Lessons from MIP Search

John Hooker
Carnegie Mellon University

November 2009

Outline

- MIP search
  - The main ideas
- Duality and nogoods
  - From MIP to AI (and back)
- Binary decision diagrams
  - From MIP to constraint programming

Background reading – MIP search

General Survey

J. T. Linderoth and M. W. P. Savelsbergh, A computational study of search strategies for mixed integer programming, *INFORMS Journal on Computing* 11 (1999) 173-187.

Survey of Cutting Planes

H. Marchand, A. Martin, R. Weismantel, and L. Wolsey. Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics*, 123 (2002) 397–446.

Some Recent Developments

M. Fischetti, F. Glover, A. Lodi, The feasibility pump, *Mathematical Programming* 104 (2005) 91-104.

M. Fischetti and A. Lodi, Repairing MIP infeasibility through local branching, *Computers and Operations Research* 35 (2008) 1436-1445,

T. Sandholm, R. Shields, Nogood learning for mixed integer programming, CMU, November 2006

Background reading – BDDs

Basic Ideas

H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann, A constraint store based on multivalued decision diagrams, in C. Bessiere, ed., *CP 2007,* 118-132.

Refinements

Tarik Hadzic and J. N. Hooker, Cost-bounded binary decision diagrams for 0-1 programming, in E. Loute and L. Wolsey, eds., CPAIOR 2007, 84-98.

T. Hadzic, J. N. Hooker, and P. Tiedemann, Approximate compilation of constraints into multivalued decision diagrams, in P. J. Stuckey, ed., *CP 2007,* 448-462.

MIP search

MIP and the LP relaxation

Branch and bound

Variable selection

Node selection

Feasibility heuristics

Mixed integer programming

- We focus on mixed integer/**linear** programming (MILP)

$$\min\ cx$$
$$Ax \geq b,\ x \geq 0$$
$$x_j \in \mathbb{Z},\ j \in J$$

LP relaxation

- An advantage of MIP: ready-made global relaxation
  - Linear programming (LP) problem

$$\min\ cx$$

$$Ax \geq b,\ x \geq 0$$

$$x_j \in \mathbb{R},\ \text{all } j$$

Drop integrality condition

LP relaxation

- An advantage of MIP: ready-made global relaxation
  - Linear programming (LP) problem

$$\min\ cx$$

$$Ax \geq b,\ x \geq 0$$

$$x_j \in \mathbb{R},\ \text{all } j$$

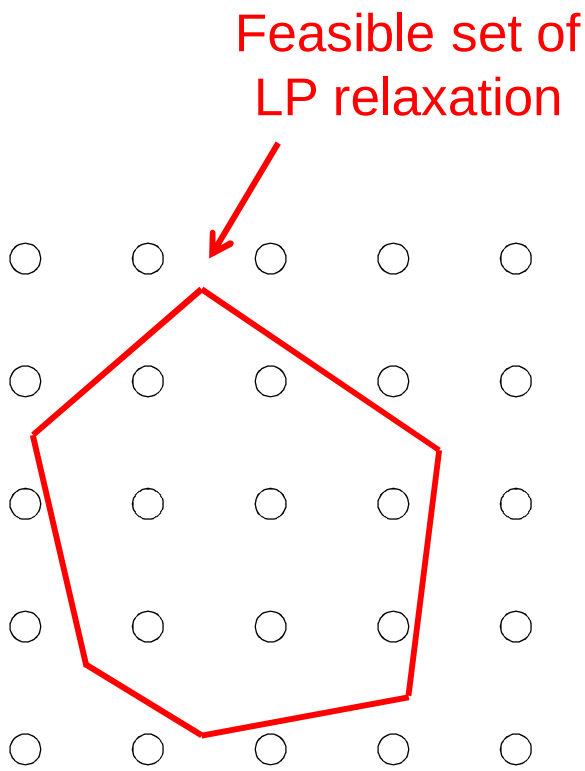
Drop integrality condition

Cutting planes can strengthen the relaxation

LP relaxation

- An advantage of MIP: ready-made global relaxation
  - Linear programming (LP) problem

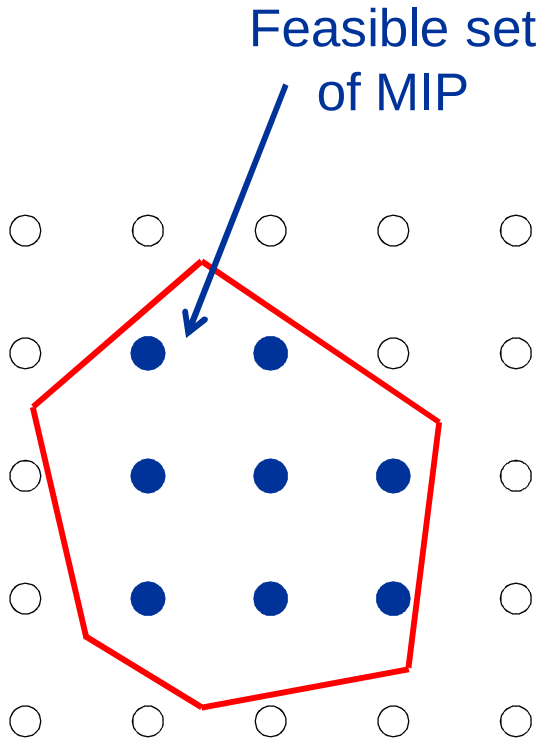
$$\min\ cx$$
$$Ax \geq b,\ x \geq 0$$
$$x_j \in \mathbb{R},\ \text{all } j$$



LP relaxation

- An advantage of MIP: ready-made global relaxation
  - Linear programming (LP) problem

$$\min\ cx$$
$$Ax \geq b,\ x \geq 0$$
$$x_j \in \mathbb{R},\ \text{all } j$$



LP relaxation

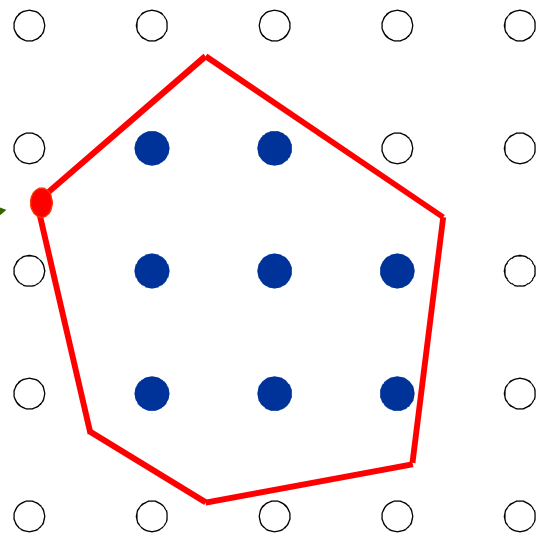
- An advantage of MIP: ready-made global relaxation
  - Linear programming (LP) problem

$$\min\ cx$$

$$Ax \geq b,\ x \geq 0$$

$$x_j \in \mathbb{R},\ \text{all } j$$

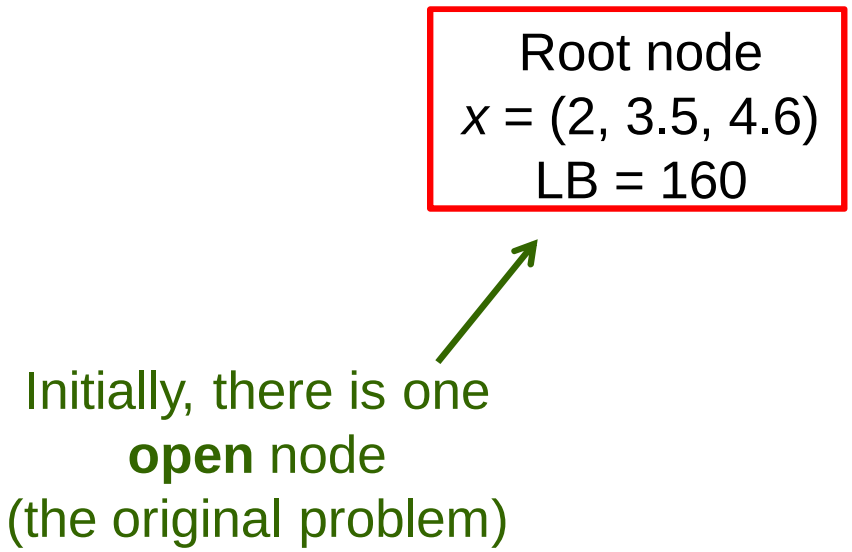
Simplex method finds a vertex solution of the LP relaxation



Branch and bound

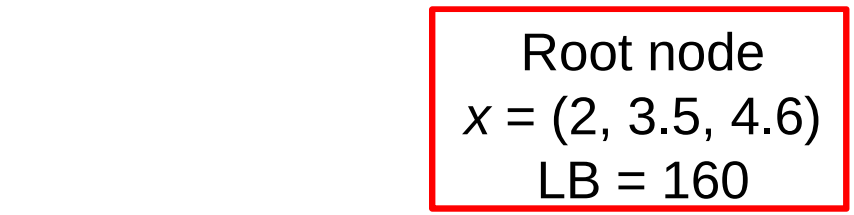
- Used for ≥ 50 years
  - Land and Doig 1960.
- Creates search tree by branching on variables
  - Solves continuous relaxation (LP) at each node
- Basic search framework for all general-purpose MIP codes
  - **Branch and cut** adds cutting planes at some nodes.

Branch and bound - illustration



Remember:
we are minimizing

Branch and bound - illustration



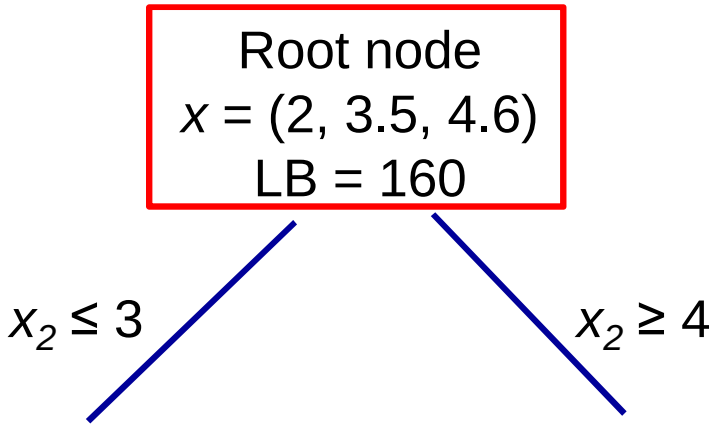
Remember:
we are minimizing

Initially, there is one **open** node
(the original problem)

Optimal value of LP relaxation is lower bound on optimal value of original problem

