Decision Diagrams: Tutorial

John Hooker
Carnegie Mellon University

CP Summer School
Cork, Ireland, June 2016

Decision Diagrams

- Used in **computer science** and **AI** for decades
  - Logic circuit design
  - Product configuration
- A **new perspective** on problem solving
  - Constraint programming
  - Discrete optimization

Elements of a DD-based Solver

- CP solver
  - Build on **existing solver**.
  - Use **relaxed** DDs for enhanced **propagation**.
  - Plug in DDs as **additional global constraints**.
- Discrete optimization solver
  - Obtain **bounds** from **relaxed** DDs.
  - Use **restricted** DDs for **primal heuristic**.
  - Use **dynamic programming** formulation of problem.
  - **Branch** inside relaxed DD.

Decision Diagrams

- Advantages for **constraint programming**:
  - **Stronger** propagation, filtering.
  - Easily added to **existing solver**.
- Advantages for **optimization**:
  - No need for **inequality** formulations.
  - No need for **linear** or **convex** relaxations.
  - New approach to solving **dynamic programming** models.
  - Very effective **parallel** computation.
  - Ideal for **postoptimality** anaylsis
- Disadvantage:
  - Developed only for **discrete**, **deterministic** optimization.
  - …so far.

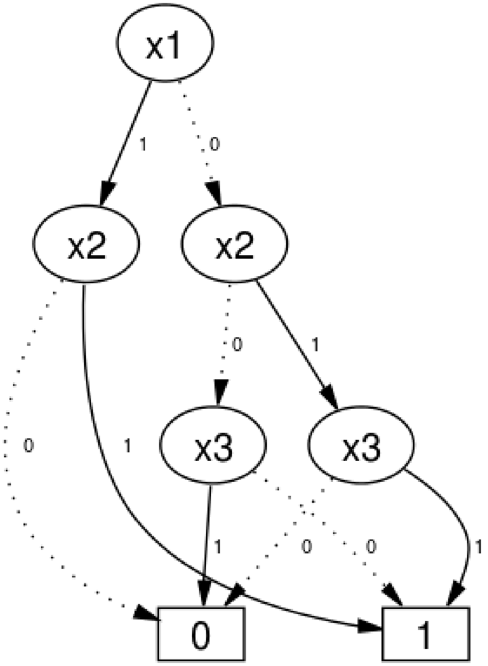
Outline

- Decision diagram **basics**
- Optimization with **exact** decision diagrams
- **Relaxed** decision diagrams
  - Relaxation by **node merger**
  - Relaxation by **node splitting**
- **Propagation** in relaxed diagrams
- **Restricted** decision diagrams
- **Dynamic programming** model
- **Branching** in a relaxed DD
- Modeling the **objective function**
  - Inventory management example
- **Nonserial** decision diagrams
- References

Decision Diagram Basics

- Binary decision diagrams encode Boolean functions

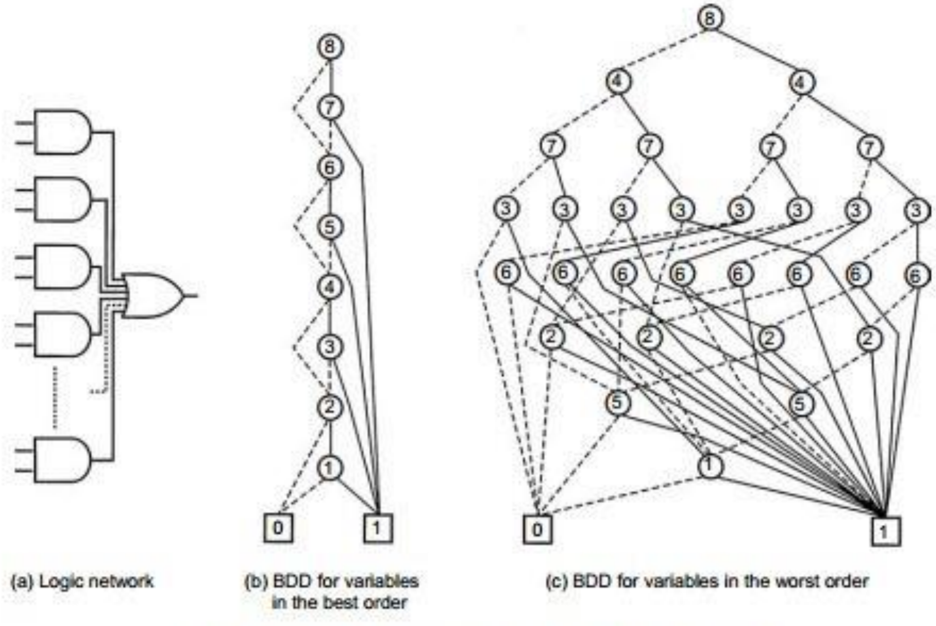
x_1	x_2	x_3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Lee (1959), Akers (1978)

Decision Diagram Basics

- Binary decision diagrams encode Boolean functions
  - Historically used for circuit design & verification



(a) Logic network

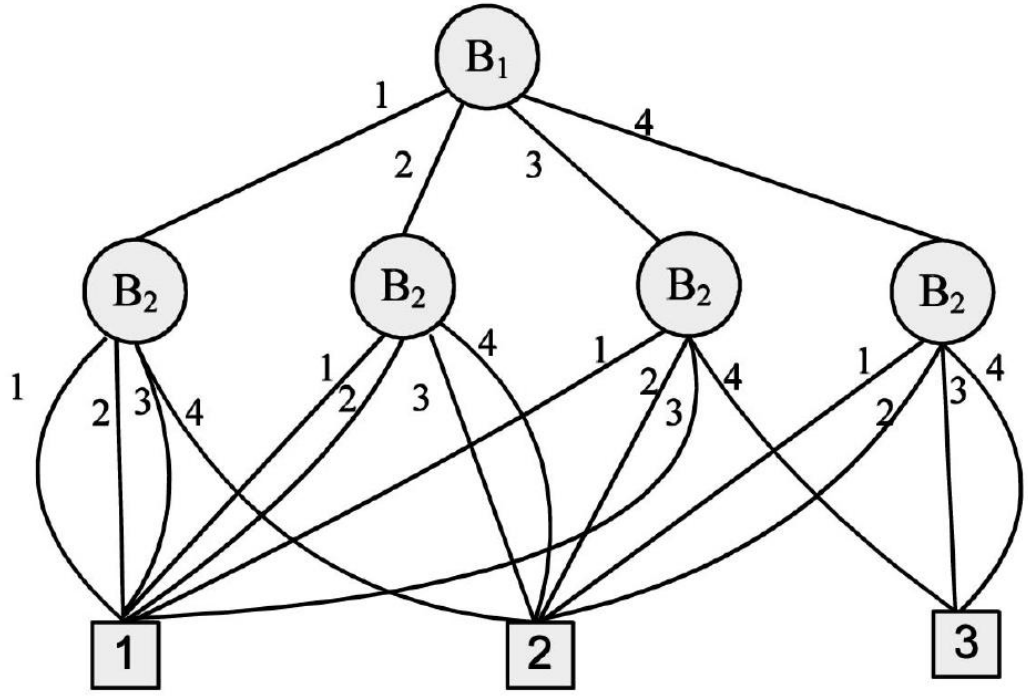
(b) BDD for variables in the best order

(c) BDD for variables in the worst order

Bryant (1986), etc.

Decision Diagram Basics

- Binary decision diagrams encode Boolean functions
  - Historically used for circuit design & verification
  - Easily generalized to multivalued decision diagrams



Reduced Decision Diagrams

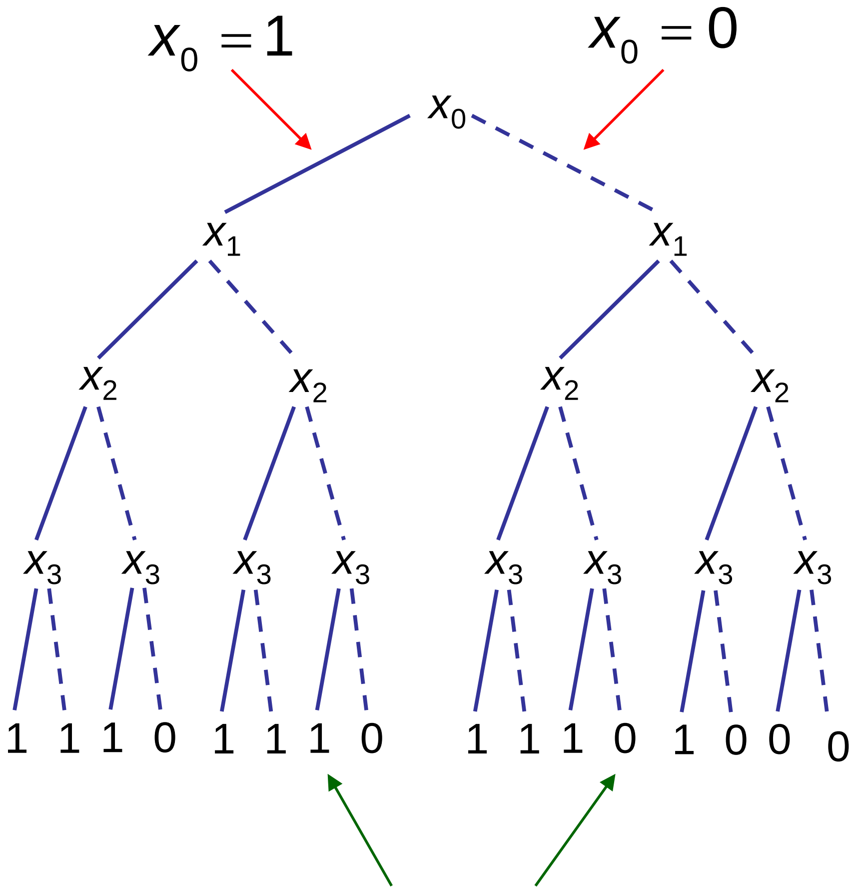
- There is a **unique reduced** DD representing any given Boolean function.
  - Once the variable ordering is specified.

Bryant (1986)

- The reduced DD can be viewed as a branching tree with **redundancy** removed.
  - Superimpose isomorphic subtrees.
  - Remove redundant nodes.

