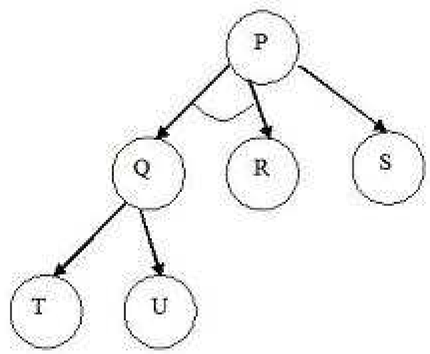
Multivalued Decision Diagrams and What They Can Do for You

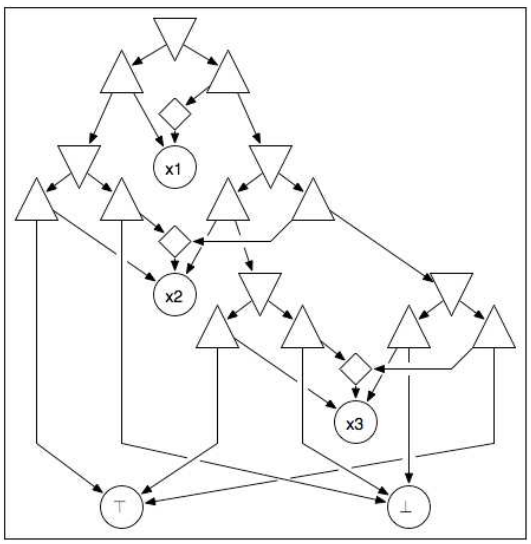
J. N. Hooker
Carnegie Mellon University

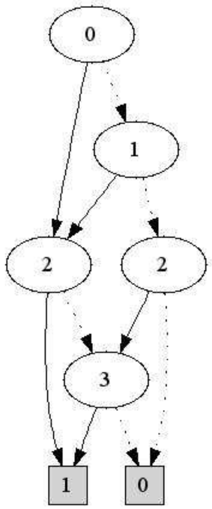
Workshop on Constraint Reasoning and Graphical Structures
CP 2010

Two Roles for Graphical Structures

- Problem-solving device

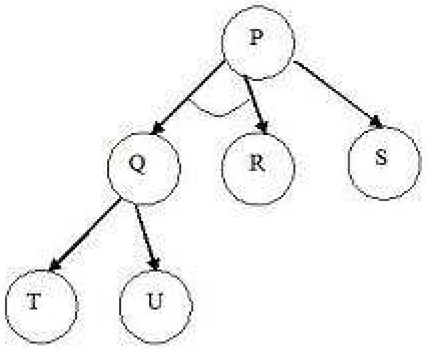


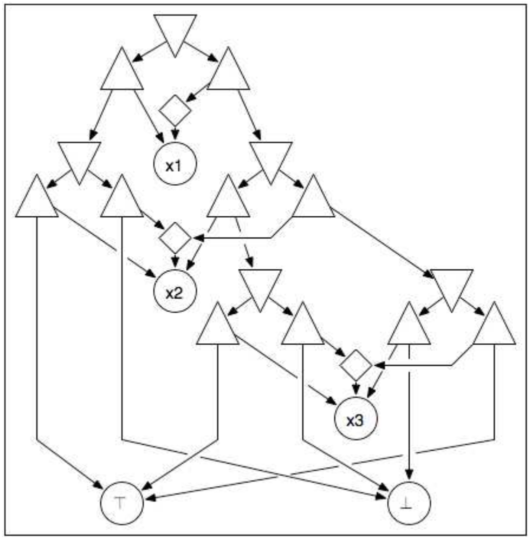


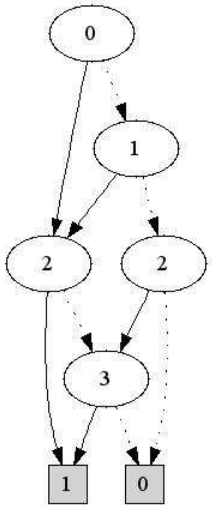


Two Roles for Graphical Structures

- Problem-solving device
- The **solution**.
  - Transform the problem to a more **transparent** data structure..

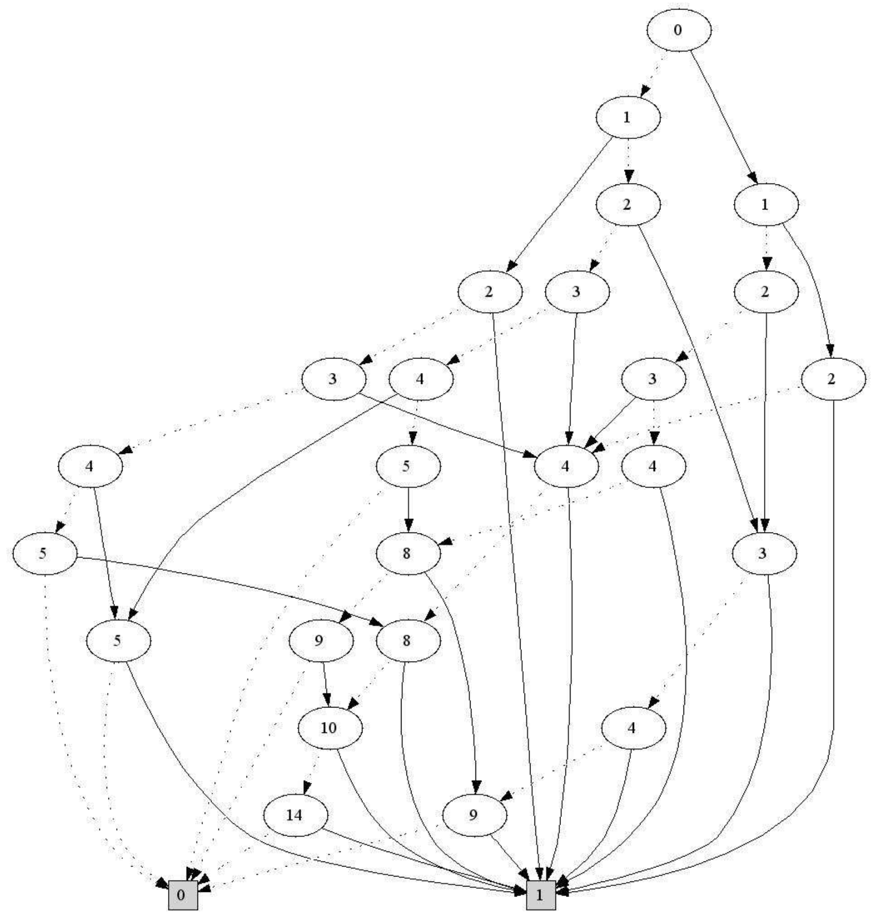






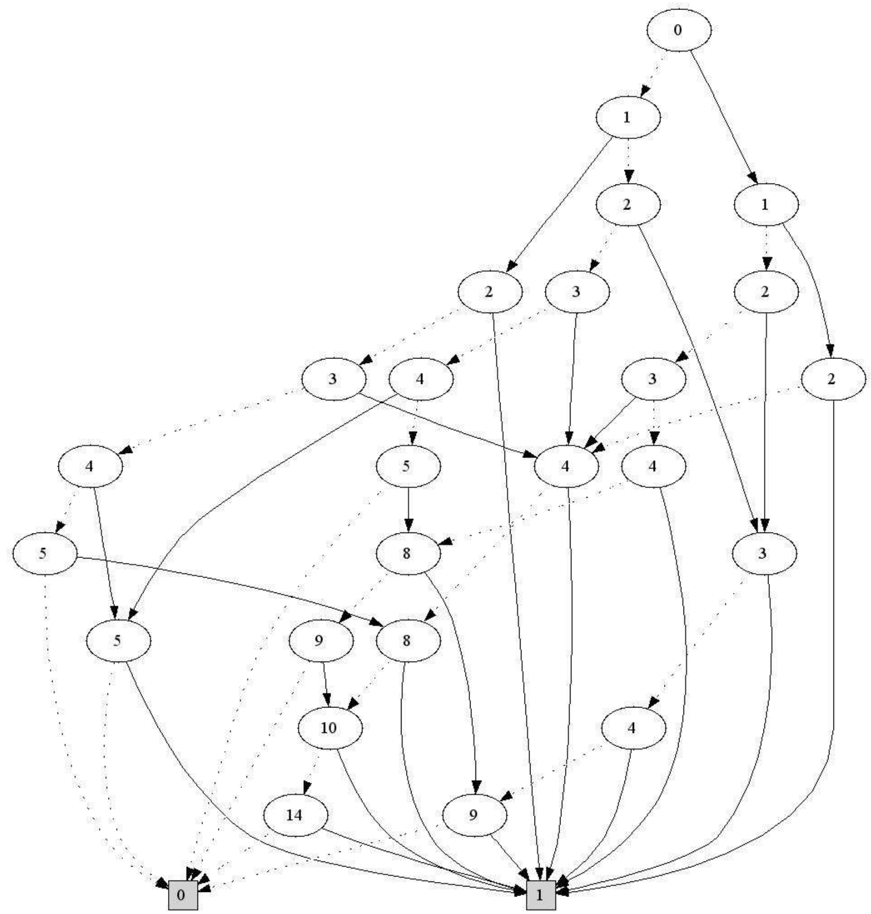
MDDs

- MDDs can play both roles.
  - An aid to constraint solving and optimization
  - A transparent representation of the solution space



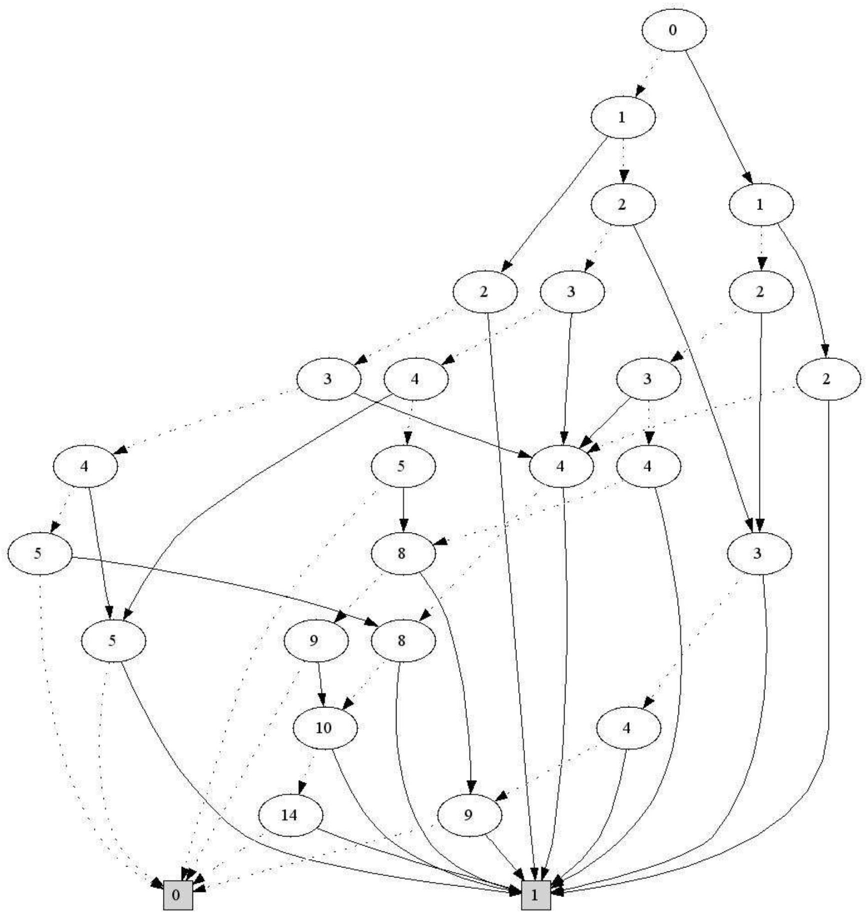
MDDs

- MDDs can play both roles.
  - An aid to constraint solving and optimization
  - A transparent representation of the solution space
- This is a conceptual overview.
  - By no means complete.



Outline

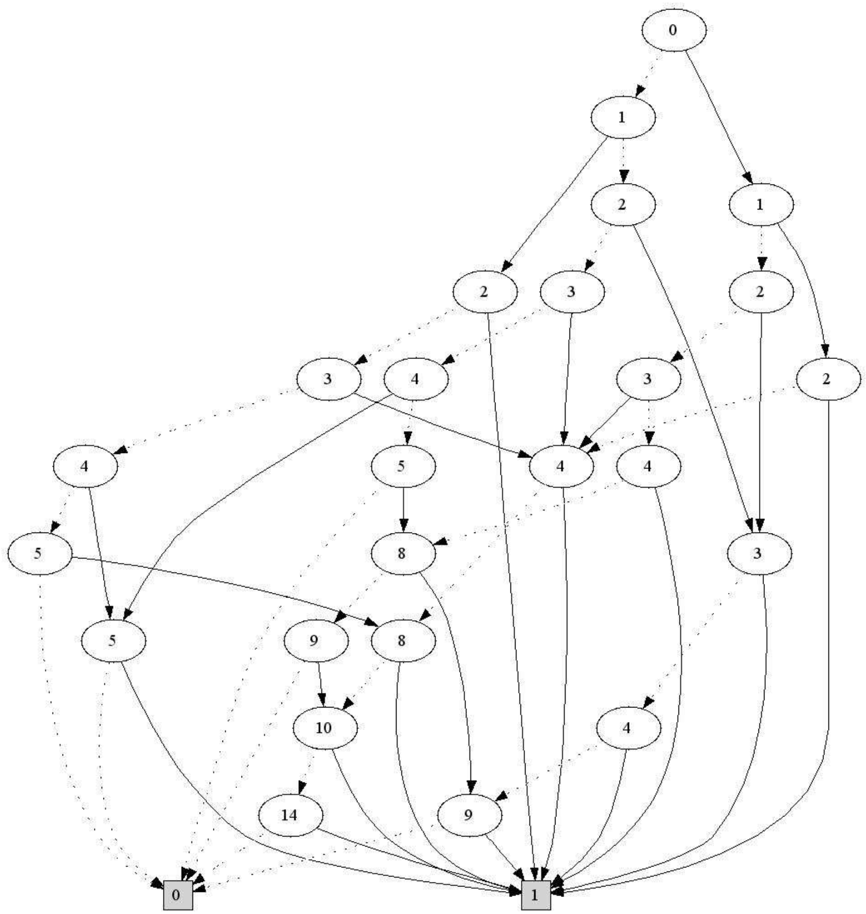
- Introduction to MDDs
- MDDs as Propagators
- MDDs as Relaxations
- MDDs as Restrictions
- MDDs as Transparent Data Structures
- Cost-Bounded MDDs
- Nonserial MDDs and Dynamic Programming



Introduction to MDDs

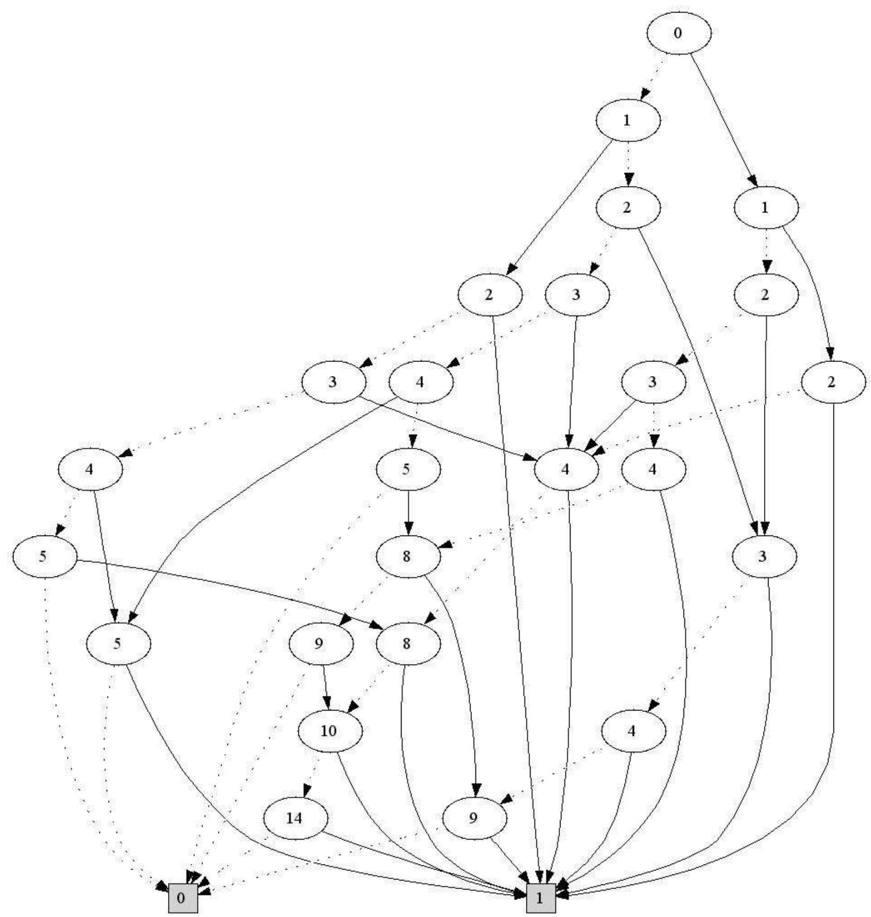
BDDs and MDDs

- A **binary decision diagram** (BDD) represents a boolean function $f(x_1, \ldots, x_n)$.
  - With binary x_j.



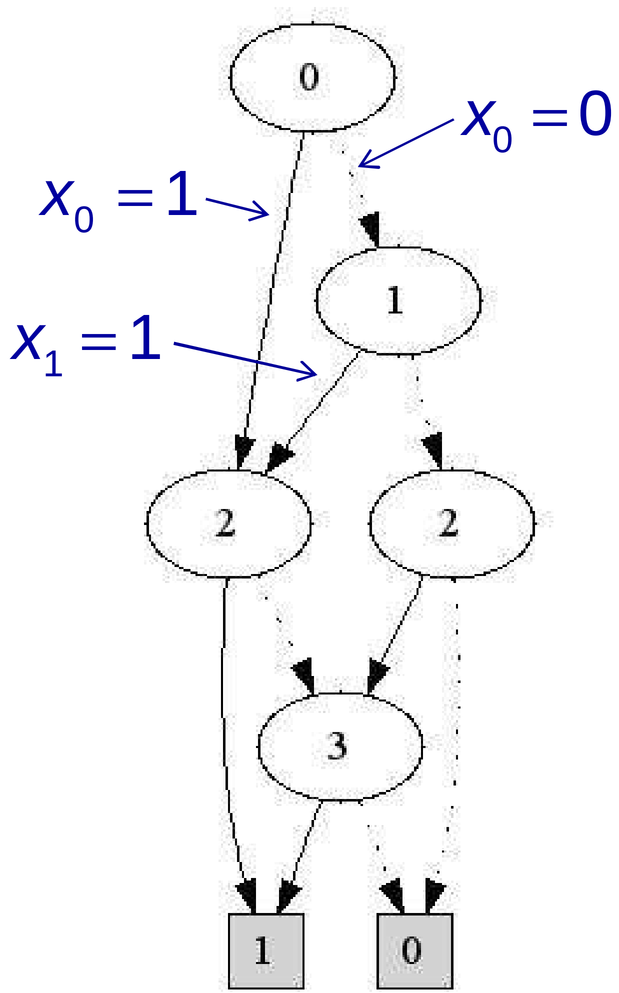
BDDs and MDDs

- A **binary decision diagram** (BDD) represents a boolean function $f(x_1, \ldots, x_n)$.
  - With binary x_j.
- **Multivalued decision diagram** (MDD) .
  - General discrete x_j



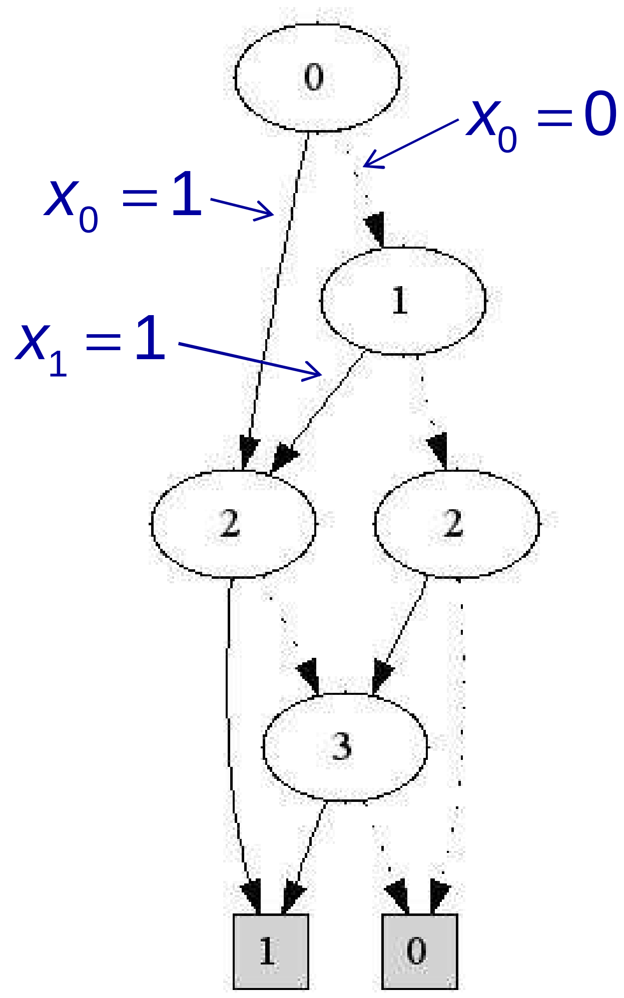
BDDs and MDDs

- Every path to 1 represents assignments to $x_0, \ldots, x_{n-1}$ for which $f(x_0, \ldots, x_{n-1}) = 1$.



BDDs and MDDs

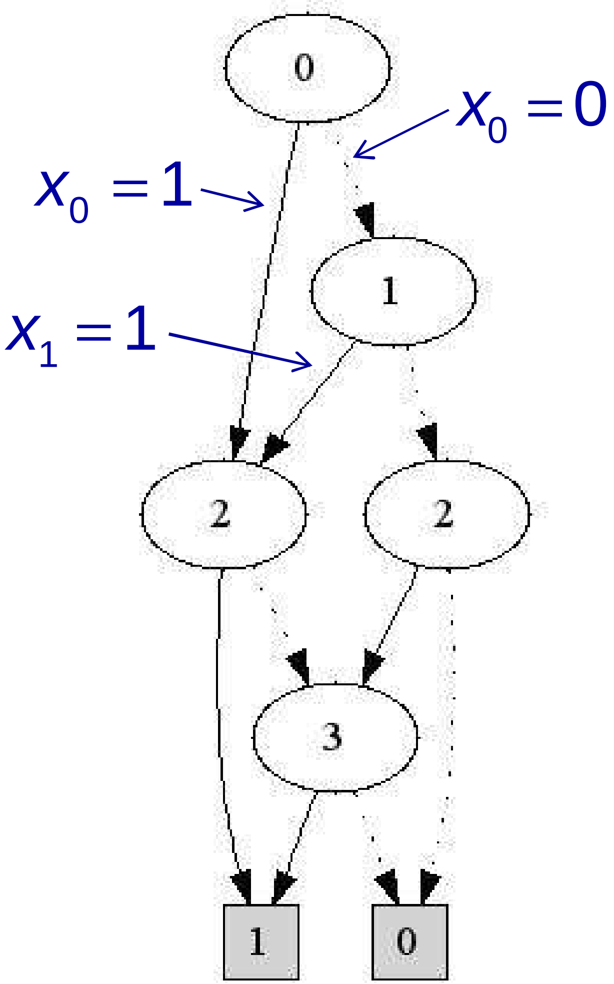
- Every path to 1 represents assignments to $x_0, \ldots, x_{n-1}$ for which $f(x_0, \ldots, x_{n-1}) = 1$.
  - Can be viewed as feasible solutions of a constraint.



BDDs and MDDs

- Every path to 1 represents assignments to $x_0, \ldots, x_{n-1}$ for which $f(x_0, \ldots, x_{n-1}) = 1$.
  - Can be viewed as feasible solutions of a constraint.
  - In this case,

    $$2x_0 + 3x_1 + 5x_2 + 5x_3 \geq 7$$

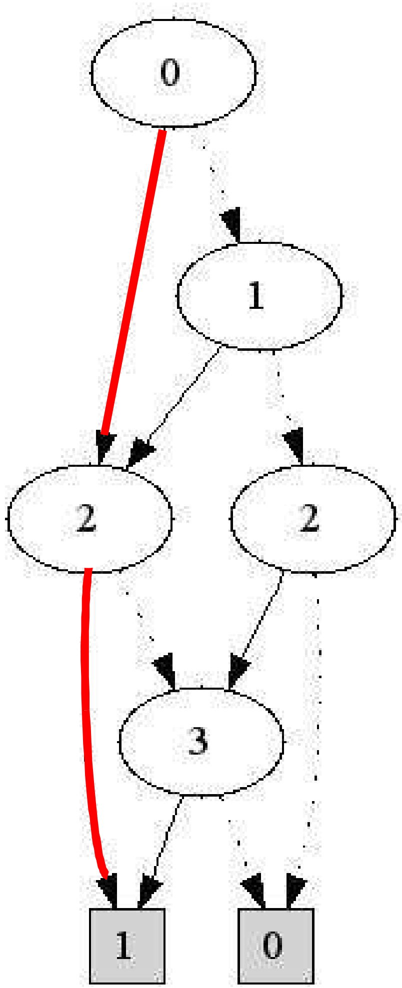


BDDs and MDDs

- Every path to 1 represents assignments to $x_0, \ldots, x_{n-1}$ for which $f(x_0, \ldots, x_{n-1}) = 1$.
  - Can be viewed as feasible solutions of a constraint.
  - In this case,

    $$2x_0 + 3x_1 + 5x_2 + 5x_3 \geq 7$$

$x = (1,*,1,*)$

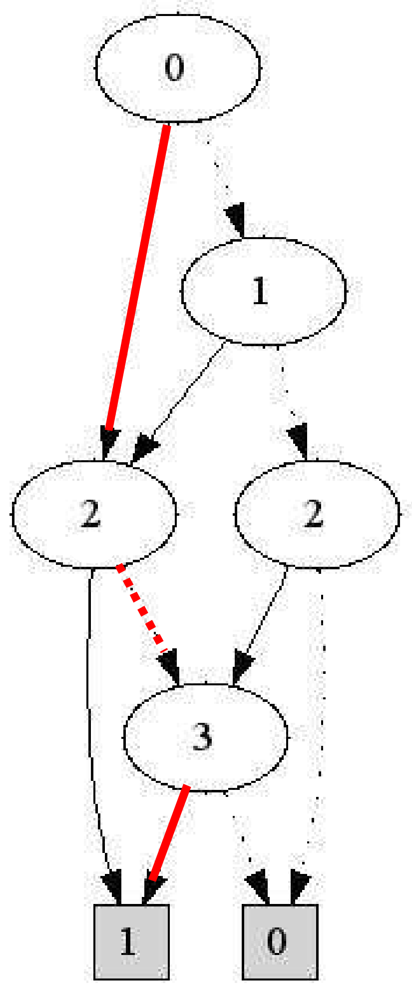


BDDs and MDDs

- Every path to 1 represents assignments to $x_0, \ldots, x_{n-1}$ for which $f(x_0, \ldots, x_{n-1}) = 1$.
  - Can be viewed as feasible solutions of a constraint.
  - In this case,

$$2x_0 + 3x_1 + 5x_2 + 5x_3 \geq 7$$

$x = (1,*,1,*), (1,*,0,1)$

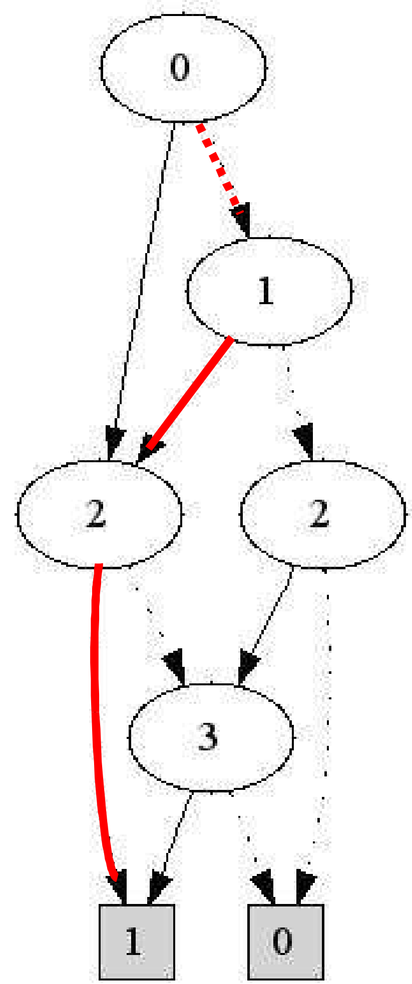


BDDs and MDDs

- Every path to 1 represents assignments to $x_0, \ldots, x_{n-1}$ for which $f(x_0, \ldots, x_{n-1}) = 1$.
  - Can be viewed as feasible solutions of a constraint.
  - In this case,

$$2x_0 + 3x_1 + 5x_2 + 5x_3 \geq 7$$

$x = (1,*,1,*), (1,*,0,1), (0,1,1,*)$

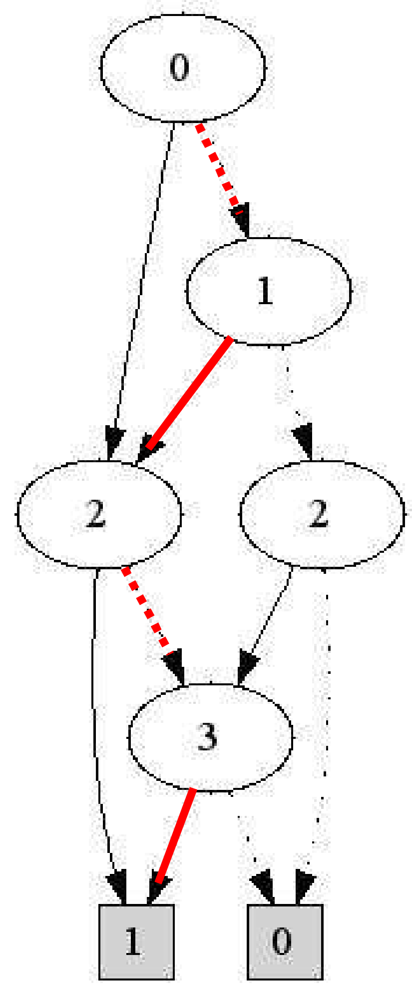


BDDs and MDDs

- Every path to 1 represents assignments to $x_0, \ldots, x_{n-1}$ for which $f(x_0, \ldots, x_{n-1}) = 1$.
  - Can be viewed as feasible solutions of a constraint.
  - In this case,

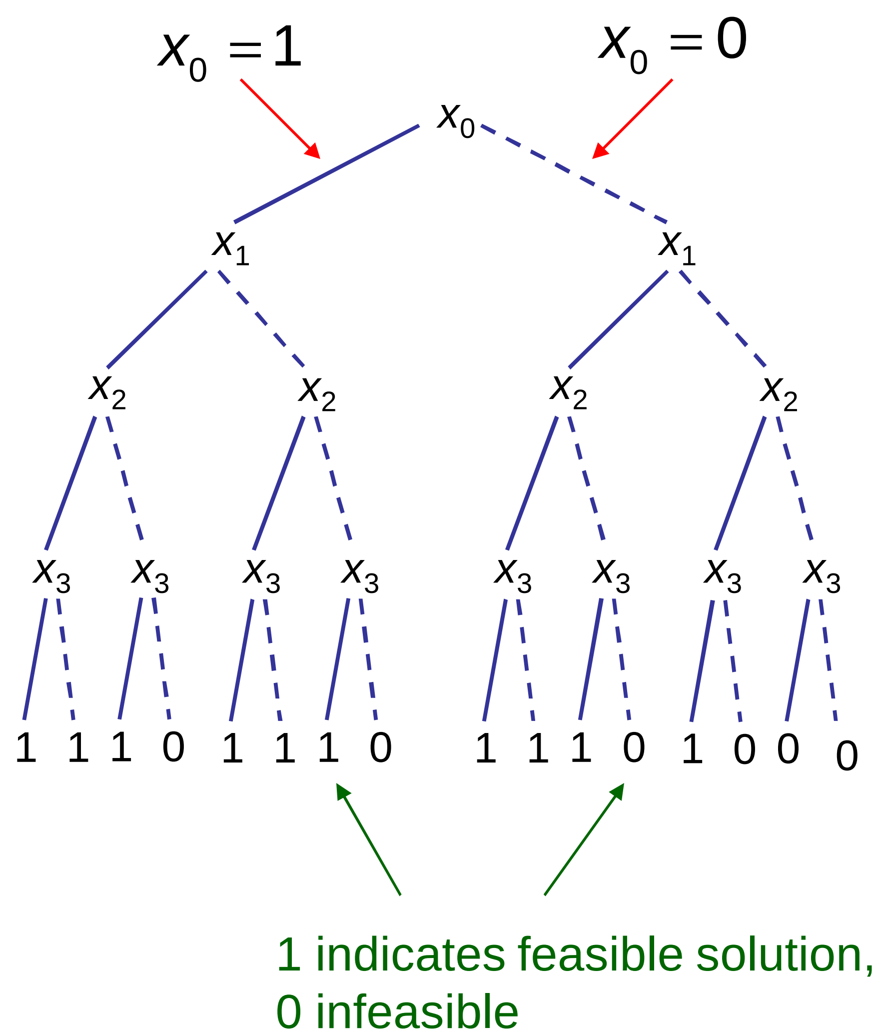
$$2x_0 + 3x_1 + 5x_2 + 5x_3 \geq 7$$

$x = (1,*,1,*), (1,*,0,1), (0,1,1,*), (0,1,0,1)$



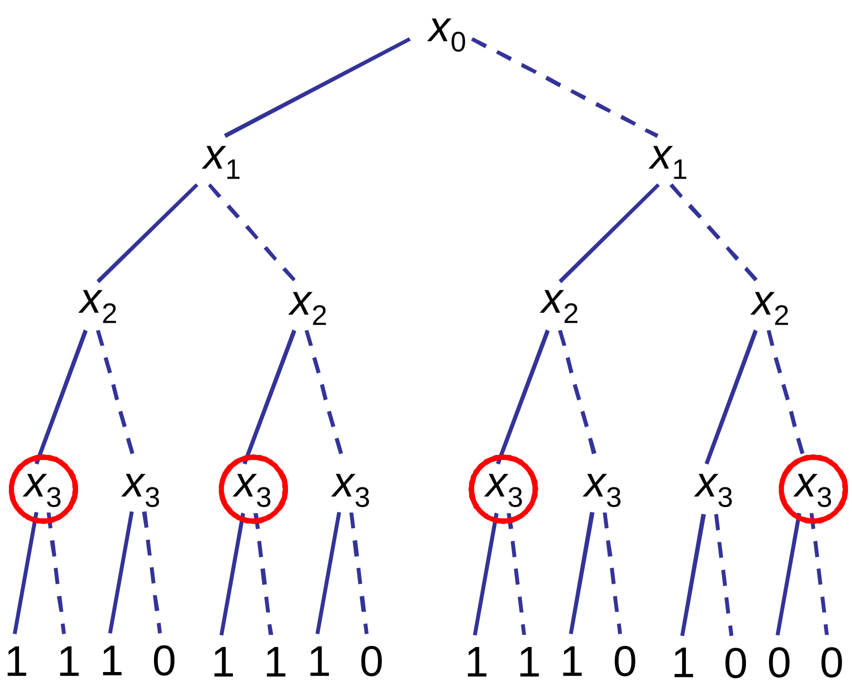
Reduced MDDs

- There is a **unique reduced** MDD for any given constraint.
  - Once the variable ordering is specified.
- The reduced MDD can be viewed as a branching tree with redundancy removed.
  - Superimpose isomorphic subtrees.
  - Remove redundant nodes.