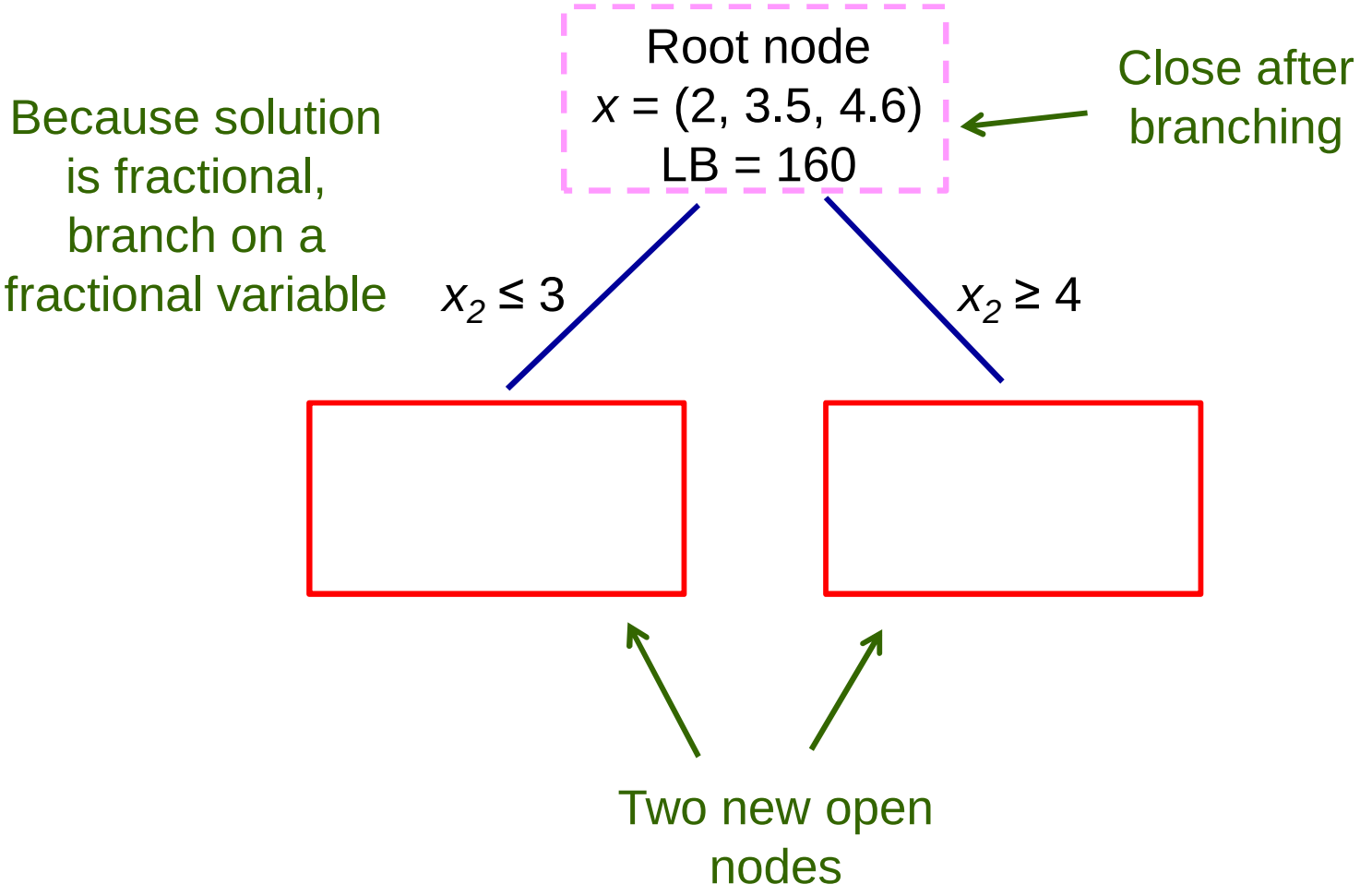
Branch and bound - illustration

Because solution is fractional, branch on a fractional variable

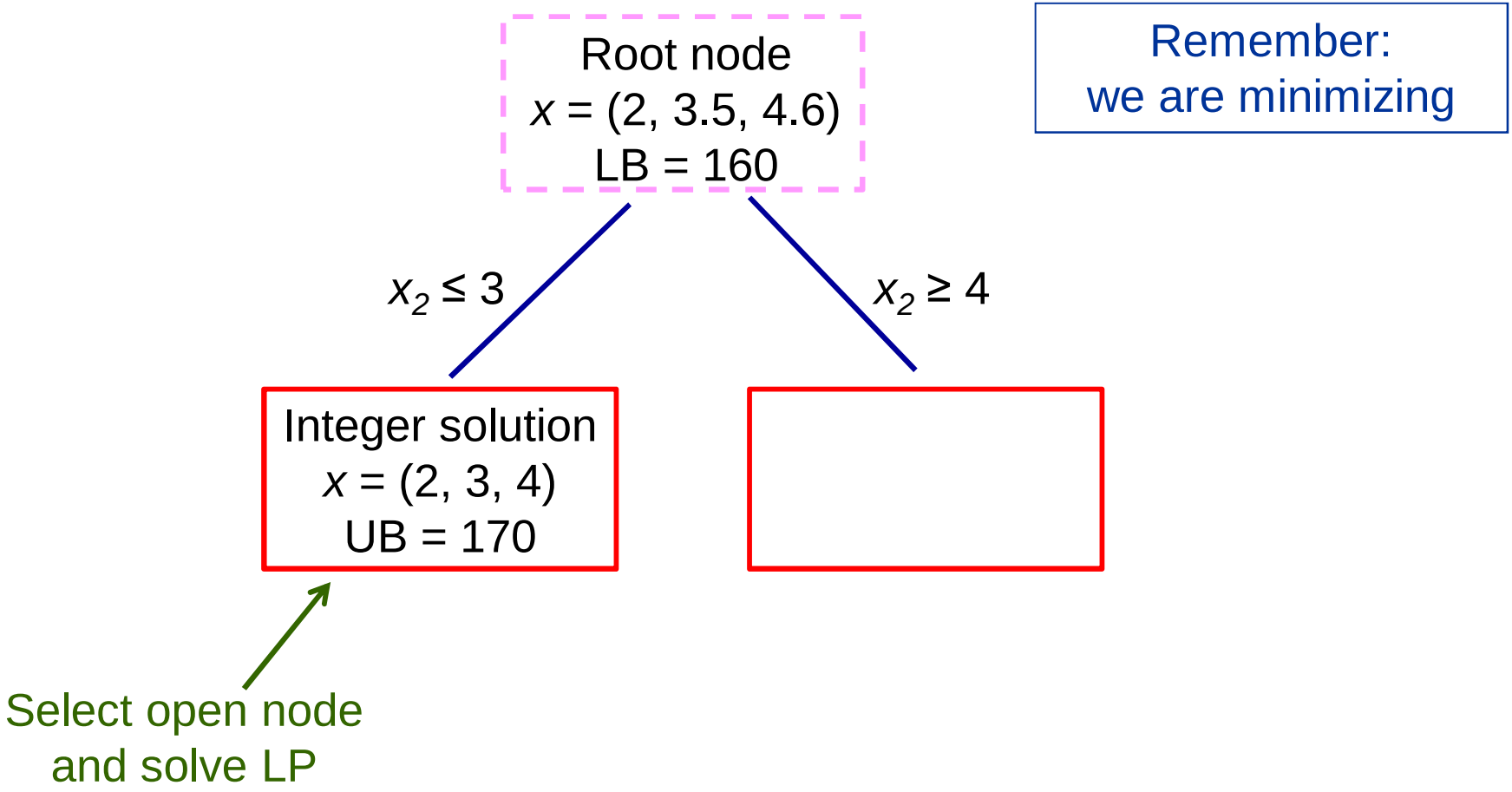


Remember:
we are minimizing

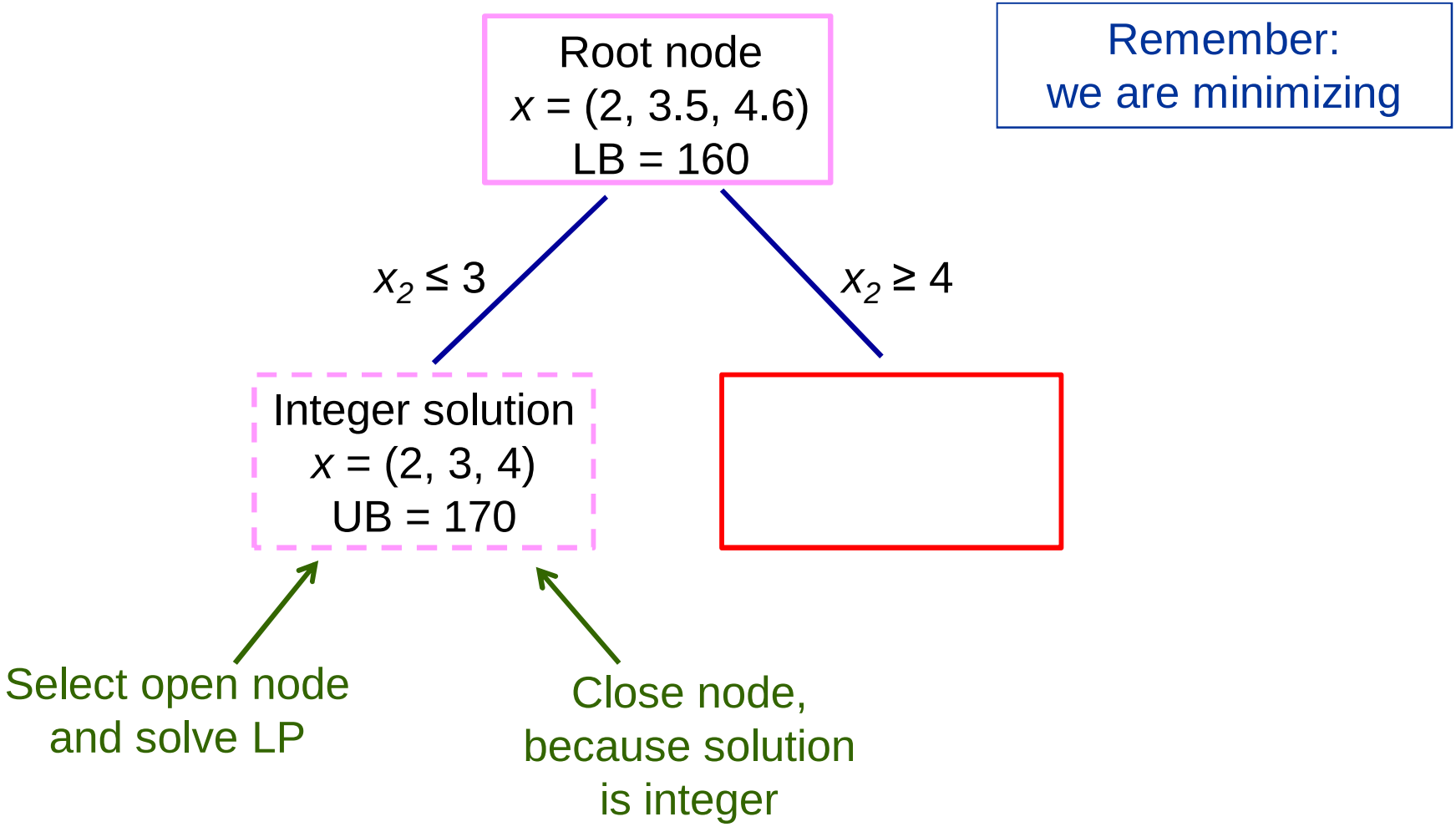
Branch and bound - illustration



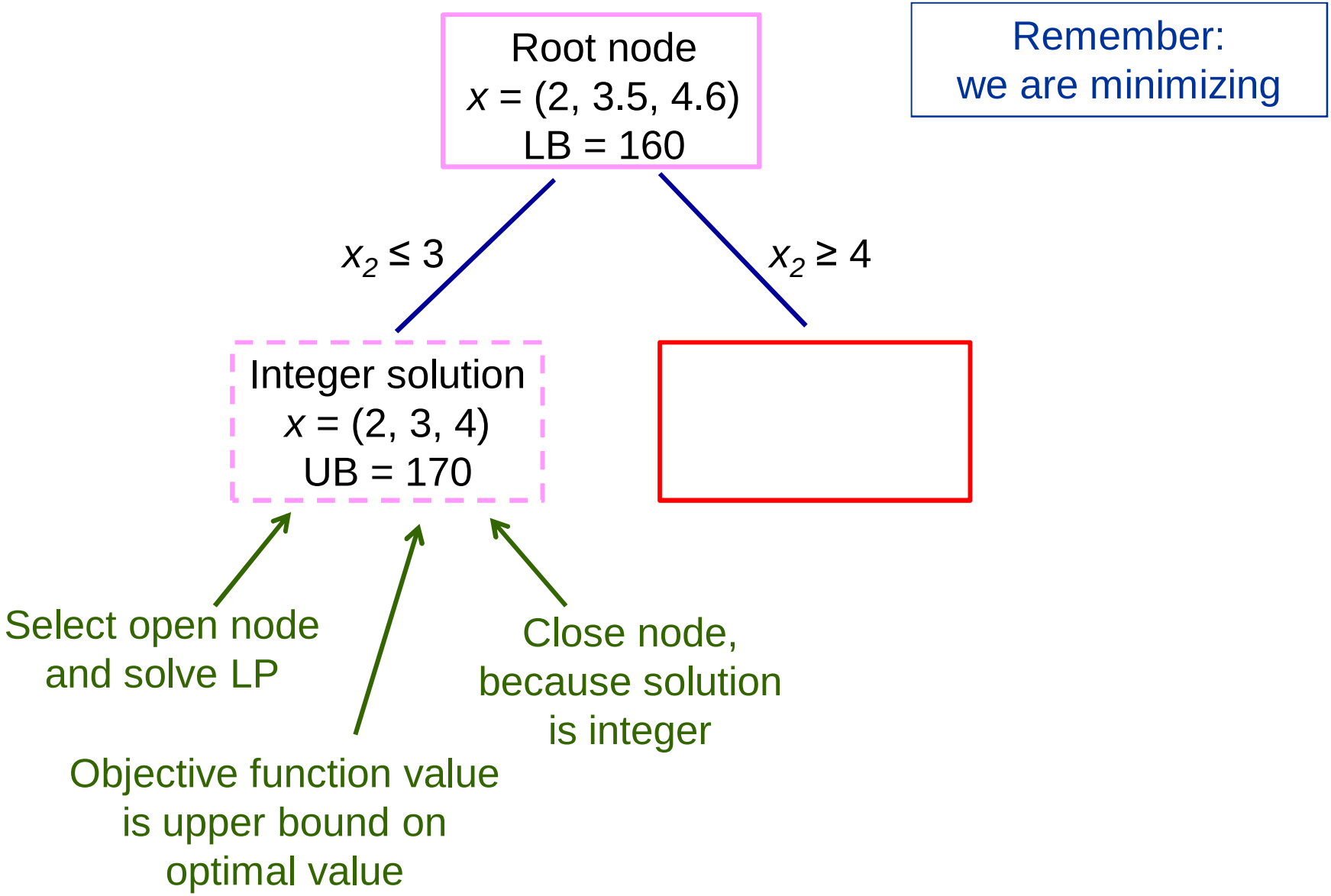
Branch and bound - illustration



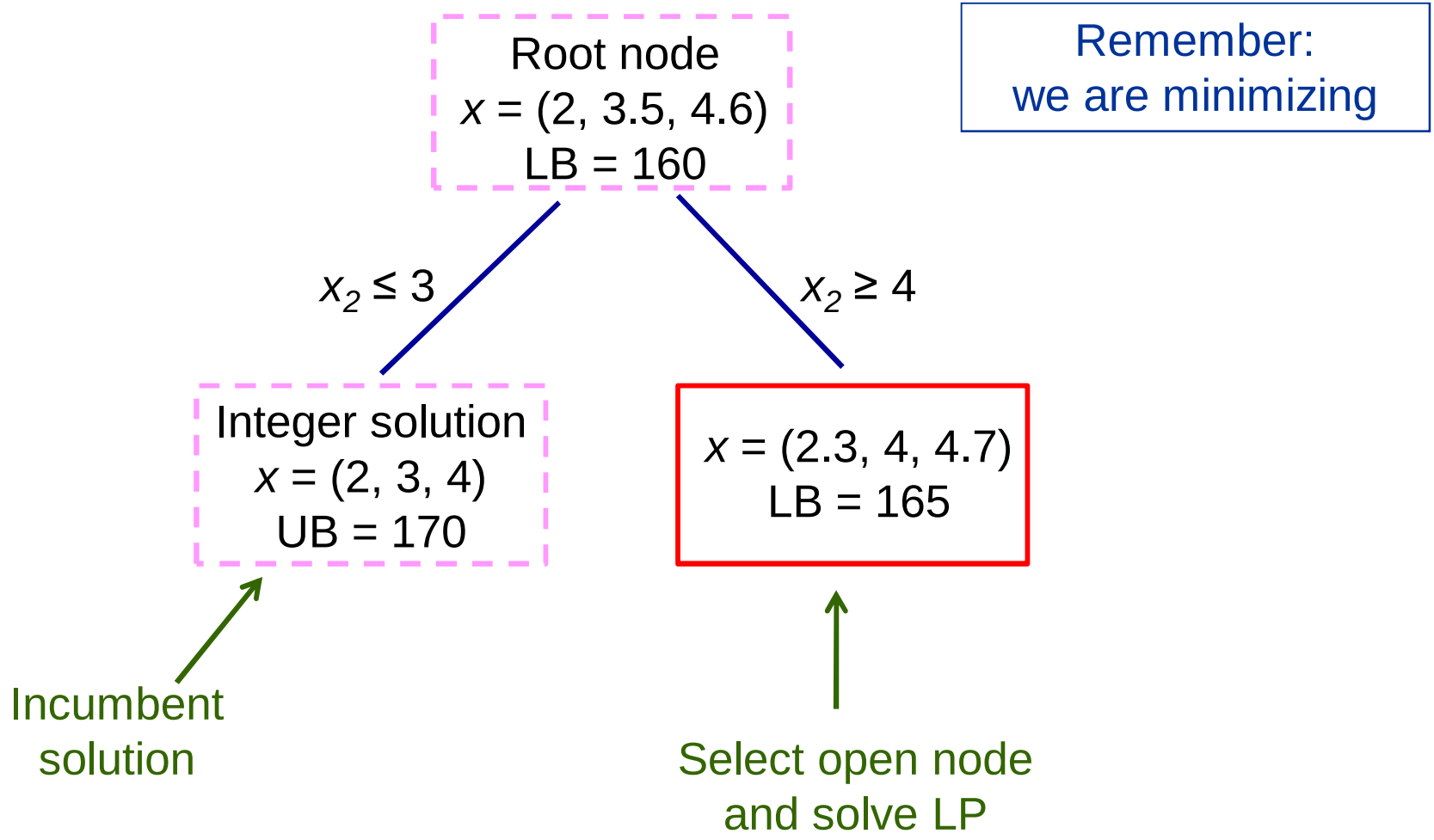
Branch and bound - illustration



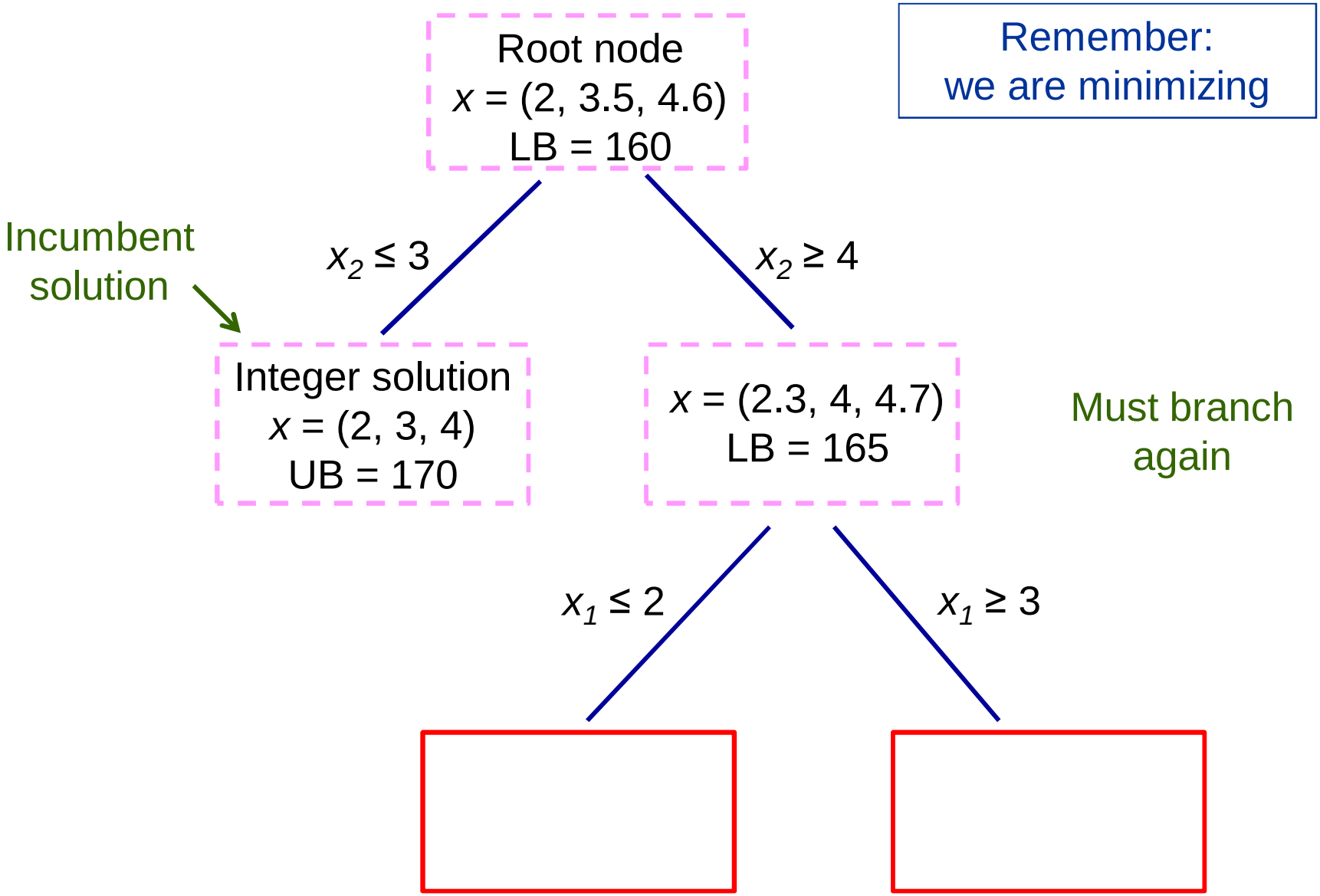
Branch and bound - illustration



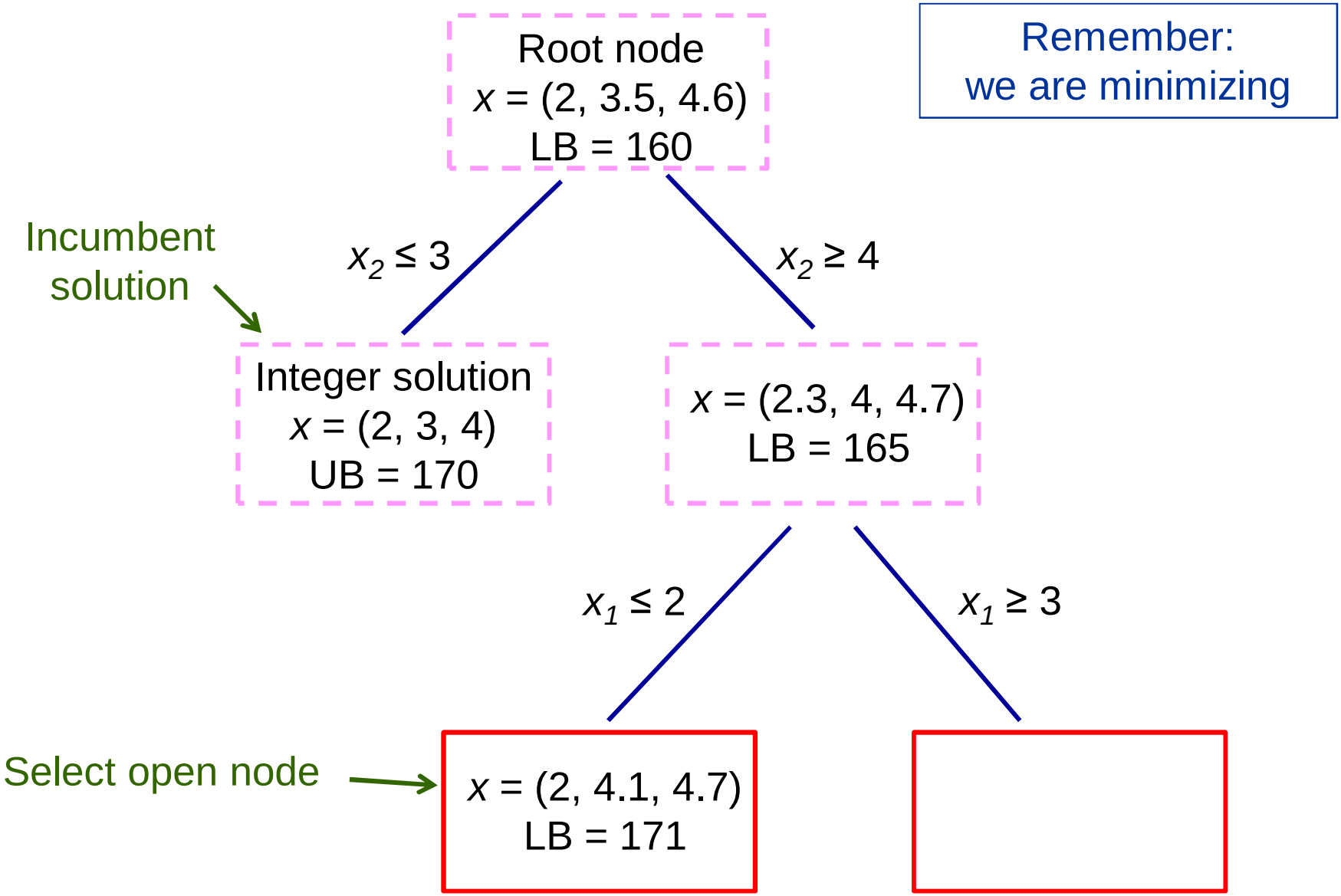
Branch and bound - illustration



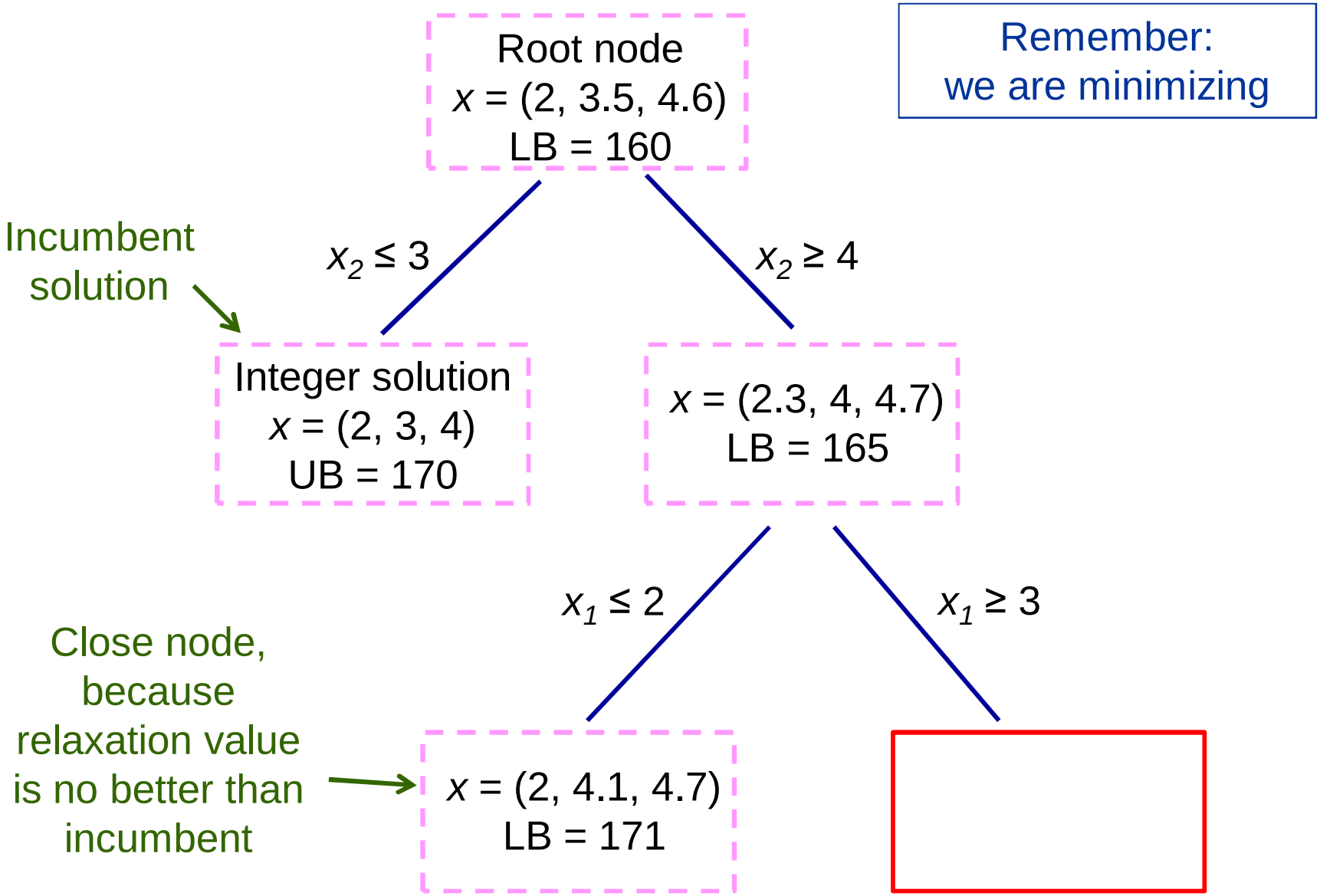
Branch and bound - illustration



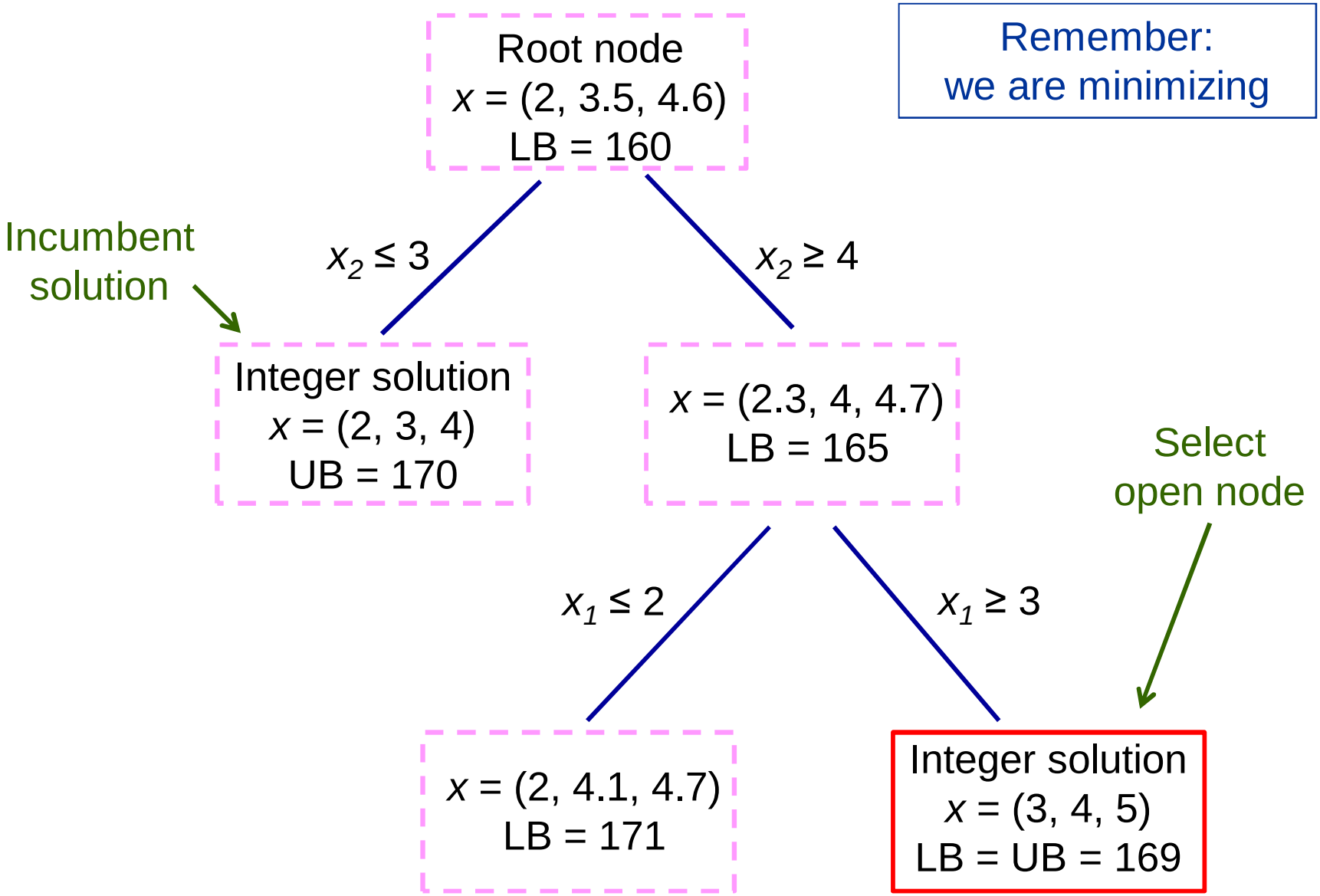
Branch and bound - illustration



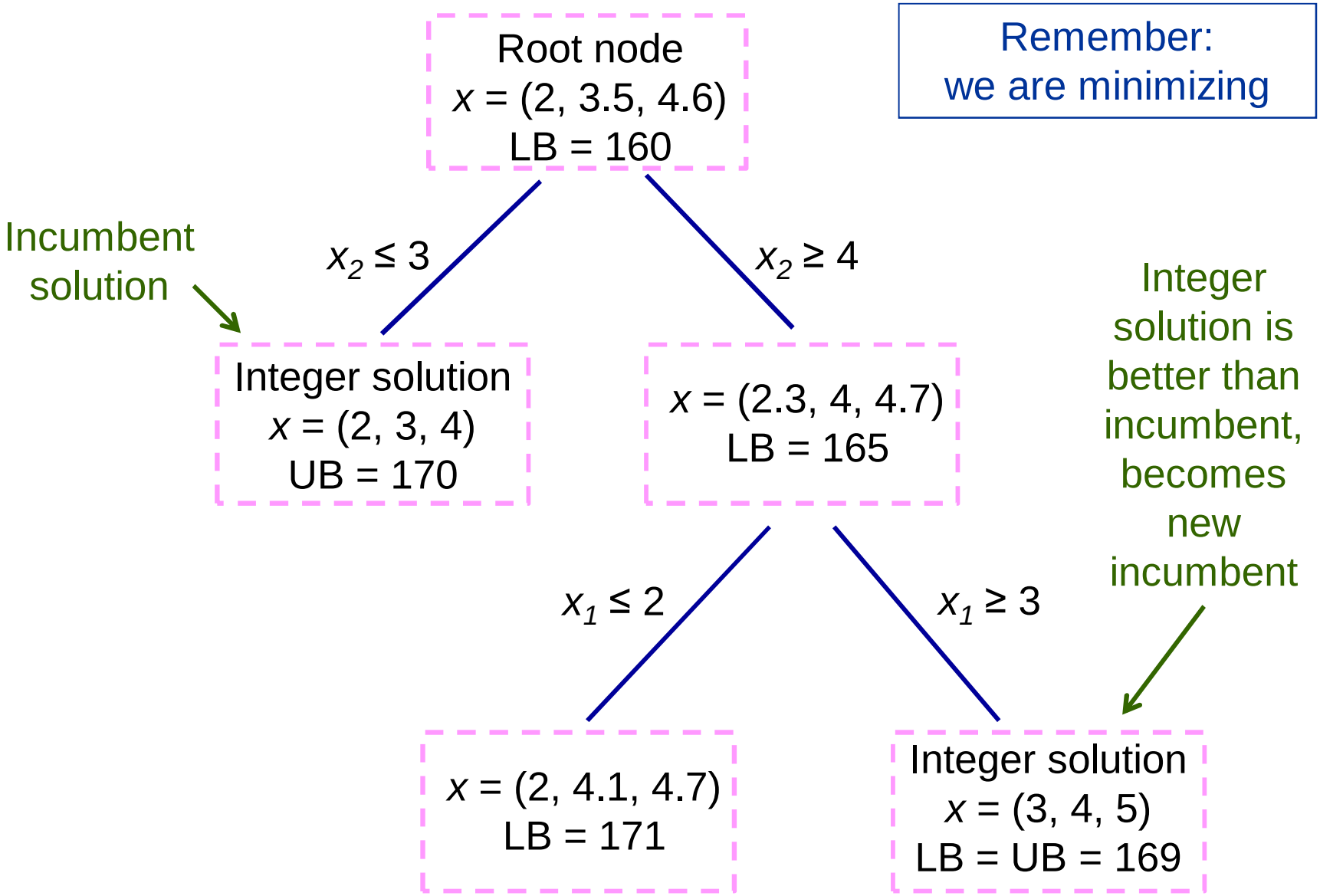
Branch and bound - illustration



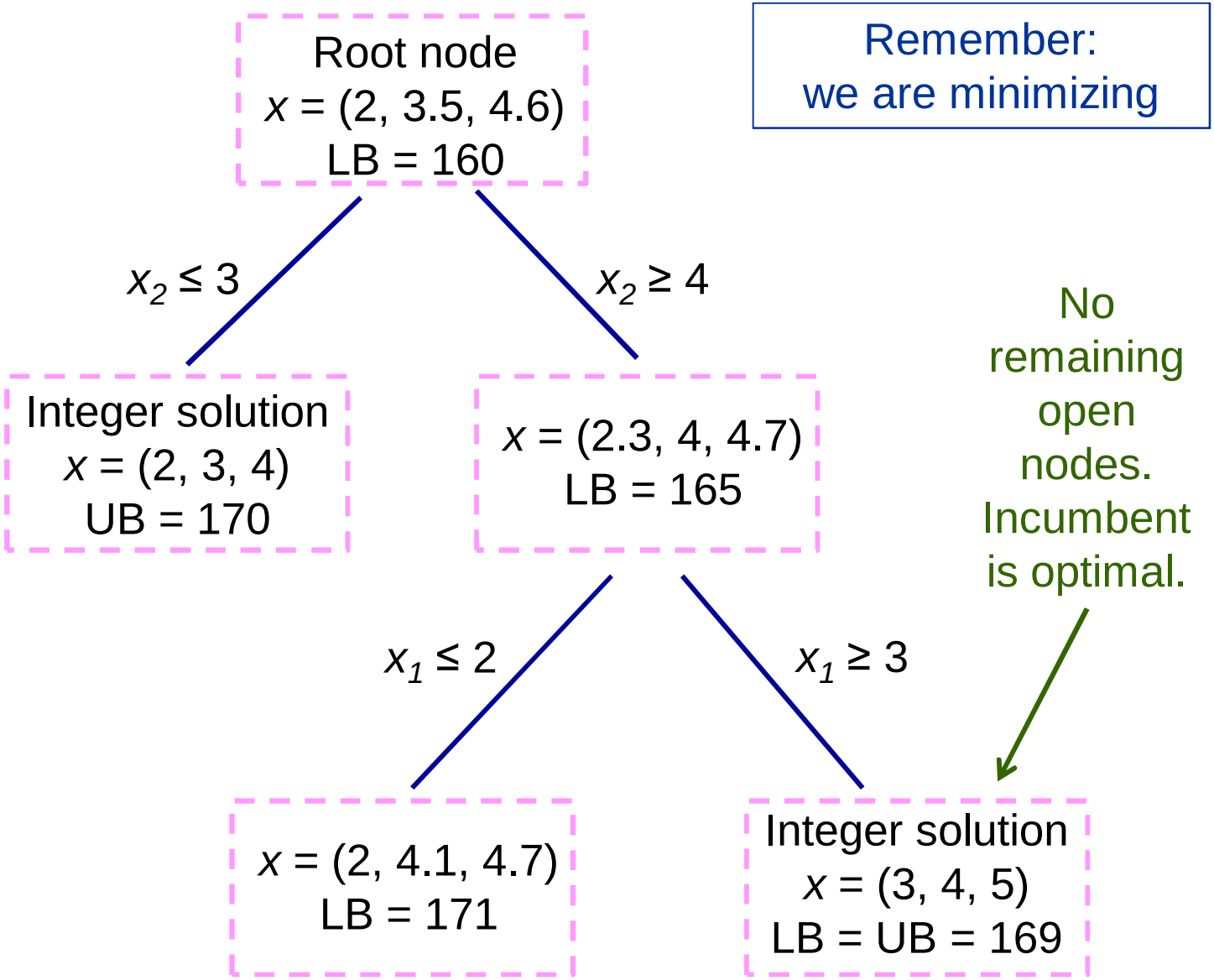
Branch and bound - illustration



Branch and bound - illustration



Branch and bound - illustration



Branch and bound procedure

Let original problem be an open node

Let UB = ∞

While open nodes remain

Select an open node *N* and solve its continuous relaxation *R*

Let LB = value of *R* and close *N*

If solution of *R* is integer **then**

If LB < UB **then** let UB = LB

Else if LB < UB **then**

Branch on fractional variable and create 2 new open nodes

Why it works

- Luck
  - Most variables tend to be integer in solution of relaxation.

Why it works

- Luck
  - Most variables tend to be integer in solution of relaxation.
- Global continuous relaxation
  - Cutting planes, good formulations

Why it works

- Luck
  - Most variables tend to be integer in solution of relaxation.
- Global continuous relaxation
  - Cutting planes, good formulations
- Presolve
  - Bag of tricks to simplify problem

Why it works

- Luck
  - Most variables tend to be integer in solution of relaxation.
- Global continuous relaxation
  - Cutting planes, good formulations
- Presolve
  - Bag of tricks to simplify problem
- Feasibility heuristics
  - Find feasible solutions before branching

Why it works

- Luck
  - Most variables tend to be integer in solution of relaxation.
- Global continuous relaxation
  - Cutting planes, good formulations
- Presolve
  - Bag of tricks to simplify problem
- Feasibility heuristics
  - Find feasible solutions before branching
- Good engineering
  - 50 years of experience

Basic issues

- Variable selection
  - Branch on which variable at current node?
- Node selection
  - Process which node next?
- Feasibility heuristics
  - How to find integer solutions early?
  - …to get good upper bounds

Variable selection

- Goal is to tighten lower bound after branching
  - We don’t know how to increase probability of finding integer solutions.
- Pseudocosts
  - Estimate change in objective function after branching
- Strong branching
  - Put bound on change in objective function after branching

Pseudocosts

- Pseudocost = rate of increase of LP lower bound when fractional variable x_j is rounded up or down

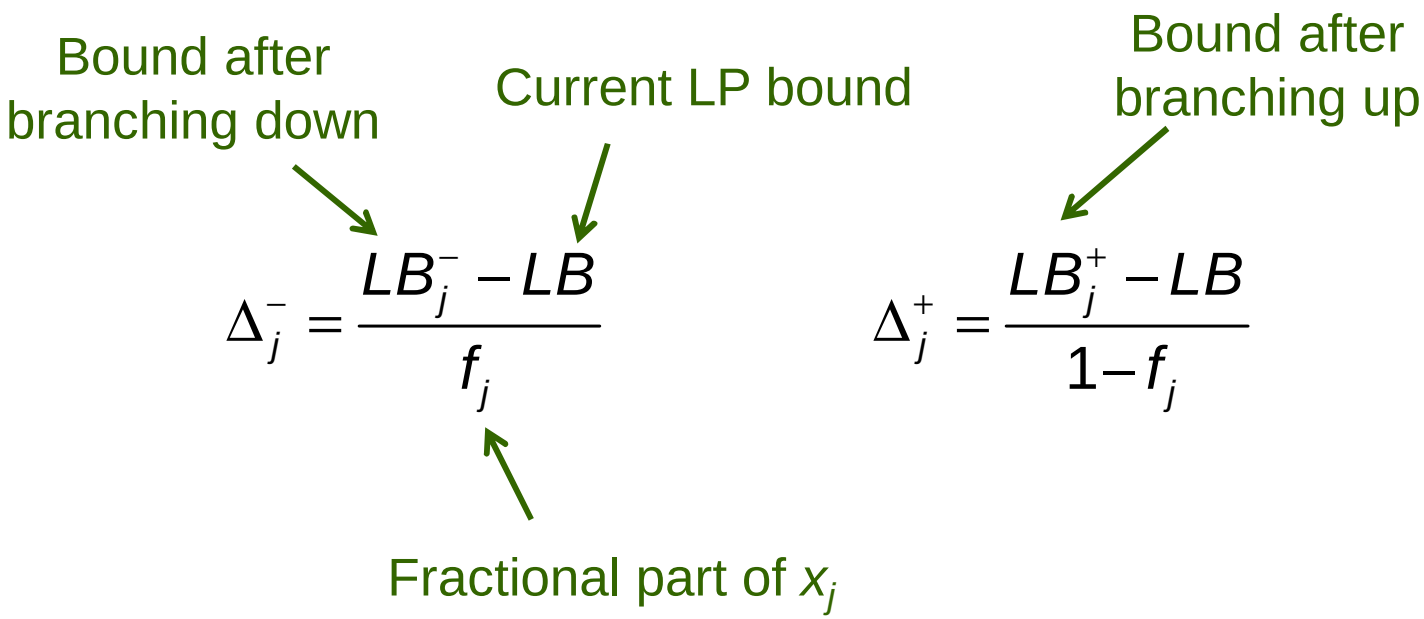
Current LP bound

$$\Delta_j^- = \frac{LB_j^- - LB}{f_j} \qquad \Delta_j^+ = \frac{LB_j^+ - LB}{1 - f_j}$$

Fractional part of x_j

Pseudocosts

- Pseudocost = rate of increase of LP lower bound when fractional variable x_j is rounded up or down



Pseudocosts

- Using pseudocosts
  - One method: branch on variable x_j with largest $\Delta_j^- f_j + \Delta_j^+ (1 - f_j)$

$$\Delta_j^- = \frac{LB_j^- - LB}{f_j} \qquad \Delta_j^+ = \frac{LB_j^+ - LB}{1 - f_j}$$

Pseudocosts

- Computing pseudocosts at root node
  - One method: Explicitly compute LB_j^-, LB_j^+ for fractional variables by solving 2 LPs.
    - Perhaps with limited number of simplex iterations.

$$\Delta_j^- = \frac{LB_j^- - LB}{f_j} \qquad \Delta_j^+ = \frac{LB_j^+ - LB}{1 - f_j}$$

Pseudocosts

- Computing pseudocosts lower in the tree
  - For variables not yet branched on or evaluated:
    - Explicitly compute LB_j^-, LB_j^+
  - For variables previously branched on:
    - Use average of Δ_j^-, Δ_j^+ values obtained after previous branches.

$$\Delta_j^- = \frac{LB_j^- - LB}{f_j} \qquad \Delta_j^+ = \frac{LB_j^+ - LB}{1 - f_j}$$

Strong branching

- We need the concept of LP duality
  - Will explain origin of dual later

$$\max\ ub \qquad\qquad \min\ cx$$

$$uA \le c,\ u \ge 0 \qquad\qquad Ax \ge b,\ x \ge 0$$

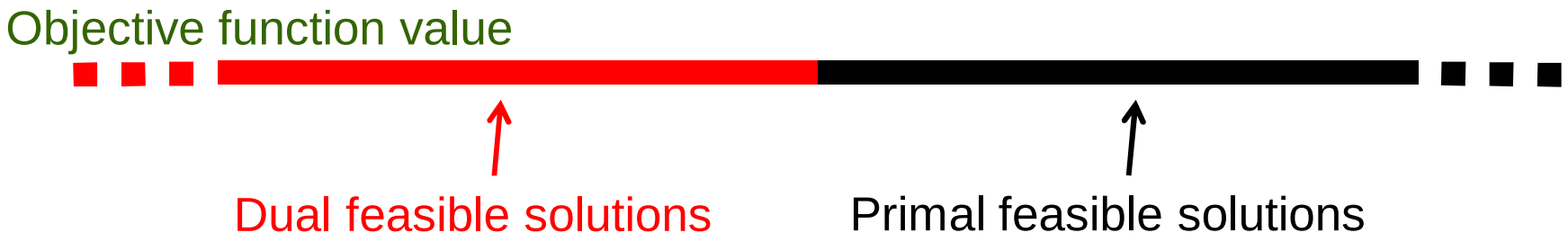
$$u_i \in \mathbb{R},\ \text{all } i \qquad\qquad x_j \in \mathbb{R},\ \text{all } j$$

Dual problem

Primal problem

LP dual

Dual	Primal
$\max\ ub$	$\min\ cx$
$uA \le c, u \ge 0$	$Ax \ge b, x \ge 0$
$u_i \in \mathbb{R},\ \text{all } i$	$x_j \in \mathbb{R},\ \text{all } j$



"Weak duality"

LP dual

Dual	Primal
$\max\ ub$	$\min\ cx$
$uA \le c, u \ge 0$	$Ax \ge b, x \ge 0$
$u_i \in \mathbb{R},\ \text{all } i$	$x_j \in \mathbb{R},\ \text{all } j$

Objective function value

When primal and dual feasible solutions have the same value, both are **optimal**

"Strong duality"

LP dual

Dual	Primal
$\max\ ub$	$\min\ cx$
$uA \le c, u \ge 0$	$Ax \ge b, x \ge 0$
$u_i \in \mathbb{R},\ \text{all}\ i$	$x_j \in \mathbb{R},\ \text{all}\ j$

Objective function value

Simplex method maintains primal feasibility and strives for dual feasibility

LP dual

Dual	Primal
$\max\ ub$	$\min\ cx$
$uA \le c, u \ge 0$	$Ax \ge b, x \ge 0$
$u_i \in \mathbb{R},\ \text{all } i$	$x_j \in \mathbb{R},\ \text{all } j$

Objective function value

Dual simplex method maintains dual feasibility and strives for primal feasibility

Simplex method maintains primal feasibility and strives for dual feasibility

Dual simplex

- Dual simplex is used to reoptimize after adding a constraint
  - …such as new bound on variable after branching.
- Solution remains dual feasible after adding constraint.
  - Because this adds a variable in the dual.
  - Dual simplex restores primal feasibility.
- Premature termination of dual simplex provides lower bound on optimal value.
  - Due to weak duality.