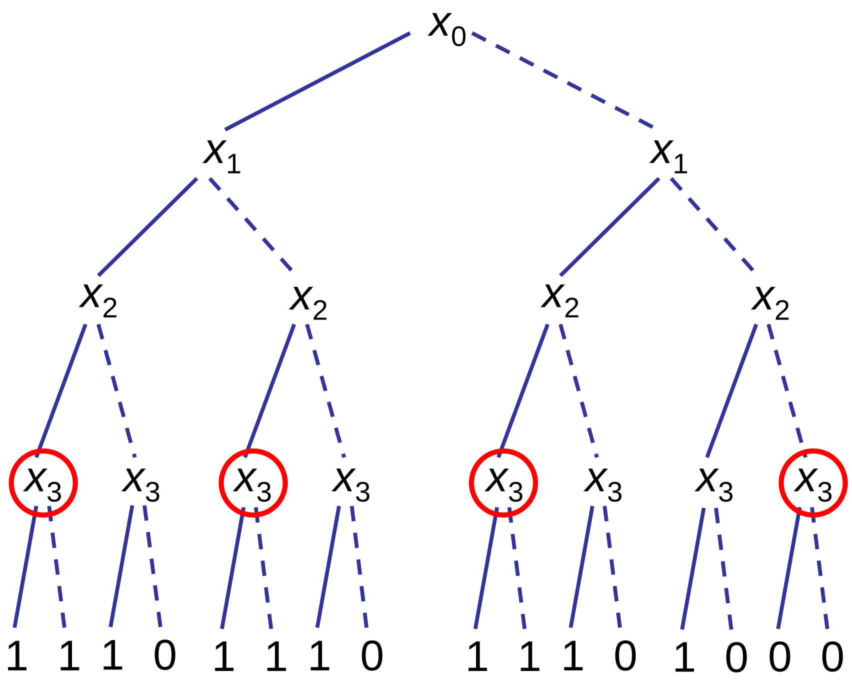
Branching tree for 0-1 inequality

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



1 indicates feasible solution,
0 infeasible

1

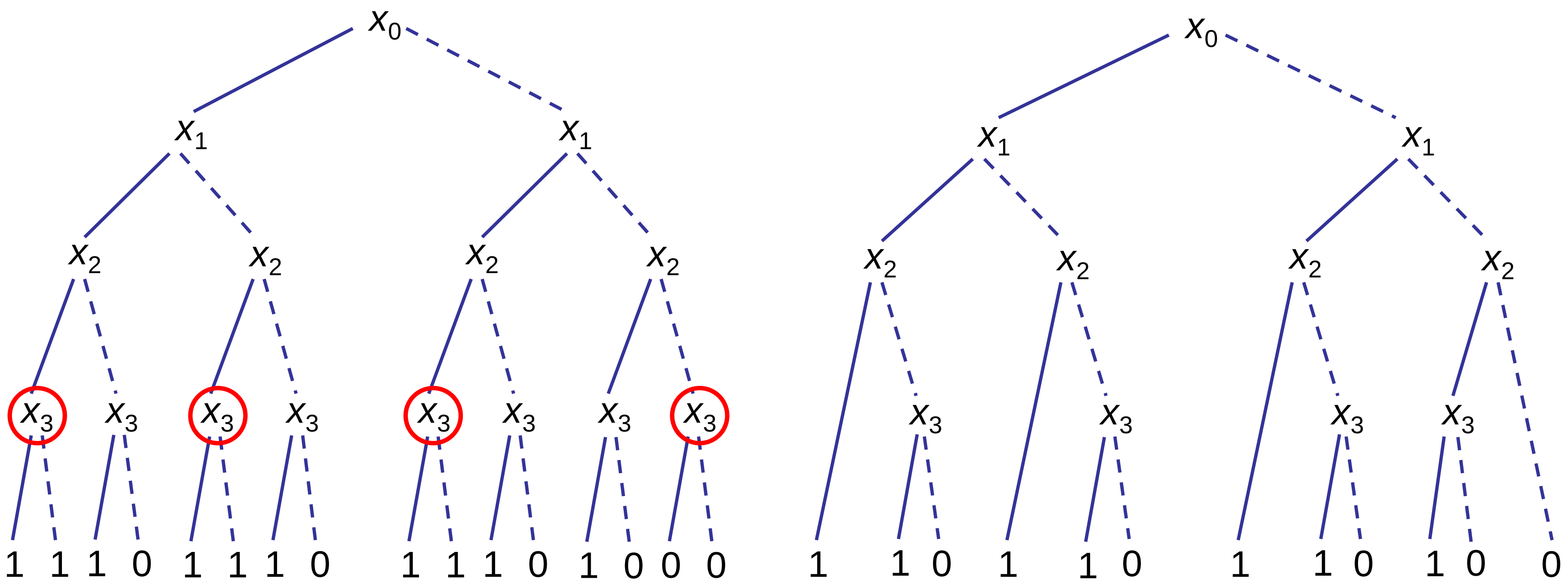


Branching tree for 0-1 inequality

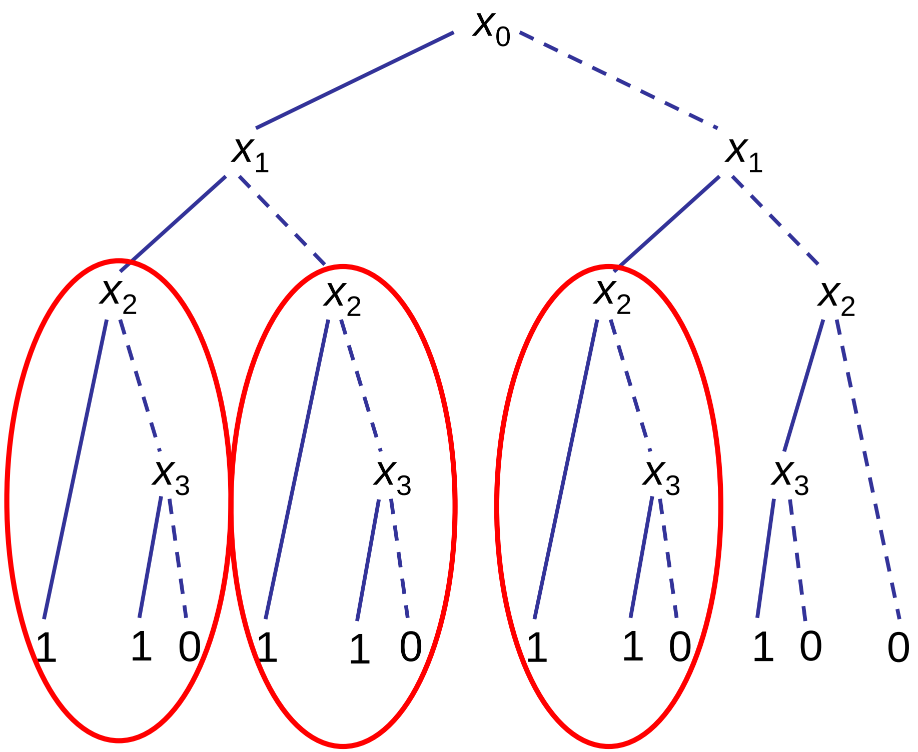
$$2x_0 + 3x_1 + 5x_2 + 5x_3 \geq 7$$

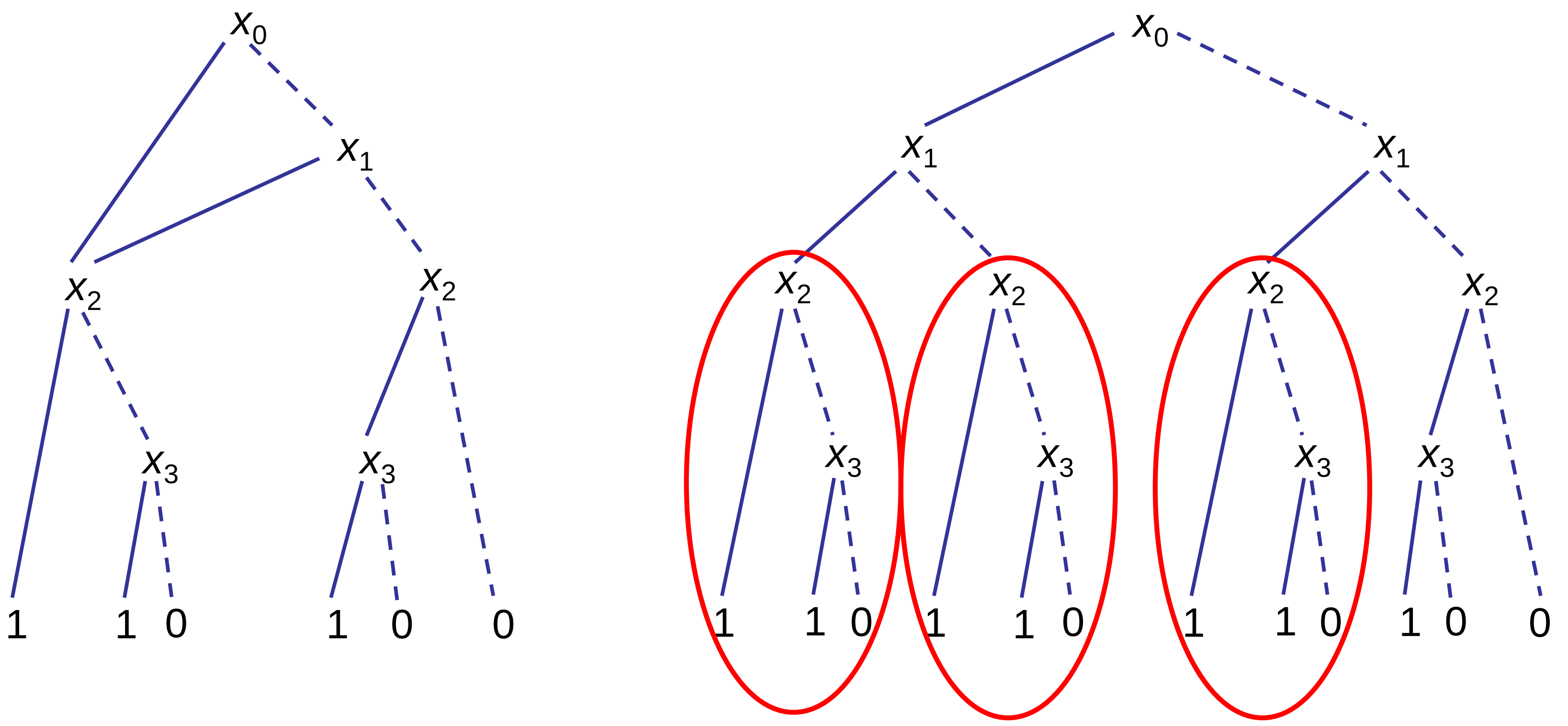
Remove redundant nodes…

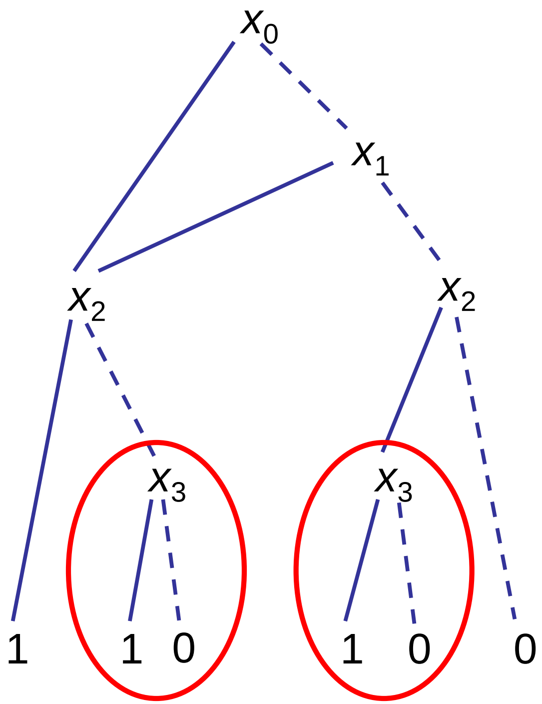
1


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

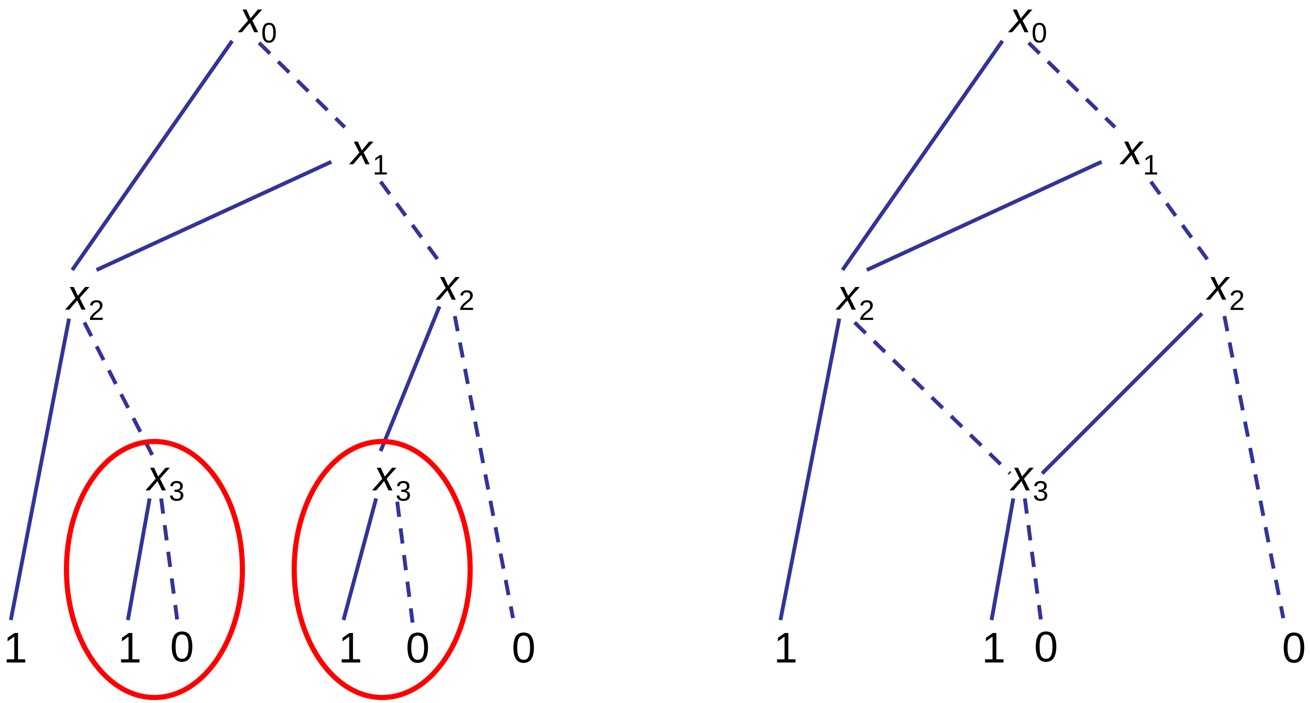
Superimpose identical subtrees…



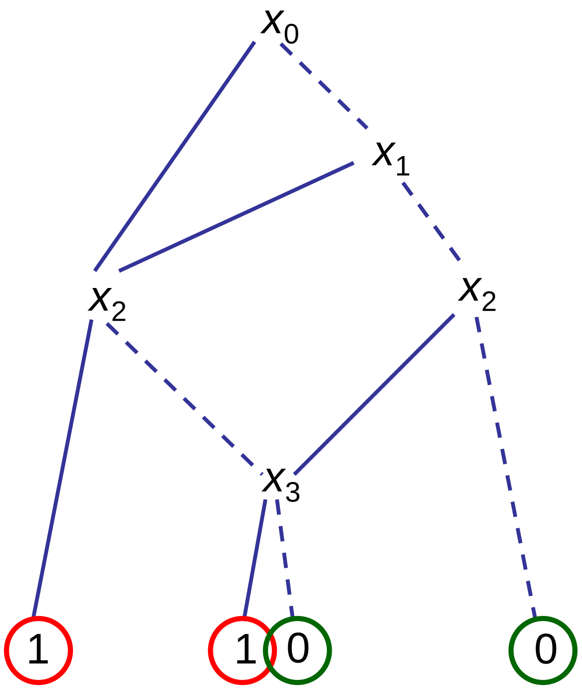

x_0
x_1
x_2
x_3
1
1
0
1
0
0
x_0
x_1
x_1
x_2
x_2
x_2
x_2
x_3
x_3
x_3
x_3
1
1
0
1
1
0
1
1
0
1
0
0

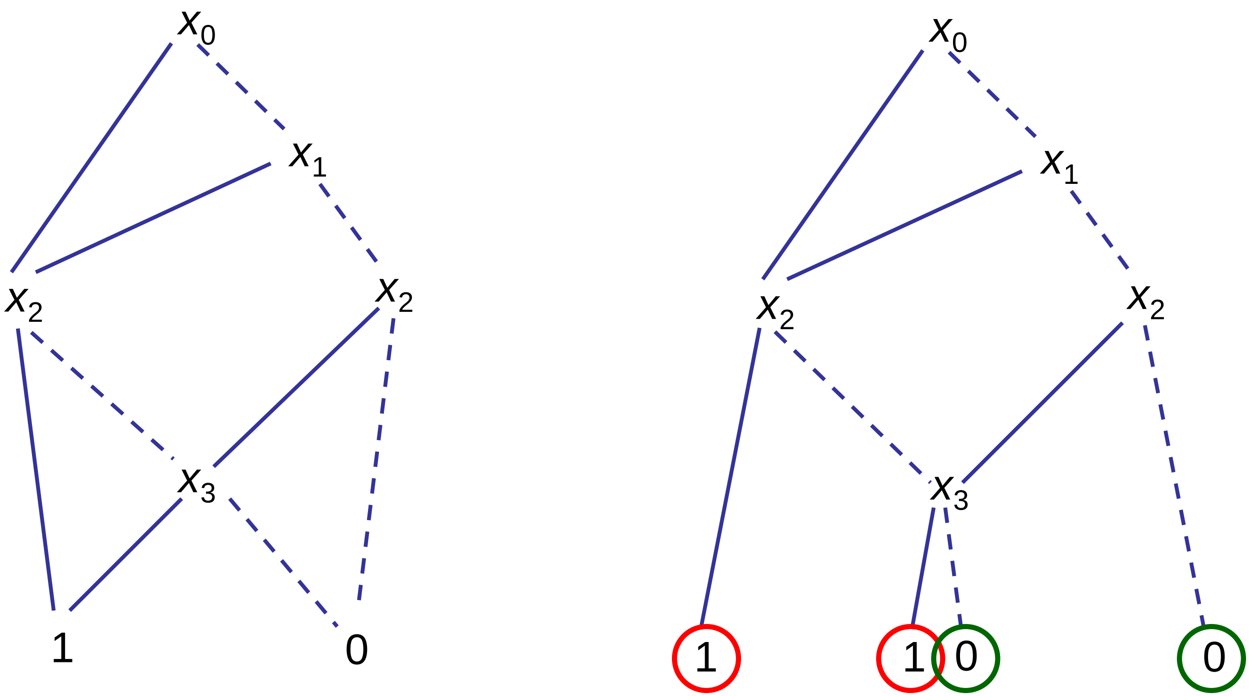


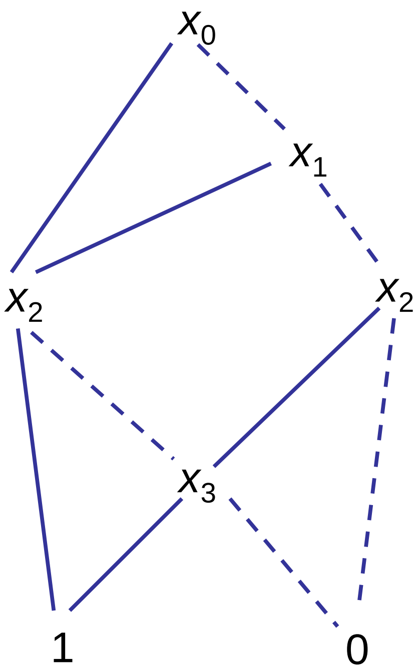
Superimpose identical subtrees…

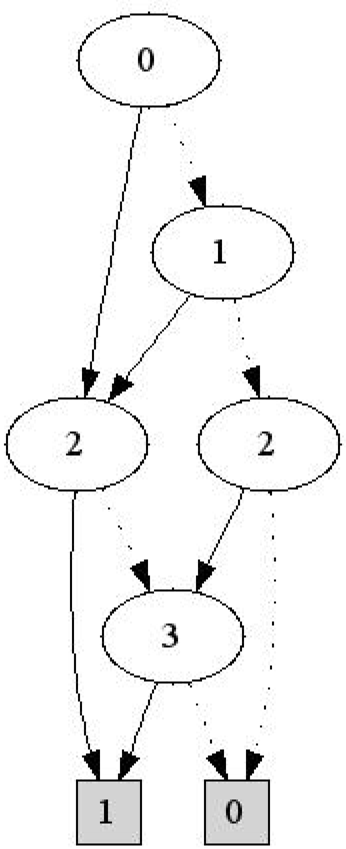

x_0
x_1
x_2
x_2
x_3
x_3
1
1
0
1
0
0
x_0
x_1
x_2
x_2
x_3
1
1
0
0

Superimpose identical leaf nodes…




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





as generated by software

Reduced Decision Diagrams

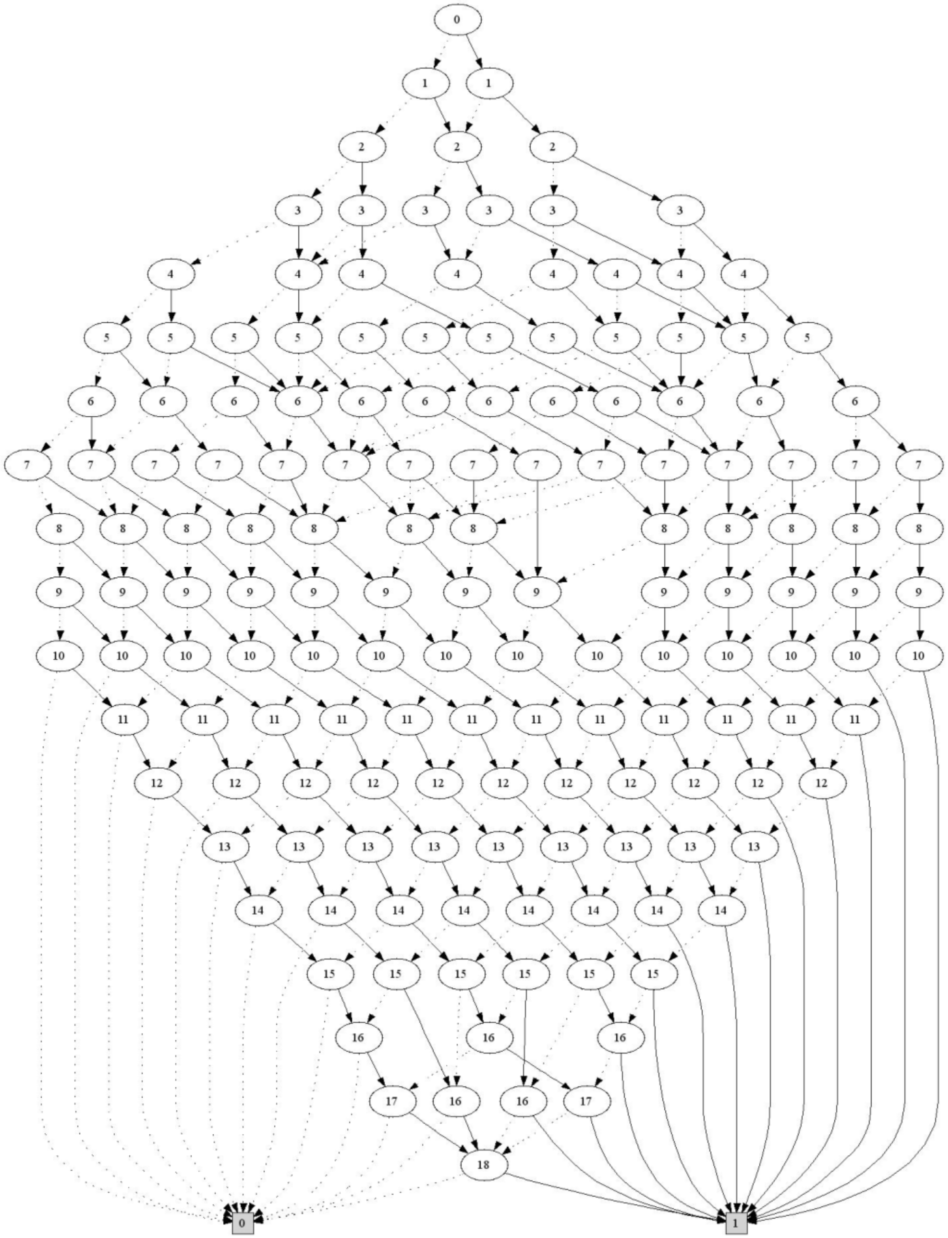
- Reduced DD for a knapsack constraint can be surprisingly small…

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 maximal feasible solutions (or minimal covers)

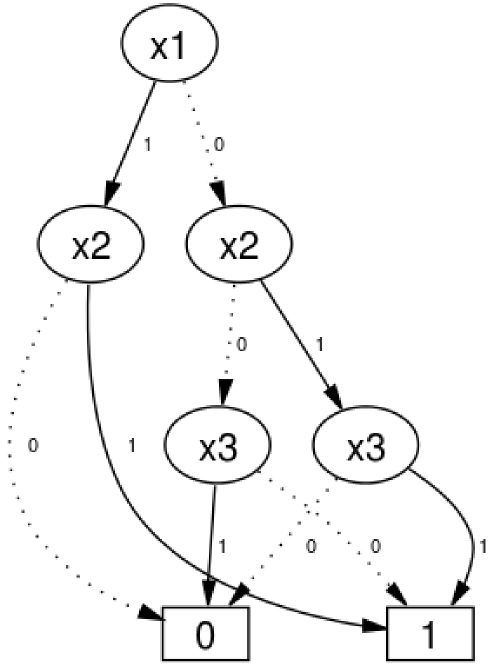
But its reduced BDD has only 152 nodes…



Optimization with Exact Decision Diagrams

- Decision diagrams can represent feasible set
  - Remove paths to 0.
  - Paths to 1 are feasible solutions.
  - Associate costs with arcs.
  - Find longest/shortest path

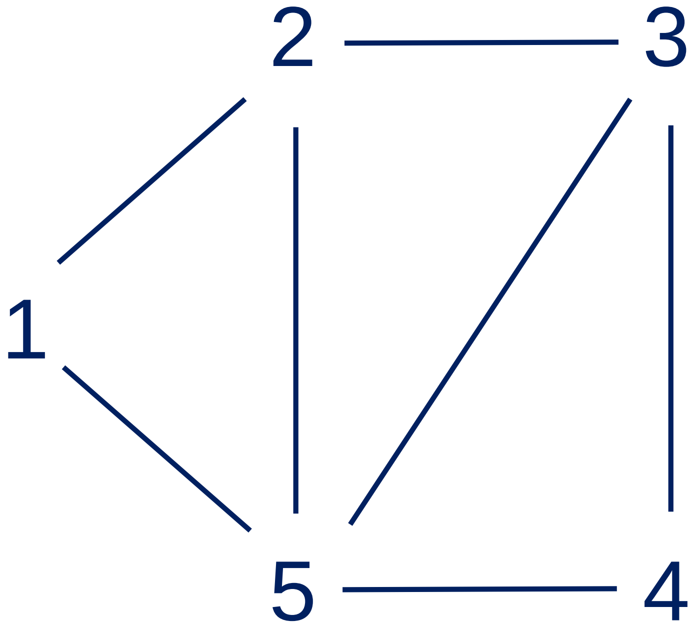
Hadžić and JH (2006, 2007)

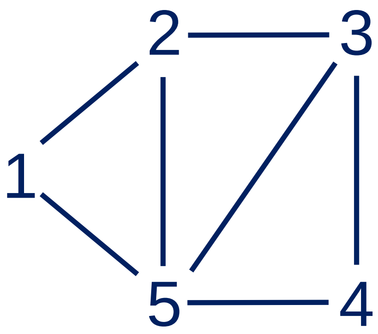


Stable Set Problem

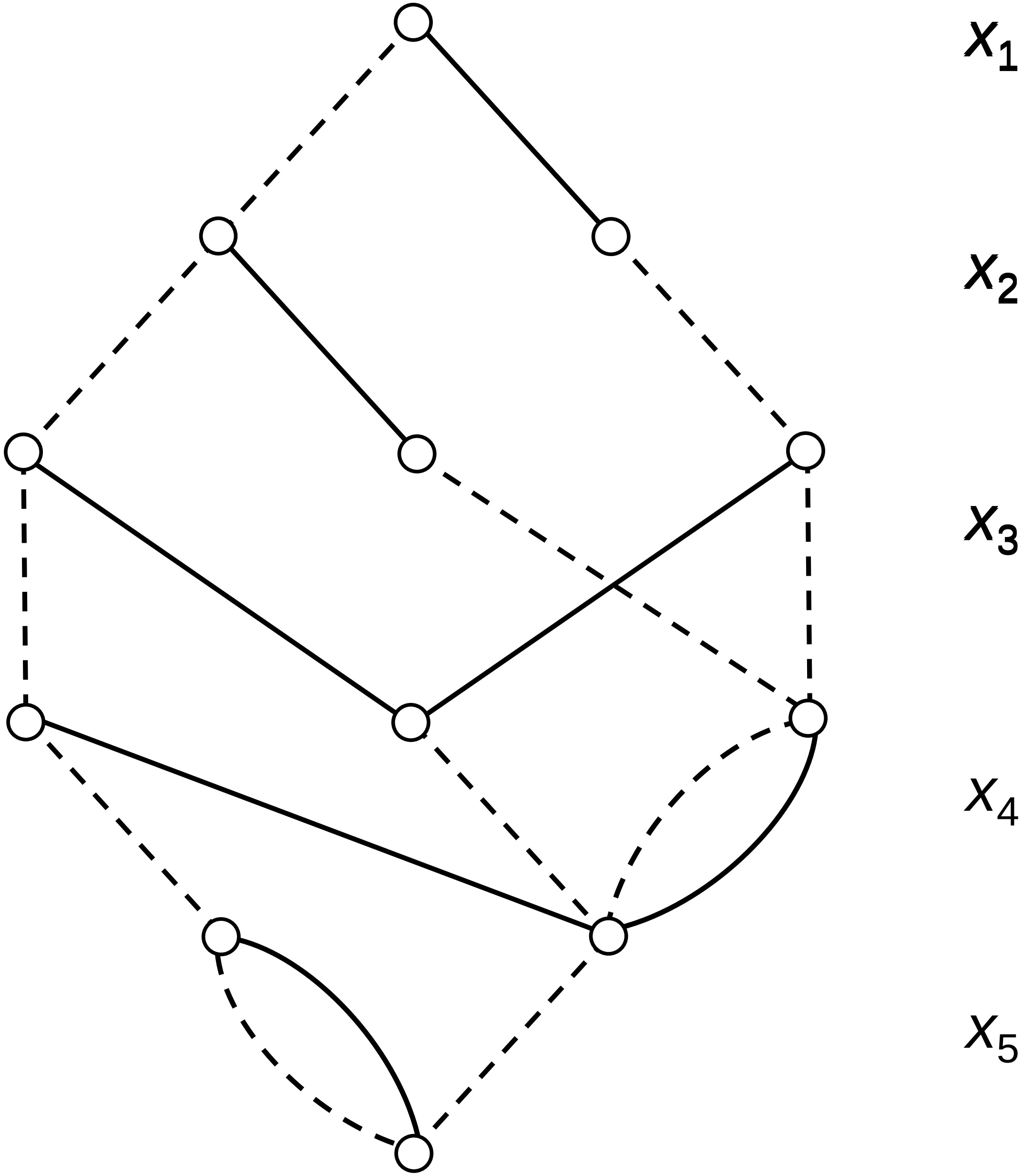
Let each vertex have weight w_i

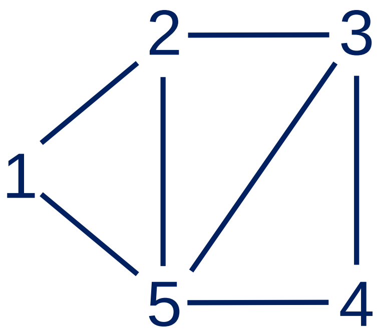
Select nonadjacent vertices to maximize $\sum_i w_i x_i$