Branching tree for 0-1 inequality

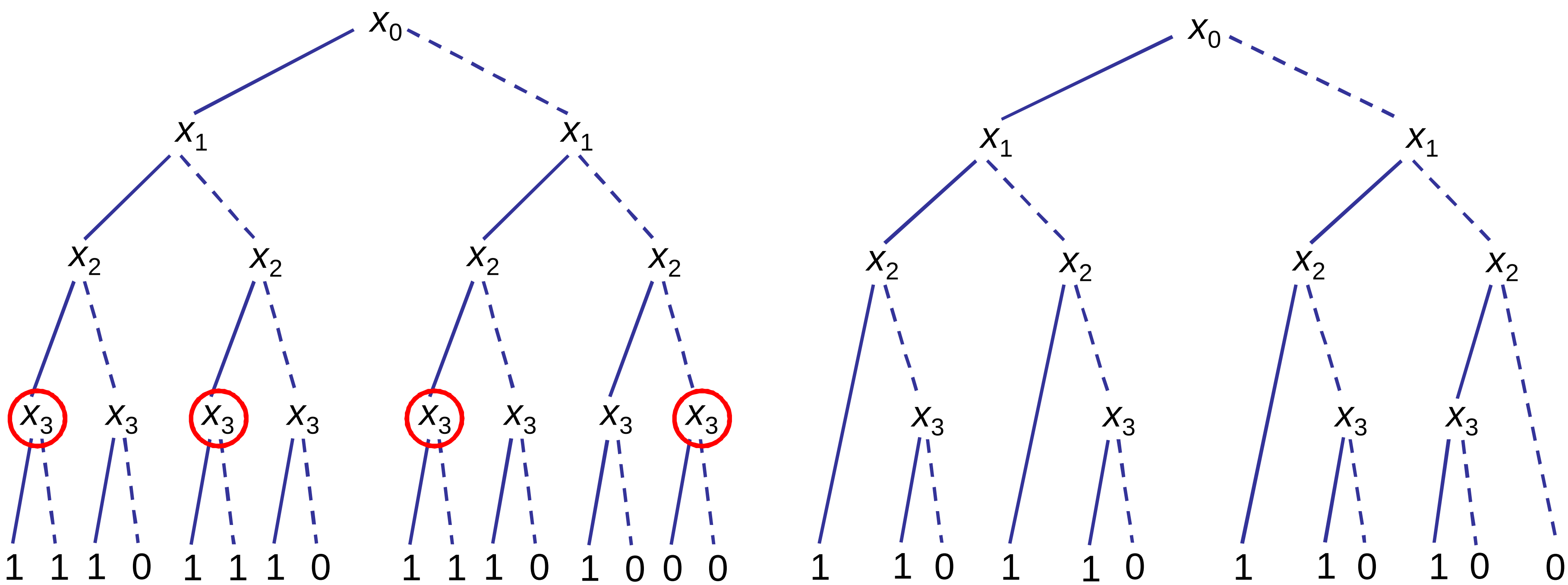
$$2x_0 + 3x_1 + 5x_2 + 5x_3 \geq 7$$



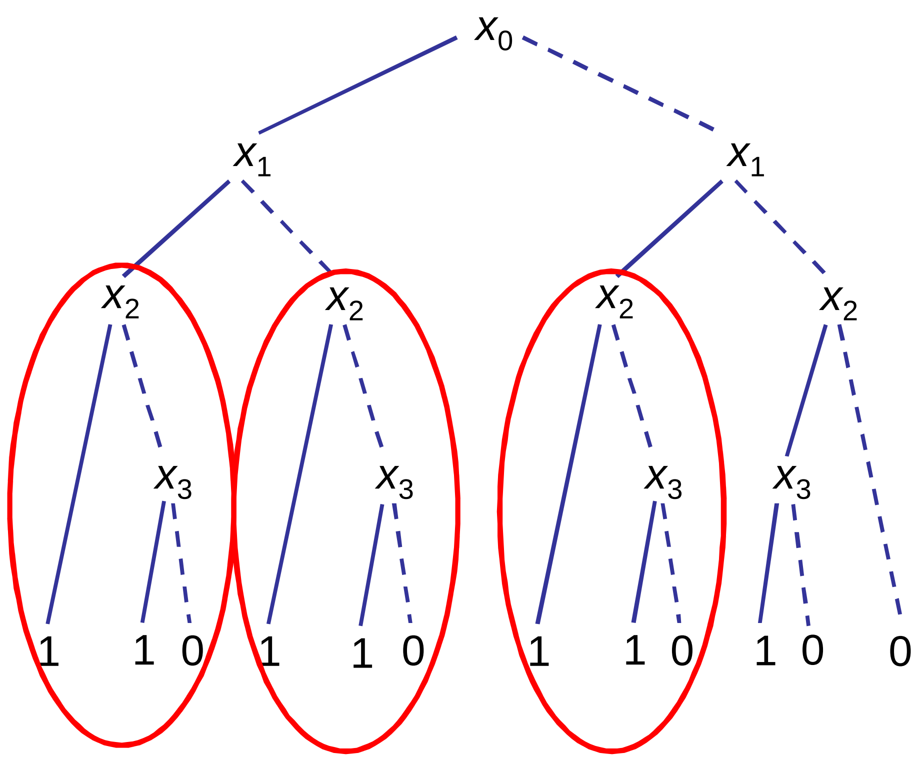
Branching tree for 0-1 inequality

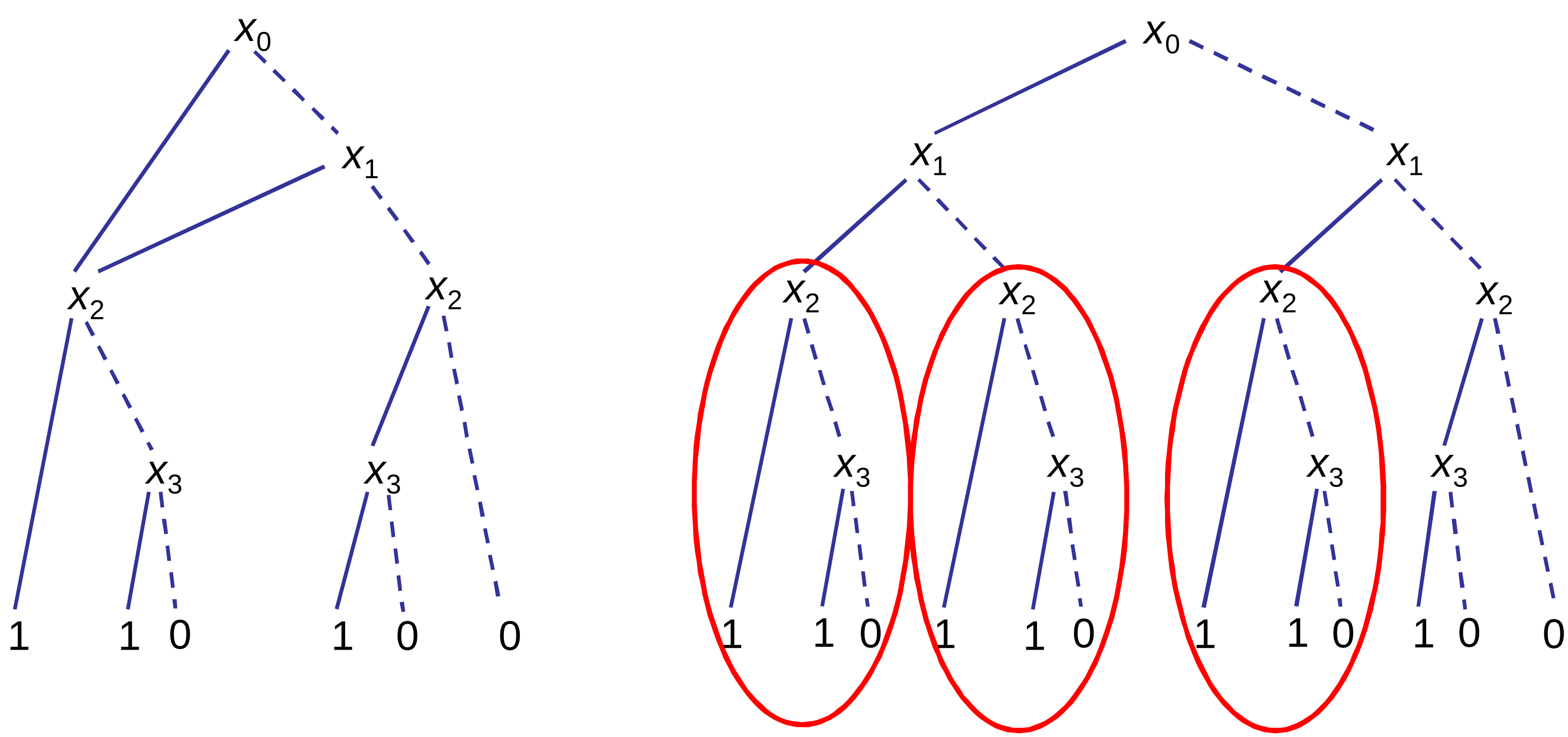
$$2x_0 + 3x_1 + 5x_2 + 5x_3 \geq 7$$

Remove redundant nodes…

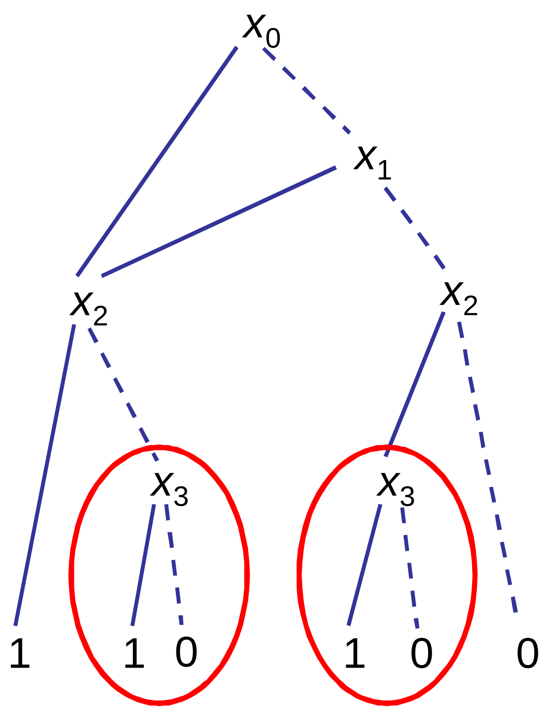

x_0
x_1 x_1
x_2 x_2 x_2 x_2
x_3 x_3 x_3 x_3 x_3 x_3 x_3 x_3
1 1 1 0 1 1 1 0 1 1 1 0 1 0 0 0
x_0
x_1 x_1
x_2 x_2 x_2 x_2
x_3 x_3 x_3 x_3
1 1 0 1 1 0 1 1 0 1 0 0

Superimpose identical subtrees…

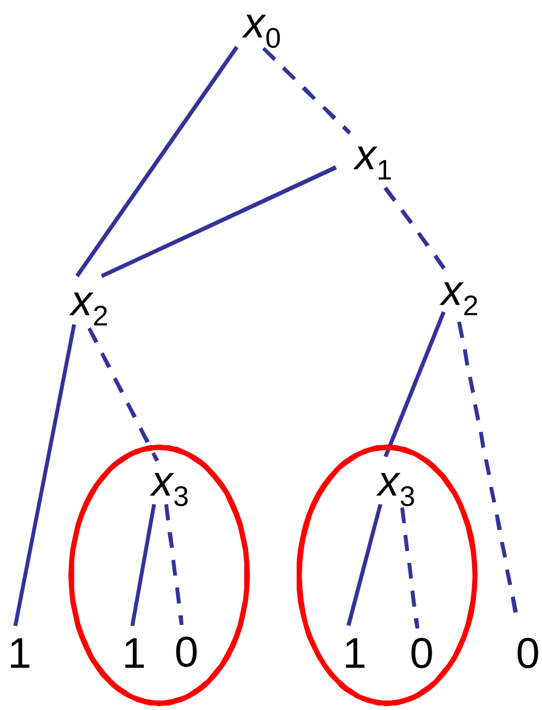


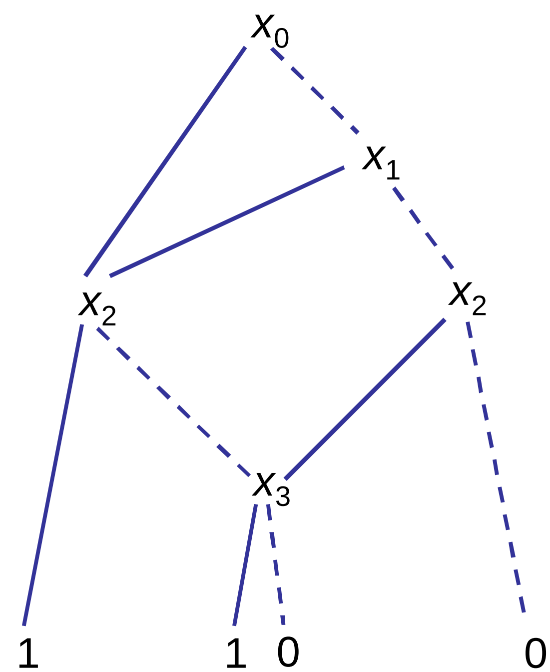


x_0
x_1
x_2
x_2
x_3
x_3
1
1
0
1
0
0
x_0
x_1
x_1
x_2
x_2
x_2
x_2
x_3
x_3
x_3
x_3
1
1
0
1
1
0
1
1
0
1
0
0

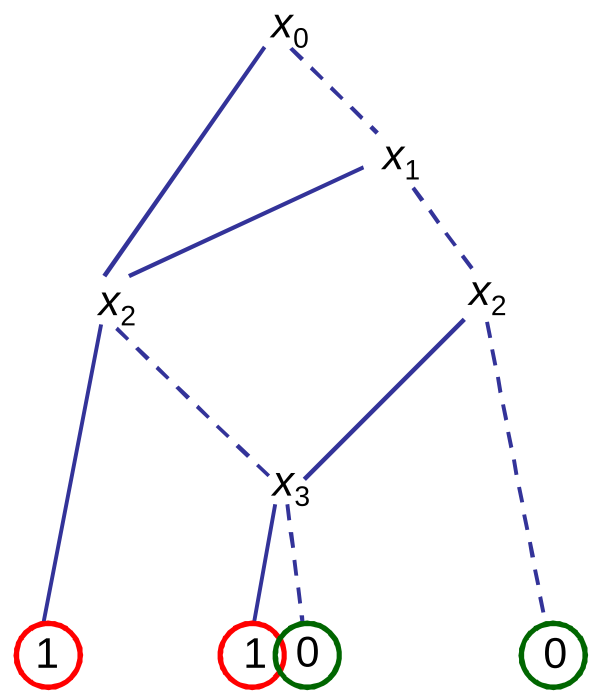


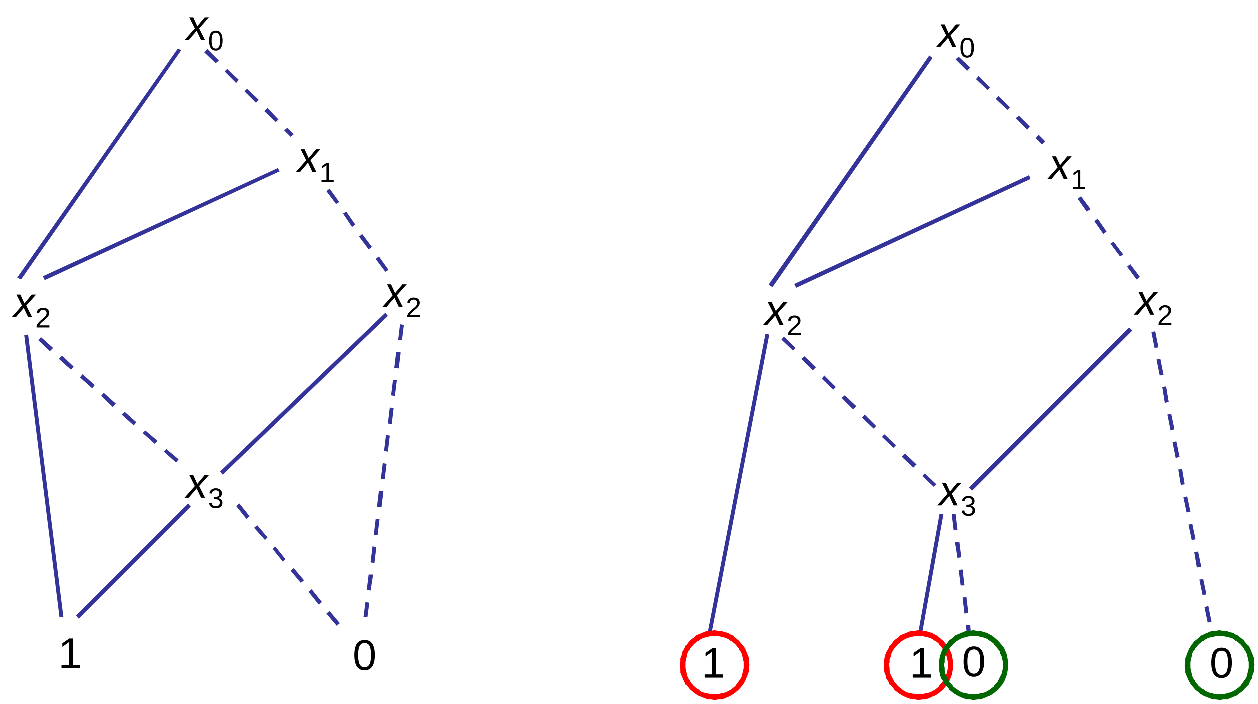
Superimpose identical subtrees…


x_0
x_1
x_2
x_2
x_3
x_3
1
1
0
1
0
0

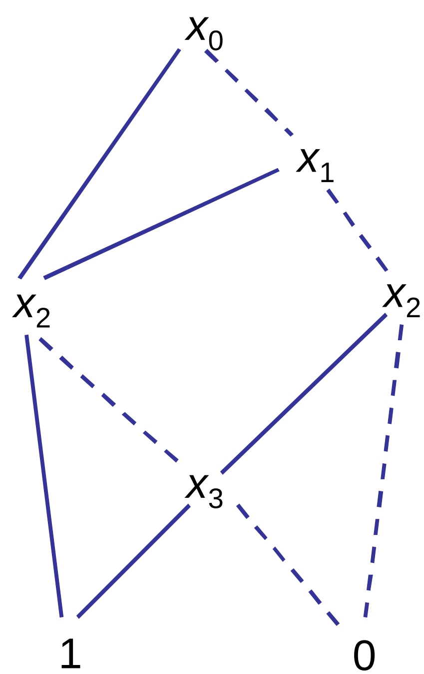

x_0
x_1
x_2
x_2
x_3
1
1
0
0

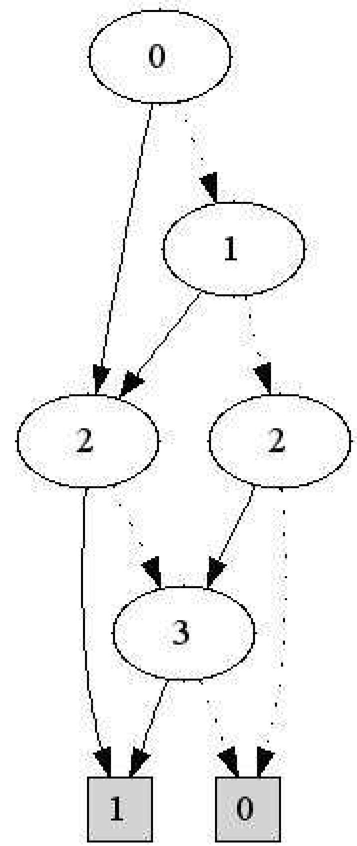
Superimpose identical leaf nodes…





x0
x1
x2
x2
x3
1
0
x0
x1
x2
x2
x3
1
1
0
0





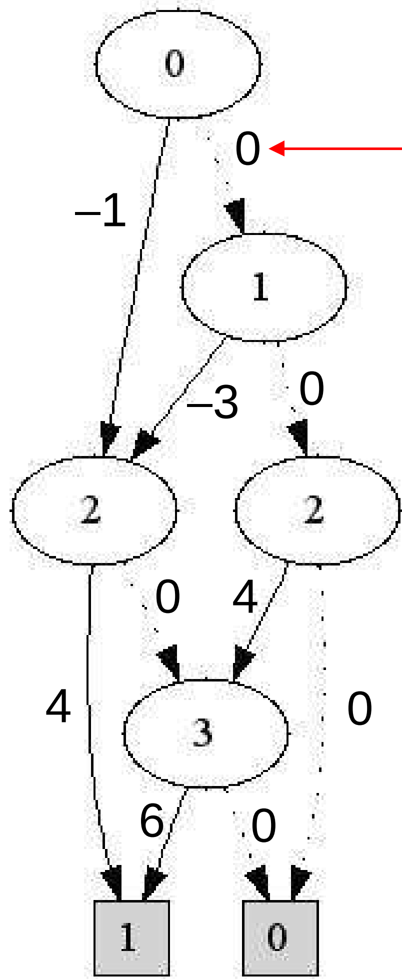
as generated by software

Optimizing with MDDs

- Optimal solution corresponds to a **shortest path** to 1 in the MDD.
  - Provided the objective function is **additively separable**

$$g(x) = \sum_j g_j(x_j)$$

$$\min 2x_0 - 3x_1 + 4x_2 + 6x_3 \quad \text{subject to} \quad 2x_0+3x_1+5x_2+5x_3 \geq 7$$



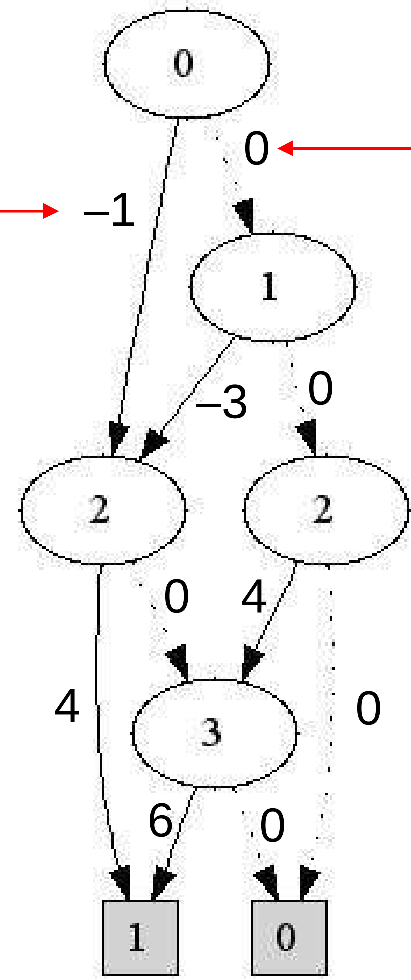
$2 \cdot 0$

Edge lengths reflect terms of objective function

$$\min 2x_0 - 3x_1 + 4x_2 + 6x_3 \quad \text{subject to} \quad 2x_0 + 3x_1 + 5x_2 + 5x_3 \geq 7$$

$$2 \cdot 1 + \min_{x_1 \in \{0,1\}} \{-3x_1\}$$

$$2 \cdot 0$$

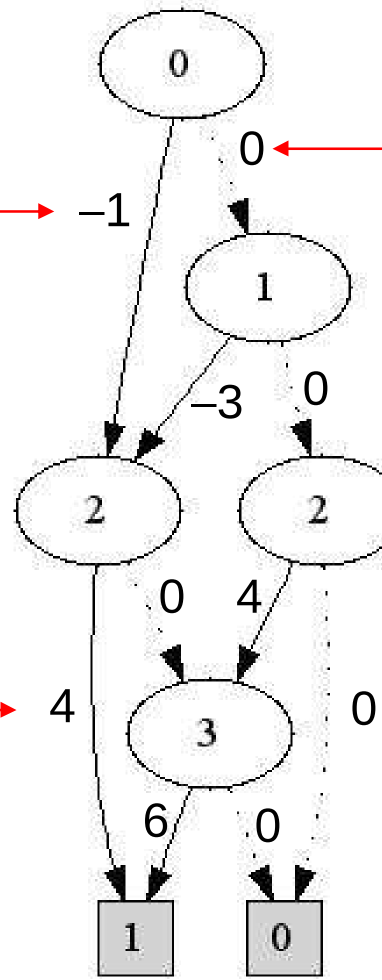


Edge lengths reflect terms of objective function

$$\min\ 2x_0 - 3x_1 + 4x_2 + 6x_3 \quad \text{subject to} \quad 2x_0 + 3x_1 + 5x_2 + 5x_3 \geq 7$$

$$2 \cdot 0$$

$$2 \cdot 1 + \min_{x_1 \in \{0,1\}} \{-3x_1\}$$



Edge lengths reflect terms of objective function

$$4 \cdot 1 + \min_{x_3 \in \{0,1\}} \{6x_3\}$$

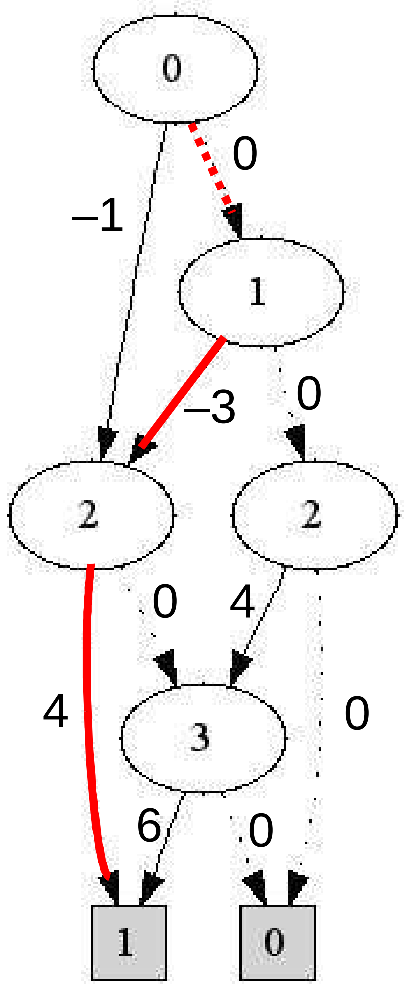
$$\min 2x_0 - 3x_1 + 4x_2 + 6x_3 \quad \text{subject to} \quad 2x_0 + 3x_1 + 5x_2 + 5x_3 \geq 7$$

Shortest path has length 1

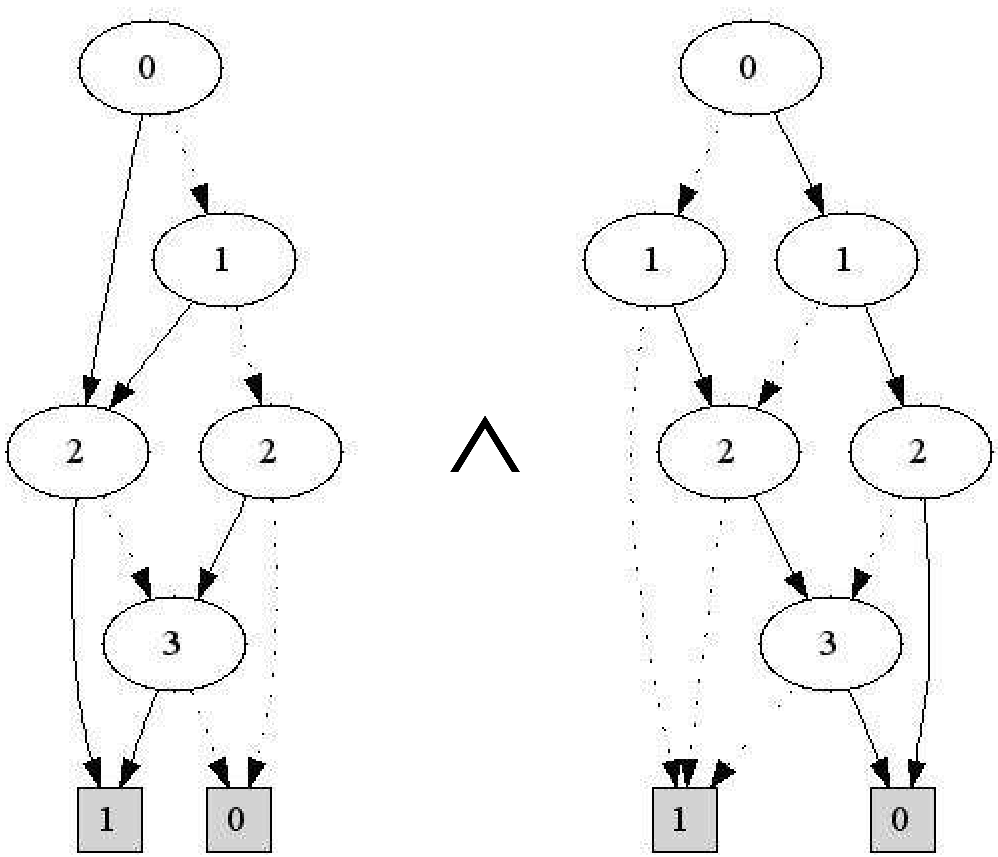
Optimal solution:

$$(x_0, x_1, x_2, x_3) = (0,1,1,0)$$

Set to minimizing value

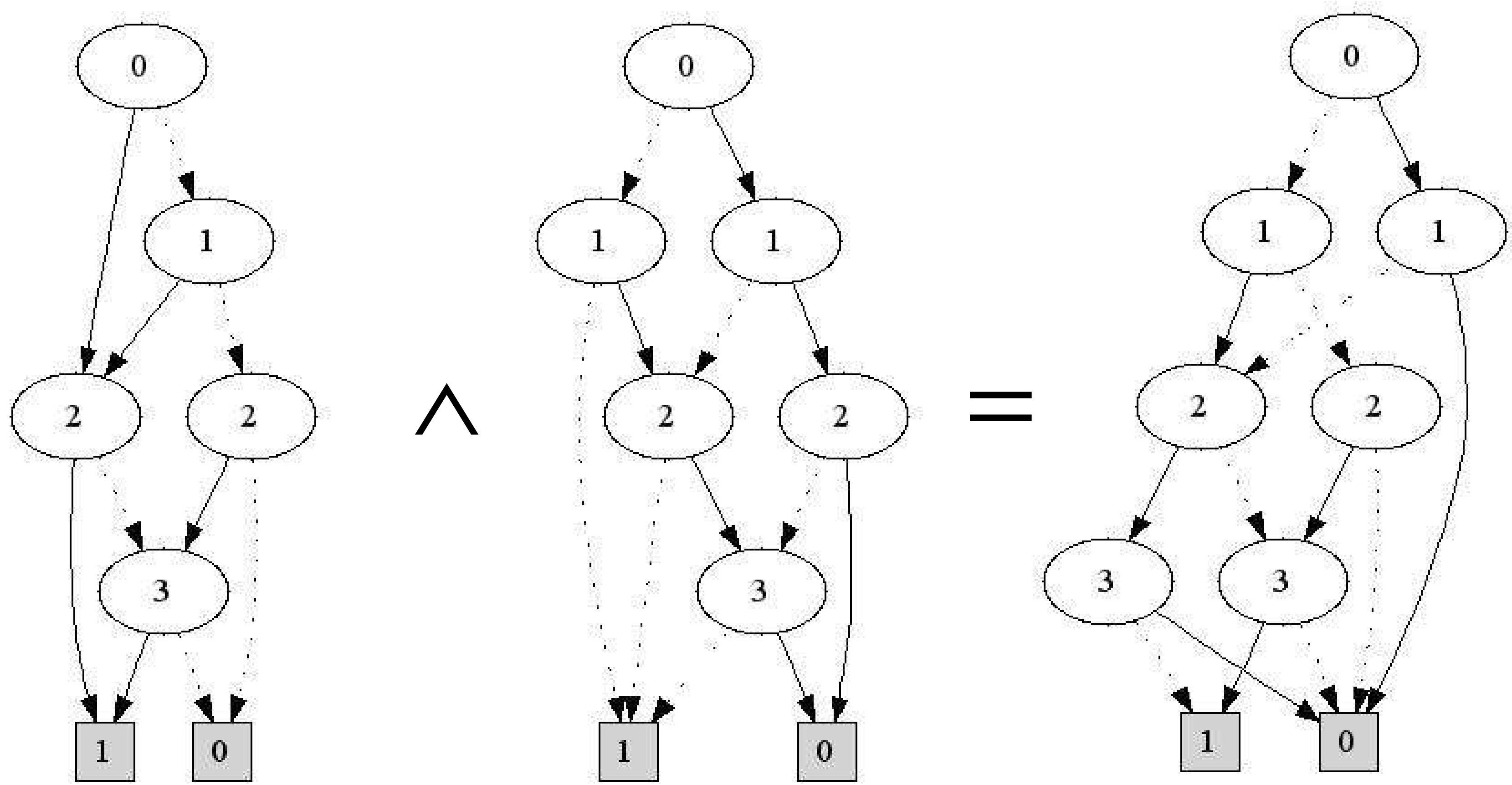


Combine constraints by conjoining MDDs



$$2x_0 + 3x_1 + 5x_2 + 5x_3 \geq 7 \qquad x_0 + x_1 + x_2 + x_3 \leq 2$$

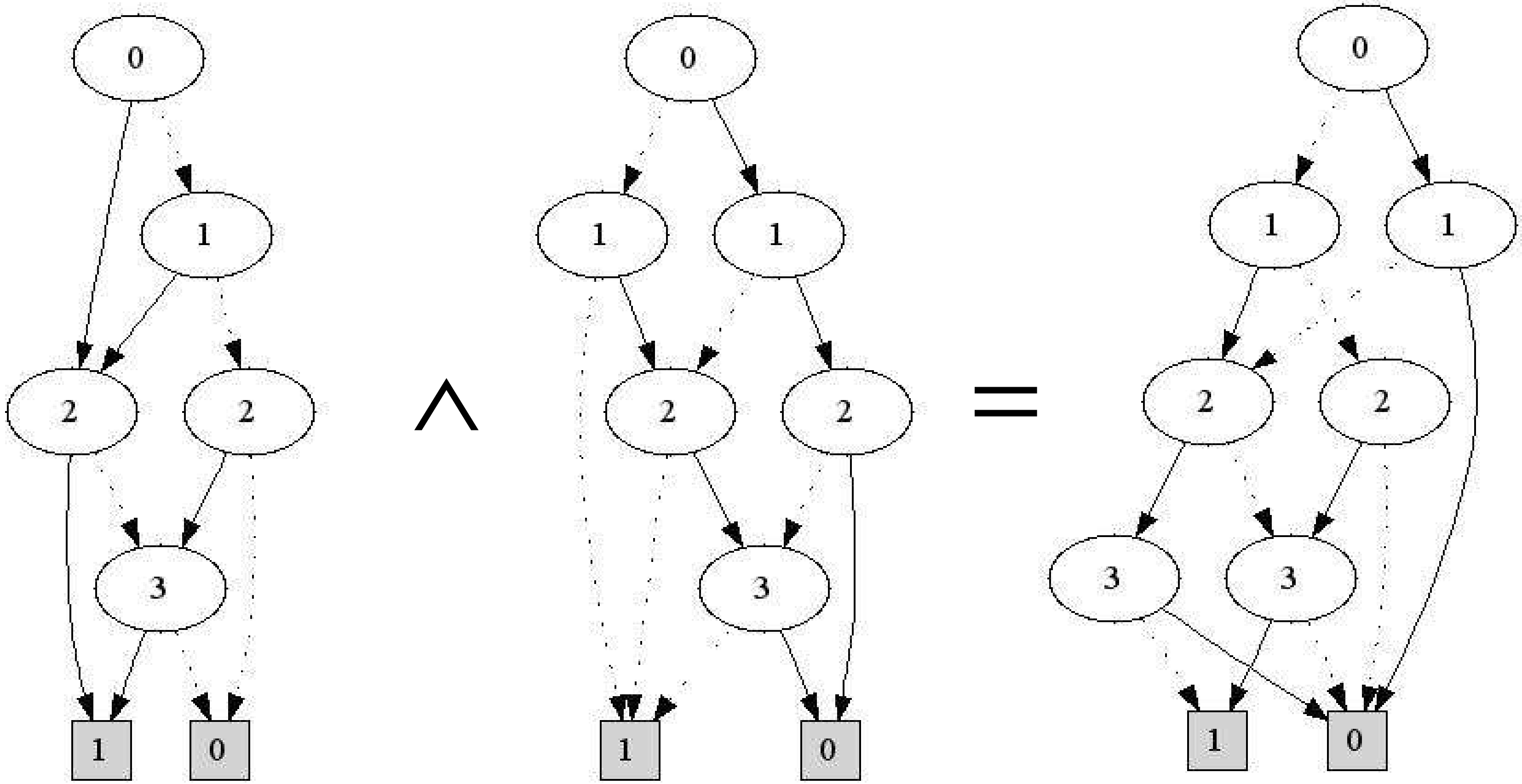
Combine constraints by conjoining MDDs



$2x_0 + 3x_1 + 5x_2 + 5x_3 \geq 7$ $\quad\quad$ $x_0 + x_1 + x_2 + x_3 \leq 2$

Combine constraints by conjoining MDDs

Complexity ≤ product of MDD node counts



$\wedge$

$=$

$2x_0 + 3x_1 + 5x_2 + 5x_3 \geq 7$

$x_0 + x_1 + x_2 + x_3 \leq 2$

BDDs and MDDs

- MDD can grow exponentially with number of variables
  - …but is compact in some important cases.
  - Size can be sensitive to variable ordering.

- MDD doesn't care whether the problem is linear or nonlinear, convex or nonconvex.
  - Functions can take any form (polynomial, etc.).
  - But objective function must be separable.

Constructing MDDs

- One **can** generate an MDD by enumerating search tree.
  - Intelligent caching to identify reduced form.
  - Known bound on optimal value can prune the search.

- Most BDD software computes BDD for an expression by combining BDDs for **subexpressions**.

Constructing MDDs

- One **can** generate an MDD by enumerating search tree.
  - Intelligent caching to identify reduced form.
  - Known bound on optimal value can prune the search.
- Most BDD software computes BDD for an expression by combining BDDs for **subexpressions**.
- Several ways to **limit size** of MDD:
  - Use cost bounding to exclude bad solutions when combining.
  - Create **relaxed** or **restricted** MDD of limited width.

Constructing MDDs

- BDD for a knapsack constraint can be surprisingly small…

Constructing MDDs

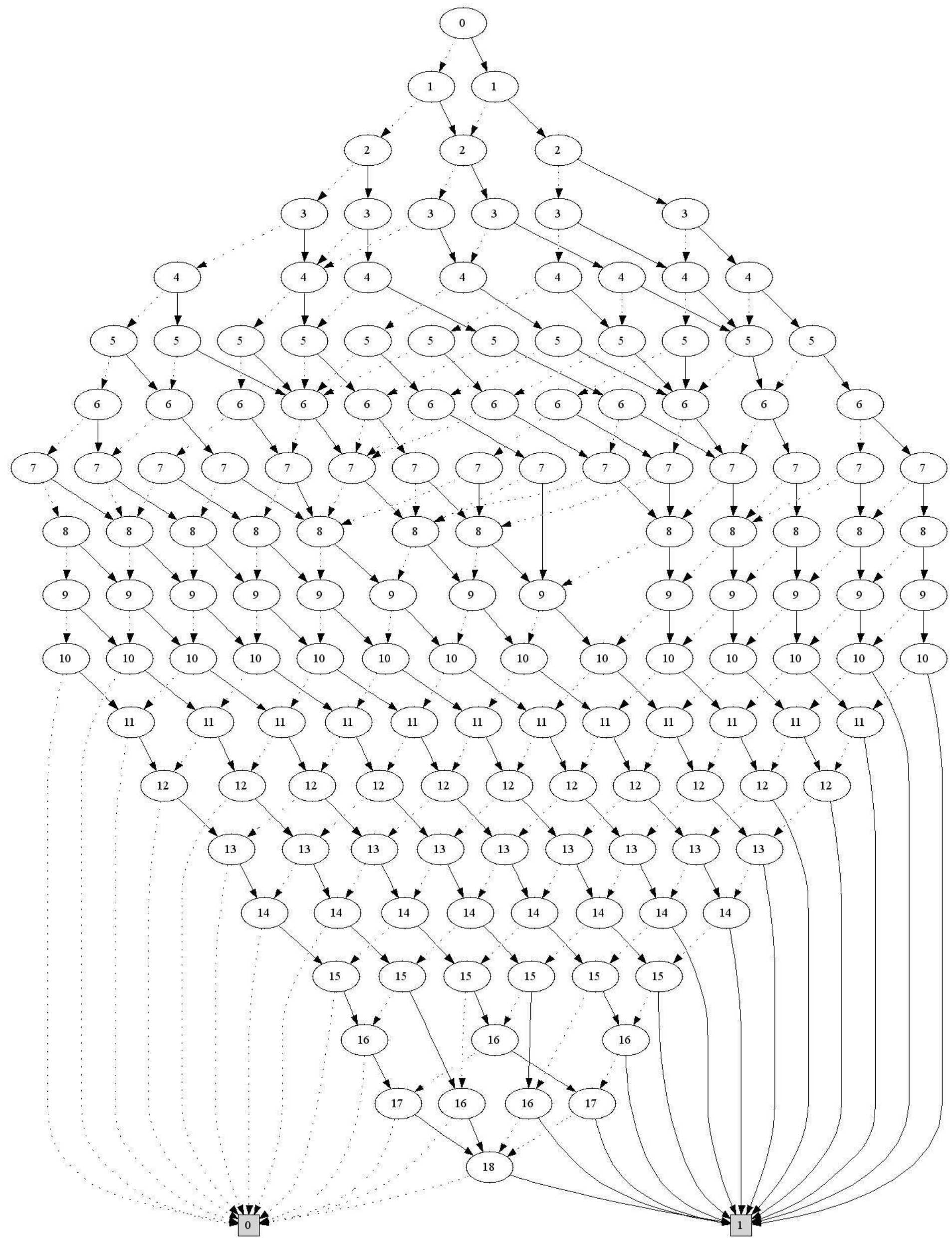
- BDD for a knapsack constraint can be surprisingly small…

The 0-1 inequality

$$300x_0 + 300x_1 + 285x_2 + 285x_3 + 265x_4 + 265x_5 + 230x_6 + 230x_7 + 190x_8 + 200x_9 + \\ 400x_{10} + 200x_{11} + 400x_{12} + 200x_{13} + 400x_{14} + 200x_{15} + 400x_{16} + 200x_{17} + 400x_{18} \le 2700$$

has 117,520 minimal feasible solutions (or minimal covers)

But its reduced BDD has only 152 nodes…



Historical Applications of BDDs

- Circuit checking.
  - Want to implement a boolean function on a **chip**.
  - Generate **reduced BDDs** for the function and the chip and compare.
  - BDD size varies.
    - Adders are polynomial.
    - Multipliers are exponential.

Historical Applications of BDDs

- Circuit checking.
  - Want to implement a boolean function on a **chip**.
  - Generate **reduced BDDs** for the function and the chip and compare.
  - BDD size varies.
    - Adders are polynomial.
    - Multipliers are exponential.

- Configuration
  - Encode possible **design configurations** in a BDD.
  - Quickly deduce consequences of **fixing** some variables.
  - **Transparency** of BDD – easily queried.

MDDs as Propagators

MDDs as Propagators

- We can replace the **domain store** in CP with a **relaxed MDD**.
  - All feasible solutions are paths in the MDD
  - But not all paths are feasible solutions.

MDDs as Propagators

- We can replace the **domain store** in CP with a **relaxed MDD**.
  - All feasible solutions are paths in the MDD
  - But not all paths are feasible solutions.

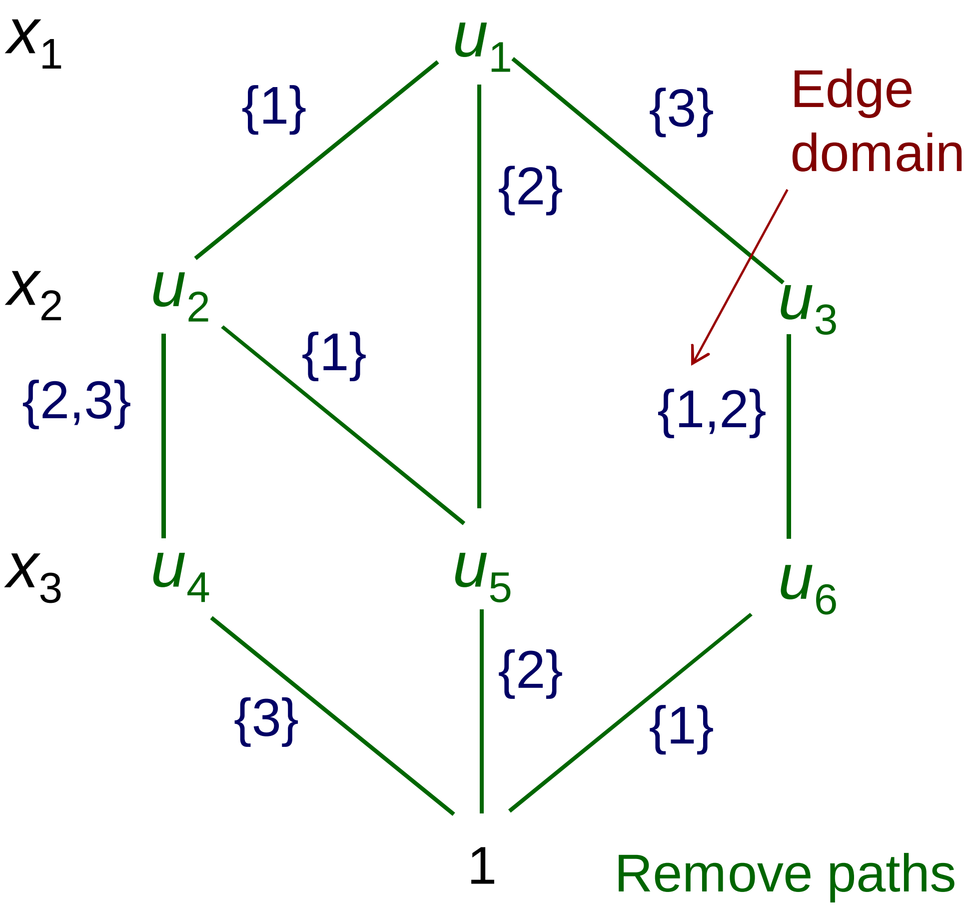
- **Limit the width** of the relaxed MDD to avoid blowup.
  - Width of 1 = traditional domain store

MDDs as Propagators

- We can replace the **domain store** in CP with a **relaxed MDD**.
  - All feasible solutions are paths in the MDD
  - But not all paths are feasible solutions.
- **Limit the width** of the relaxed MDD to avoid blowup.
  - Width of 1 = traditional domain store
- Achieve **balance** between search and inference
  - More **information shared** between constraints.
  - It pays to enumerate fewer nodes and spend more time processing each node.

