Constraint Programming

Tutorial

Air Products and Chemicals, Inc.

April 2006

John Hooker

Carnegie Mellon University

Outline

- Overview
- Examples
  - Freight transfer
  - Traveling salesman problem
  - Continuous global optimization
  - Product configuration
  - Machine scheduling

Overview

- Basic idea of CP:
- Each **constraint** is viewed as (invoking) a **procedure**.
  - The procedure reduces the search space by reducing (**filtering**) the **variable domains**.
- MILP models are purely declarative.
  - Constraints merely describe what the solution should be like.

Overview

- CP can exploit problem **substructure**.
- A highly structured subset of constraints is represented by a single **global constraint**.
  - Each global constraint is associated with a specialized **filtering** or **domain** reduction algorithm.
- Domains reduced by one constraint are passed on to the next constraint for further reduction.
  - This is **constraint propagation**.

Overview

- Advantages of CP relative to MILP:
  - More powerful **modeling** framework.
  - Allows the solver to exploit problem **substructure**.
  - **More constraints** make the problem **easier**, even when they destroy overall problem structure.
  - Can use **multiple formulations** and link them with channeling constraints.
  - Good performance when constraints contain **only a few** variables, or with min/max objectives.
  - Good performance on **sequencing**, **resource-constrained scheduling**, combinatorial, or disjunctive problems for which the MILP relaxation tends to be **weak**.

Overview

- Disadvantages of CP relative to MILP:
  - User must draw from **large library** of global constraints.
  - User must think about the **solution** algorithm when writing the model (often true in MILP!).
  - Models may not be **transportable** from one solver to another.
  - Slower on highly-analyzed, highly structured problems (e.g., TSP).
  - Slower on problems that have good MILP relaxations (e.g., knapsack constraints, min cost objectives).

Overview

- Myth: CP is better on "highly constrained" problems.
  - A tight constraint with many variables (i.e., cost constraint) may make the problem insoluble for CP.
  - CP can be very fast on a loosely constrained problem if the constraints "propagate well."

Overview

- Integrated methods.
  - There is no need to decide between MILP and CP.
  - **Combine** them to exploit complementary strengths.
  - Some of the following examples will suggest how to do this.

Overview

- Similarity between MILP/CP solvers:
  - Both use **branching**. What happens at a node of the search tree is different.
- Differences:
  - CP seeks feasible rather than optimal solution.
    - A minor difference, since CP easily incorporates optimization.
  - CP relies on **filtering & propagation** at a node, whereas MILP relies on **relaxation**.

MILP vs. CP Solution Approaches

	MILP	CP (feasibility)
Branching	Branch on variable with fractional value	Branch by splitting nonsingleton domain
Constraint store/ relaxation	Continuous relaxation (LP, Lagrangean, etc.)	Variable domains
Inference	Cutting planes, Benders cuts	Domain filtering, propagation, nogoods

MILP vs. CP Solution Approaches

	MILP	CP (feasibility)
Bounding	Use bound from continuous relaxation.	None.
Feasible solution obtained at a node…	When solution of relaxation is integral.	When domains are singletons.
Search backtracks…	When relaxation is infeasible, has integral solution, or tree pruned by bound	When at least one domain is empty.

Outline

- Example: **Freight transfer**
  - Bounds propagation
  - Bounds consistency
  - Knapsack cuts
  - Linear Relaxation
  - Branching search

Outline

- Example: **Traveling salesman problem**
  - Filtering for **all-different** constraint
  - Relaxation of **all-different**
  - Arc consistency

Outline

- Example: **Continuous global optimization**
  - Nonlinear bounds propagation
  - Function factorization
  - Interval splitting
  - Lagrangean propagation
  - Reduced-cost variable fixing

Outline

- Example: **Product configuration**
  - Variable indices
  - Filtering for **element** constraint
  - Relaxation of **element**
  - Relaxing a disjunction of linear systems

Outline

- Example: **Machine scheduling**
  - Edge finding
  - Benders decomposition and nogoods

Example: Freight Transfer

- Bounds propagation
- Bounds consistency
- Knapsack cuts
- Linear Relaxation
- Branching search

Freight Transfer

- Transport 42 tons of freight using at most 8 trucks.
- Trucks have 4 different sizes: 7,5,4, and 3 tons.
- How many trucks of each size should be used to minimize cost?
- x_j = number of trucks of size *j*.

Cost of size 4 truck

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

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

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

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

Freight Transfer

- We will solve the problem by branching search.
- At each node of the search tree:
  - Reduce the variable domains using *bounds propagation.*
  - Solve a *linear relaxation* of the problem to get a bound on the optimal value.

Bounds Propagation

- The *domain* of x_j is the set of values x_j can have in a some feasible solution.
  - Initially each x_j has domain {0,1,2,3}.
- *Bounds propagation* reduces the domains.
  - Smaller domains result in less branching.

Bounds Propagation

- First reduce the domain of x_1:

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

$$\Downarrow$$

$$x_1 \geq \left\lceil \frac{42 - 5x_2 + 4x_3 + 3x_4}{7} \right\rceil$$

$$\Downarrow$$

Max element of domain

$$x_1 \geq \left\lceil \frac{42 - 5 \cdot 3 - 4 \cdot 3 - 3 \cdot 3}{7} \right\rceil = \left\lceil \frac{6}{7} \right\rceil = 1$$

- So domain of x_1 is reduced from {0,1,2,3} to {1,2,3}.
- No reductions for x_2, x_3, x_4 possible.

Bounds Propagation

- In general, let the domain of x_i be $\{L_i, \ldots, U_i\}$.
- An inequality $ax \geq b$ (with $a \geq 0$) can be used to raise L_i to

$$\max\left\{L_i, b - \frac{\sum_{j \neq i} a_j U_j}{a_i}\right\}$$

- An inequality $ax \leq b$ (with $a \geq 0$) can be used to reduce U_i to

$$\min\left\{U_i, b - \frac{\sum_{j \neq i} a_j L_j}{a_i}\right\}$$

Bounds Propagation

- Now **propagate** the reduced domains to the other constraint, perhaps reducing domains further:

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

$$\Downarrow$$

$$x_1 \le \left\lceil \frac{8 - x_2 - x_3 - x_4}{1} \right\rceil$$

$$\Downarrow$$

$$x_1 \le \left\lceil \frac{8 - 1 - 0 - 0}{1} \right\rceil = \left\lceil \frac{7}{1} \right\rceil = 7$$

Min element of domain

- No further reduction is possible.

Bounds Consistency

- Again let $\{L_j, \ldots, U_j\}$ be the domain of x_j
- A constraint set is **bounds consistent** if for each j:
  - $x_j = L_j$ in some feasible solution and
  - $x_j = U_j$ in some feasible solution.
- Bounds consistency $\Rightarrow$ we will not set x_j to any infeasible values during branching.
- Bounds propagation achieves bounds consistency for a **single inequality**.
  - $7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$ is bounds consistent when the domains are $x_1 \in \{1,2,3\}$ and $x_2, x_3, x_4 \in \{0,1,2,3\}$.
- But not necessarily for a **set** of inequalities.

Bounds Consistency

- Bounds propagation may not achieve consistency for a set.
- Consider set of inequalities

$$x_1 + x_2 \geq 1$$

$$x_1 - x_2 \geq 0$$

with domains x_1, $x_2 \in \{0,1\}$, solutions $(x_1,x_2) = (1,0), (1,1)$.

- Bounds propagation has no effect on the domains.

From $x_1 + x_2 \geq 1$:

$$x_1 \geq \lceil 1 - U_2 \rceil = \lceil 1 - 1 \rceil = 0$$

$$x_2 \geq \lceil 1 - U_1 \rceil = \lceil 1 - 1 \rceil = 0$$

From $x_1 - x_2 \geq 1$:

$$x_1 \geq \lceil 1 - U_2 \rceil = \lceil 1 - 1 \rceil = 0$$

$$x_2 \leq \lfloor 1 + L_1 \rfloor = \lfloor 1 + 0 \rfloor = 1$$

- Constraint set is not bounds consistent because $x_1 = 0$ in no feasible solution.

Knapsack Cuts

- Inequality constraints (**knapsack** constraints) imply **cutting planes**.
- Cutting planes make the linear relaxation tighter, and its solution is a stronger bound.
- For the ≥ constraint, each **maximal packing** corresponds to a cutting plane (**knapsack cut**).

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

These terms form a maximal packing because:

(a) They alone cannot sum to ≥ 42, even if each x_j takes the largest value in its domain (3).

(b) No superset of these terms has this property.

Knapsack Cuts

- This maximal packing:

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

Min value of $4x_3 + 3x_4$ needed to satisfy inequality

corresponds to a knapsack cut:

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

Coefficients of x_3, x_4

Knapsack Cuts

- In general, J is a packing for $ax \geq b$ (with $a \geq 0$) if

$$\sum_{j \in J} a_j U_j < b$$

- If J is a packing for $ax \geq b$, then the remaining terms must cover the gap:

$$\sum_{j \notin J} a_j x_j \geq b - \sum_{j \in J} a_j U_j$$

- So we have a knapsack cut:

$$\sum_{j \notin J} x_j \geq \left\lceil \frac{b - \sum_{j \in J} a_j U_j}{\max_{j \notin J} \{a_j\}} \right\rceil$$

Knapsack Cuts

- Valid knapsack cuts for

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

are

$$x_3 + x_4 \geq 2$$
$$x_2 + x_4 \geq 2$$
$$x_2 + x_3 \geq 3$$

Linear Relaxation

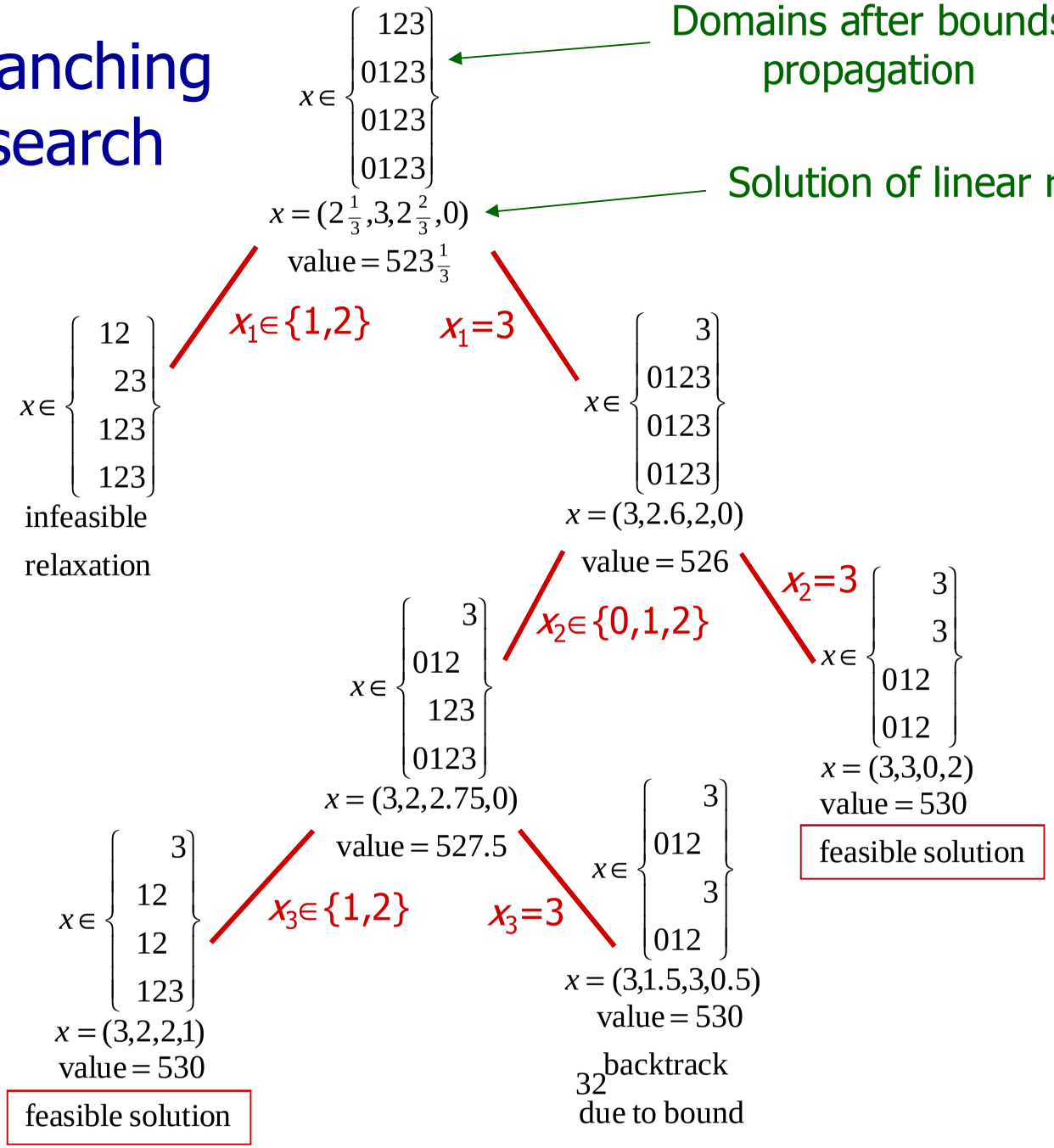
- We now have a linear relaxation of the freight transfer problem:

$$\min 90x_1 + 60x_2 + 50x_3 + 40x_4$$
$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$$
$$x_1 + x_2 + x_3 + x_4 \leq 8$$
$$x_3 + x_4 \geq 2$$
$$x_2 + x_4 \geq 2$$
$$x_2 + x_3 \geq 3$$
$$L_j \leq x_j \leq U_j$$