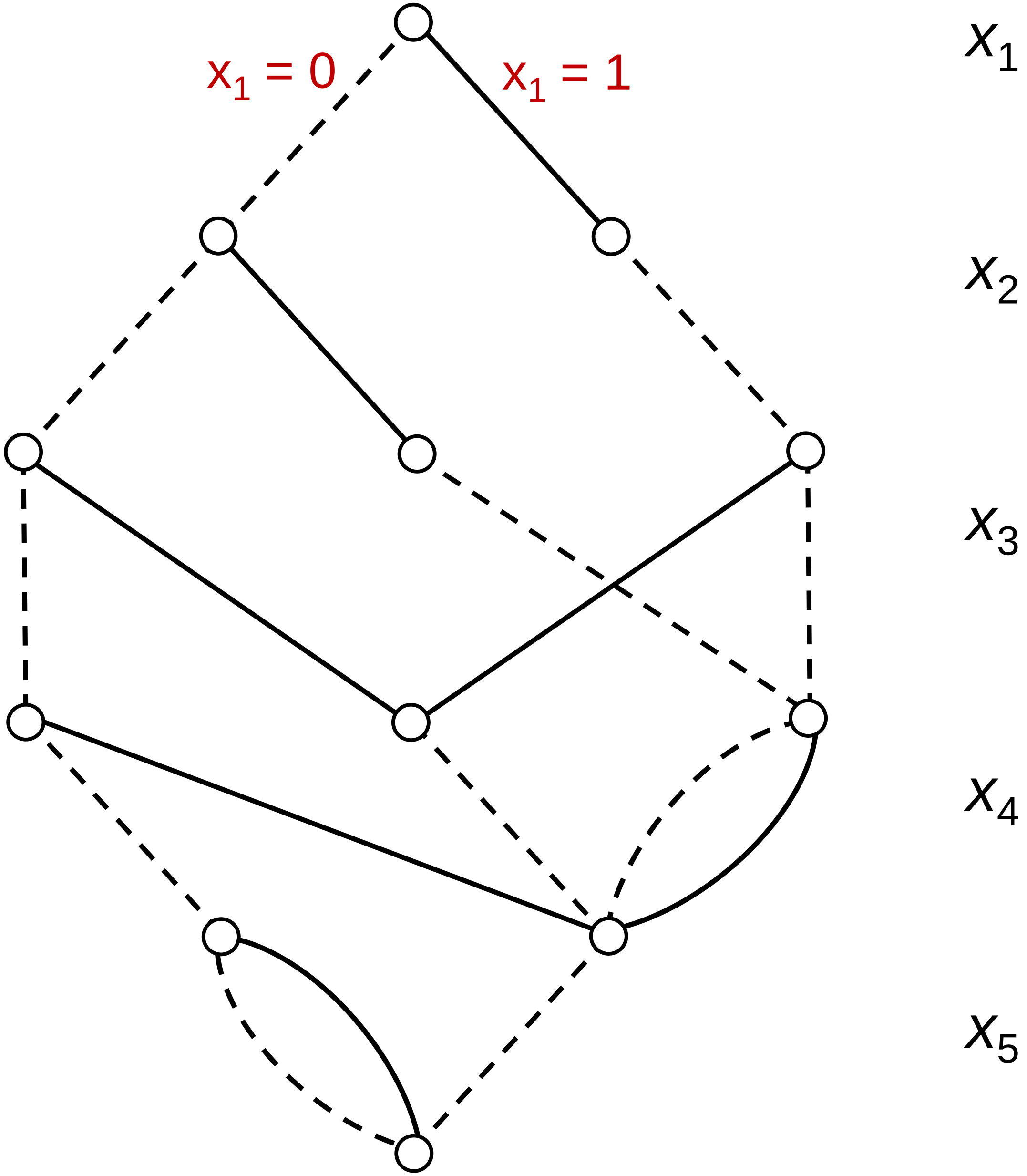


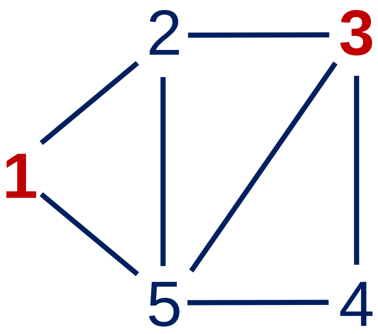
Exact DD for stable set problem



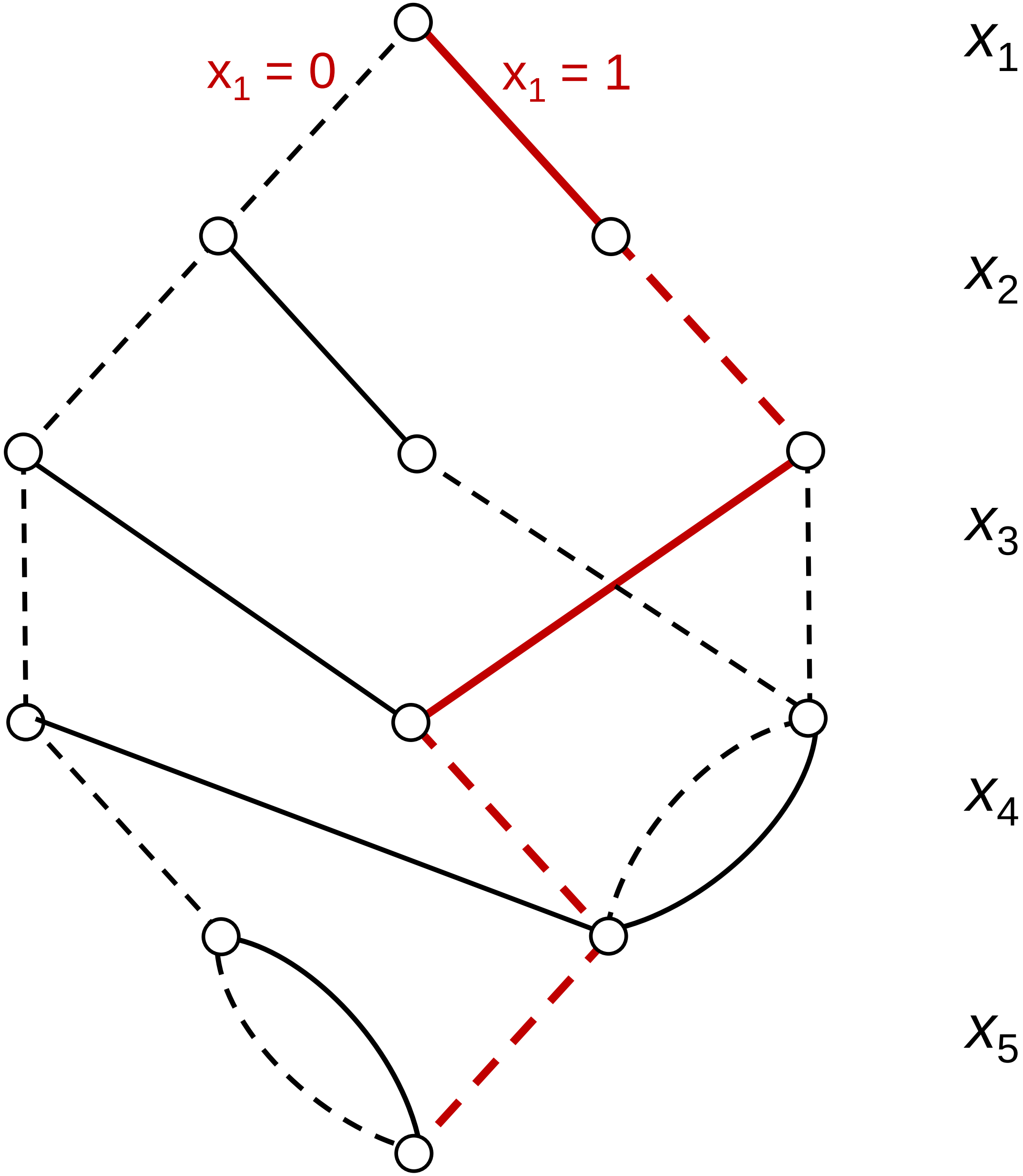


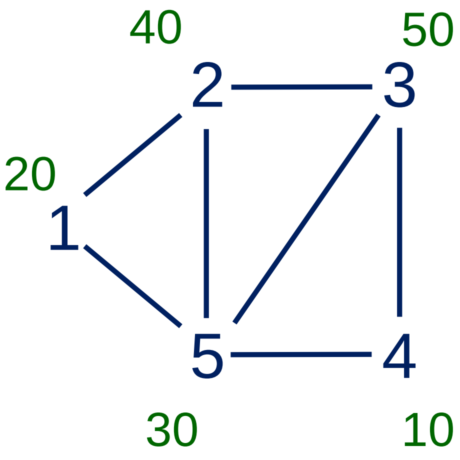
Exact DD for stable set problem



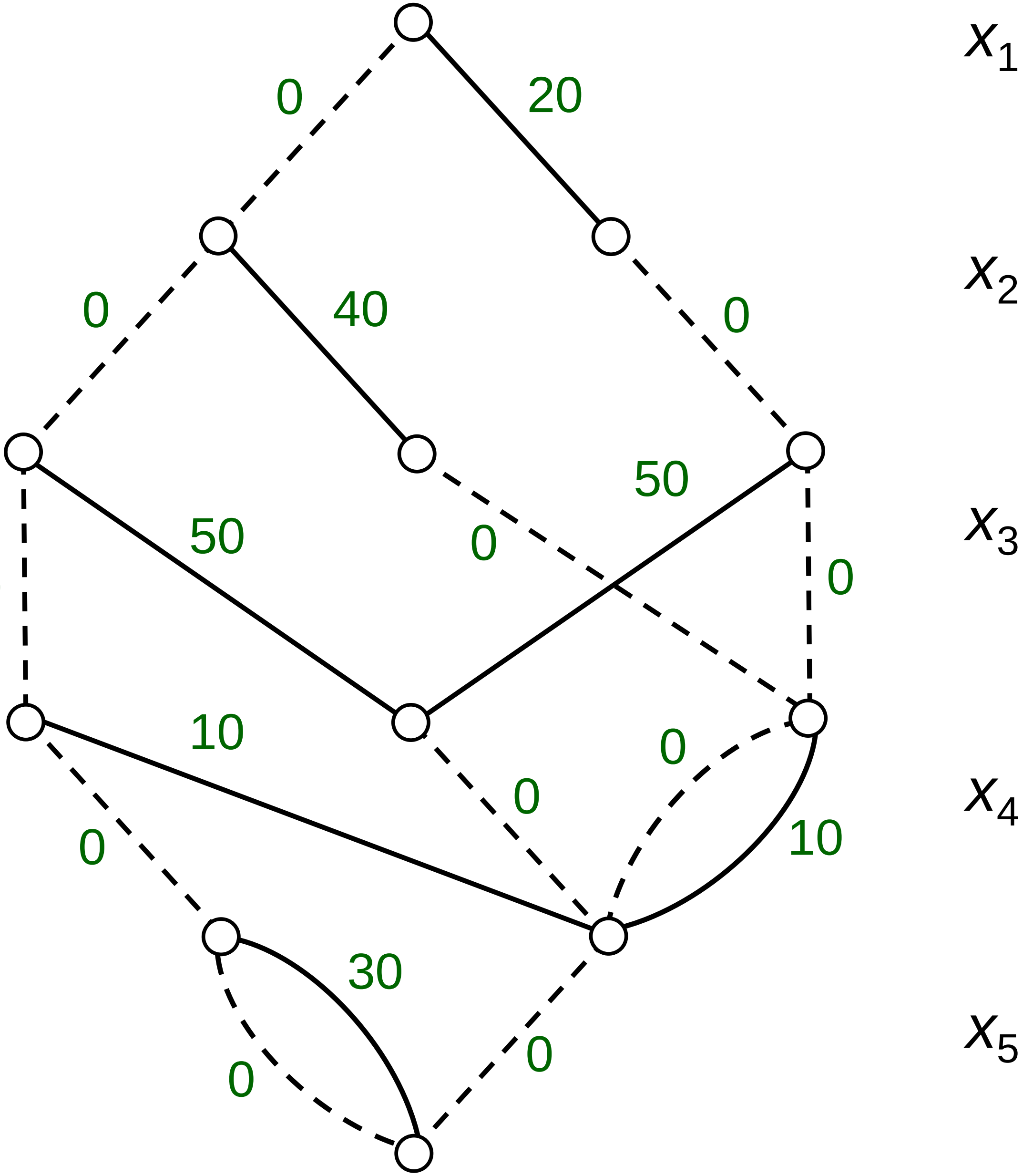


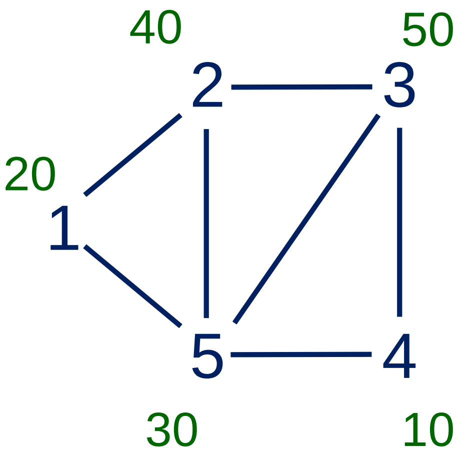
Paths from top to bottom correspond to the 9 feasible solutions



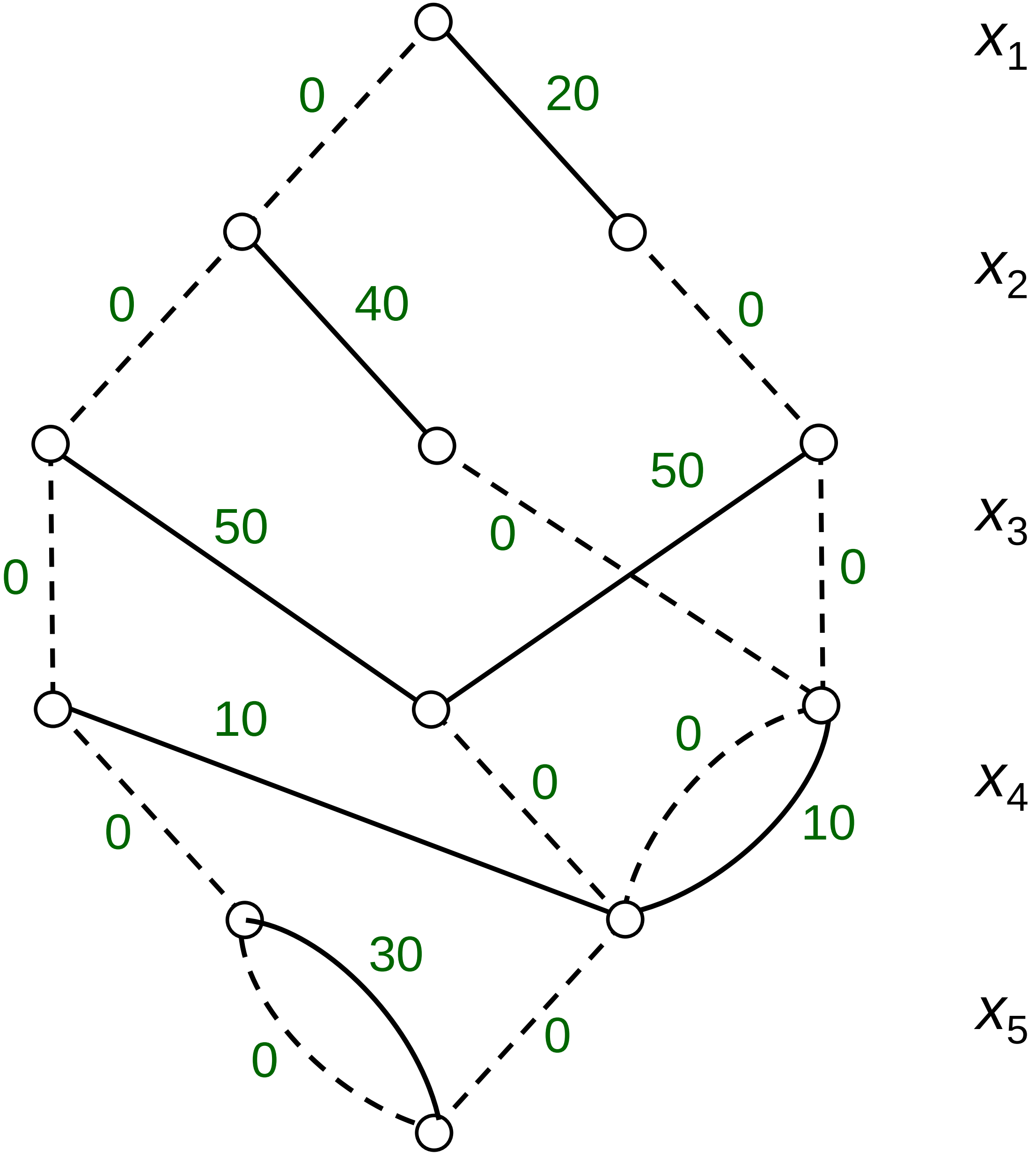


For objective function, associate weights with arcs

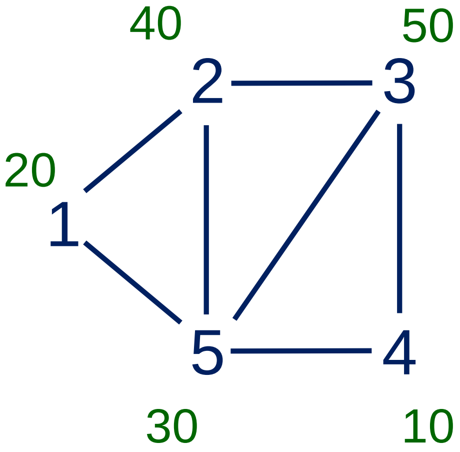




For objective function, associate weights with arcs

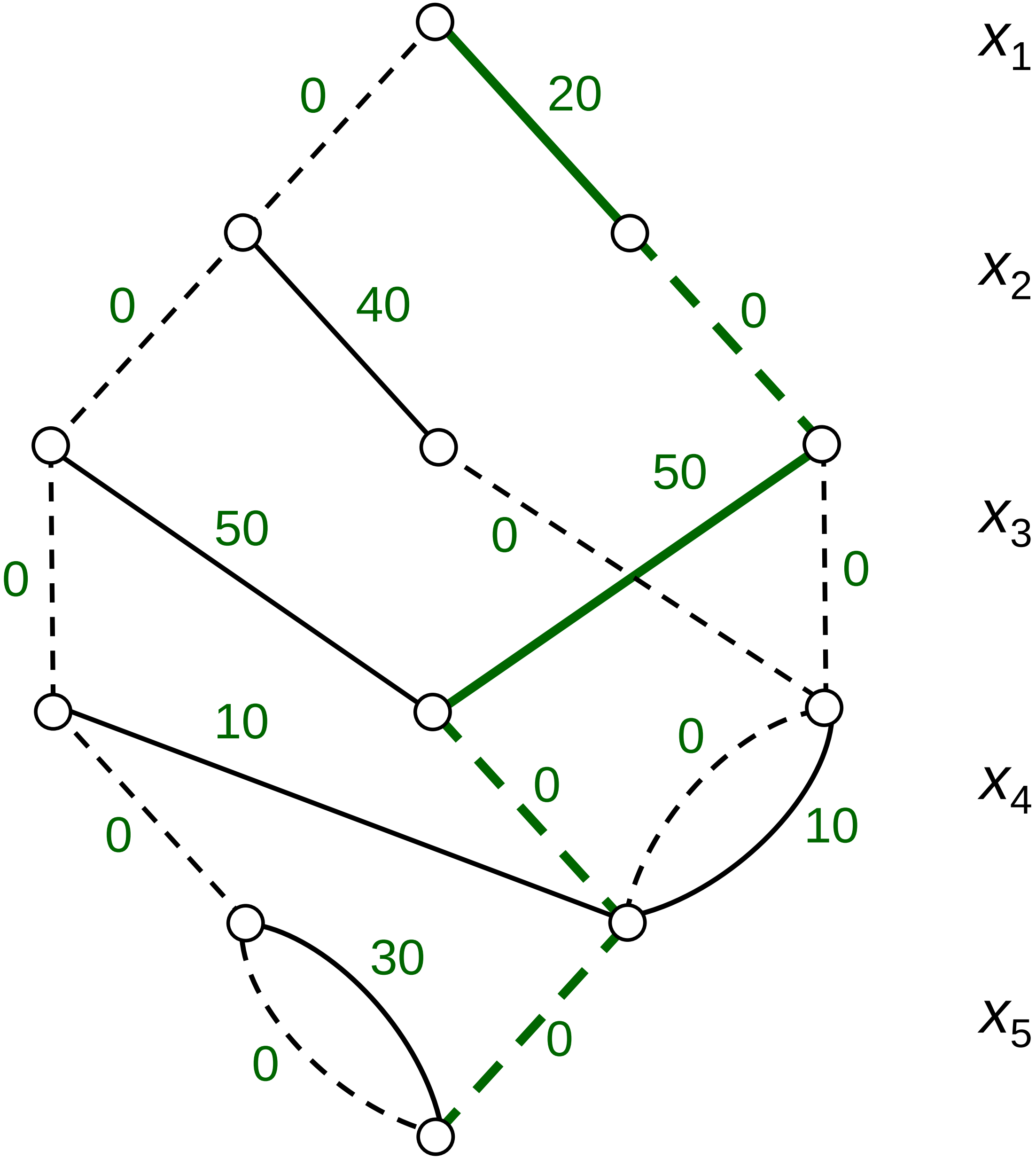


Optimal solution is **longest path**



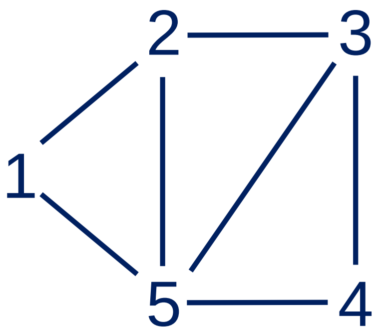
For objective function, associate weights with arcs

Optimal solution is **longest path**



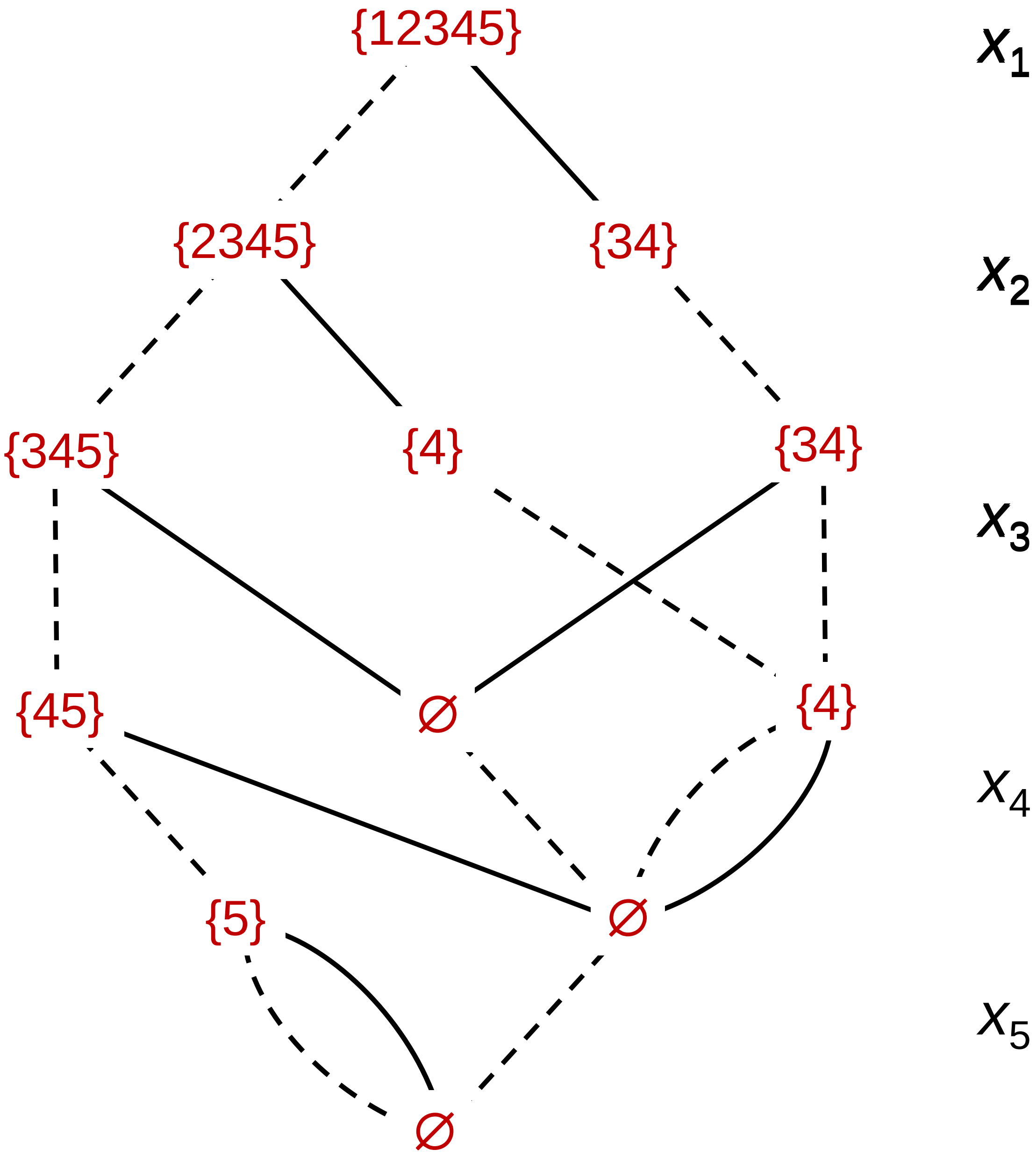
Exact DD Compilation

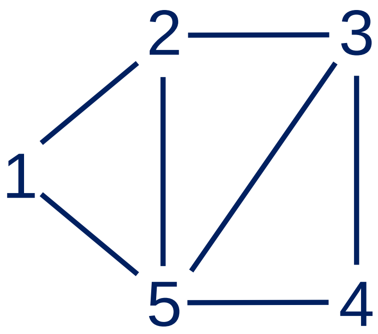
- Build an exact DD by associating a **state** with each node.
    - Merge nodes with **identical states**.