MDDs as a Propagators

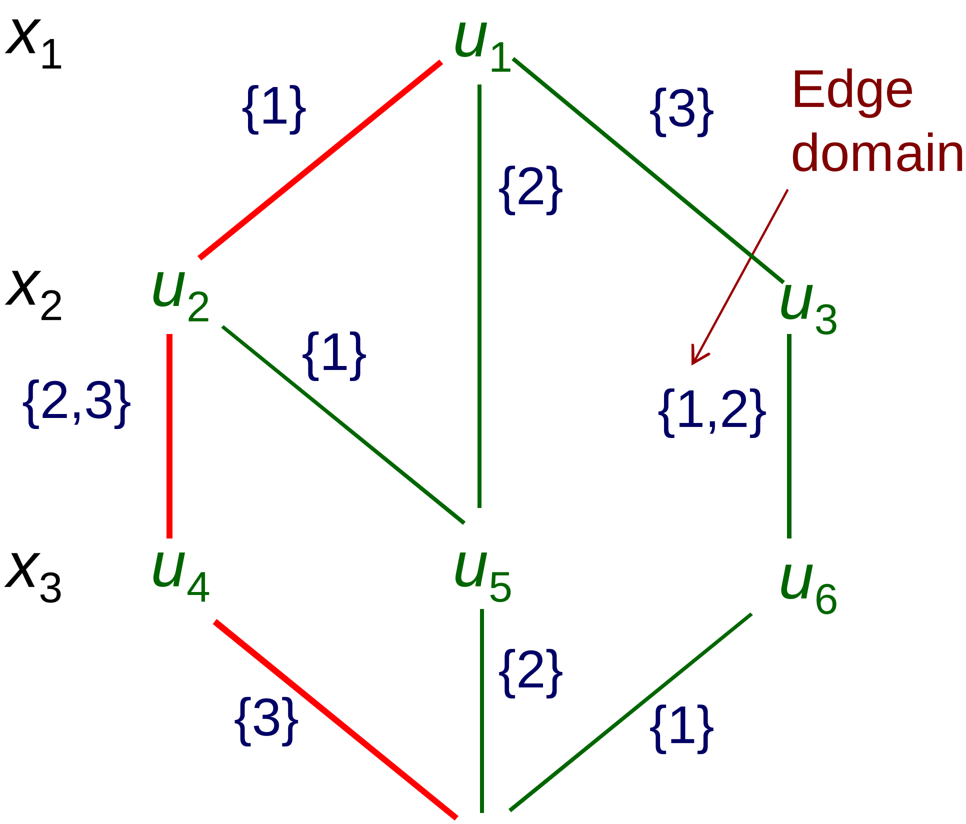
Consider the MDD



Remove paths to 0

MDDs as a Propagators

Consider the MDD



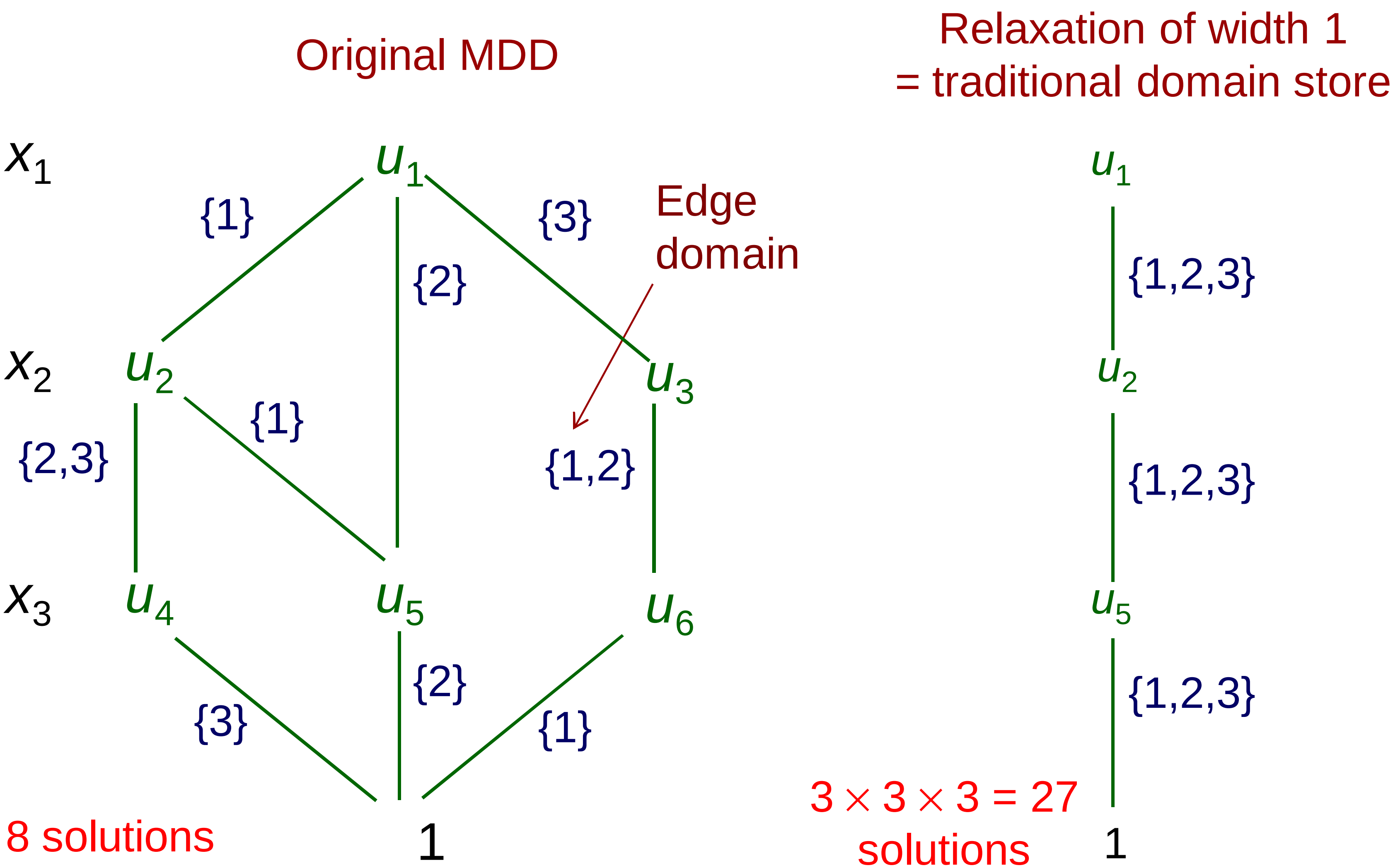
8 solutions

Each path represents a set of solutions

$\{1\}\times\{2,3\}\times\{3\}$ =

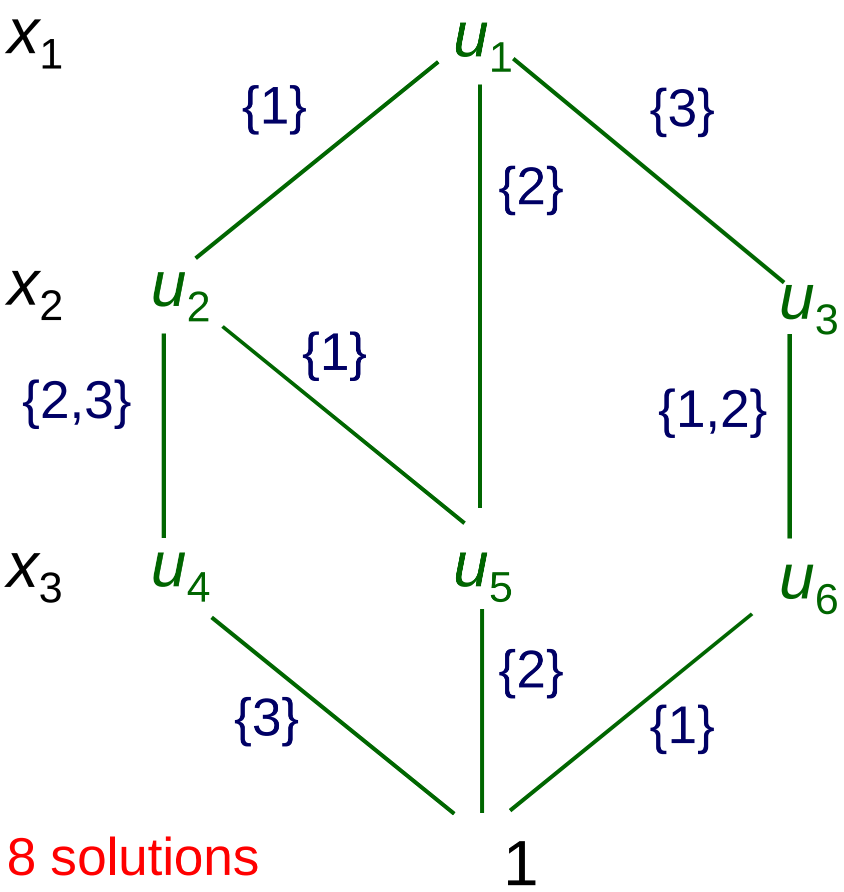
{(1,2,3), (1,3,3)}

MDD Relaxation



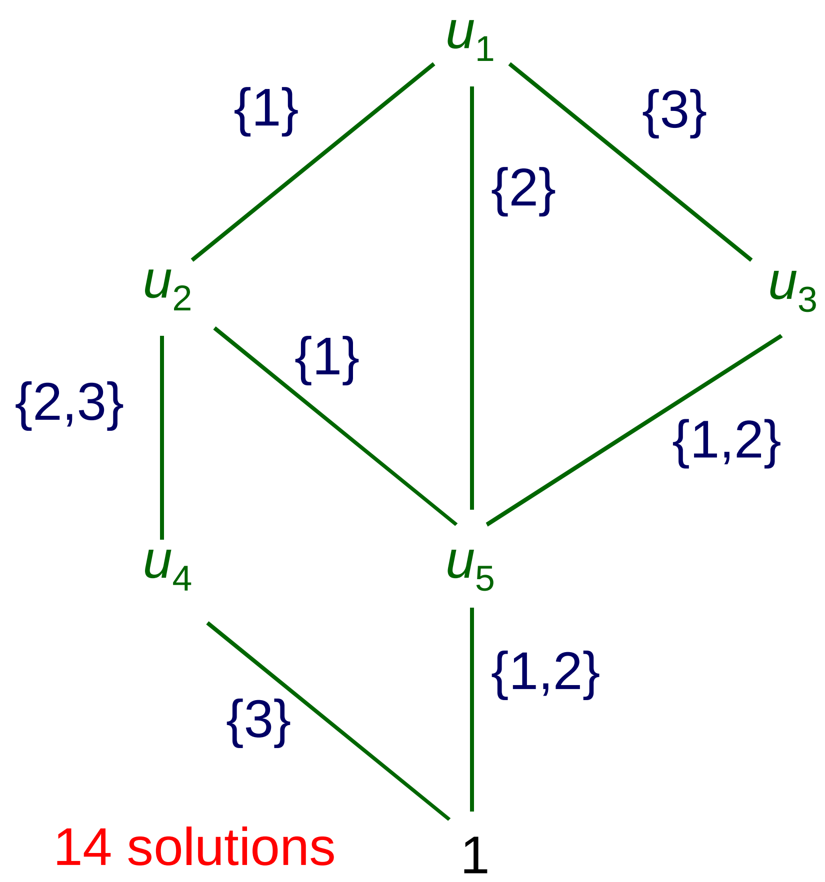
MDD Relaxation

Original MDD



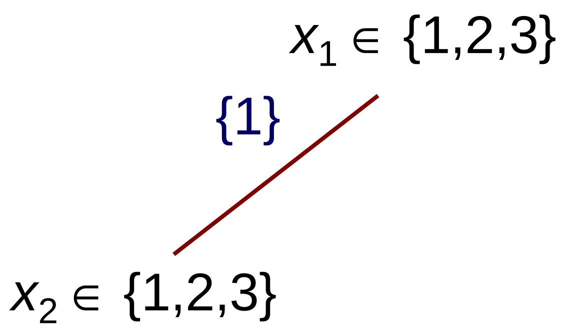
8 solutions

Relaxation of width 2

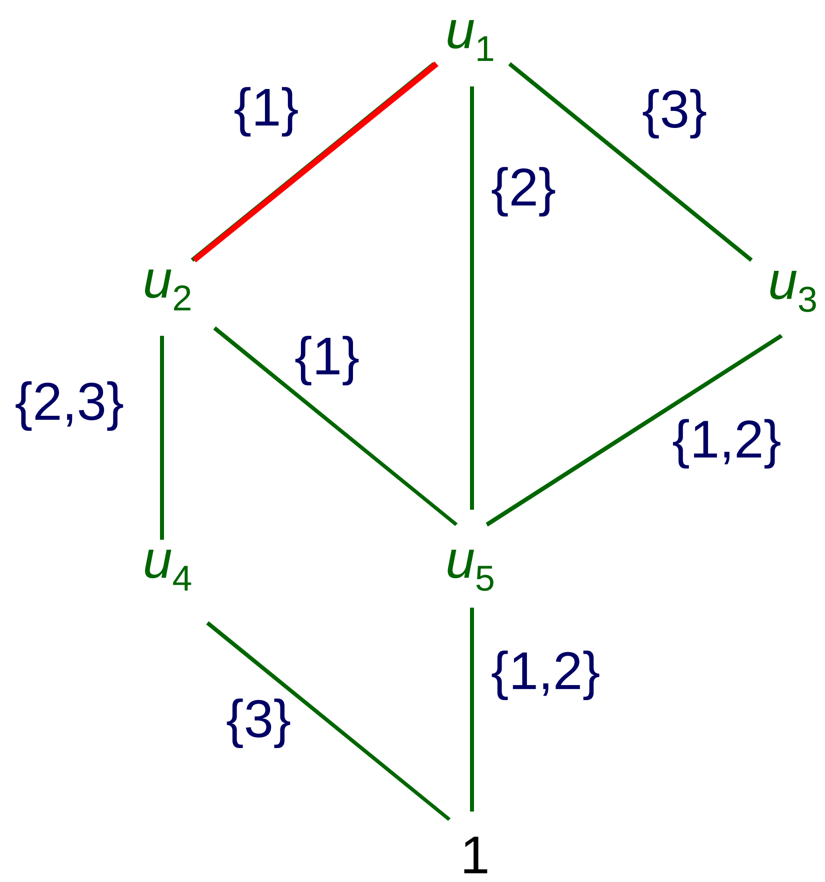


14 solutions

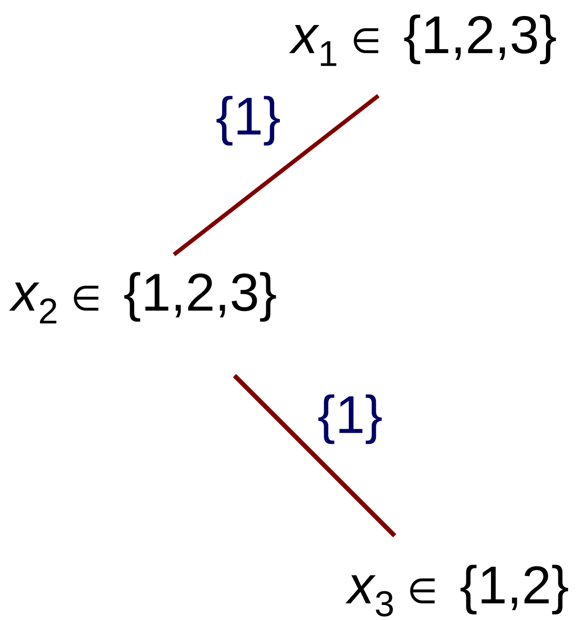
MDD-Guided Branching



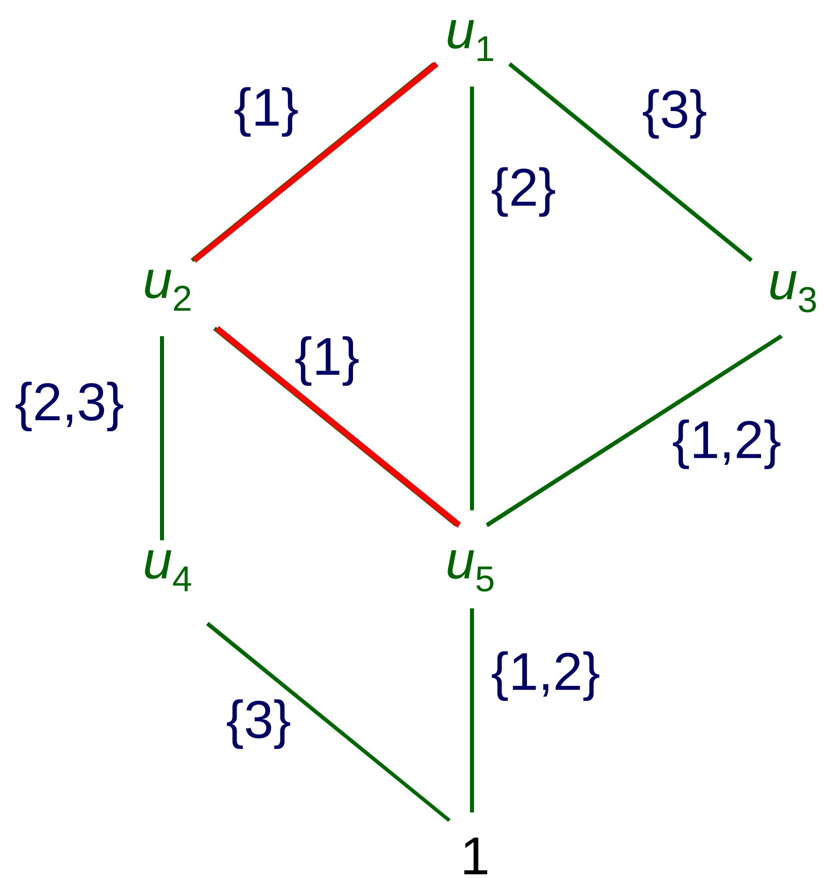
Relaxation of width 2



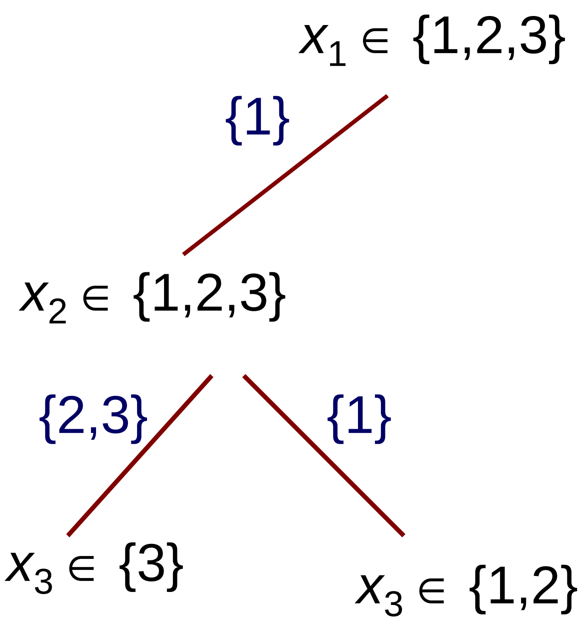
MDD-Guided Branching



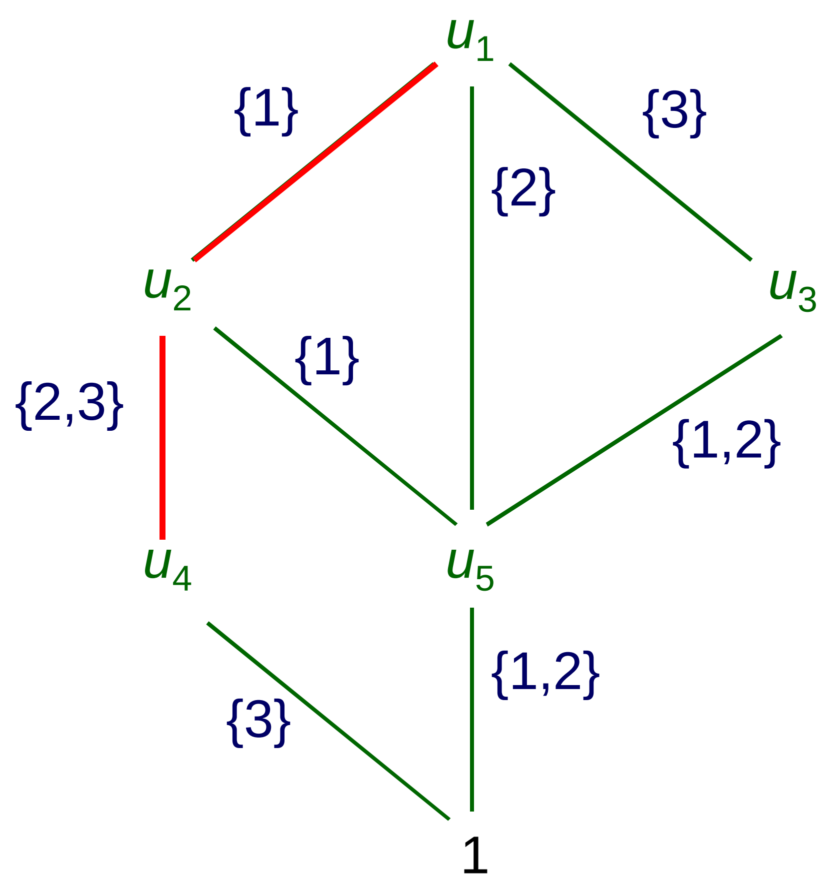
Relaxation of width 2



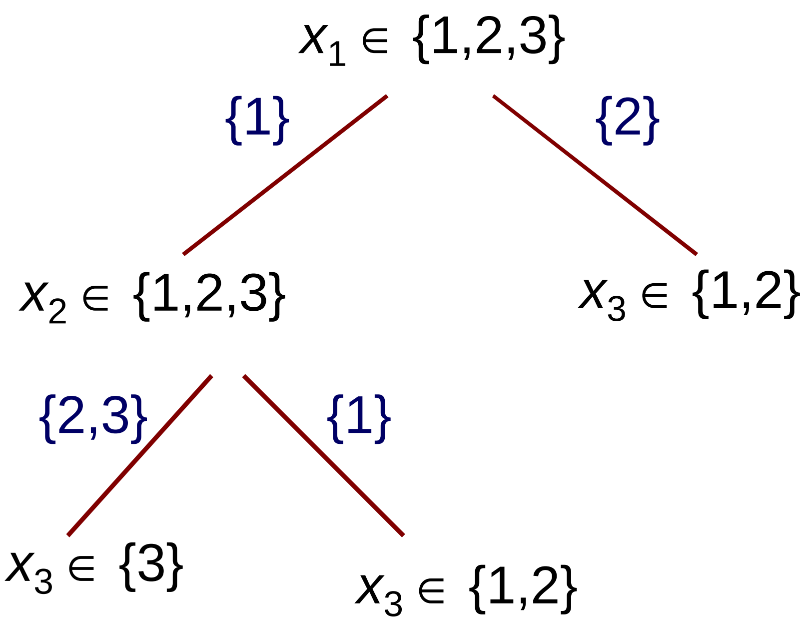
MDD-Guided Branching



Relaxation of width 2



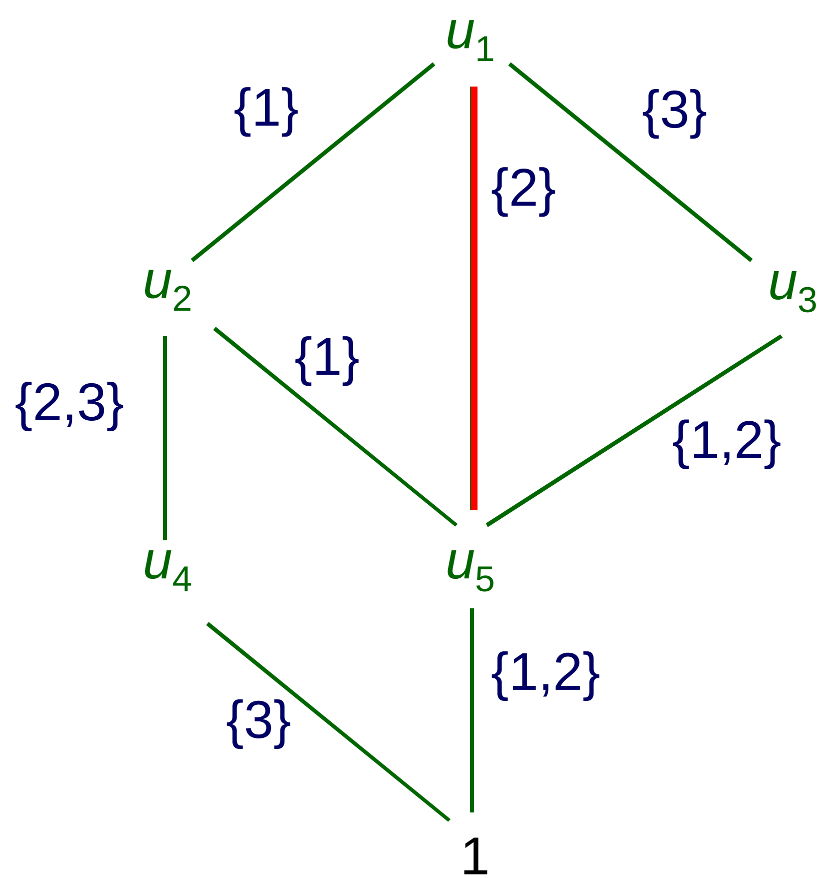
MDD-Guided Branching



And so forth.

Less branching than with domain store.

Relaxation of width 2



MDDs as Propagators

- To propagate a constraint through an MDD:
  - **Filter** edge domains
  - Refine the MDD by **node splitting**
  - Do not exceed max width

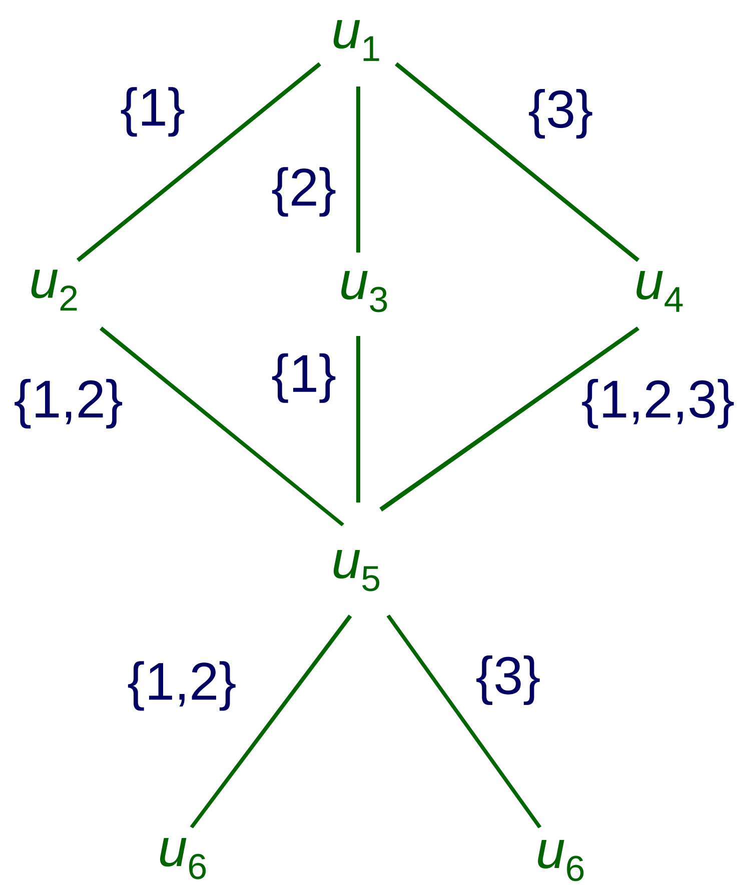
MDDs as Propagators

- To propagate a constraint through an MDD:
  - **Filter** edge domains
  - Refine the MDD by **node splitting**
  - Do not exceed max width

- **Example:**
  - Propagate **alldiff** in an MDD relaxation of width 3

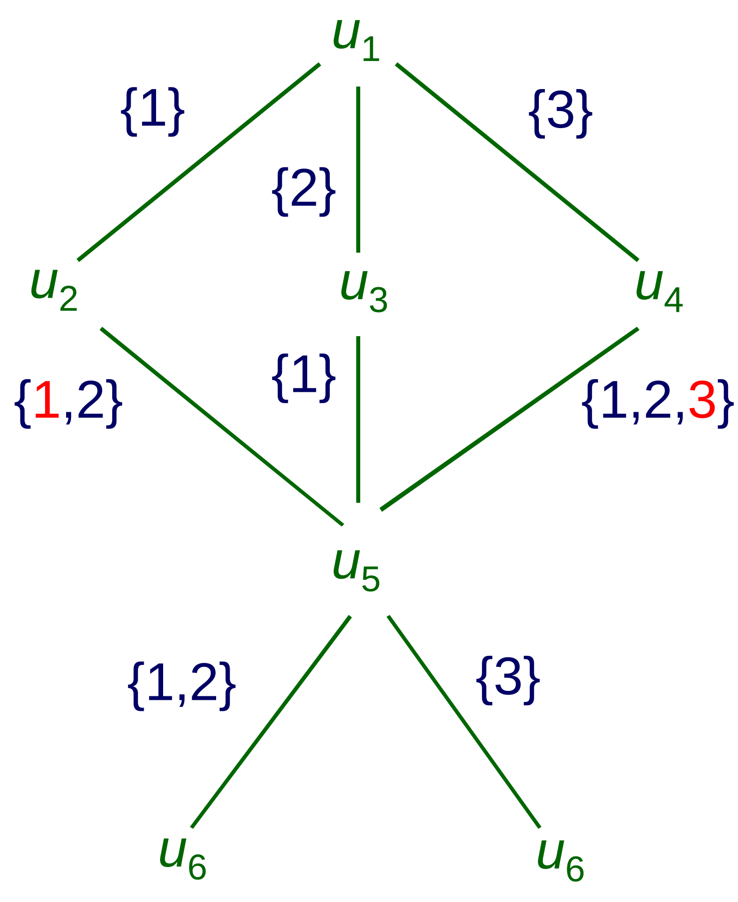
Propagation in an MDD

Current MDD relaxation



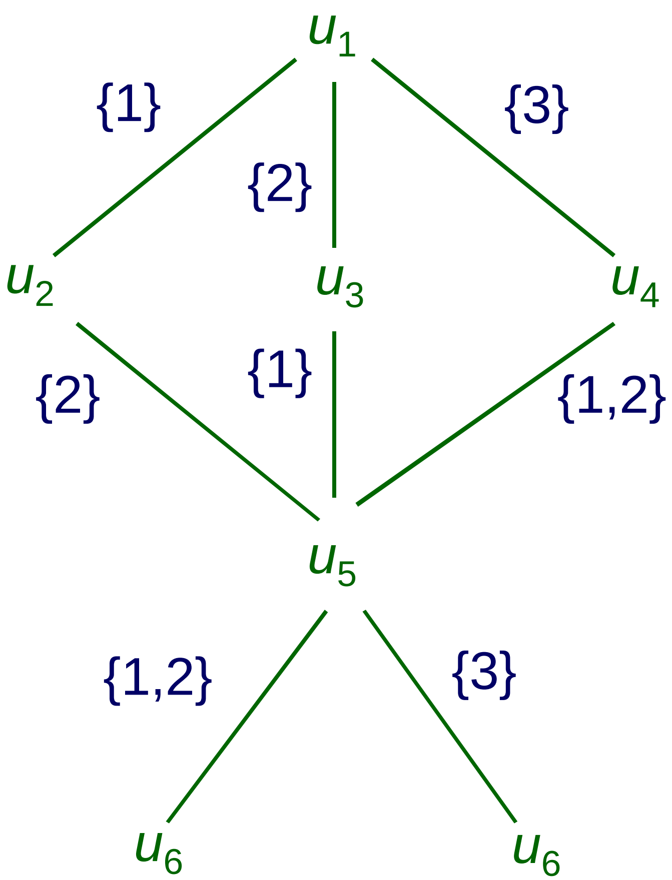
Propagation in an MDD

First filter edge
domains using alldiff



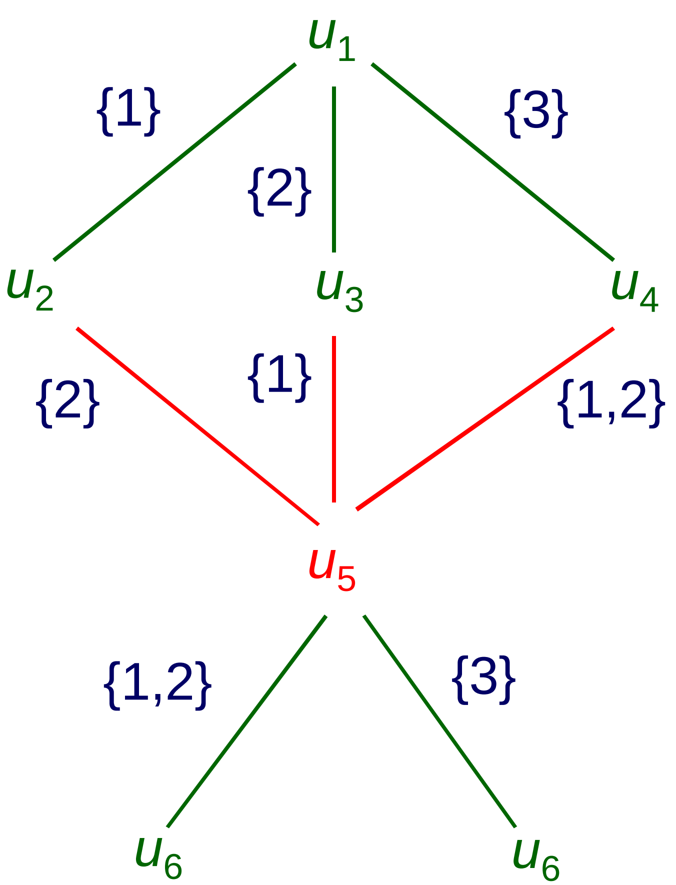
Propagation in an MDD

First filter edge domains using alldiff



Propagation in an MDD

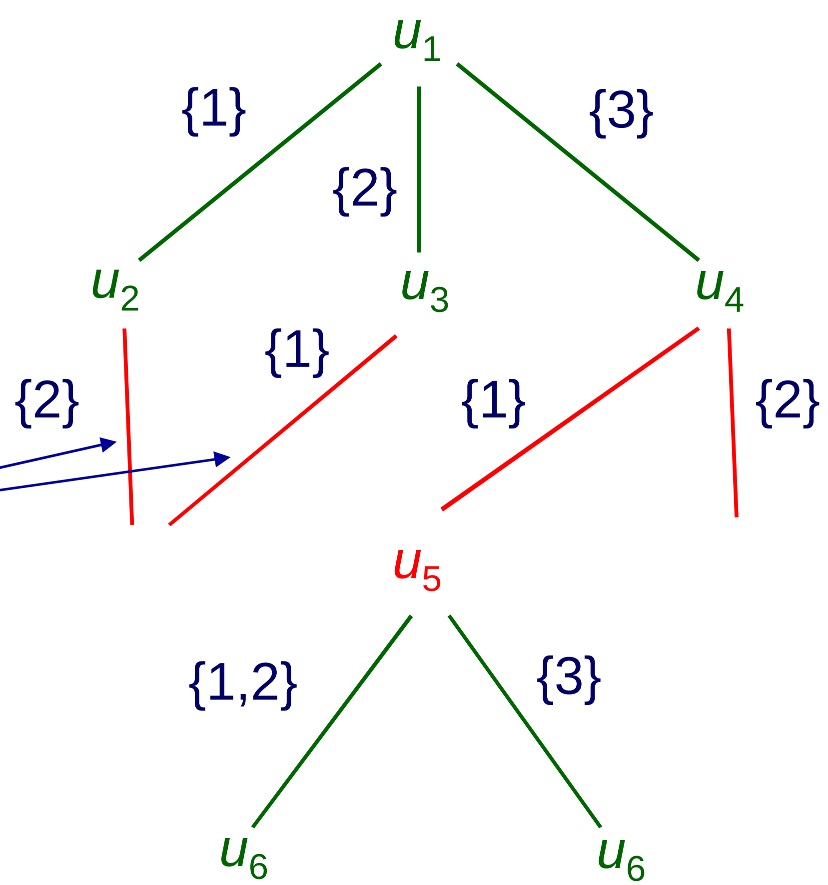
To split u_5:
Identify equivalence
classes of incoming edges



Propagation in an MDD

To split u_5:
Identify equivalence classes of incoming edges

These are equivalent for alldiff because they lead to the same set of feasible paths.

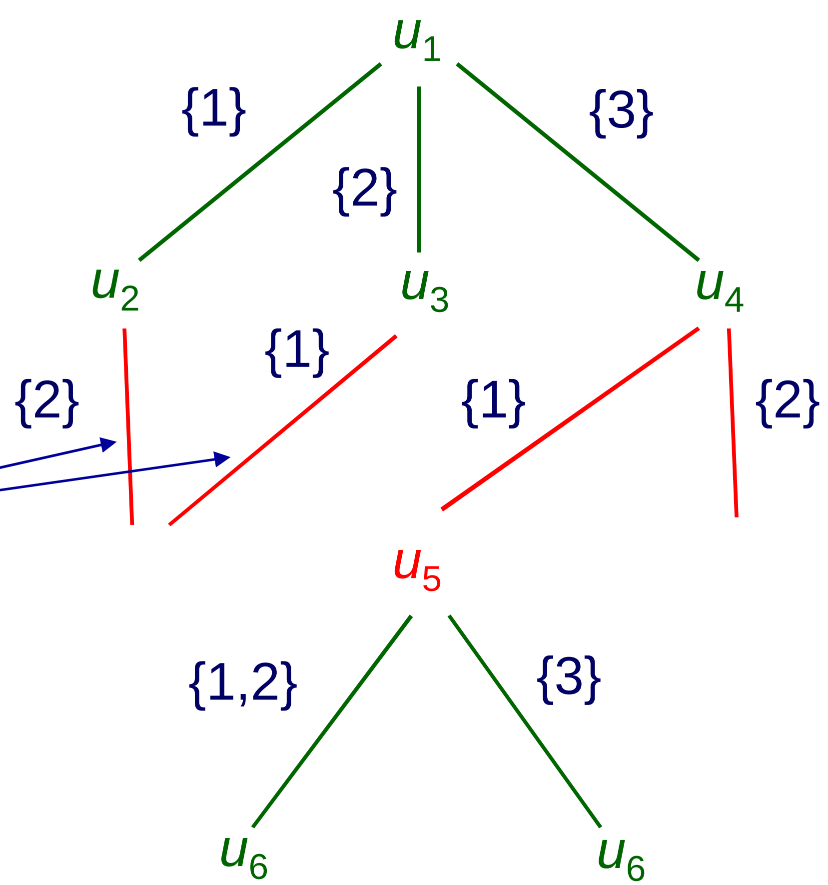


Propagation in an MDD

To split u_5:
Identify equivalence classes of incoming edges

These are equivalent for alldiff because they lead to the same set of feasible paths.