Branching Search

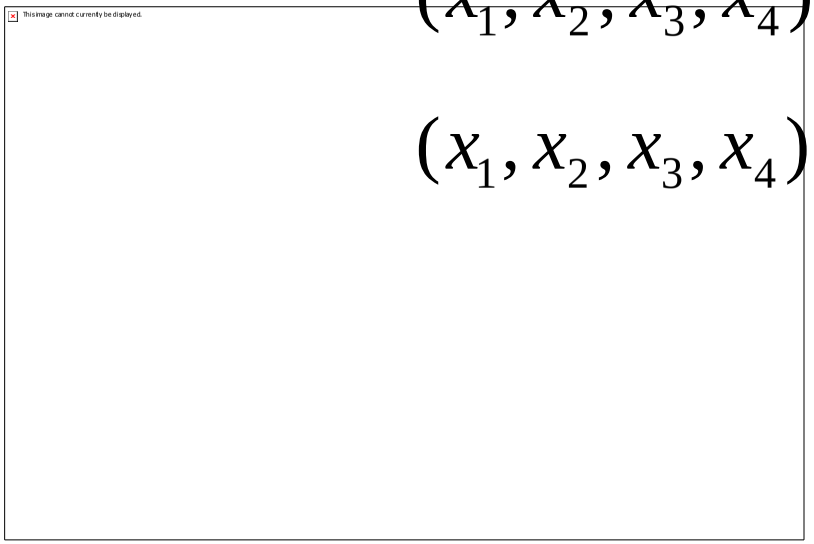
- At each node of the search tree:
  - Reduce domains with bounds propagation.
  - Solve a linear relaxation to obtain a lower bound on any feasible solution in the subtree rooted at current node.
- If relaxation is infeasible, or its optimal value is no better than the best feasible solution found so far, **backtrack**.
- Otherwise, if solution of relaxation is feasible, remember it and **backtrack**.
- Otherwise, **branch** by splitting a domain.

Branching search



Branching Search

- Two optimal solutions found (cost = 530):



$$(x_1, x_2, x_3, x_4) = (3,2,2,1)$$

$$(x_1, x_2, x_3, x_4) = (3,3,0,2)$$

Example: Traveling Salesman

- Filtering for **all-different** constraint
- Relaxation of **all-different**
- Arc consistency

Traveling Salesman

- Salesman visits each city once.
- Distance from city i to city j is c_{ij}.
- Minimize distance traveled.
- x_i = ith city visited.

Distance from ith city visited to next city

(city n+1 = city 1)

$$\min \sum_i c_{x_i x_{i+1}}$$

$$\text{all-different}(x_1, \ldots, x_n)$$

$$x_j \in \{1, \ldots, n\}$$

- Can be solved by branching + domain reduction + relaxation.

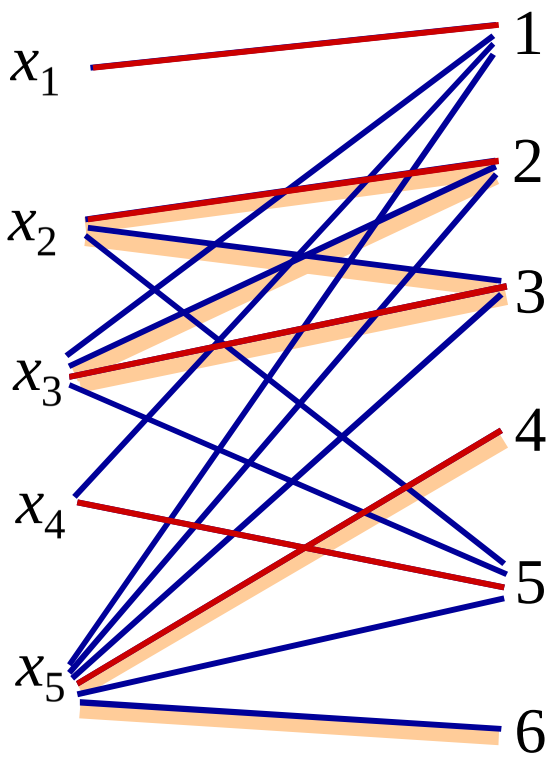
Filtering for All-different

- Goal: **filter** infeasible values from variable domains.
  - Filtering reduces the domains and therefore reduces branching.
- The best known filtering algorithm for all-different is based on **maximum cardinality bipartite matching** and a theorem of Berge.

Filtering for All-different

- Consider all-different(x_1,x_2,x_3,x_4,x_5) with domains

$$x_1 \in \{1\}$$
$$x_2 \in \{2,3,5\}$$
$$x_3 \in \{1,2,3,5\}$$
$$x_4 \in \{1,5\}$$
$$x_5 \in \{1,2,3,4,5,6\}$$



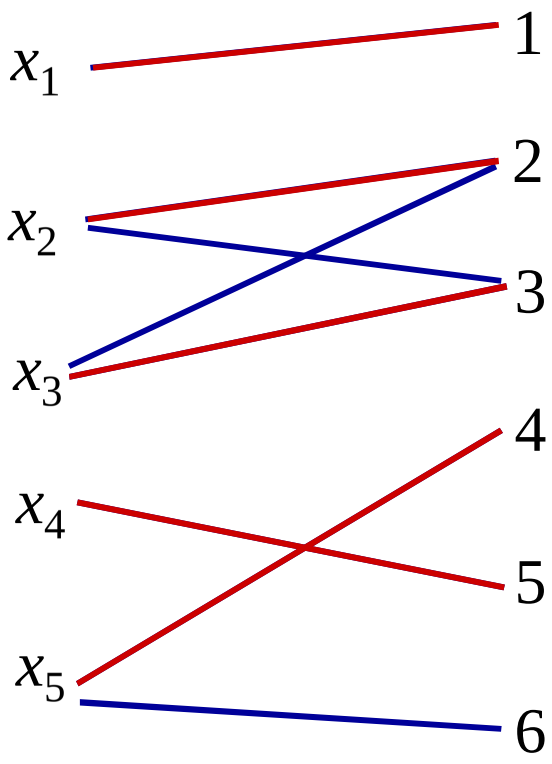
Indicate domains with edges

Find maximum cardinality bipartite matching.

Mark edges in alternating paths that start at an uncovered vertex.

Mark edges in alternating cycles.

Remove unmarked edges not in matching.



Indicate domains with edges

Find maximum cardinality bipartite matching.

Mark edges in alternating paths that start at an uncovered vertex.

Mark edges in alternating cycles.

Remove unmarked edges not in matching.

Filtering for All-different

$x_1 \in \{1\}$

$x_2 \in \{2,3,5\} \Rightarrow x_2 \in \{2,3\}$

$x_3 \in \{1,2,3,5\} \Rightarrow x_3 \in \{2,3\}$

$x_4 \in \{1,5\} \Rightarrow x_4 \in \{5\}$

$x_5 \in \{1,2,3,4,5,6\} \Rightarrow x_5 \in \{4,6\}$

Relaxation of All-different

- All-different(x_1,x_2,x_3,x_4) with domains $x_j \in \{1,2,3,4\}$ has the **convex hull** relaxation:

$$x_1 + x_2 + x_3 + x_4 = 1+2+3+4$$

$$\left\{\begin{array}{l} x_1 + x_2 + x_3 \\ x_1 + x_2 + x_4 \\ x_1 + x_3 + x_4 \\ x_2 + x_3 + x_4 \end{array}\right\} \geq 1+2+3$$

$$x_j \geq 1$$

$$\left\{\begin{array}{l} x_1 + x_2 \\ x_1 + x_3 \\ x_1 + x_4 \\ x_2 + x_3 \\ x_2 + x_4 \\ x_3 + x_4 \end{array}\right\} \geq 1+2$$

- This is the tightest possible linear relaxation, but it may be too weak (and have too many inequalities) to be useful.

Arc Consistency

- A constraint set S containing variables $x_1, \ldots, x_n$ with domains $D_1, \ldots, D_n$ is **arc consistent** if the domains contain no infeasible values.
- That is, for every j and every $v \in D_j$, $x_j = v$ in some feasible solution of S.
  - This is actually **generalized arc consistency** (since there may be more than 2 variables per constraint).
- Arc consistency can be achieved by filtering all infeasible values from the domains.

Arc Consistency

- The matching algorithm achieves arc consistency for all-different.
- Practical filtering algorithms often do not achieve full arc consistency, since it is not worth the time investment.
- The **primary tools of constraint programming** are filtering algorithms that achieve or approximate arc consistency.
    - Filtering algorithms are analogous to cutting plane algorithms in integer programming.

Arc Consistency

- Domains that are reduced by a filtering algorithm for one constraint can be **propagated** to other constraints.
  - Similar to bounds propagation.
- But propagation may not achieve arc consistency for a constraint **set**, even if it achieves arc consistency for each individual constraint.
  - Again, similar to bounds propagation.

Arc Consistency

- For example, consider the constraint set

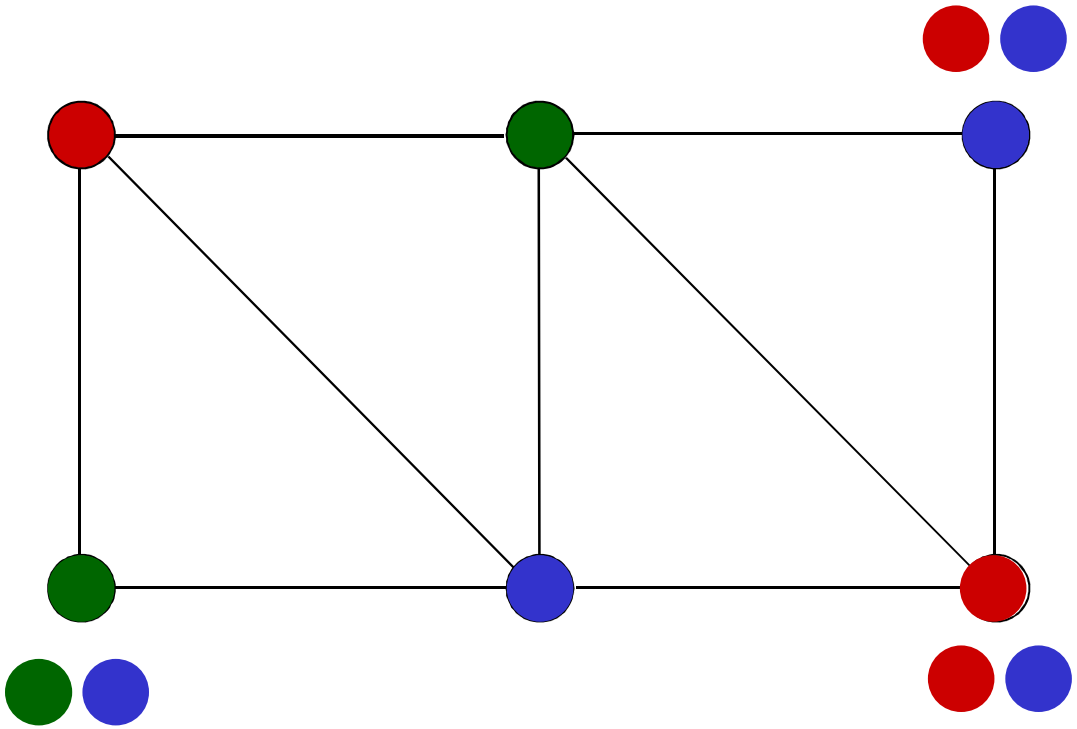
$$\text{all-different}(x_1, x_2) \qquad x_1 \in \{1,2\ \}$$
$$\text{all-different}(x_1, x_3) \quad \text{with domains} \quad x_2 \in \{\ 2,3\}$$
$$\text{all-different}(x_2, x_3) \qquad x_3 \in \{\ 2,3\}$$

- No further domain reduction is possible.
  - Each constraint is arc consistent for these domains.
- But $x_1 = 2$ in no feasible solution of the constraint set.
  - So the constraint set is not arc consistent.

Arc Consistency

- On the other hand, sometimes arc consistency maintenance alone can solve a problem—without branching.
- Consider a graph coloring problem.
  - Color the vertices red, blue or green so that adjacent vertices have different colors.
  - These are **binary** constraints (they contain only 2 variables).
  - Arbitrarily color two adjacent vertices red and green.
  - Reduce domains to maintain arc consistency for each constraint.

Graph coloring problem that can be solved by arc consistency maintenance alone.