Strong branching

- For fractional variables x_j :
  - Compute lower bounds L_j^-, L_j^+ on increase in LP value after branching down or up.
  - Branch on x_j with largest $L_j^- + L_j^+$
- Obtain L_j^-, L_j^+ by running a few iterations of dual simplex.
  - Perhaps only one.
- To save time, evaluate only a few fractional variables.

Node selection

- Select which of the open nodes to process.
- Goal is to find good feasible solution, or prove that there is no solution better than incumbent.
  - In either case, it is reasonable to look at quality of LP bounds.
- Two common methods:
  - Depth first
  - Best bound

Depth first

- Advantages:
  - Easy to implement.
  - Fast reoptimization of LP.
  - Minimal memory requirements.

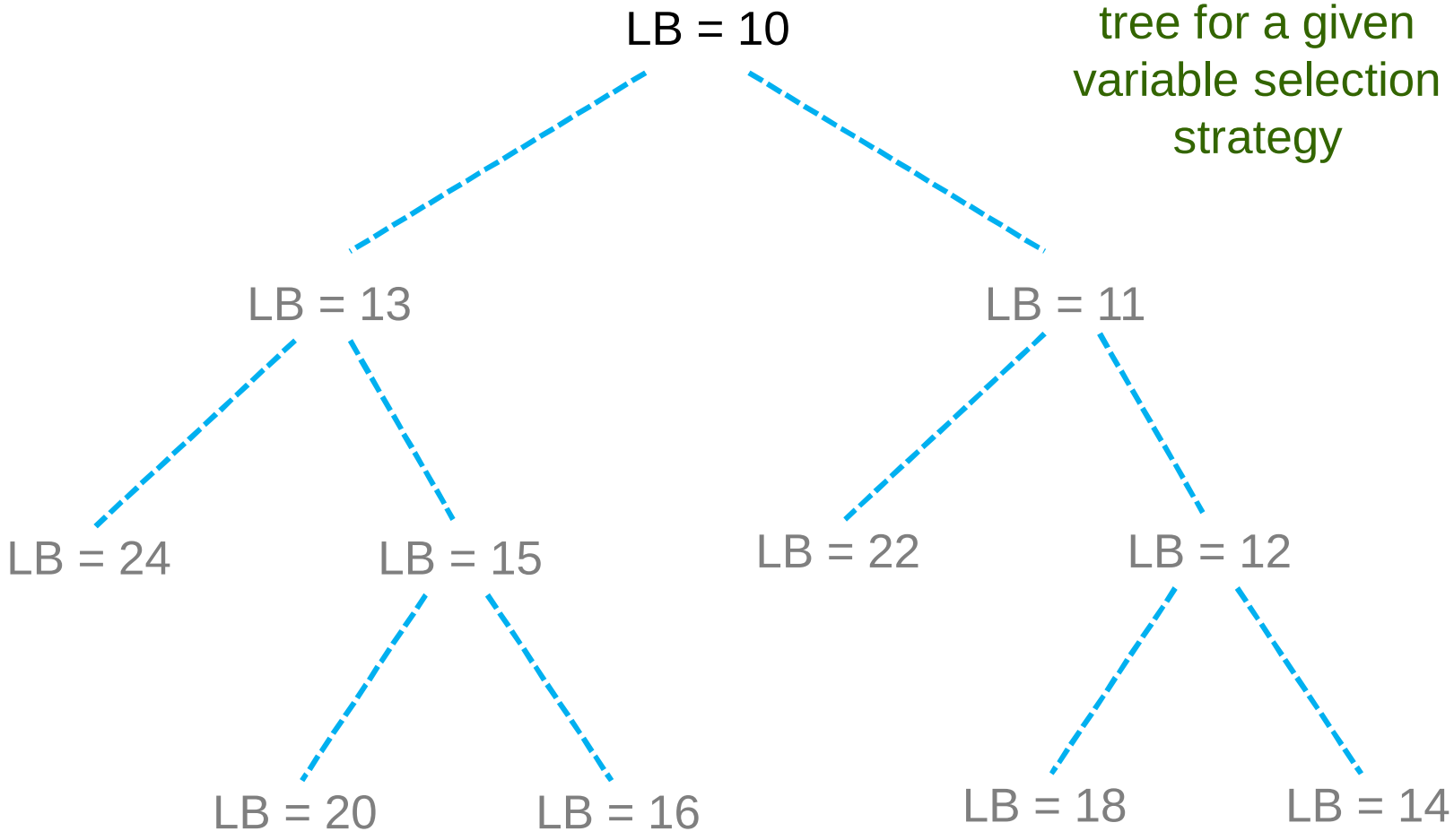
- Disadvantages:
  - Blind to what is going on.
  - Must search entire subtree before moving to a more promising subtree.
  - Tends to explode in hard problems.

Best bound

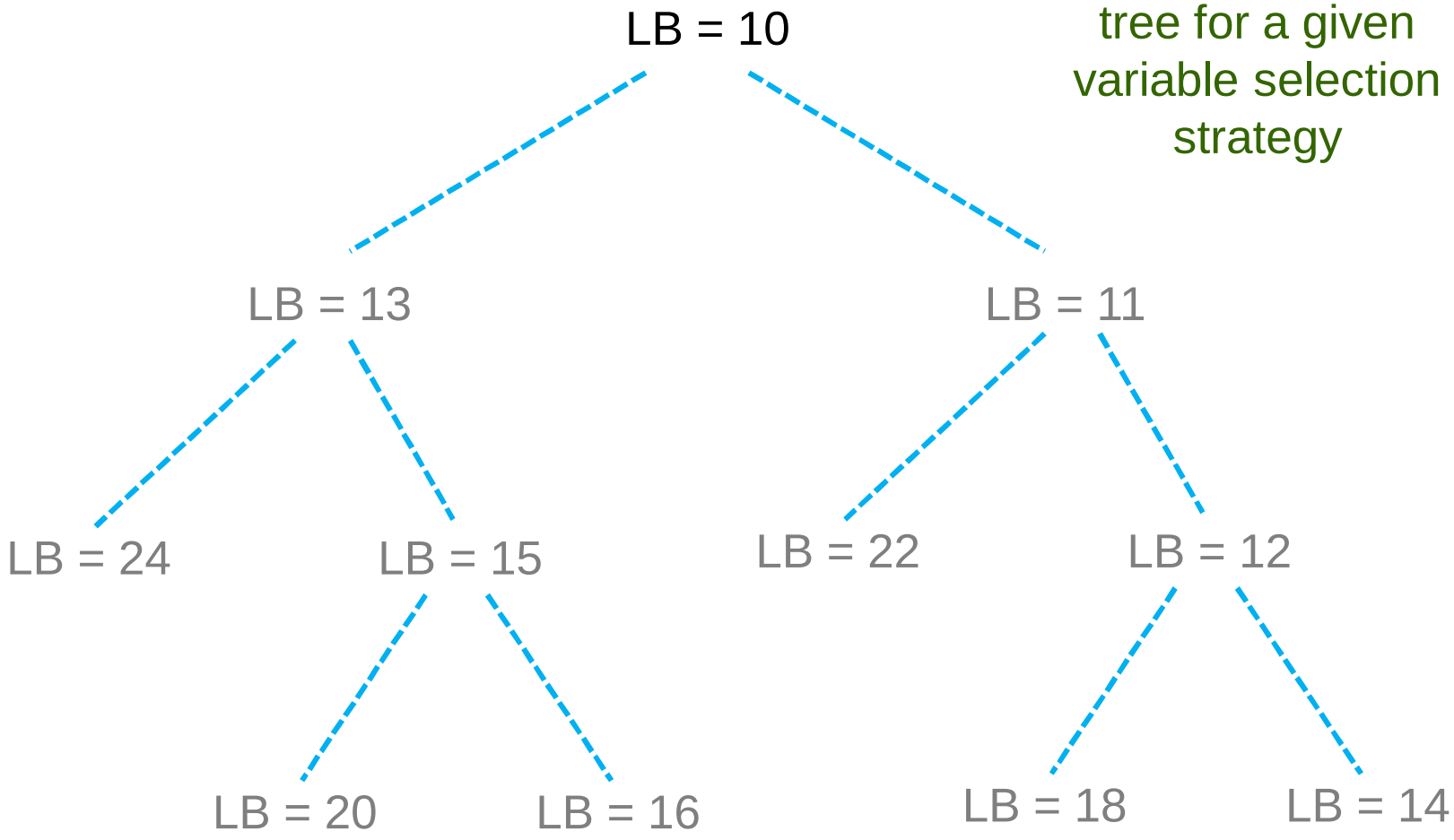
- Select open node with best LP value.
  - For a given branching rule, this minimizes number of nodes processed.

Best bound

Complete search tree for a given variable selection strategy



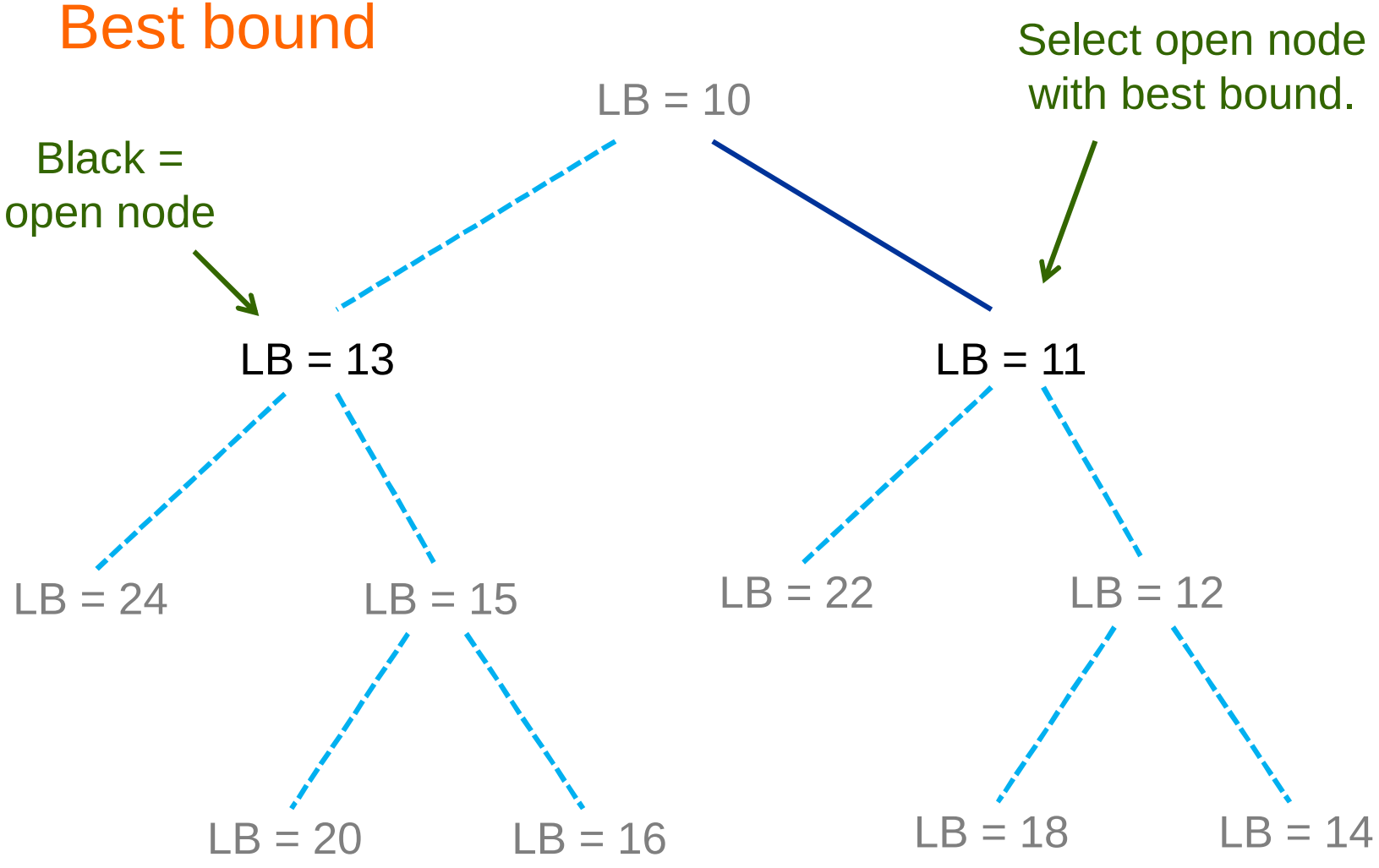
Best bound



Complete search tree for a given variable selection strategy

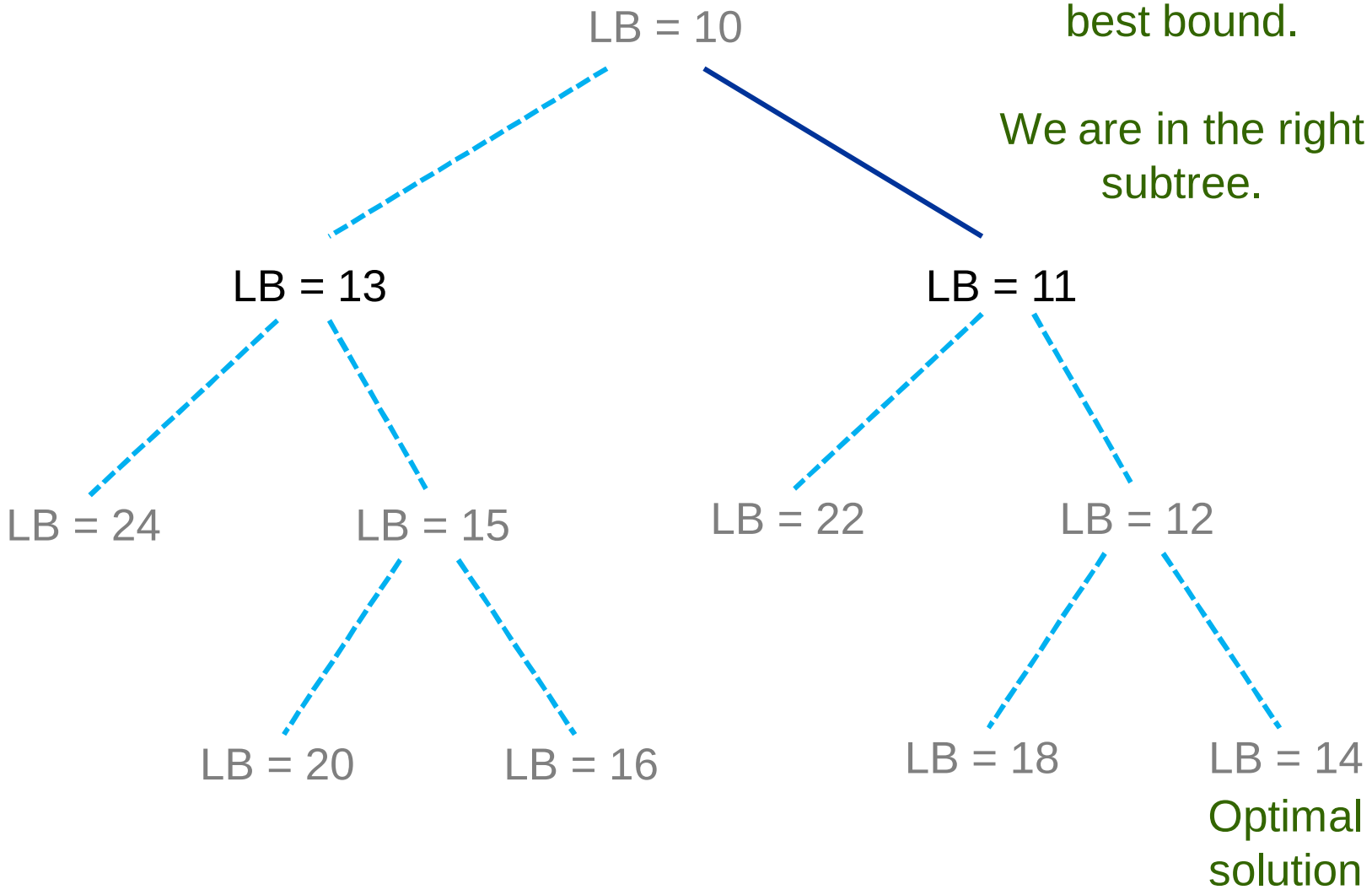
Leaf nodes represent integer solutions

Best bound



Leaf nodes represent integer solutions

Best bound

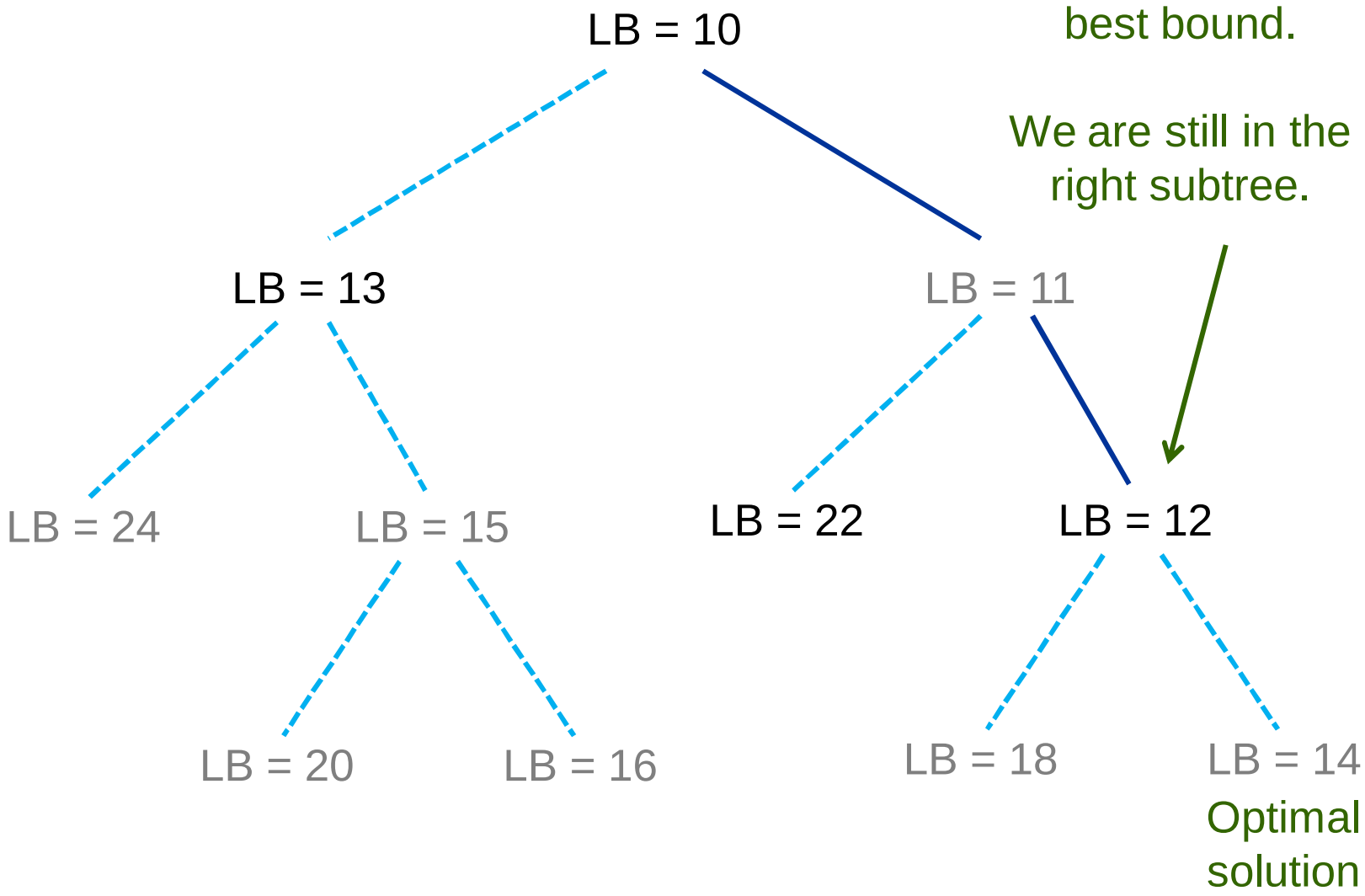


Select node with best bound.

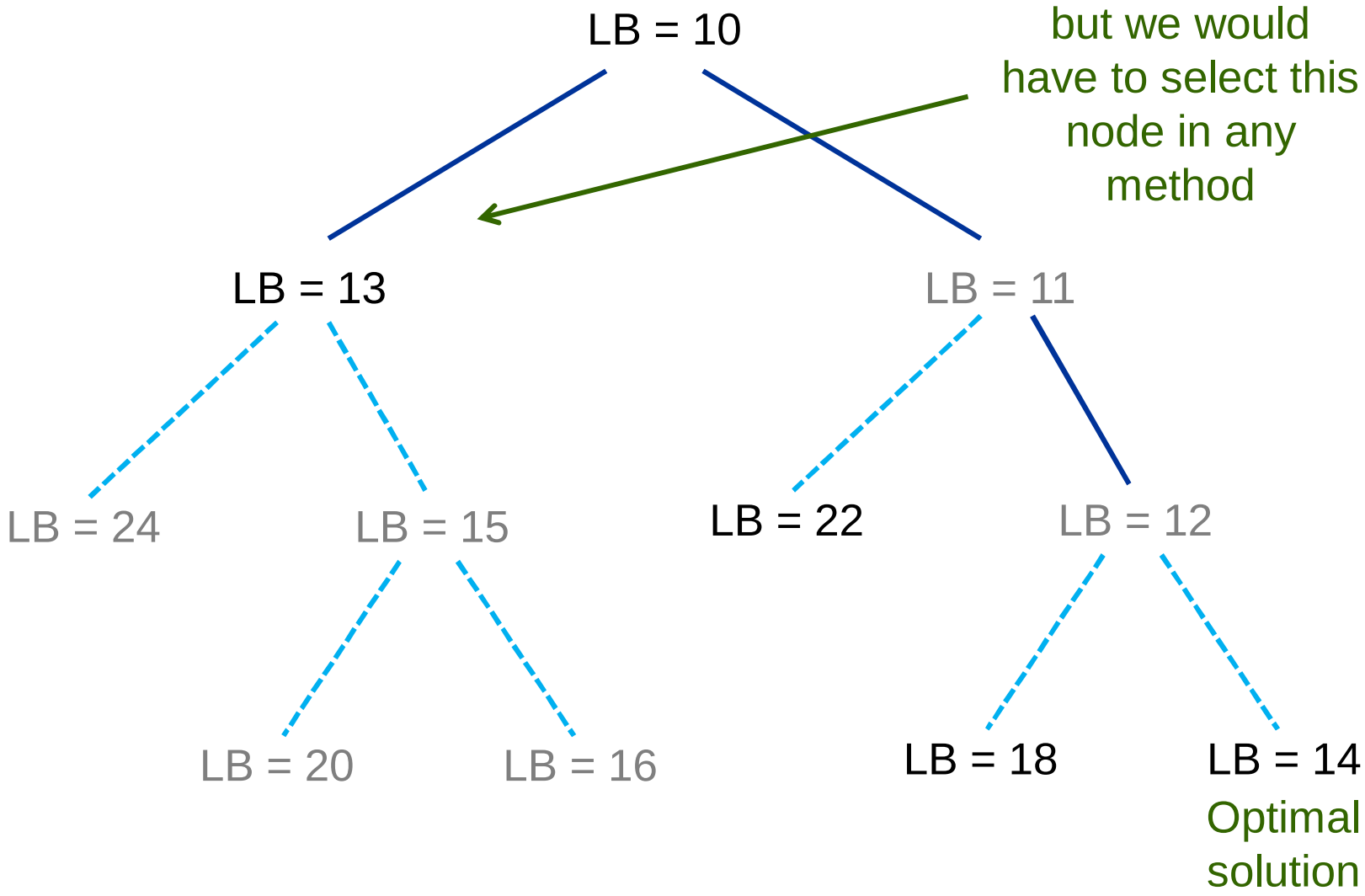
We are in the right subtree.

Optimal solution

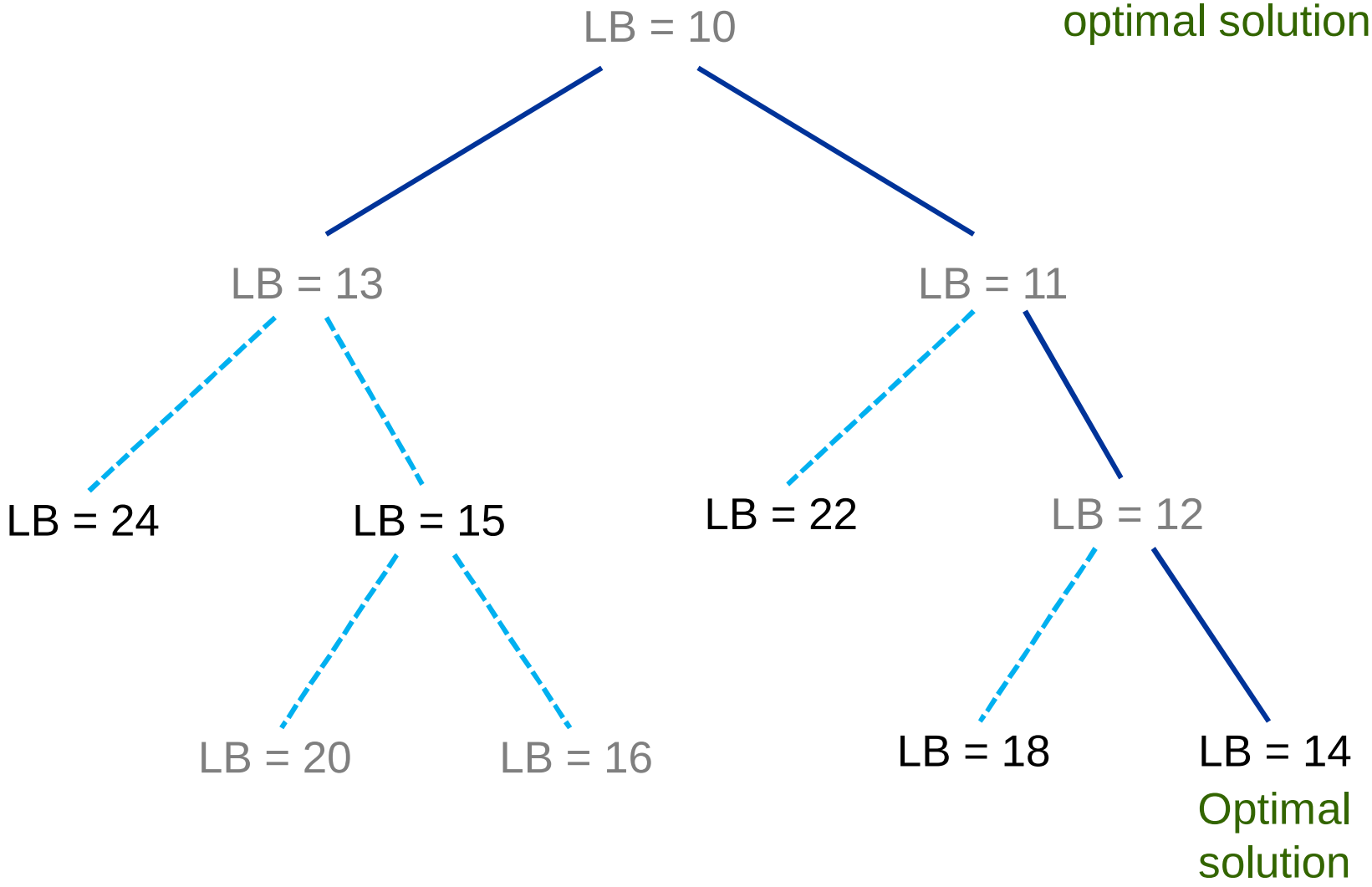
Best bound


LB = 10
LB = 13
LB = 11
LB = 24
LB = 15
LB = 22
LB = 12
LB = 20
LB = 16
LB = 18
LB = 14
Optimal solution
Select node with best bound.
We are still in the right subtree.