Exact DD for stable set problem

To build DD, associate **state** with each node





{12345}

x_1

x_2

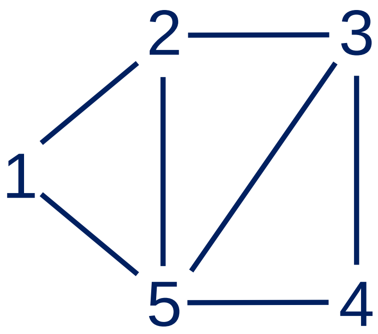
x_3

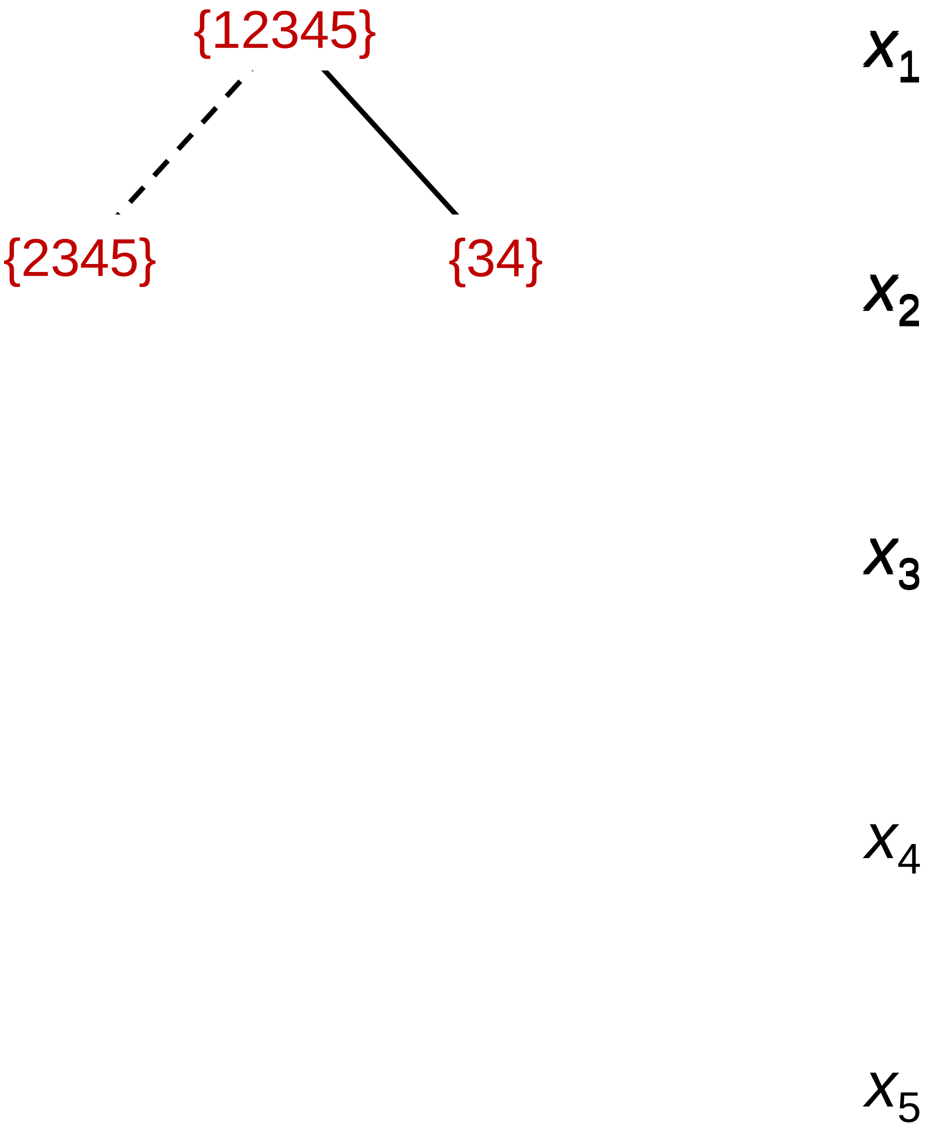
x_4

x_5

Exact DD for
stable set
problem

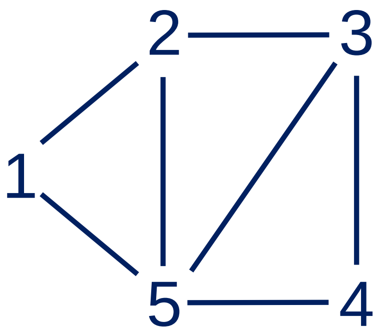
To build DD,
associate **state**
with each node

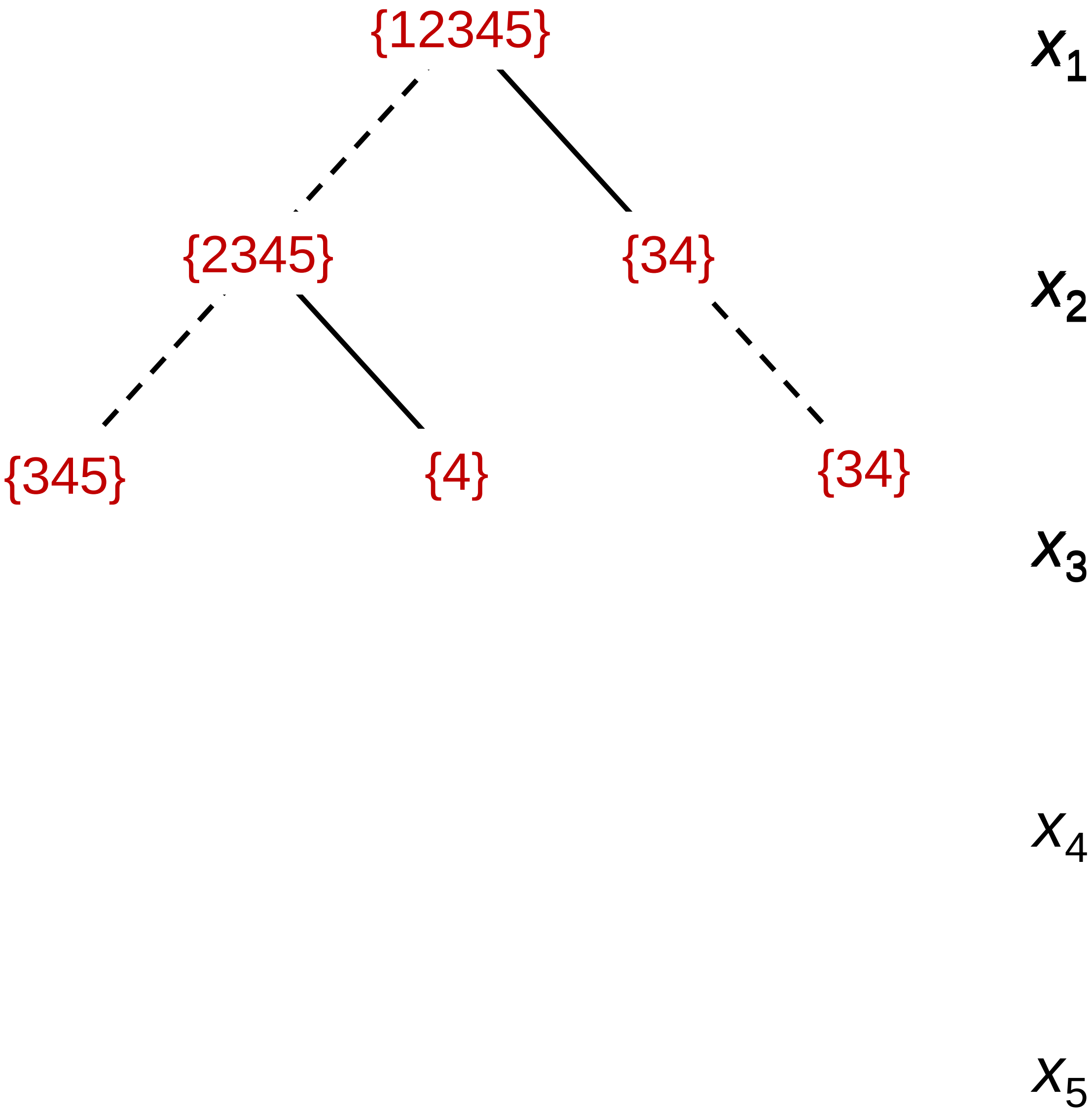




Exact DD for stable set problem

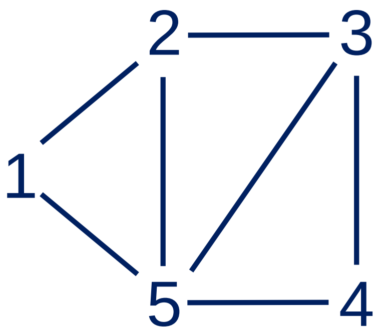
To build DD, associate **state** with each node

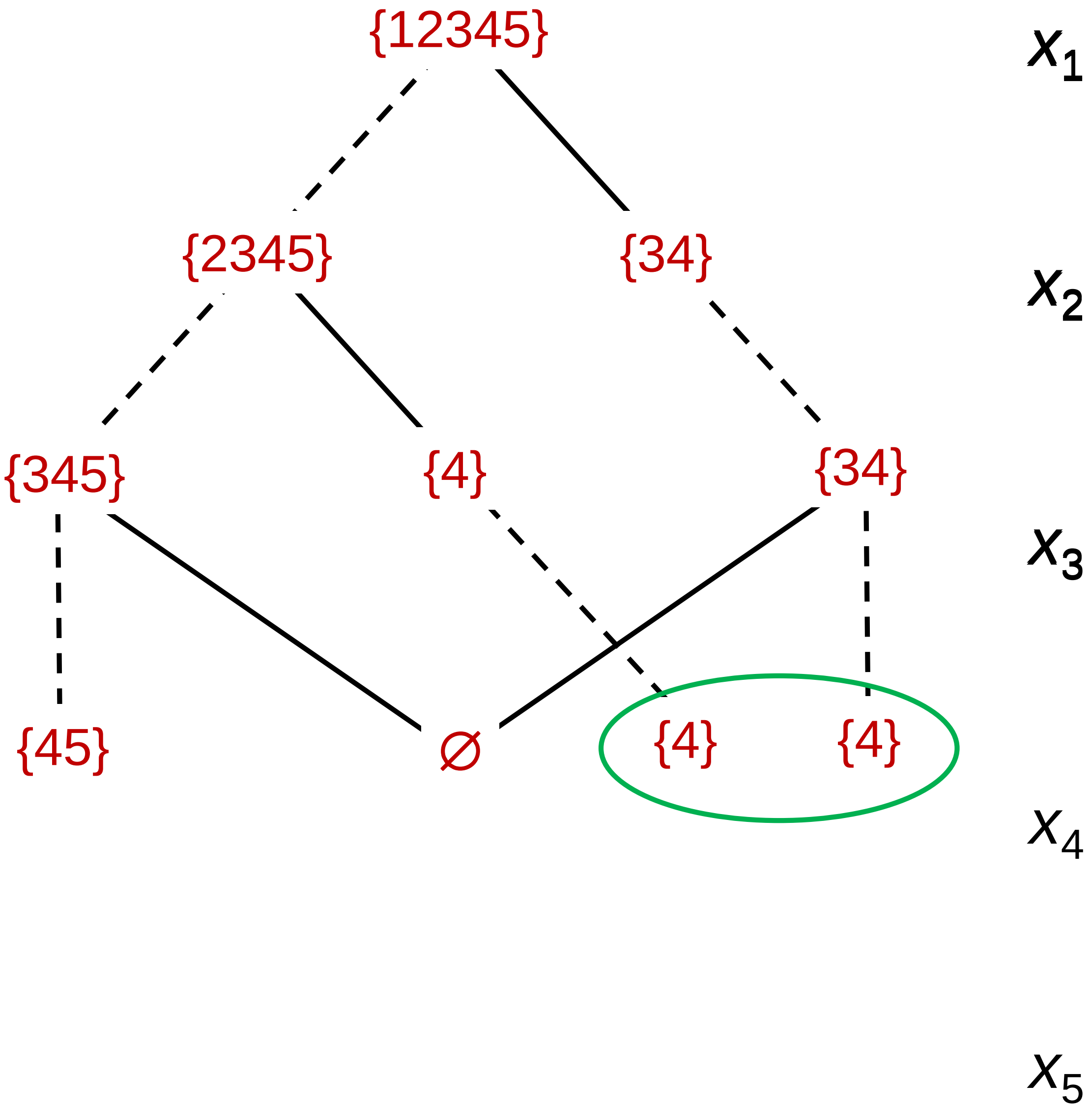




Exact DD for
stable set
problem

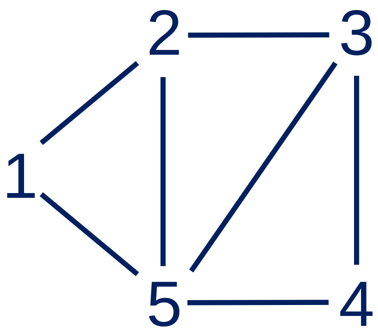
To build DD,
associate **state**
with each node

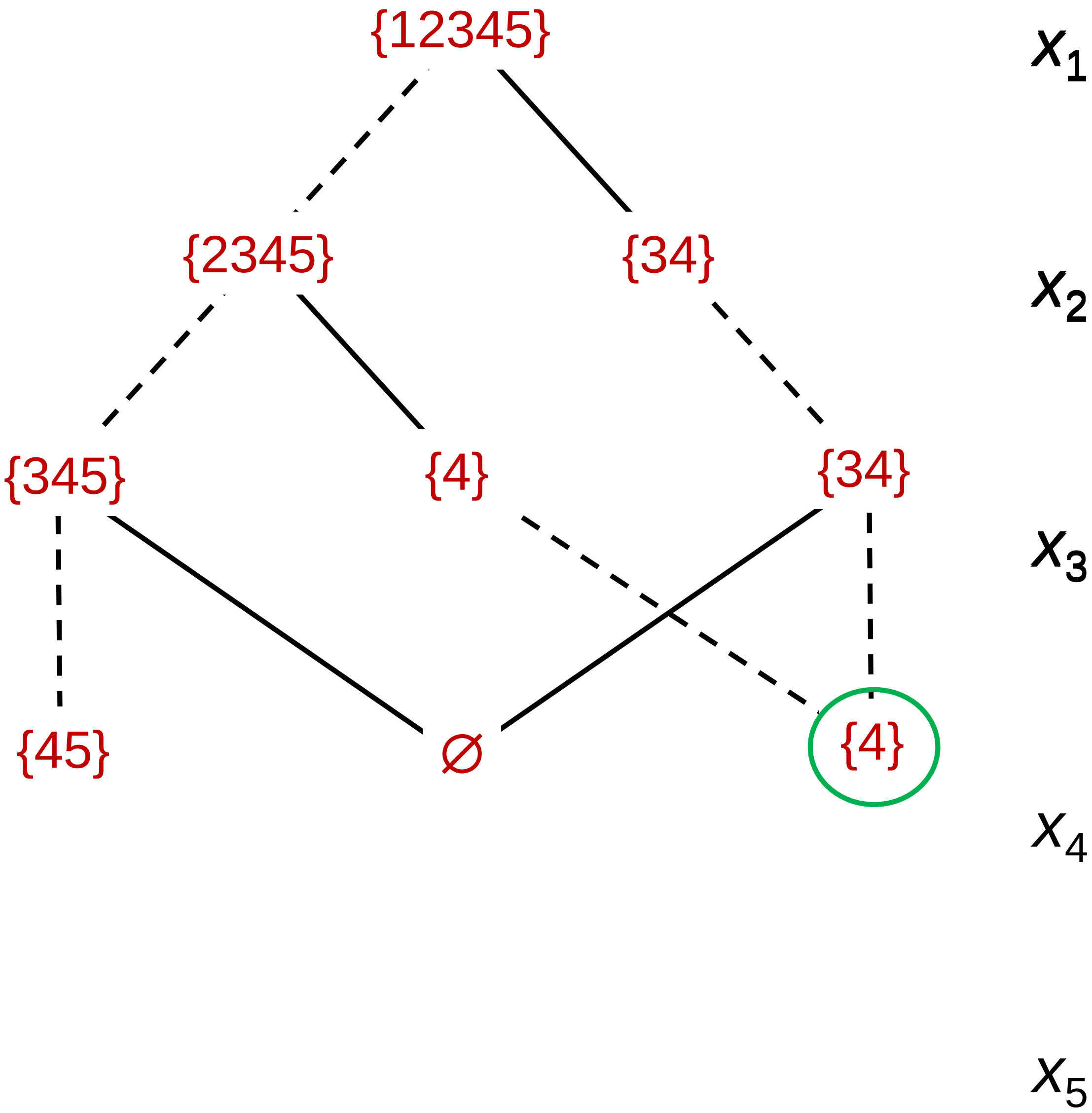




Exact DD for stable set problem

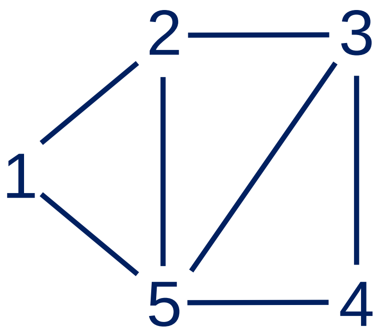
Merge nodes that correspond to the same state





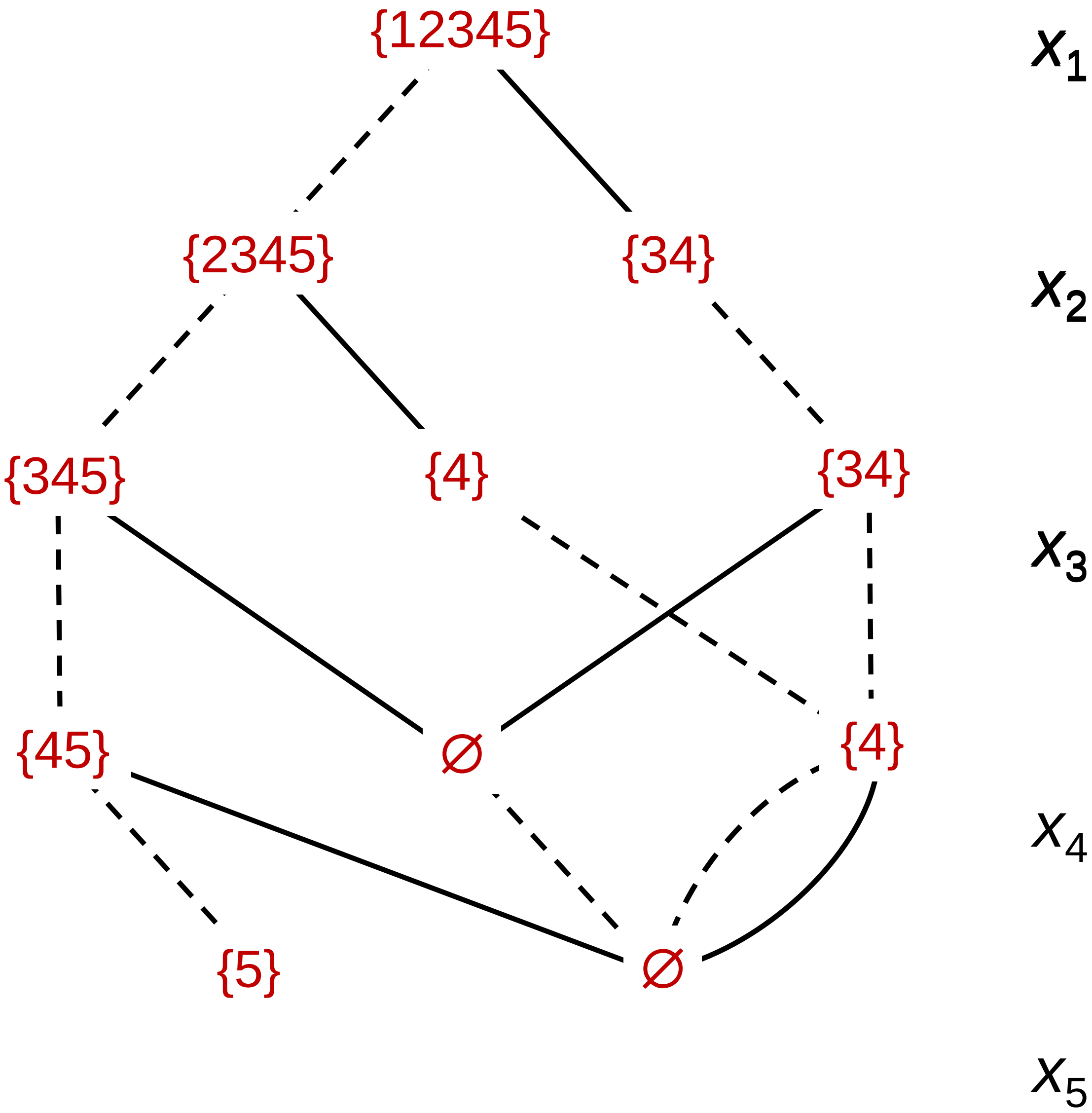
Exact DD for stable set problem

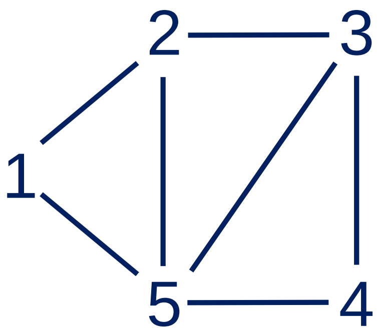
Merge nodes that correspond to the same state