Example: Continuous Global Optimization

- Nonlinear bounds propagation
- Function factorization
- Interval splitting
- Lagrangean propagation
- Reduced-cost variable fixing

Continuous Global Optimization

- Today's **constraint programming solvers** (CHIP, ILOG Solver) emphasize **propagation**.
- Today's **integer programming solvers** (CPLEX, Xpress-MP) emphasize **relaxation**.
- Current **global solvers** (BARON, LGO) already combine propagation and relaxation.
- Perhaps in the near future, **all solvers** will combine propagation and relaxation.

Continuous Global Optimization

- Consider a continuous global optimization problem:

$$\max x_1 + x_2$$
$$2x_1 + x_2 \le 2$$
$$4x_1x_2 = 1$$
$$x_1 \in [0,1], \quad x_2 \in [0,2]$$

- The feasible set is **nonconvex**, and there is more than one local optimum.
- **Nonlinear programming** techniques try to find a local optimum.
- Global optimization techniques try to find a **global optimum**.

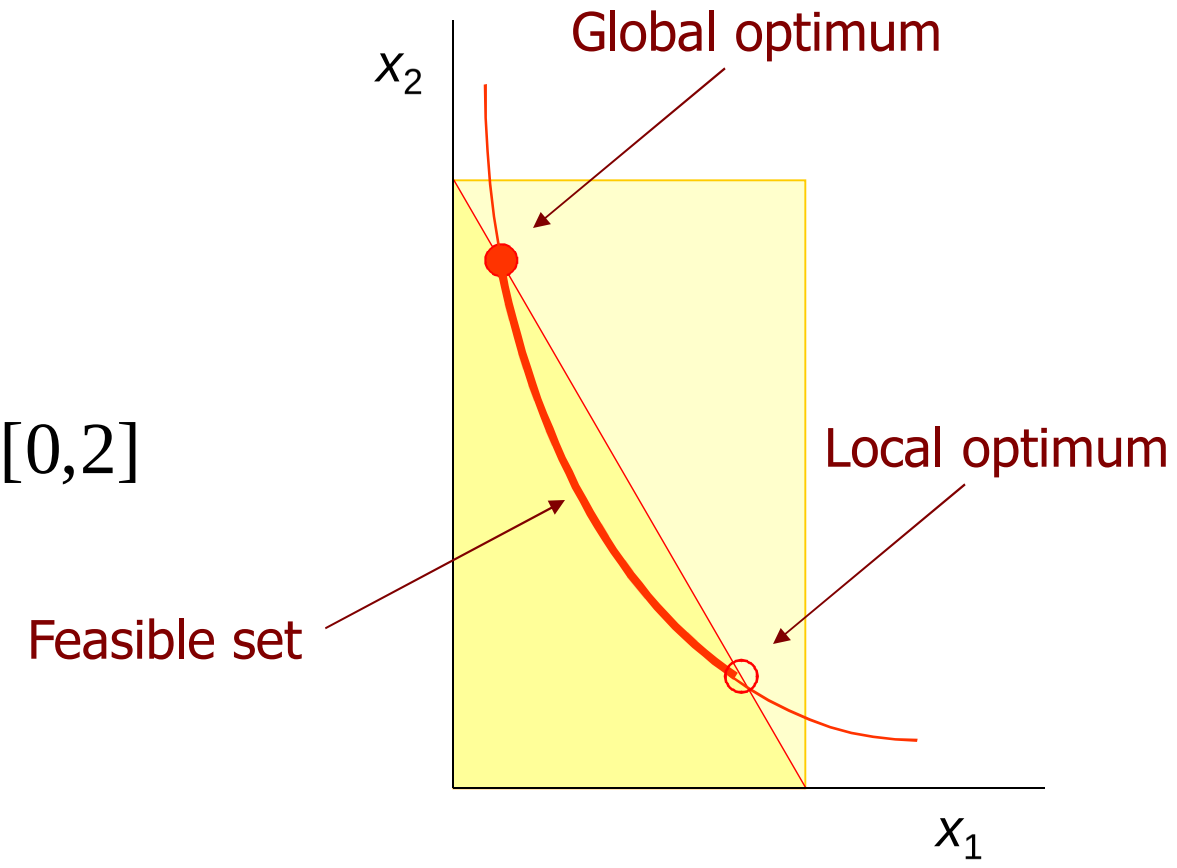
Continuous Global Optimization

$$\max x_1 + x_2$$

$$4x_1x_2 = 1$$

$$2x_1 + x_2 \leq 2$$

$$x_1 \in [0,1],\ x_2 \in [0,2]$$



Continuous Global Optimization

- **Branch** by splitting continuous interval domains.
- **Reduce domains** with:
  - Nonlinear bounds propagation.
  - Lagrangean propagation (reduced cost variable fixing)..
- **Relax** the problem by factoring functions.

Nonlinear Bounds Propagation

- To propagate $4x_1x_2 = 1$, solve for $x_1 = 1/4x_2$ and $x_2 = 1/4x_1$.

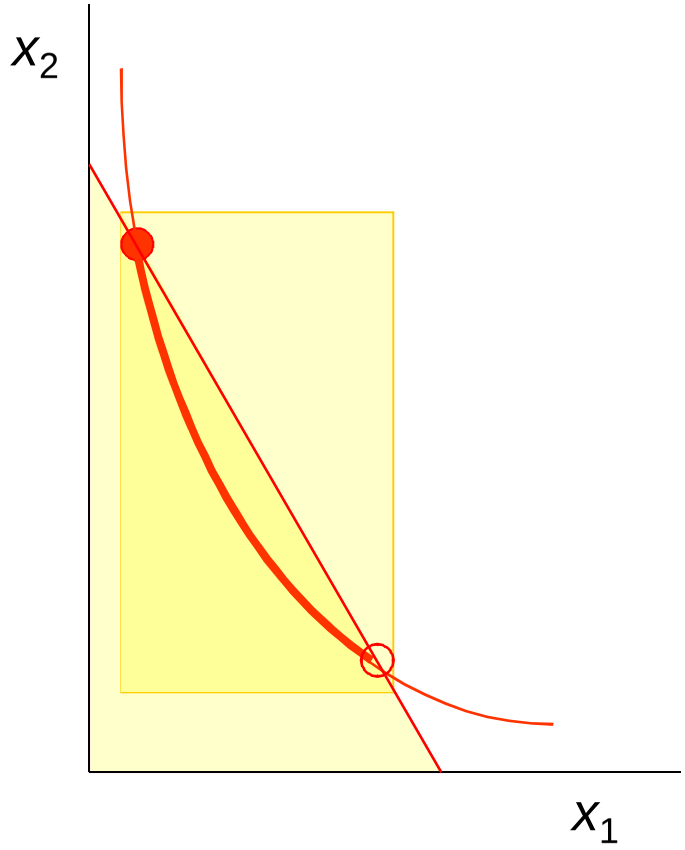
Then:

$$x_1 \geq \frac{1}{4U_2} = \frac{1}{4 \cdot 2} = \frac{1}{8} \qquad x_2 \geq \frac{1}{4U_1} = \frac{1}{4 \cdot 1} = \frac{1}{4}$$

- This yields domains $x_1 \in [0.125,1]$, $x_2 \in [0.25,2]$.
- Further propagation of $2x_1 + x_2 \leq 2$ yields $x_1 \in [0.125,0.875]$, $x_2 \in [0.25,1.75]$.
- Cycling through the 2 constraints converges to the fixed point $x_1 \in [0.146,0.854]$, $x_2 \in [0.293,1.707]$.
  - In practice, this process is terminated early, due to decreasing returns for computation time,

Nonlinear Bounds Propagation

Propagate intervals
[0,1], [0,2]
through constraints
to obtain
[1/8,7/8], [1/4,7/4]



Function Factorization

- Factor complex functions into elementary functions that have known linear relaxations.
- Write $4x_1x_2 = 1$ as $4y = 1$ where $y = x_1x_2$.
  - This factors $4x_1x_2$ into linear function $4y$ and bilinear function x_1x_2.
  - Linear function $4y$ is its own linear relaxation.
  - Bilinear function $y = x_1x_2$ has relaxation:

$$L_2x_1 + L_1x_2 - L_1L_2 \le y \le L_2x_1 + U_1x_2 - U_1L_2$$
$$U_2x_1 + U_1x_2 - U_1U_2 \le y \le U_2x_1 + L_1x_2 - L_1U_2$$

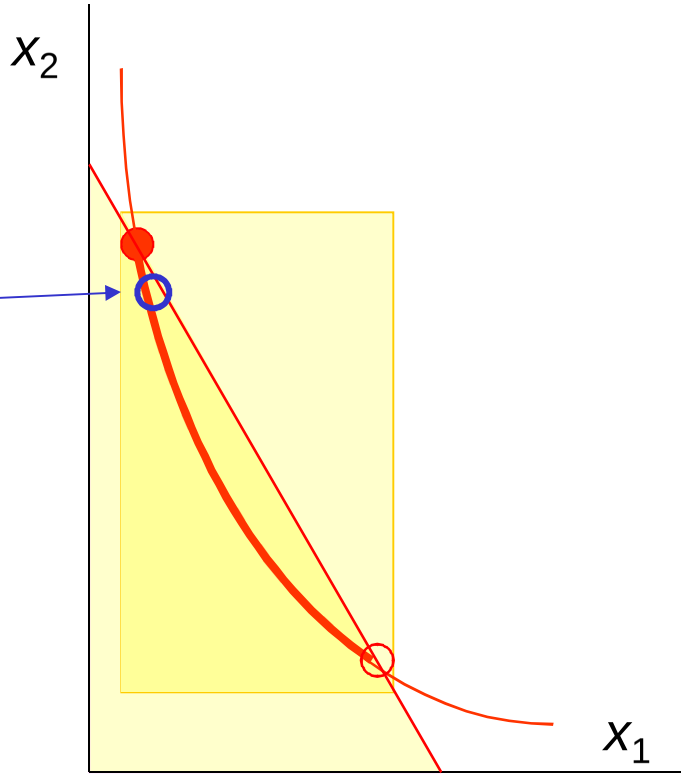
Function Factorization

- We now have a linear relaxation of the nonlinear problem:

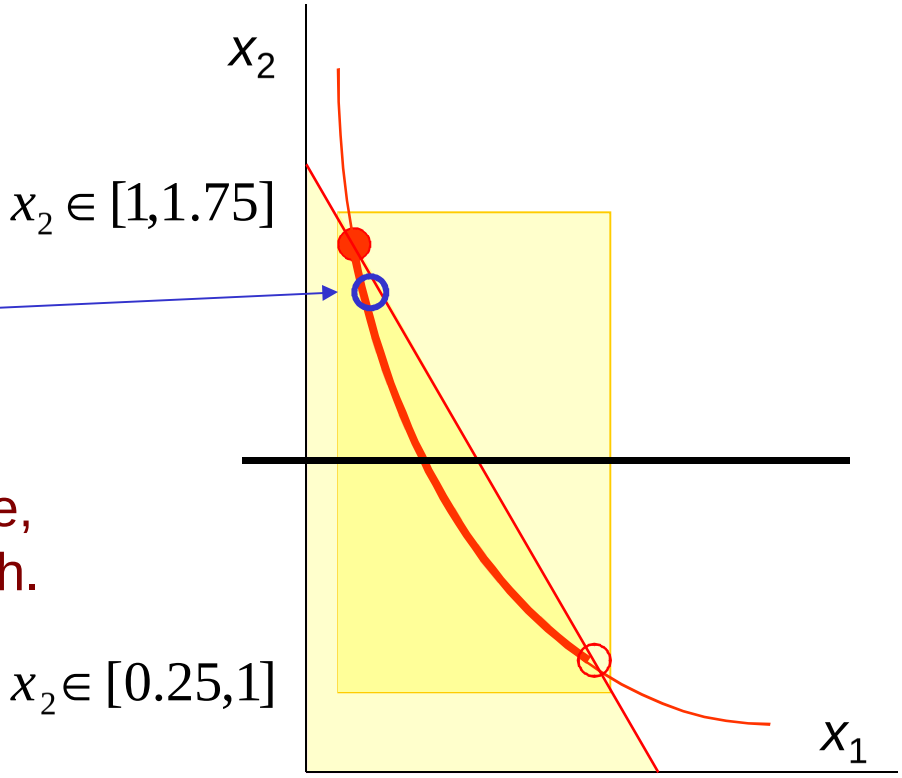
$$\max x_1 + x_2$$
$$4y = 1$$
$$2x_1 + x_2 \le 2$$
$$L_2 x_1 + L_1 x_2 - L_1 L_2 \le y \le L_2 x_1 + U_1 x_2 - U_1 L_2$$
$$U_2 x_1 + U_1 x_2 - U_1 U_2 \le y \le U_2 x_1 + L_1 x_2 - L_1 U_2$$
$$L_j \le x_j \le U_j$$

Interval Splitting

Solve linear relaxation.