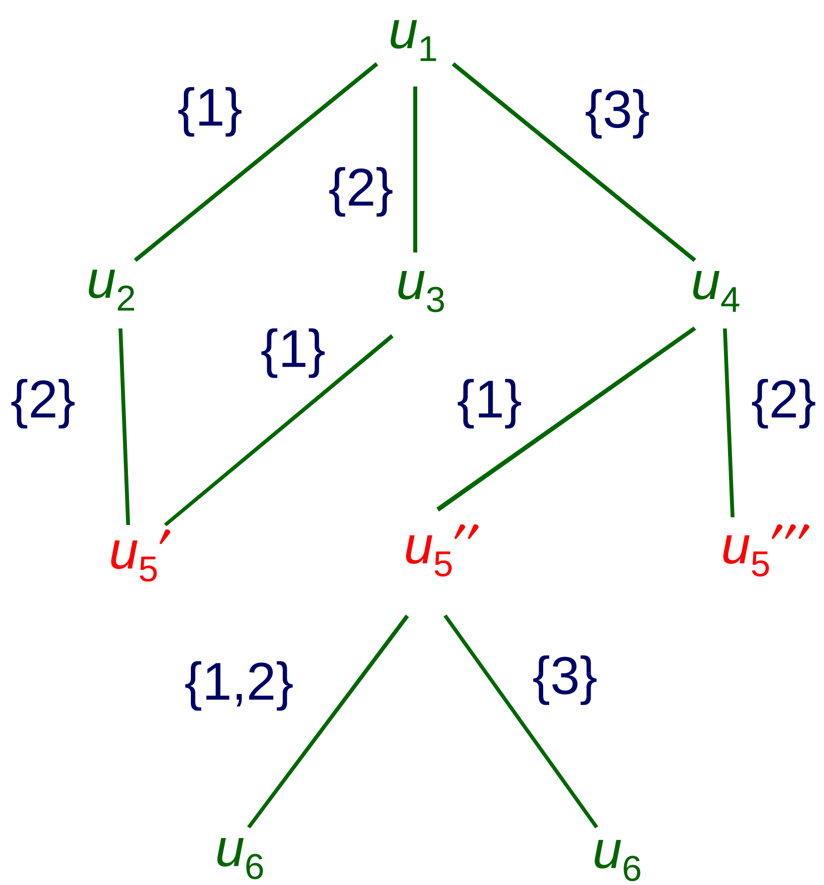
Easy to check equivalence for alldiff – all incoming paths use same values



Propagation in an MDD

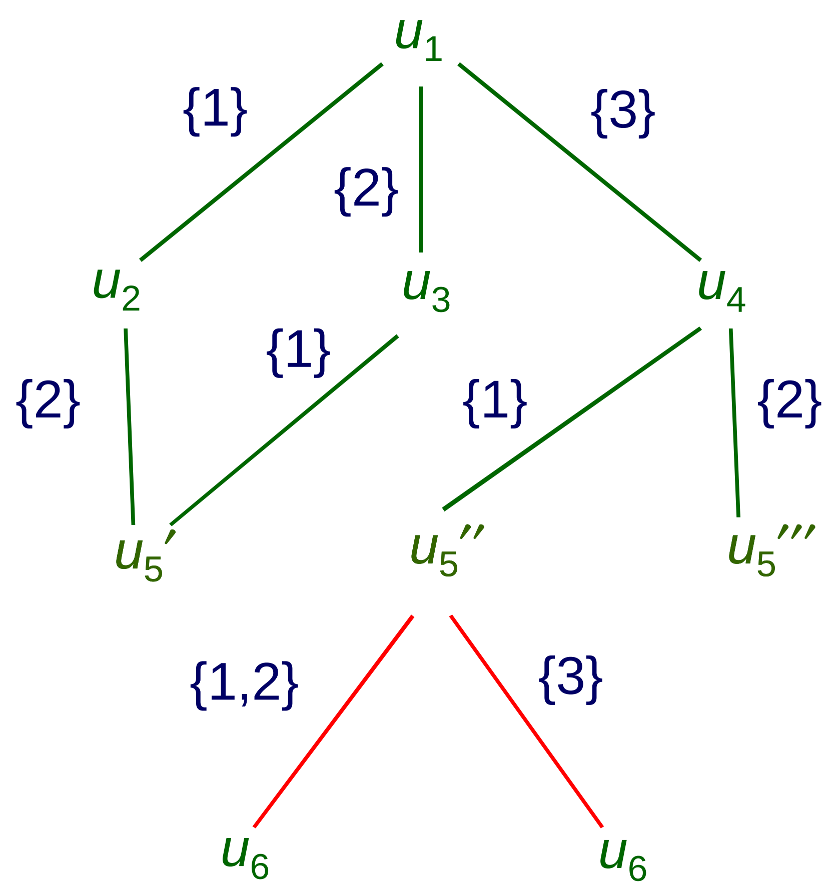
To split u_5:
Identify equivalence classes of incoming edges.

Split u_5 to receive ≤ 3 equivalence classes.



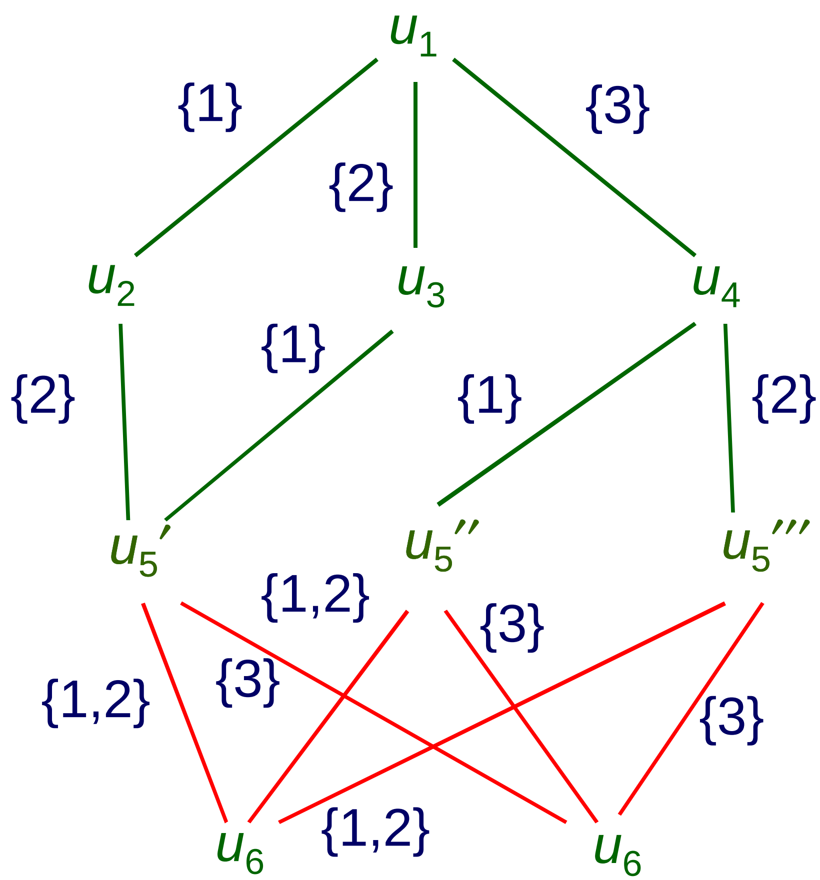
Propagation in an MDD

Duplicate outgoing edges.



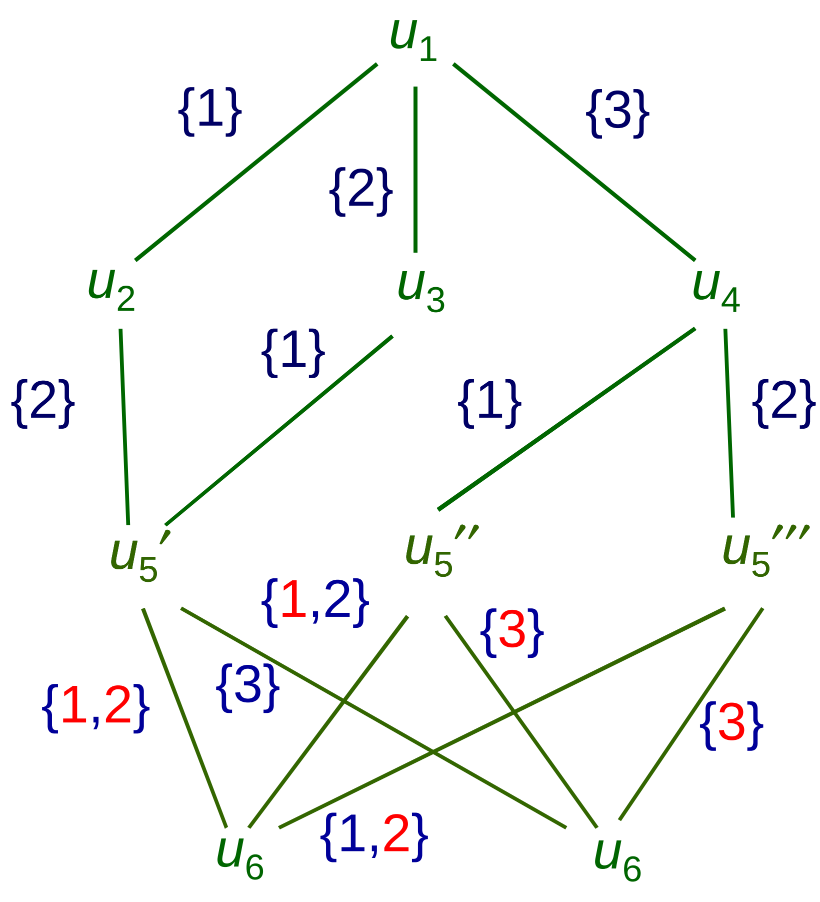
Propagation in an MDD

Duplicate outgoing edges.



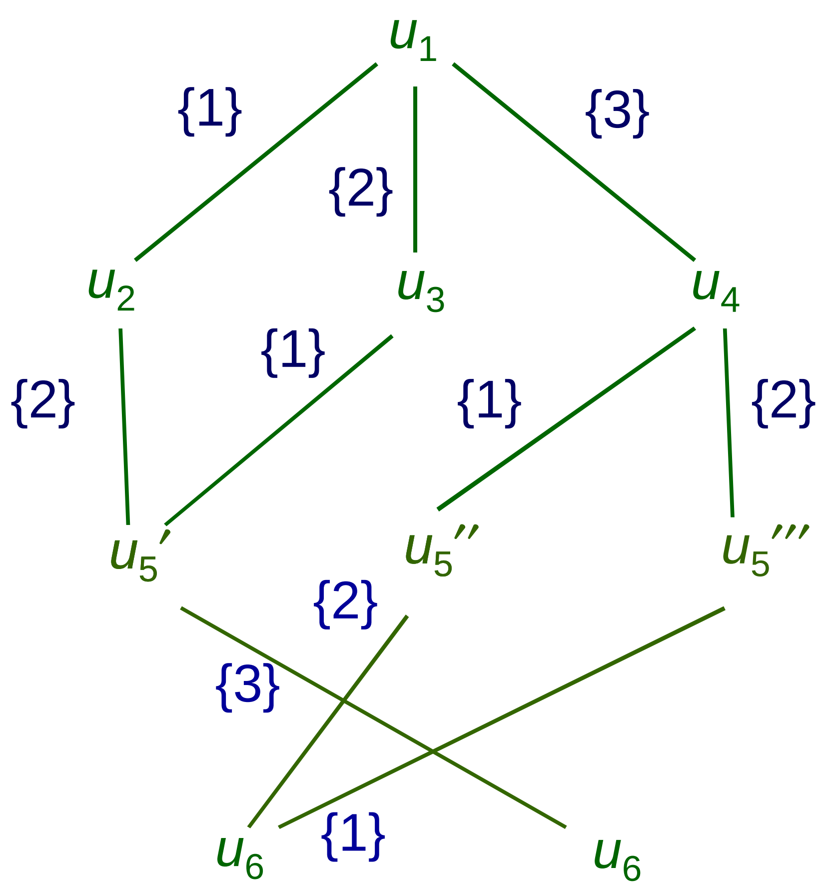
Propagation in an MDD

Filter domains.



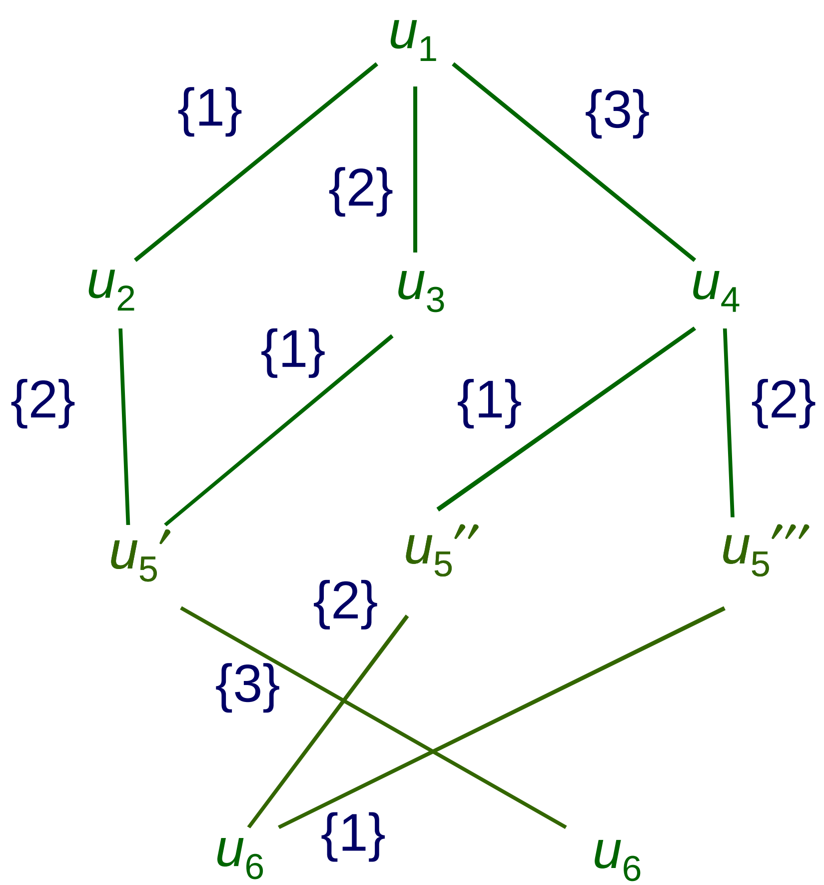
Propagation in an MDD

Filter domains.



Propagation in an MDD

Alldiff has now been propagated.



MDDs as Propagators

- Computational results – **alldiffs**
  - Conventional domain store – about a million search tree nodes
  - MDD constraint store (width 5) – one node
  - MDD solution about 30 times faster

MDDs as Propagators

- Computational results – **alldiffs**
  - Conventional domain store – about a million search tree nodes
  - MDD constraint store (width 5) – one node
  - MDD solution about 30 times faster

- Computational results – **amongs**
  - CP talk Wednesday

MDDs as Relaxations

MDDs as Relaxations

- Shortest path in MDD relaxation provides a bound on optimal value
  - Use as in conventional branch and bound.

MDDs as Relaxations

- Shortest path in MDD relaxation provides a bound on optimal value
  - Use as in conventional branch and bound.

- Strengthen bound by removing infeasible shortest paths.
  - Until shortest path becomes feasible, in which case problem is solved.
  - Or until MDD becomes too large.
  - Roughly analogous to adding Gomory cuts to an IP.

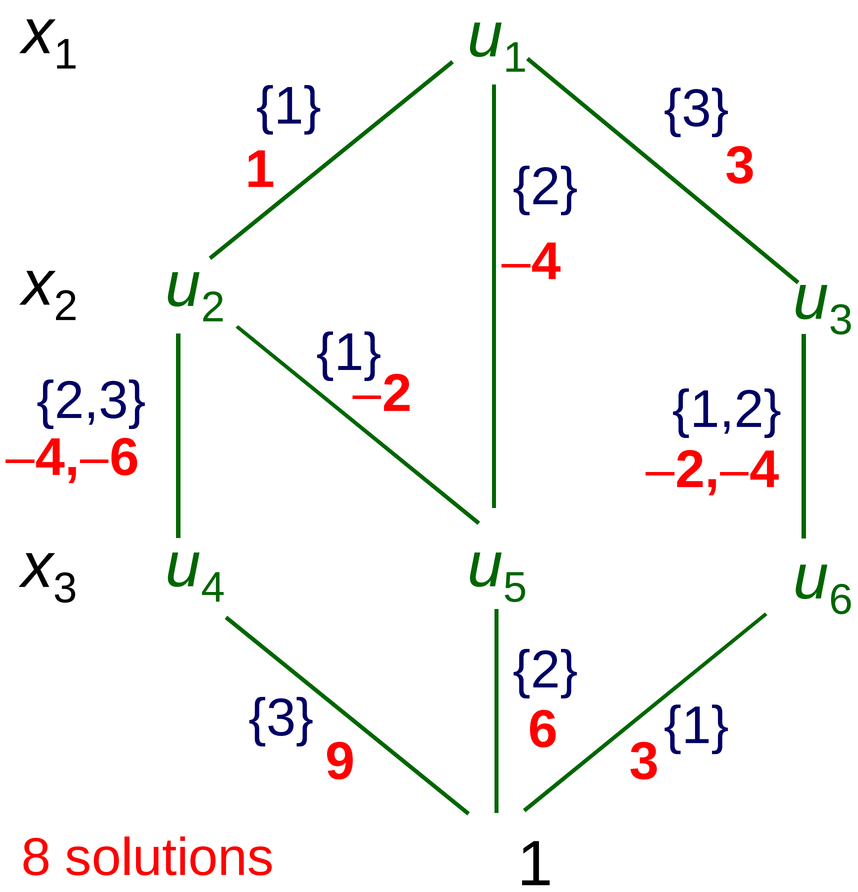
MDDs as Relaxations

- Shortest path in MDD relaxation provides a bound on optimal value
  - Use as in conventional branch and bound.
- Strengthen bound by removing infeasible shortest paths.
  - Until shortest path becomes feasible, in which case problem is solved.
  - Or until MDD becomes too large.
  - Roughly analogous to adding Gomory cuts to an IP.
- MDD can provide **pseudocosts** to guide branching.
  - Test effect on shortest path length of fixing a variable

MDDs as Relaxations

Original MDD

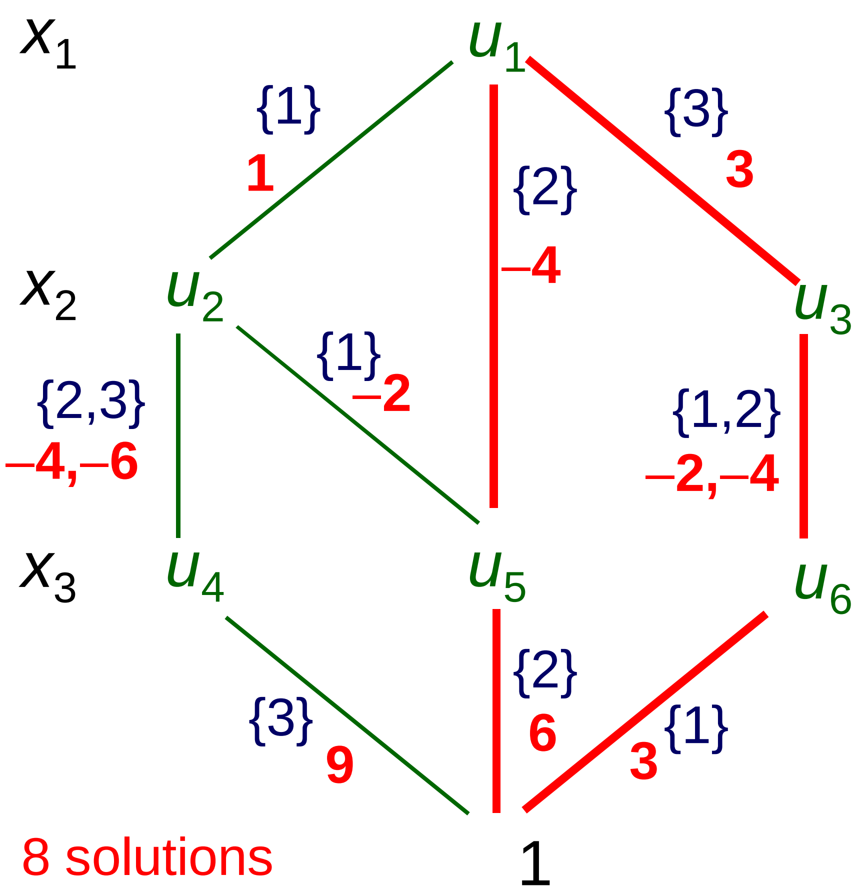
$\min\ x_1 - 2x_2 + 3x_3$



8 solutions

MDDs as Relaxations

Original MDD



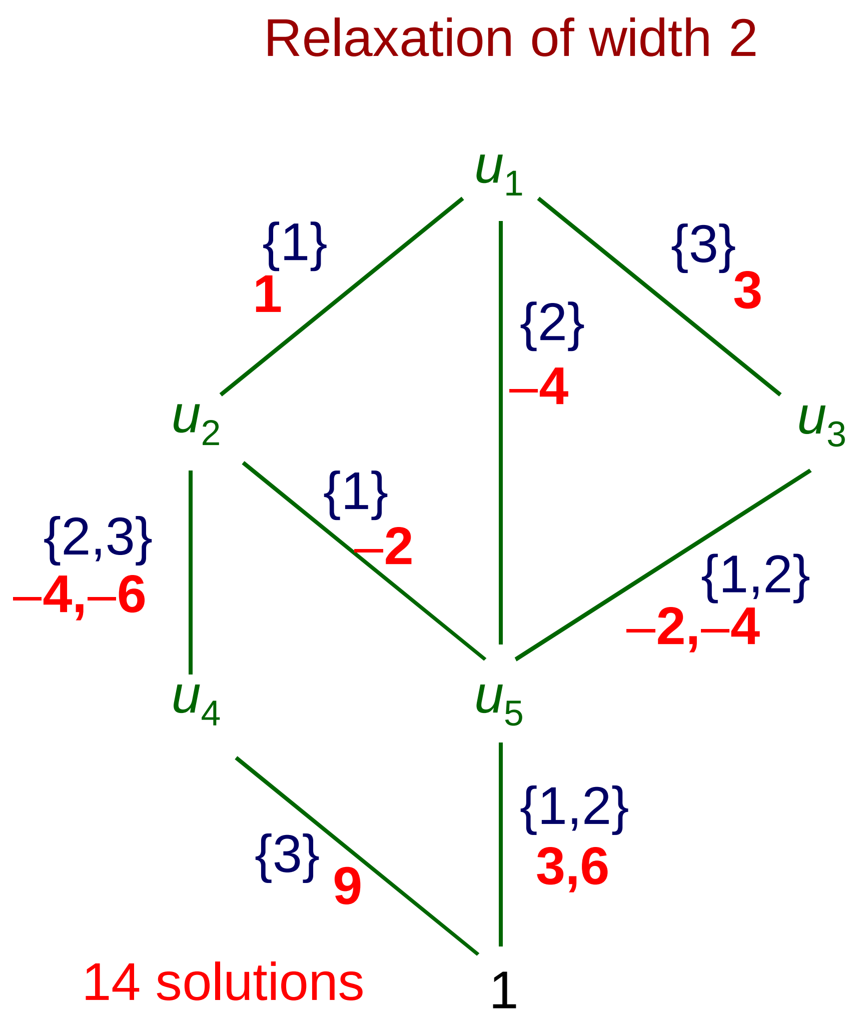
8 solutions

min $x_1 - 2x_2 + 3x_3$

Optimal value = 2
(shortest path length)

Tightening the MDD Relaxation

$\min x_1 - 2x_2 + 3x_3$



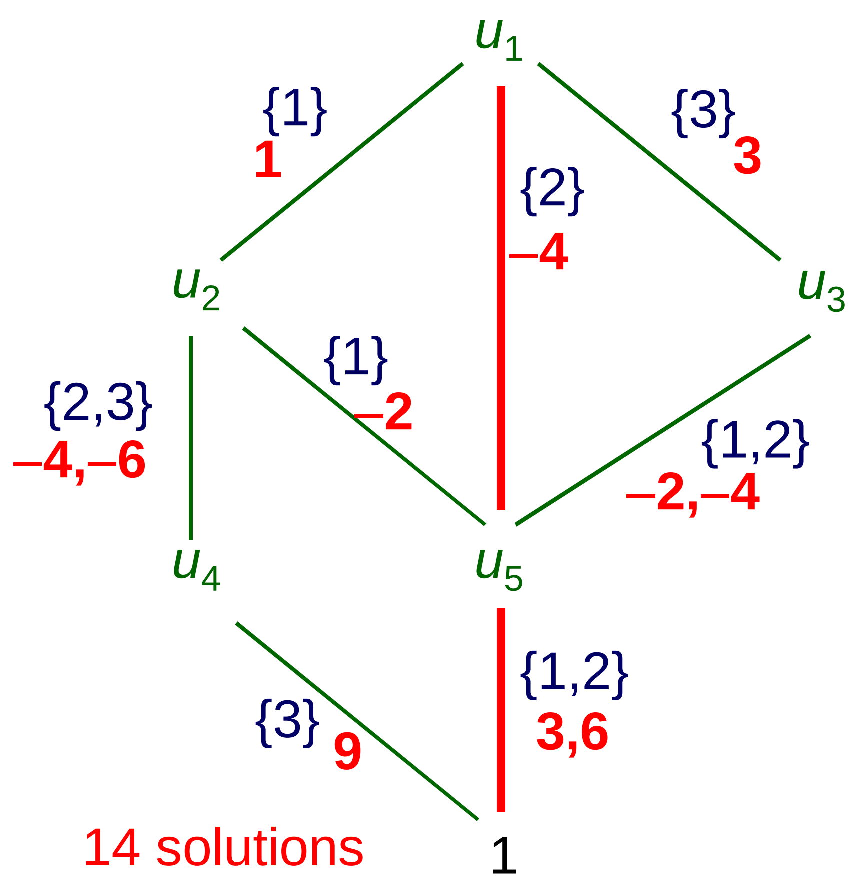
Tightening the MDD Relaxation

min $x_1 - 2x_2 + 3x_3$

Shortest path length = –1

We would like a tighter bound (closer to 2)

Relaxation of width 2



14 solutions

Tightening the MDD Relaxation

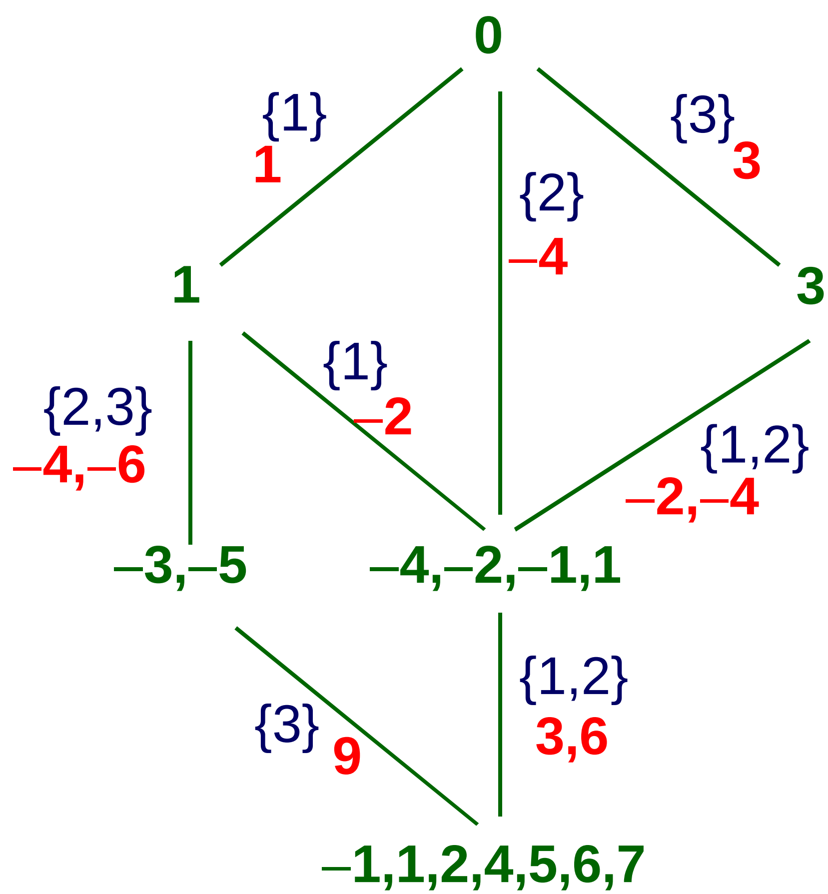
min $x_1 - 2x_2 + 3x_3$

Shortest path length = –1

We would like a tighter bound (closer to 2)

Compute possible path lengths at each node.

Relaxation of width 2



Tightening the MDD Relaxation

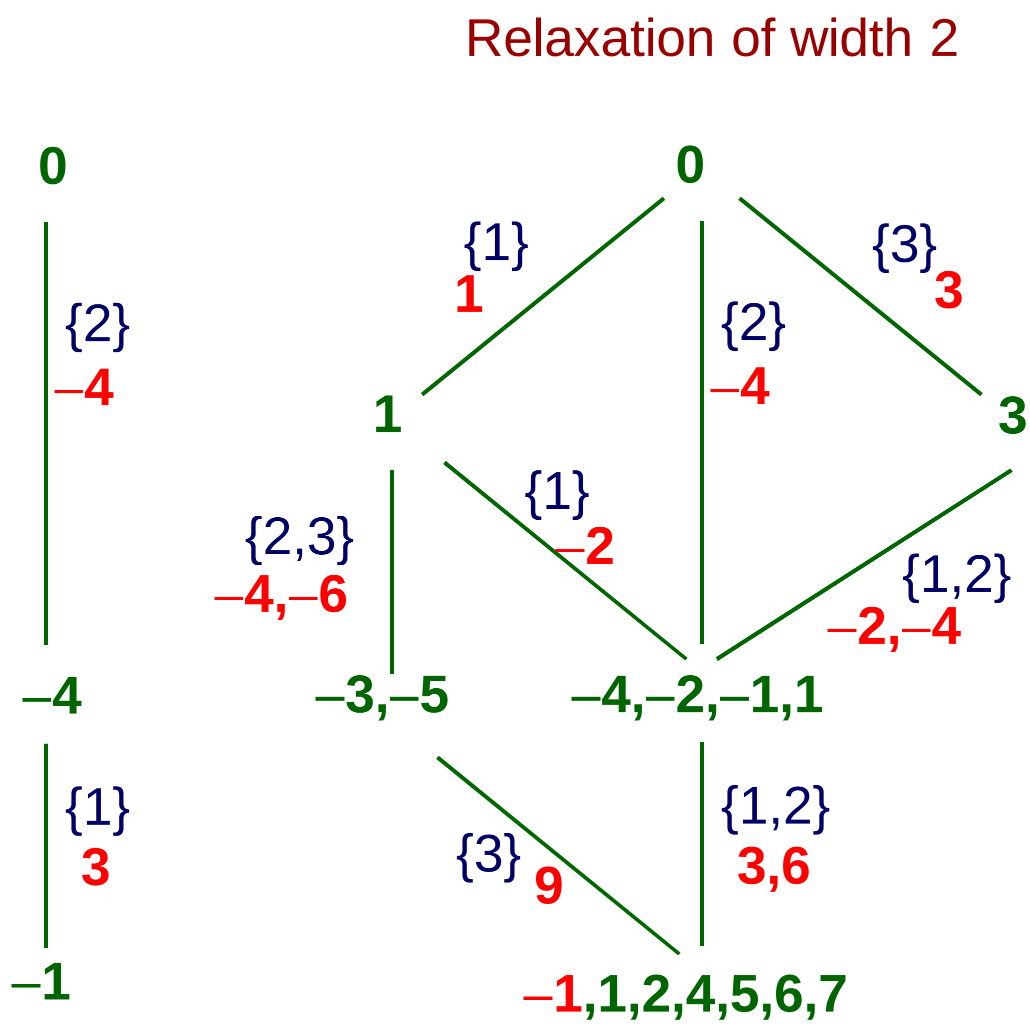
min $x_1 - 2x_2 + 3x_3$

Shortest path length = –1

We would like a tighter bound (closer to 2)

Compute possible path lengths at each node.

Construct MDD of paths with length –1



Tightening the MDD Relaxation

min $x_1 - 2x_2 + 3x_3$

Shortest path length = –1

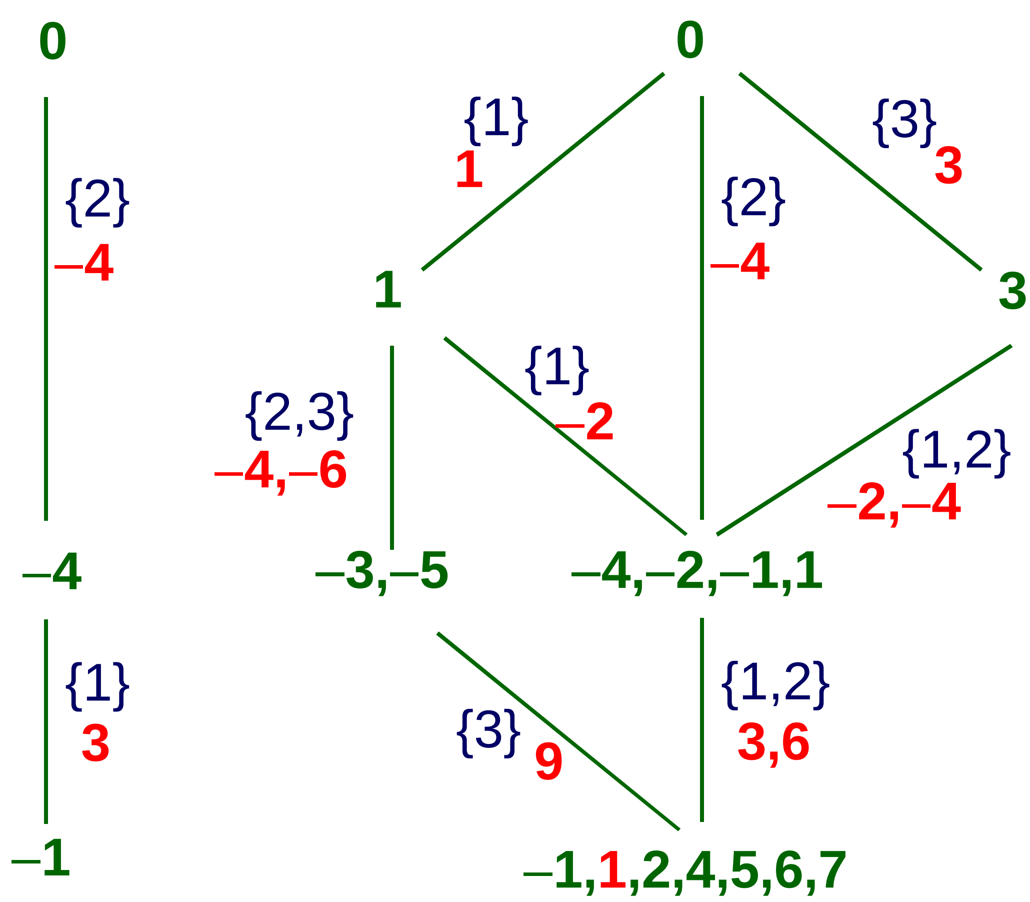
We would like a tighter bound (closer to 2)

Compute possible path lengths at each node.

Construct MDD of paths with length –1

This path is infeasible, so we have new upper bound of 1. Continue.

Relaxation of width 2



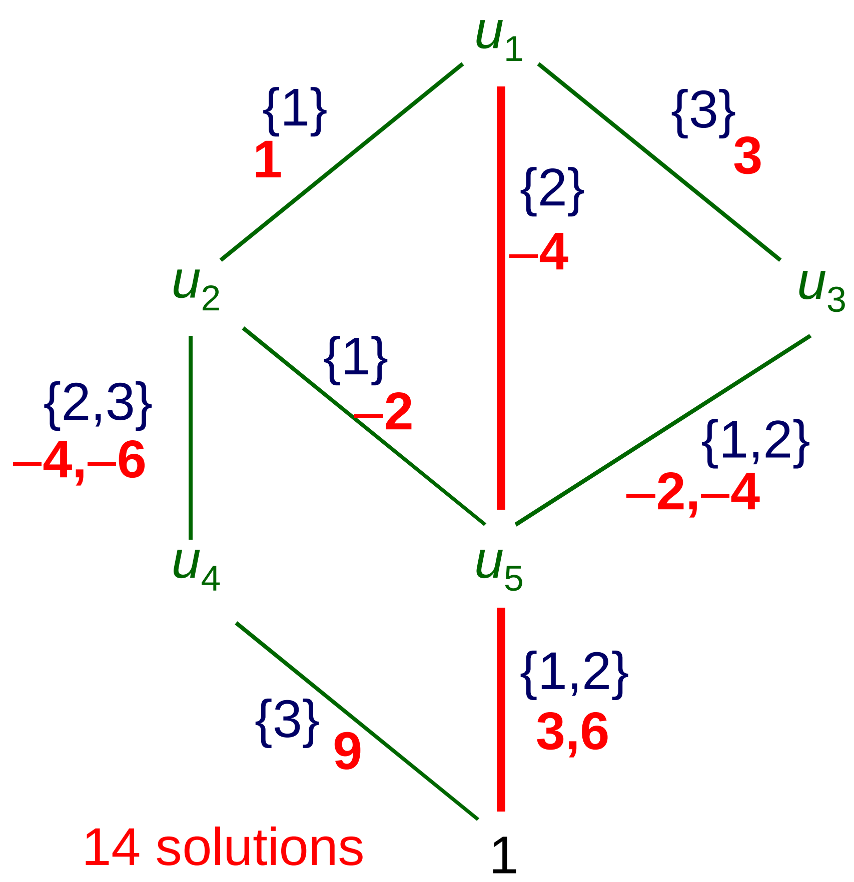
Pseudocosts

min $x_1 - 2x_2 + 3x_3$

Shortest path length = –1

Pseudocost = effect on optimal value of relaxation of fixing a variable.

Relaxation of width 2



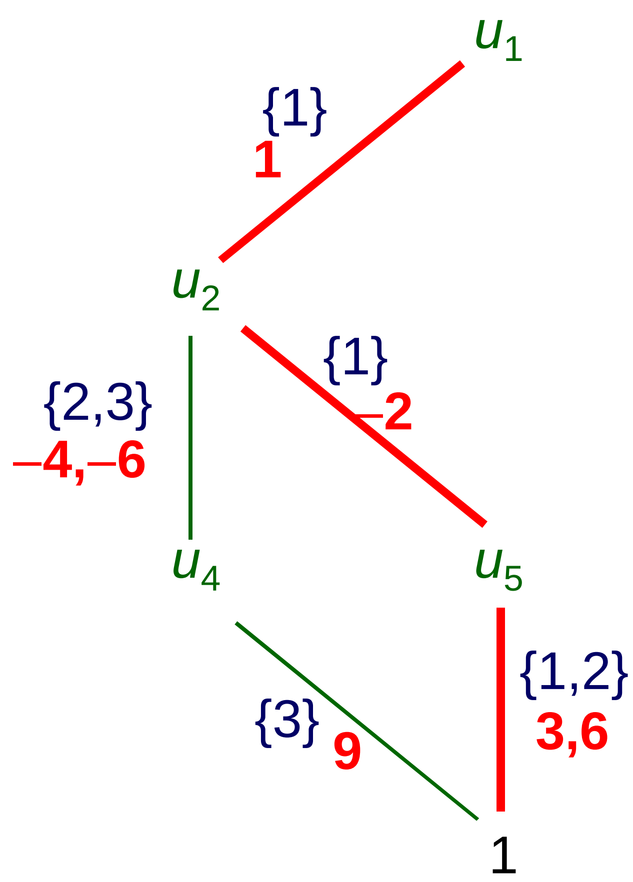
Pseudocosts

min $x_1 - 2x_2 + 3x_3$

Shortest path length = –1

Pseudocost = effect on optimal value of relaxation of fixing a variable.

Branch on $x_1 = 1$.
Optimal value increases by **3** (–1 to 2).



Pseudocosts

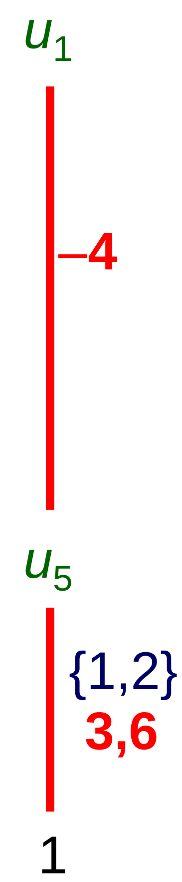
$\min x_1 - 2x_2 + 3x_3$

Shortest path length = –1

Pseudocost = effect on optimal value of relaxation of fixing a variable.

Branch on $x_1 = 1$.
Optimal value increases by **3** (–1 to 2).

Branch on $x_1 = 2$.
Optimal value increases by **0**.



Pseudocosts

min $x_1 - 2x_2 + 3x_3$

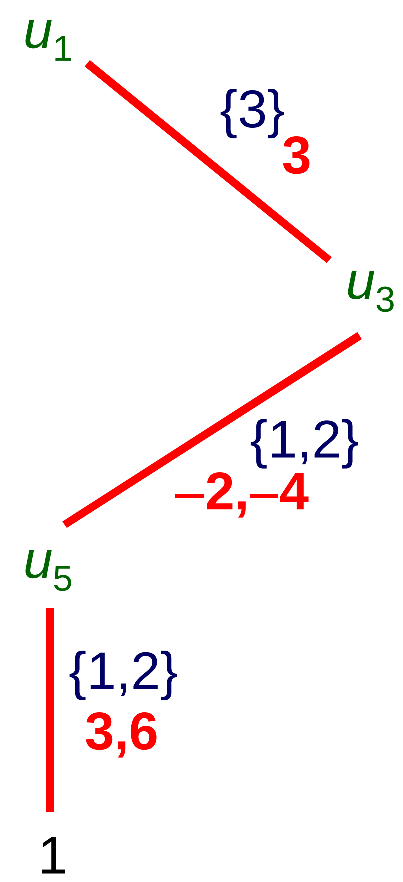
Shortest path length = –1

Pseudocost = effect on optimal value of relaxation of fixing a variable.

Branch on $x_1 = 1$.
Optimal value increases by **3** (–1 to 2).