Exact DD for stable set problem

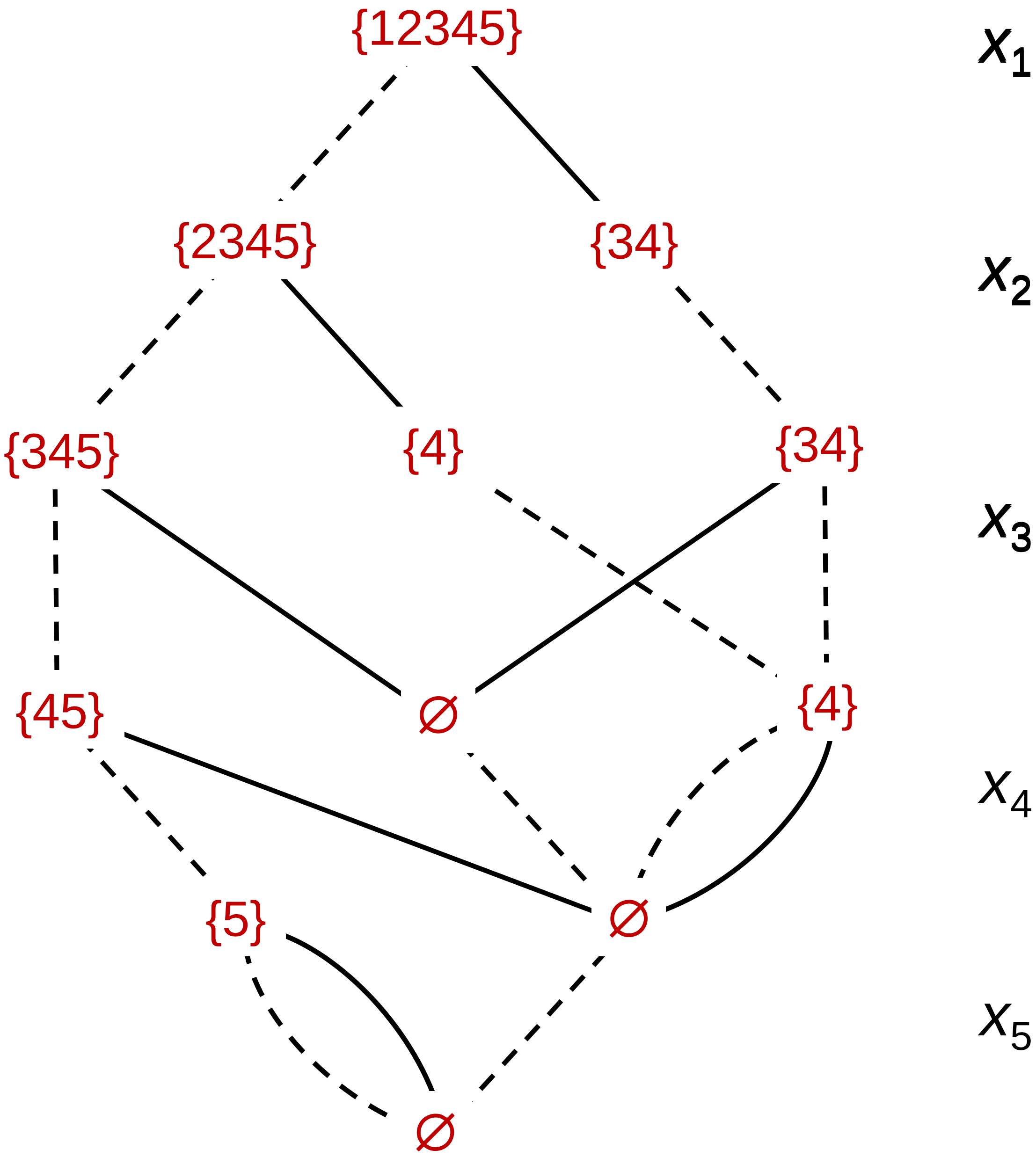
To build DD, associate **state** with each node





Exact DD for stable set problem

Resulting DD is **not necessarily reduced** (it is in this case).



Relaxed Decision Diagrams

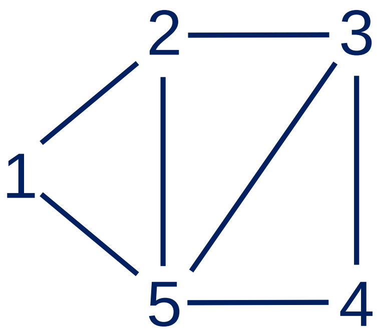
- A **relaxed** DD represents a superset of feasible set.
  - Shortest (longest) path length is a **bound** on optimal value.
  - **Size of DD is controlled**.
  - Analogous to LP relaxation in IP, but **discrete**.
  - Does **not** require **linearity**, **convexity**, or **inequality** constraints.

Andersen, Hadžić, JH, Tiedemann (2007)

Relaxation by Node Merger

- One way to relax a DD is to **merge nodes** during top-down compilation.
  - Make sure **state** of merged node excludes no feasible solutions.

Hoda, van Hoeve, JH (2010)



{12345}

x_1

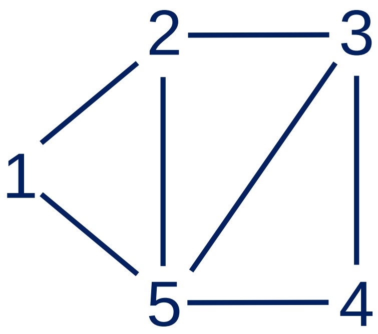
x_2

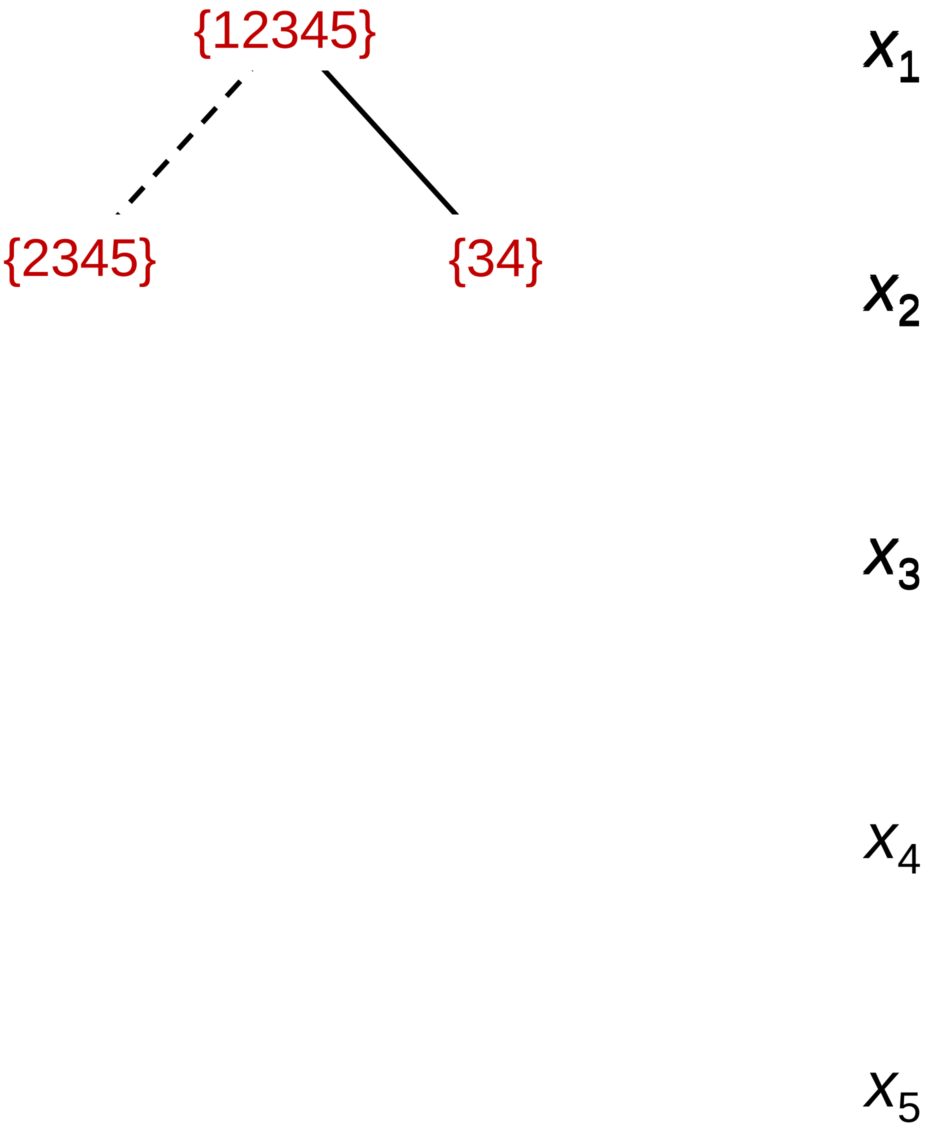
x_3

x_4

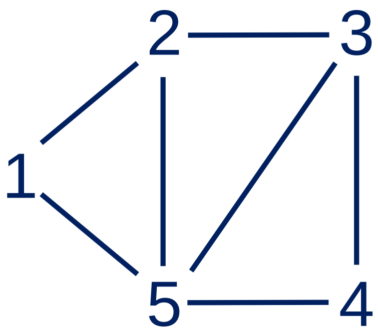
x_5

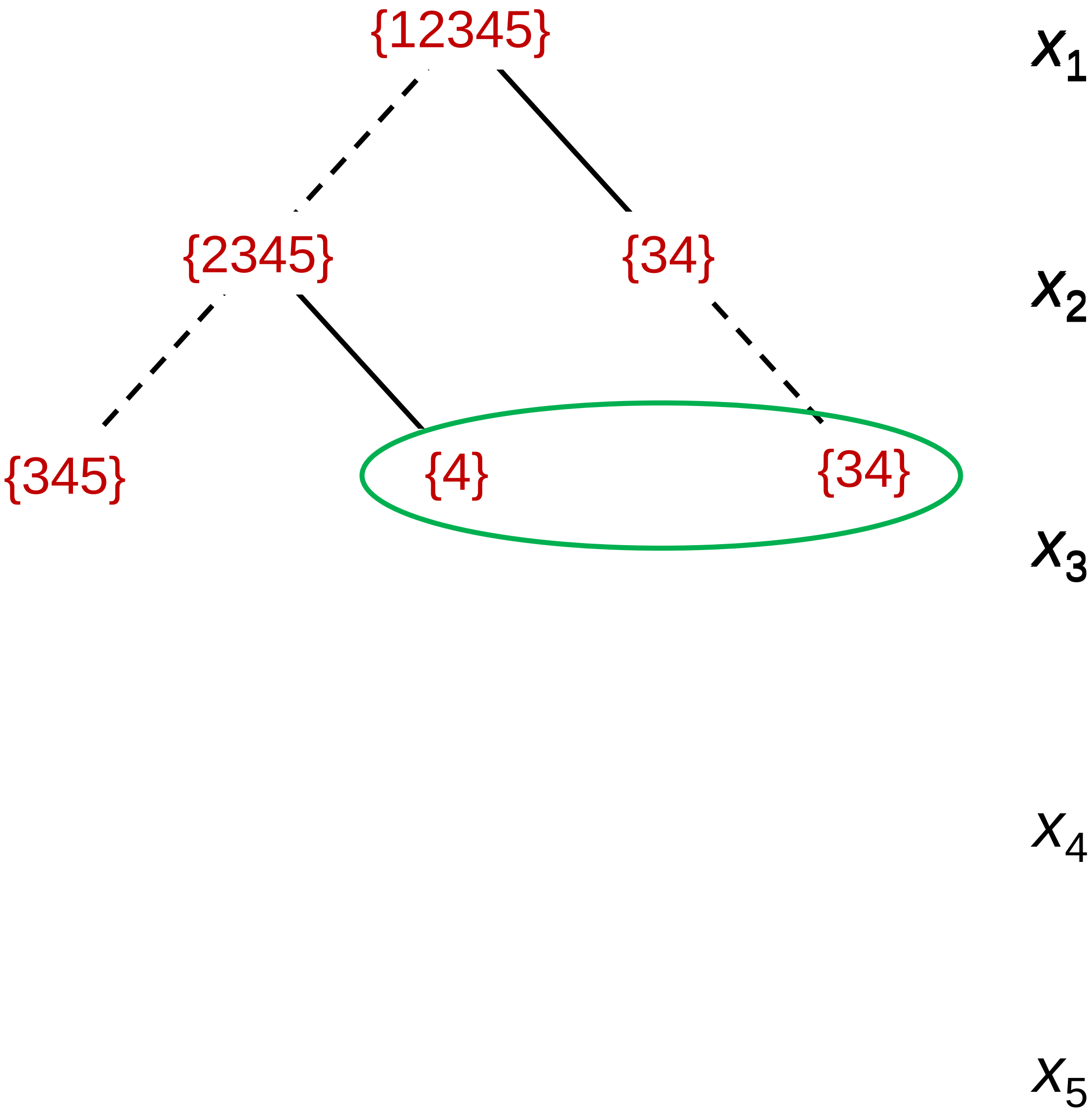
To build **relaxed** DD, merge some additional nodes as we go along





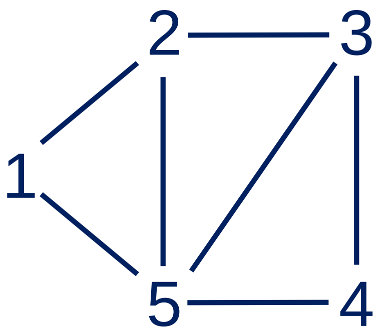
To build **relaxed** DD, merge some additional nodes as we go along

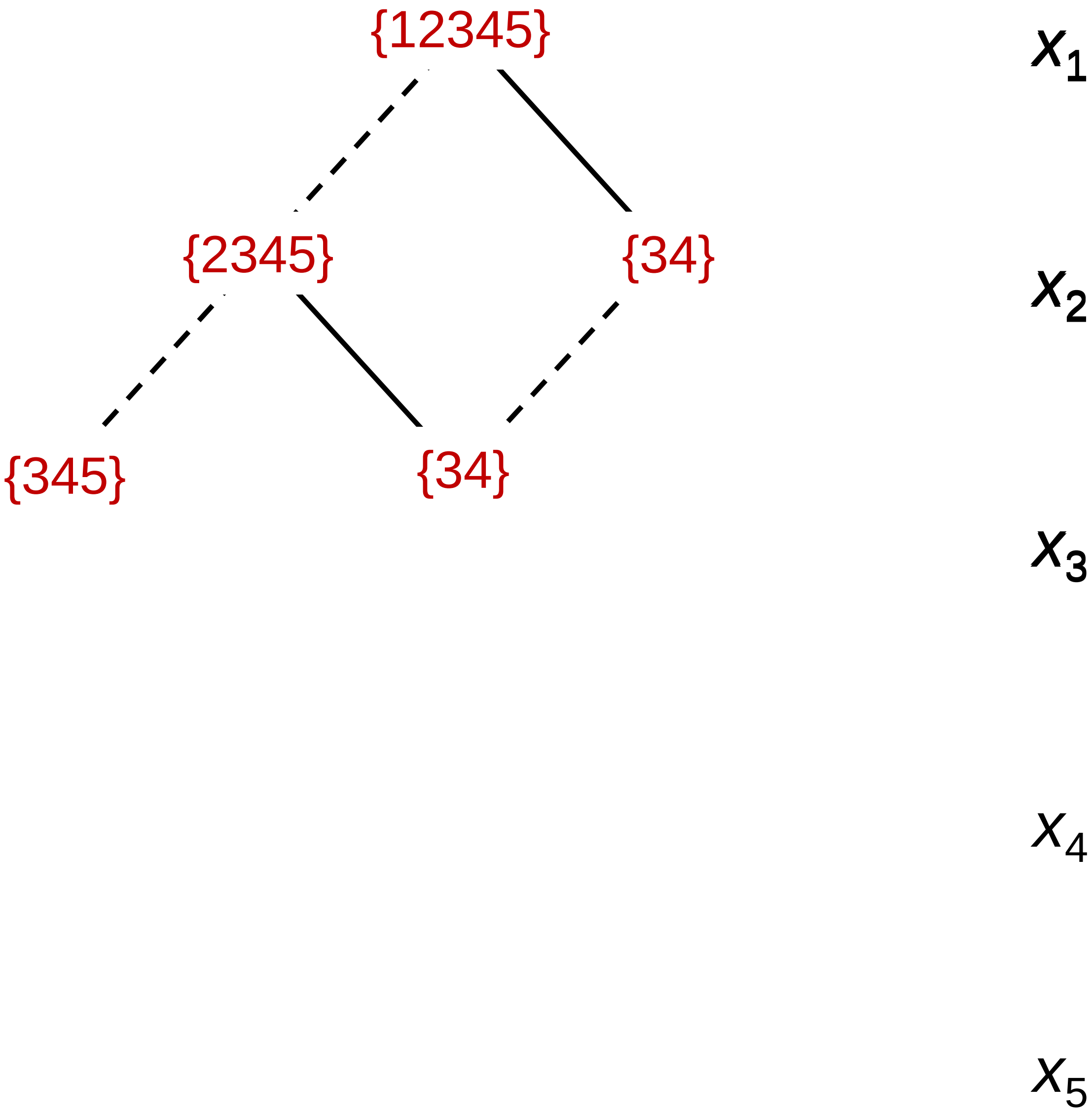




To build **relaxed** DD, merge some additional nodes as we go along.

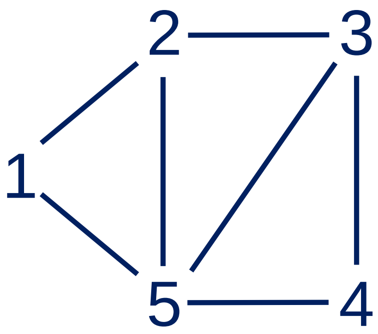
Take the **union** of merged states

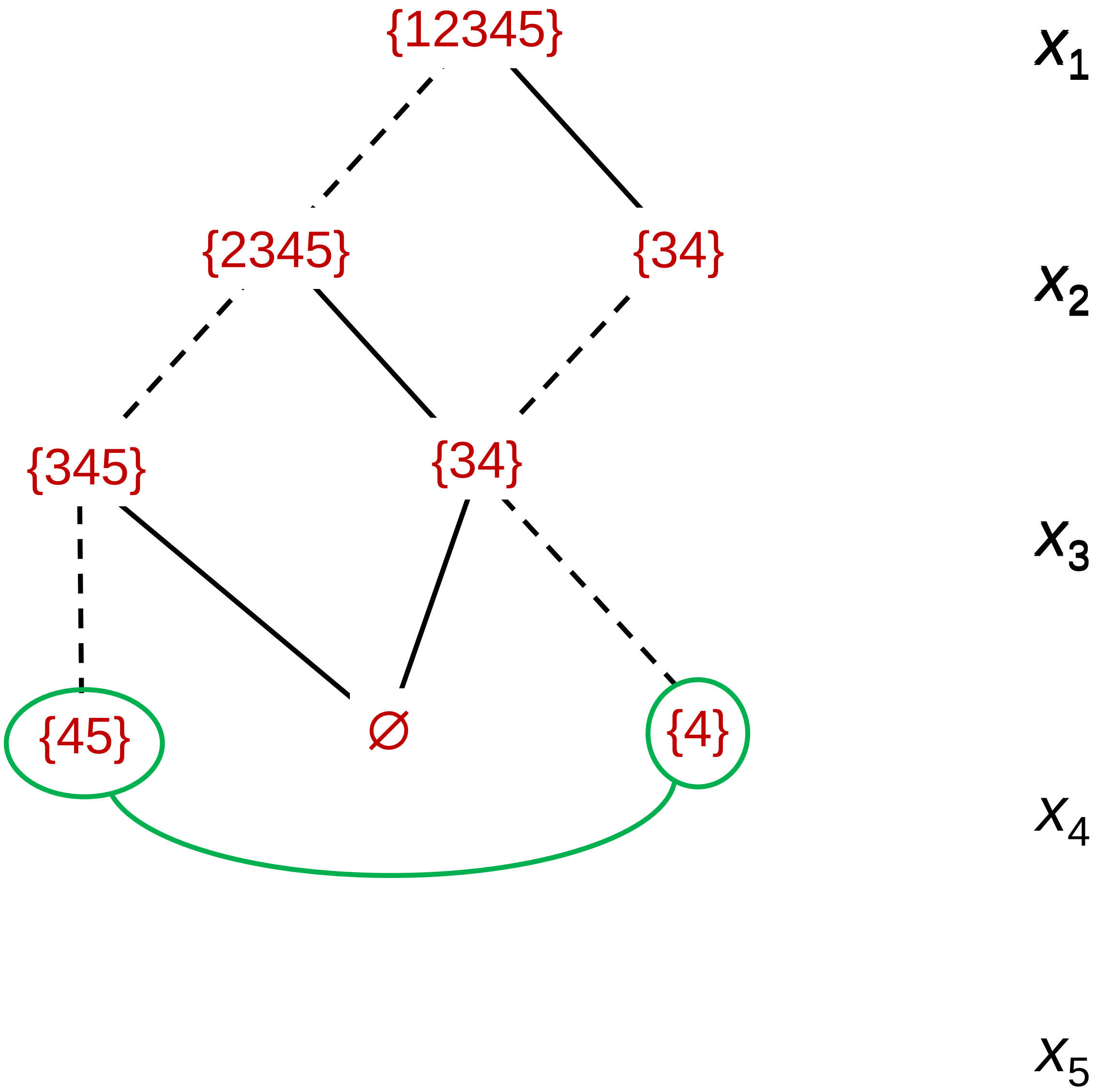




To build **relaxed** DD, merge some additional nodes as we go along.

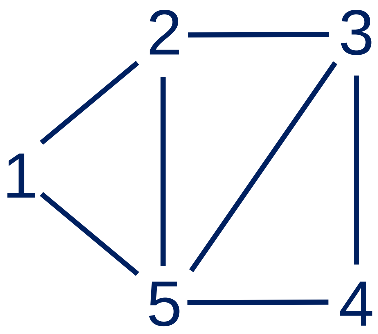
Take the **union** of merged states.