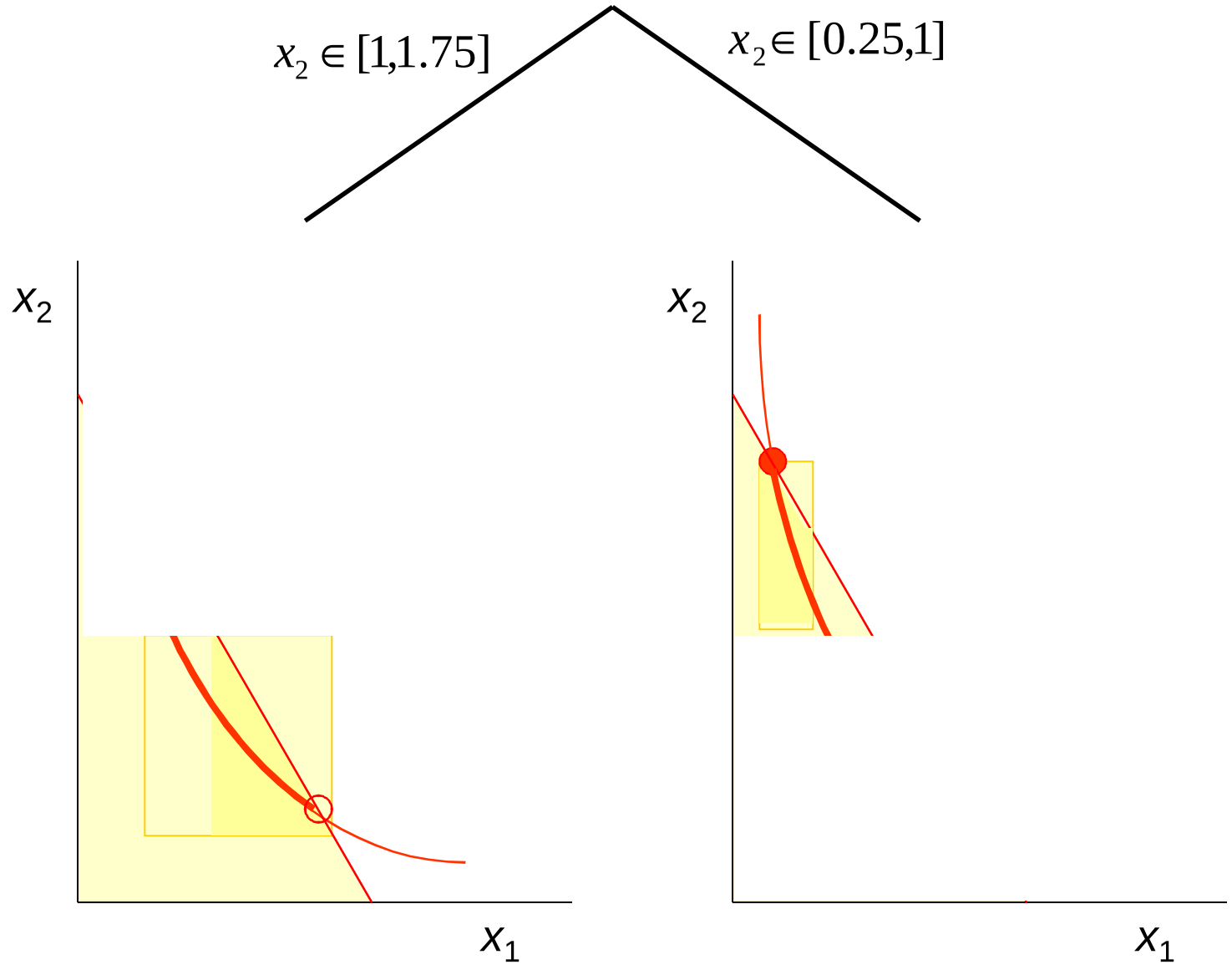


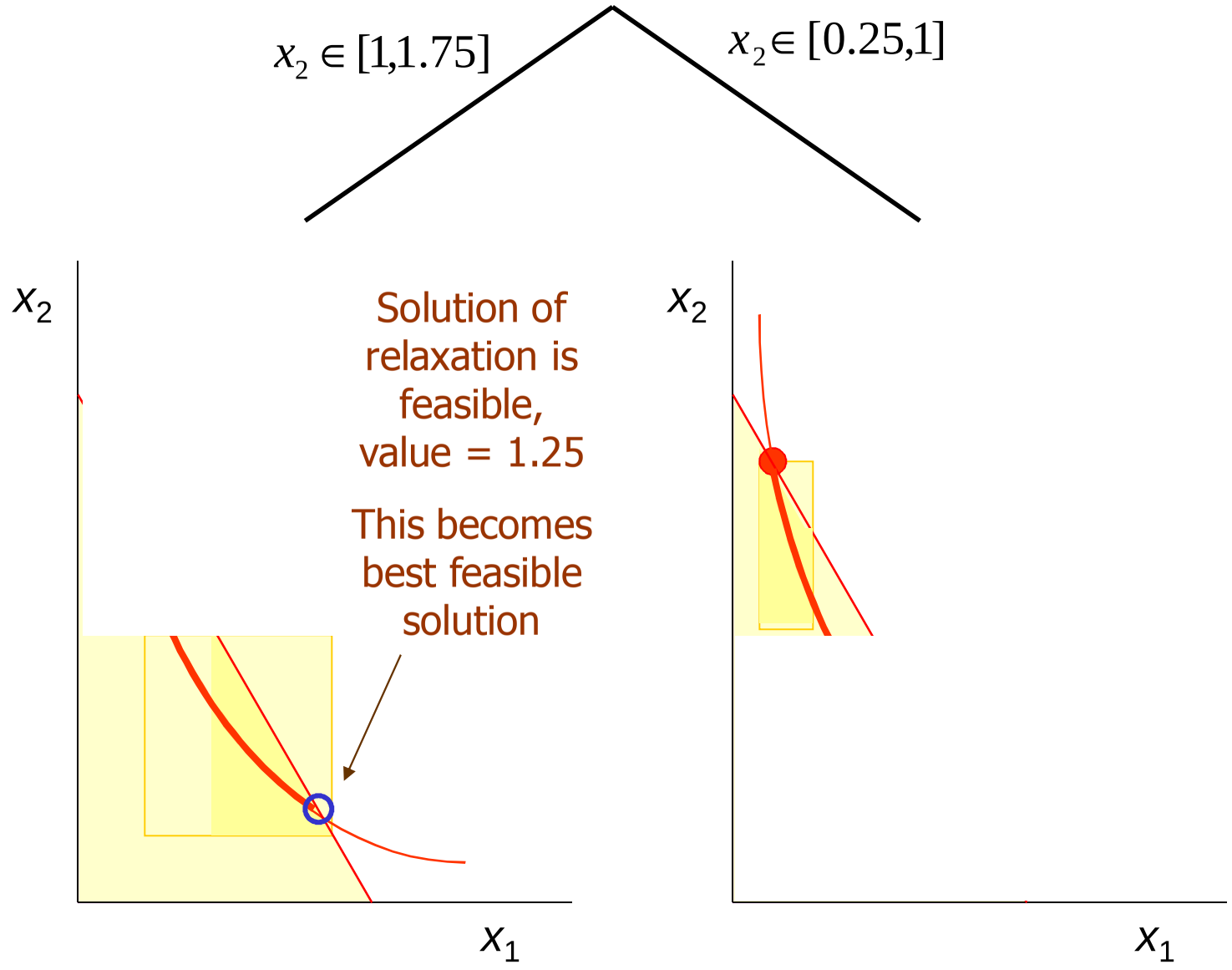
Interval Splitting

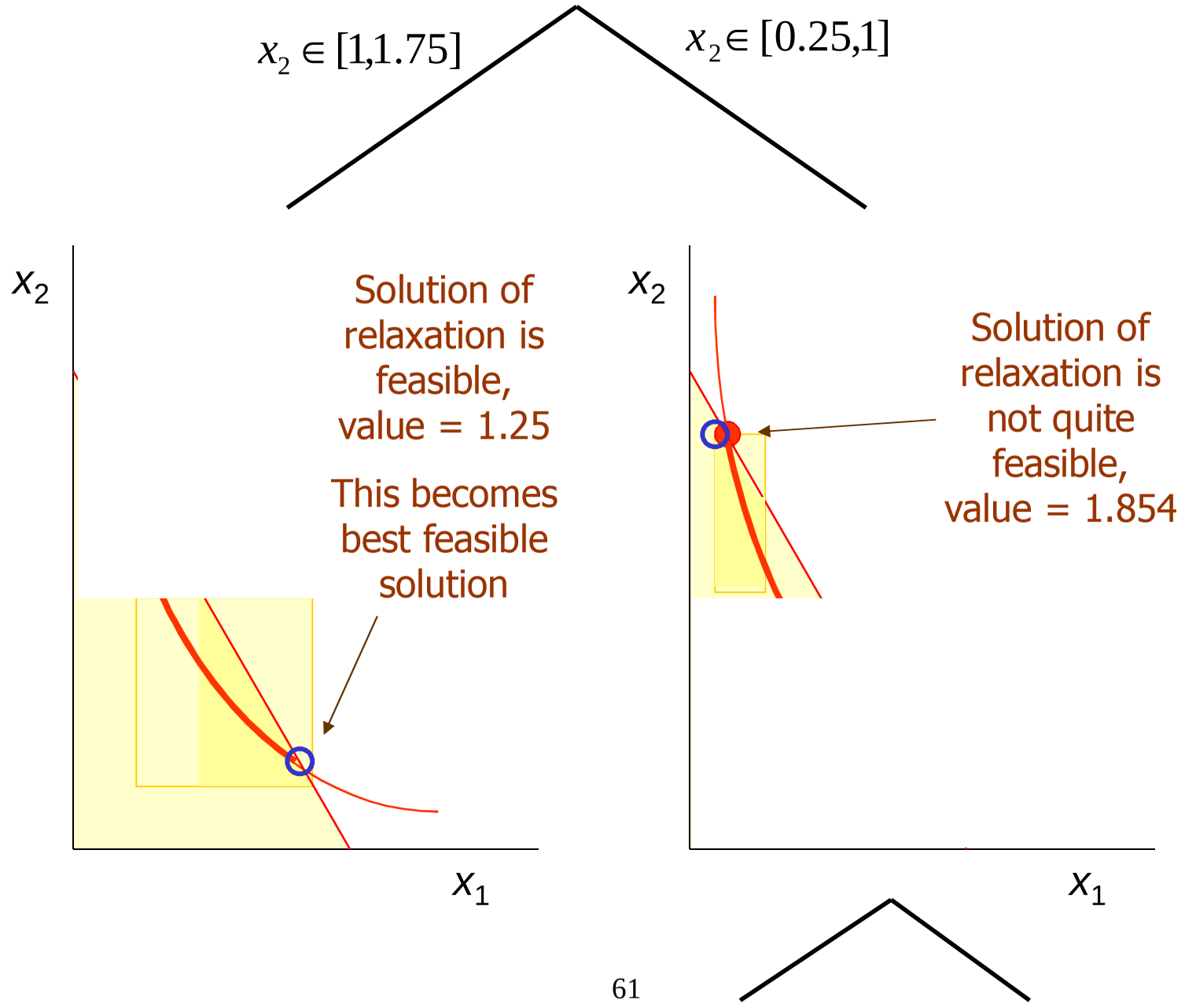


Solve linear relaxation.

Since solution is infeasible, split an interval and branch.


$x_2 \in [1,1.75]$
$x_2 \in [0.25,1]$
x_2
x_1
x_2
x_1


$x_2 \in [1,1.75]$
$x_2 \in [0.25,1]$
x_2
Solution of relaxation is feasible, value = 1.25
This becomes best feasible solution
x_1
x_2
x_1


$x_2 \in [1,1.75]$
$x_2 \in [0.25,1]$
x_2
Solution of relaxation is feasible, value = 1.25
This becomes best feasible solution
x_1
x_2
Solution of relaxation is not quite feasible, value = 1.854
x_1

Lagrangean Propagation

$$\max x_1 + x_2$$

Optimal value of relaxation is 1.854

$$4y = 1$$

$$2x_1 + x_2 \le 2$$

Lagrange multiplier (dual variable) in solution of relaxation is 1.1

$$L_2 x_1 + L_1 x_2 - L_1 L_2 \le y \le L_2 x_1 + U_1 x_2 - U_1 L_2$$

$$U_2 x_1 + U_1 x_2 - U_1 U_2 \le y \le U_2 x_1 + L_1 x_2 - L_1 U_2$$

$$L_j \le x_j \le U_j$$

- Any reduction Δ in right-hand side of $2x_1 + x_2 \le 2$ reduces the optimal value of relaxation by 1.1 Δ.
- So any reduction Δ in the left-hand side (now 2) has the same effect.