Branch on $x_1 = 2$.
Optimal values increases by **0**.

Branch on $x_1 = 3$.
Optimal value increases by **3**



Pseudocosts

min $x_1 - 2x_2 + 3x_3$

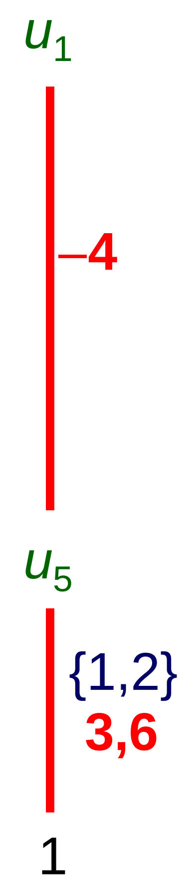
Shortest path length = –1

Pseudocost = effect on optimal value of relaxation of fixing a variable.

Branch on $x_1 = 1$.
Optimal value increases by **3** (–1 to 2).

Branch on $x_1 = 2$.
Optimal values increases by **0**.

Branch on $x_1 = 3$.
Optimal value increases by **3**



MDDs as Restrictions

MDDs as Restrictions

- A **restricted** MDD represents a subset of the feasible solutions
- Restricted MDDs provide a basis for a **feasibility heuristic** in optimization problems.
  - Shortest paths in the restricted MDD provide good feasible solutions.

MDDs as Restrictions

- To generate a restriction of limited width:
  - Create relaxed MDD for **negation** of the constraint and **complement** the MDD
  - Complementation adds at most one to width of the MDD
- Or use a **constraint-specific** algorithm.

To generate a restricted MDD for the set covering problem

$$x_1 + x_2 + x_3 \geq 1$$
$$x_1 + x_4 + x_5 \geq 1$$
$$x_2 + x_4 + x_6 \geq 1$$

Find collection of sets that cover elements A, B, C

Sets

	1	2	3	4	5	6
A	●	●	●			
B	●			●	●	
C		●		●		●

52 feasible solutions.

Minimum cover of 2, e.g. x_1, x_2

To generate a restricted MDD for the set covering problem

$$x_1 + x_2 + x_3 \geq 1$$
$$x_1 + x_4 + x_5 \geq 1$$
$$x_2 + x_4 + x_6 \geq 1$$

Build a relaxed MDD for its negation

$$\left(x_1 + x_2 + x_3 < 1\right)$$
$$\vee \left(x_1 + x_4 + x_5 < 1\right)$$
$$\vee \left(x_2 + x_4 + x_6 < 1\right)$$

using method described earlier.

Find collection of sets that cover elements A, B, C

Sets

	1	2	3	4	5	6
A	●	●	●			
B	●			●	●	
C		●		●		●

52 feasible solutions.

Minimum cover of 2, e.g. x_1, x_2

To generate a restricted MDD for the set covering problem

$$x_1 + x_2 + x_3 \geq 1$$

$$x_1 + x_4 + x_5 \geq 1$$

$$x_2 + x_4 + x_6 \geq 1$$

Build a relaxed MDD for its negation

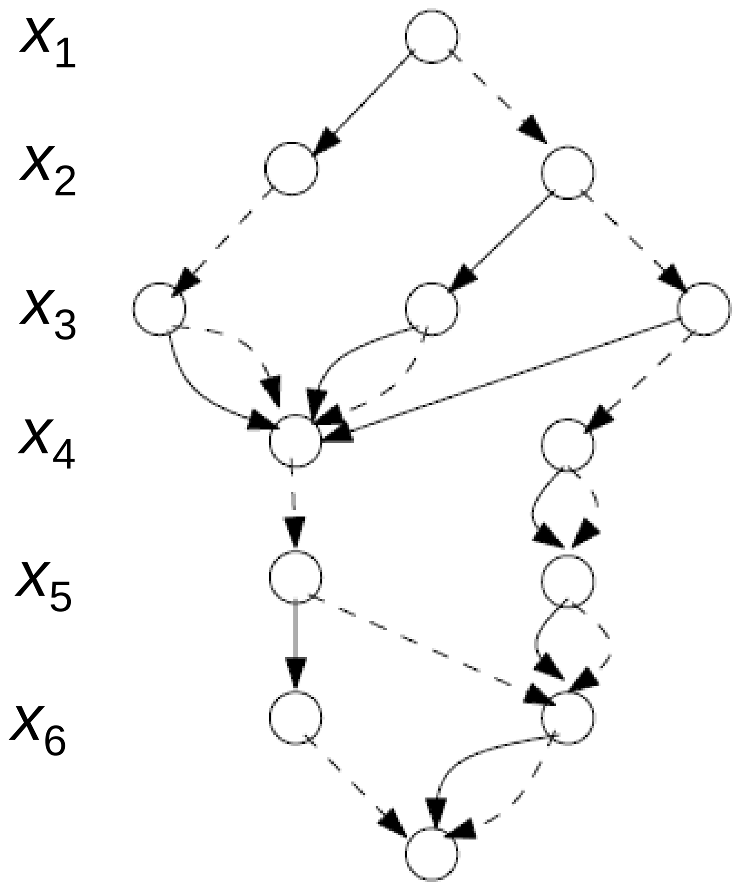
$$(x_1 + x_2 + x_3 < 1)$$

$$\vee (x_1 + x_4 + x_5 < 1)$$

$$\vee (x_2 + x_4 + x_6 < 1)$$

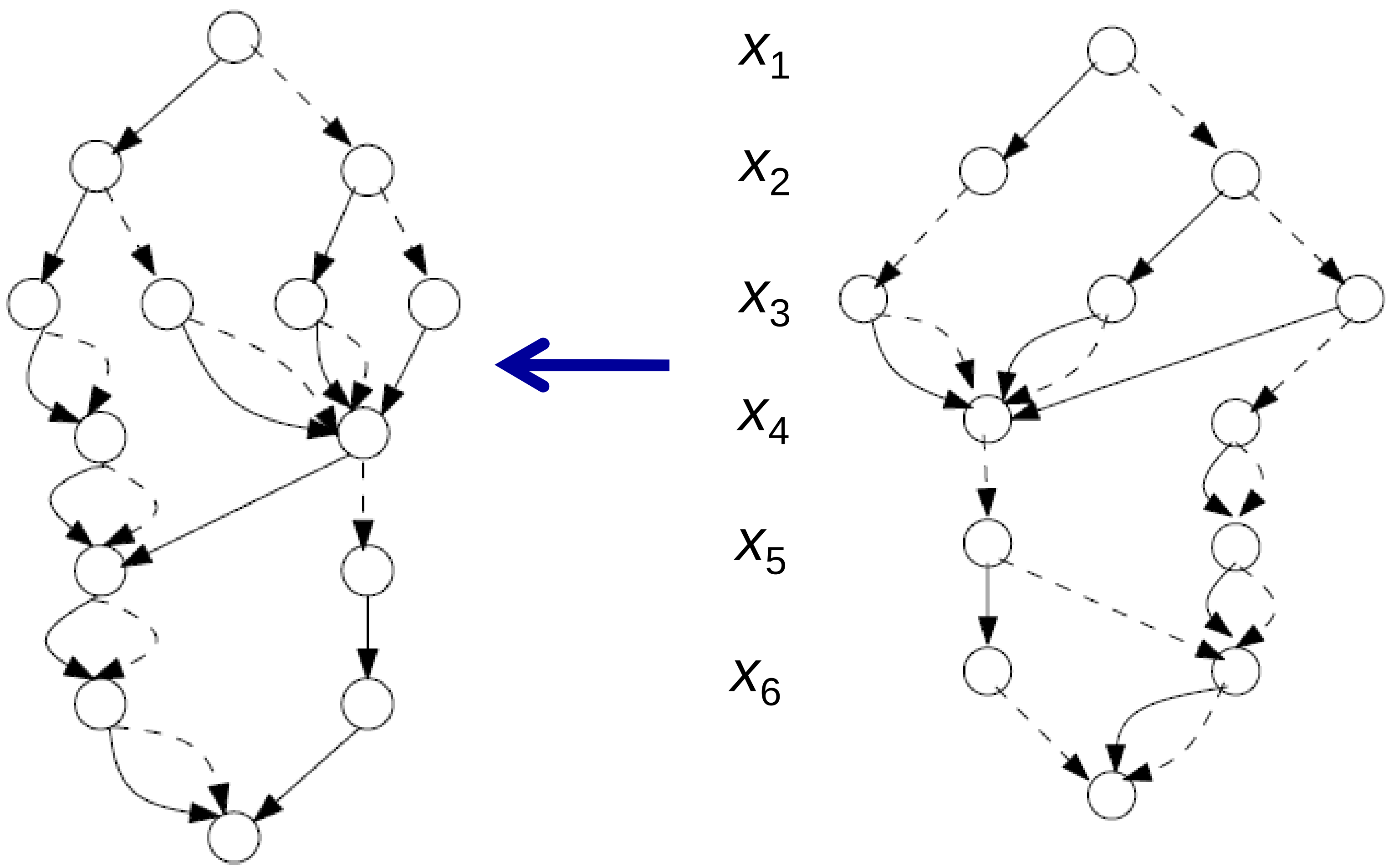
using method described earlier.

Relaxed MDD for negation
Width 3



Complement of relaxed MDD
Width 4

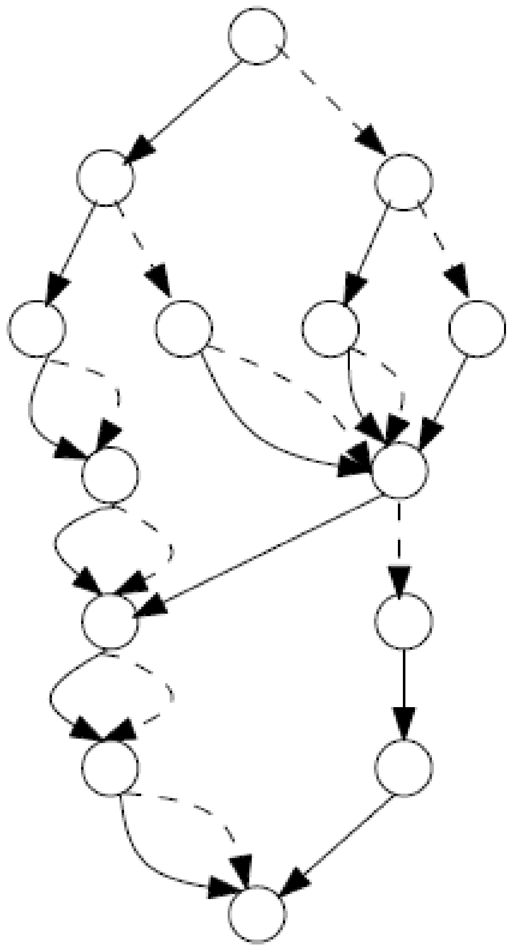
Relaxed MDD for negation
Width 3



This is a restricted MDD for the set covering problem.

41 feasible solutions (< 52)

Complement of relaxed MDD
Width 4

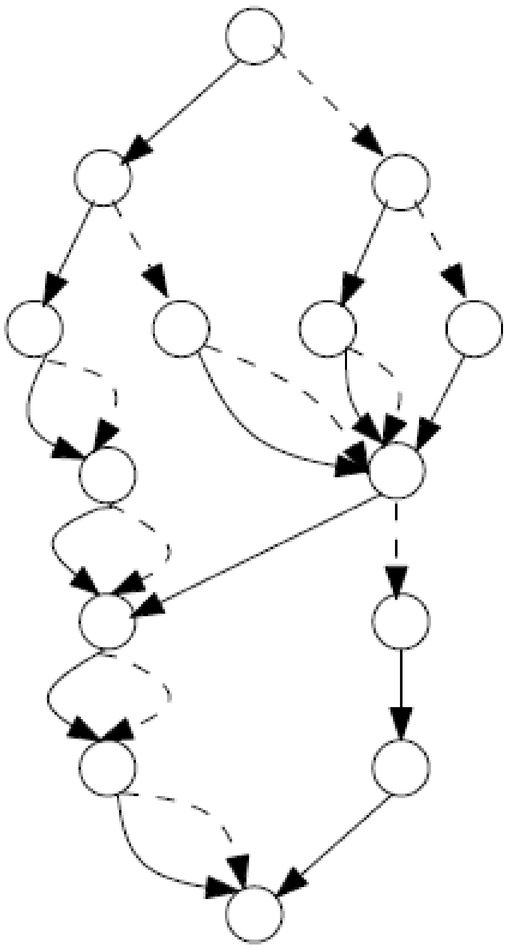


Several shortest paths have length 2.

All are minimum covers.

This is a restricted MDD for the set covering problem.
41 feasible solutions (< 52)

Complement of relaxed MDD
Width 4



Several shortest paths have length 2.

All are minimum covers.

In this case, feasibility heuristic delivers optimal solutions.

This is a restricted MDD for the set covering problem.
41 feasible solutions (< 52)

MDDs as Transparent Data Structures

Transparent Data Structure

- An alternate paradigm for problem solving:
  - Rather than generate a solution, transform the problem to a **more transparent data structure.**
  - MDDs can serve this purpose.

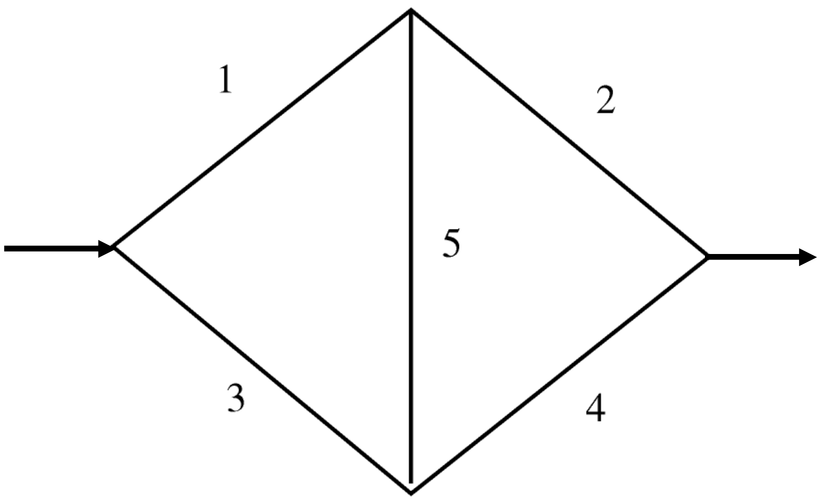
Transparent Data Structure

- An alternate paradigm for problem solving:
  - Rather than generate a solution, transform the problem to a **more transparent data structure.**
  - MDDs can serve this purpose.

- In particular, use MDDs for **postoptimality analysis** / **explanation**.
  - Rapid processing of **queries**.

Postoptimality Analysis

- Reliability networks
  - Minimize cost subject to a bound on reliability
  - System of 5 bridges:



$$R = R_1R_2 + (1-R_2)R_3R_4 + (1-R_1)R_2R_3R_4 + R_1(1-R_2)(1-R_3)R_4R_5 + (1-R_1)R_2R_3(1-R_4)R_5$$

The problem:

Number of links j

$$\min \sum_j c_j x_j$$

$$R \geq R_{\min}$$

$$R = R_1R_2 + (1-R_2)R_3R_4 + (1-R_1)R_2R_3R_4$$
$$+ R_1(1-R_2)(1-R_3)R_4R_5 + (1-R_1)R_2R_3(1-R_4)R_5$$

$$R_j = 1-(1-r_j)^{x_j}, \ \text{all } j$$

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

Set $R_{\min}$ = 60 in all examples

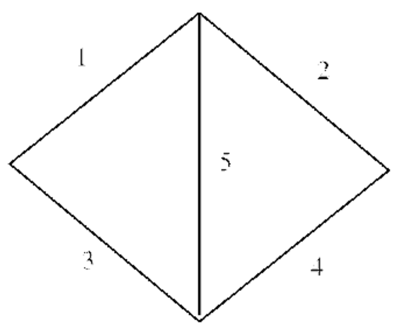
$r = (0.9, 0.85, 0.8, 0.9, 0.95)$

$c = (25, 35, 40, 10, 60)$

Cost-based domain analysis

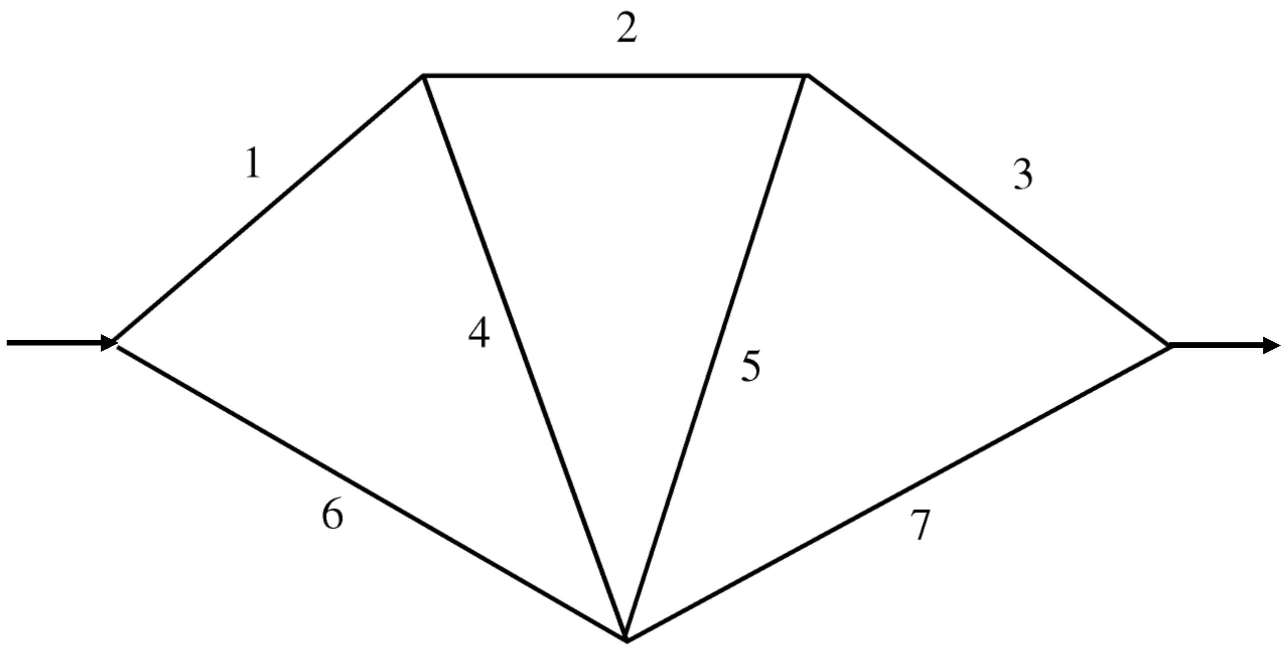
89 nodes in MDD

1.2 seconds to compile MDD



$c_{opt} + \Delta$	x_1	x_2	x_3	x_4	x_5	R
50:	0	0	1	1	0	72
60:	1	1	0	0,2		79
85:	2					84
90:			2	3		86
95:		2			1	88
100:						95
120:						97
125:	3					
155:		3			2	
160:						98
170:						99
180:			3			
230:					3	

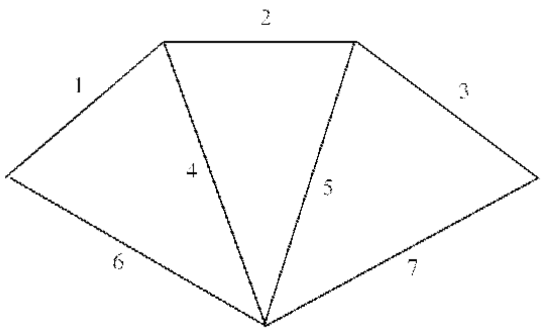
7 bridges



Cost-based domain analysis

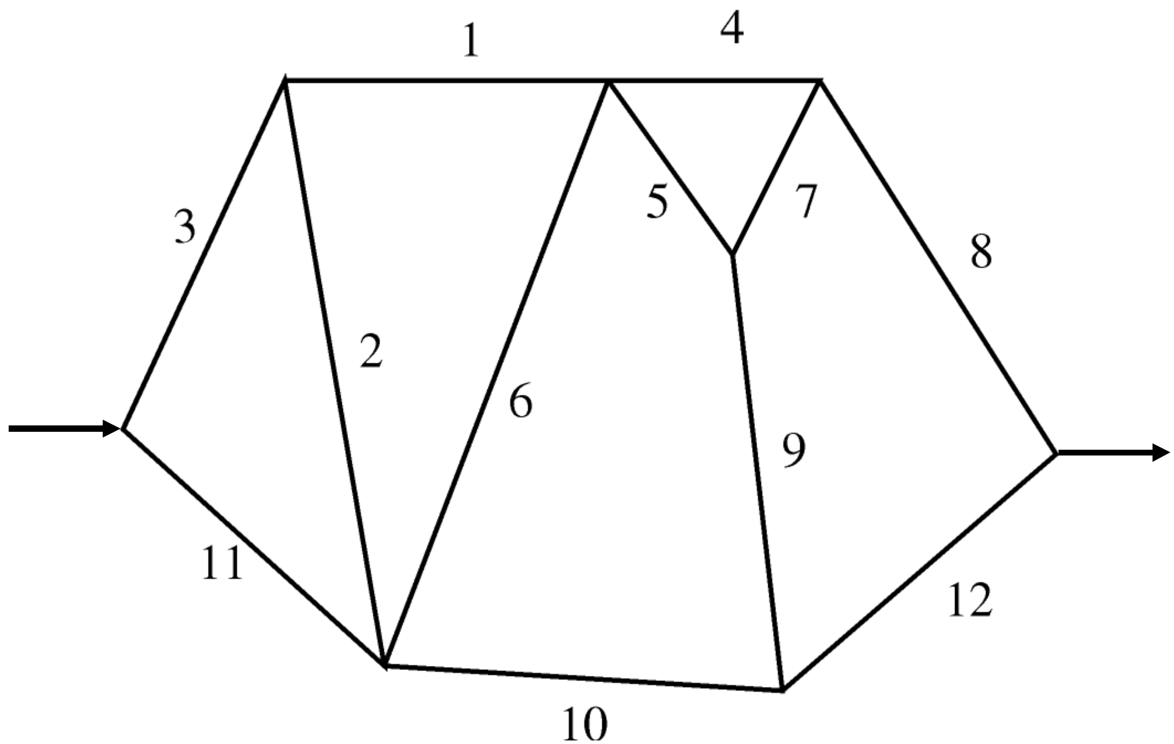
3126 nodes in MDD

9.0 seconds to compile MDD



$c_{opt} + \Delta$	x_1	x_2	x_3	x_4	x_5	x_6	x_7	R
9:	0	0	0	0	0	1	1	72.2
11:			1		1		0	
12:	1			1		0		
13:		1				2		82.9
14:					2		2	
15:			2	2				
16:	2							
17:					3	3		84.6
18:		2		3				95.2
19:			3				3	
20:	3							
22:								97.2
23:		3						
27:								99.2
34:								99.4
40:								99.6
43:								99.7
47:								99.8
54:								99.9

12 bridges

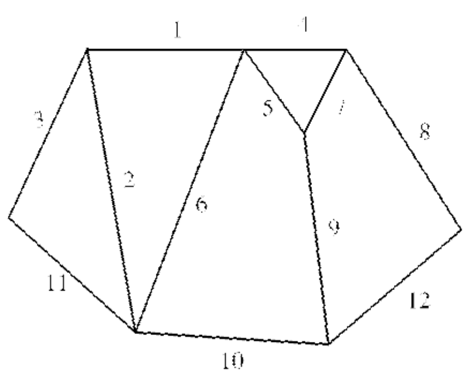


Cost-based
domain analysis

36,301 nodes in MDD

1980 seconds
to compile MDD

1.8 seconds query
time to build this table



$c_{opt} + \Delta$	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	R
180	1	0	2	3	0	0	0	2	0	0	0	0	80
185			3	2									82
190								3					
195													83
200										1			
205													86
210	2						1						
215													88
220										2			
225					1				1				
230	0		0			1,2					1		
235			1										
240		1		1			2			3	2	1	
250				0				0,1				2	
255													91
260	3								2				
265													93
270					2	3	3						
290											3		
300		2											
305									3				
310												3	
315					3								94
340													95
360		3											
365													96
380													97
430													98
485													99

Cost-Bounded MDDs

Cost-Bounded MDDs

- **Original** MDD
  - Represents entire feasible set
  - Can be very large.

Cost-Bounded MDDs

- **Original** MDD
  - Represents entire feasible set
  - Can be very large.
- **Near-optimal** MDD
  - Exactly represents solutions within a given distance from optimal value.
  - Can be even larger than the original MDD!

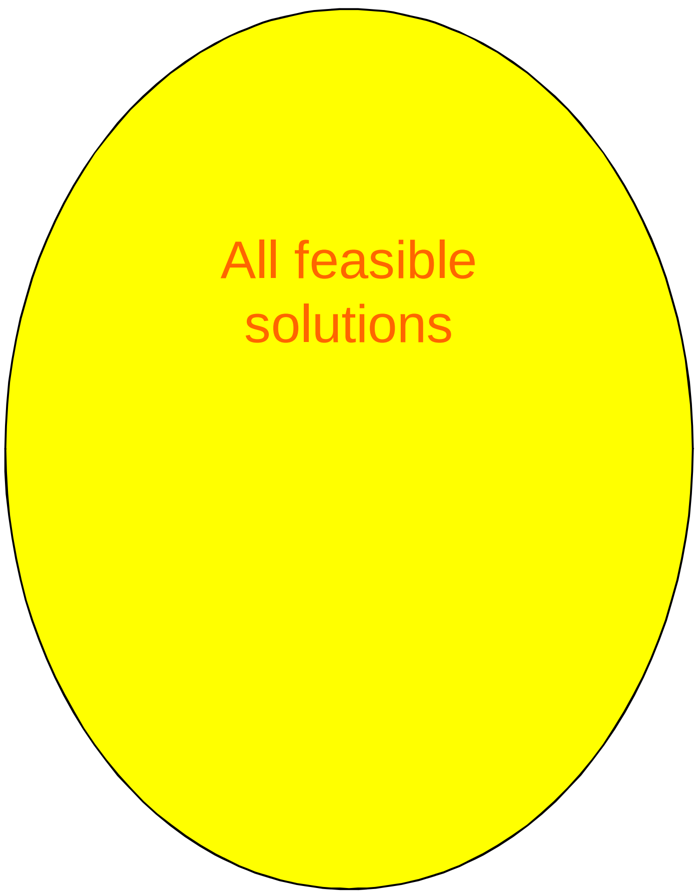
Cost-Bounded MDDs

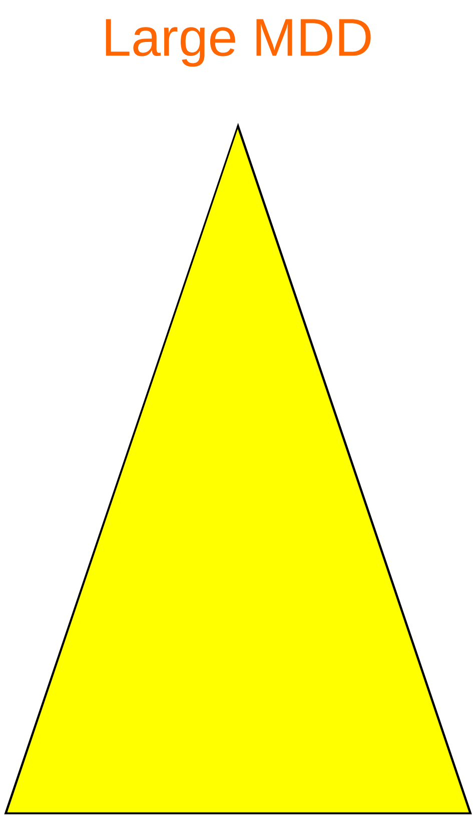
- **Original** MDD
  - Represents entire feasible set
  - Can be very large.

- **Near-optimal** MDD
  - Exactly represents solutions within a given distance from optimal value.
  - Can be even larger than the original MDD!

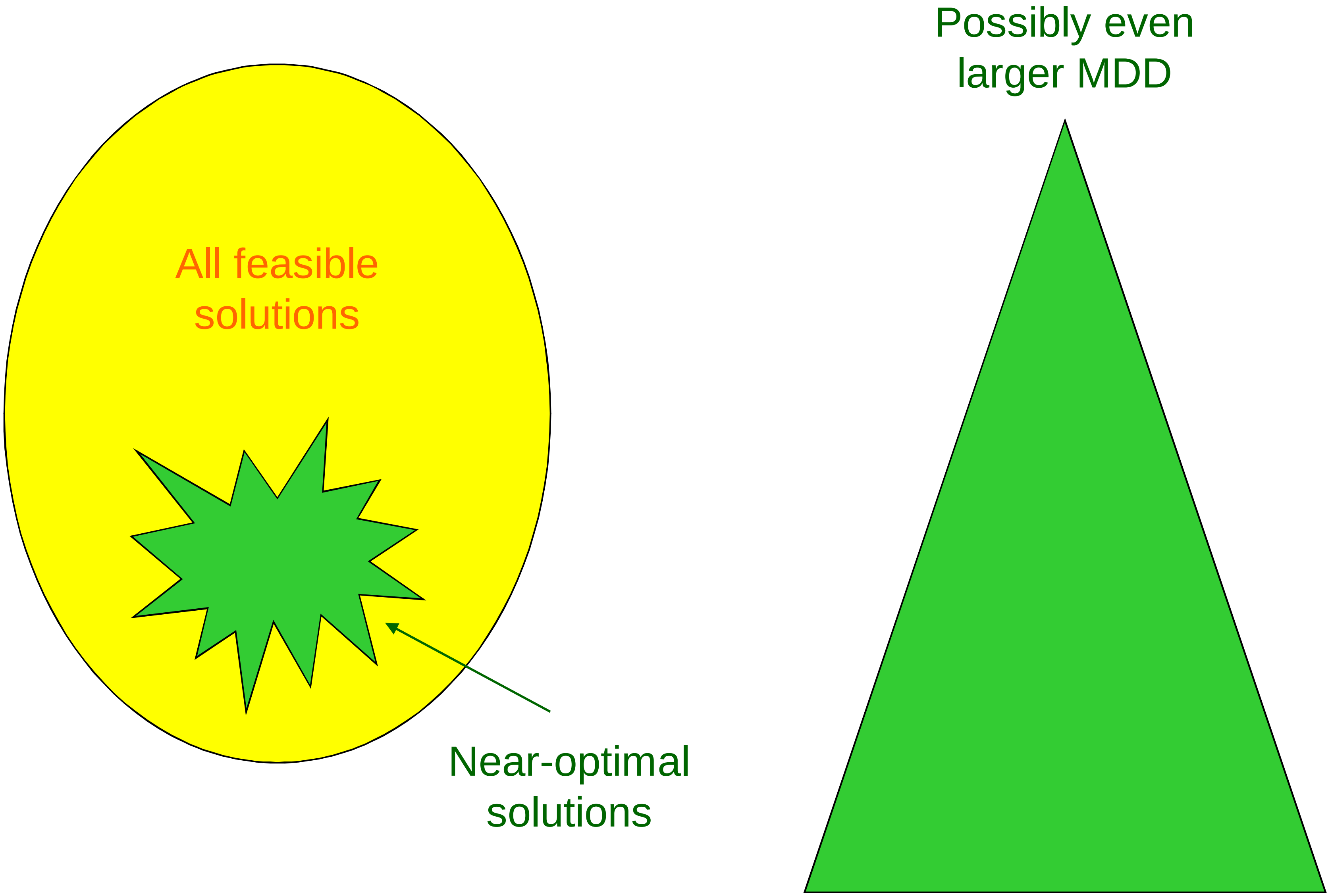
- **Compressed** MDD.
  - Includes all near-optimal solutions, plus some others.
  - Much smaller than near-optimal MDD.
  - Constructed by pruning and contraction.

Cost-Bounded MDDs

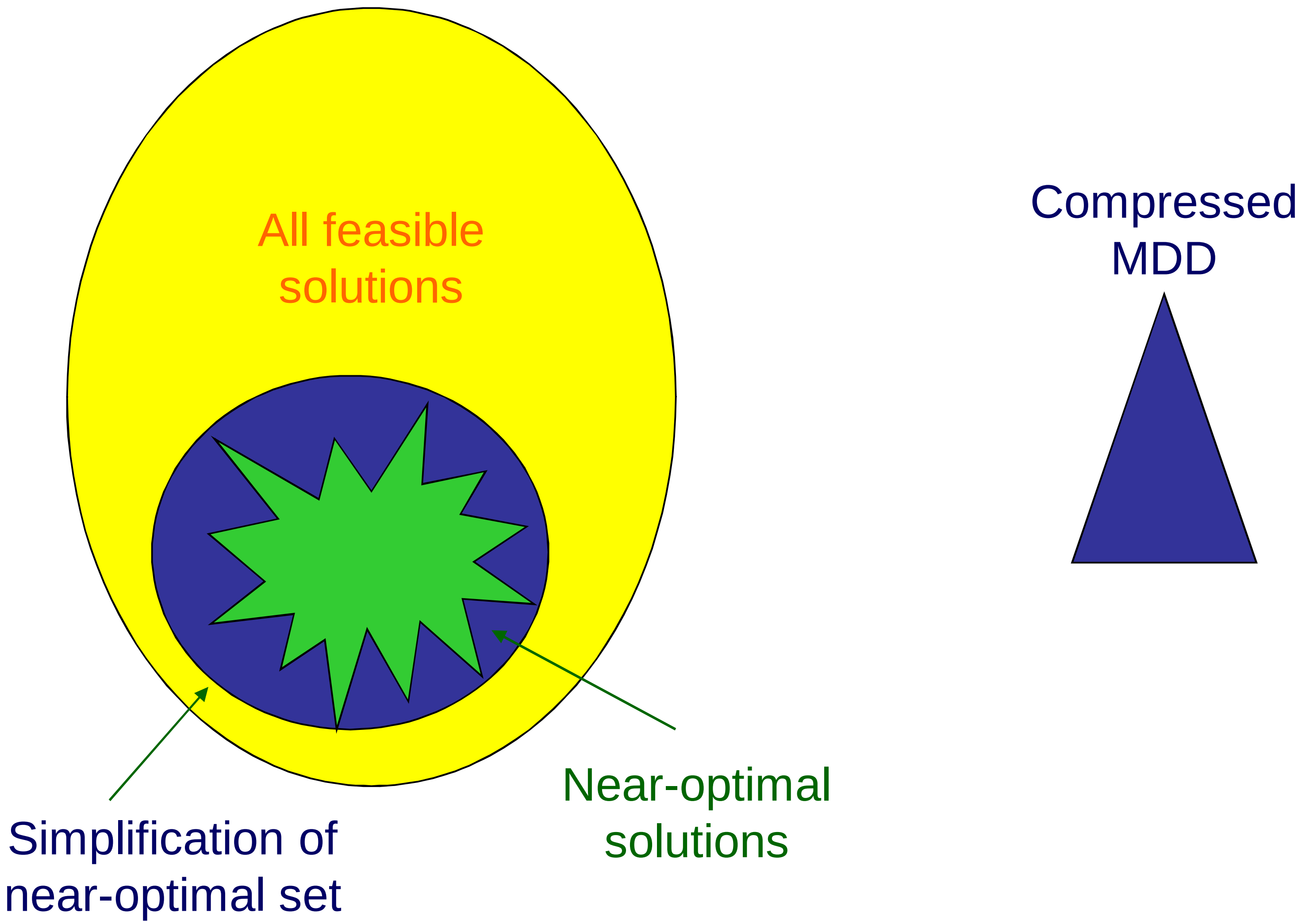




Cost-Bounded MDDs



Cost-Bounded MDDs

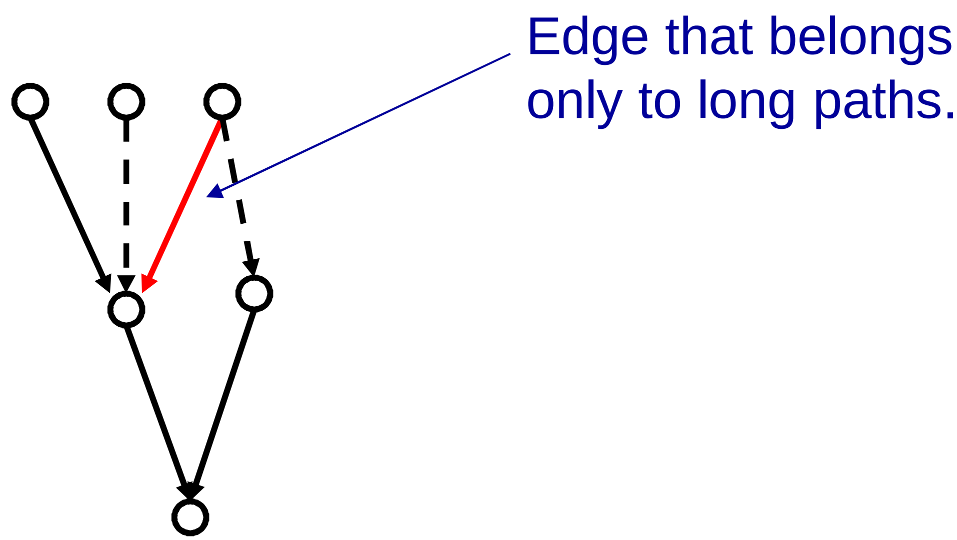


Pruning and Contracting

- Heuristic methods for generating **small compressed** MDDs when conjoining constraints:
  - Pruning edges
  - Contracting nodes

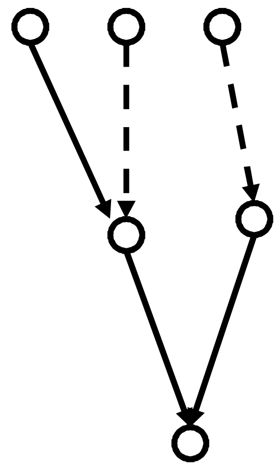
Pruning

- Delete all edges that belong only to paths longer than optimal value + tolerance.



Pruning

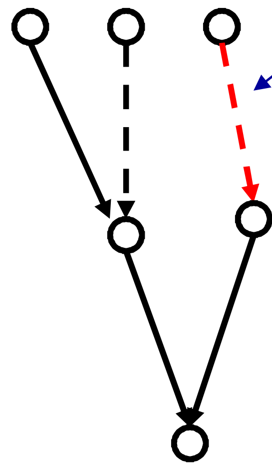
- Delete all edges that belong only to paths longer than optimal value + tolerance.



Pruning

- Delete all edges that belong only to paths longer than optimal value + tolerance.

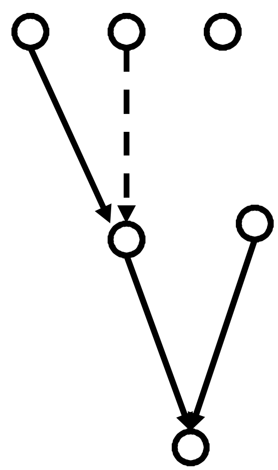
If another edge now belongs only to long paths



Pruning

- Delete all edges that belong only to paths longer than optimal value + tolerance.

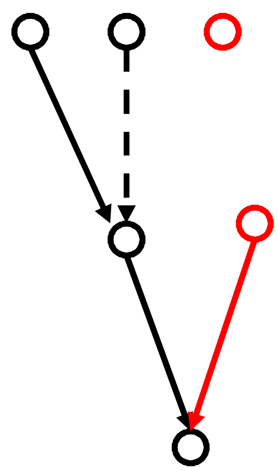
Delete it, too.



Pruning

- Delete all edges that belong only to paths longer than optimal value + tolerance.

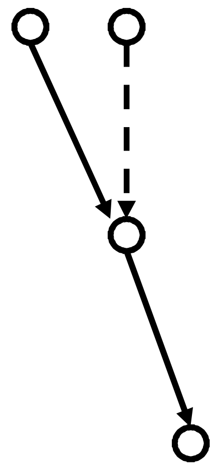
And simplify the BDD.



Pruning

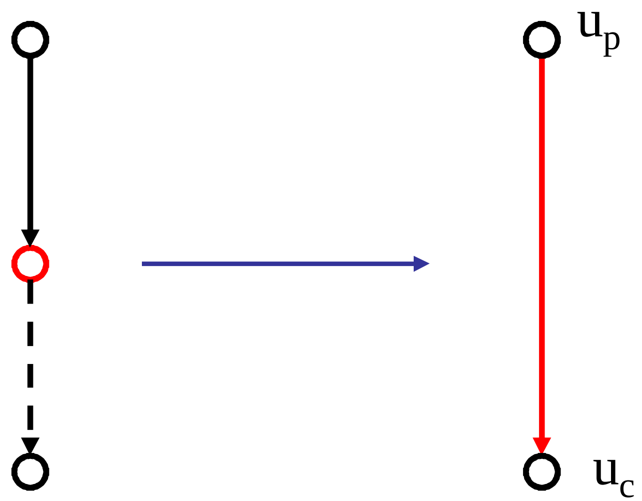
- Delete all edges that belong only to paths longer than optimal value + tolerance.