Best bound

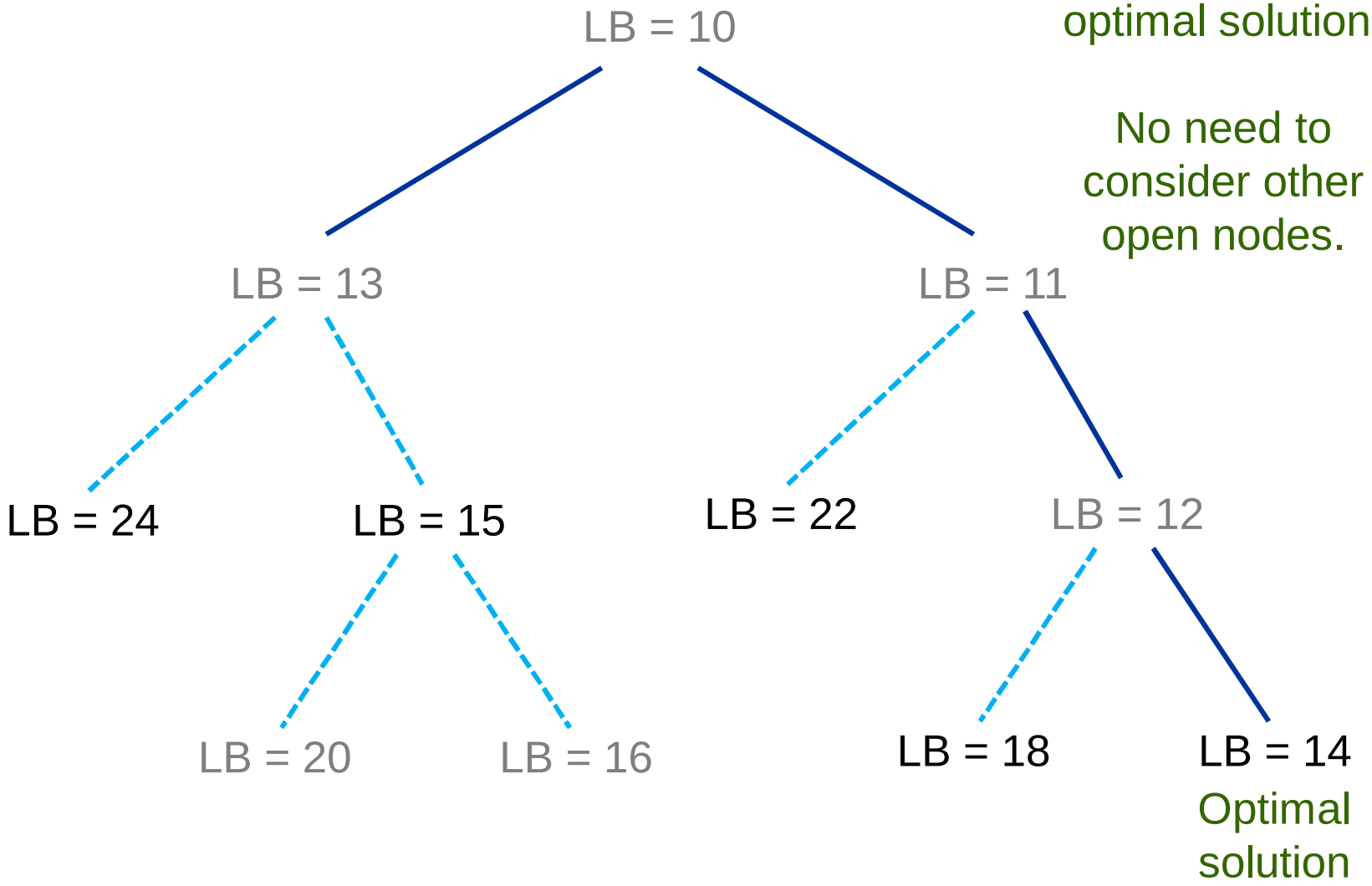

LB = 10
Wrong subtree, but we would have to select this node in any method
LB = 13
LB = 11
LB = 24
LB = 15
LB = 22
LB = 12
LB = 20
LB = 16
LB = 18
LB = 14
Optimal solution

Best bound



Next choice is optimal solution.

Best bound



Next choice is optimal solution.

No need to consider other open nodes.

Feasibility heuristics

- We wish find good integer solutions as early as possible.
  - To get useful upper bounds.

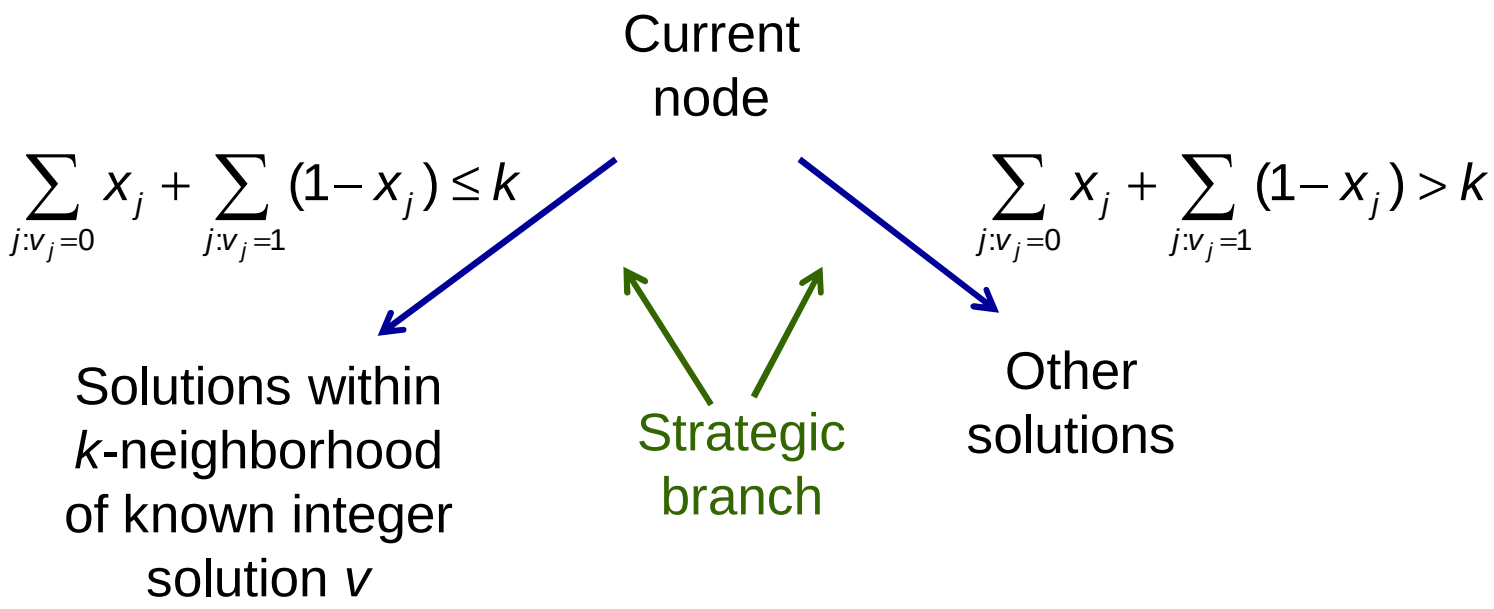
- Two approaches (among others):
  - Local branching.
  - Feasibility pump.

Local branching

- Must begin with known integer solution
- Two level branching procedure
  - Strategic branches define neighborhoods of known integer solutions.
  - Low-level branching searches each neighborhood using MIP

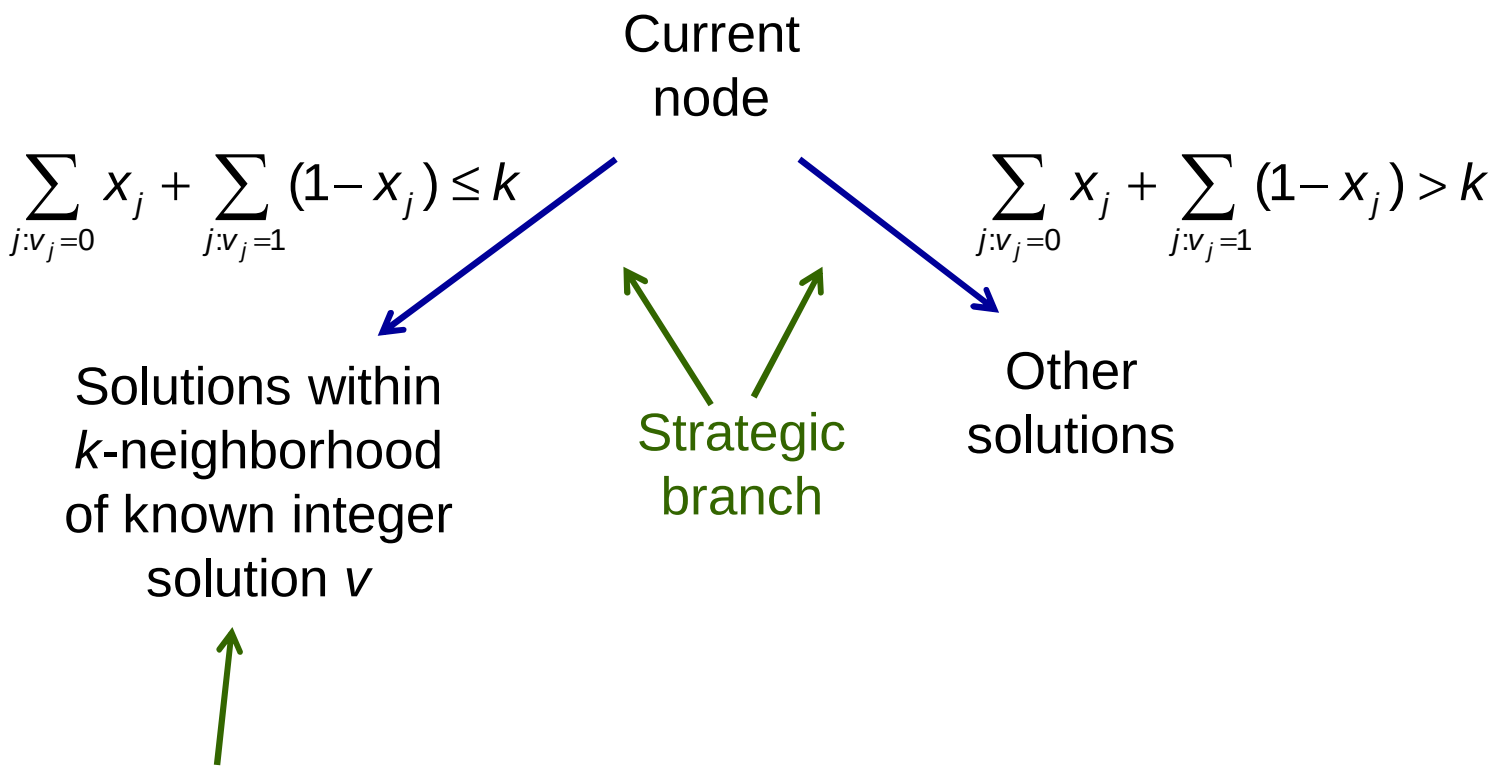
Local branching

Assume integer variables are binary.



Local branching

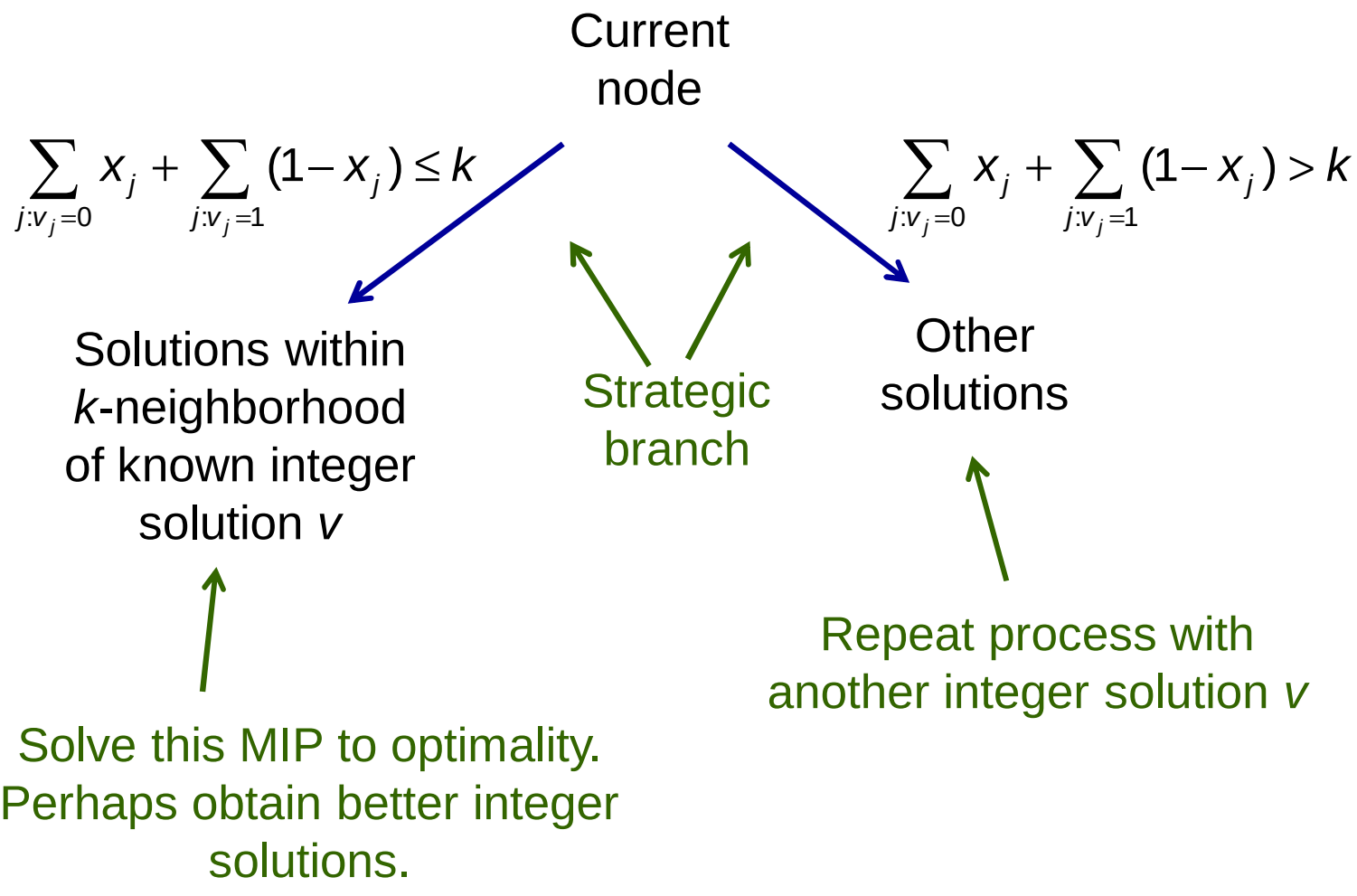
Assume integer variables are binary.



Solve this MIP to optimality. Perhaps obtain better integer solutions.

Local branching

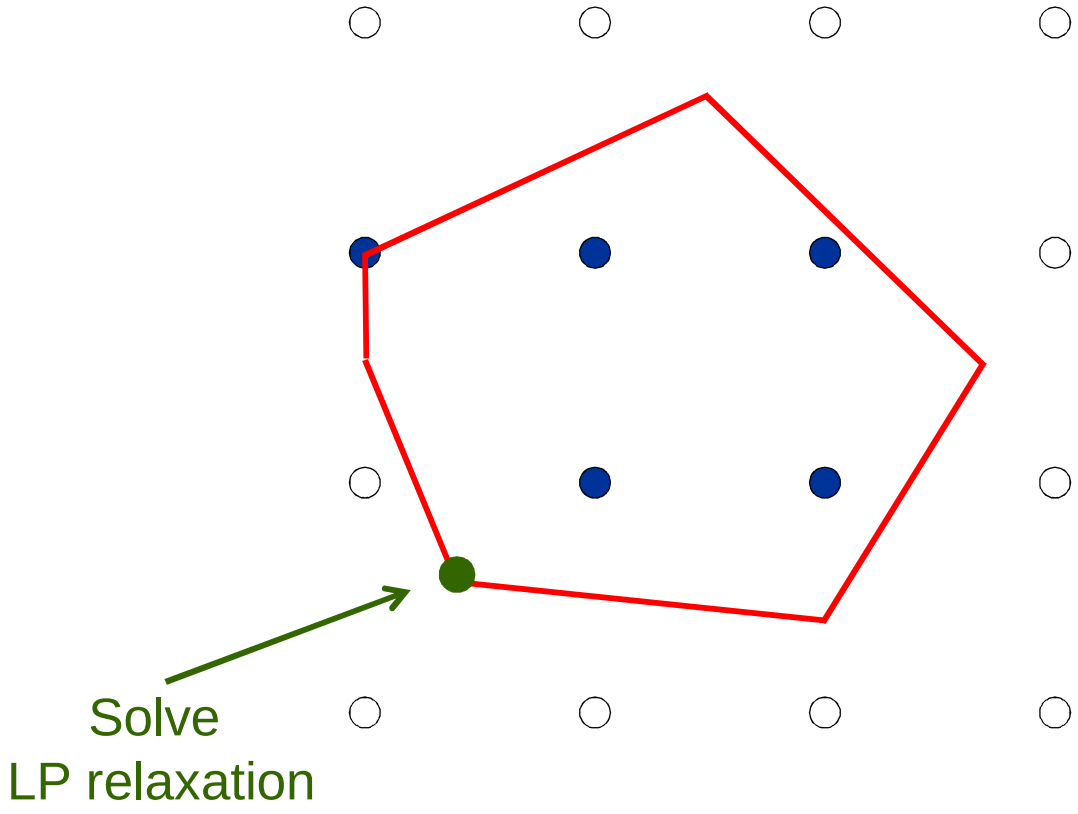
Assume integer variables are binary.



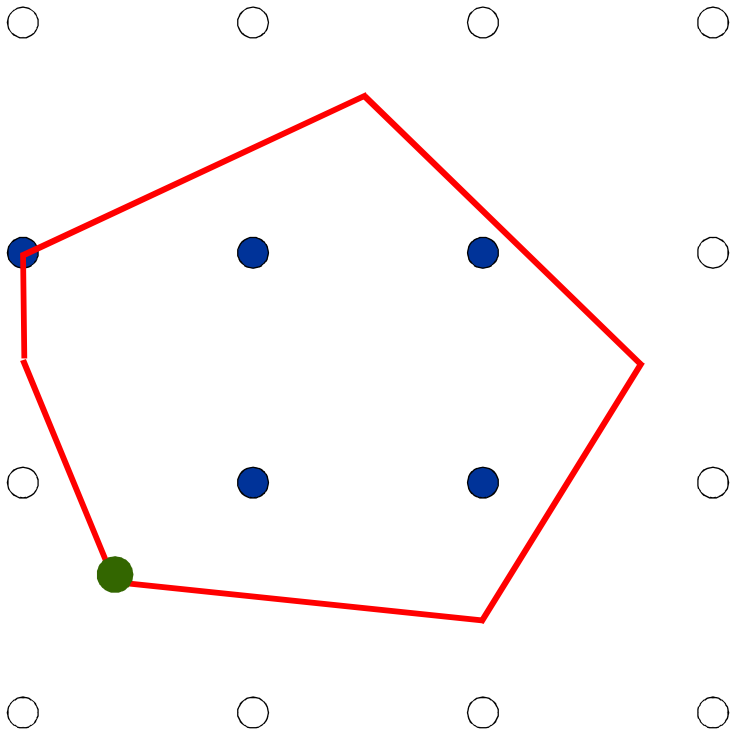
Feasibility pump

- Can find initial integer solution
- Alternate between integrality and linear feasibility
  - Solve LP
  - Round to integer
  - Find closest LP solution