Lagrangean Propagation

- Any reduction Δ in left-hand side of $2x_1 + x_2 \leq 2$ reduces the optimal value of the relaxation (now 1.854) by 1.1 Δ.
- The optimal value should not be reduced below the lower bound 1.25 (value of best feasible solution so far).
- So we should have $1.1\,\Delta \leq 1.854 - 1.25$, or

$$1.1[2-(2x_1+x_2)] \leq 1.854-1.25$$

$$\text{or } \; 2x_1+x_2 \geq 2-\frac{1.854-1.25}{1.1}=1.451$$

- This inequality can be propagated to reduce domains.
  - Reduces domain of x_2 from [1,1.714] to [1.166,1.714].

Reduced-cost Variable Fixing

- In general, given a constraint $g(x) \le b$ with Lagrange multiplier λ, the following constraint is valid:

Optimal value of relaxation

Lower bound on optimal value of original problem

$$g(x) \ge b - \frac{v - L}{\lambda}$$

- If $g(x) \le b$ is a nonnegativity constraint $-x_j \le 0$ with Lagrange multiplier λ (i.e, reduced cost of x_j is $-\lambda$), then

$$-x_j \ge -\frac{v - L}{\lambda} \qquad \text{or} \qquad x_j \le \frac{v - L}{\lambda}$$

- If x_j is a 0-1 variable and $(v - L)/\lambda < 1$, then we can fix x_j to 0. This is **reduced cost variable fixing**.

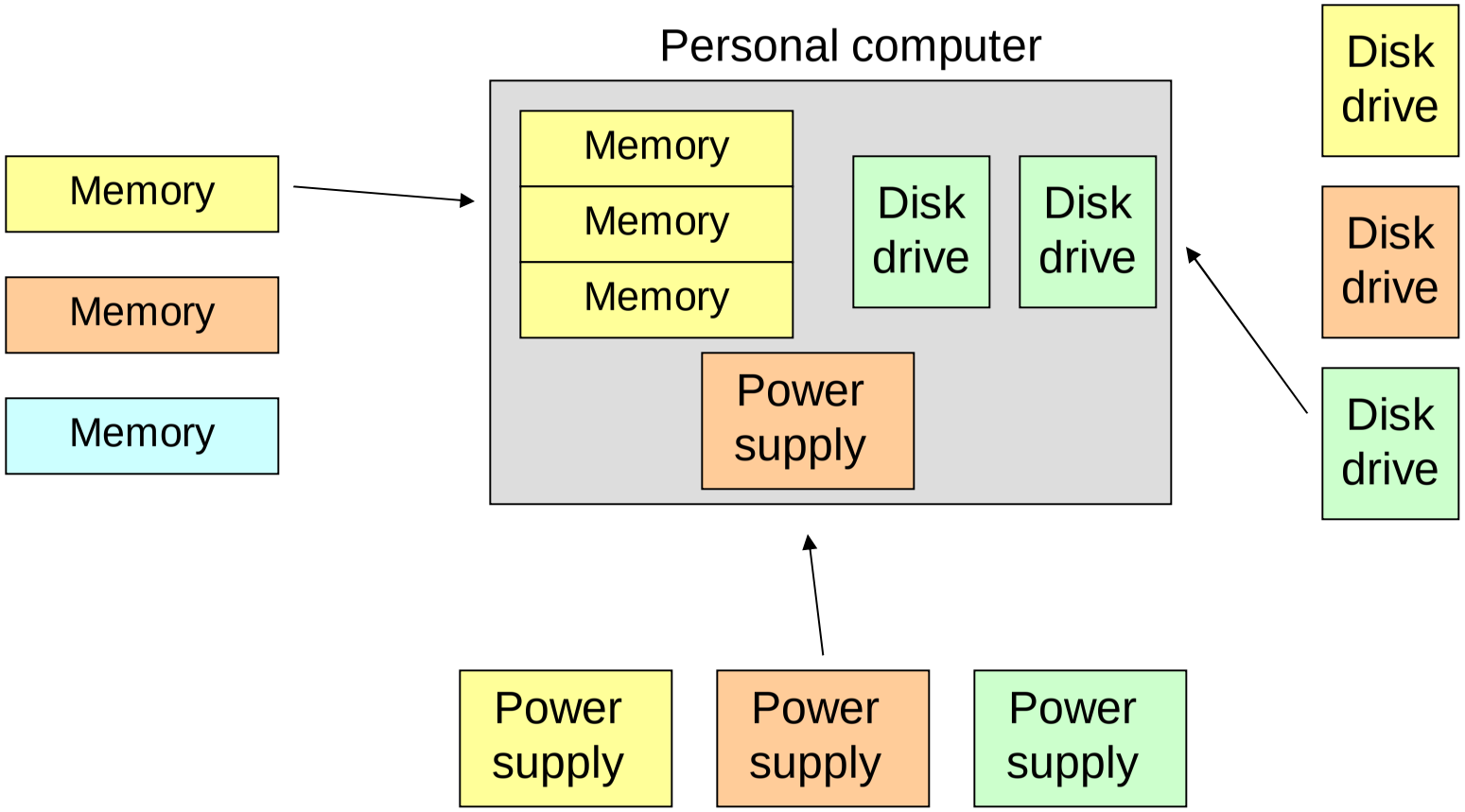
Example: Product Configuration

- Variable indices
- Filtering for **element** constraint
- Relaxation of **element**
- Relaxing a disjunction of linear systems

Product Configuration

- We want to configure a computer by choosing the type of power supply, the type of disk drive, and the type of memory chip.
- We also choose the number of disk drives and memory chips.
  - Use only 1 type of disk drive and 1 type of memory.
- Constraints:
  - Generate enough power for the components.
  - Disk space at least 700.
  - Memory at least 850.
- Minimize weight subject to these constraints.

Product Configuration



Product Configuration

Component i	Type k	Net power generation A_{1jk}	Disk space A_{2jk}	Memory capacity A_{3jk}	Weight A_{4jk}	Max number used
1. Power supply	A	70	0	0	200	1
	B	100	0	0	250	
	C	150	0	0	350	
2. Disk drive	A	−30	500	0	140	3
	B	−50	800	0	300	
3. Memory chip	A	−20	0	250	20	3
	B	−25	0	300	25	
	C	−30	0	400	30	
Lower bound L_j		0	700	850	0	
Upper bound U_j		∞	∞	∞	∞	
Unit cost c_j		0	0	0	1	

Product Configuration

- Let

  t_i = type of component i installed.

  q_i = quantity of component i installed.

- These are the problem variables.
- This problem will illustrate the **element** constraint.