And simplify the BDD.



Contracting

- Remove a node if this creates no new paths shorter than optimal value + tolerance.



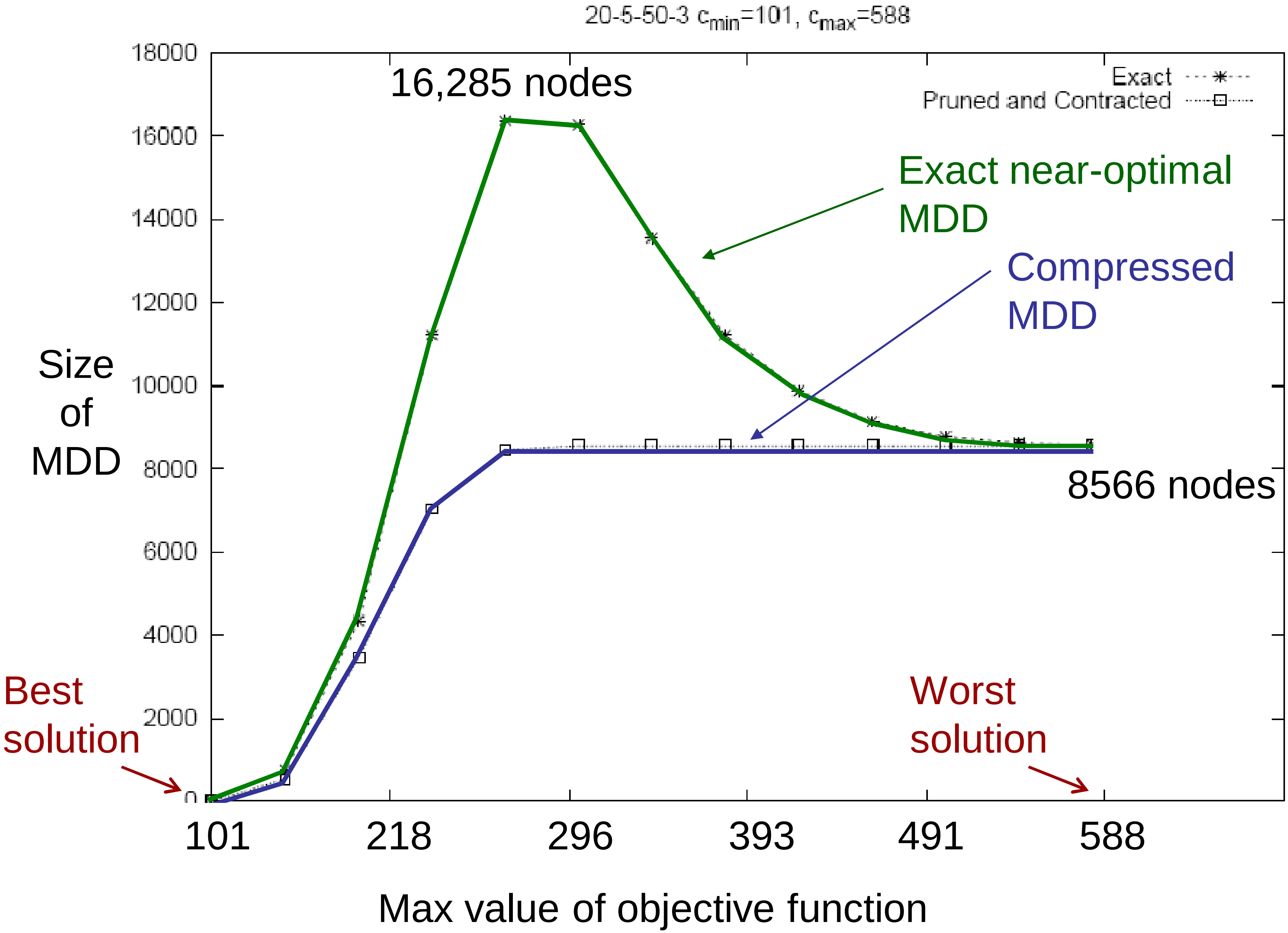
Experimental Results

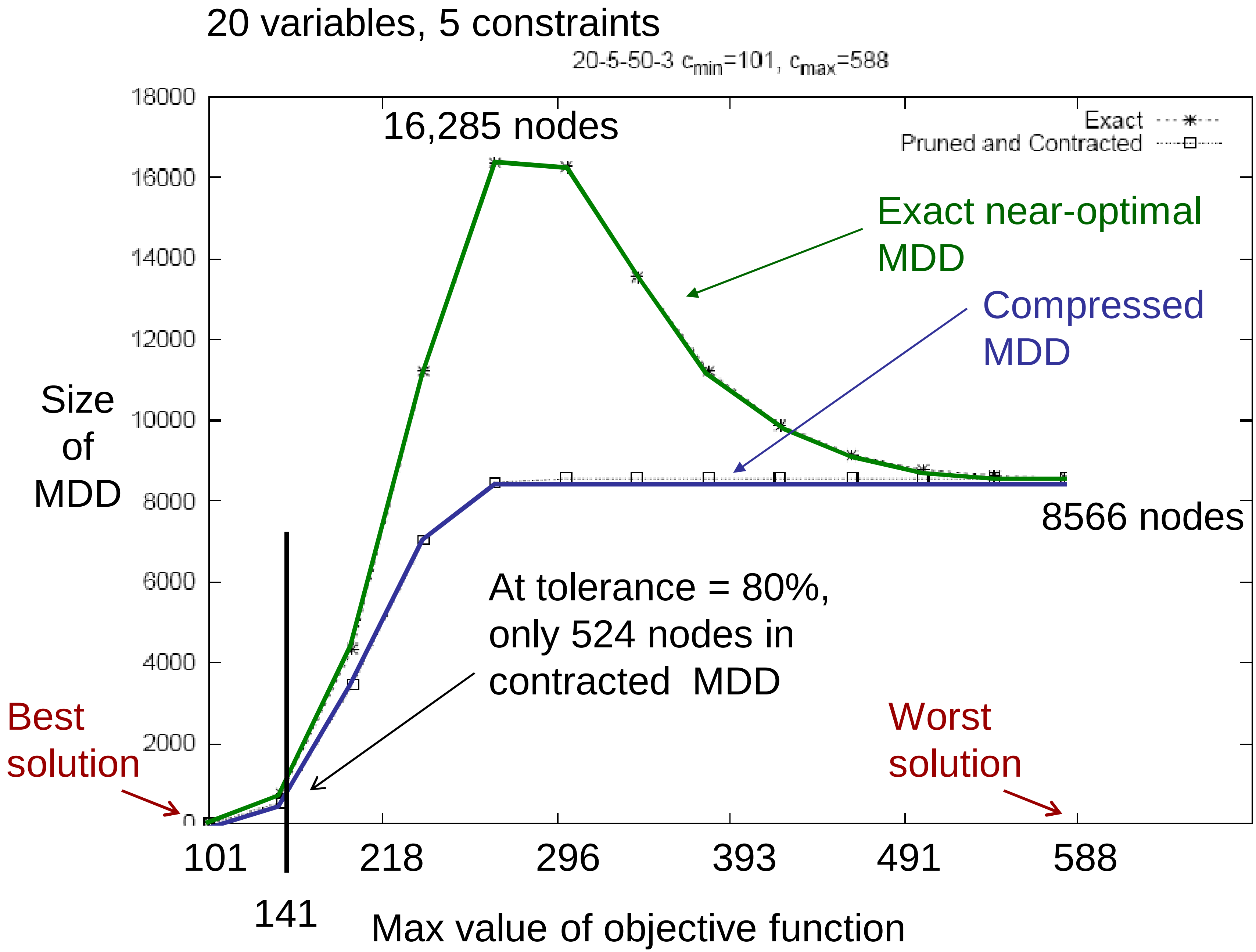
- Solve the 0-1 problem

$$\min cx$$
$$Ax \geq b$$
$$x \in \{0,1\}^n$$

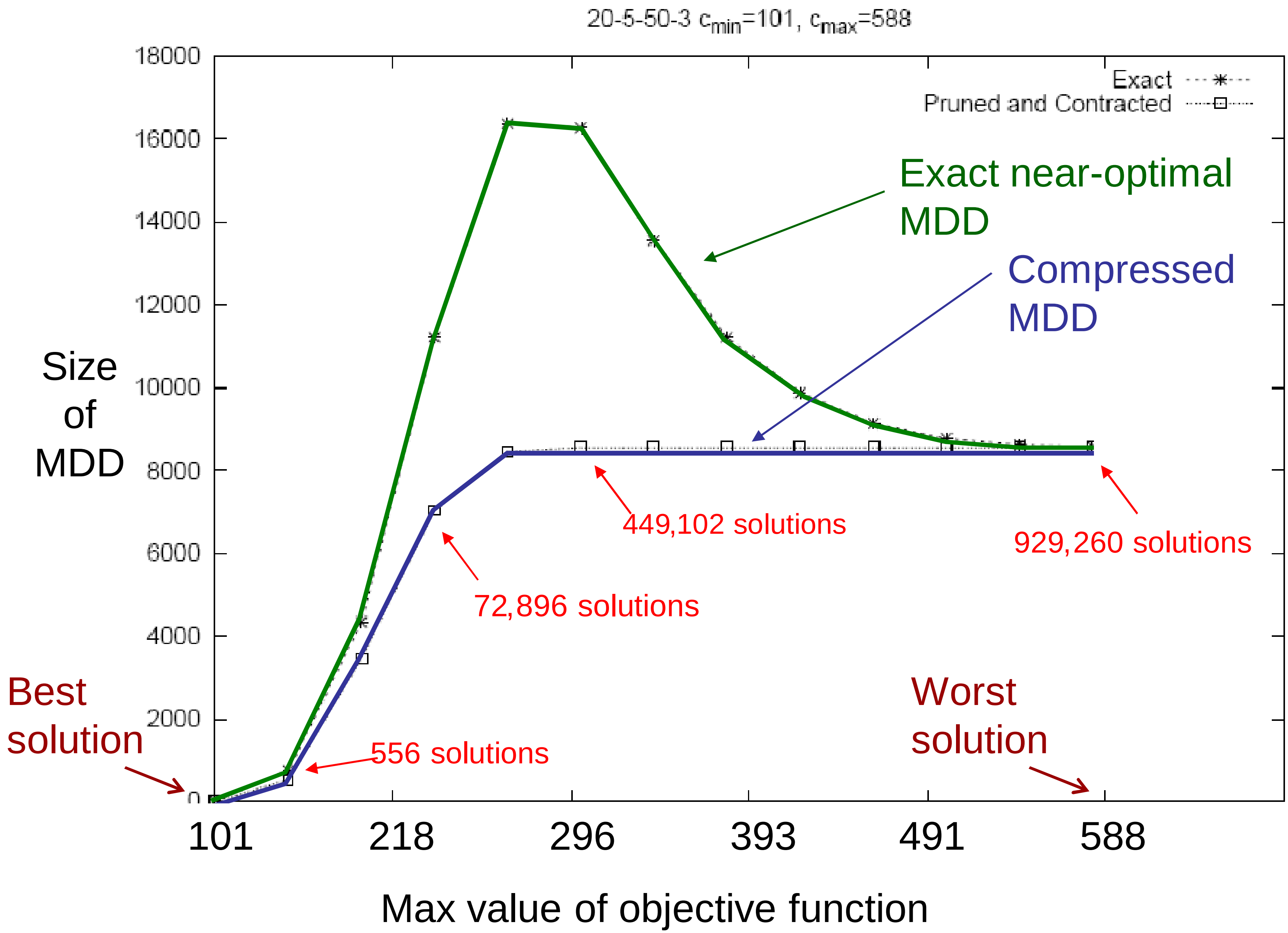
A_{ij} drawn uniformly from $[0,r]$

20 variables, 5 constraints





20 variables, 5 constraints
20-5-50-3 c_{min}=101, c_{max}=588
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
16,285 nodes
Exact
Pruned and Contracted
Exact near-optimal MDD
Compressed MDD
Size of MDD
8566 nodes
At tolerance = 80%, only 524 nodes in contracted MDD
Best solution
Worst solution
101
218
296
393
491
588
141
Max value of objective function



20-5-50-3 c_{min}=101, c_{max}=588
Exact
Pruned and Contracted
Exact near-optimal MDD
Compressed MDD
Size of MDD
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
449,102 solutions
929,260 solutions
72,896 solutions
556 solutions
Best solution
Worst solution
101
218
296
393
491
588
Max value of objective function

Experimental results

- MDD representation of all **optimal** solutions
- MIPLIB instances

MIPLIB instance	Size of **exact** MDD for all **feasible** solutions	Size of **exact** MDD for all **optimal** solutions	Size of **compressed** MDD for all **optimal** solutions
lseu	?	99	19
p0033	375	41	21
p0201	310,420	737	84
stein27	25,202	6260	4882
stein45	5,102,257	1765	1176

Nonserial MDDs and Dynamic Programming

Nonserial Dynamic Programming

- Independently(?) rediscovered many times:
  - Nonserial DP (1972)
  - Constraint satisfaction (1981)
  - Data base queries (1983)
  - *k*-trees (1985)
  - Belief logics (1986)
  - Bucket elimination (1987)
  - Bayesian networks (1988)
  - Pseudoboolean optimization (1990)
  - Location analysis (1994)

Set Partitioning example

Find collection of sets that partition elements A, B, C, D

Sets

	1	2	3	4	5	6
A	●	●	●			
B		●		●		
C			●		●	●
D				●		●

Set Partitioning example

Find collection of sets that partition elements A, B, C, D

Sets

	1	**2**	**3**	**4**	**5**	**6**
A	●	●	●			
B		●		●		
C			●		●	●
D				●		●

For example…

Set Partitioning example

Find collection of sets that partition elements A, B, C, D

Sets

	1	2	3	4	5	6
A	●	●	●			
B		●		●		
C			●		●	●
D				●		●

Or…

Set Partitioning example

Find collection of sets that partition elements A, B, C, D

Sets

	1	2	3	4	5	6
A	•	•	•			
B		•		•		
C			•		•	•
D				•		•

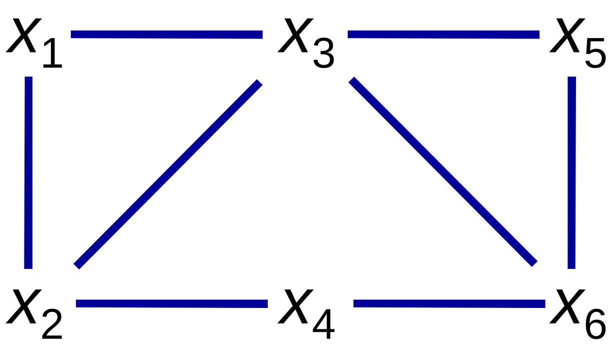
0-1 formulation

$$
\begin{aligned}
x_1 + x_2 + x_3 \quad\quad\quad\quad &= 1 \\
x_2 \quad\quad + x_4 \quad\quad\quad &= 1 \\
x_3 \quad\quad + x_5 + x_6 &= 1 \\
x_4 \quad\quad + x_6 &= 1
\end{aligned}
$$

$x_j = 1 \quad \Rightarrow \quad$ set j selected

Set Partitioning example

Dependency graph



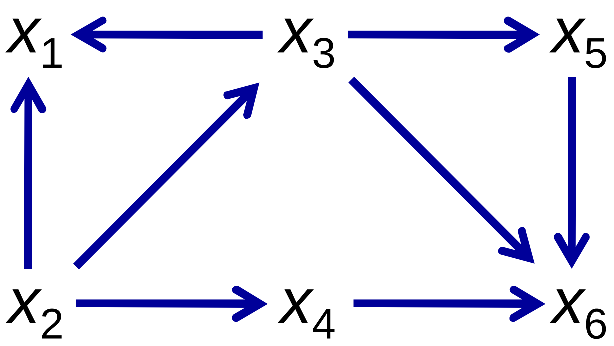
0-1 formulation

$$
\begin{aligned}
x_1 + x_2 + x_3 \qquad\qquad\qquad &= 1 \\
x_2 \quad + x_4 \qquad\qquad &= 1 \\
x_3 \quad + x_5 + x_6 &= 1 \\
x_4 \quad + x_6 &= 1
\end{aligned}
$$

$x_j = 1 \quad \Rightarrow \quad$ set j selected

Set Partitioning example

Dependency graph



Enumeration order

x_2

x_3

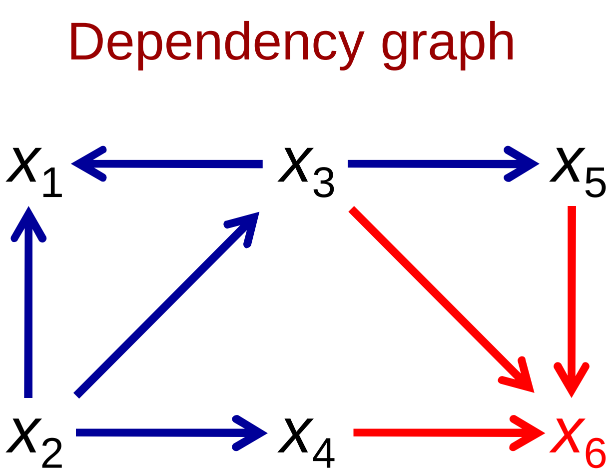
x_4

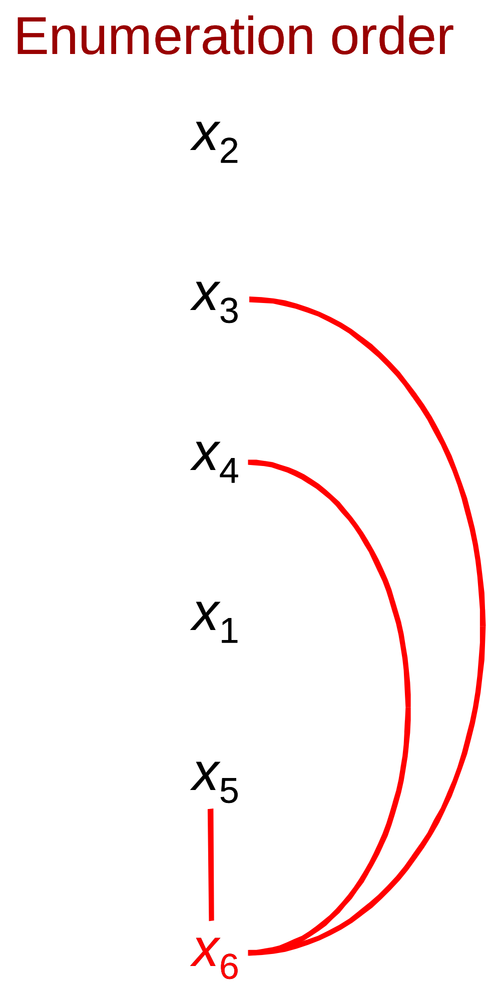
x_1

x_5

x_6

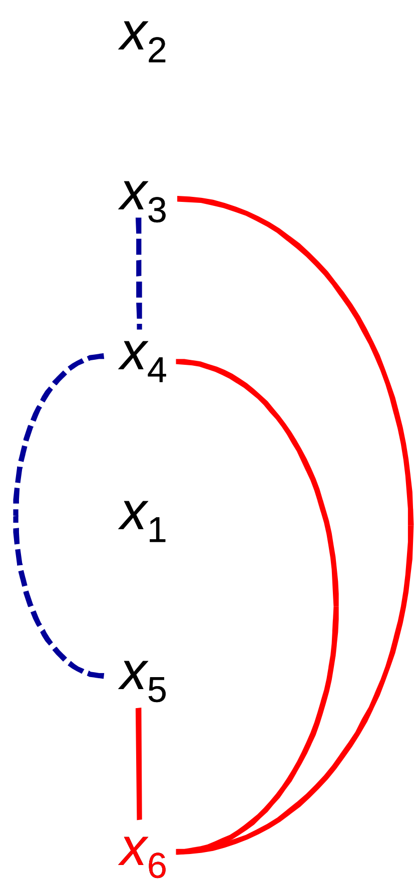
Set Partitioning example



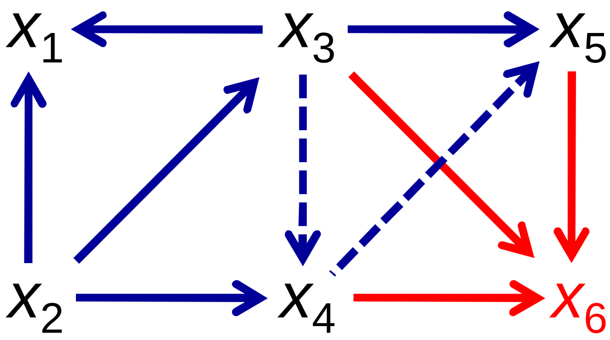


Set Partitioning example

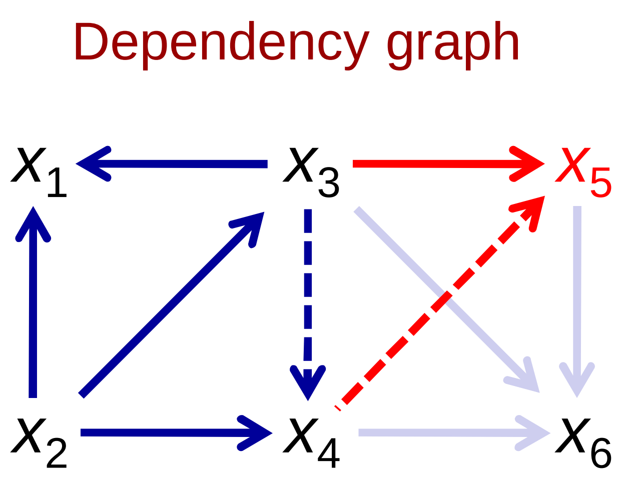
Enumeration order

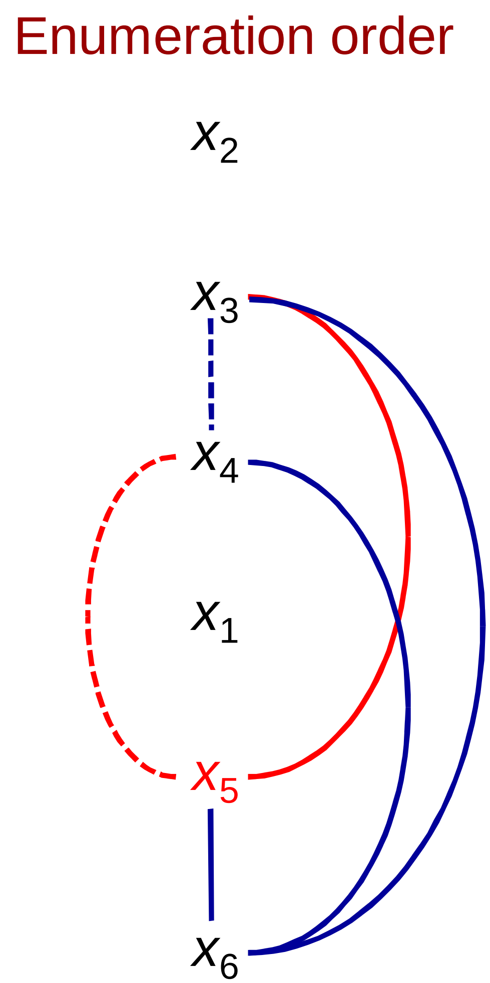


Dependency graph



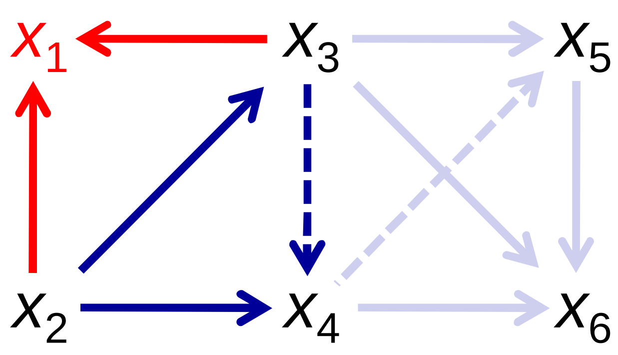
Set Partitioning example



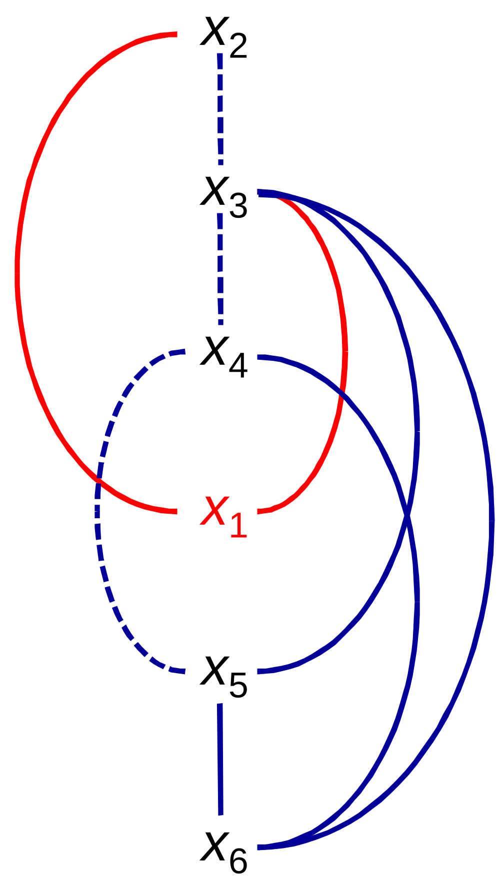


Set Partitioning example

Dependency graph

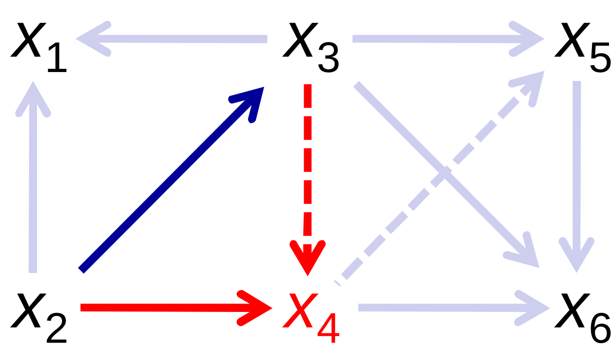


Enumeration order

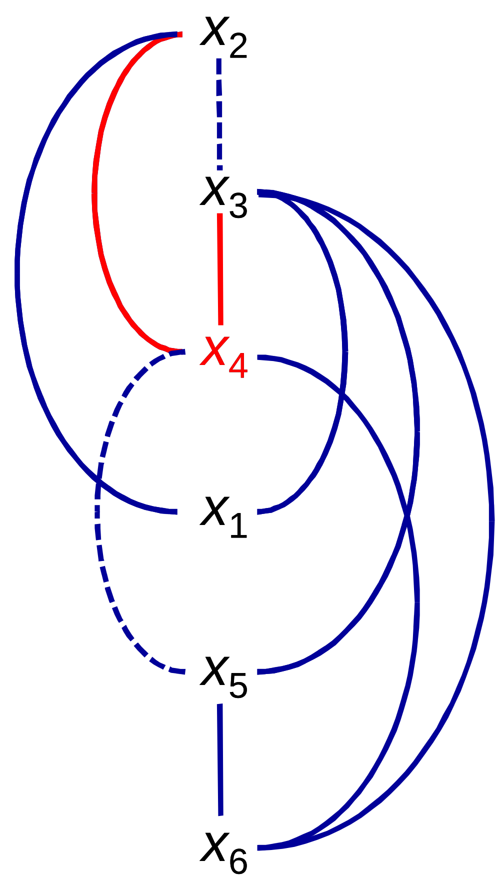


Set Partitioning example

Dependency graph

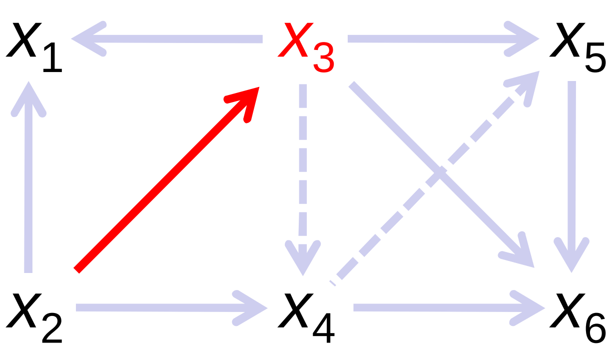


Enumeration order

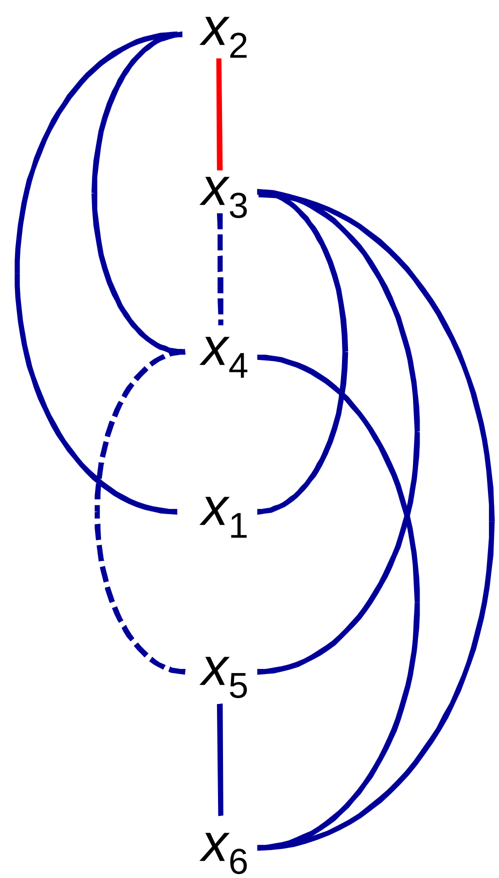


Set Partitioning example

Dependency graph

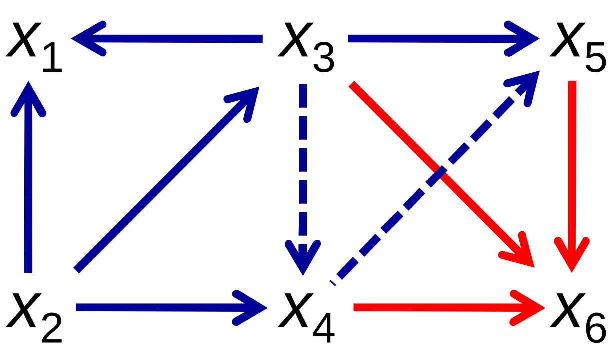


Enumeration order



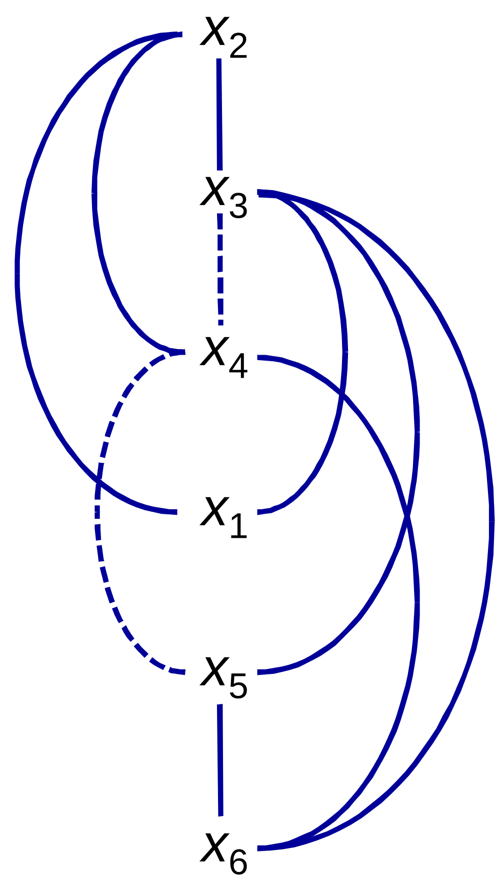
Set Partitioning example

Dependency graph



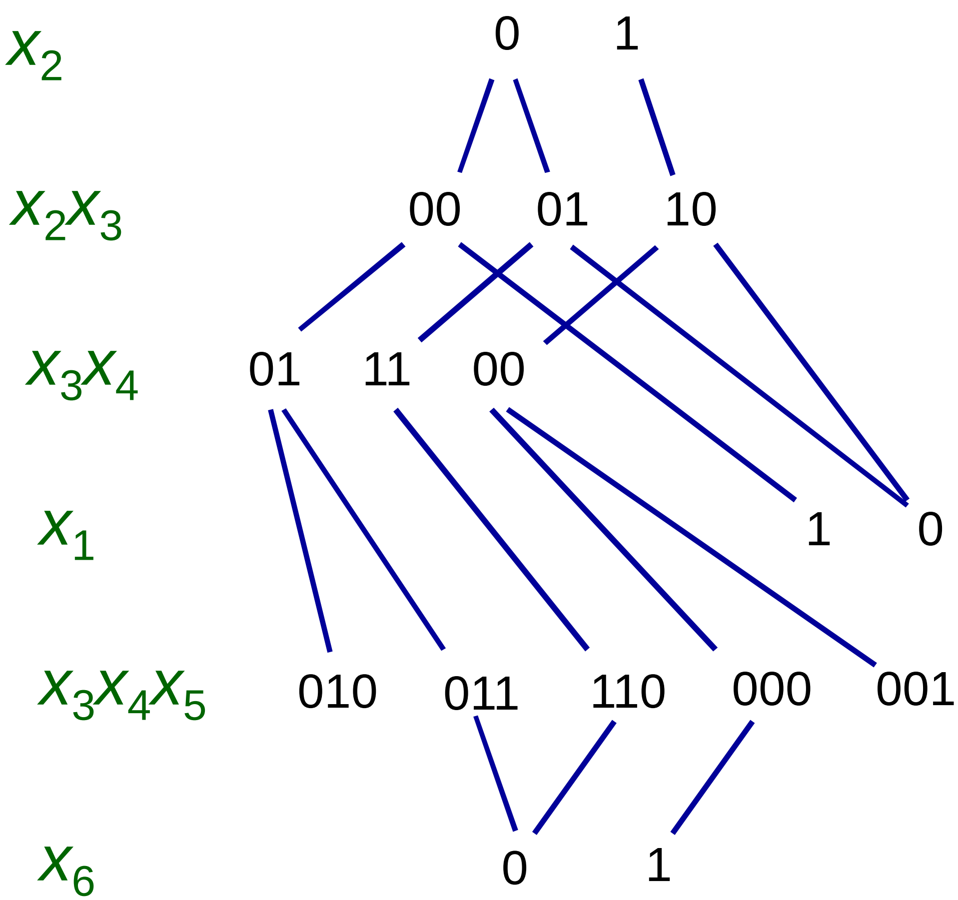
Induced width = 3
(max in-degree)

