logic cuts**.

Original hand-coded method: Bollapragada, Ghattas, and JNH 2001.

Truss Structure Design

Theorem (JNH 2005)

Suppose we minimize cx subject to $g(x,y) \leq 0$.

If $g(x,y)$ is semihomogeneous in $x \in R^n$ and concave in scalar y, then the following is a **quasi-relaxation** of $g(x,y) \leq 0$:

$$g(x^1, y_L) + g(x^2, y_U) \leq 0$$
$$\alpha x^L \leq x^1 \leq \alpha x^U$$
$$(1-\alpha)x^L \leq x^2 \leq (1-\alpha)x^U$$
$$x = x^1 + x^2$$

Truss Structure Design

Theorem (JNH 2005)

Suppose we minimize cx subject to $g(x,y) \le 0$.

If $g(x,y)$ is semihomogeneous in $x \in R^n$ and concave in scalar y, then the following is a **quasi-relaxation** of $g(x,y) \le 0$:

$$g(x^1, y_L) + g(x^2, y_U) \le 0$$
$$\alpha x^L \le x^1 \le \alpha x^U$$
$$(1-\alpha)x^L \le x^2 \le (1-\alpha)x^U$$
$$x = x^1 + x^2$$

Its optimal value is a lower bound on the optimal value of the original problem, if cost is a function of x alone.

Truss Structure Design

Theorem (JNH 2005)

Suppose we minimize cx subject to $g(x,y) \le 0$.

If $g(x,y)$ is semihomogeneous in $x \in R^n$ and concave in scalar y, then the following is a **quasi-relaxation** of $g(x,y) \le 0$:

$$g(x^1, y_L) + g(x^2, y_U) \le 0$$
$$\alpha x^L \le x^1 \le \alpha x^U$$
$$(1-\alpha)x^L \le x^2 \le (1-\alpha)x^U$$
$$x = x^1 + x^2$$

$g(\alpha x, y) \le \alpha g(x,y)$ for all x, y and $\alpha \in [0,1]$

$g(0,y) = 0$ for all y

Truss Structure Design

Theorem (JNH 2005)

Suppose we minimize cx subject to $g(x,y) \leq 0$.

If $g(x,y)$ is semihomogeneous in $x \in R^n$ and concave in scalar y, then the following is a **quasi-relaxation** of $g(x,y) \leq 0$:

$$g(x^1, y_L) + g(x^2, y_U) \leq 0$$
$$\alpha x^L \leq x^1 \leq \alpha x^U$$
$$(1-\alpha)x^L \leq x^2 \leq (1-\alpha)x^U$$
$$x = x^1 + x^2$$

Bounds on y

Truss Structure Design

Theorem (JNH 2005)

Suppose we minimize cx subject to $g(x,y) \le 0$.

If $g(x,y)$ is semihomogeneous in $x \in R^n$ and concave in scalar y, then the following is a **quasi-relaxation** of $g(x,y) \le 0$:

$$g(x^1, y_L) + g(x^2, y_U) \le 0$$
$$\alpha x^L \le x^1 \le \alpha x^U$$
$$(1-\alpha)x^L \le x^2 \le (1-\alpha)x^U$$
$$x = x^1 + x^2$$

Bounds on x

Truss Structure Design

Theorem (JNH 2005)

Suppose we minimize cx subject to $g(x,y) \le 0$.

If $g(x,y)$ is semihomogeneous in $x \in R^n$ and concave in scalar y, then the following is a **quasi-relaxation** of $g(x,y) \le 0$:

$$g(x^1, y_L) + g(x^2, y_U) \le 0$$
$$\alpha x^L \le x^1 \le \alpha x^U$$
$$(1-\alpha)x^L \le x^2 \le (1-\alpha)x^U$$
$$x = x^1 + x^2$$

$\frac{E_i}{h_i} A_i v_i = s_i$ has the form $g(x,y) = 0$ with g semihomogenous in x because we can write it $\frac{E_i}{h_i} A_i v_i - s_i = 0$

with $x = (A_i, s_i)$, $y = v_i$.