Product Configuration

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

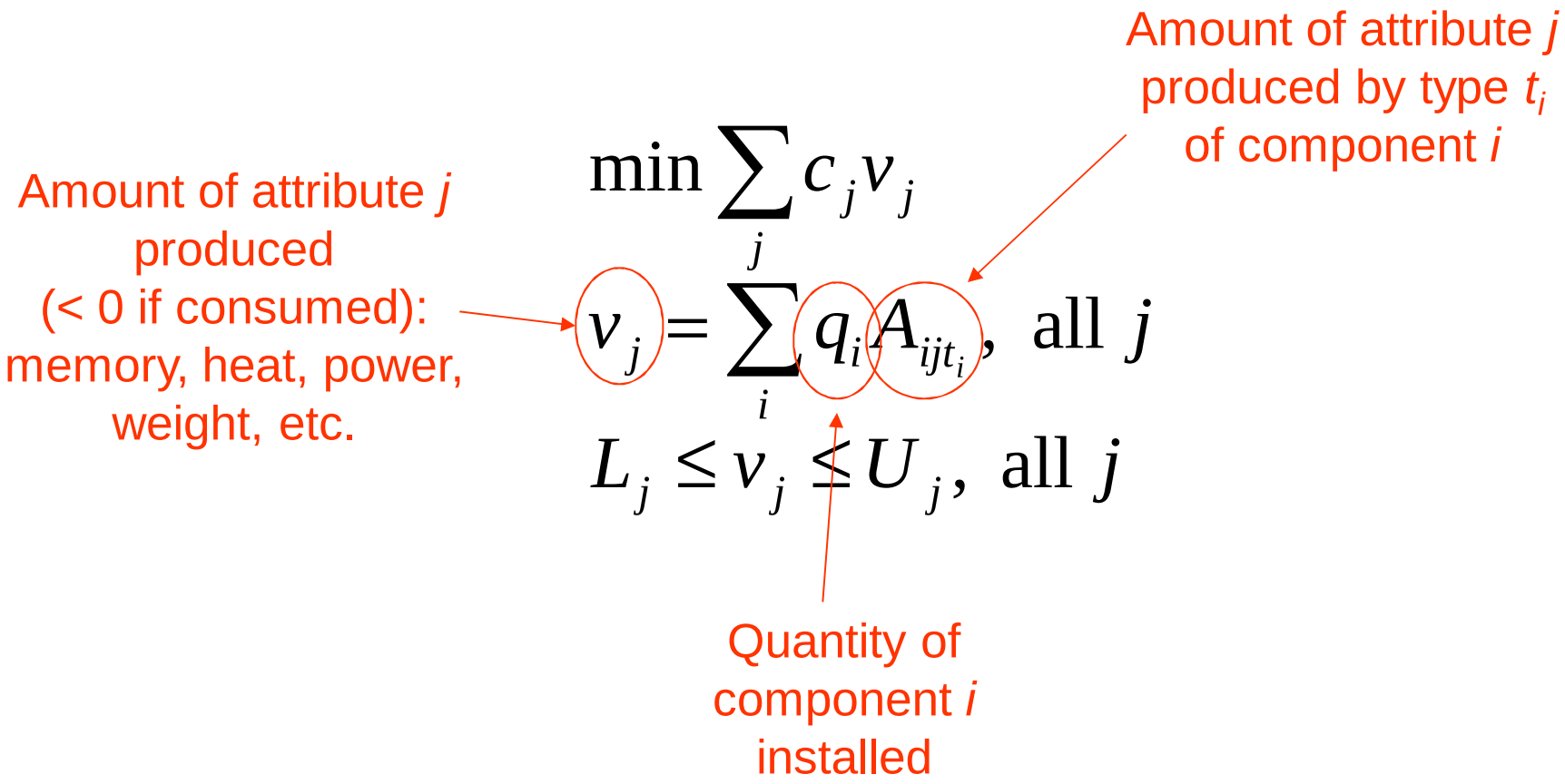
$$\min \sum_j c_j v_j$$

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

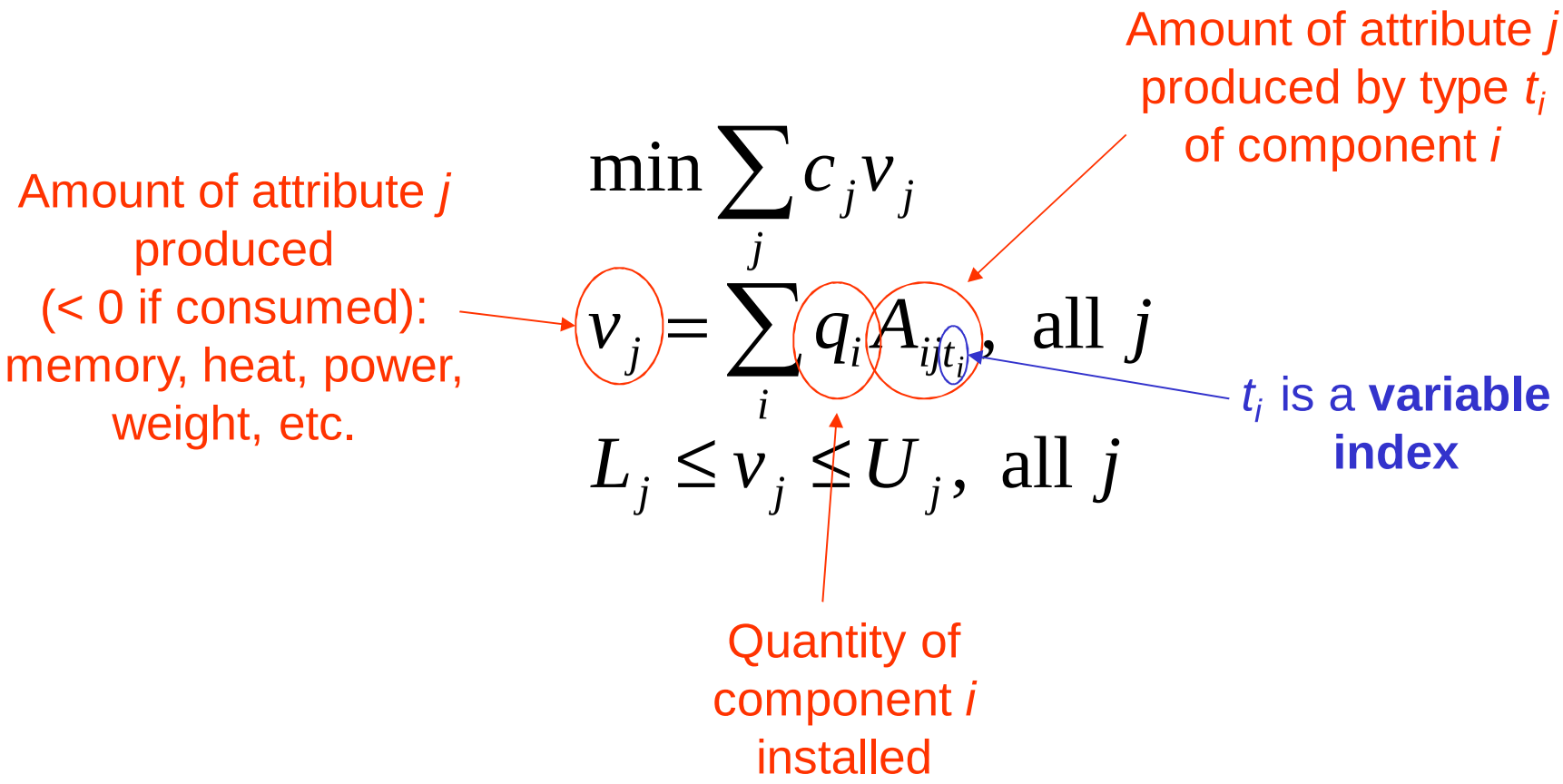
$$L_j \le v_j \le U_j, \text{ all } j$$

Quantity of component *i* installed

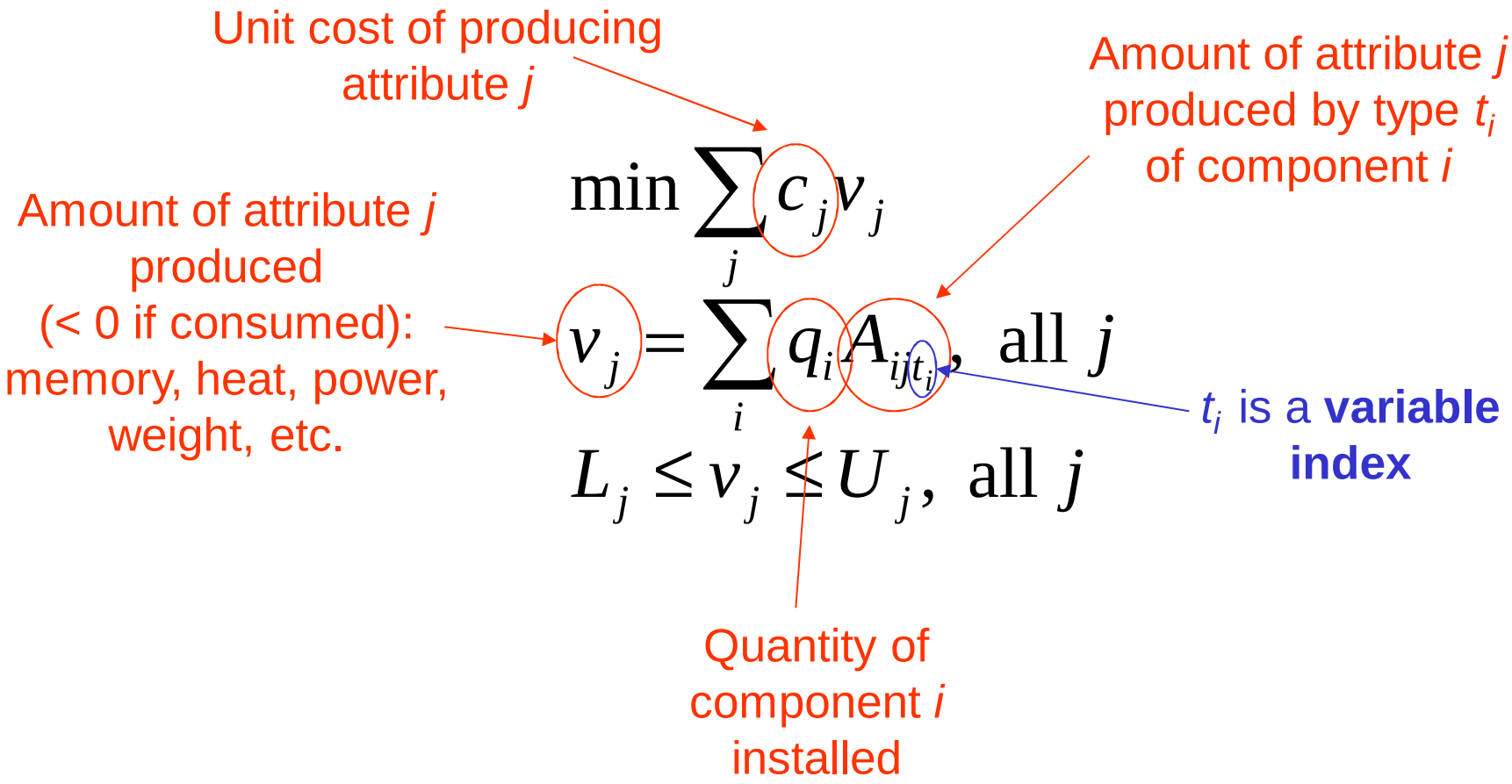
Product Configuration



Product Configuration



Product Configuration



Variable Indices

- Variable indices are implemented with the **element** constraint.
- The y in x_y is a **variable index**.
  - Constraint programming solvers implement x_y by replacing it with a new variable z and adding the constraint element(y,(x_1,…,x_n),z).
  - This constraint sets z equal to the y th element of the list (x_1,…,x_n).
- So $\sum_i q_i A_{ijt_i}$ is replaced by $\sum_i z_{ij}$ and the new constraint $\text{element}\left(t_i, (q_i A_{ij1}, \ldots, q_i A_{ijn}), z_{ij}\right)$ for each i.
- The element constraint can be propagated and relaxed.

Filtering for Element

$$\text{element}(y, (x_1, \ldots, x_n), z)$$

can be processed with a domain reduction algorithm that maintains arc consistency.

Domain of *z*

$$D_z \leftarrow D_z \cap \bigcup_{j \in D_y} D_{x_j}$$

$$D_y \leftarrow D_y \cap \{ j \mid D_x \cap D_{x_j} \neq \emptyset \}$$

$$D_{x_j} \leftarrow \begin{cases} D_z & \text{if } D_y = \{j\} \\ D_{x_j} & \text{otherwise} \end{cases}$$

Filtering for Element

Example... $\text{element}(y,(x_1,x_2,x_3,x_4),z)$

The initial domains are:

$$D_z = \{20,30,60,80,90\}$$
$$D_y = \{1,3,4\}$$
$$D_{x_1} = \{10,50\}$$
$$D_{x_2} = \{10,20\}$$
$$D_{x_3} = \{40,50,80,90\}$$
$$D_{x_4} = \{40,50,70\}$$

The reduced domains are:

$$D_z = \{80,90\}$$
$$D_y = \{3\}$$
$$D_{x_1} = \{10,50\}$$
$$D_{x_2} = \{10,20\}$$
$$D_{x_3} = \{80,90\}$$
$$D_{x_4} = \{40,50,70\}$$

Relaxation of Element

- element(y,(x_1,…,x_n),z) can be given a continuous relaxation.
- It implies the **disjunction** $\bigvee_k (z = x_k)$

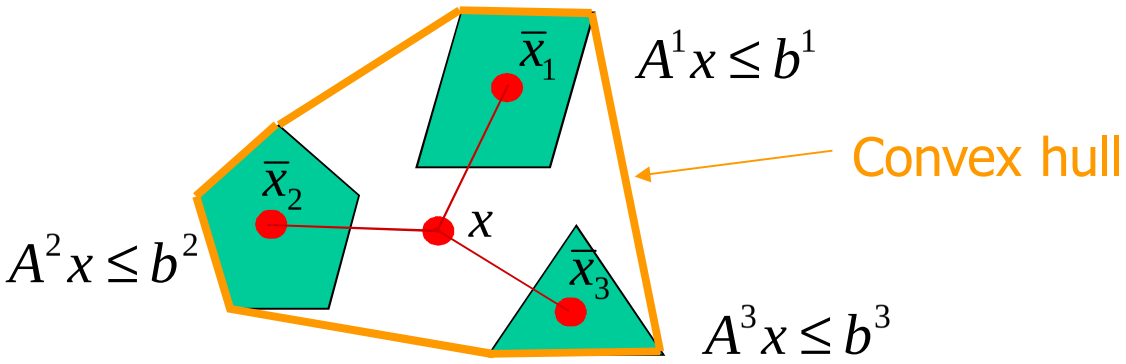
- In general, a disjunction of linear systems can be given a **convex hull** relaxation.
  - This provides a way to relax the element constraint.

Relaxing a Disjunction of Linear Systems

- Consider the disjunction

$$\bigvee_k (A^k x \le b^k)$$

- It describes a union of polyhedra defined by $A^k x \le b^k$.
- Every point x in the convex hull of this union is a convex combination of points $\bar{x}^k$ in the polyhedra.



Relaxing a Disjunction of Linear Systems

- So we have

$$x = \sum_k \alpha_k \bar{x}^k$$
$$A^k \bar{x}^k \le b^k, \ \text{all } k$$
$$\sum_k \alpha_k = 1, \quad \alpha_k \ge 0$$

Using the change of variable $x^k = \alpha_k \bar{x}^k$ we get the convex hull relaxation

$$x = \sum_k x^k$$
$$A^k x^k \le b^k \alpha_k, \ \text{all } k$$
$$\sum_k \alpha_k = 1, \quad \alpha_k \ge 0$$

Relaxation of Element

- The convex hull relaxation of element(y,(x_1, …, x_n), z) is the convex hull relaxation of the disjunction $\bigvee_k (z = x_k)$, which simplifies to

$$z = \sum_k x_{kk}$$

$$x_i = \sum_k x_{ik}, \text{ all } i$$

Product Configuration

- Returning to the product configuration problem, the model with the element constraint is

$$\min \sum_j c_j v_j$$

$$v_j = \sum_i z_{ij}, \;\; L_j \le v_j \le U_j, \quad \text{all } j$$

$$\text{element}\left(t_i, (A_{ij1}, \ldots, A_{ijm}), z_{ij}\right), \;\; \text{all } i, j$$

with domains

Type of power supply

Type of disk

Type of memory

$$t_1 \in \{A, B, C\}, \quad t_2 \in \{A, B\}, \quad t_3 \in \{A, B, C\}$$

$$v_j \in [L_j, U_j], \;\; \text{all } j$$

Number of power supplies

$$q_1 \in \{1\}, \quad q_2, q_3 \in \{1, 2, 3\}$$

Number of disks, memory chips

Product Configuration

- We can use knapsack cuts to help filter domains. Since

$v_j \in [L_j, U_j]$, the constraint $v_j = \sum_j z_{ij}$ implies $v_j^L \le \sum_j z_{ij} \le v_j^U$

where each z_{ij} has one of the values $q_i A_{ij1}, \ldots, q_i A_{ijm}$

- This implies the knapsack inequalities

$$L_j \le q_i \sum_i \max_k \{A_{ijk}\} \qquad q_i \sum_i \min_k \{A_{ijk}\} \le U_j$$

- These can be used to reduce the domain of q_i.

Product Configuration

- Using the lower bounds L_j, for power, disk space and memory, we have the knapsack inequalities

$$0 \le q_1 \max\{70,100,150\} + q_2 \max\{-30,-50\} + q_3 \max\{-20,-25,-30\}$$
$$700 \le q_1 \max\{0,0,0\} + q_2 \max\{500,800\} + q_3 \max\{0,0,0\}$$
$$850 \le q_1 \max\{0,0,0\} + q_2 \max\{0,0\} + q_3 \max\{250,300,400\}$$

which simplify to

$$0 \le 150q_1 - 30q_2 - 20q_3$$
$$700 \le 800q_2$$
$$850 \le 400q_3$$

- Propagation of these reduces domain of q_3 from {1,2,3} to {3}.

Product Configuration

- Propagation of all constraints reduces domains to

$q_1 \in \{1\}$	$q_2 \in \{1,2\}$	$q_3 \in \{3\}$	
$t_1 \in \{C\}$	$t_2 \in \{A, B\}$	$t_3 \in \{B, C\}$	
$z_{11} \in [150,150]$	$z_{21} \in [-75,-30]$	$z_{31} \in [-90,-75]$	
$z_{12} \in [0,0]$	$z_{22} \in [700,1600]$	$z_{32} \in [0,0]$	
$z_{13} \in [0,0]$	$z_{23} \in [0,0]$	$z_{33} \in [900,120]$	
$z_{14} \in [350,350]$	$z_{24} \in [140,600]$	$z_{34} \in [75,90]$	
$v_1 \in [0,45]$	$v_2 \in [700,1600]$	$v_3 \in [900,1200]$	$v_4 \in [565,1040]$

Product Configuration

- Using the convex hull relaxation of the element constraints, we have a linear relaxation of the problem:

$$\min \sum_j c_j v_j$$

$$v_j = \sum_i \sum_{k \in D_{t_i}} A_{ijk} q_{ik}, \text{ all } j$$

$$q_i = \sum_{k \in D_{t_i}} q_{ik}, \text{ all } i$$

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

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

$$v_j^L \le \sum_i \max_{k \in D_{t_i}} \{A_{ijk} q_i\}, \text{ all } j$$

$$\sum_i \min_{k \in D_{t_i}} \{A_{ijk} q_i\} \le v_j^U, \text{ all } j$$

$$q_{ik} \ge 0, \text{ all } i, k$$

$q_{ik} > 0$ when type k of component i is chosen, and q_{ik} is the quantity installed

Current bounds on v_j

Current bounds on q_i

Product Configuration

- The solution of the linear relaxation at the root node is

$q_1 = q_{1C} = 1$ ← Use 1 type C power supply (t_1 = C)

$q_2 = q_{2A} = 2$ ← Use 2 type A disk drives (t_2 = A)

$q_3 = q_{3B} = 3$ ← Use 3 type B memory chips (t_3 = B)

other q_{ij}s $= 0$

$(v_1, \ldots, v_4) = (15{,}1000{,}900{,}705)$ ← Min weight is 705.