Feasibility pump

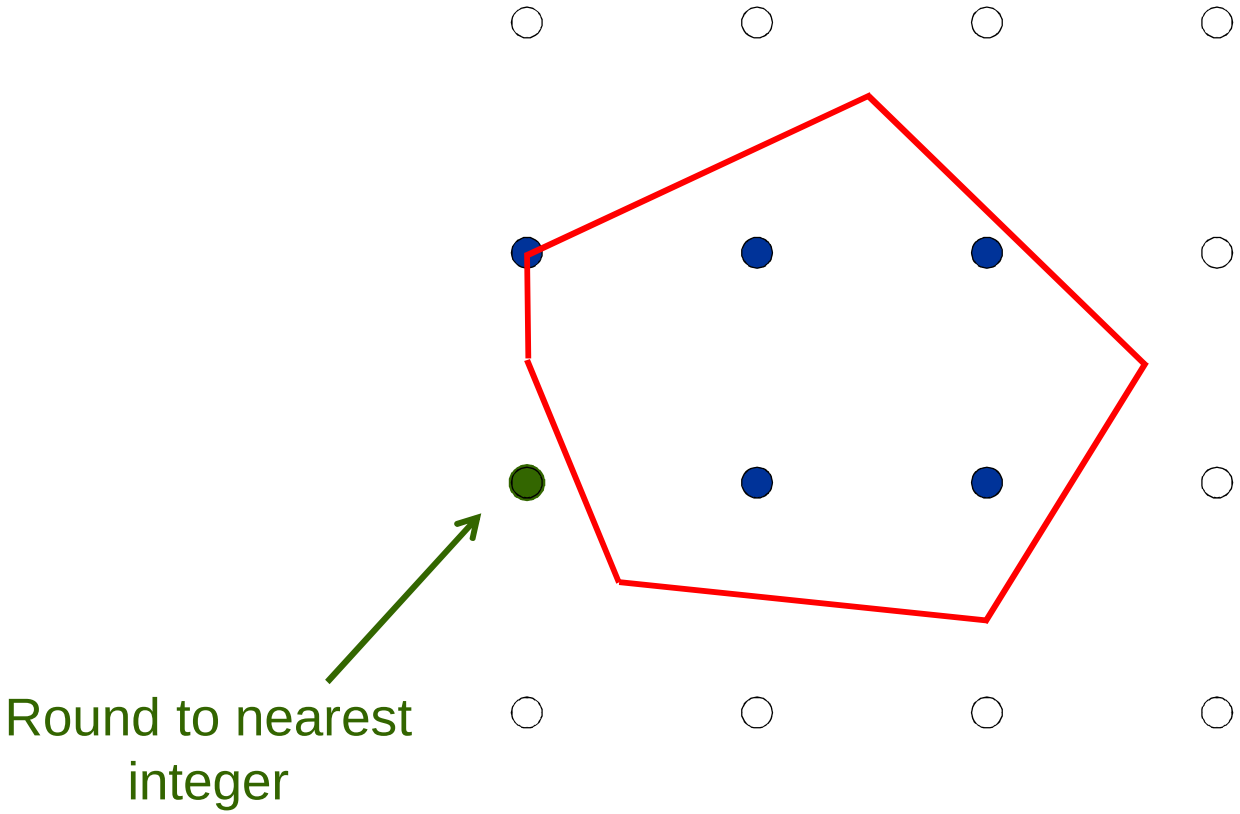


Feasibility pump

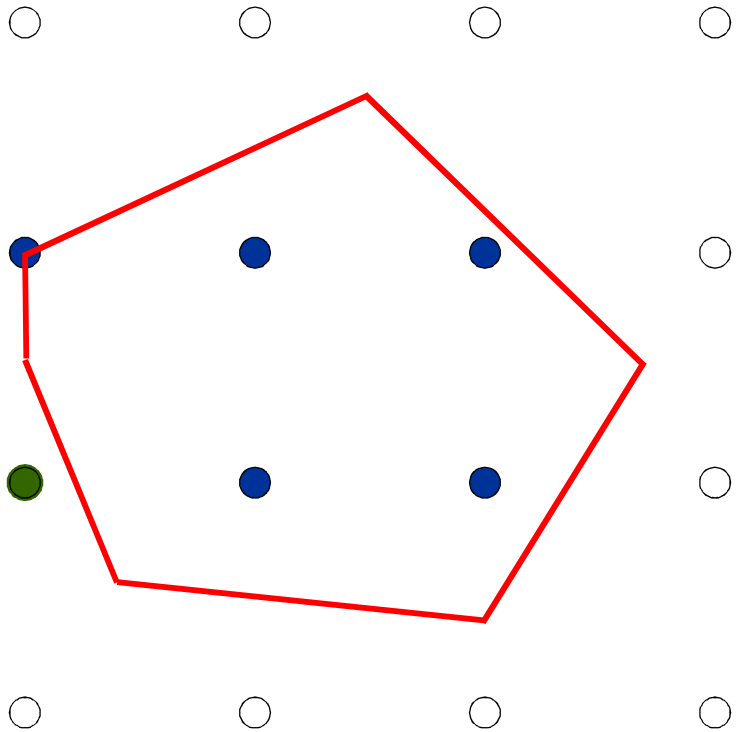


Round to nearest integer

Feasibility pump

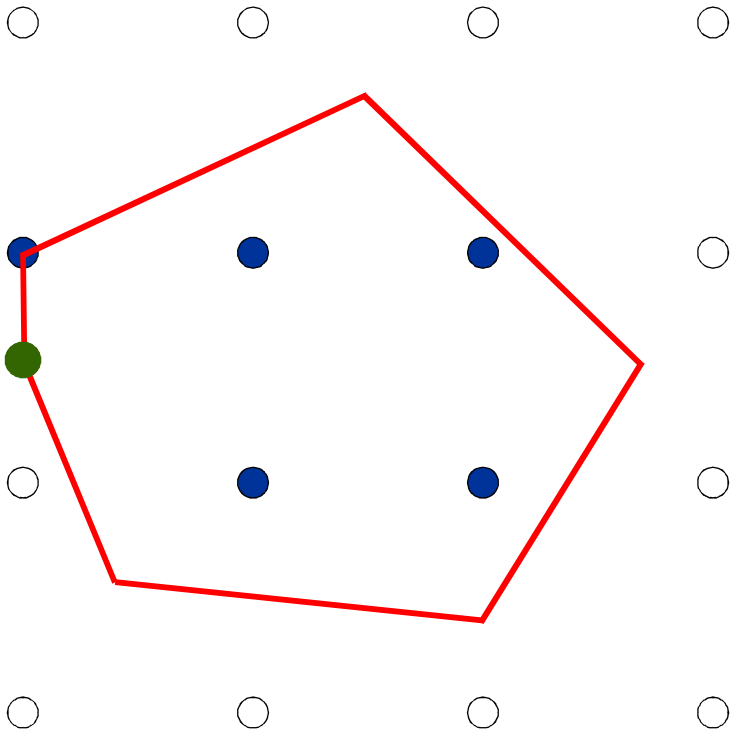


Feasibility pump



Find closest
LP-feasible point
using L_1 norm

Feasibility pump

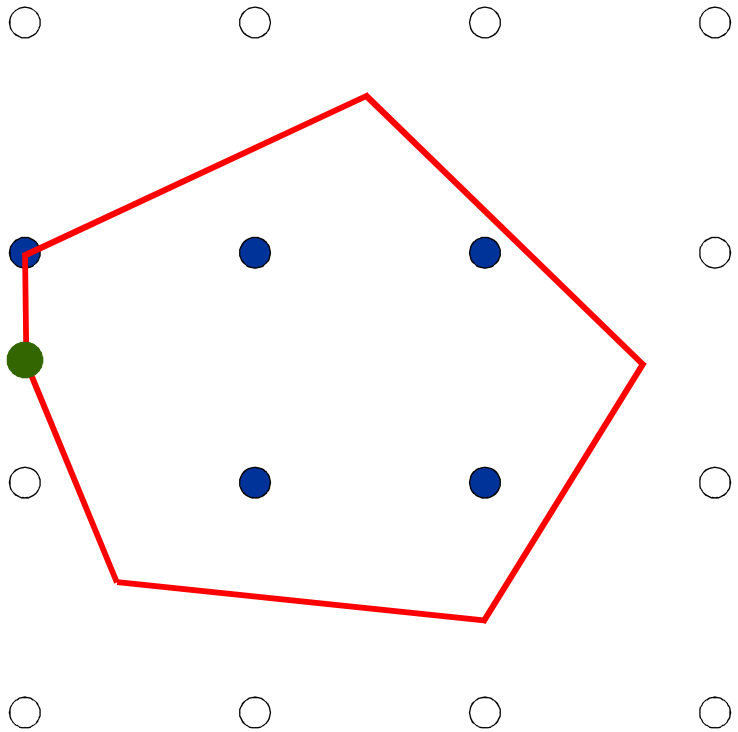


Find closest LP-feasible point using L_1 norm

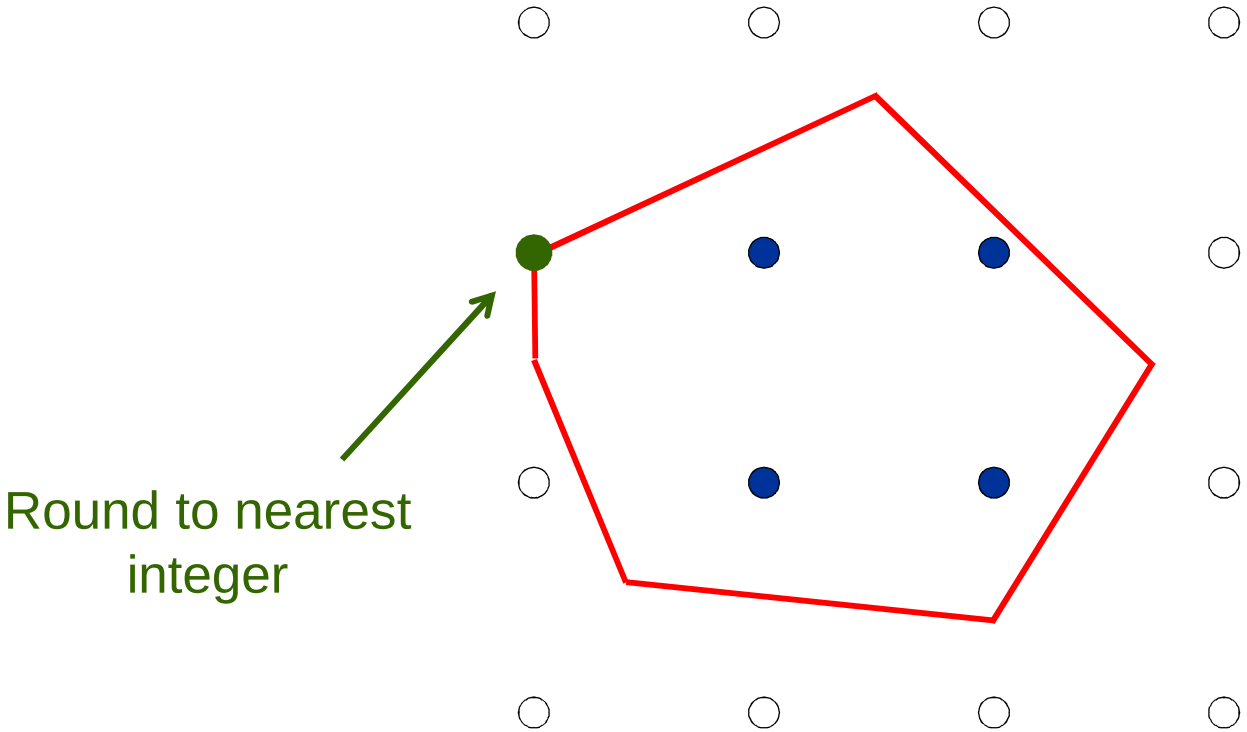
This problem is itself an LP

Feasibility pump

Round to nearest integer

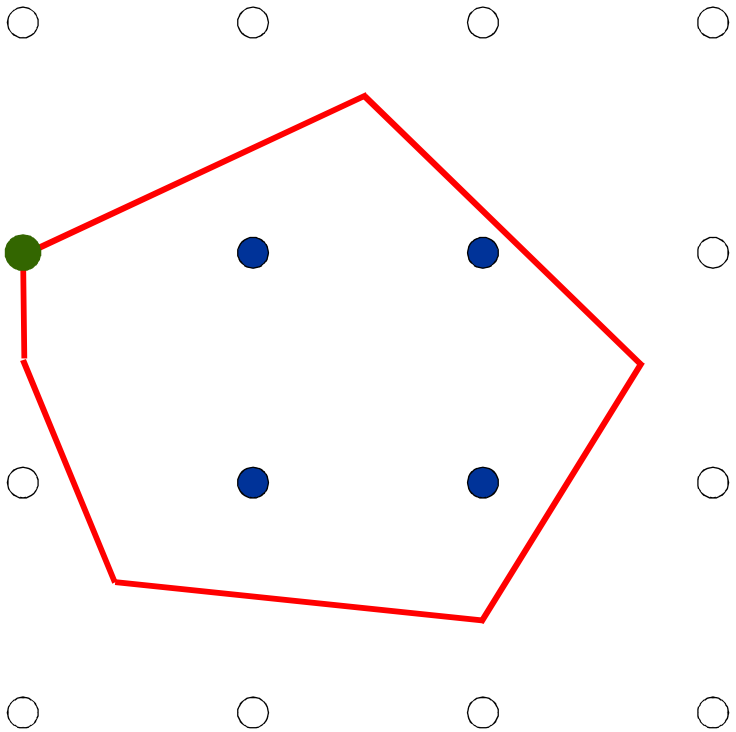


Feasibility pump



Procedure terminates

Feasibility pump



Round to nearest integer

Use random perturbations to break out of loops

Procedure terminates

Duality and Nogoods

Inference duality

Nogood-based search

Example: SAT

Example: MIP

Inference duality

- Duality in optimization is closely related to nogood-based search in AI.
  - Nogood = set of solutions to avoid in the rest of the search.
- All optimization duals are inference duals
  - LP, Lagrangean, surrogate, subadditive, etc.
  - Solution of inference dual is **proof** of optimality
- Dual solution provides nogoods
  - Including Benders cuts, conflict clauses for SAT
  - Nogood = conditions under which proof is still valid

Inference duality

Primal

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

Feasible set

Dual

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

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

$\max v$

$\left\{ \begin{array}{l} Ax \geq b \\ x \geq 0 \end{array} \right\} \overset{P}{\Rightarrow} cx \geq v$

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

$= \max \lambda b$

$\lambda A \leq c$

$\lambda \geq 0$

$Ax \geq b \overset{x \geq 0}{\Rightarrow} cx \geq v$ 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).

Duals for MIP

- Lagrangean dual
  - Inference = same as LP
  - Inference method incomplete $\rightarrow$ duality gap
- Surrogate dual
  - Inference = same as LP, except domination is weaker
  - Therefore smaller duality gap than Lagrangean dual
- Subadditive dual
  - Inference = subadditive, homogeneous function + domination
  - Inference method complete (cf. Gomory's method) $\rightarrow$ no gap

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

Example: Logic-based Benders

- Master and subproblem may be solved by different techniques
  - MIP for master problem
  - CP for subproblem
- Benders cuts are problem-specific
  - Based on type of inference in dual
- Some applications
  - Planning & scheduling (e.g., assembly lines)
  - Computer processor allocation and scheduling
  - Chemical batch sizing and scheduling
  - Sports scheduling
  - Location/allocation

Example: 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

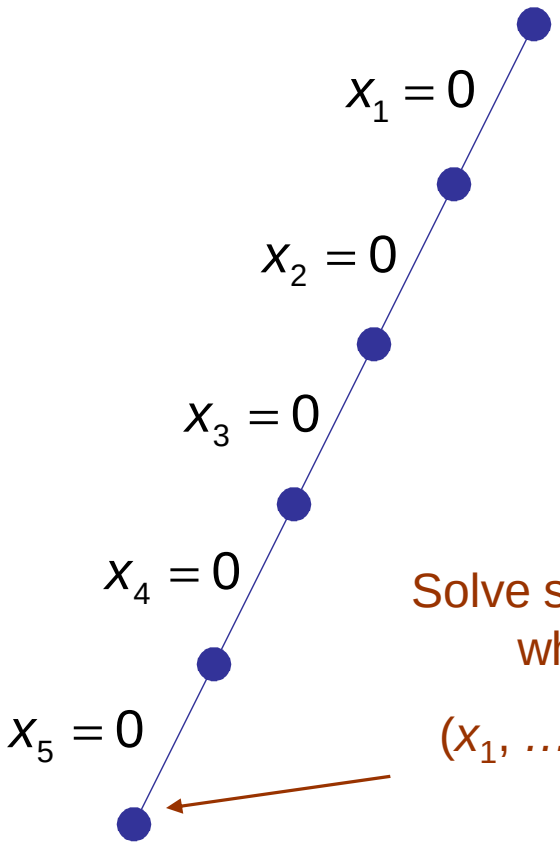
- Solve SAT by chronological backtracking + unit clause rule = DPL.
  - Chronological = fixed branching order.

- To get nogood, solve subproblem at current node.
  - Solve 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

DPL with chronological backtracking

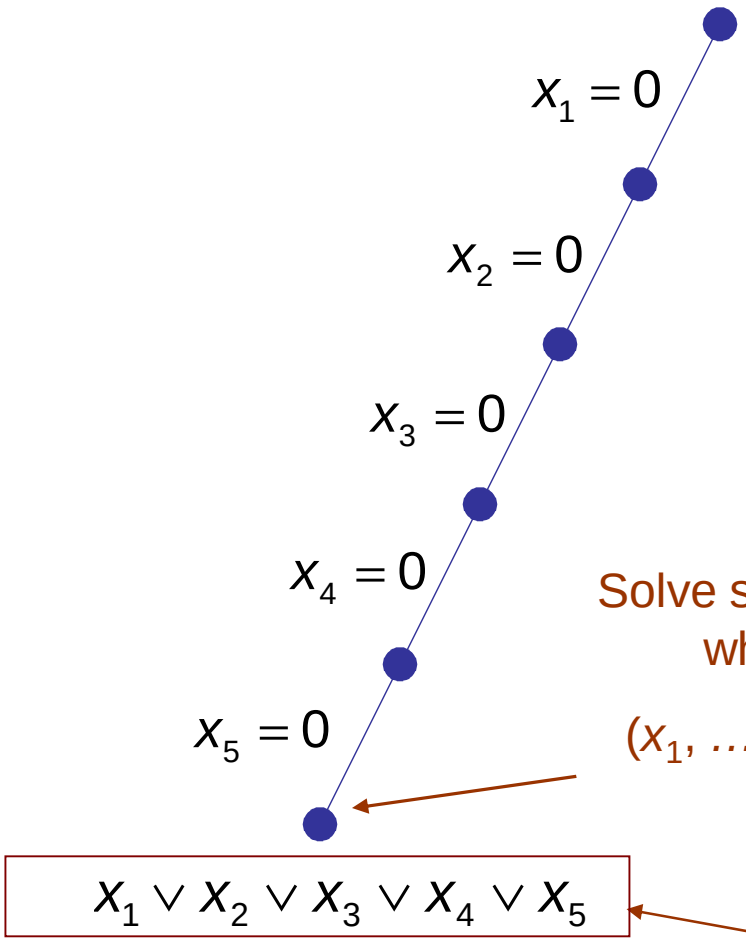


Branch to here.

Solve subproblem with unit clause , which proves infeasibility.

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

DPL with chronological backtracking



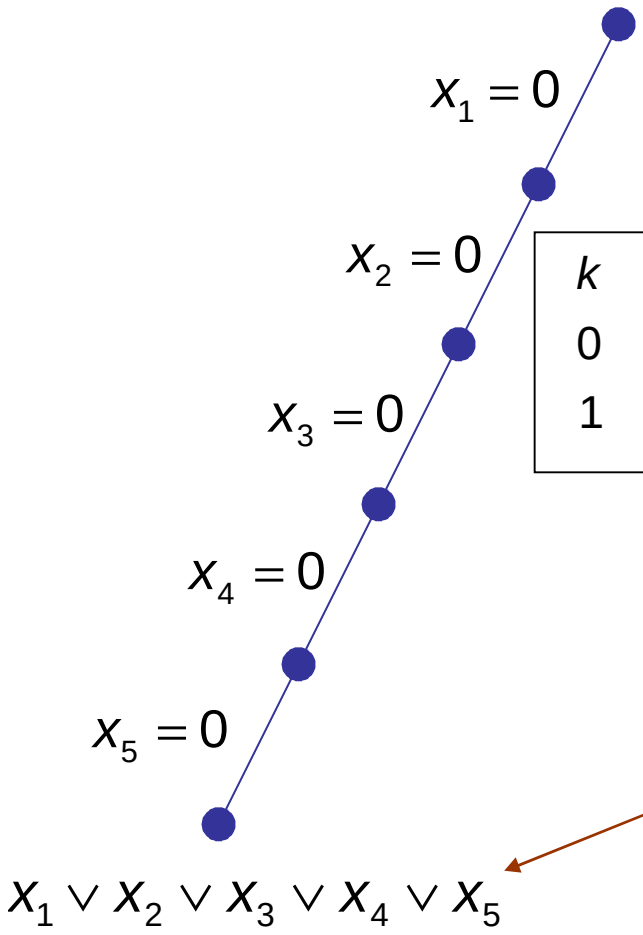
Branch to here.

Solve subproblem with unit clause , which proves infeasibility.

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

Generate nogood.

DPL with chronological backtracking

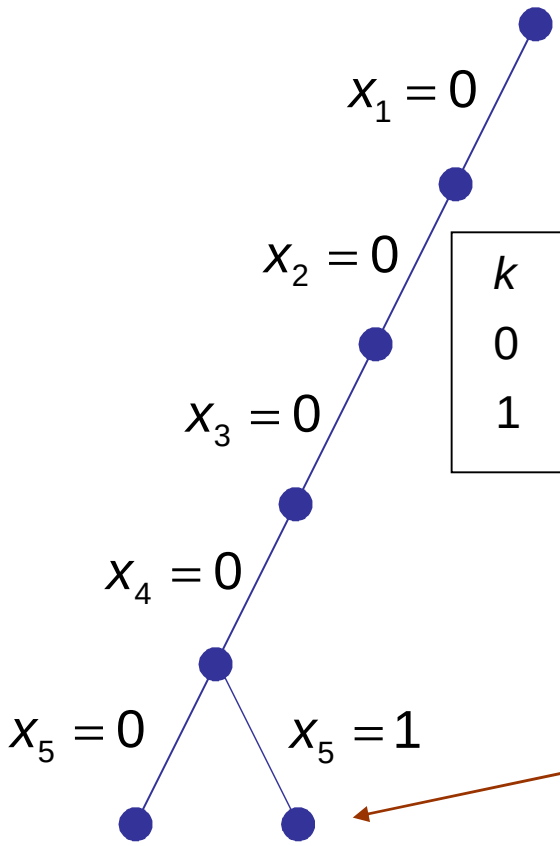


Consists of processed nogoods

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.

DPL with chronological backtracking

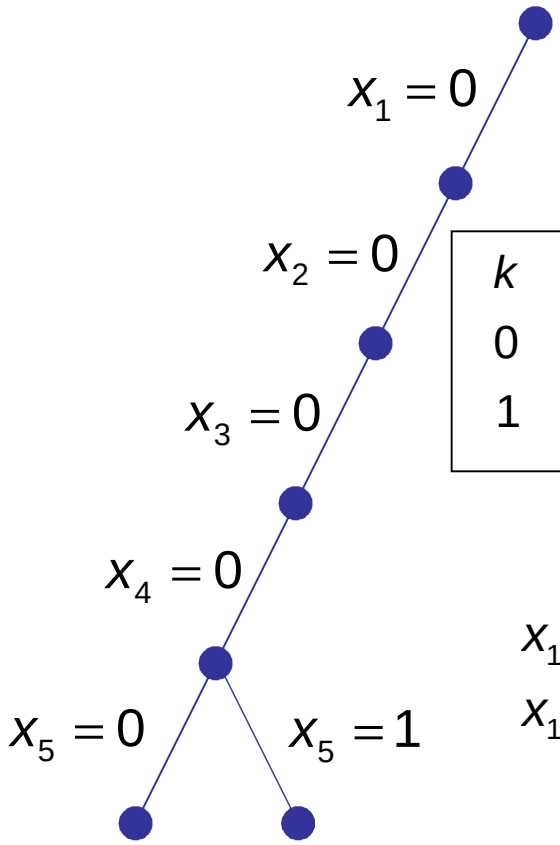


Consists of processed nogoods

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

DPL 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	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$	$(0,0,0,0,1,\cdot)$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \bar{x}_5$

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

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

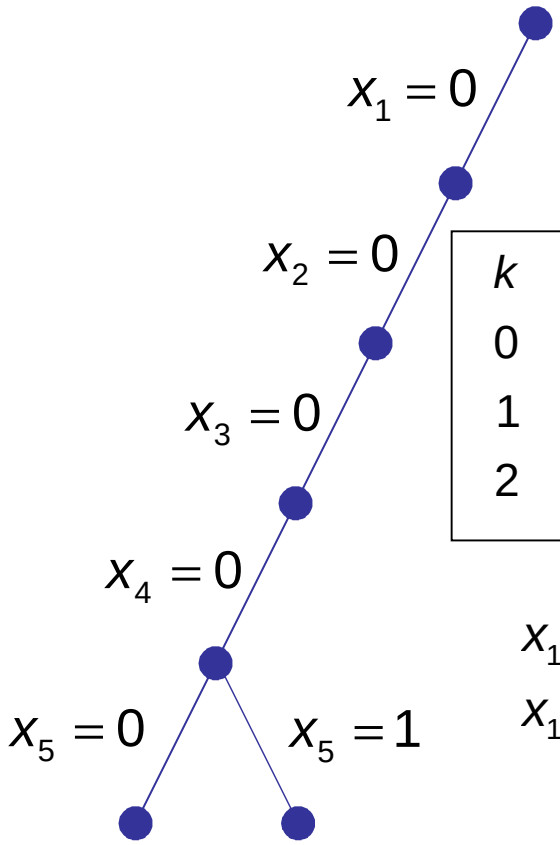
Process nogood set with parallel resolution

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

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

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

DPL 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	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$	$(0,0,0,0,1,\cdot)$	$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \bar{x}_5$
2	$x_1 \vee x_2 \vee x_3 \vee x_4$		

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

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

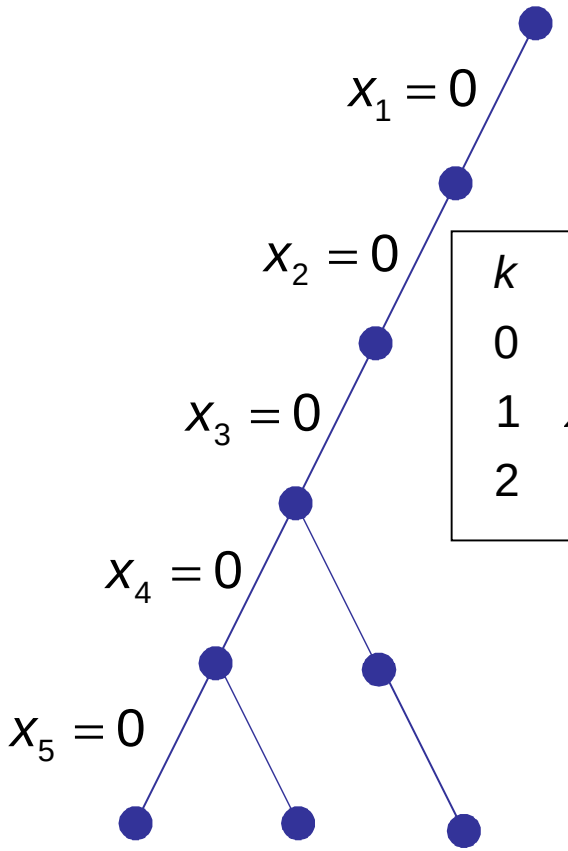
Process nogood set with parallel resolution

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

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

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

DPL 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	$x_1 \vee x_2 \vee x_3 \vee x_4$	$(0,0,0,1,0,\cdot)$	

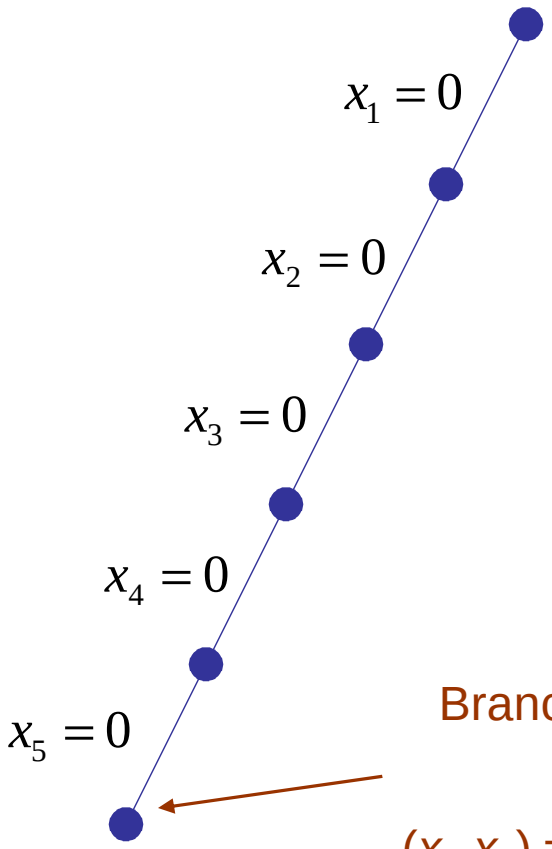
Solve relaxation again, continue.

So backtracking is nogood-based search with parallel resolution

Example: SAT and conflict clauses

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

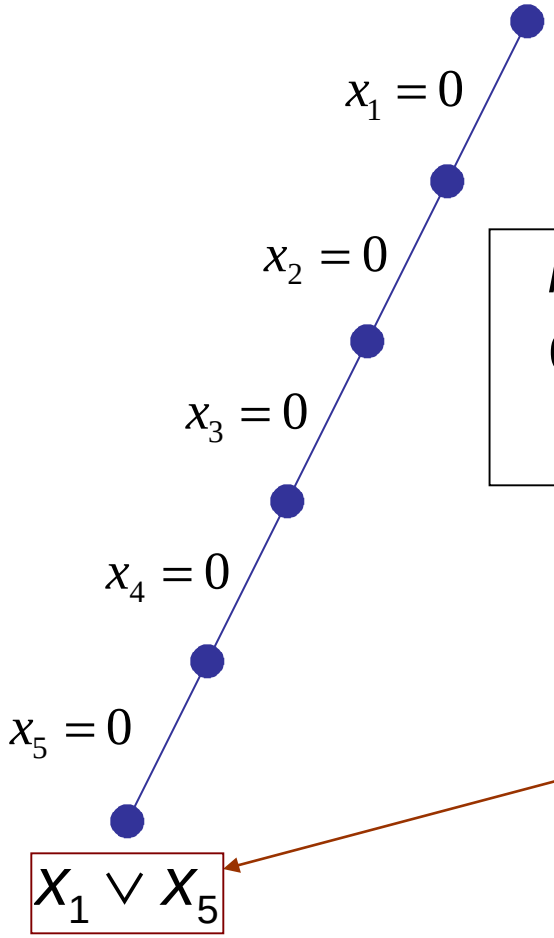
DPL with conflict clauses



Branch to here. Unit clause rule proves infeasibility.

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

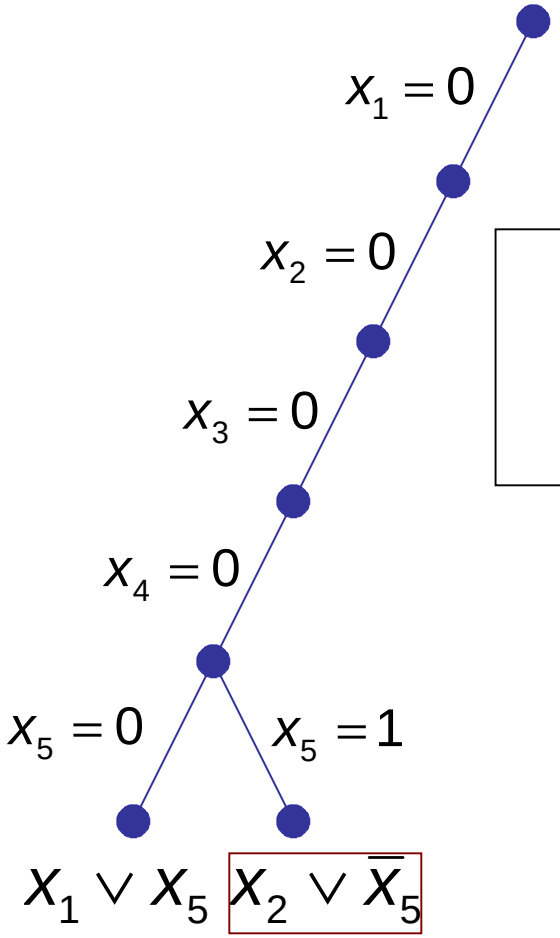
DPL with conflict clauses



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.

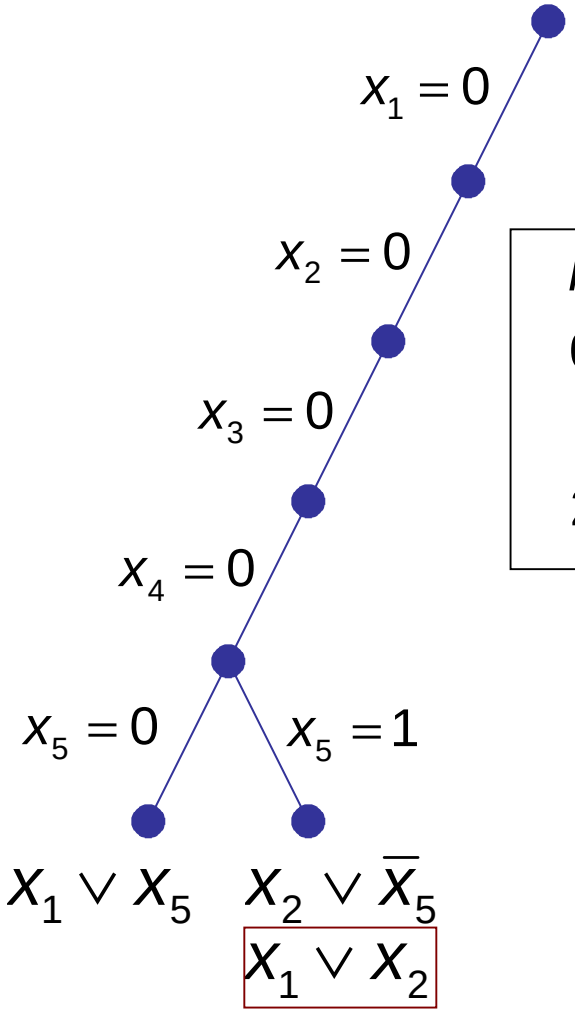
DPL with conflict clauses



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	$x_1 \vee x_5$	$(0,0,0,0,1,\cdot)$	$x_2 \vee \bar{x}_5$

DPL 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	$x_1 \vee x_2$		

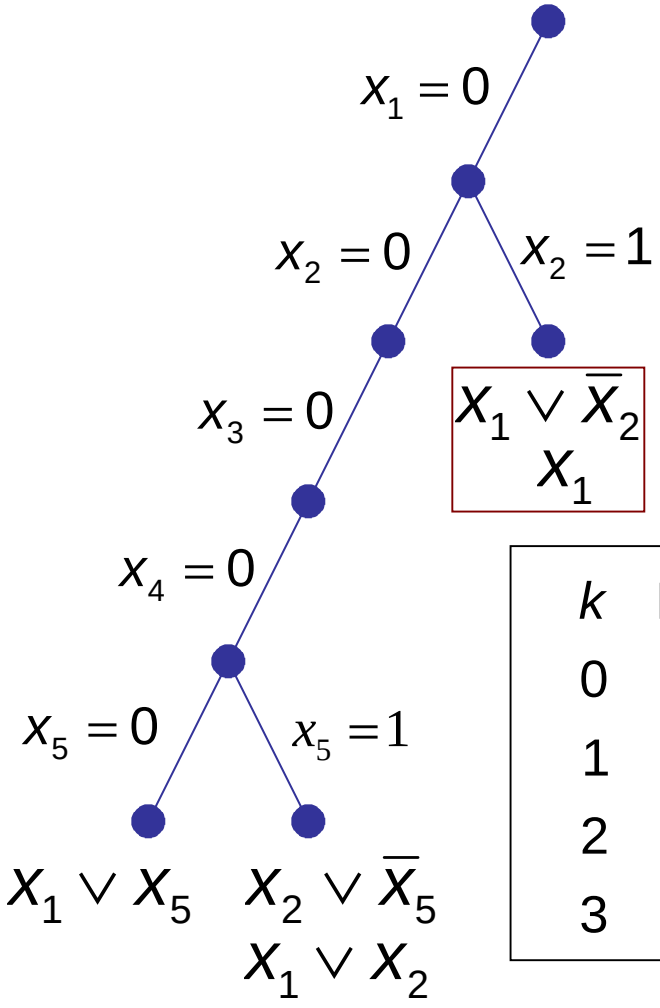
$x_1 \vee x_5$

$x_2 \vee \bar{x}_5$

parallel-resolve to yield $x_1 \vee x_2$

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

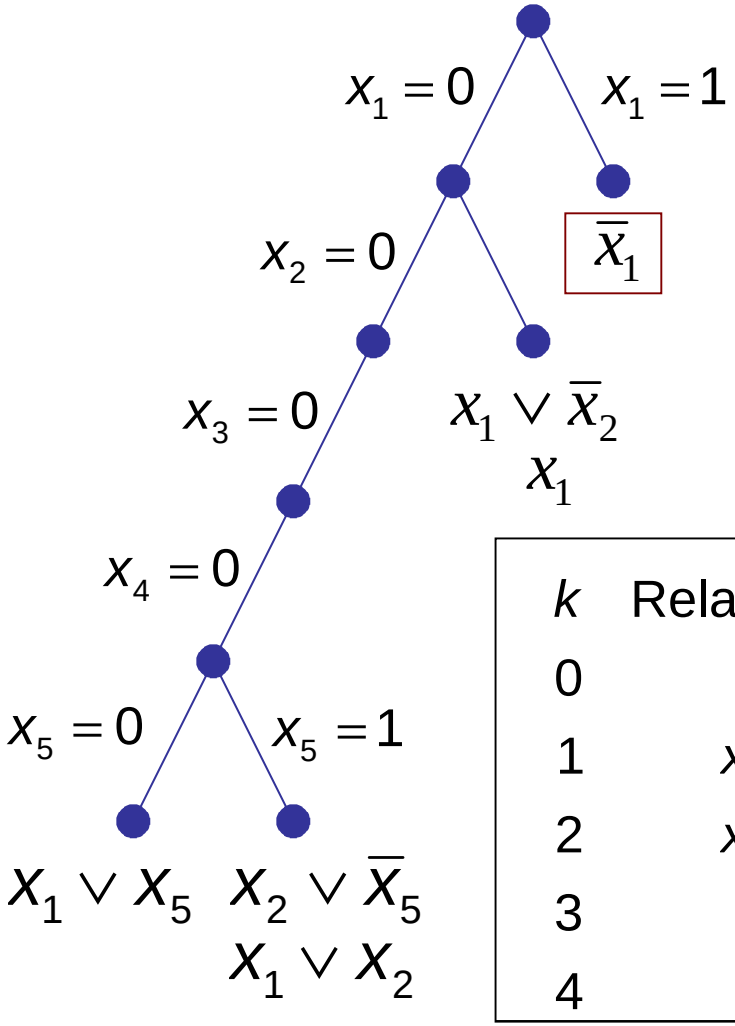
DPL with conflict clauses



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

DPL 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	$\emptyset$		

Search terminates

Example: SAT and 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	$x_1 \vee x_5$	$(1,\cdot,\cdot,\cdot,0,\cdot)$	$x_5 \vee \bar{x}_1$
2	x_5	$(\cdot,0,\cdot,\cdot,1,\cdot)$	$\bar{x}_5 \vee x_2$
3	$\{x_5, \bar{x}_5 \vee x_2\}$	$(\cdot,1,\cdot,\cdot,1,\cdot)$	$\bar{x}_2$
4	$\emptyset$		

Search terminates

Must conform

Example: MIP

- Nogoods now being used in MIP
  - For example, in SIMPL and SCIP.
- So we come full circle MIP $\rightarrow$ AI $\rightarrow$ MIP.