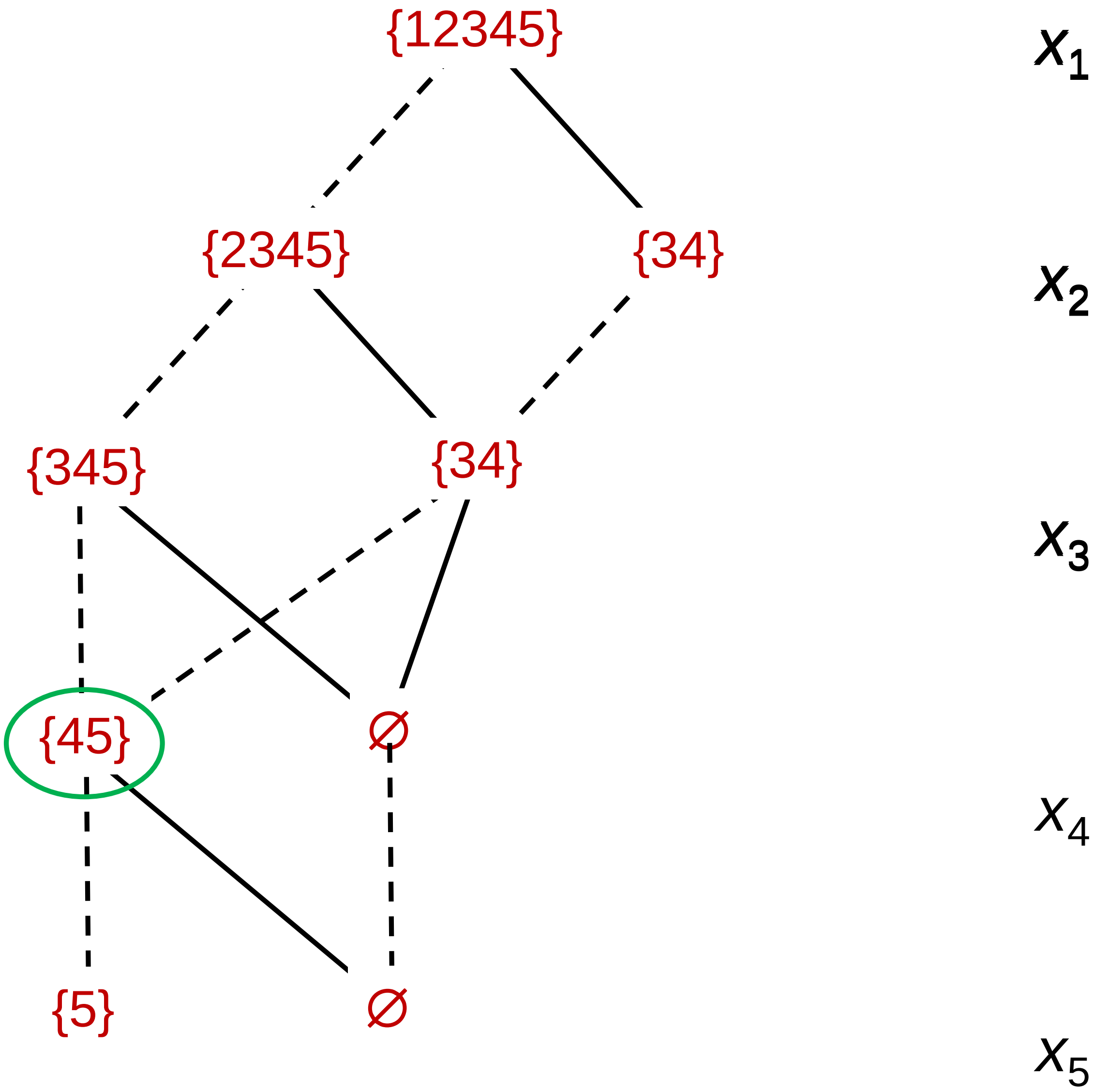




To build **relaxed** DD, merge some additional nodes as we go along.

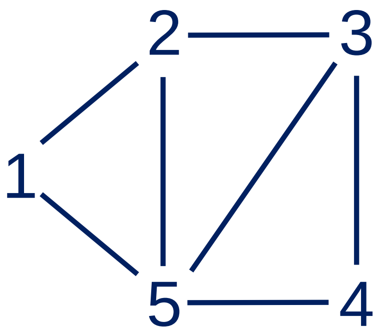
Take the **union** of merged states.





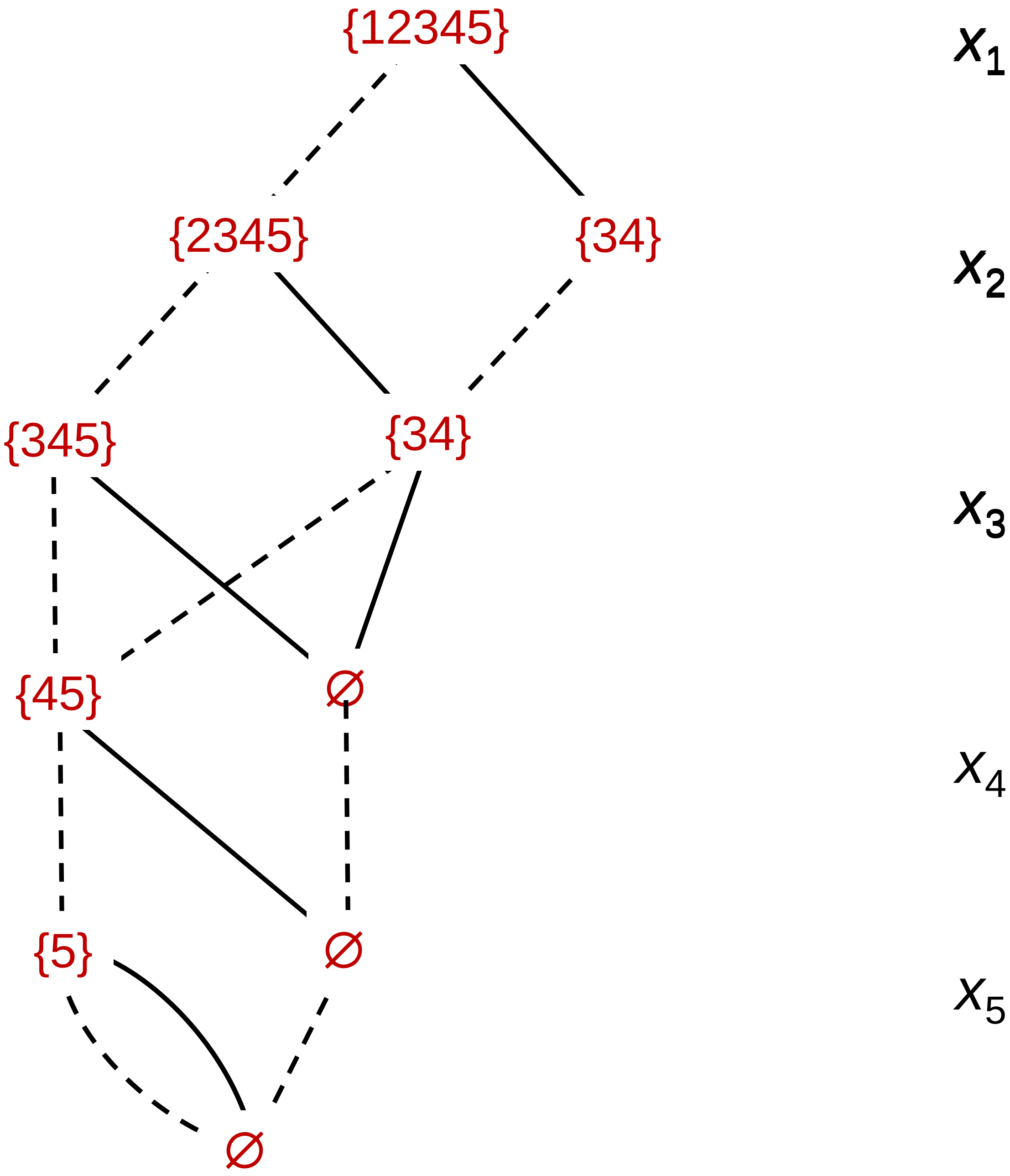
To build **relaxed** DD, merge some additional nodes as we go along.

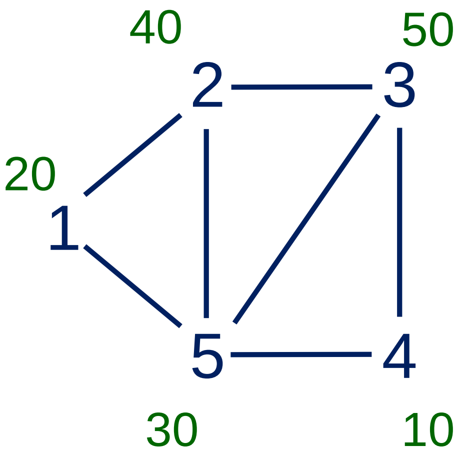
Take the **union** of merged states.



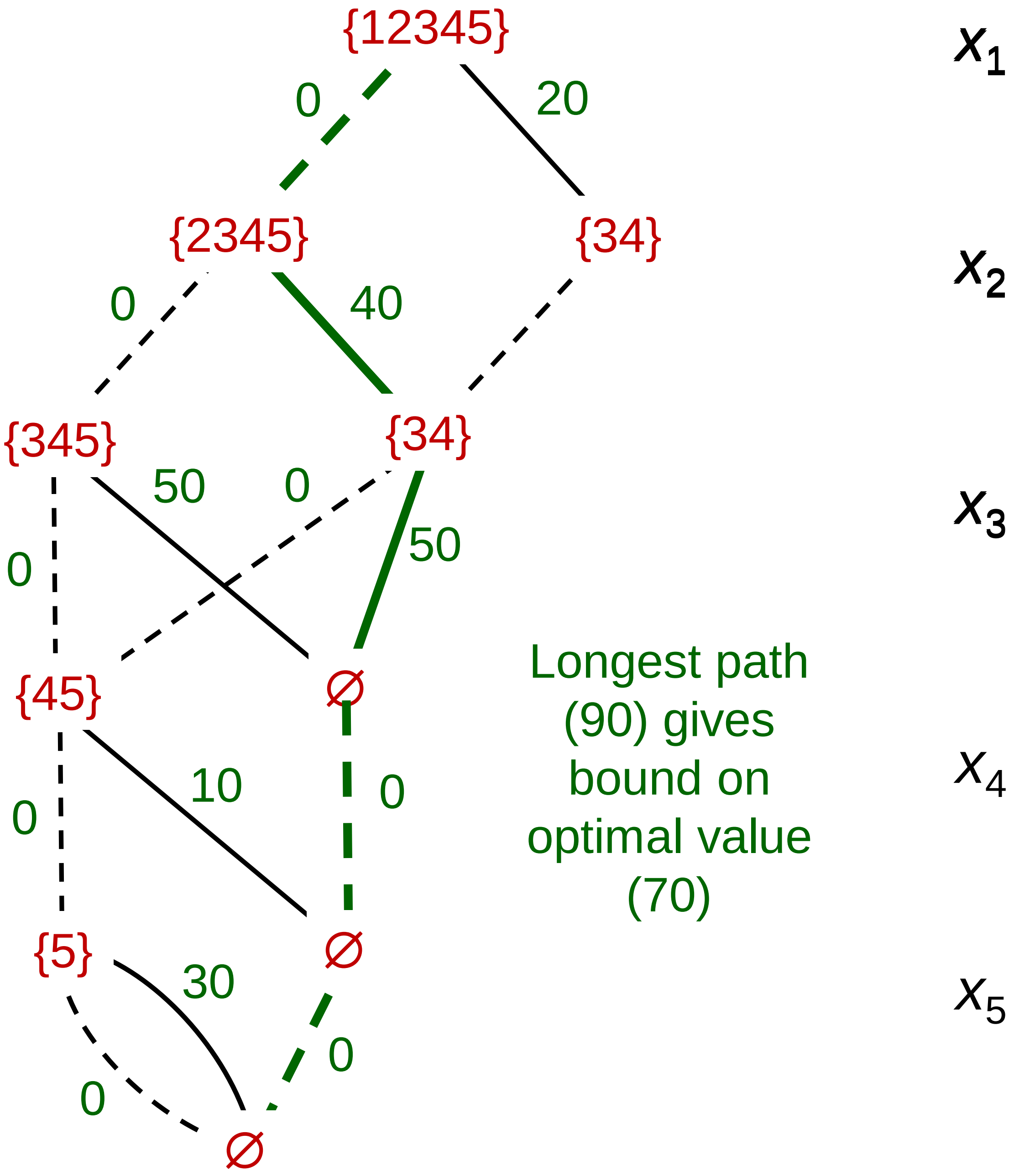
Width = 2

Represents 11 solutions, including 9 feasible solutions





Width = 2



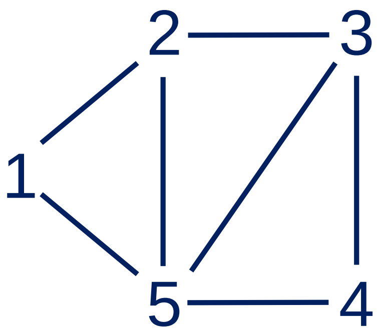
Represents
11 solutions,
including
9 feasible
solutions

Longest path
(90) gives
bound on
optimal value
(70)

Relaxation by Node Splitting

- Alternate relaxation method: **node refinement** during top-down compilation
  - Start with DD of width 1 representing Cartesian product of variable domains.
  - Split nodes so as to remove some infeasible paths.

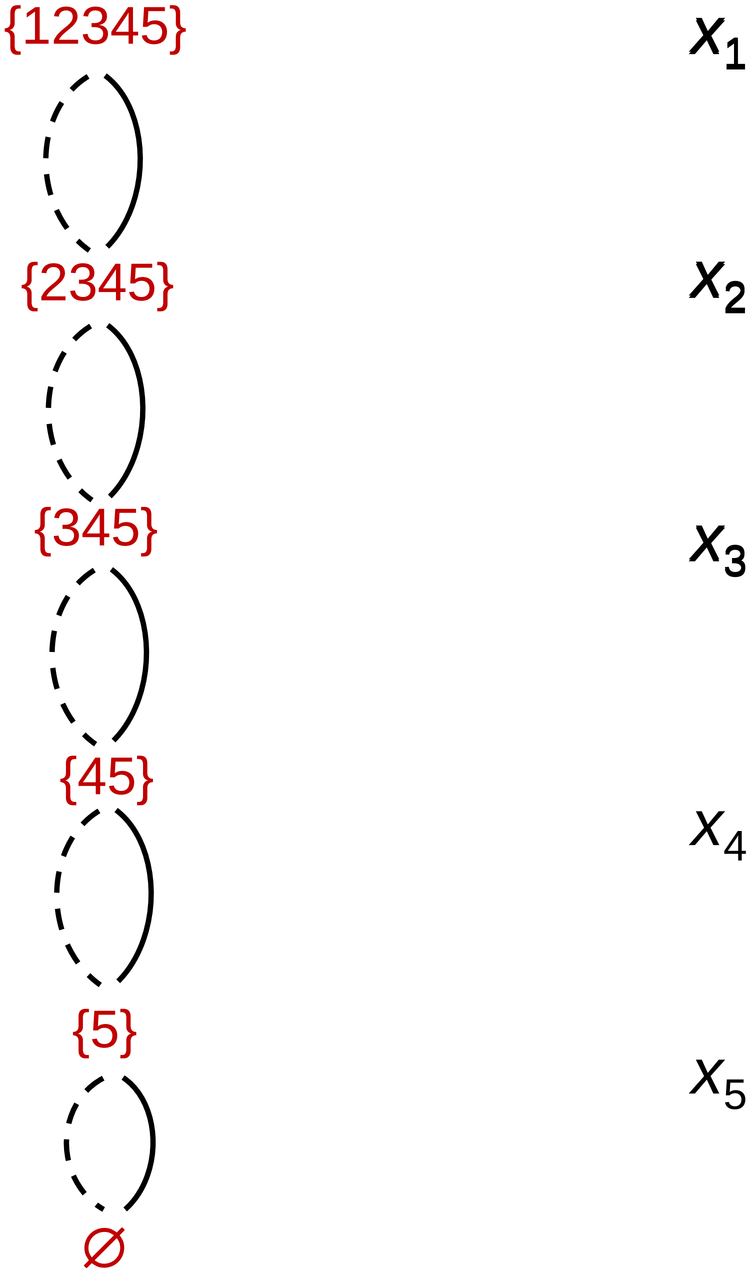
Andersen, Hadžić, JH, Tiedemann (2007)

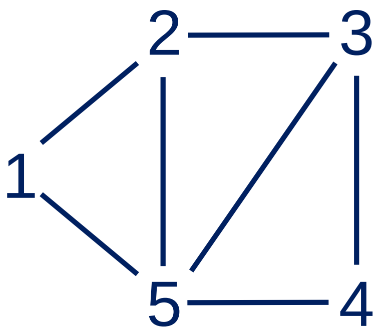


Aim for
width = 2

Start with DD of
width 1

32 solutions,
9 of which are
feasible

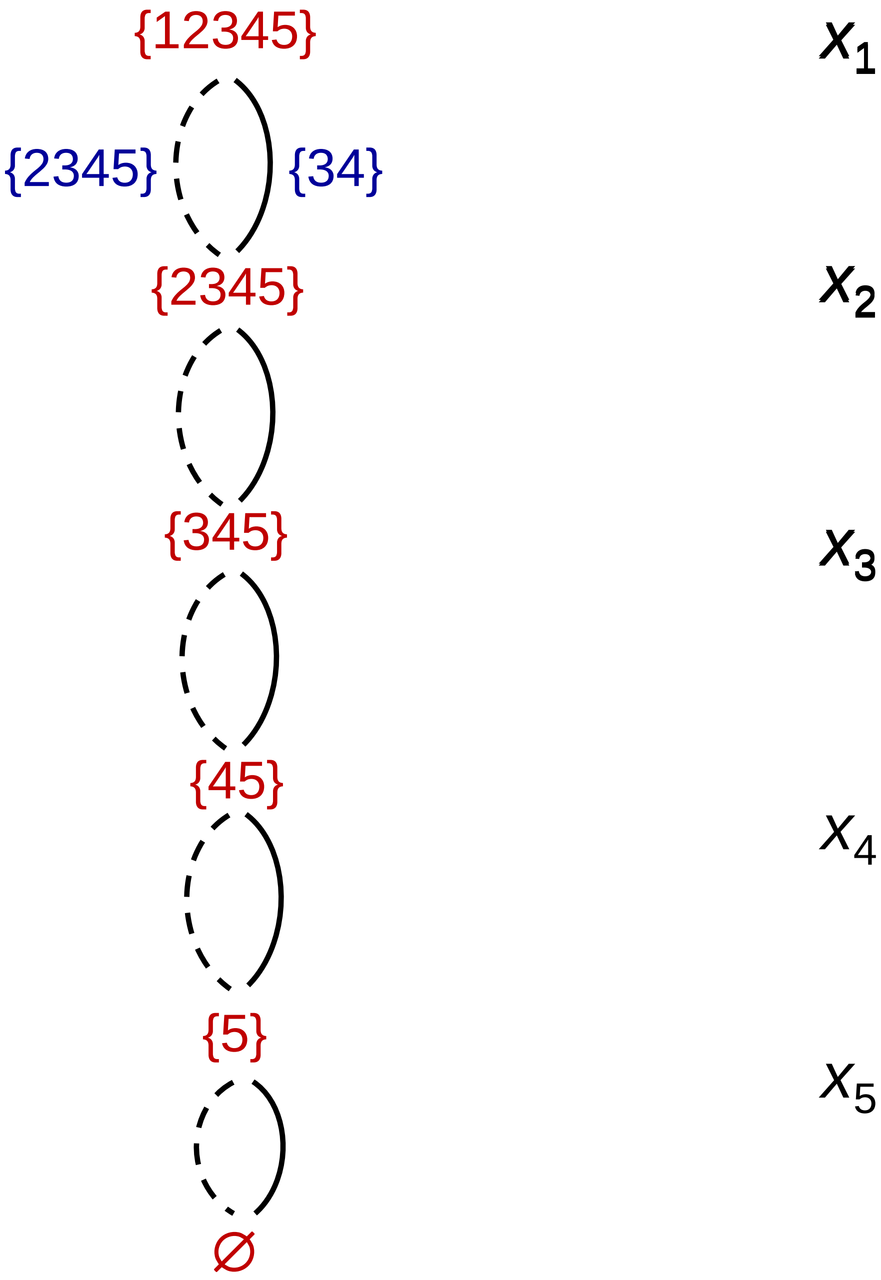


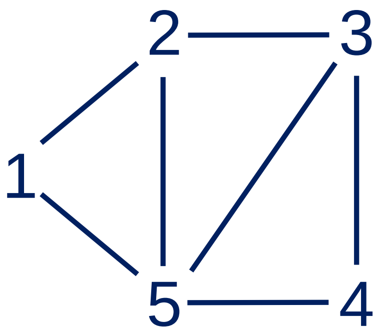


Aim for width = 2

Start with DD of **width 1**

Examine states that result from arcs leaving top node.