Truss Structure Design

So we have a quasi-relaxation of the truss problem:

$$\min \quad \sum_i h_i [A_i^L y_i + A_i^U (1 - y_i)]$$

$$s.t. \quad \sum_i \cos\theta_{ij} s_i = p_j, \text{ all } j$$

$$\sum_j \cos\theta_{ij} d_j = v_{i0} + v_{i1}, \text{ all } i$$

$$\frac{E_i}{h_i}(A_i^L v_{i0} + A_i^U v_{i1}) = s_i, \text{ all } i$$

Hooke's law is linearized

$$v_i^L y_i \le v_{i0} \le v_i^U y_i, \text{ all } i$$

$$v_i^L (1 - y_i) \le v_{i1} \le v_i^U (1 - y_i), \text{ all } i$$

Elongation bounds split into 2 sets of bounds

$$d_j^L \le d_j \le d_j^U, \text{ all } j$$

$$0 \le y_i \le 1, \text{ all } i$$

Truss Structure Design

Logic cuts

v_{i0} and v_{i1} must have same sign in a feasible solution.

If not, we branch by adding logic cuts

$$v_{i0}, v_{i1} \leq 0, \qquad v_{i0}, v_{i1} \geq 0$$

Truss Structure Design

SIMPL model

Recognized as linear systems

```
01. OBJECTIVE
02.   maximize sum i of c[i]*h[i]*A[i]
03. CONSTRAINTS
04.   equilibrium means {
05.     sum i of b[i,j]*s[i,l] = p[j,l] forall j,l
06.     relaxation = { lp } }
07.   compatibility means {
08.     sum j of b[i,j]*d[j,l] = v[i,l] forall i,l
09.     relaxation = { lp } }
10.   hooke means {
11.     E[i]/h[i]*A[i]*v[i,l] = s[i,l] forall i
12.     relaxation = { lp:quasi } }
13. SEARCH
14.   type = { bb:bestdive }
15.   branching = { hooke:first:quasicut, A:splitup }
```

Truss Structure Design

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of c[i]*h[i]*A[i]
03. CONSTRAINTS
04.   equilibrium means {
05.     sum i of b[i,j]*s[i,l] = p[j,l] forall j,l
06.     relaxation = { lp } }
07.   compatibility means {
08.     sum j of b[i,j]*d[j,l] = v[i,l] forall i,l
09.     relaxation = { lp } }
10.   hooke means {
11.     E[i]/h[i]*A[i]*v[i,l] = s[i,l] forall i
12.     relaxation = { lp:quasi } }
13. SEARCH
14.   type = { bb:bestdive }
15.   branching = { hooke:first:quasicut, A:splitup }
```

Recognized as bilinear system

Truss Structure Design

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of c[i]*h[i]*A[i]
03. CONSTRAINTS
04.   equilibrium means {
05.     sum i of b[i,j]*s[i,l] = p[j,l] forall j,l
06.     relaxation = { lp } }
07.   compatibility means {
08.     sum j of b[i,j]*d[j,l] = v[i,l] forall i,l
09.     relaxation = { lp } }
10.   hooke means {
11.     E[i]/h[i]*A[i]*v[i,l] = s[i,l] forall i
12.     relaxation = { lp:quasi } }
13. SEARCH
14.   type = { bb:bestdive }
15.   branching = { hooke:first:quasicut, A:splitup }
```

Generate quasi-relaxation for semihomogenous function

Truss Structure Design

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of c[i]*h[i]*A[i]
03. CONSTRAINTS
04.   equilibrium means {
05.     sum i of b[i,j]*s[i,l] = p[j,l] forall j,l
06.     relaxation = { lp } }
07.   compatibility means {
08.     sum j of b[i,j]*d[j,l] = v[i,l] forall i,l
09.     relaxation = { lp } }
10.   hooke means {
11.     E[i]/h[i]*A[i]*v[i,l] = s[i,l] forall i
12.     relaxation = { lp:quasi } }
13. SEARCH
14.   type = { bb:bestdive }
15.   branching = { hooke:first:quasicut, A:splitup }
```

Branch first on violated logic cuts for quasi-relaxation

Truss Structure Design

SIMPL model