Binary Decision Diagrams

Motivating idea

Among constraint

BDDs

Propagation through a BDD

Computational results

Motivating idea

- Domain store is key element of constraint programming (CP)
  - Consists of current variable domains.

- Domain store propagates results of filtering domains.
  - Constraint-specific filtering algorithms remove infeasible values from domains.
  - Reduced domains are handed over to next constraint.

Motivating idea

- Problem: Domain store is a weak relaxation
  - It encodes limited information.
  - Not worth investing time to process nodes of search tree.

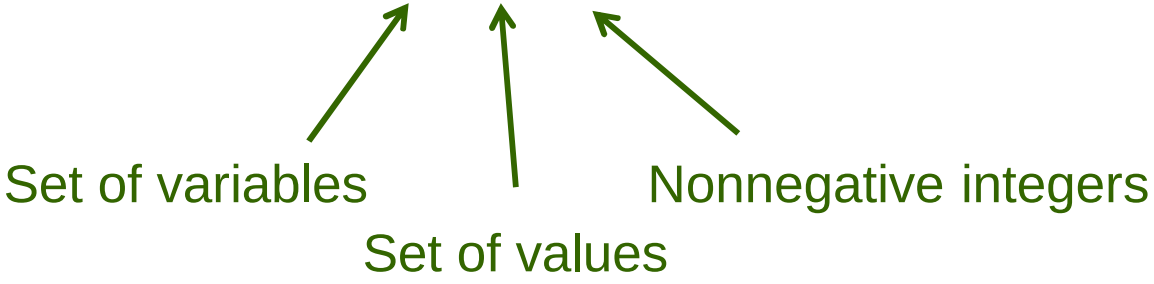
- LP is a strong relaxation for MIP
  - It pays to invest time at nodes (solve LP, add cutting planes)

- Needed: A strong, discrete relaxation for CP
  - Adaptable to any constraint
  - Domain store should be a degenerate case

Among constraint

- We illustrate the idea using the **among** constraint.
  - Important in sequencing and scheduling.

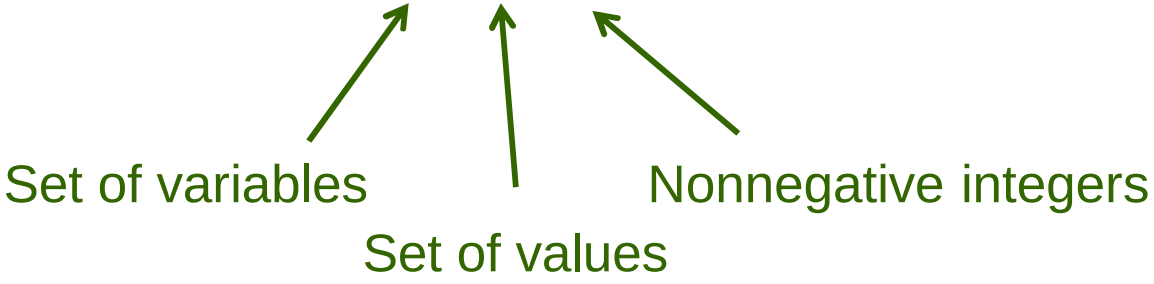
$$\text{among}(X, S, \ell, u)$$



Among constraint

- We illustrate the idea using the **among** constraint.
  - Important in sequencing and scheduling.

$$\text{among}(X, S, \ell, u)$$



The constraint requires that at least ℓ and at most u variables in X take a value in S

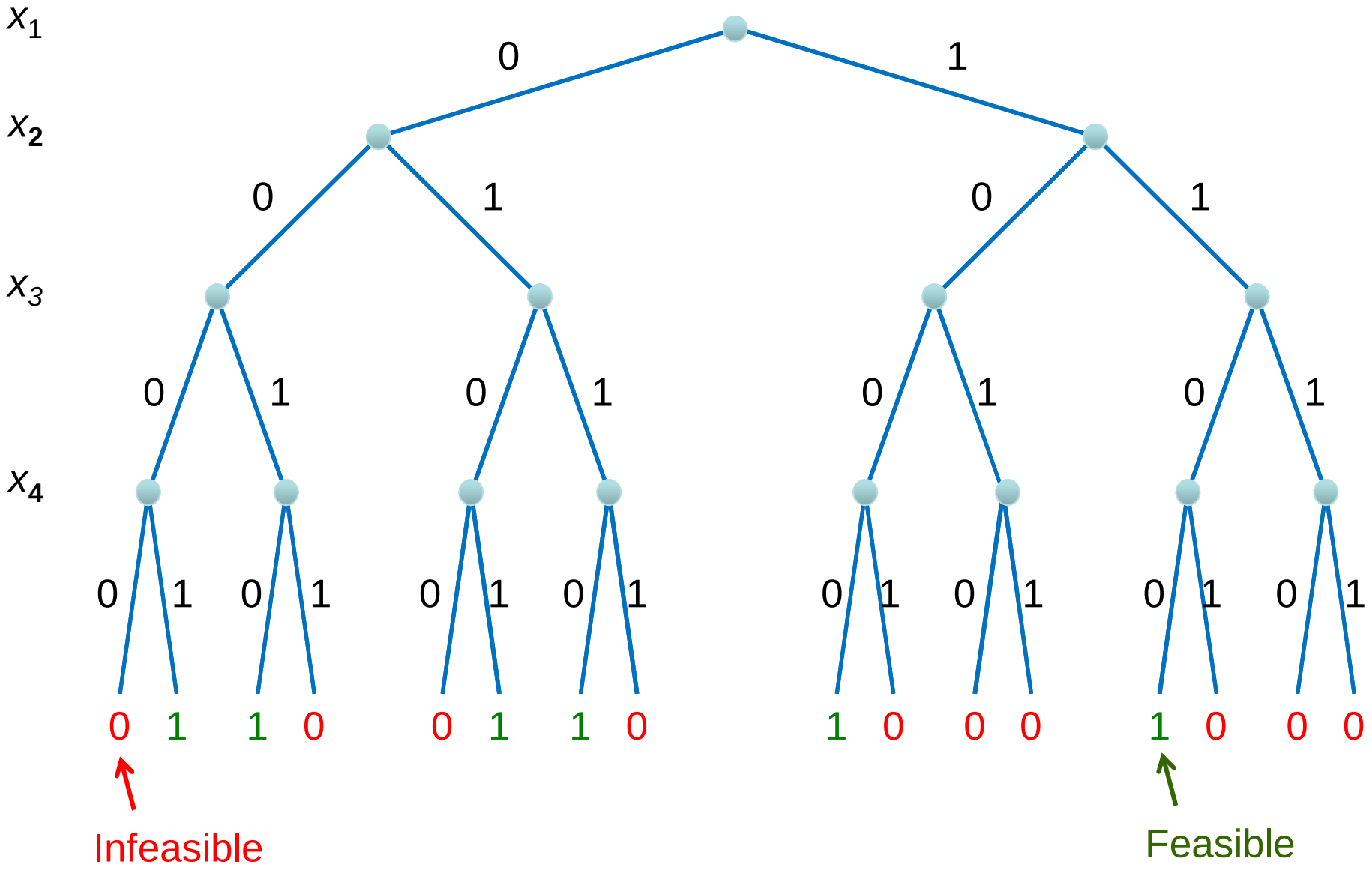
Binary decision diagrams

- A BDD is a graphical representation of a boolean function
  - It can also represent constraint on binary variables.
  - The boolean function is 1 when the constraint is satisfied.

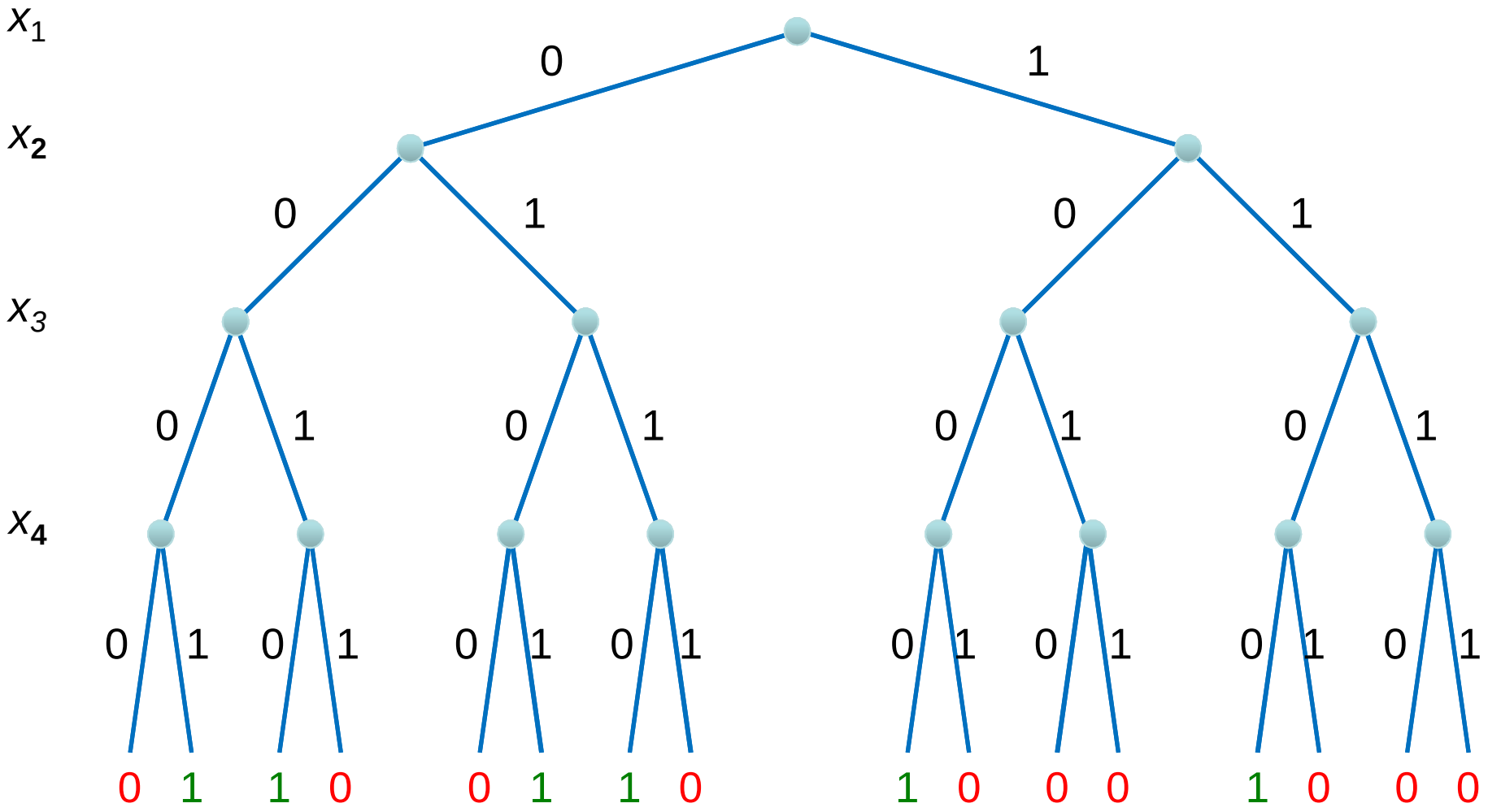
- A (reduced) BDD is basically a compact representation of a branching tree.
  - Superimpose isomorphic subtrees.
  - Remove unnecessary nodes.
  - There is a unique reduced BDD for a given branching order.

- Generalizes to multivalued decision diagrams (MDDs) for arbitrary discrete variables.