- Since only one $q_{ik} > 0$ for each i, there is no need to branch on the t_is.
- This is a feasible and therefore optimal solution.
  - The problem is solved at the root node.

Example: Machine Scheduling

- Edge finding
- Benders decomposition and nogoods
- Cumulative Scheduling

Machine Scheduling

- We want to schedule 5 jobs on 2 machines.
  - Each job has a release time and deadline.
  - Processing times and costs differ on the two machines.
  - We want to minimize total cost while observing the time widows.
- We will first study propagation for the **1-machine** problem (**edge finding**).
- We will then solve the 2-machine problem using **Benders decomposition** and propagation on the 1-machine problem

Machine Scheduling

Job j	Release time r_j	Dead-line d_j	Processing time p_{Aj}	Processing time p_{Bj}	Processing cost f_{Aj}	Processing cost f_{Bj}
1	0	10	2	2	20	30
2	0	8	3	2	20	30
3	2	7	2	2	20	30
4	2	5	4	3	30	40
5	4	7	3	2	20	30

Processing time on machine A

Processing time on machine B

Machine Scheduling

- Jobs must run one at a time on each machine (**disjunctive scheduling**). For this we use the constraint

$$\text{disjunctive}(t, p)$$

$t = (t_1, \ldots t_n)$ are start times (variables)

$p = (p_{i1}, \ldots p_{in})$ are processing times (constants)

- The best known filtering algorithm for the disjunctive constraint is **edge finding**.
  - Edge finding does not achieve full arc or bounds consistency.

Machine Scheduling

- The problem can be written

$$\min \sum_j f_{y_j j}$$

$$t_j + p_{y_j j} \le d_j, \text{ all } j$$

$$\text{disjunctive}\big((t_j \mid y_j = i), (p_{ij} \mid y_j = i)\big)$$

Start time of job j

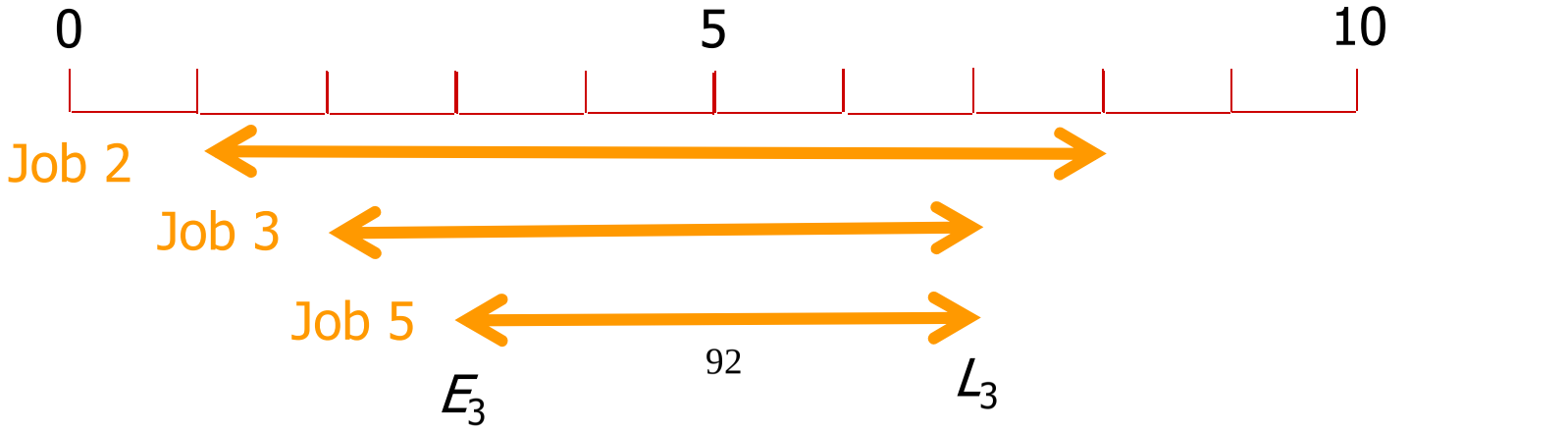
Machine assigned to job *j*

- We will first look at a scheduling problem on 1 machine.

Edge Finding

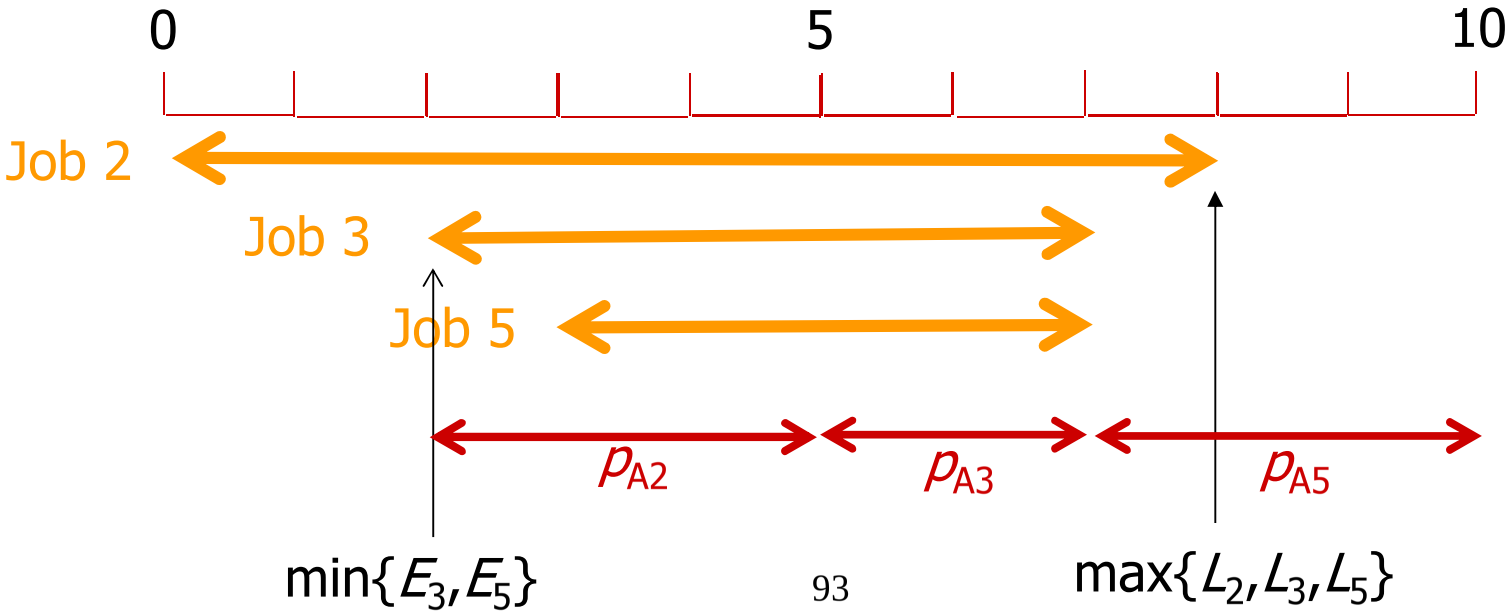
- Let's try to schedule jobs 2, 3, 5 on machine A.
  - Initially, the Earliest Start Time E_j and Latest End Time L_j are the release dates r_j and deadlines d_j:

$$[L_2, E_2] = [0,8]$$
$$[L_3, E_3] = [2,7]$$
$$[L_5, E_5] = [3,7]$$



Edge Finding

- Job 2 must precede jobs 3 and 5, because:
  - Jobs 2, 3, 5 will not fit between the earliest start time of 3, 5 and the latest end time of 2, 3, 5.
  - Therefore job 2 must end before jobs 3, 5 start.

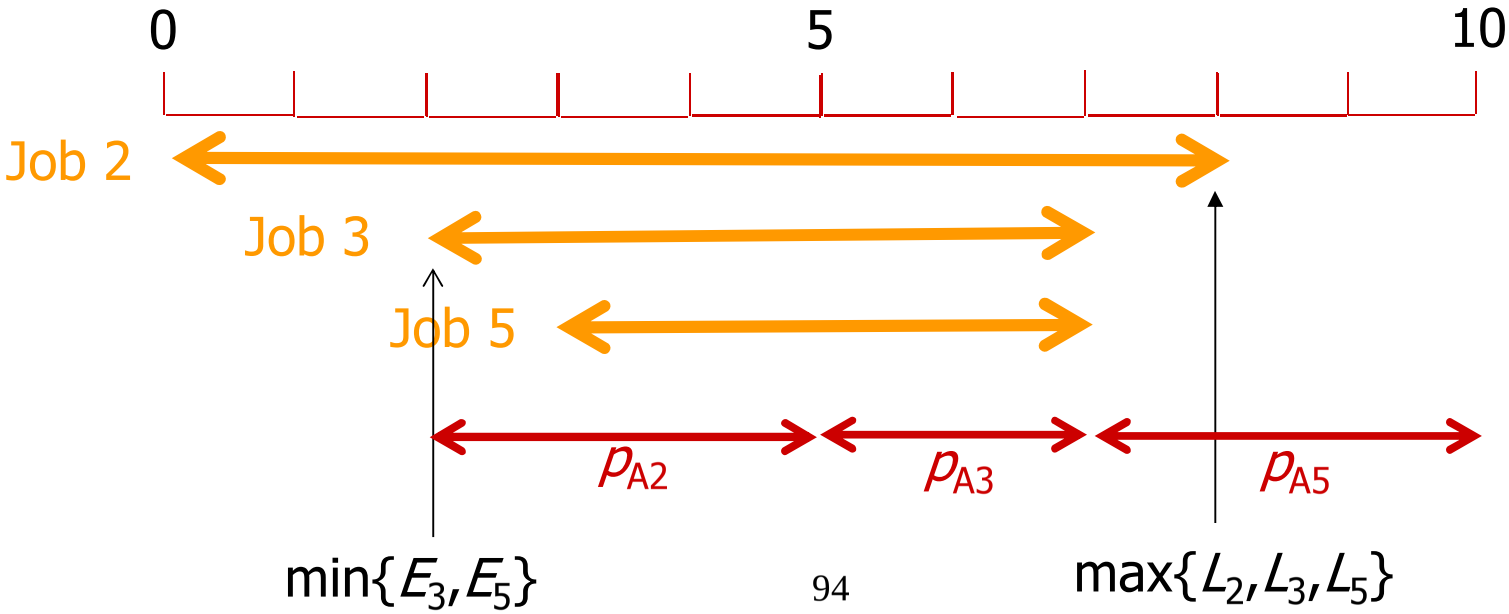


Edge Finding

- Jobs 2 precedes jobs 3, 5 because:

$$\max\{L_2, L_3, L_5\} - \min\{E_3, E_5\} < p_{A2} + p_{A3} + p_{A5}$$

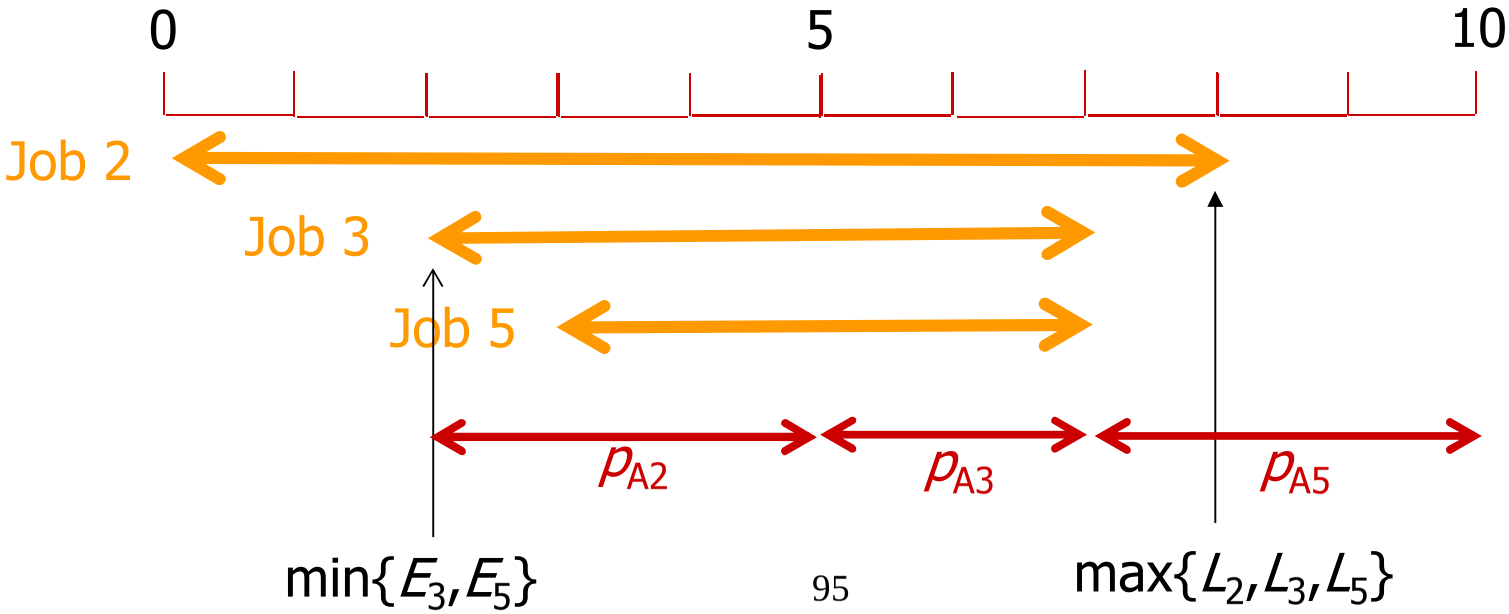
  - This is called edge finding because it finds an edge in the precedence graph for the jobs.