Enumeration order

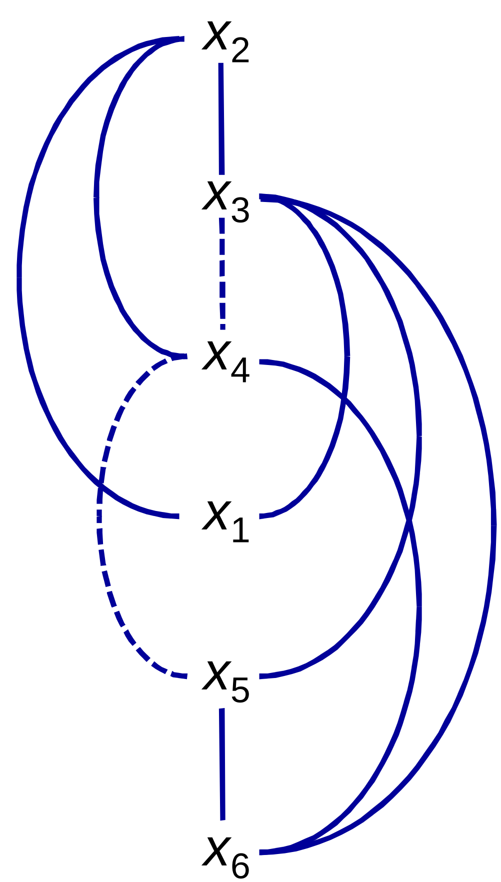


Set Partitioning example

Solution by nonserial DP

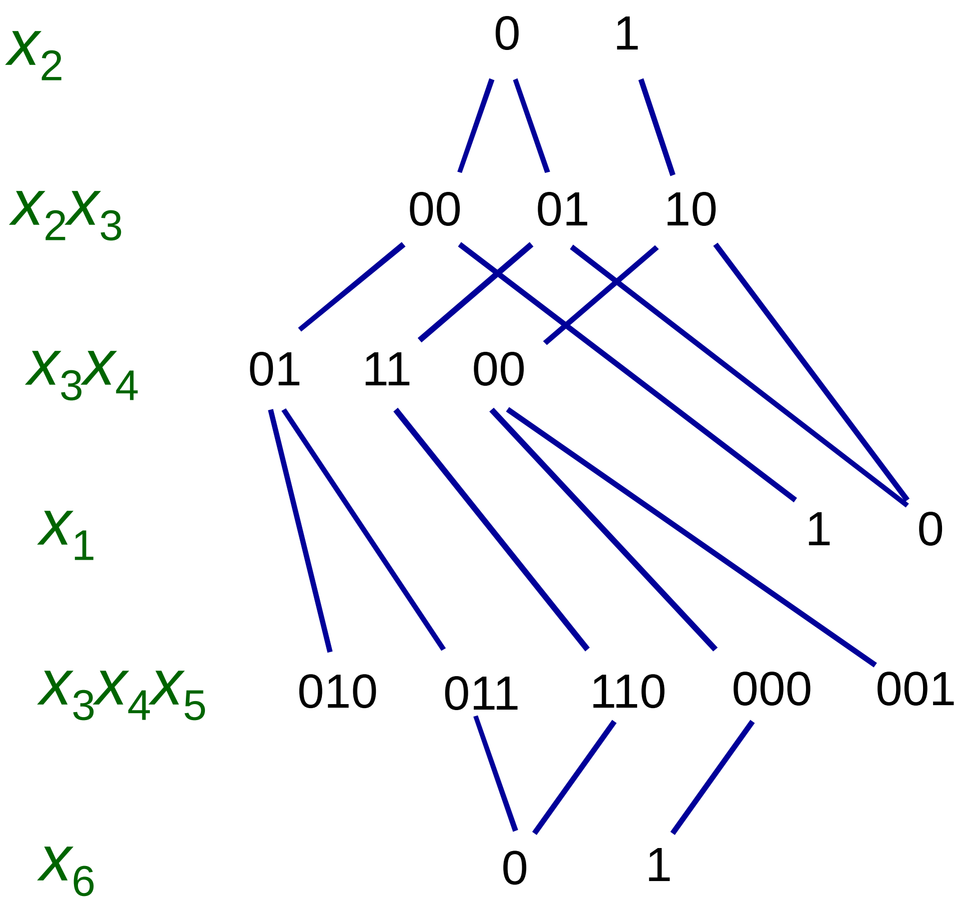


Enumeration order



Set Partitioning example

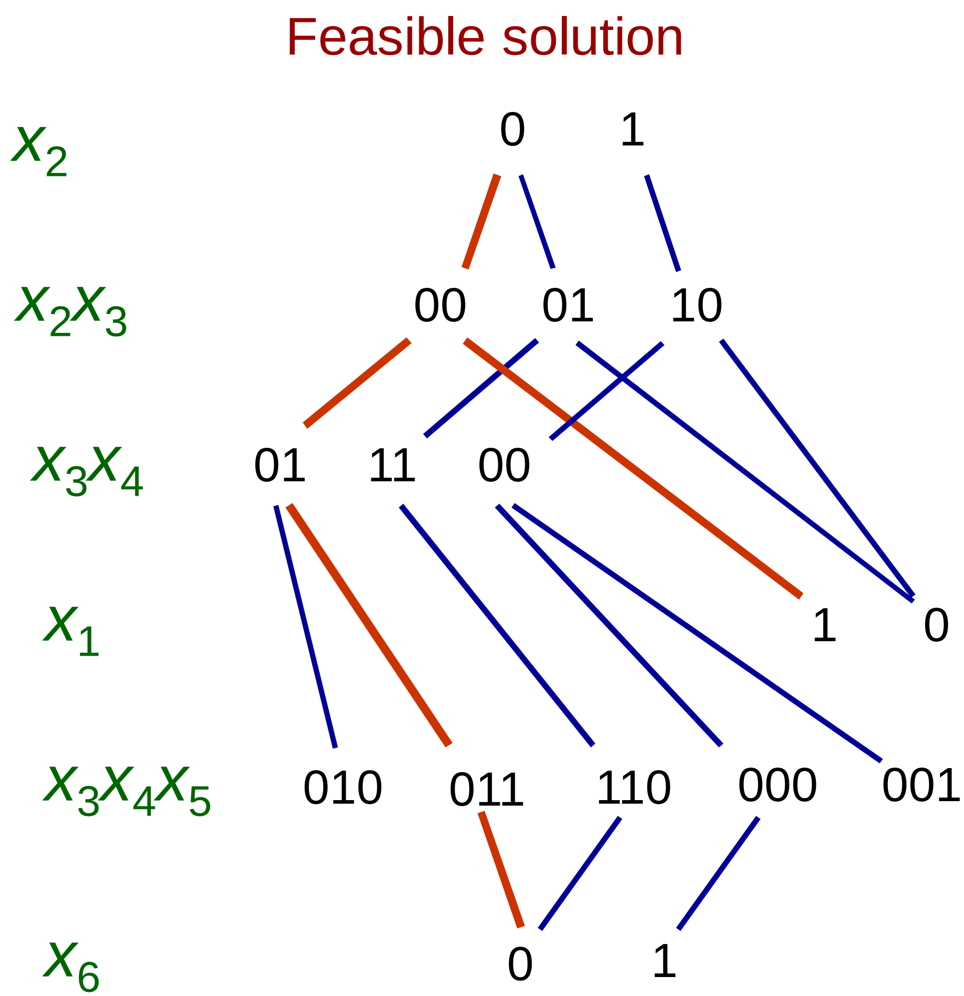
Solution by nonserial DP



Sets

	1	2	3	4	5	6
A	●	●	●			
B		●		●		
C			●		●	●
D				●		●

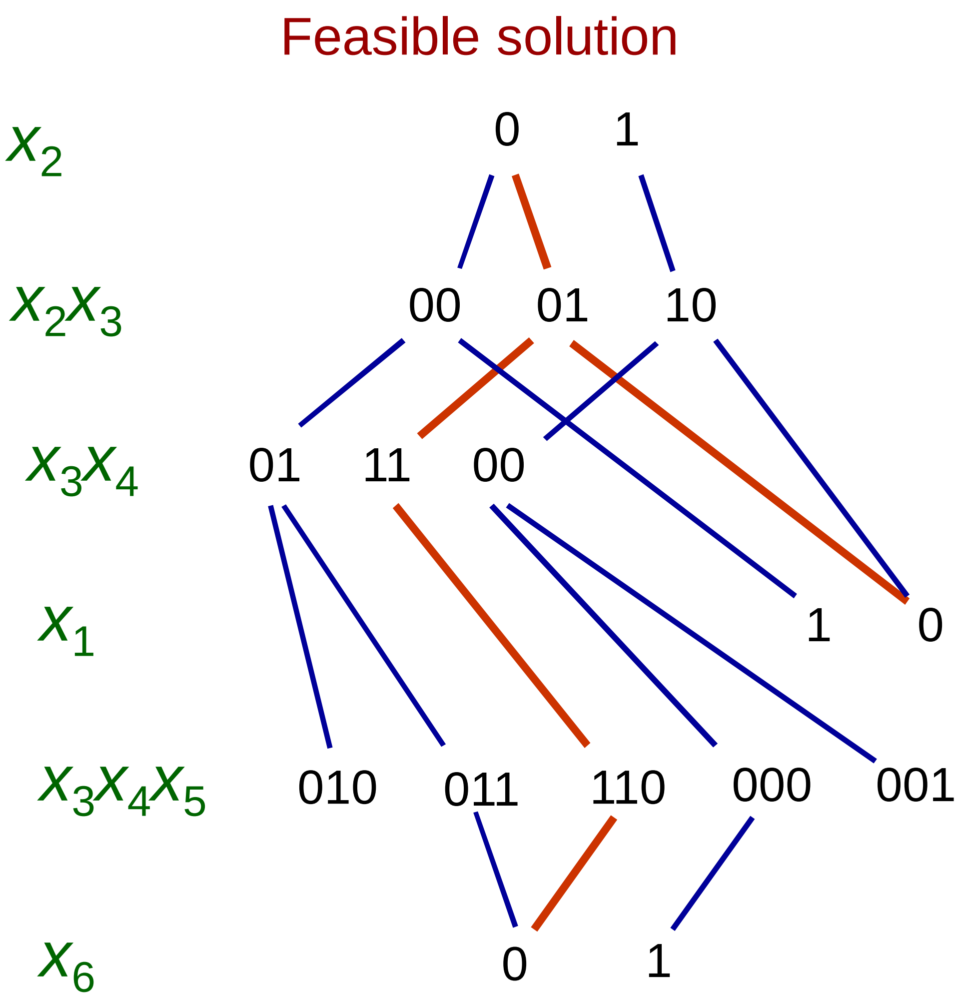
Set Partitioning example



Sets

	1	2	3	4	5	6
A	•	•	•			
B		•		•		
C			•		•	•
D				•		•
	1	0	0	1	1	0

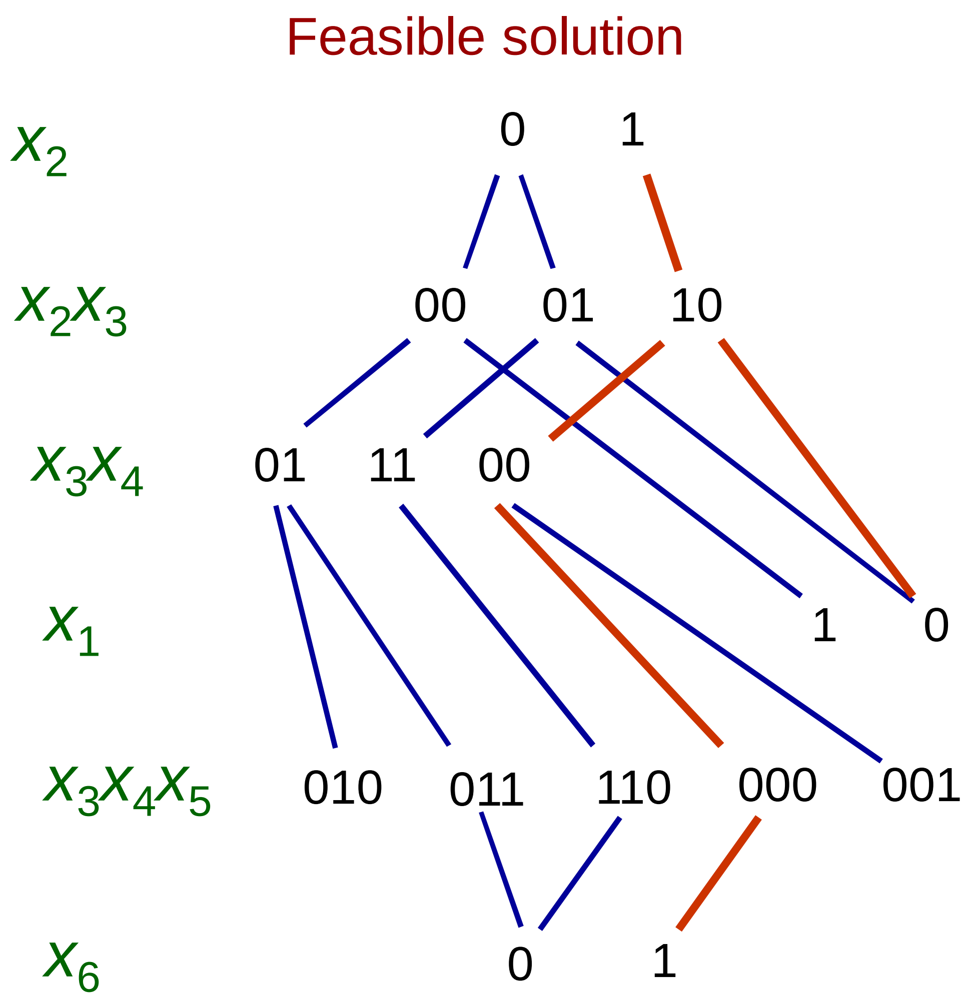
Set Partitioning example



Sets

	1	2	3	4	5	6
A	●	●	●			
B		●		●		
C			●		●	●
D				●		●
	1	0	0	1	1	0
	0	0	1	1	0	0

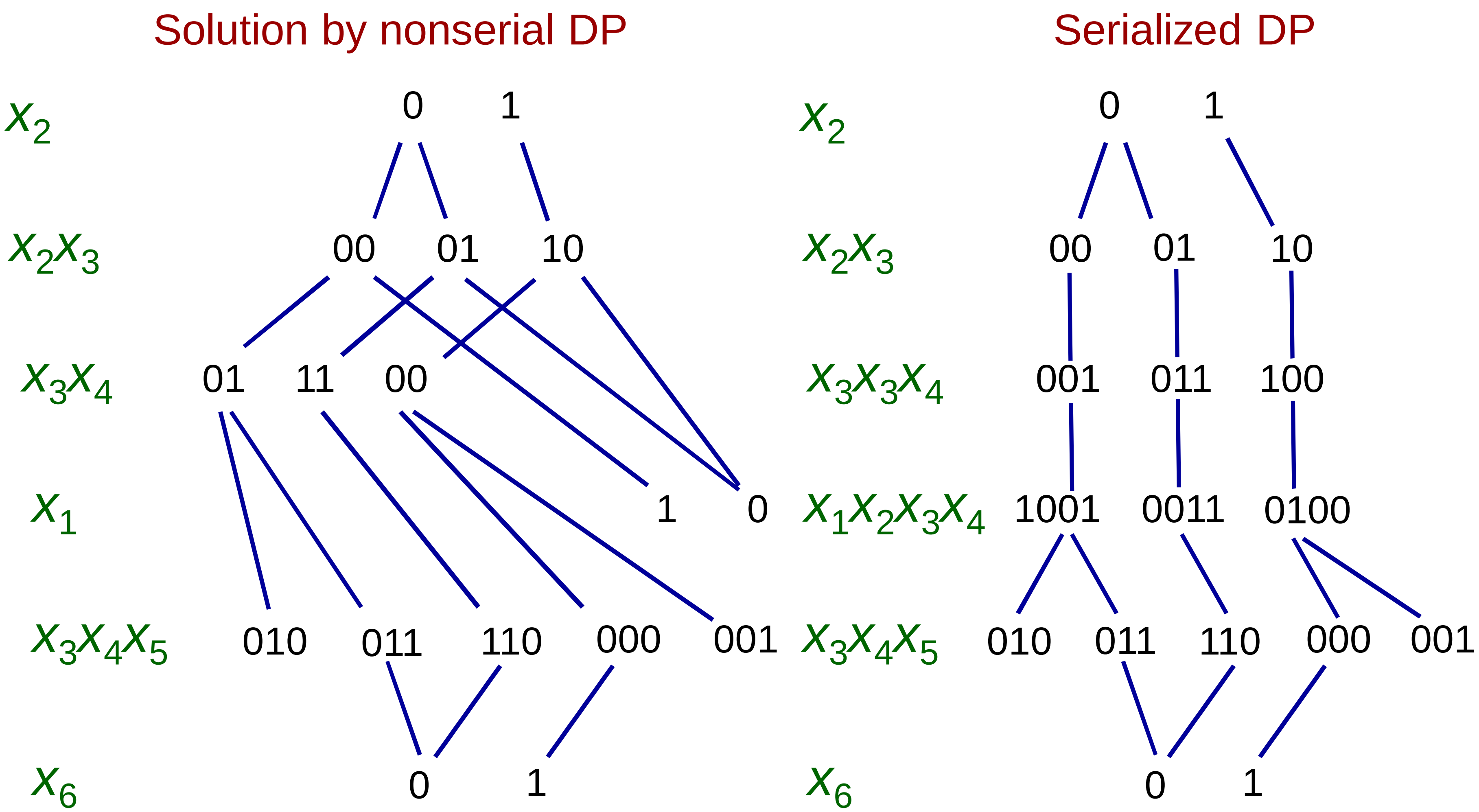
Set Partitioning example



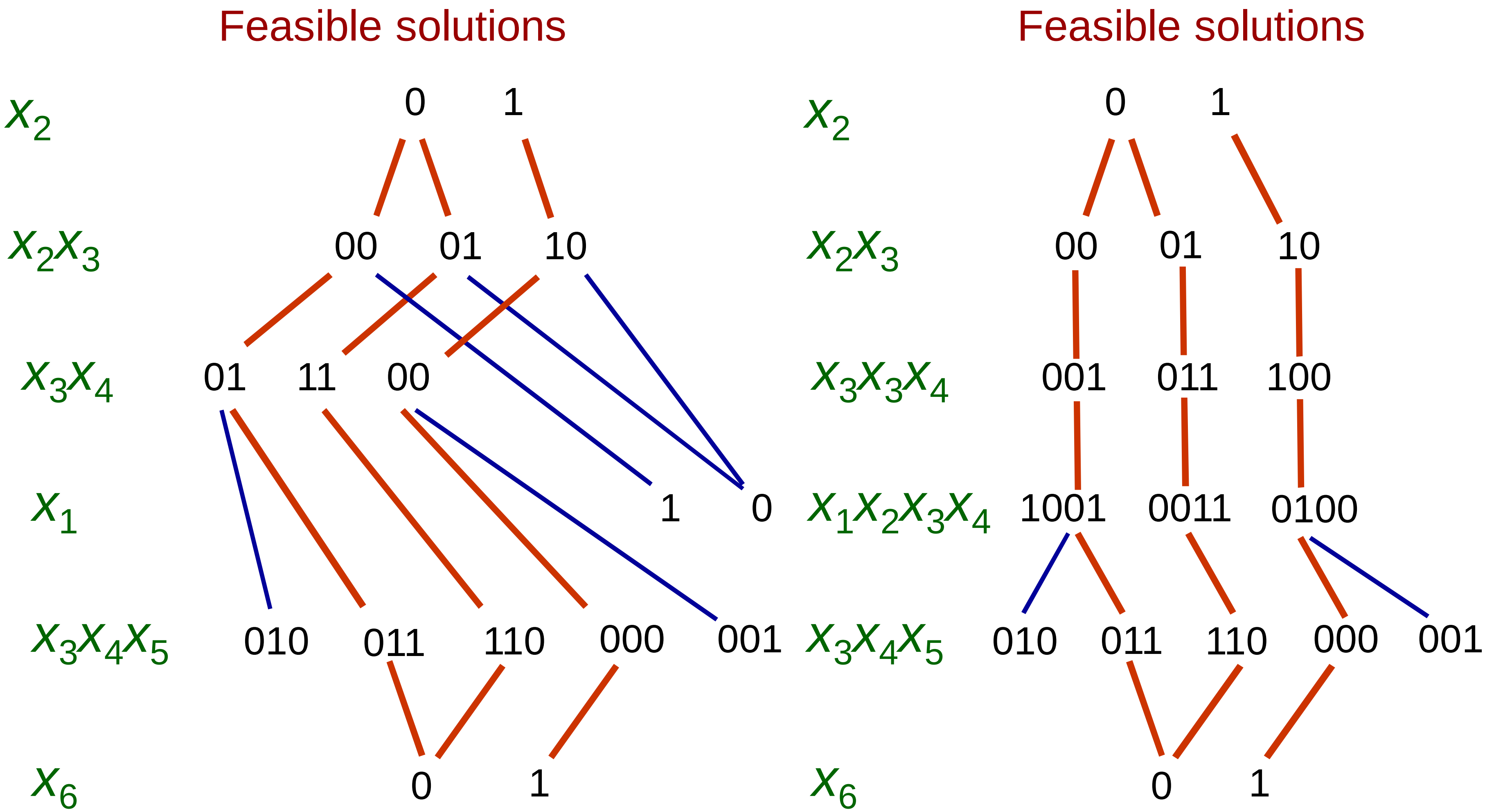
Sets

	1	2	3	4	5	6
A	●	●	●			
B		●		●		
C			●		●	●
D				●		●
	1	0	0	1	1	0
	0	0	1	1	0	0
	0	1	0	0	0	1

Set Partitioning example

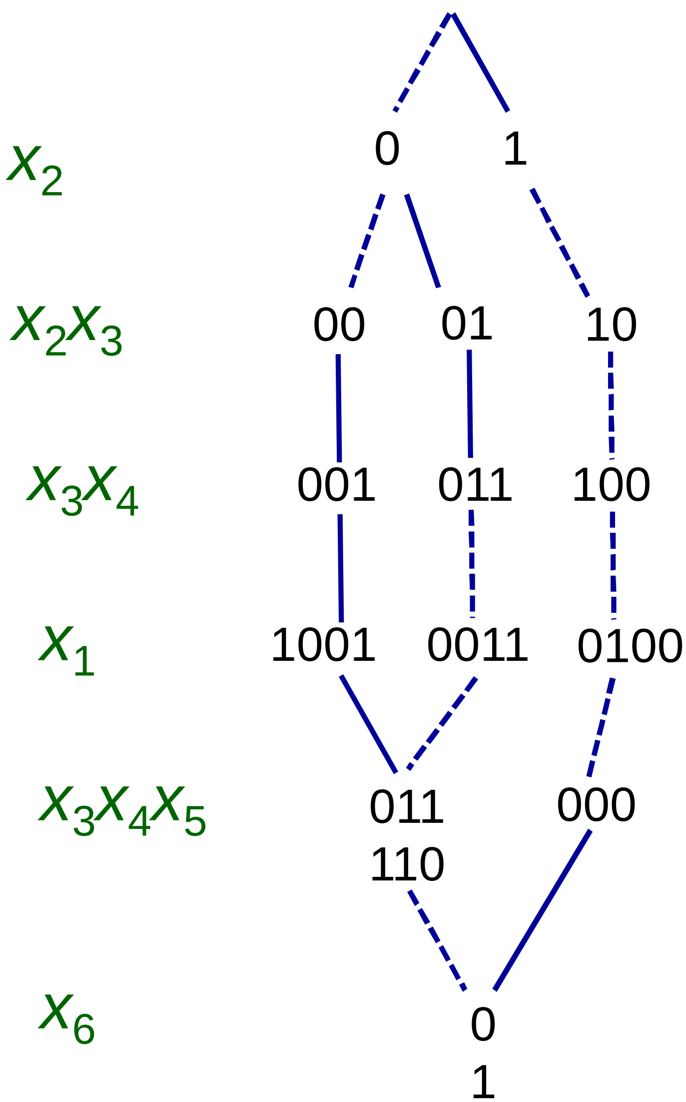


Set Partitioning example

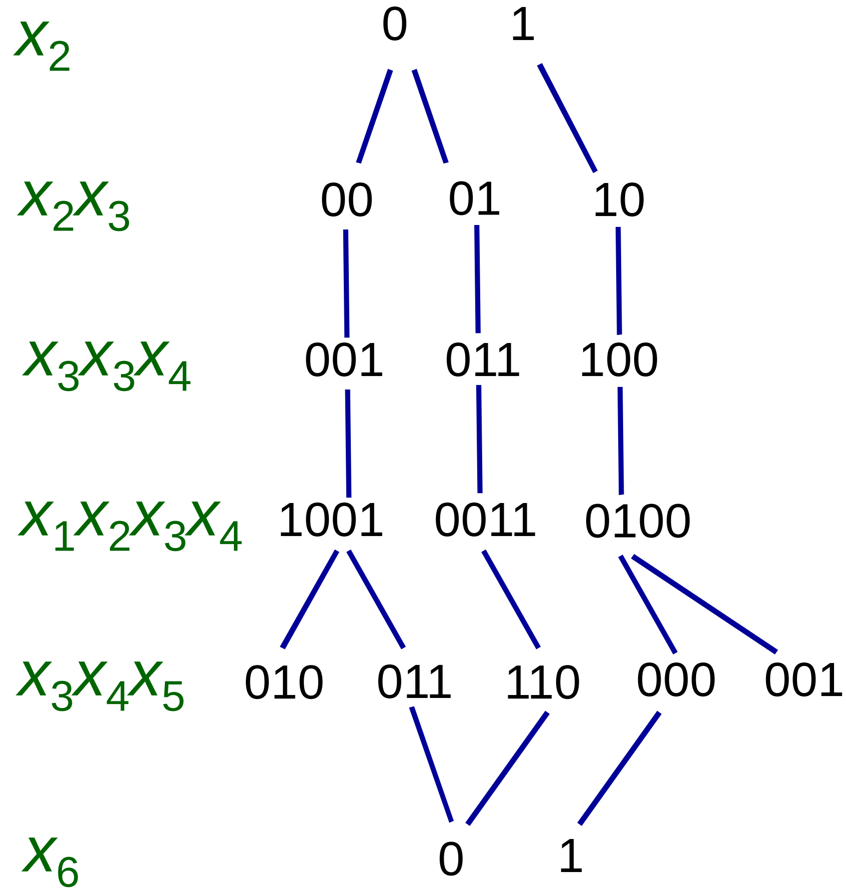


BDD vs. DP Solution

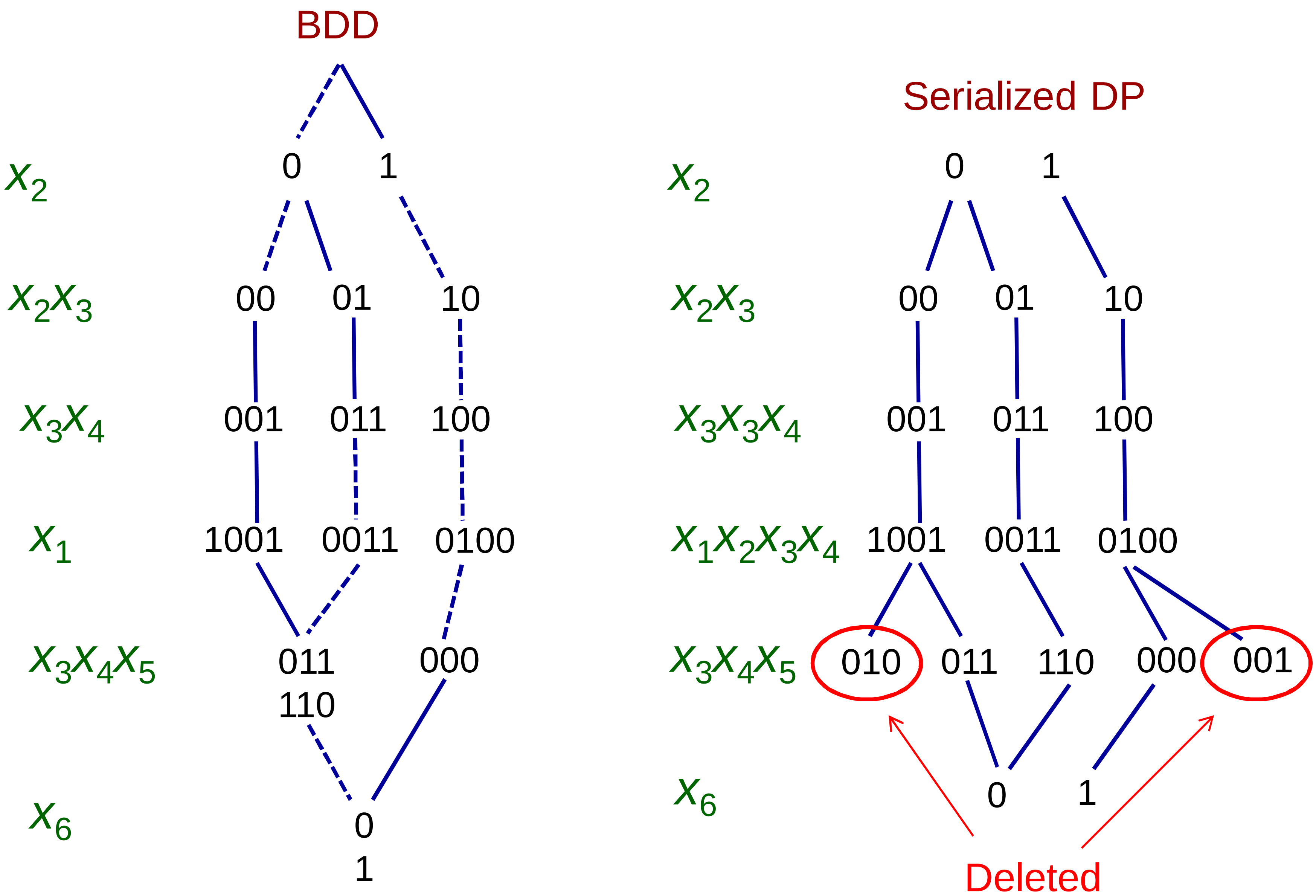
BDD



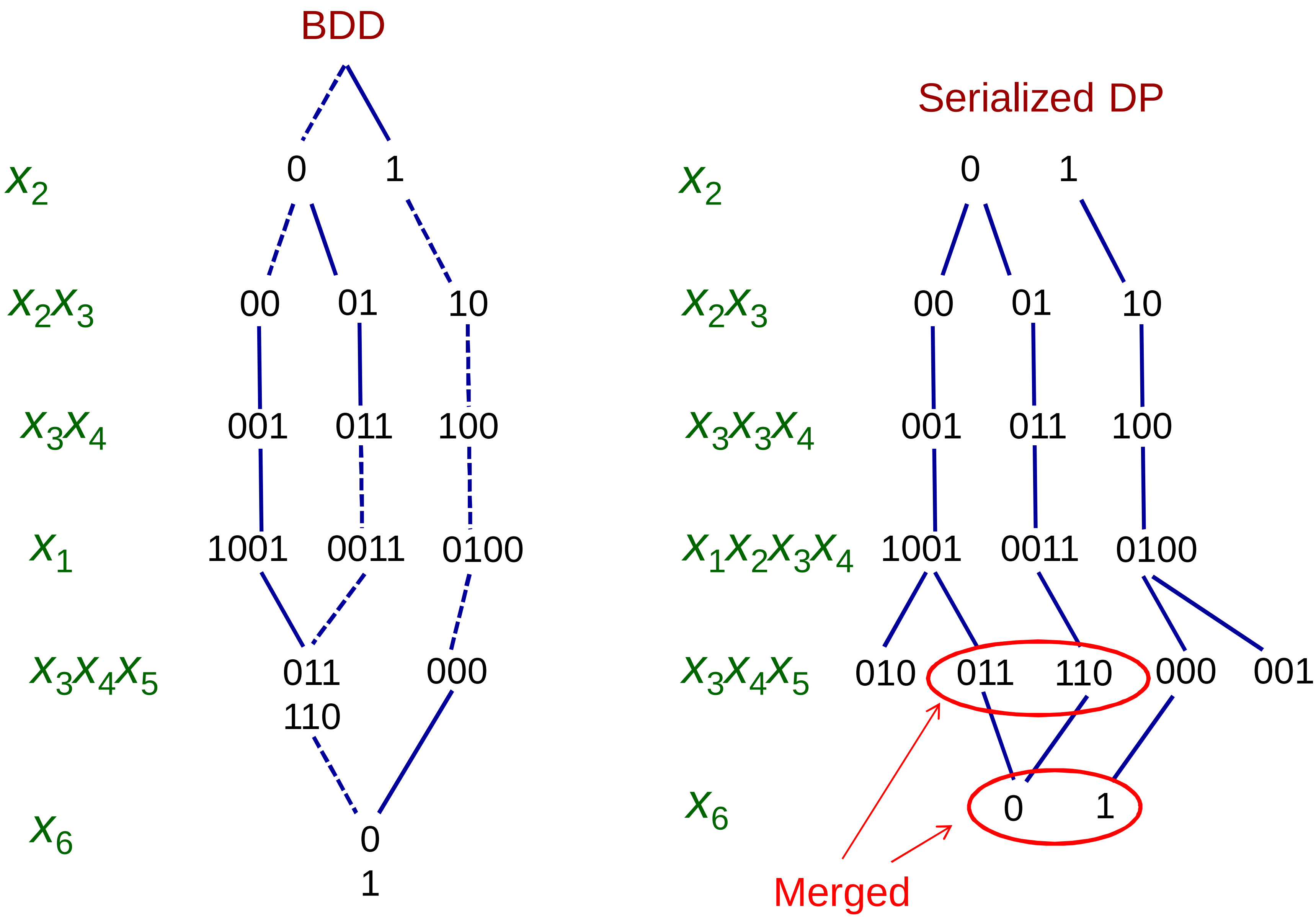
Serialized DP



BDD vs. DP Solution

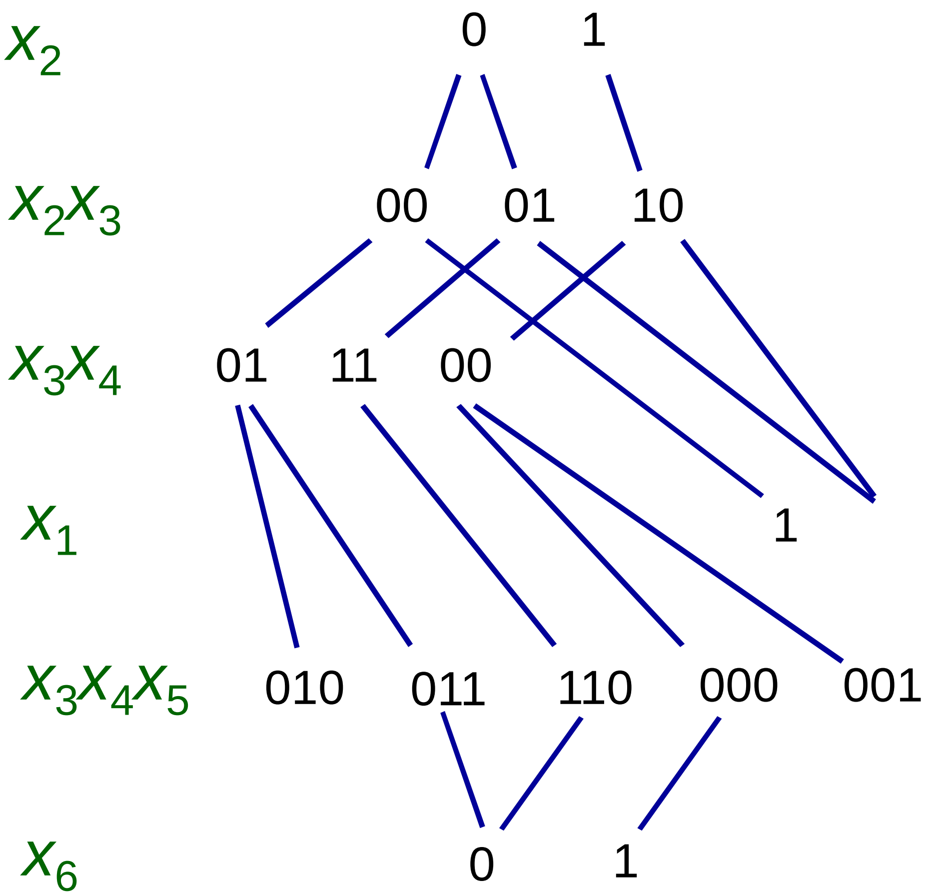


BDD vs. DP Solution



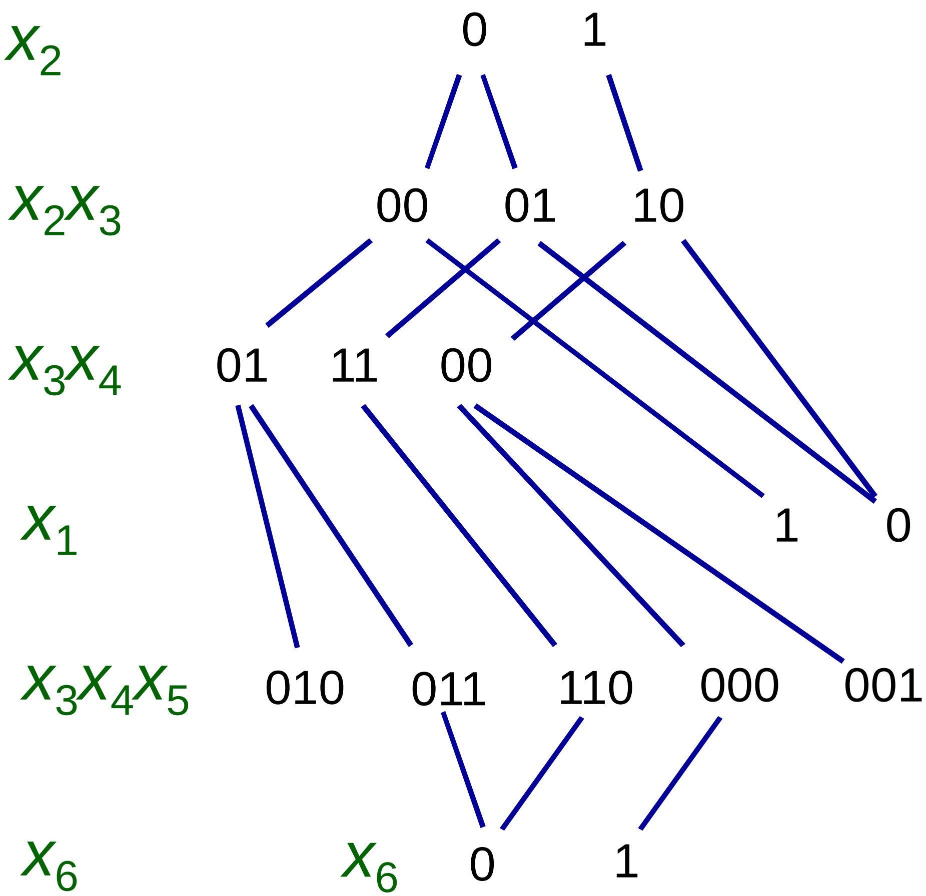
Set Partitioning example

Solution by nonserial DP

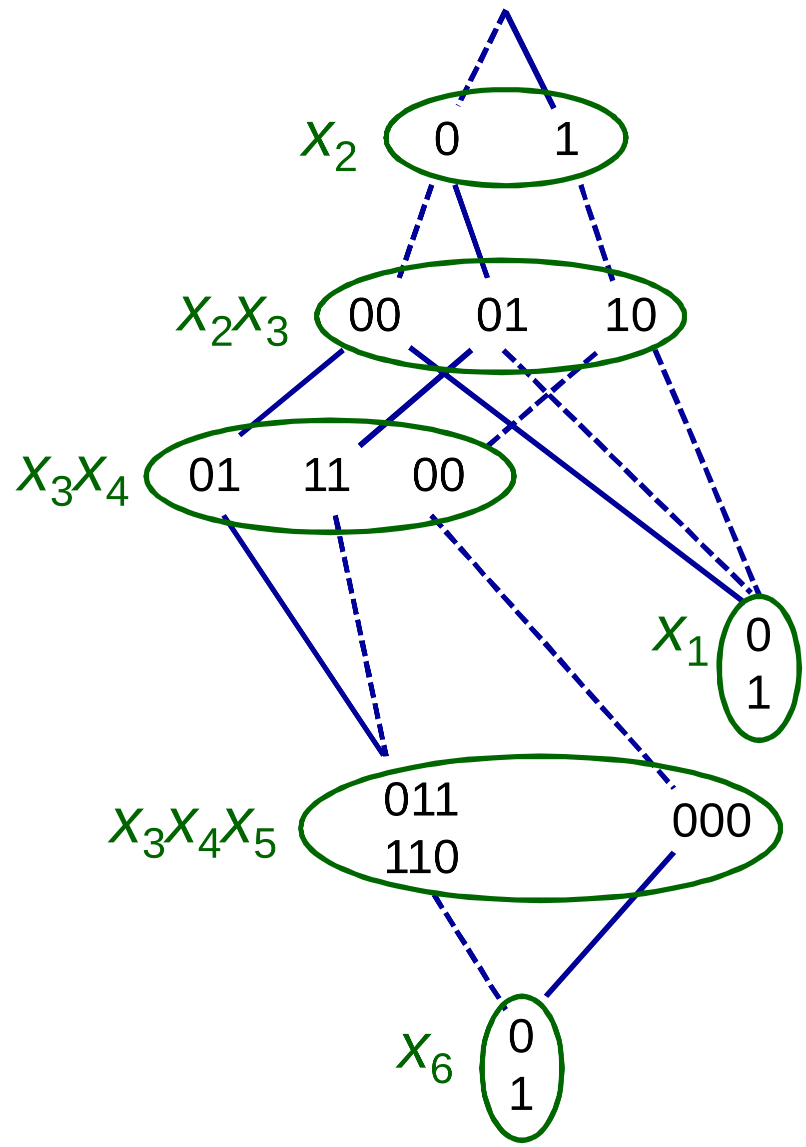


Set Partitioning example

Solution by nonserial DP



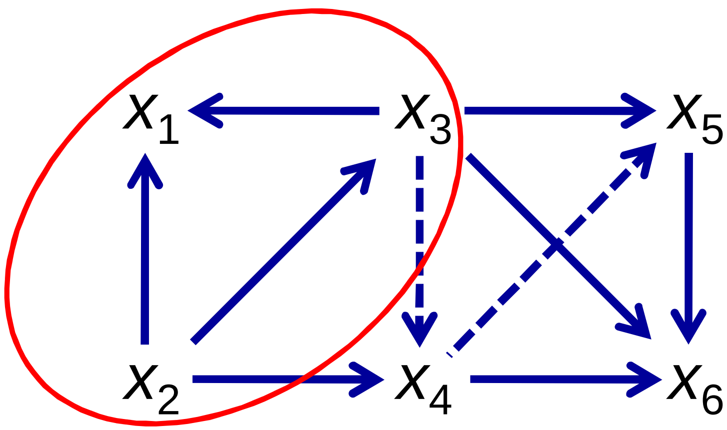
Nonserial BDD



Constructing the Join Tree

$x_2 \; x_3 \; x_4 \; x_1 \; x_5 \; x_6$

Clique in the dependency graph



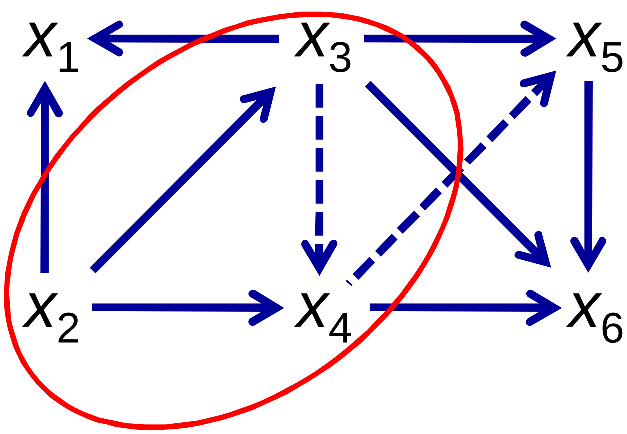
$x_1 x_2 x_3$

Constructing the Join Tree

$x_2 \ x_3 \ x_4 \ x_1 \ x_5 \ x_6$

Clique in the dependency graph

$x_1x_2x_3$

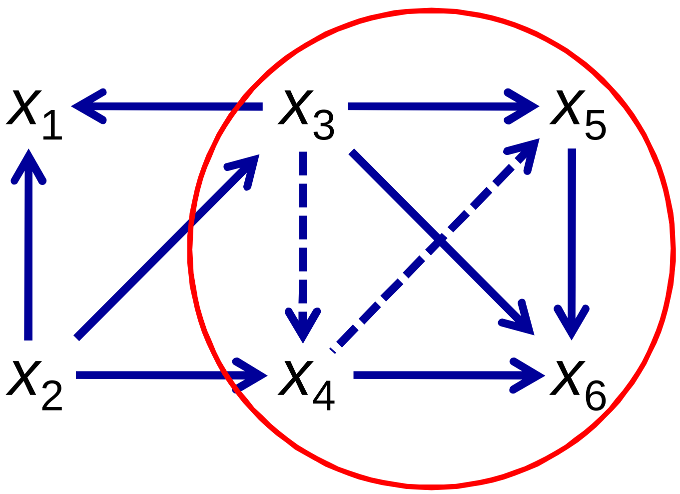


$x_2x_3x_4$

Constructing the Join Tree

$x_2 \; x_3 \; x_4 \; x_1 \; x_5 \; x_6$

Clique in the
dependency graph



$x_1x_2x_3$

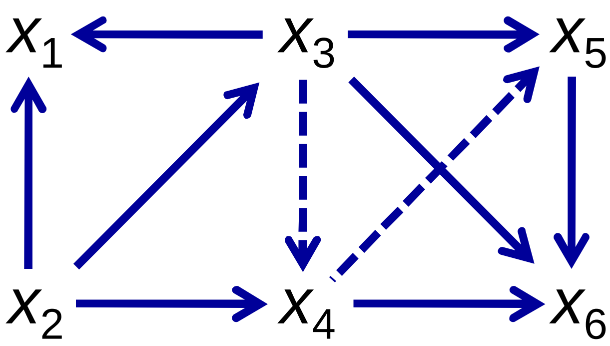
$x_2x_3x_4$

$x_3x_4x_5x_6$

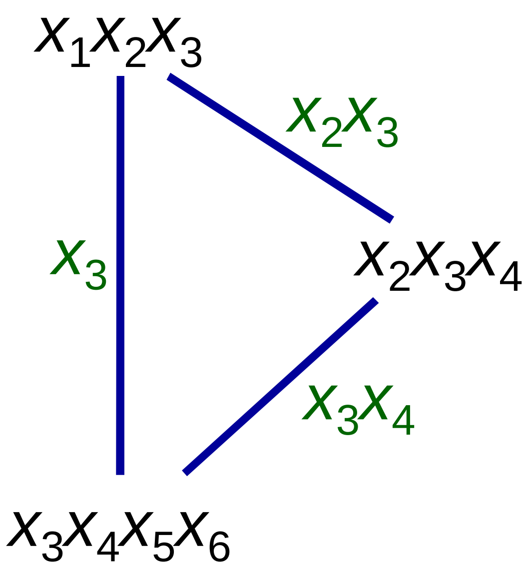
Constructing the Join Tree

$x_2 \; x_3 \; x_4 \; x_1 \; x_5 \; x_6$

Dependency graph



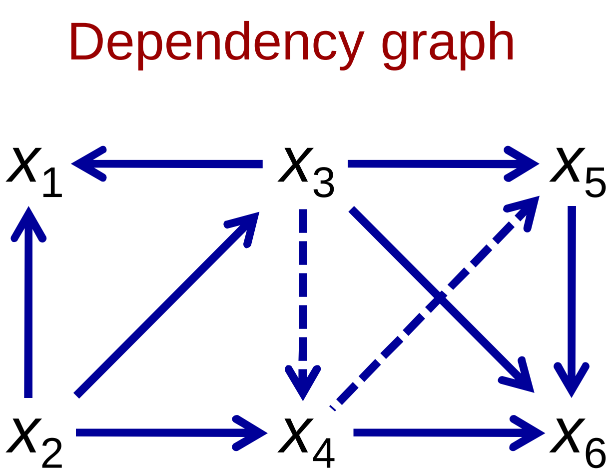
Join graph

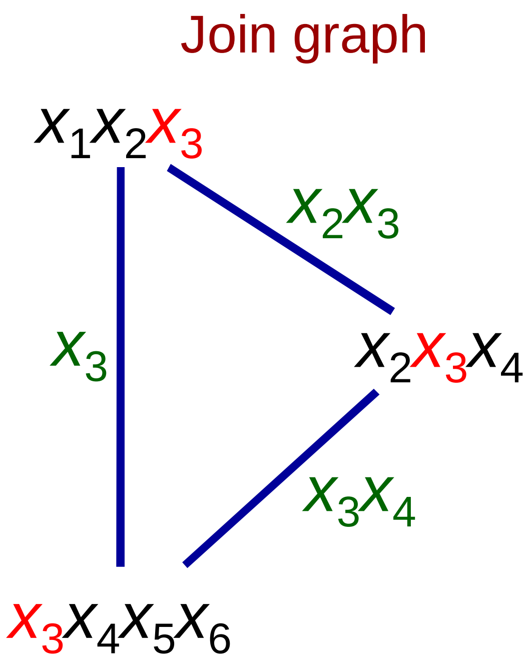


Connect nodes with common variables

Constructing the Join Tree

$$x_2\ x_3\ x_4\ x_1\ x_5\ x_6$$



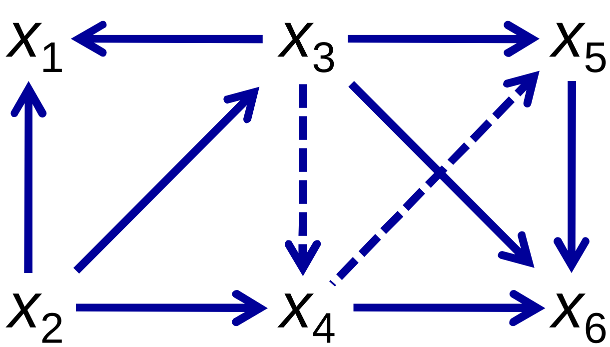


x_j occurs along every path connecting x_j with x_j

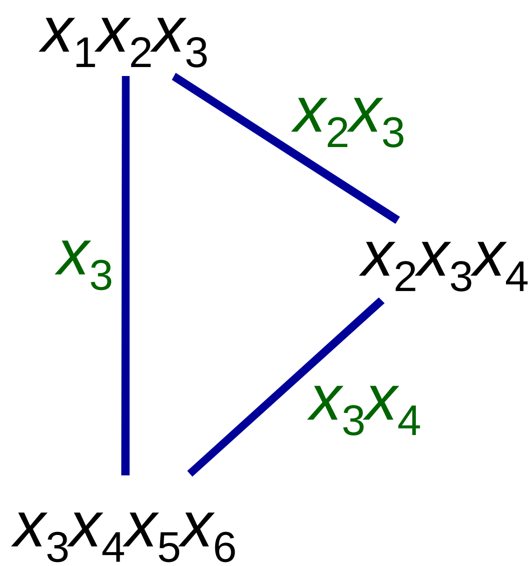
Constructing the Join Tree

$x_2\ x_3\ x_4\ x_1\ x_5\ x_6$

Dependency graph



Join graph



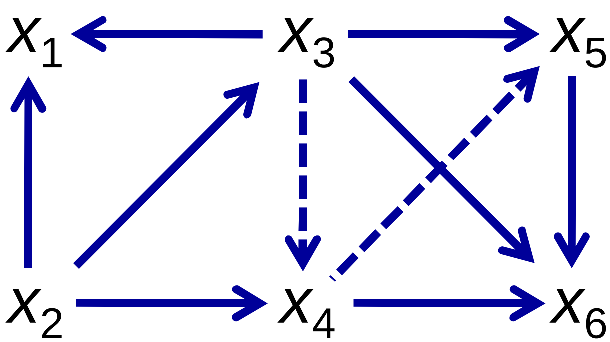
This can be viewed as the **constraint dual**