```
01. OBJECTIVE
02.   maximize sum i of c[i]*h[i]*A[i]
03. CONSTRAINTS
04.   equilibrium means {
05.     sum i of b[i,j]*s[i,l] = p[j,l] forall j,l
06.     relaxation = { lp } }
07.   compatibility means {
08.     sum j of b[i,j]*d[j,l] = v[i,l] forall i,l
09.     relaxation = { lp } }
10.   hooke means {
11.     E[i]/h[i]*A[i]*v[i,l] = s[i,l] forall i
12.     relaxation = { lp:quasi } }
13. SEARCH
14.   type = { bb:bestdive }
15.   branching = { hooke:first:quasicut, A:splitup }
```

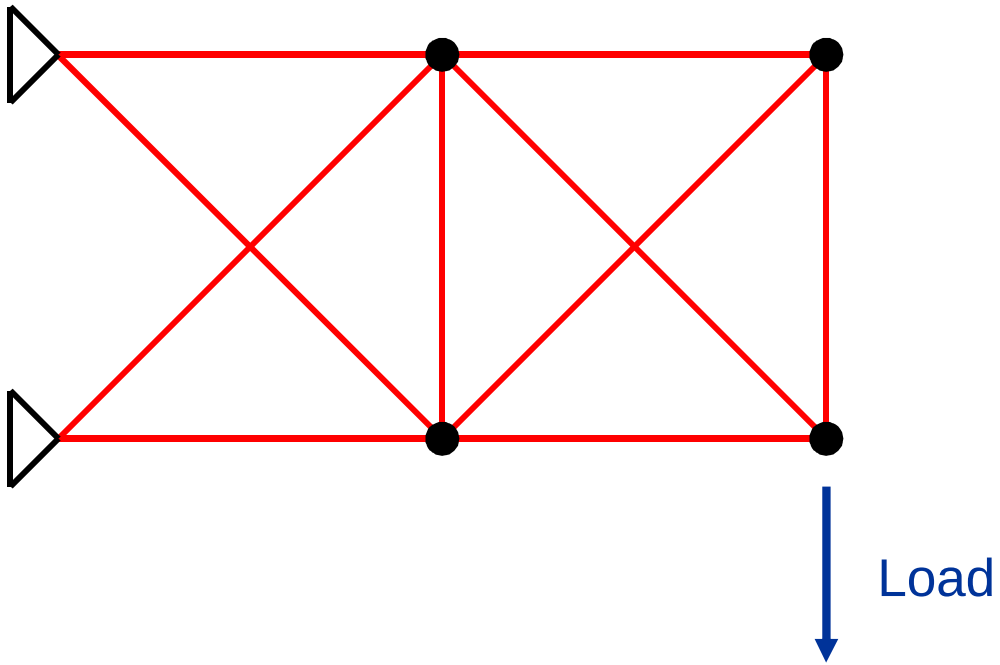
Then branch on A_i in-domain constraint.

Violated when A_i is not one of the discrete bar sizes.

Take upper branch first.

Truss Structure Design

10-bar cantilever truss



Truss Structure Design

Hand-coded integrated method

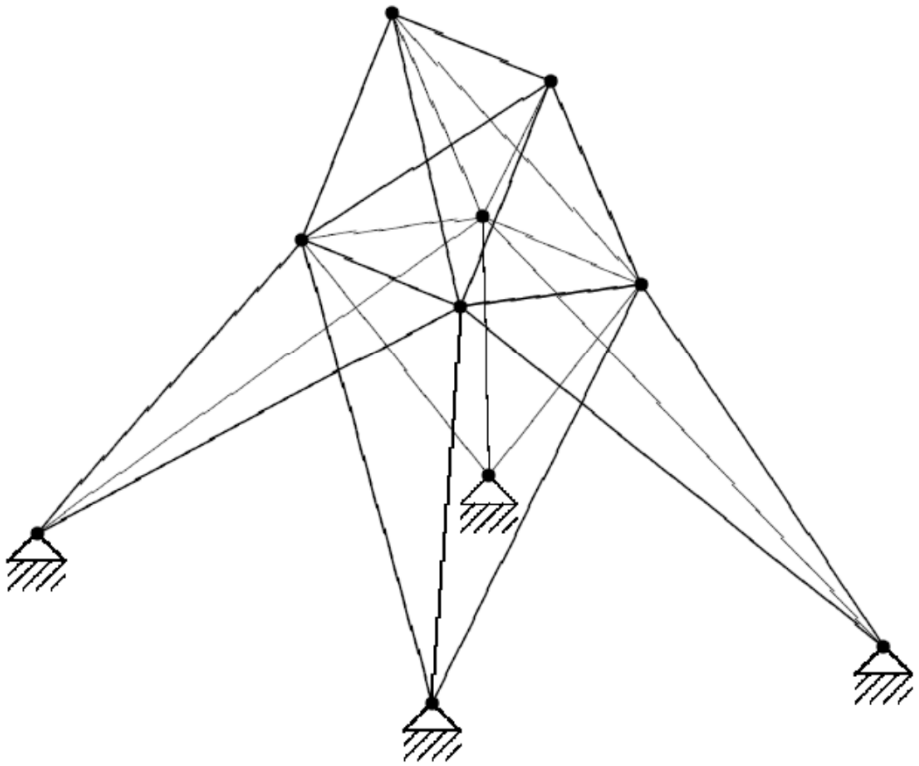
Computational results (seconds)

No. bars	Loads	BARON	CPLEX 11	Hand coded	SIMPL
10	1	5.3	0.40	0.03	0.08
10	1	3.8	0.26	0.02	0.07
10	1	8.1	0.83	0.16	0.49
10	1	8.8	1.2	0.22	0.63
10	2	24	4.9	0.64	1.84
10	2*	327	146	145	65
10	2*	2067	1087	600	651

*plus displacement bounds

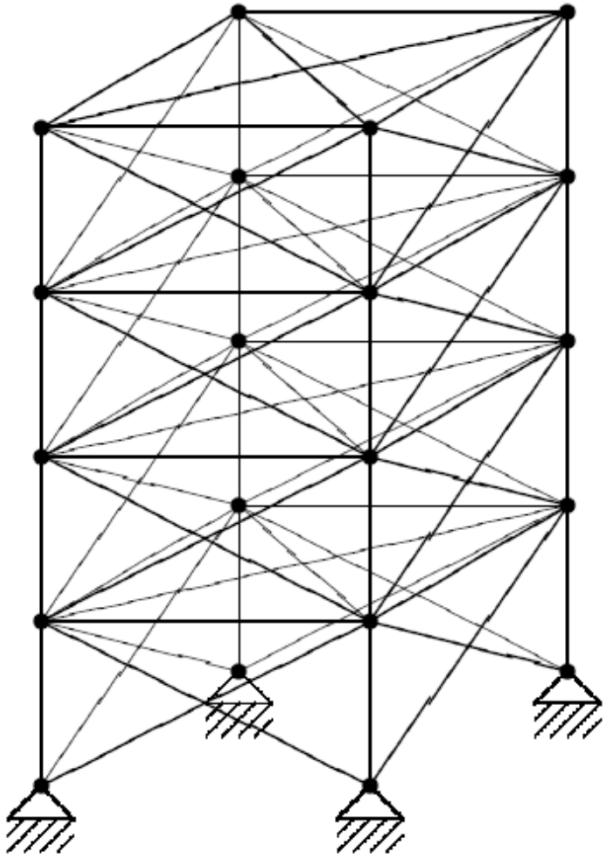
Truss Structure Design

25-bar problem



Truss Structure Design

72-bar problem



Truss Structure Design

Hand-coded integrated method

Computational results (seconds)

No. bars	Loads	BARON	CPLEX 11	Hand coded	SIMPL
25	2	3,302	44	44	20
72	2	3,376	208	33	28
90	2	21,011	570	131	92
108	2	> 24 hr*	3208	1907	1720
200	2	> 24 hr*	> 24 hr*	> 24 hr**	> 24 hr***

* no feasible solution found

** best feasible solution has cost 32,748

*** best feasible solution has cost 32,700

Summary

- We can understand intractability as an **epistemic** notion.
  - **Ignorance** of the solution space.
  - No need to view certain problems as **inherently hard**.
- To **defeat** intractability, use methods with knowledge of problem structure.
  - In optimization, **inference** and **relaxation** techniques exploit structure…
  - …in the context of **primal-dual-dual** methods.

Summary

- We can understand intractability as an epistemic notion.
    - Relaxation: Enumerate **master problems** (which assign jobs to machines)
- *Inference:* Generate **nogoods** (logic-based Benders cuts), which are added to master problem.