Edge Finding

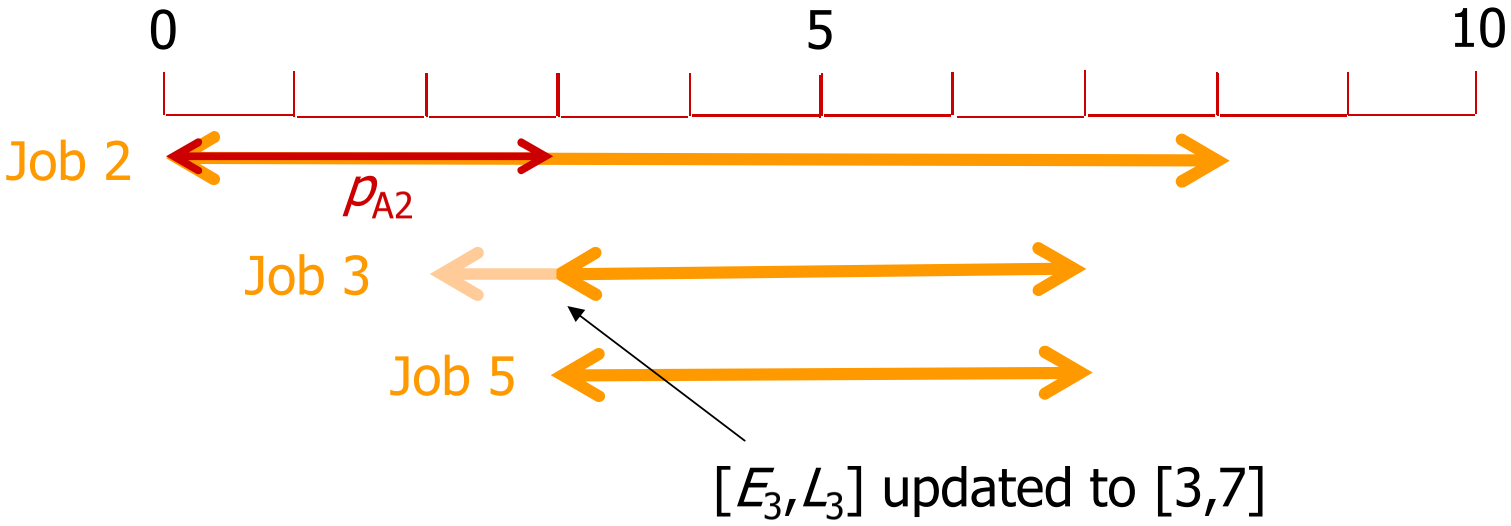
- In general, job *k* must precede set *S* of jobs on machine *i* when

$$\max_{j \in S \cup \{k\}} \{L_j\} - \min_{j \in S} \{E_j\} < \sum_{j \in S \cup \{k\}} p_{ij}$$



Edge Finding

- Since job 2 must precede 3 and 4, the earliest start times of jobs 3 and 4 can be updated.
  - They cannot start until job 2 is finished.

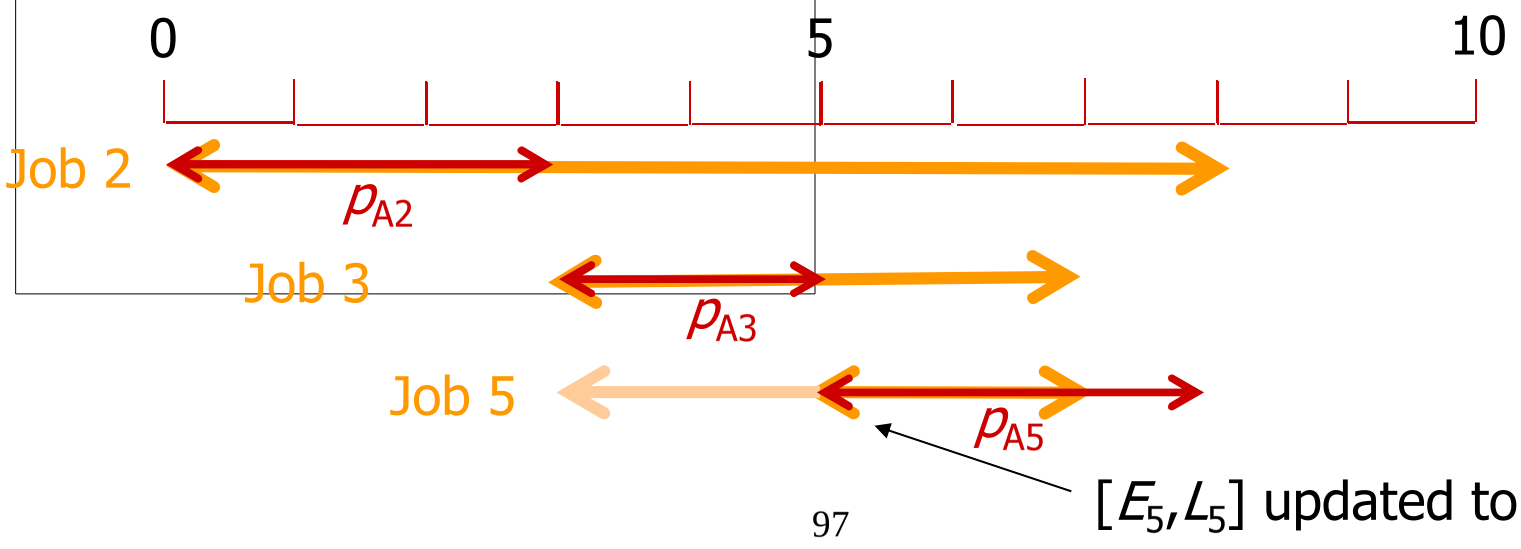


Edge Finding

- Edge finding also determines that job 3 must precede job 5, since

$$\max\{L_3, L_5\} - \min\{E_5\} = \max\{7,7\} - \min\{3\} = 4 < 5 = 2 + 3 = p_{A3} + p_{A5}$$

- This updates $[E_5, L_5]$ to [5,7], and there is too little time to run job 5. The schedule is **infeasible.**



Edge finding

- Both edge finding, and the resulting updates, require enumeration of exponentially many subsets of jobs…
  - …if done naively.
  - Both can be done in $n \log n$ time (n = number of jobs) if one is clever.
  - Start with Jackson pre-emptive schedule.

Benders Decomposition

- We will solve the 2-machine scheduling problem with **logic-based Benders decomposition**.
- First solve an assignment problem that allocates jobs to machines (**master problem**).
- Given this allocation, schedule the jobs assigned to each machine (**subproblem**).
- If a machine has no feasible schedule:
  - Determine which jobs cause the infeasibility.
  - Generate a **nogood** (**Benders cut**) that excludes assigning these jobs to that machine again
  - Add the Benders cut to the master problem and re-solve it.
- Repeat until the subproblem is feasible.

Benders Decomposition

- The **master problem** (assigns jobs to machines) is:

$$\min \sum_j f_{y_j j}$$
$$\text{Benders cuts}$$

- We will solve it as an **integer programming problem**.
  - Let binary variable $x_{ij} = 1$ when $y_j = i$.

$$\min \sum_{ij} f_{ij} x_{ij}$$
$$\text{Benders cuts}$$
$$x_{ij} \in \{0,1\}$$

Benders Decomposition

- We will add a **relaxation of the subproblem** to the master problem, to speed up solution.

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

$$p_{A3}x_{A3} + p_{A4}x_{A4} + p_{A5}x_{A5} \le 5$$

$$p_{A2}x_{A2} + p_{A3}x_{A3} + p_{A4}x_{A4} + p_{A5}x_{A5} \le 8$$

$$p_{A1}x_{A1} + p_{A2}x_{A2} + p_{A3}x_{A3} + p_{A4}x_{A4} + p_{A5}x_{A5} \le 10$$

$$p_{A4}x_{A5} \le 3$$

$$p_{A3}x_{A3} + p_{A4}x_{A4} + p_{A5}x_{A5} \le 5$$

$$p_{B2}x_{B2} + p_{B3}x_{B3} + p_{B4}x_{B4} + p_{B5}x_{B5} \le 8$$

If jobs 3,4,5 are all assigned to machine A, their processing time on that machine must fit between their earliest release time (2) and latest deadline (7).

Benders Decomposition

- The solution of the master problem is

$$x_{A1} = x_{A2} = x_{A3} = x_{A5} = x_{B4} = 1$$
$$\text{other } x_{ij}\text{s} = 0$$

- Assign jobs 1,2,3,5 to machine A, job 4 to machine B.
- The subproblem is to schedule these jobs on the 2 machines.
  - Machine B is trivial to schedule (only 1 job).
  - We found that jobs 2, 3, 5 cannot be scheduled on machine A.

Benders Decomposition

- So we create a **Benders cut** (nogood) that excludes assigning jobs 2, 3, 5 to machine A.
  - These are the jobs that caused the infeasibility.
  - The Benders cut is:

$$(1-x_{A2})+(1-x_{A3})+(1-x_{A5})\geq 1$$

- Add this constraint to the master problem.
  - The solution of the master problem now is:

$$x_{A1}=x_{A2}=x_{A5}=x_{B3}=x_{B4}=1$$
$$\text{other } x_{ij}\text{s}=0$$

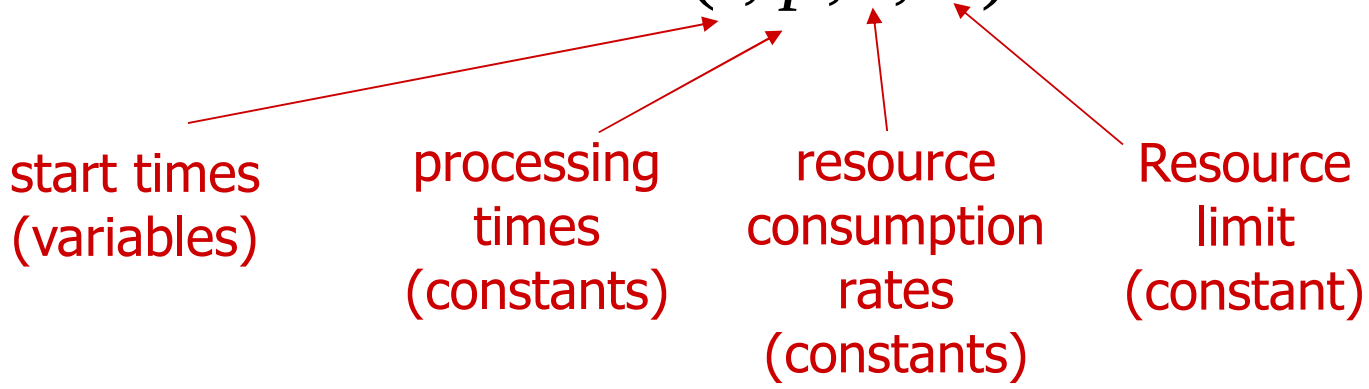
Benders Decomposition

- So we schedule jobs 1, 2, 5 on machine A, jobs 3, 4 on machine B.
  - These schedules are feasible.
  - The resulting solution is optimal, with min cost 130.

Cumulative Scheduling

- Jobs can run simultaneously provided total resource consumption rate never exceeds a limit. For this we use the constraint

$$\text{cumulative}(t, p, c, C)$$



- There are several filtering methods.

Cumulative Scheduling

- Filtering methods:
  - Time tabling.
  - Edge finding.
  - Extended edge finding.
  - Not-first/not-last.
  - Energetic reasoning.
- All can be done in polynomial time…
  - Using clever algorithms, data structures.