Binary constraints equate common variables in subproblems

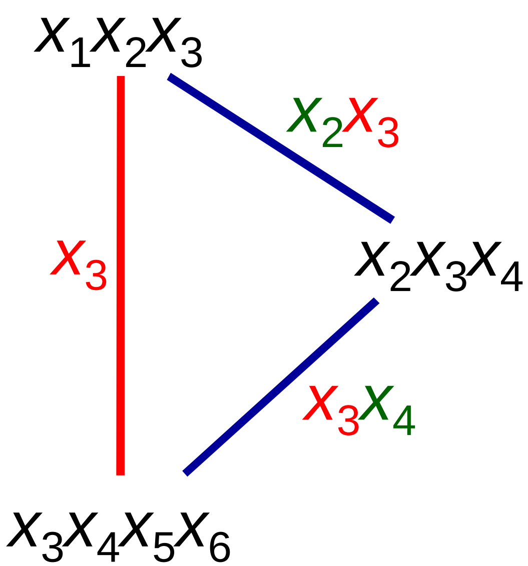
Constructing the Join Tree

$x_2 \; x_3 \; x_4 \; x_1 \; x_5 \; x_6$

Dependency graph



Join graph

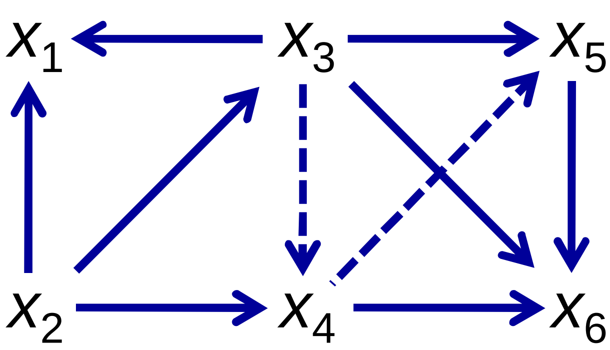


Some edges may be redundant when equating variables

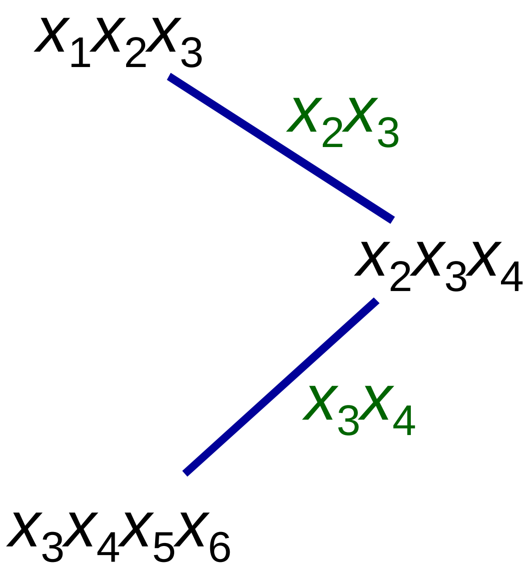
Constructing the Join Tree

$x_2\ x_3\ x_4\ x_1\ x_5\ x_6$

Dependency graph



Join tree

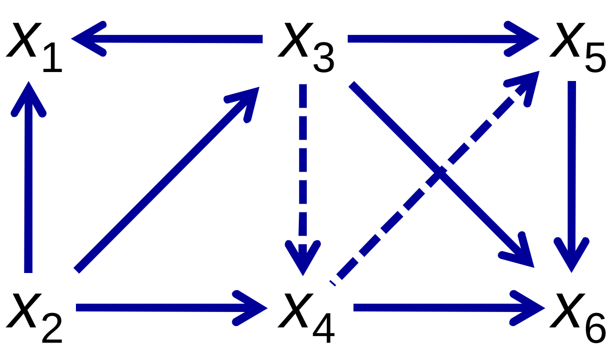


Removing redundant edges
yields **join tree**

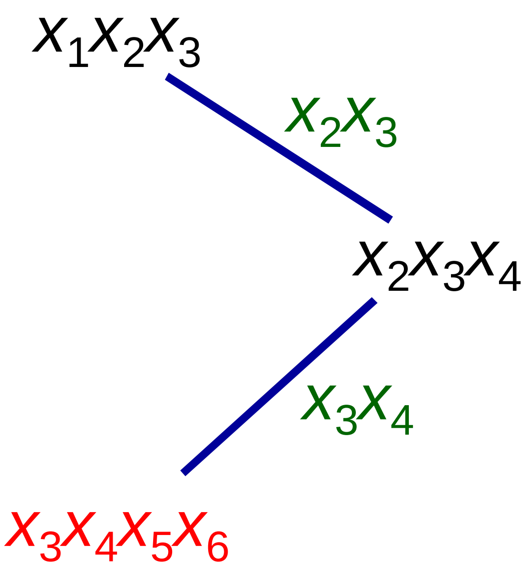
Constructing the Join Tree

$x_2 \; x_3 \; x_4 \; x_1 \; x_5 \; x_6$

Dependency graph



Join tree

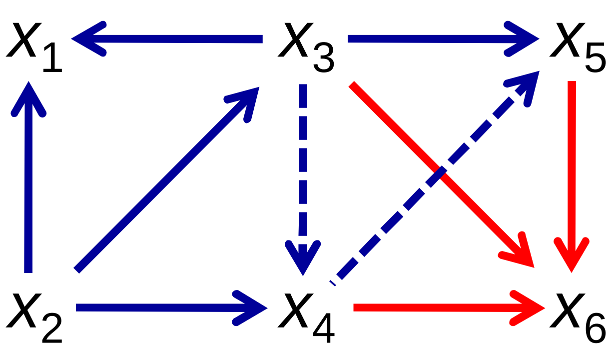


Max node cardinality is
tree width + 1 = **3** + 1

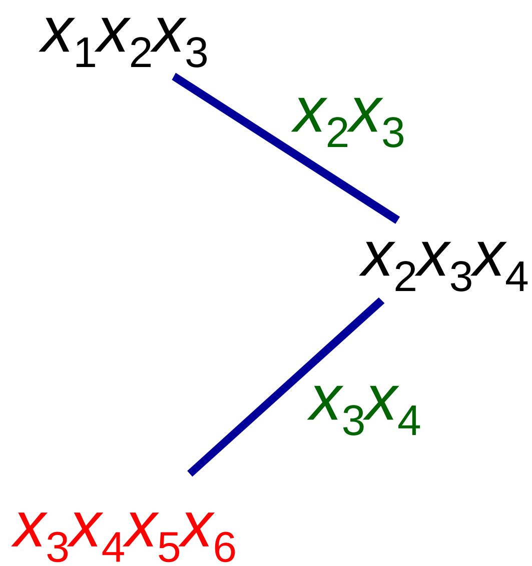
Constructing the Join Tree

$x_2\ x_3\ x_4\ x_1\ x_5\ x_6$

Dependency graph



Join tree



Induced width = tree width = 3

Designing the Nonserial BDD

$x_2 \; x_3 \; x_4 \; x_1 \; x_5 \; x_6$

BDD design

x_2

Join tree

$x_2x_3x_1$

x_2x_3

$x_2x_3x_4$

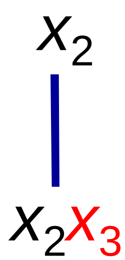
x_3x_4

$x_3x_4x_5x_6$

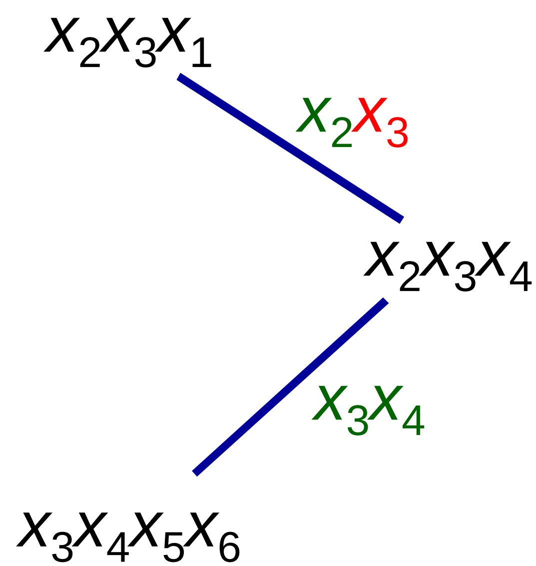
Designing the Nonserial BDD

$x_2\ x_3\ x_4\ x_1\ x_5\ x_6$

BDD design



Join tree



Designing the Nonserial BDD

$x_2\ x_3\ x_4\ x_1\ x_5\ x_6$

BDD design

x_2

$x_2 x_3$

$x_3 x_4$

Join tree

$x_2 x_3 x_1$

$x_2 x_3$

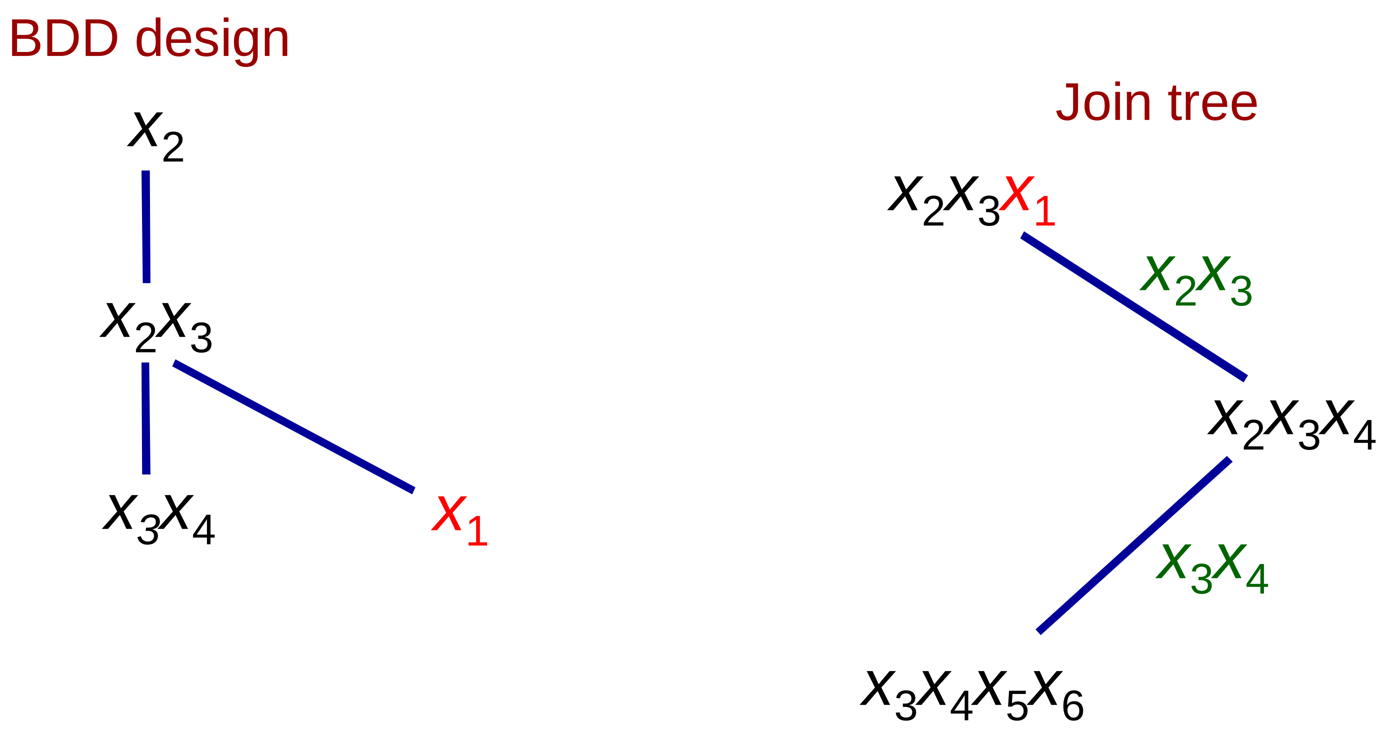
$x_2 x_3 x_4$

$x_3 x_4$

$x_3 x_4 x_5 x_6$

Designing the Nonserial BDD

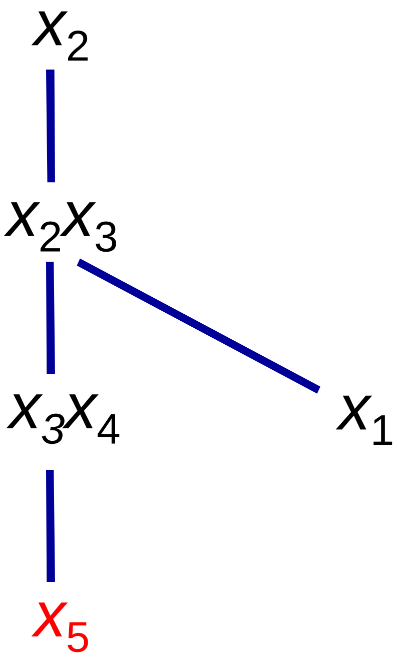
$x_2 \; x_3 \; x_4 \; x_1 \; x_5 \; x_6$



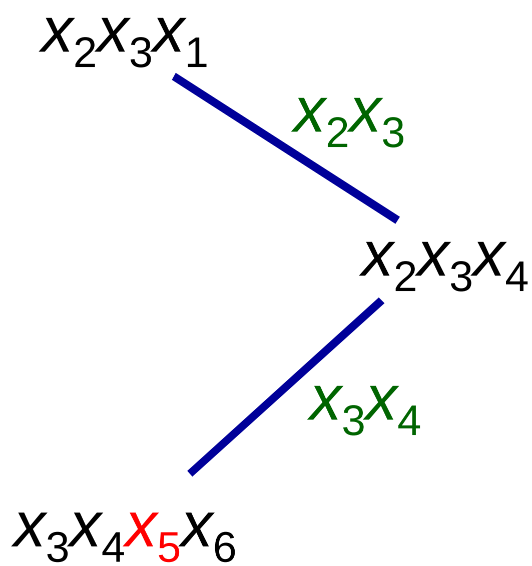
Designing the Nonserial BDD

$x_2 \; x_3 \; x_4 \; x_1 \; x_5 \; x_6$

BDD design

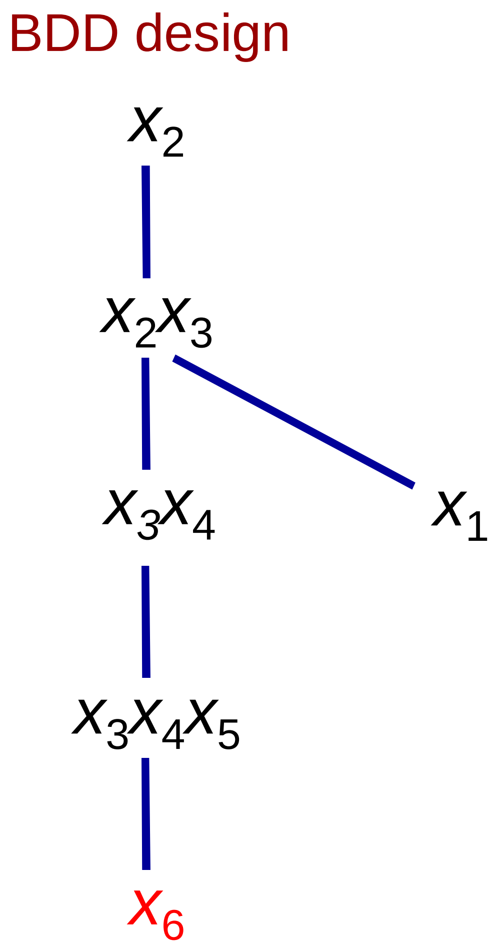


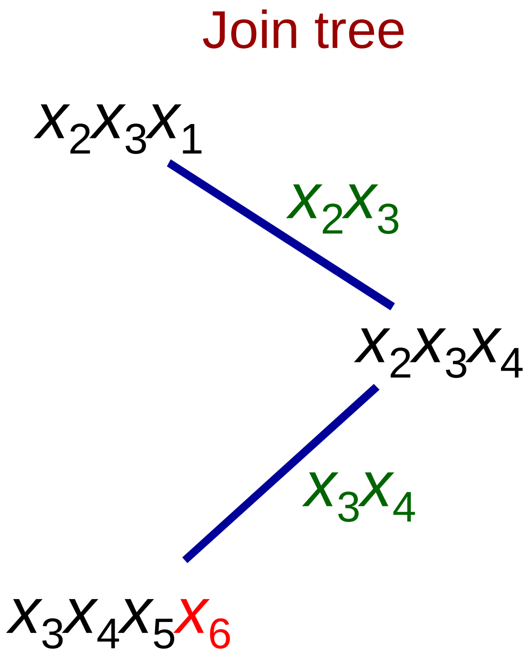
Join tree



Designing the Nonserial BDD

$x_2 \; x_3 \; x_4 \; x_1 \; x_5 \; x_6$

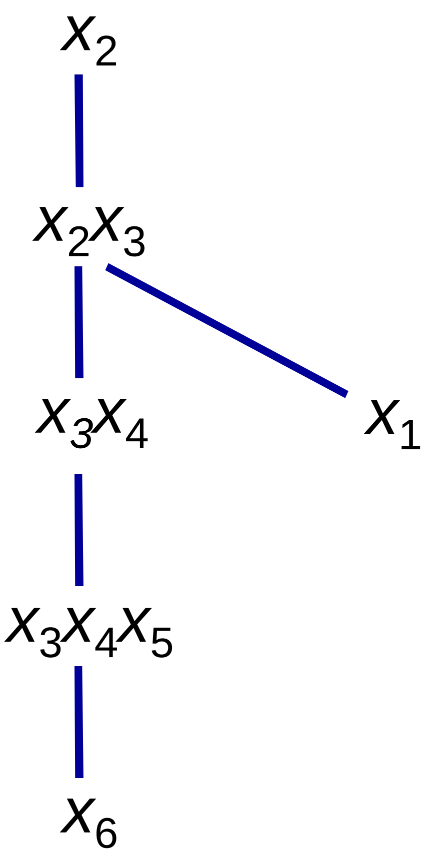




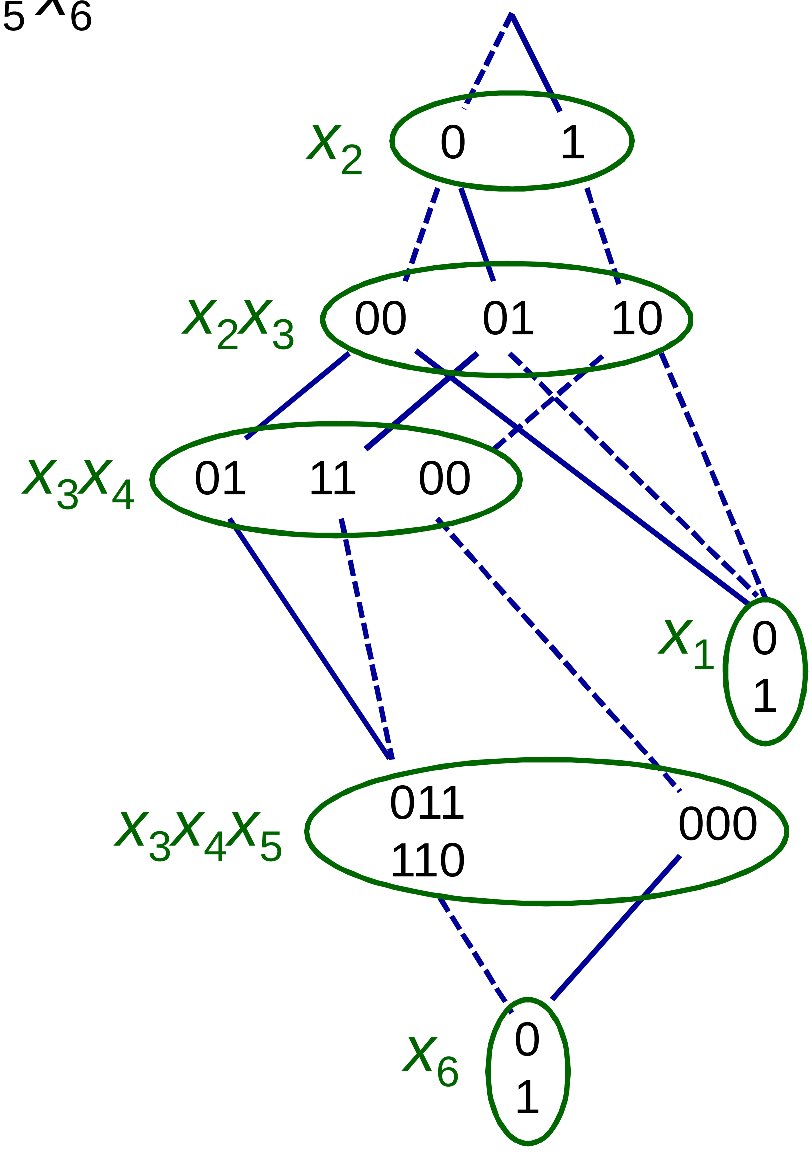
Designing the Nonserial BDD

$x_2 \; x_3 \; x_4 \; x_1 \; x_5 \; x_6$

BDD design



Nonserial BDD

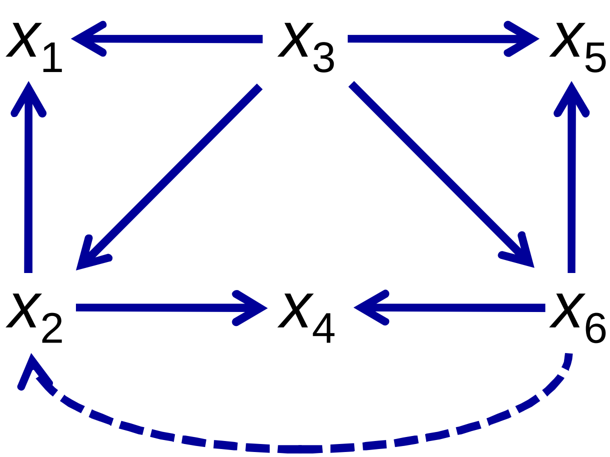


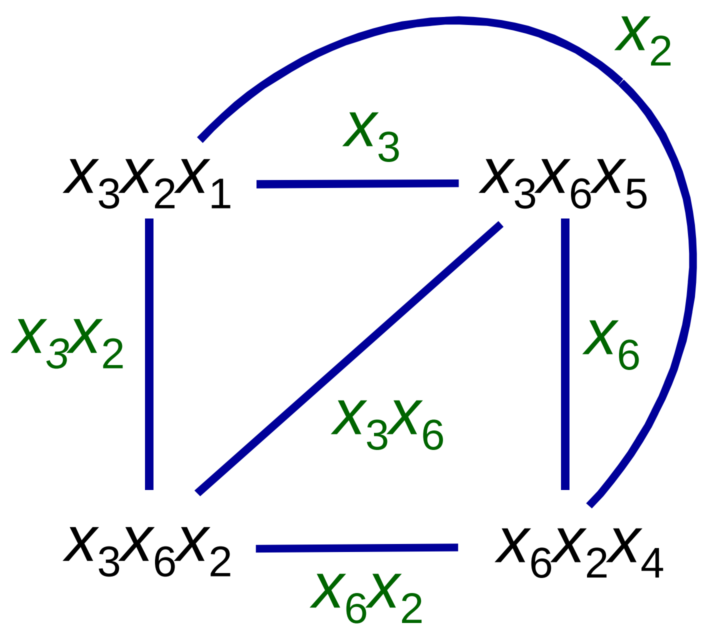
Another Variable Ordering

$X_3\ X_6\ X_2\ X_5\ X_1\ X_4$

Join graph

Dependency graph



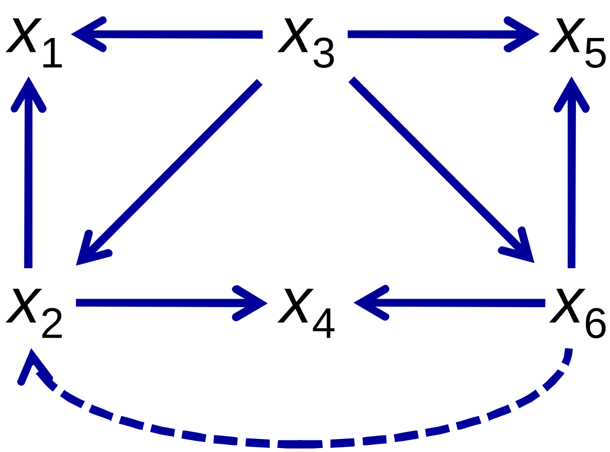


Induced width = 2

Constructing the Join Tree

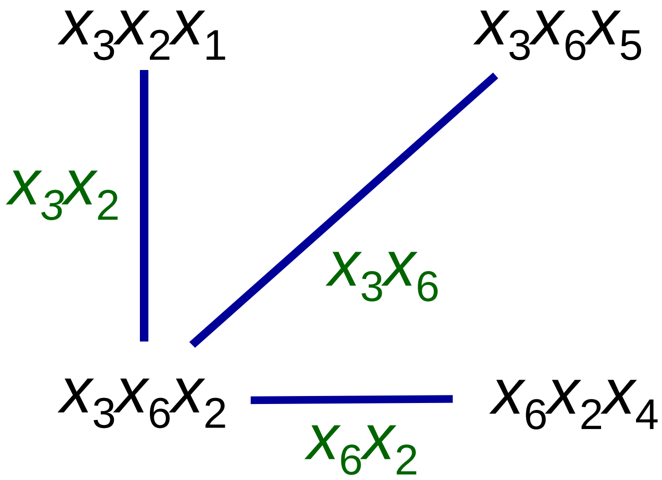
$X_3 \; X_6 \; X_2 \; X_5 \; X_1 \; X_4$

Dependency graph



Induced width = 2

Join tree



Tree width = 2

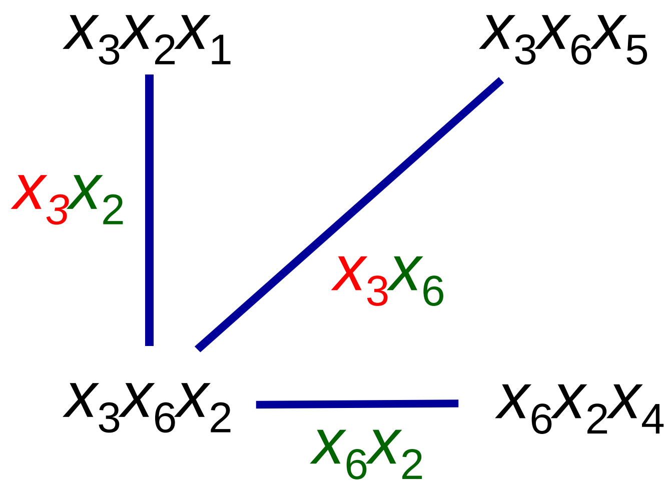
Designing the BDD

$x_3 \; x_6 \; x_2 \; x_5 \; x_1 \; x_4$

BDD design

x_3

Join tree

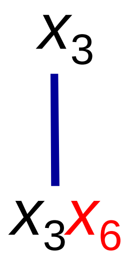


Tree width = 2

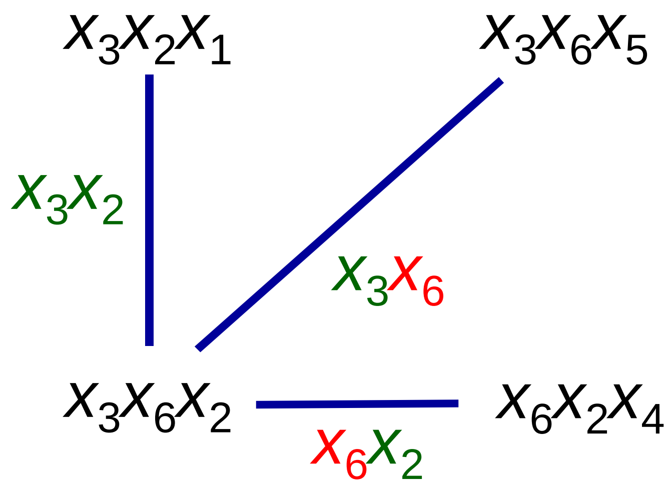
Designing the BDD

$x_3 \; x_6 \; x_2 \; x_5 \; x_1 \; x_4$

BDD design



Join tree

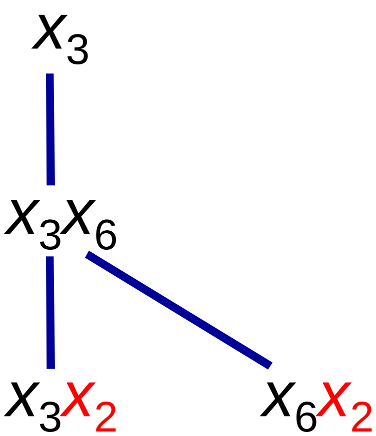


Tree width = 2

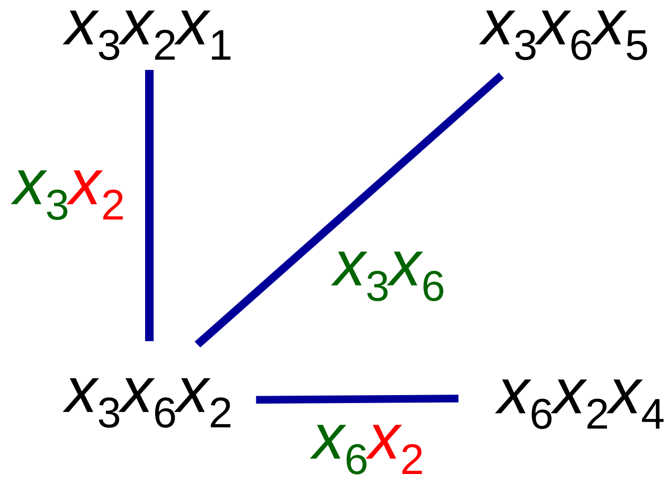
Designing the BDD

$x_3 \; x_6 \; x_2 \; x_5 \; x_1 \; x_4$

BDD design



Join tree

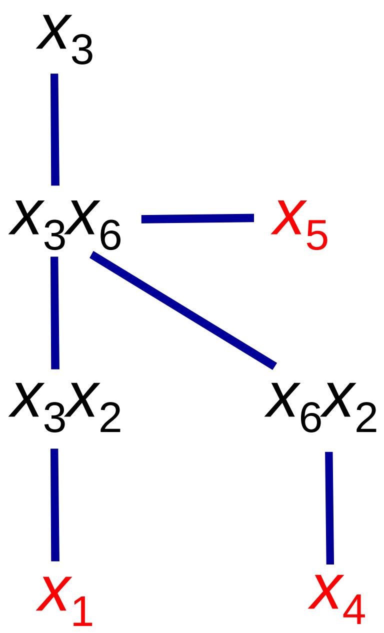


Tree width = 2

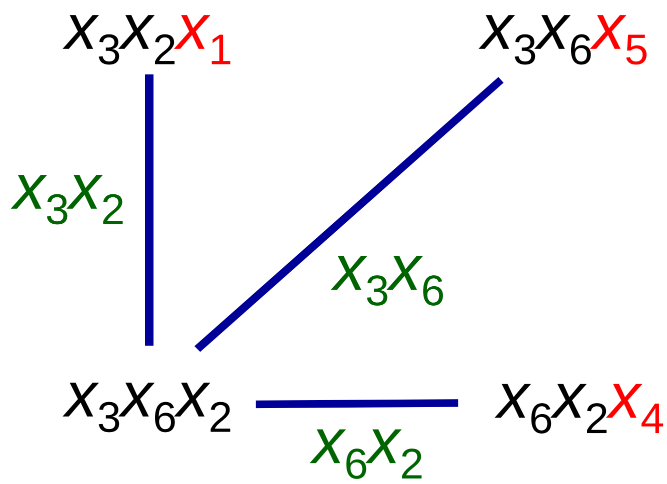
Designing the BDD

$x_3\ x_6\ x_2\ x_5\ x_1\ x_4$

BDD design



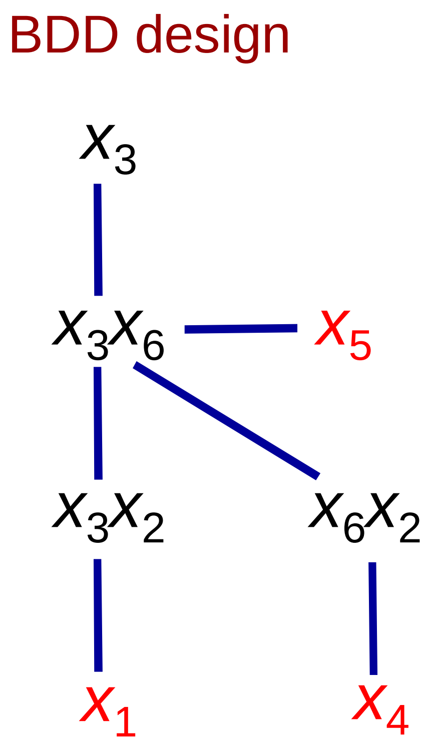
Join tree

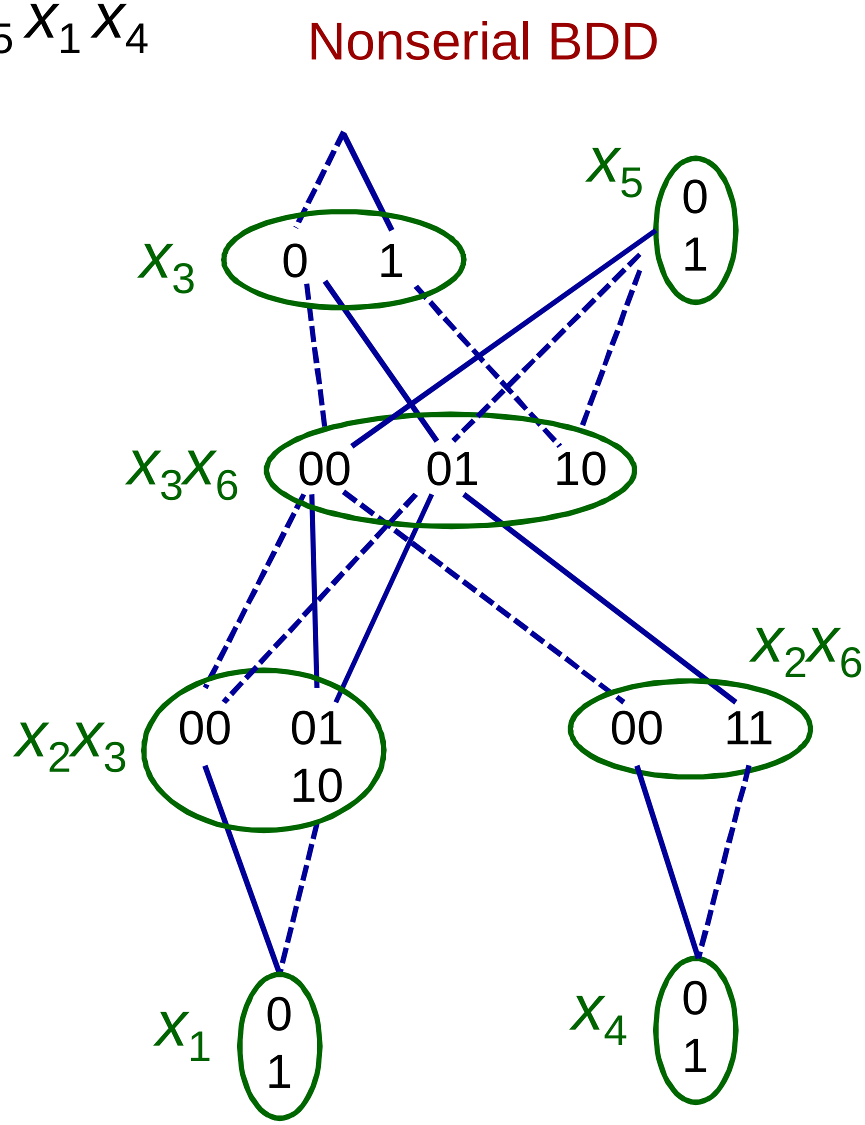


Tree width = 2

Nonserial BDD

$x_3\ x_6\ x_2\ x_5\ x_1\ x_4$





Constraints with DP Structure

- Constraints with serial DP structure
  - Stretch
  - Regular
  - Sequence
- Constraints with nonserial DP structure…