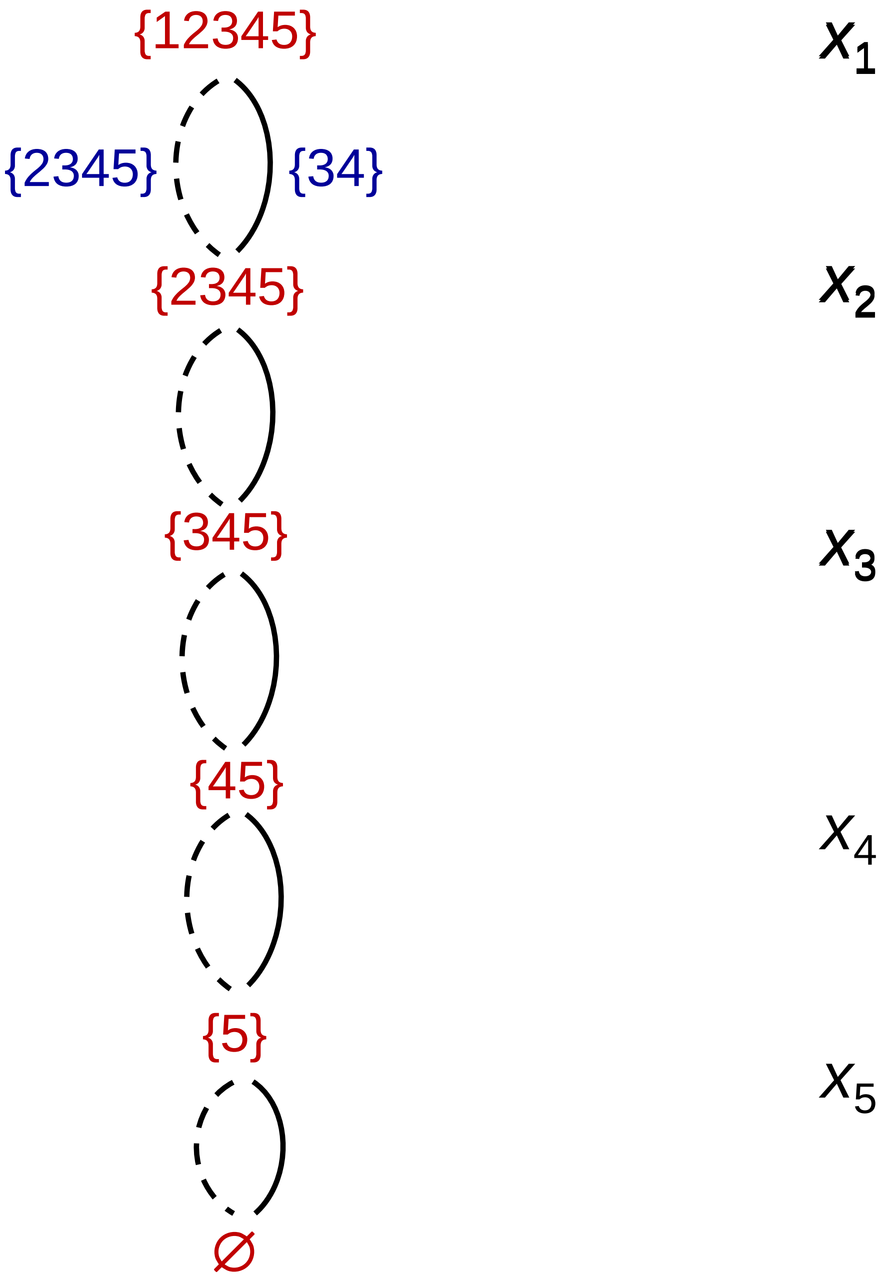


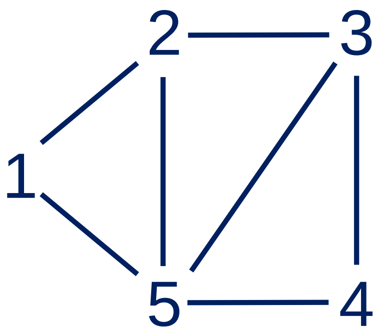


Aim for
width = 2

Start with DD of
width 1

Can split states
if they are
different
(they are).

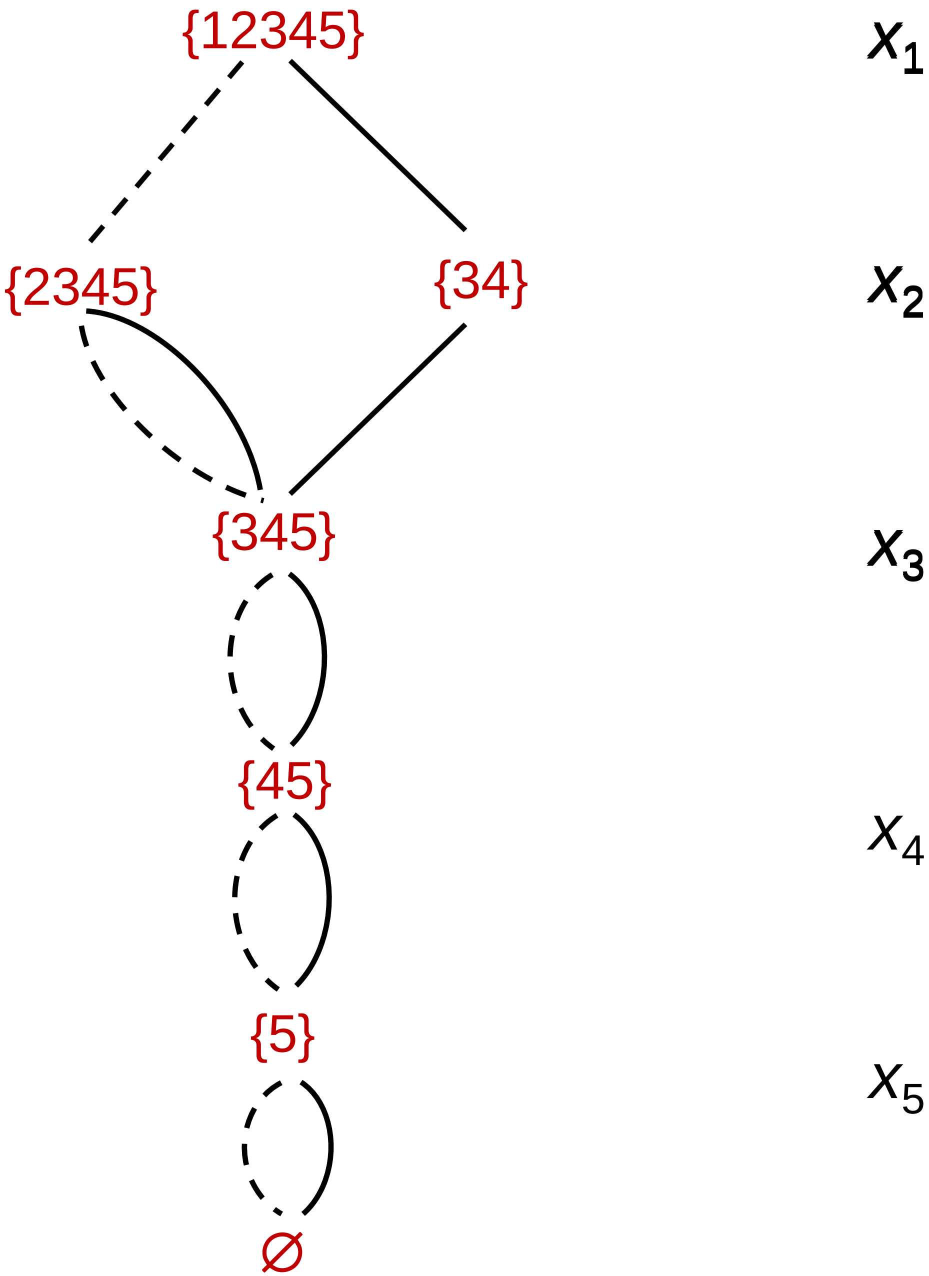


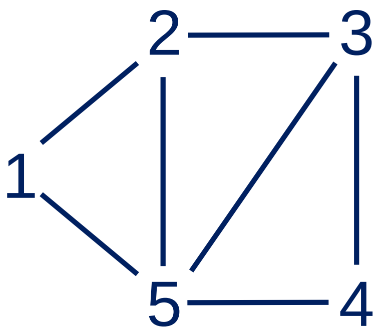


Aim for
width = 2

Start with DD of
width 1

Can split states
if they are
different
(they are).

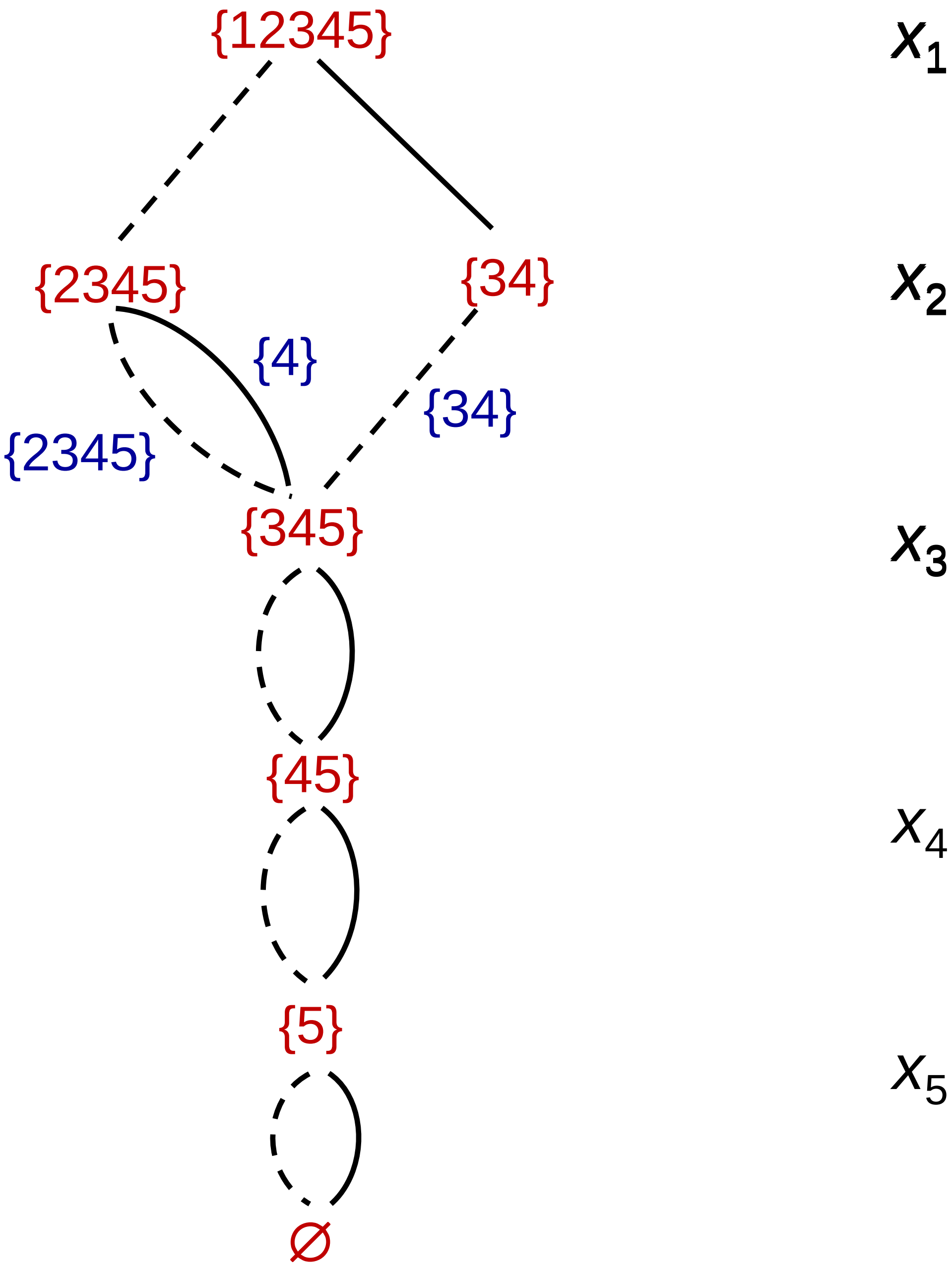


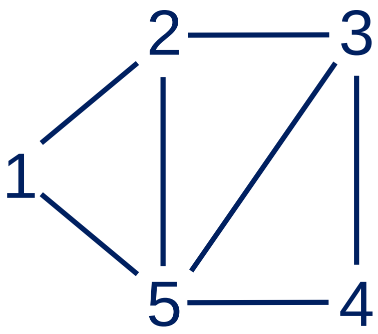


Aim for
width = 2

Start with DD of
width 1

Examine states
in next layer.



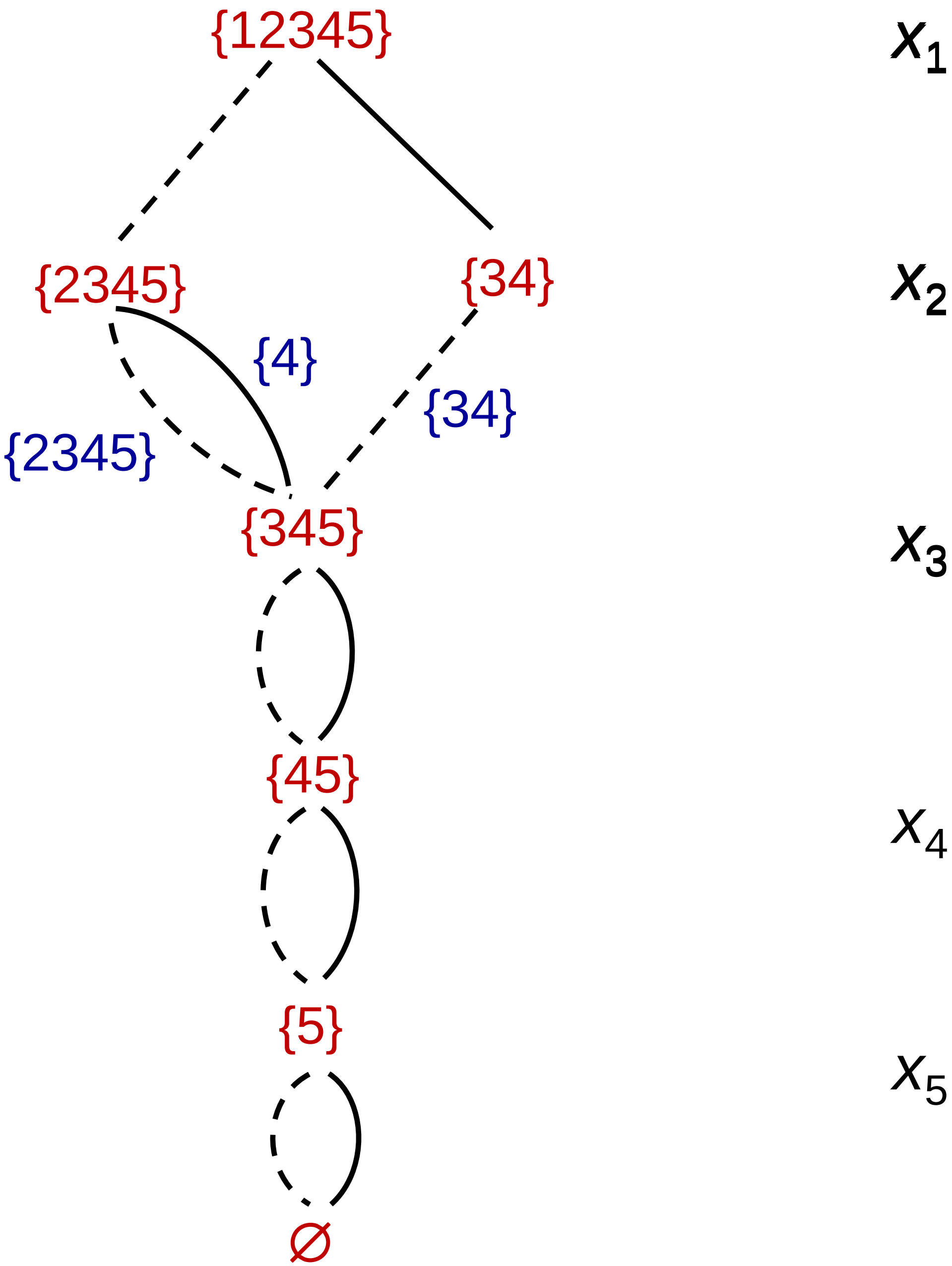


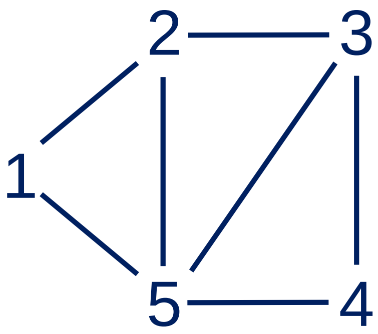
Aim for
width = 2

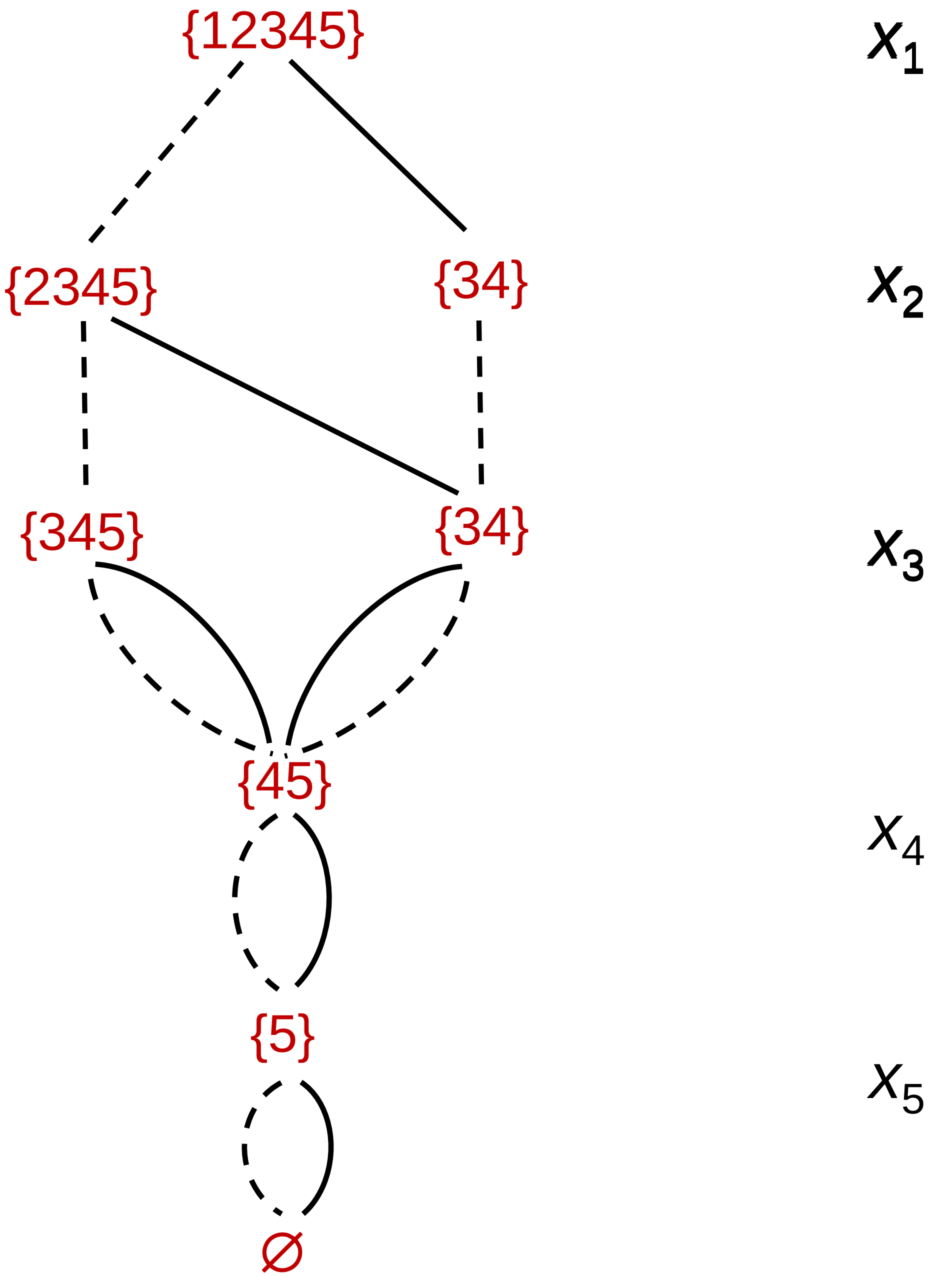
Start with DD of
width 1

Examine states
in next layer.

All distinct, split
arbitrarily.





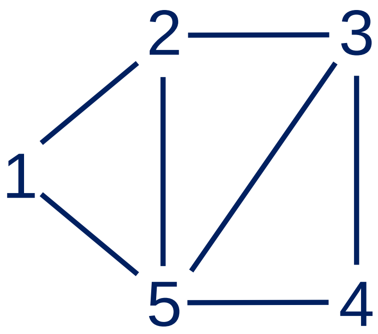


Aim for width = 2

Start with DD of **width 1**

Examine states in next layer.

All distinct, split arbitrarily.

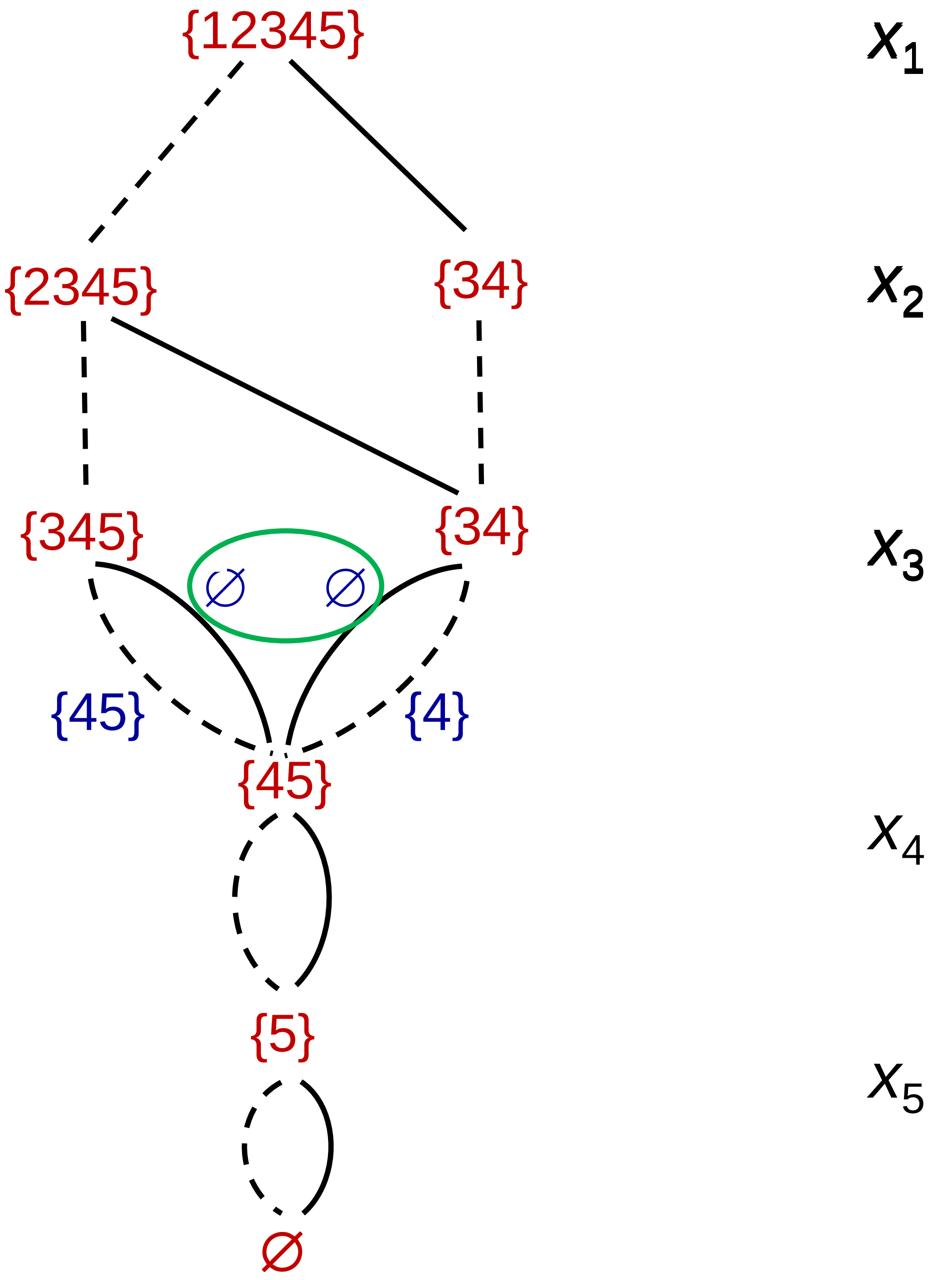


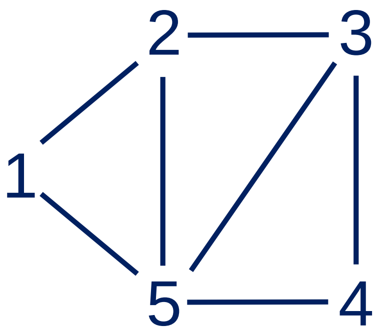
Aim for width = 2

Start with DD of **width 1**

Repeat.