Search tree for $\text{among}(\{x_1, x_3, x_4\}, \{1\}, 2, 2)$

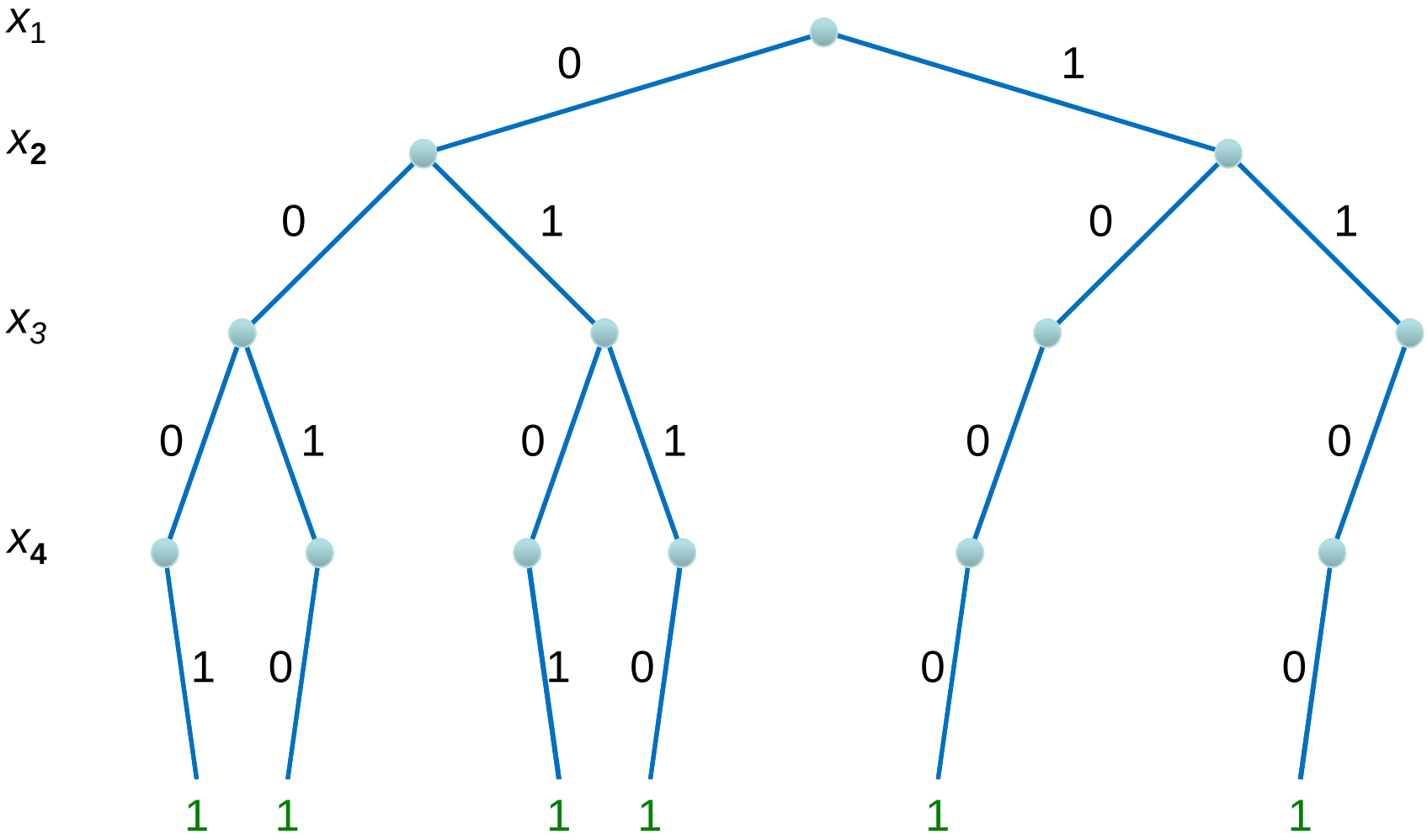


$\text{among}(\{x_1, x_3, x_4\}, \{1\}, 2, 2)$



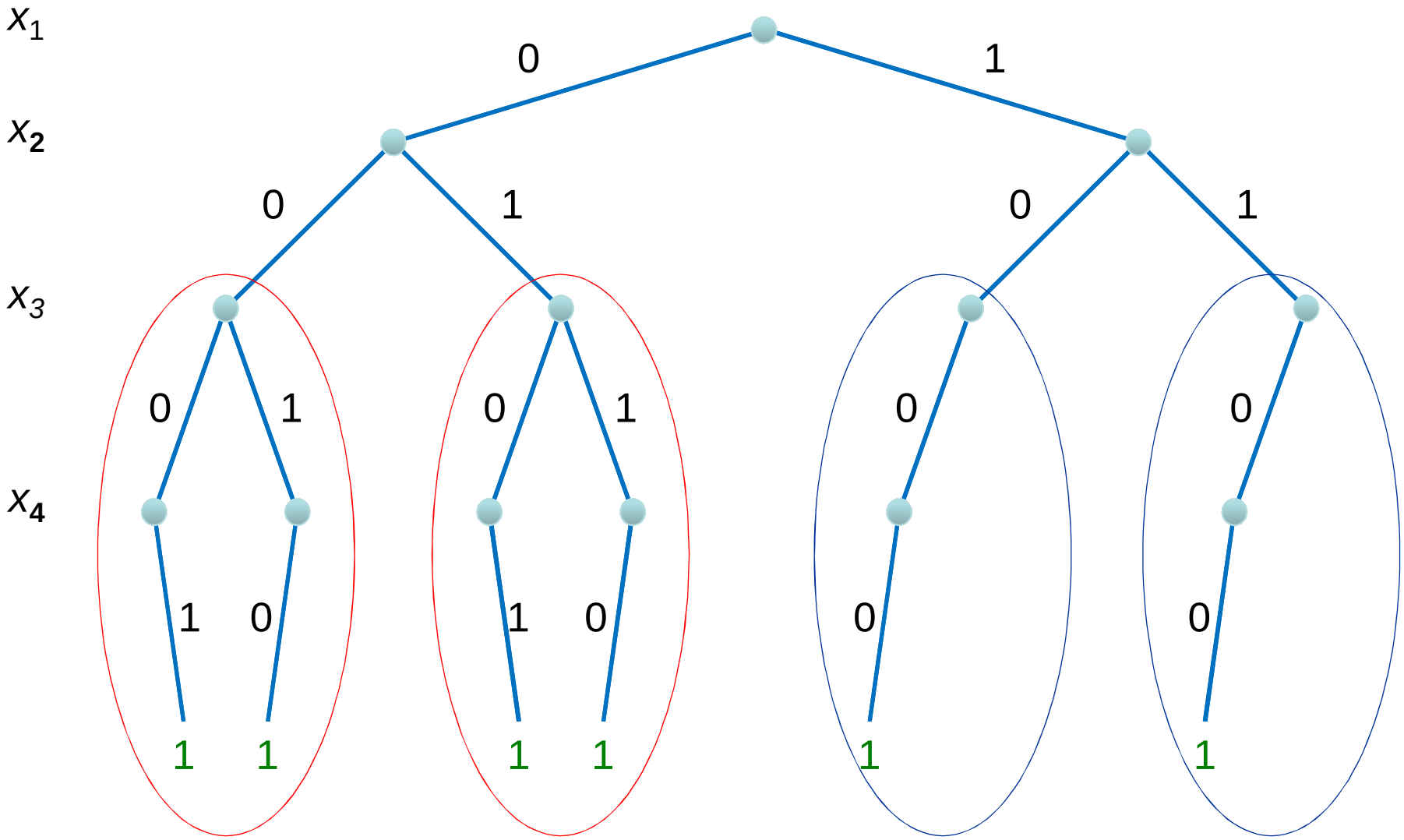
Remove infeasible paths

$\text{among}(\{x_1, x_3, x_4\}, \{1\}, 2, 2)$



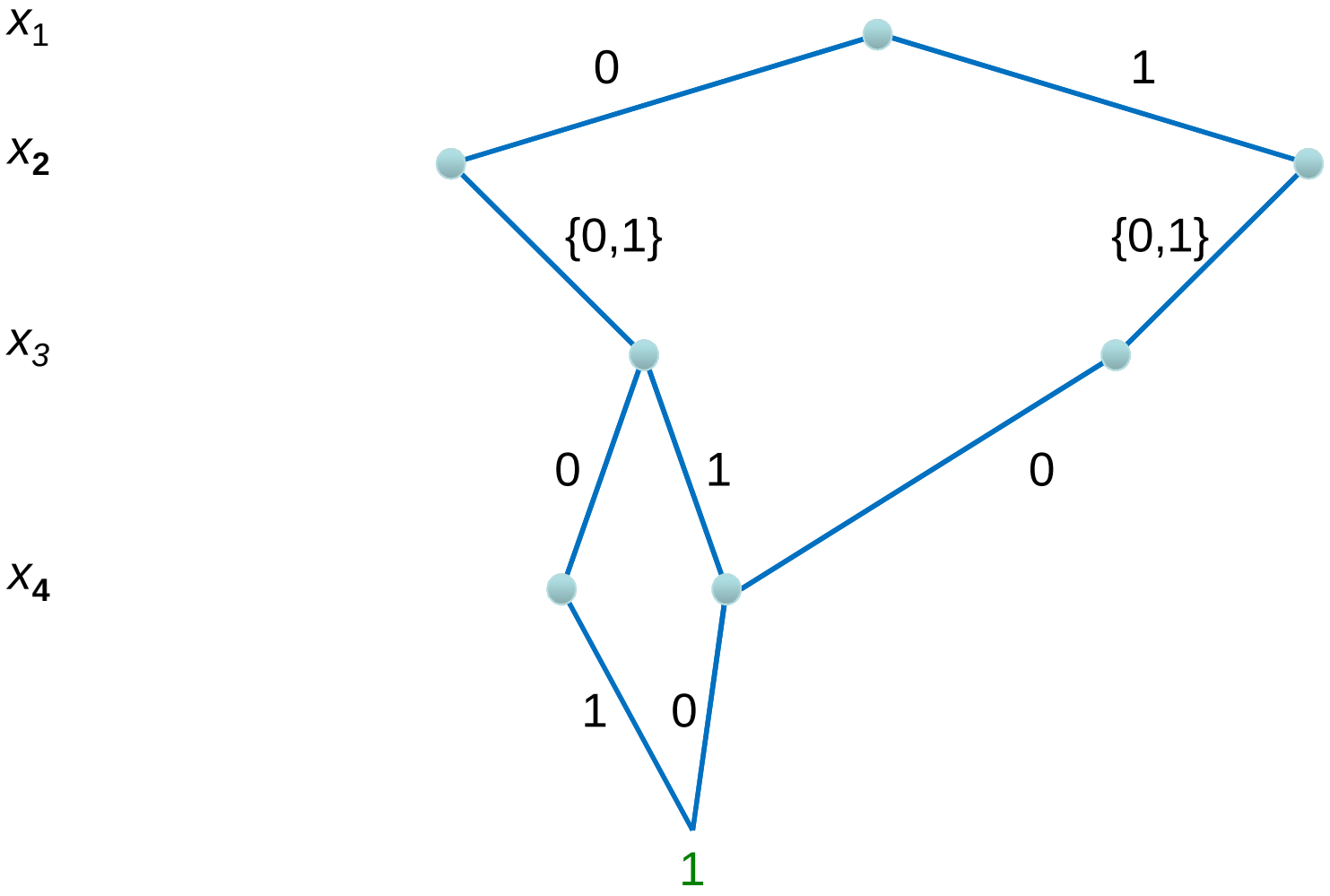
Remove infeasible paths

$\text{among}(\{x_1, x_3, x_4\}, \{1\}, 2, 2)$



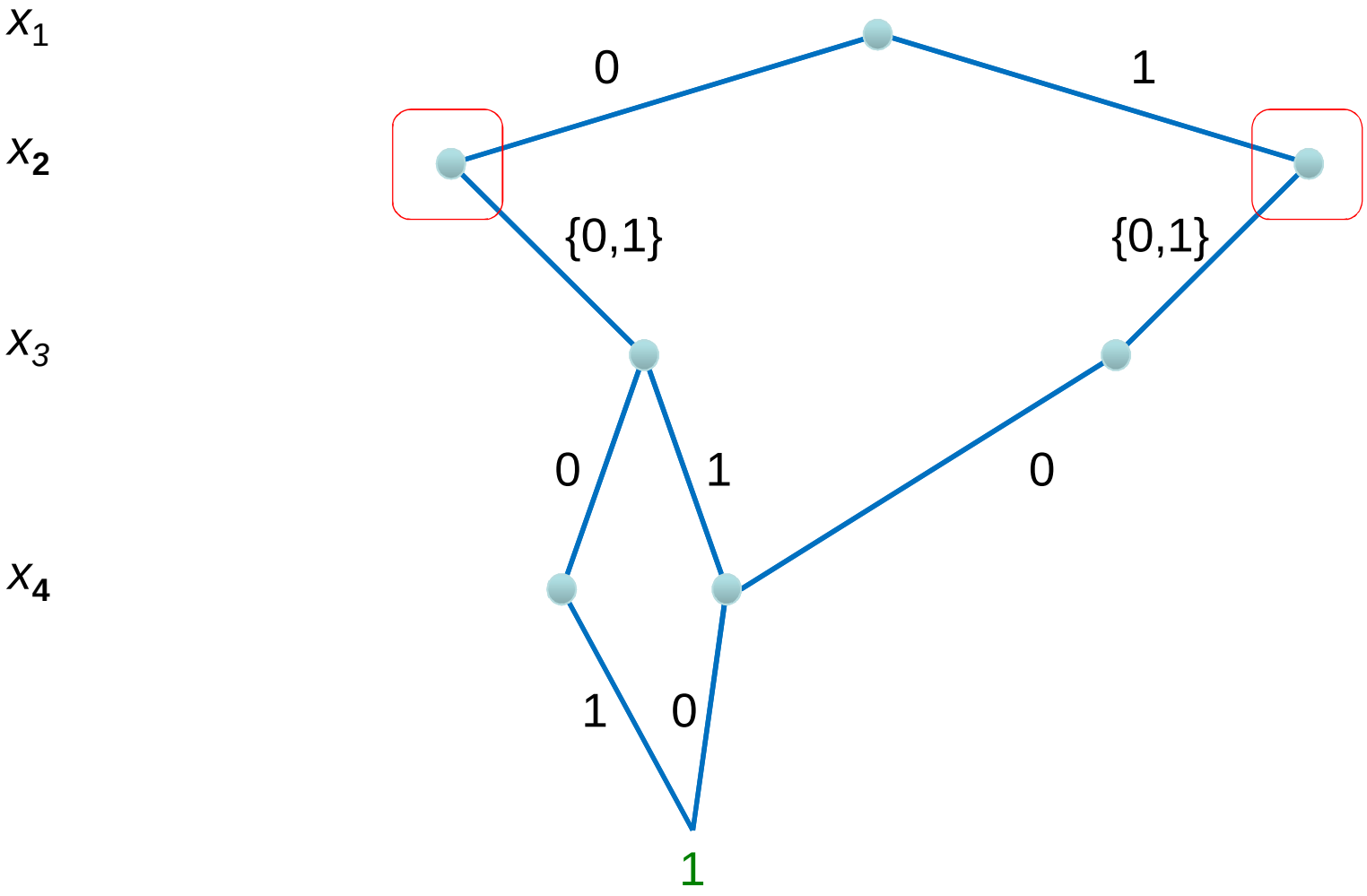
Merge isomorphic subtrees

$\text{among}(\{x_1, x_3, x_4\}, \{1\}, 2, 2)$



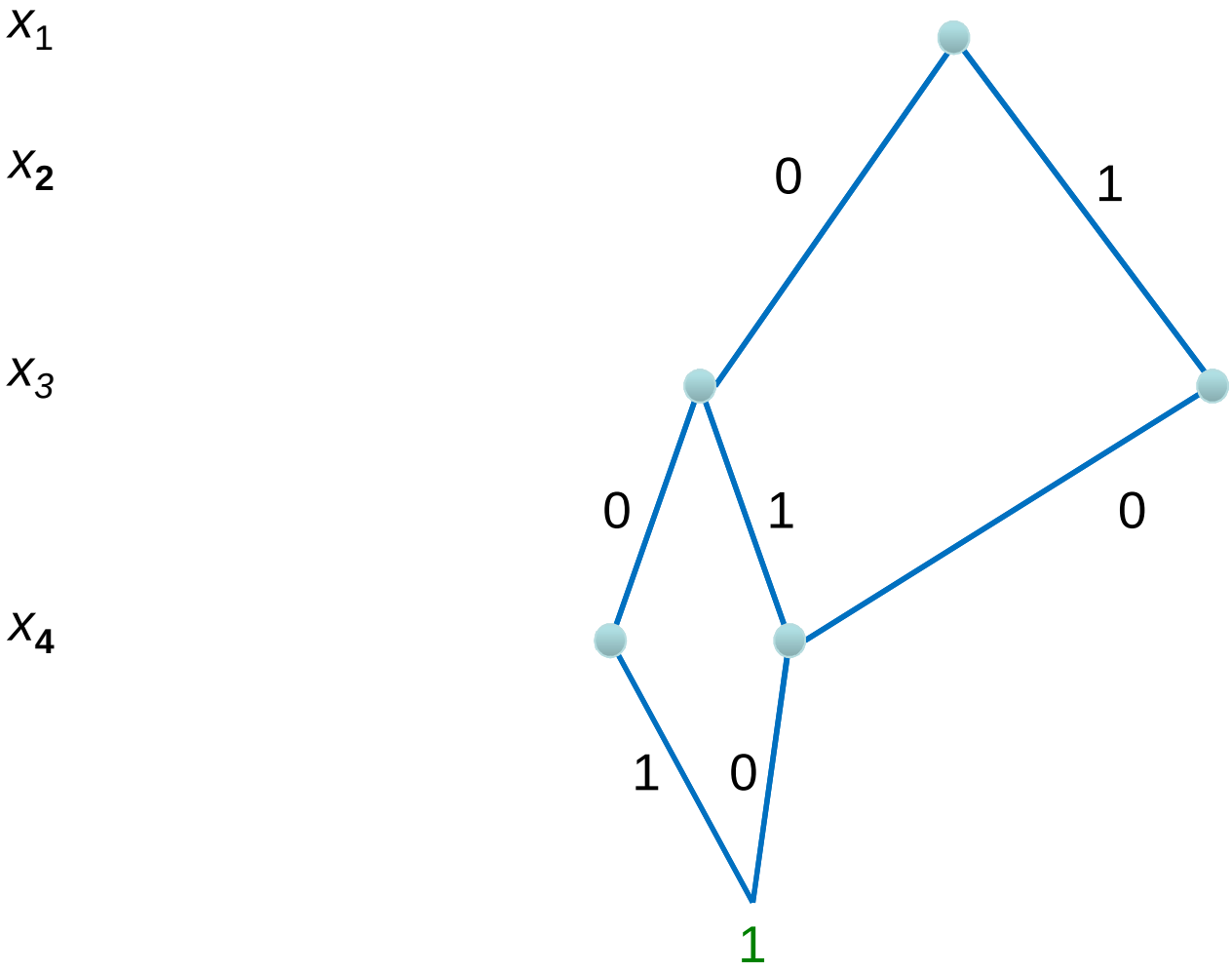
Merge isomorphic subtrees

$\text{among}(\{x_1, x_3, x_4\}, \{1\}, 2, 2)$



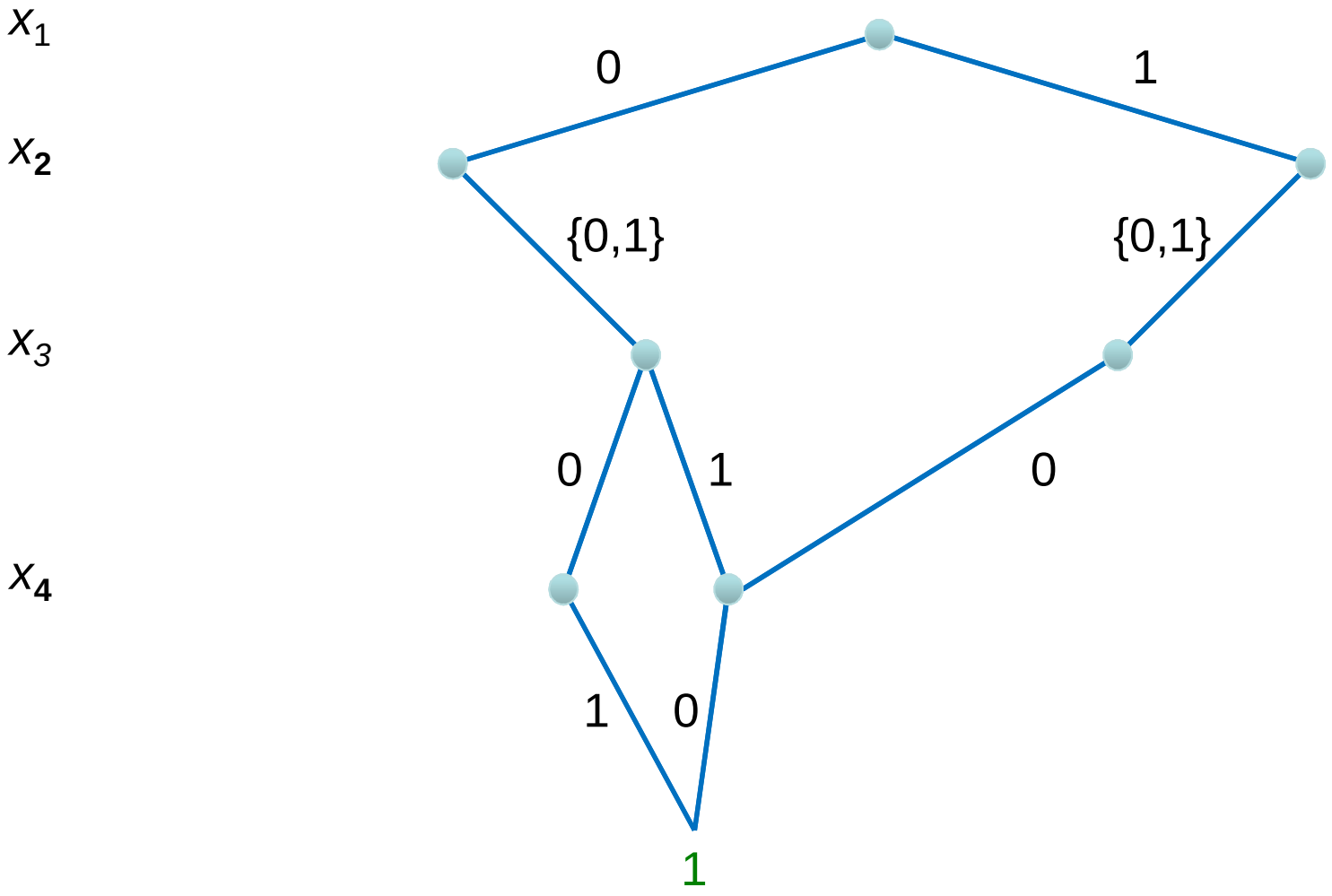
Can also remove redundant nodes

$$\text{among}(\{x_1, x_3, x_4\}, \{1\}, 2, 2)$$



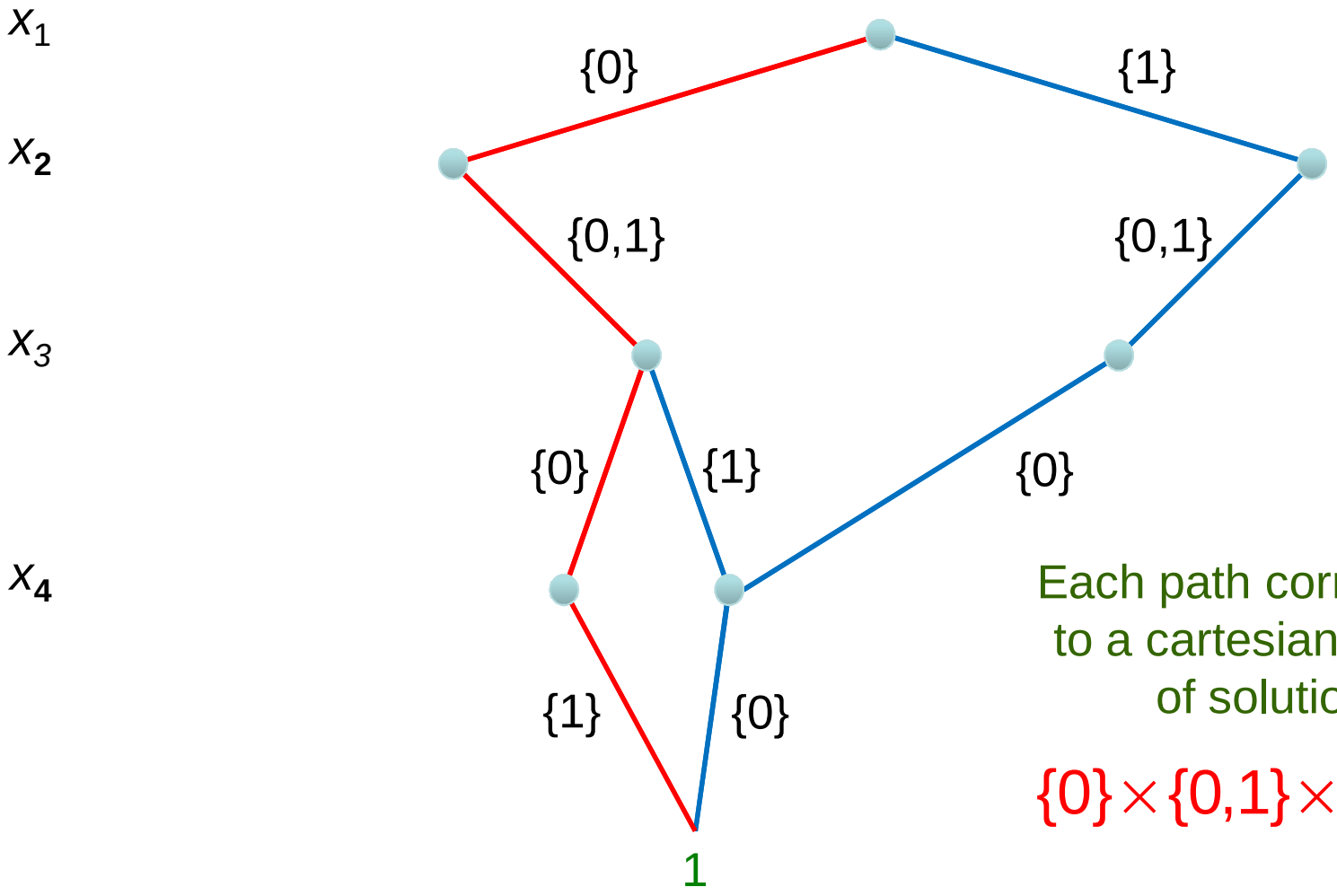
Can also remove redundant nodes

among$(\{x_1, x_3, x_4\}, \{1\}, 2, 2)$



But this is unnecessary.

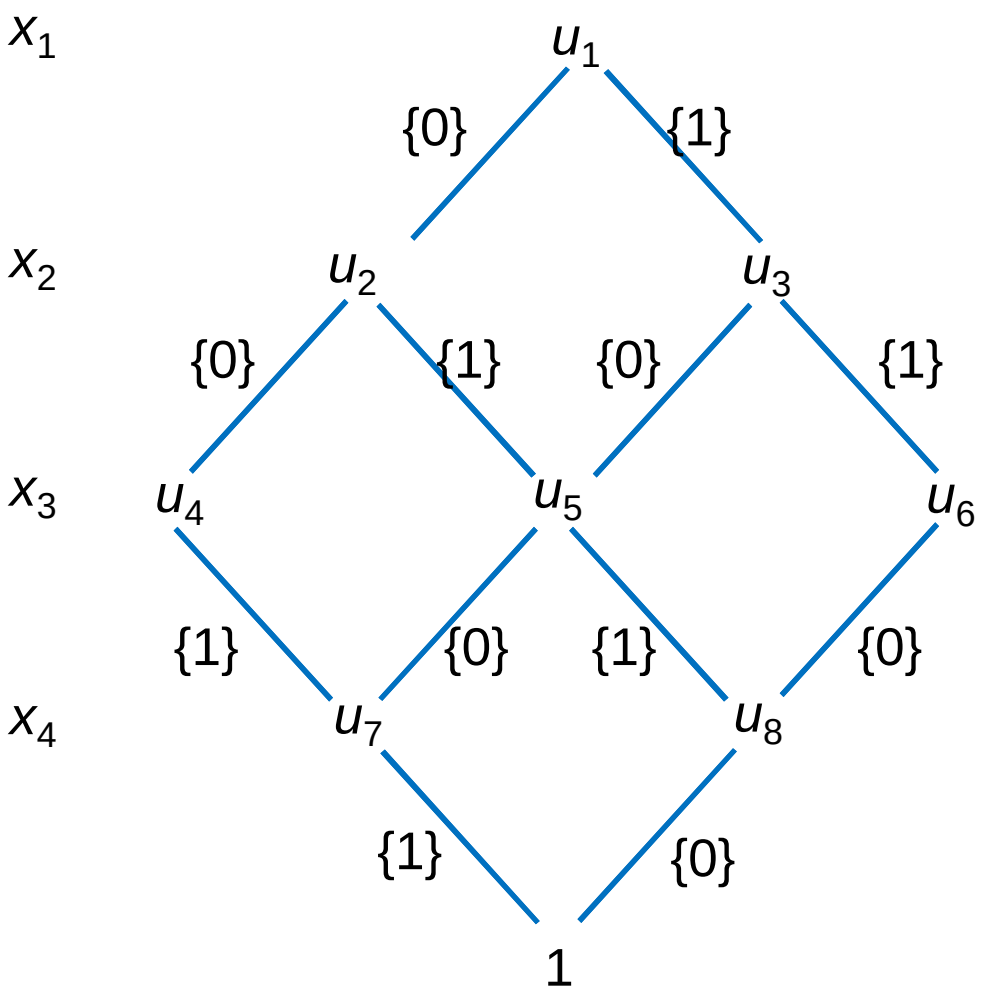
$\text{among}(\{x_1, x_3, x_4\}, \{1\}, 2, 2)$



Each path corresponds to a cartesian product of solutions.

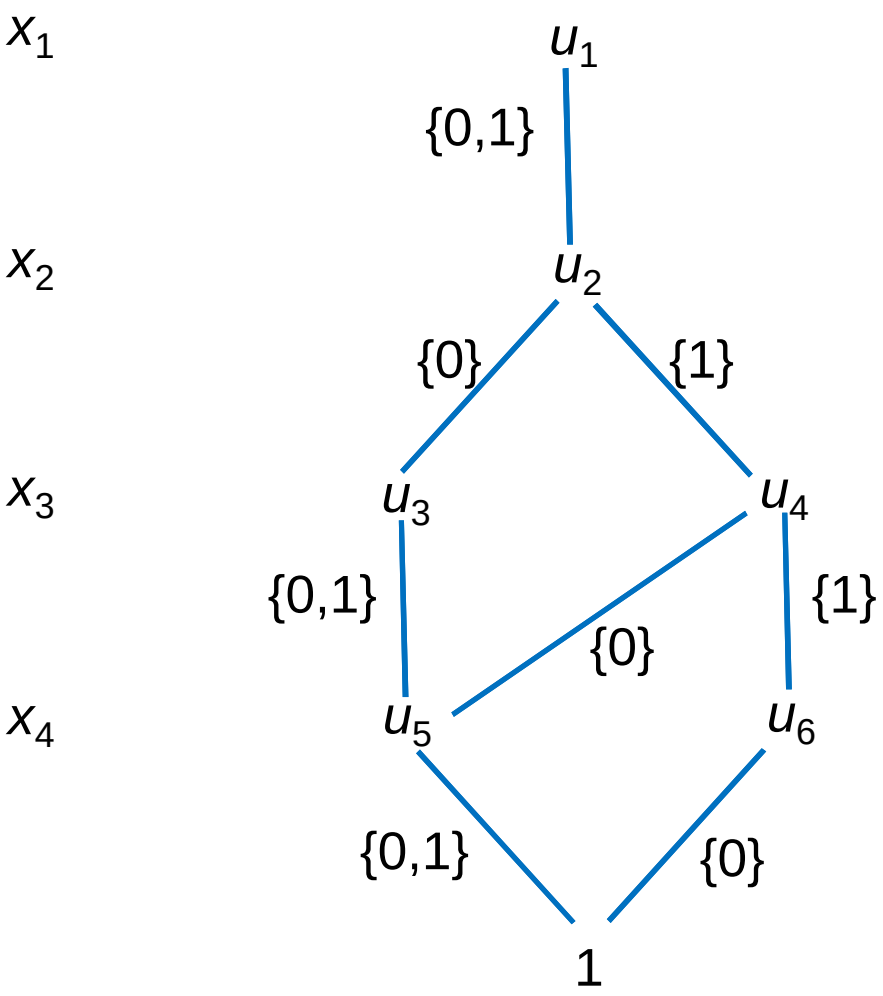
$\{0\} \times \{0,1\} \times \{0\} \times \{1\}$

New constraint: $\text{among}(\{x_1, x_2, x_3, x_4\}, \{1\}, 2, 2)$



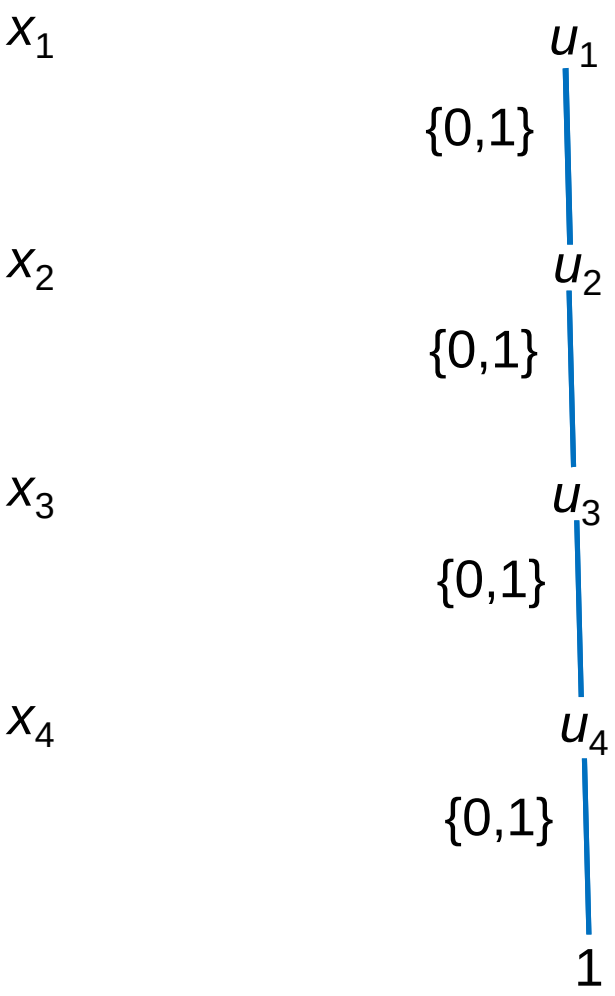
Exact BDD representation: 6 solutions

$$\text{among}(\{x_1, x_2, x_3, x_4\}, \{1\}, 2, 2)$$



Relaxed BDD of width 2: 14 solutions

$$\text{among}(\{x_1, x_2, x_3, x_4\}, \{1\}, 2, 2)$$

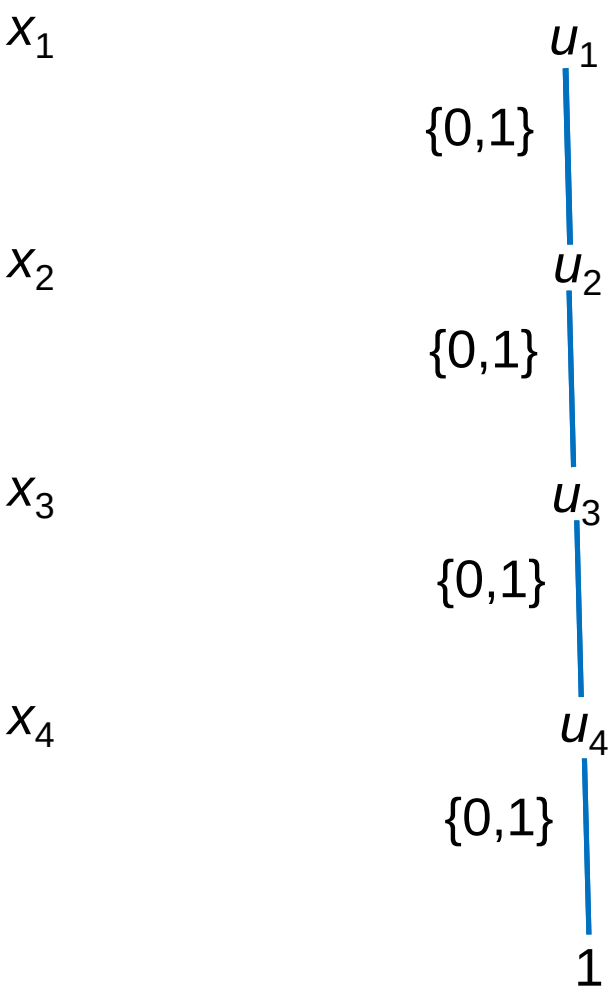


Relaxed BDD of width 1 is just the domain store

Propagation through a BDD

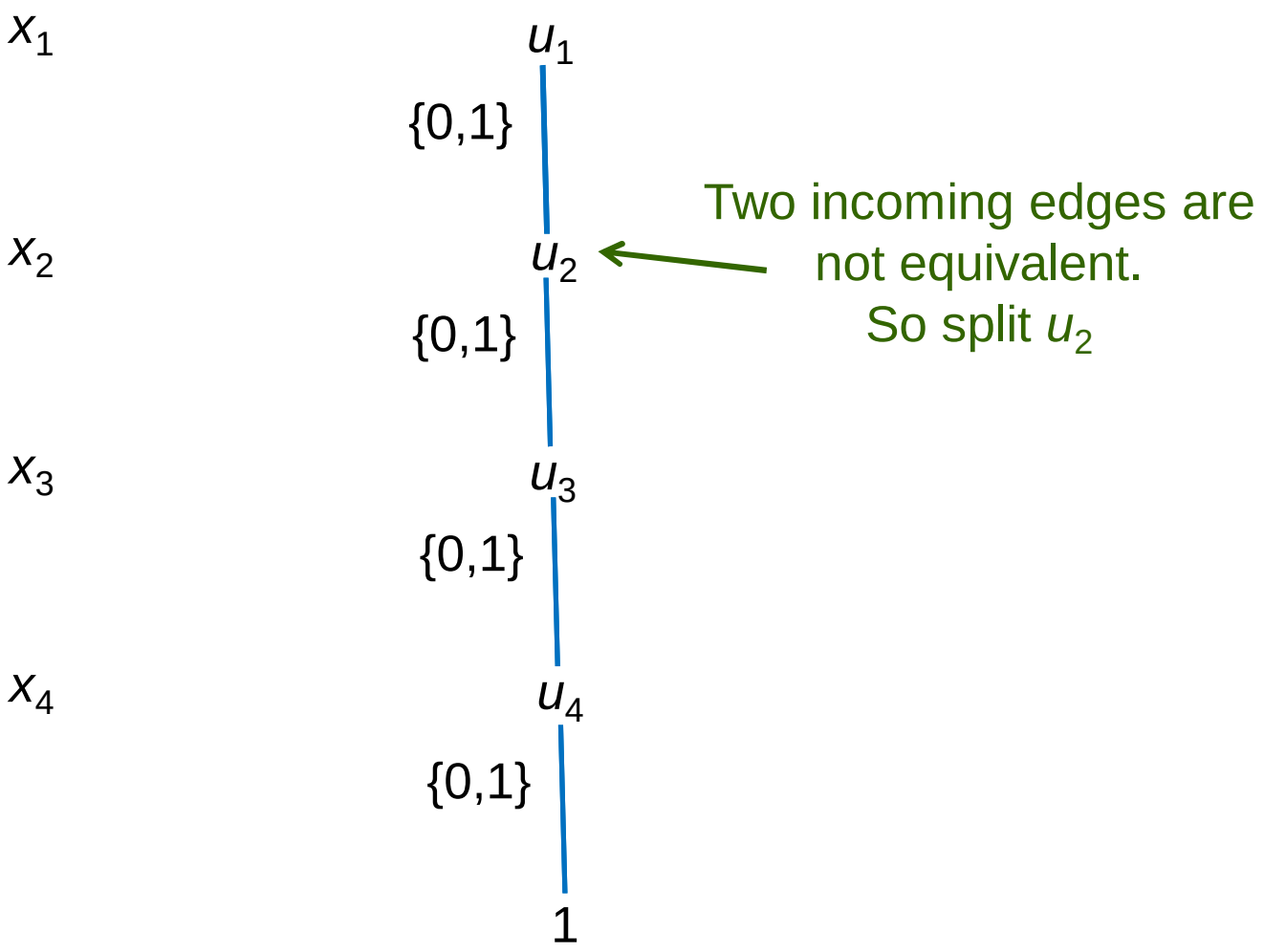
- At each node of the search tree:
  - Apply each constraint to current BDD.
- Refine BDD to incorporate relaxation of the constraint.
  - Refine by splitting nodes, subject to max width.
- Width limit keeps BDD size within bounds.
  - Greater width yields tighter relaxation.

Propagation of **among** through a BDD



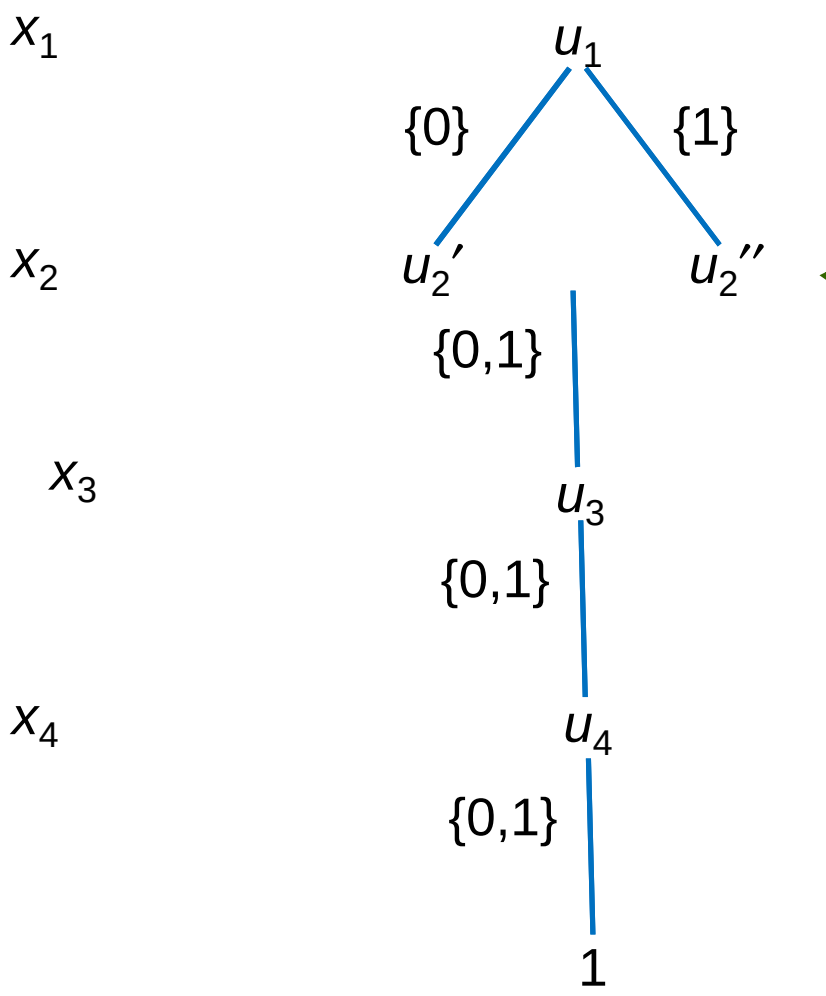
Arbitrarily start with BDD of width 1 and refine it

$$\text{among}(\{x_1, x_2, x_3, x_4\}, \{1\}, 2, 2)$$



Use a maximum width of 3

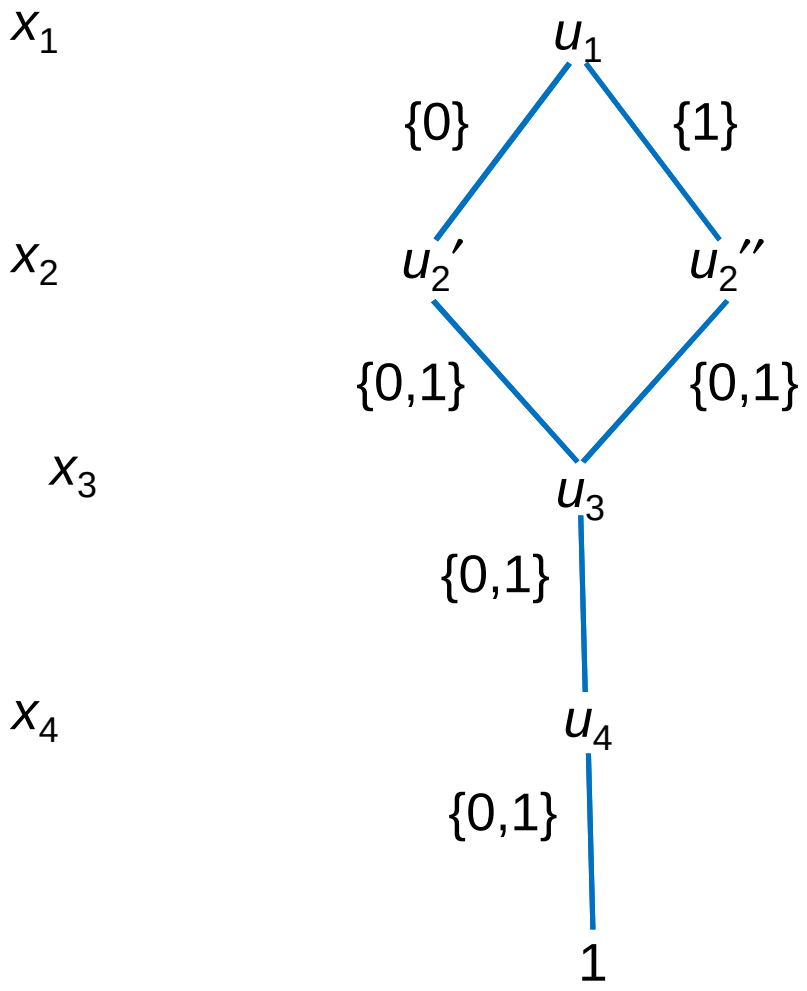
$$\text{among}(\{x_1, x_2, x_3, x_4\}, \{1\}, 2, 2)$$



Two incoming paths are not equivalent. So split u_2

Use a maximum width of 3

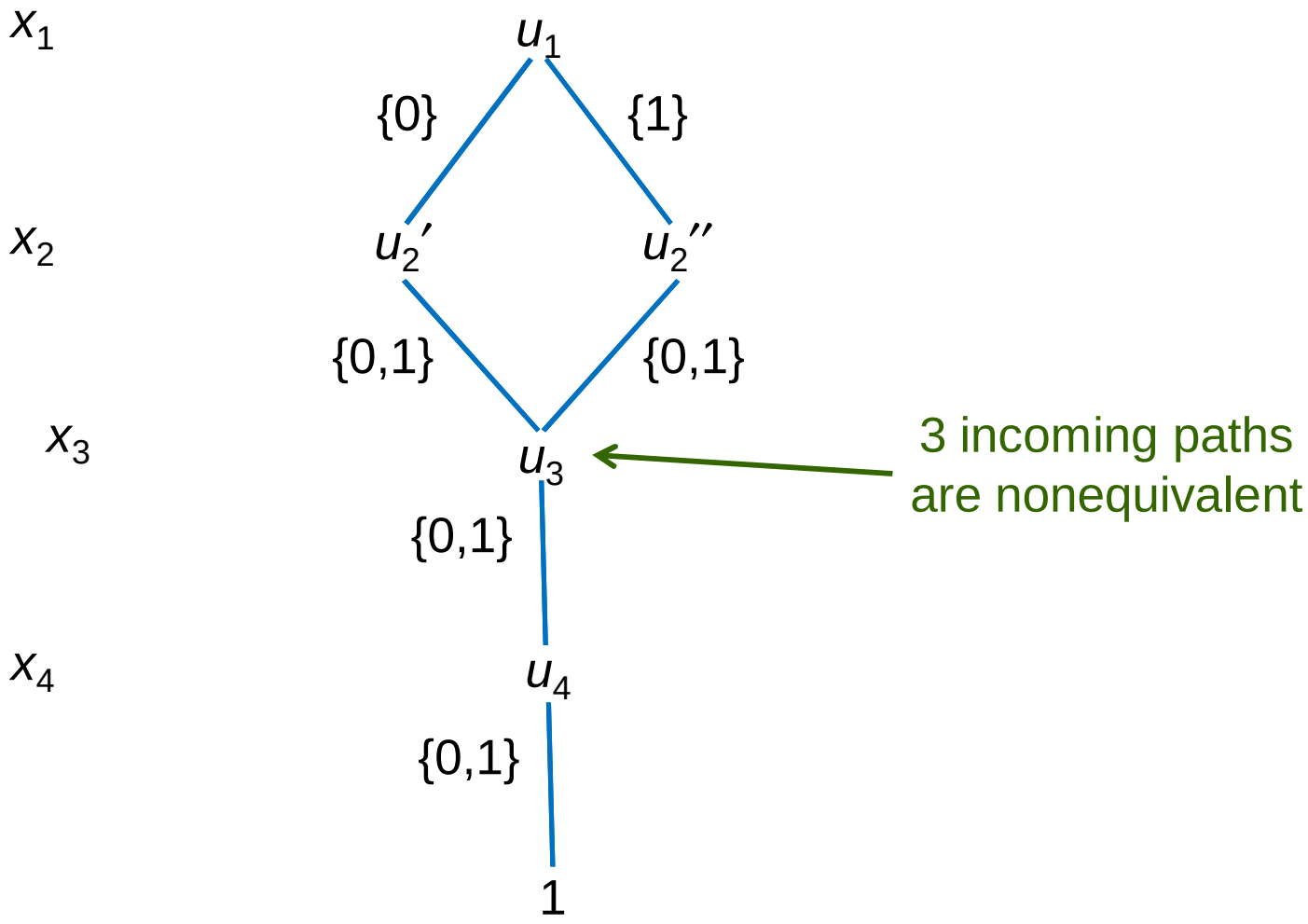
$$\text{among}(\{x_1, x_2, x_3, x_4\}, \{1\}, 2, 2)$$



Duplicate outgoing edges

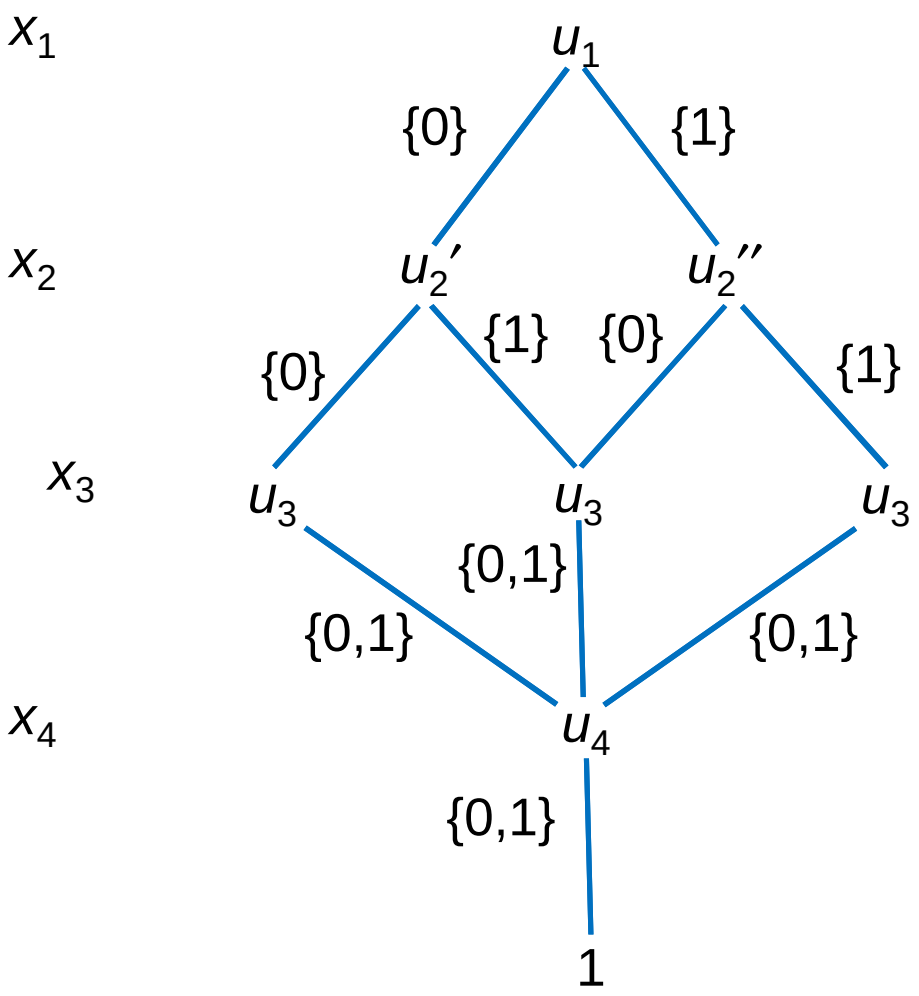
Use a maximum width of 3

$$\text{among}(\{x_1, x_2, x_3, x_4\}, \{1\}, 2, 2)$$



Use a maximum width of 3

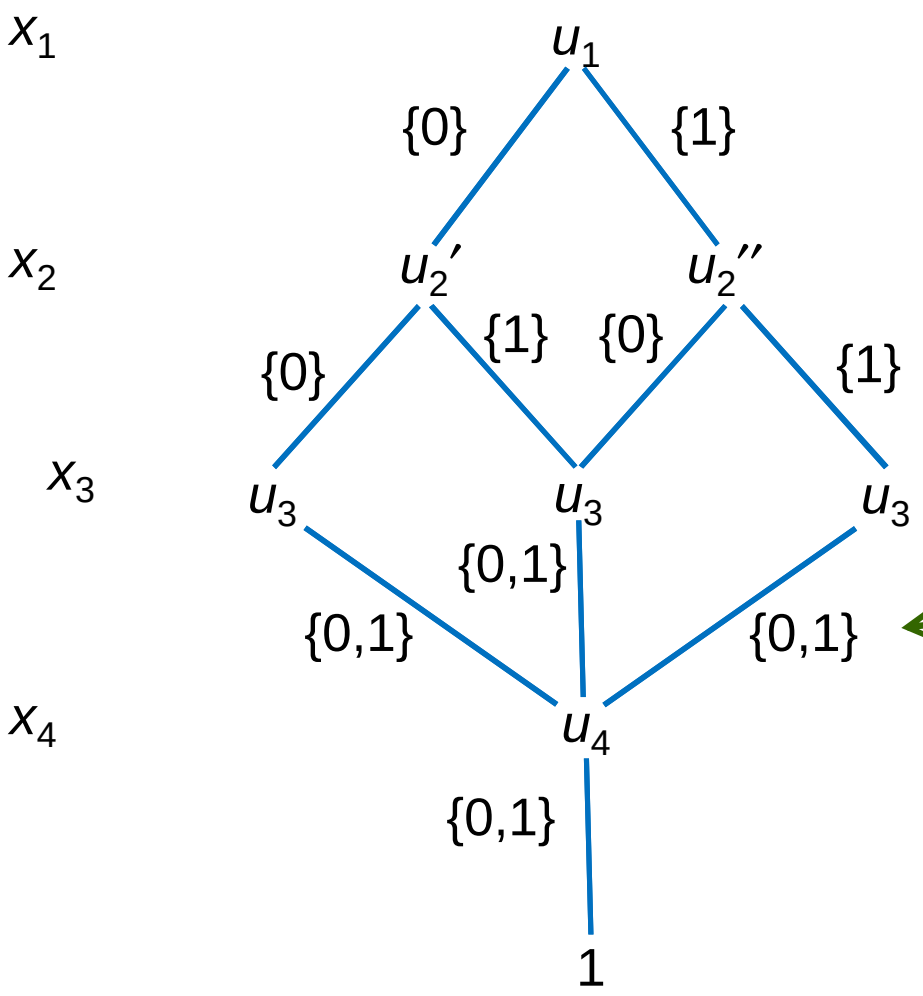
$$\text{among}(\{x_1, x_2, x_3, x_4\}, \{1\}, 2, 2)$$



Split into 3

Use a maximum width of 3

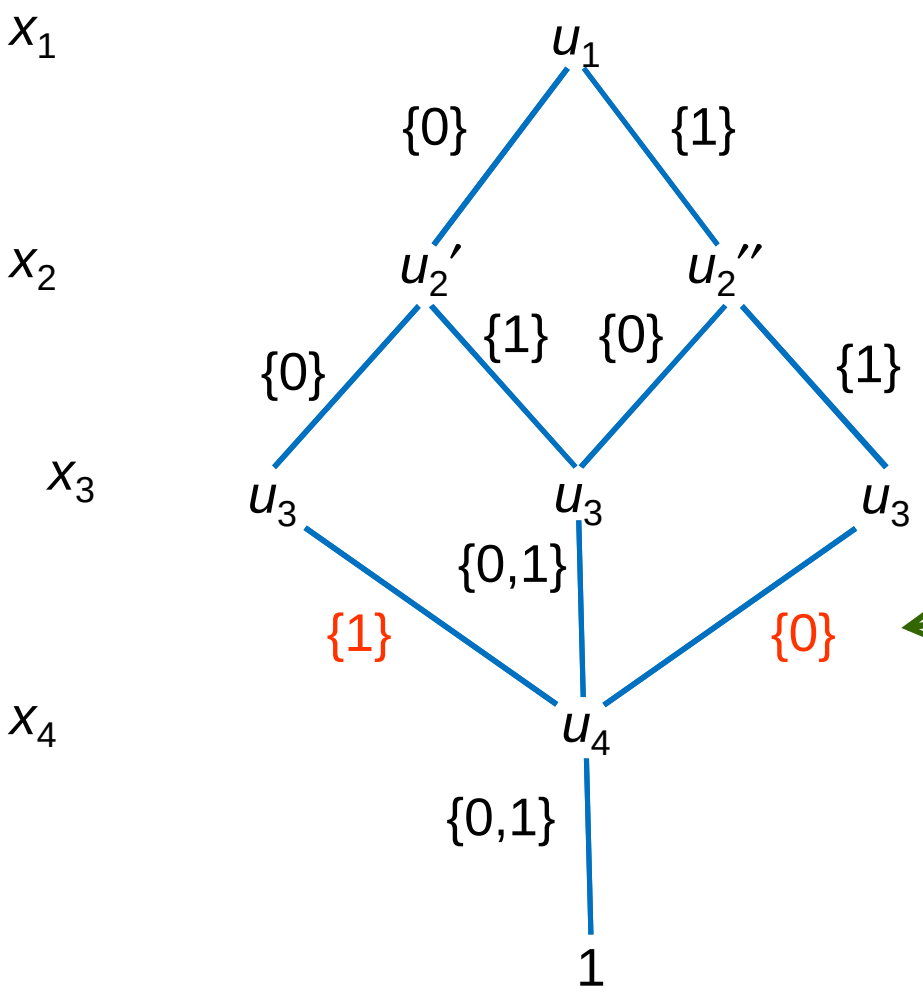
$$\text{among}(\{x_1, x_2, x_3, x_4\}, \{1\}, 2, 2)$$



Filter edge domains

Use a maximum width of 3

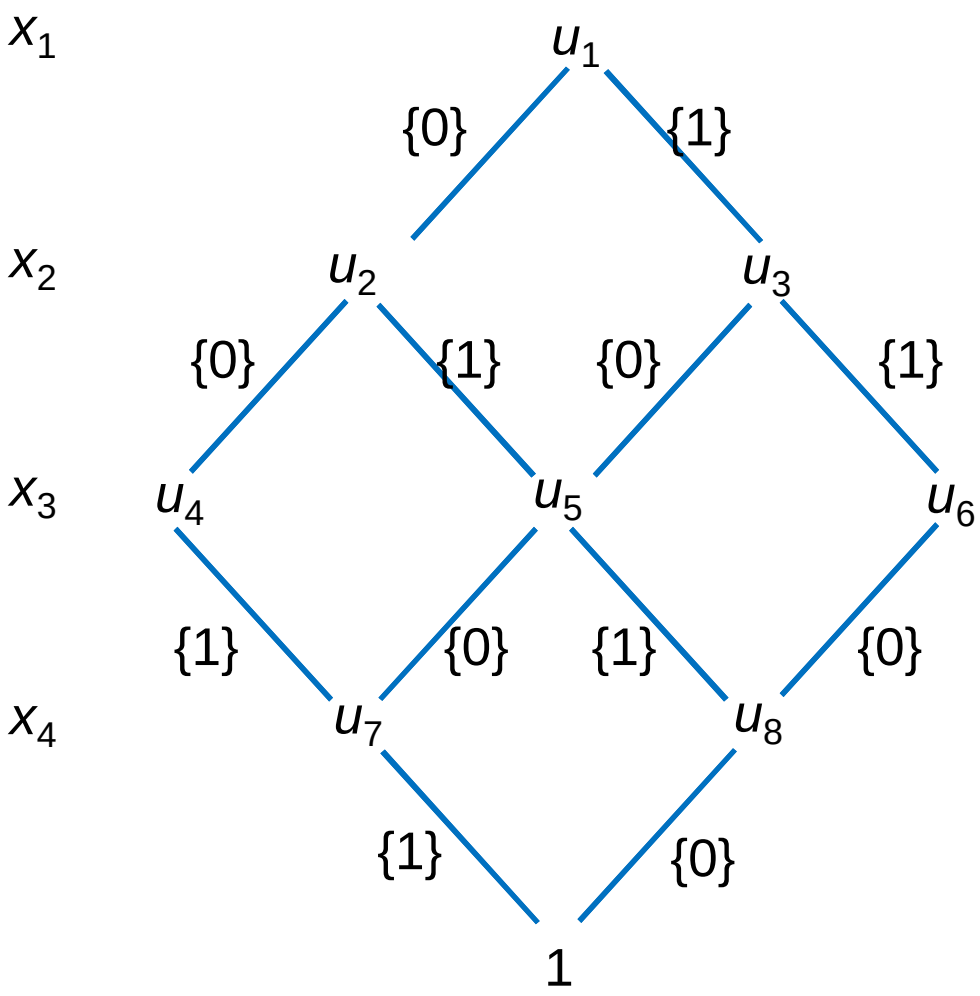
$$\text{among}(\{x_1, x_2, x_3, x_4\}, \{1\}, 2, 2)$$



Filter edge domains

Use a maximum width of 3

$\text{among}(\{x_1, x_2, x_3, x_4\}, \{1\}, 2, 2)$



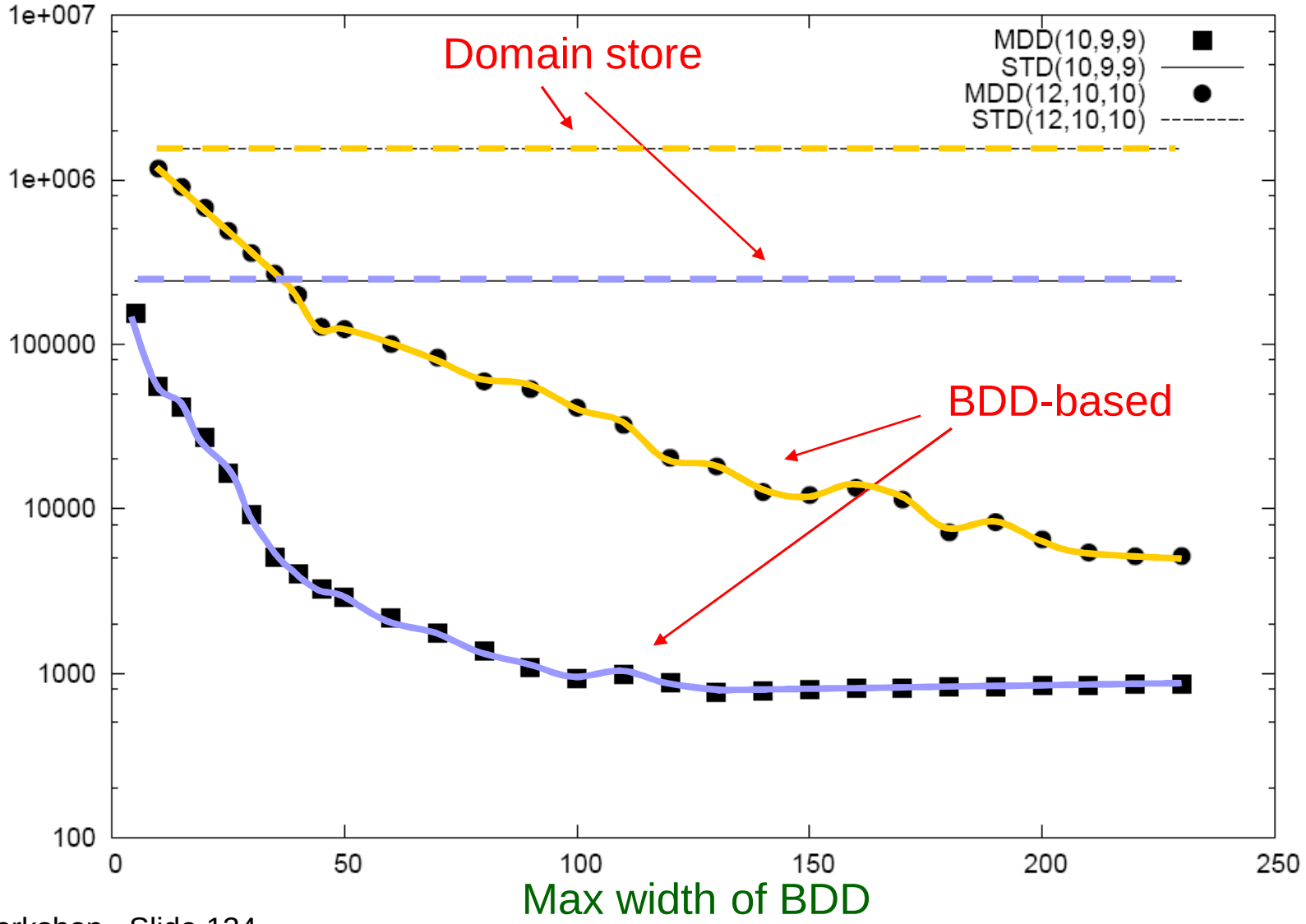
Another split generates exact BDD, which has width ≤ 3

Use a maximum width of 3

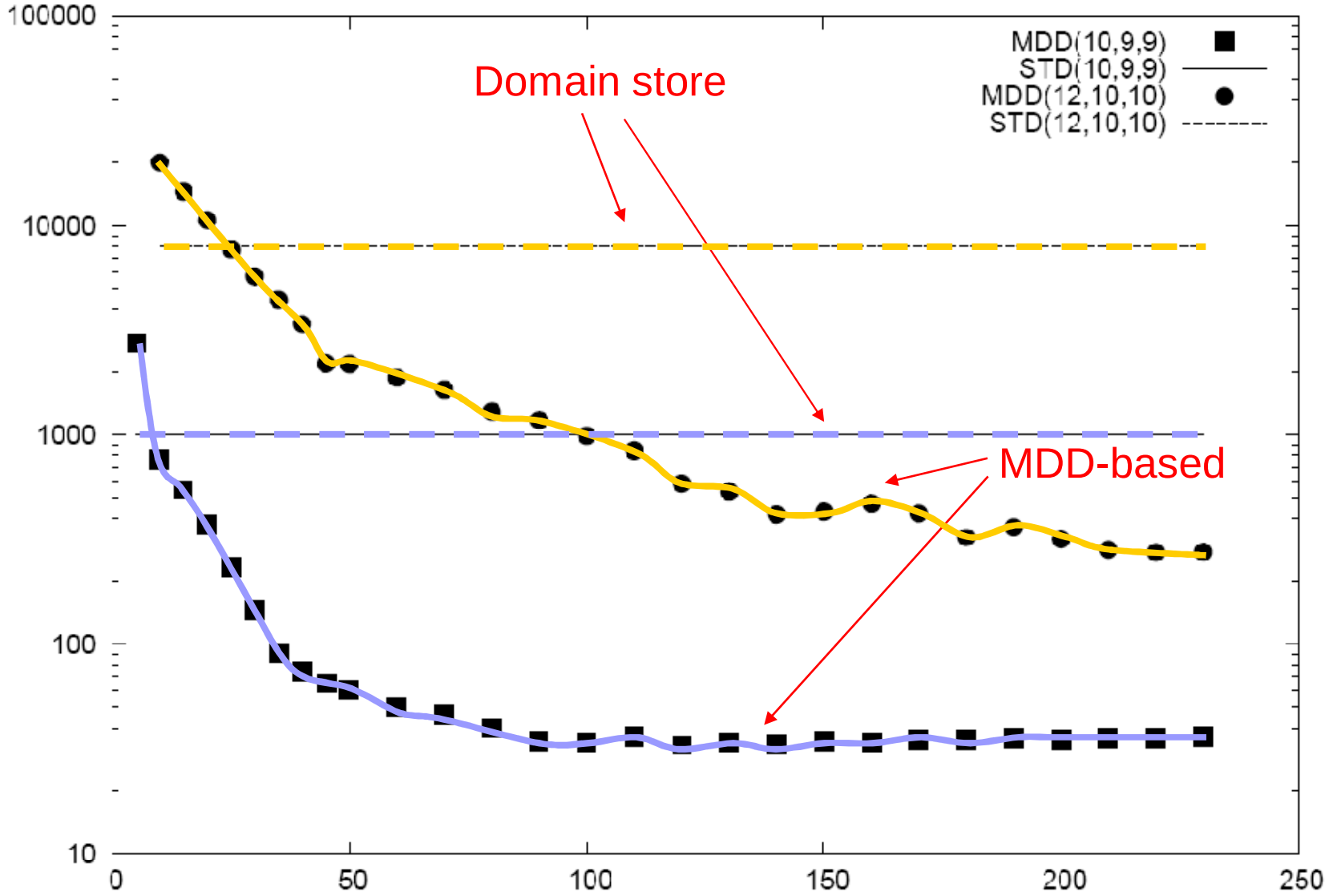
Computational results

- Multiple all-different constraints
  - Multivalued variables encoded as binary variables
- Using domain propagation only:
  - Some instances has search tree of > 1 million nodes
- Using BDD relaxation:
  - All instances solved at the root node.

Number of branches + number of splits



Computation time (milliseconds)

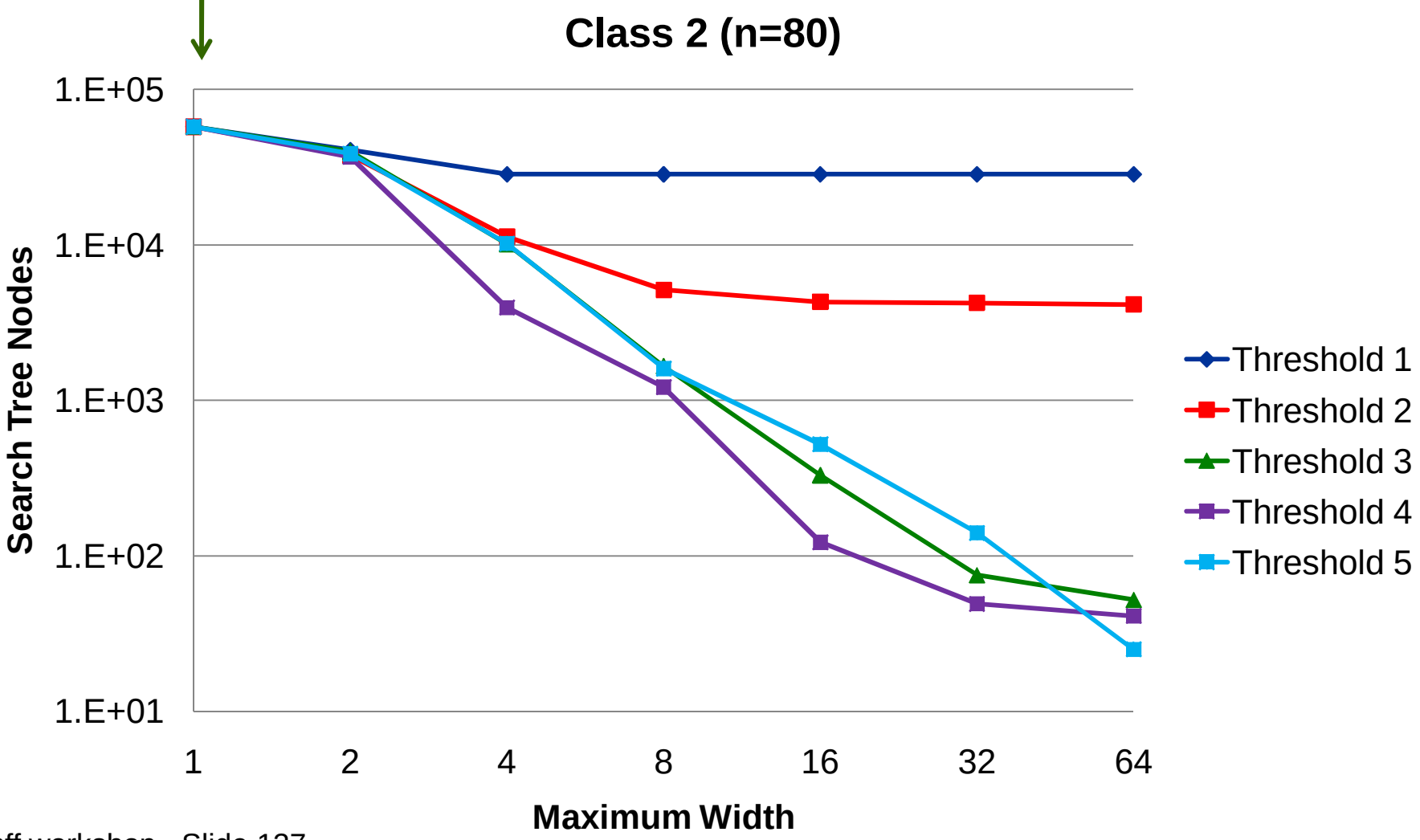


Computational results

- Multiple among constraints
  - Nurse scheduling problems
- When checking for equivalence of incoming paths…
  - Paths are viewed as nonequivalent if distance measure exceeds a threshold
- Results shown for finding first feasible solution
  - Results are very similar for finding all feasible solutions

Domain store only

Search tree nodes



Domain store only

Computation time