Two states are identical and are not split.



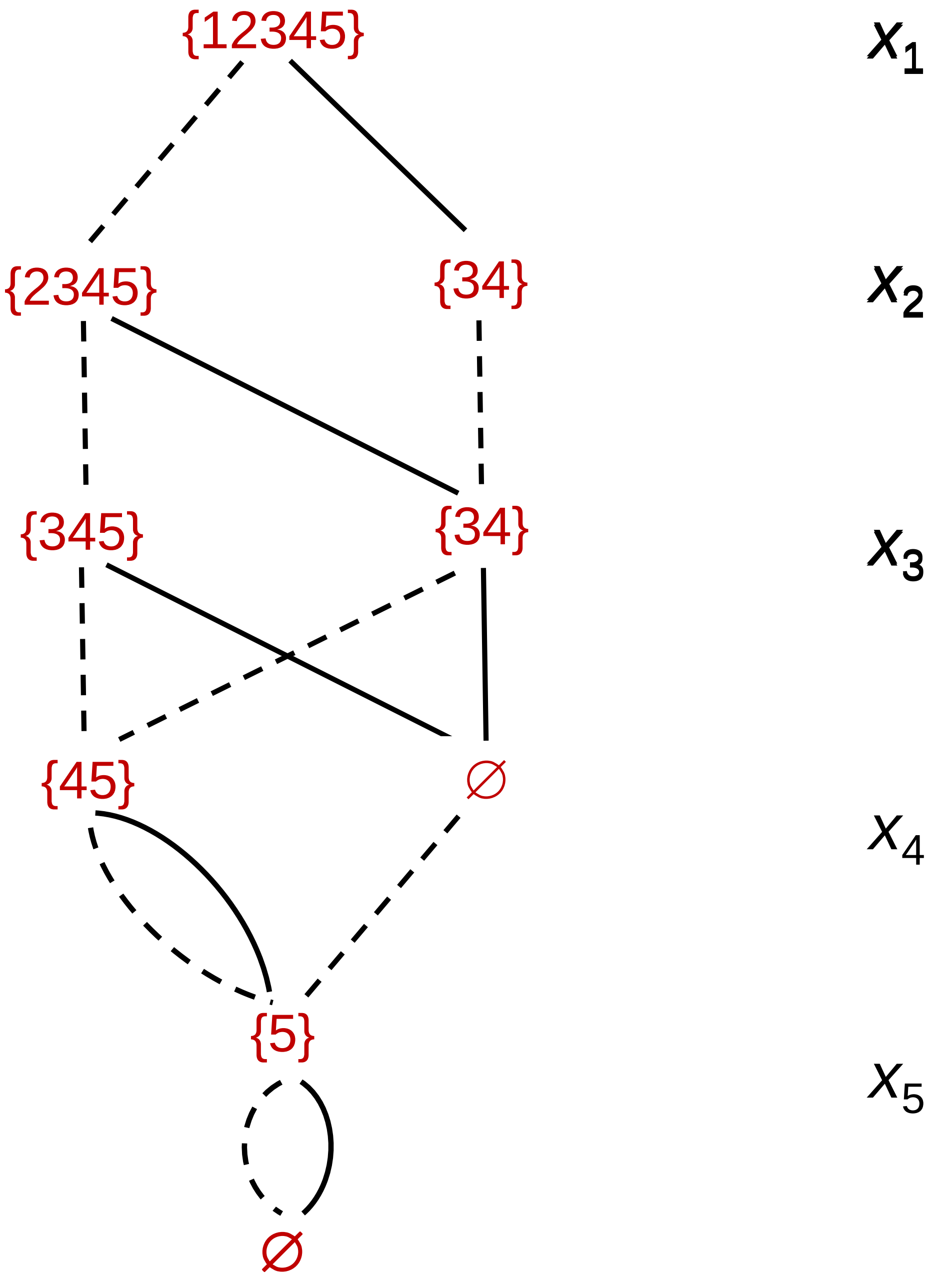


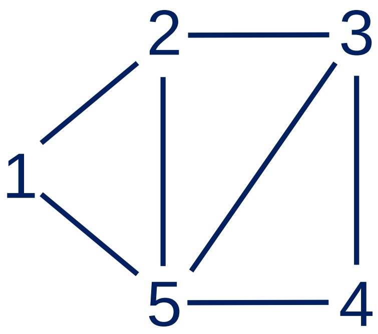
Aim for width = 2

Start with DD of **width 1**

Repeat.

Two states are identical and are not split.



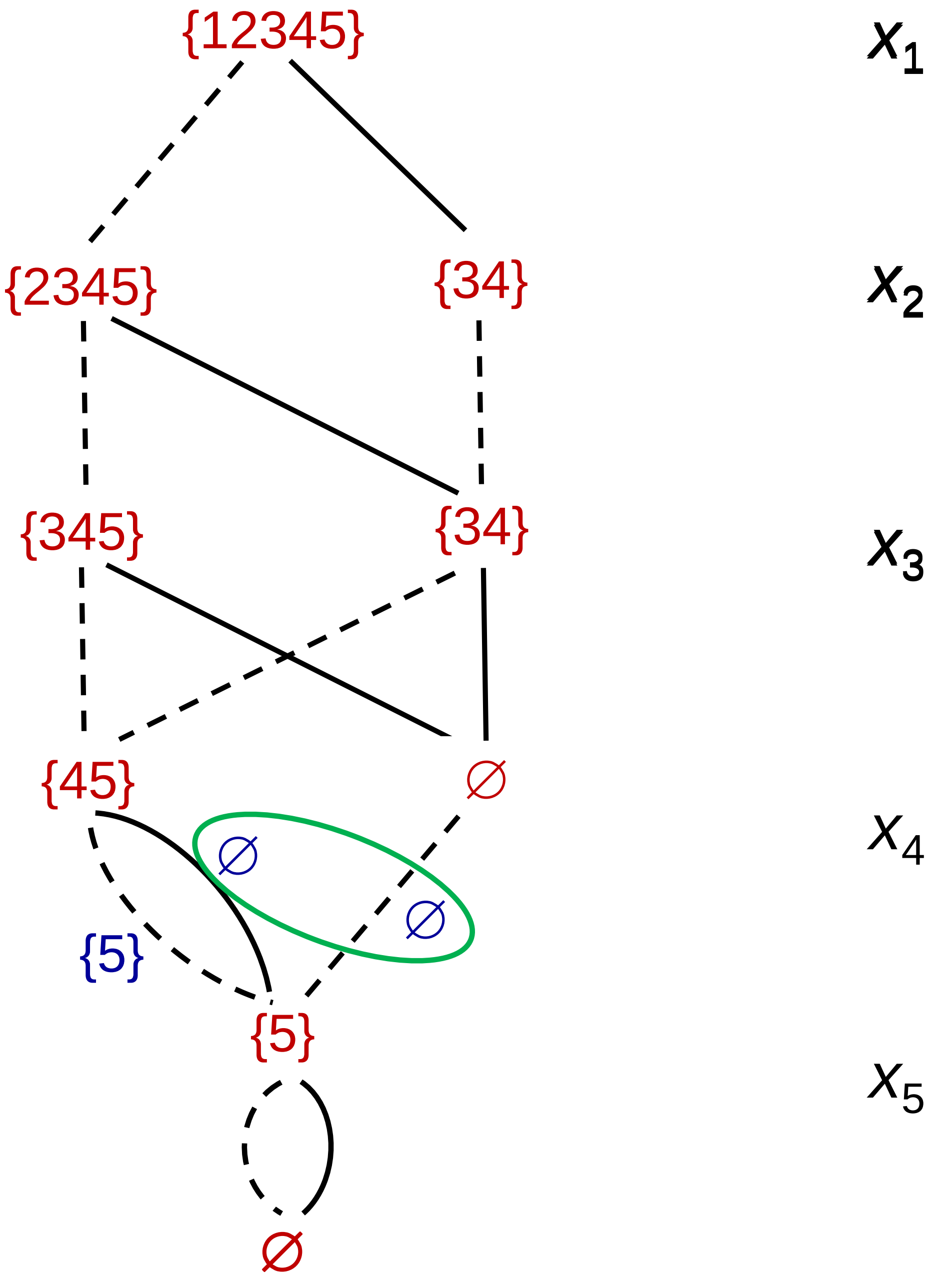


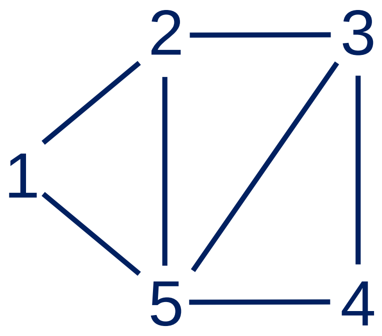
Aim for width = 2

Start with DD of **width 1**

Repeat.

Two states are identical and are not split.



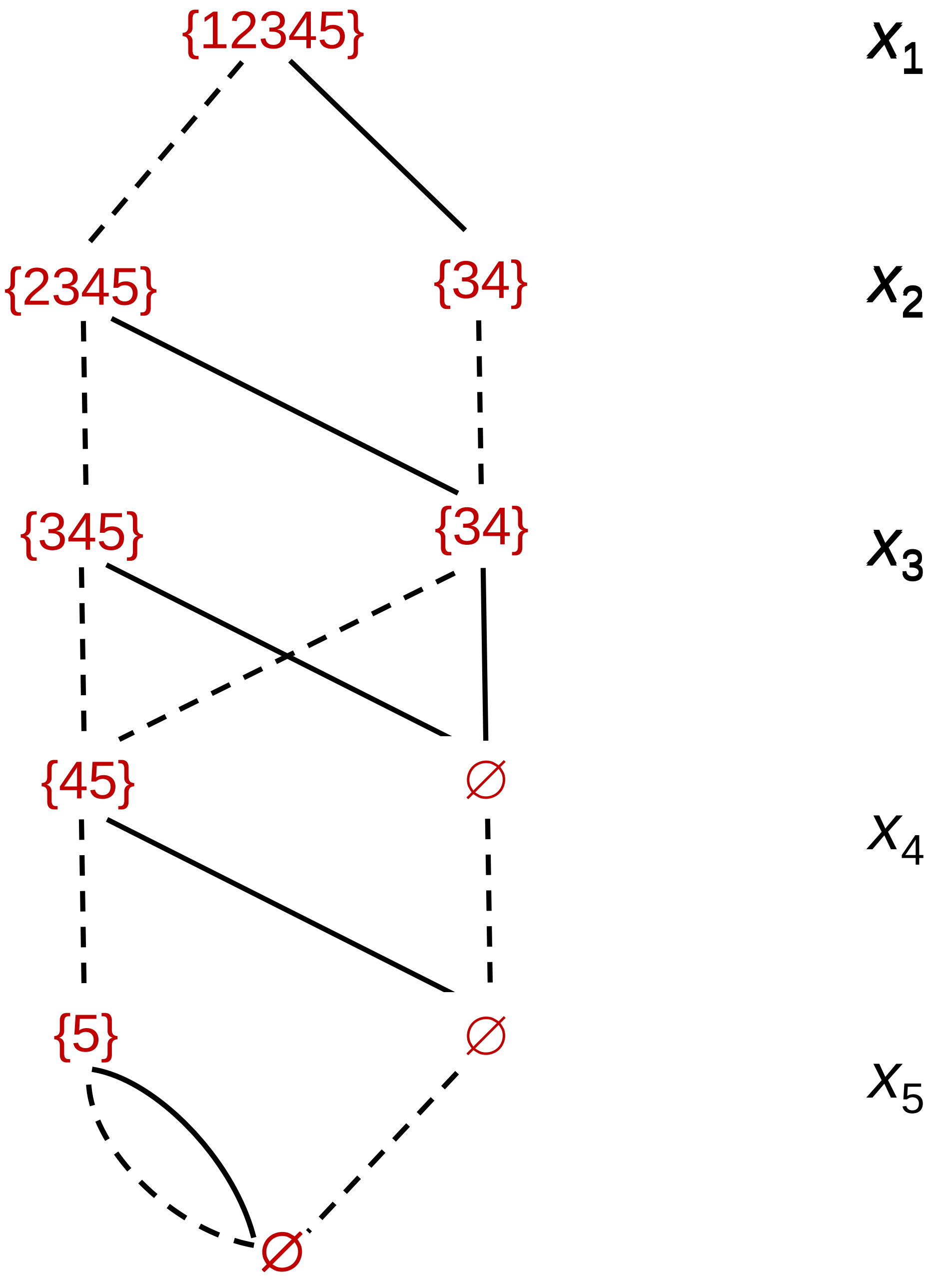


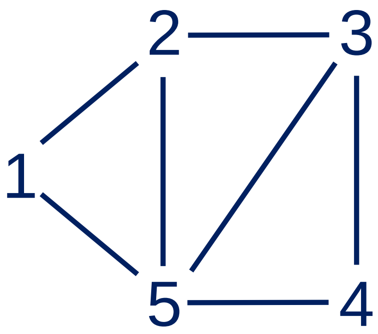
Aim for width = 2

Start with DD of **width 1**

Repeat.

Two states are identical and are not split.

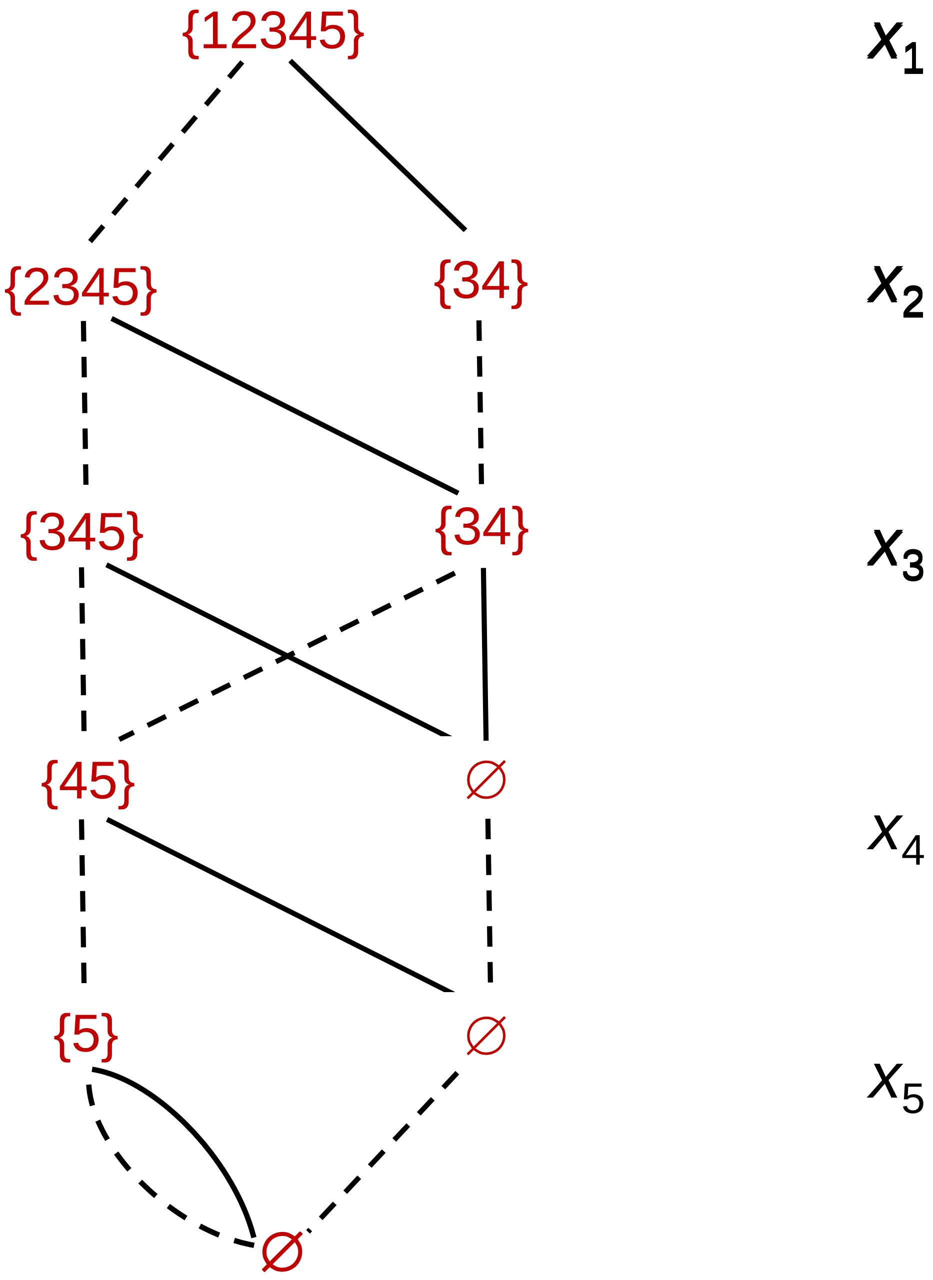




Aim for width = 2

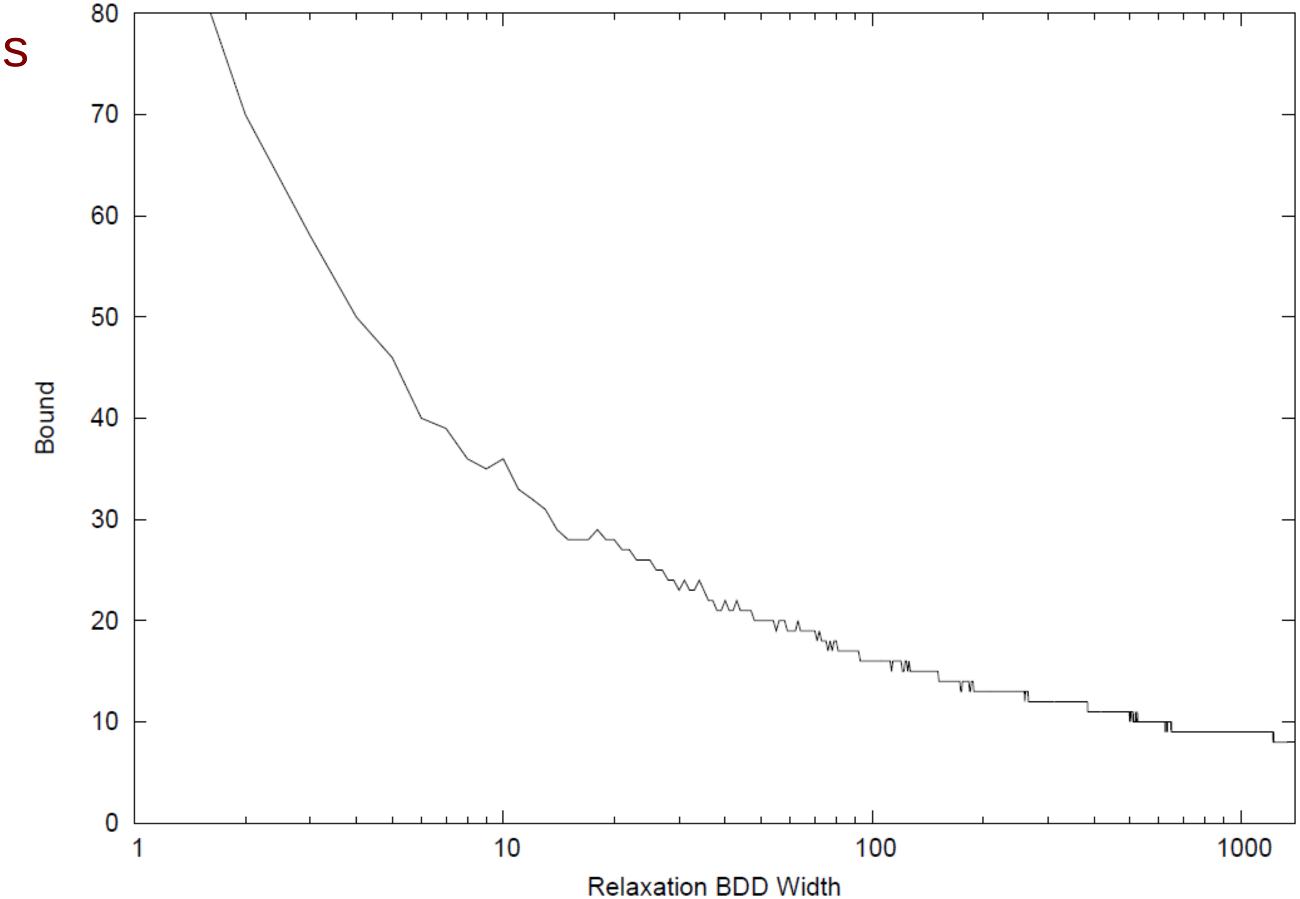
Start with DD of **width 1**

12 solutions, 9 of which are feasible



Relaxed Decision Diagrams

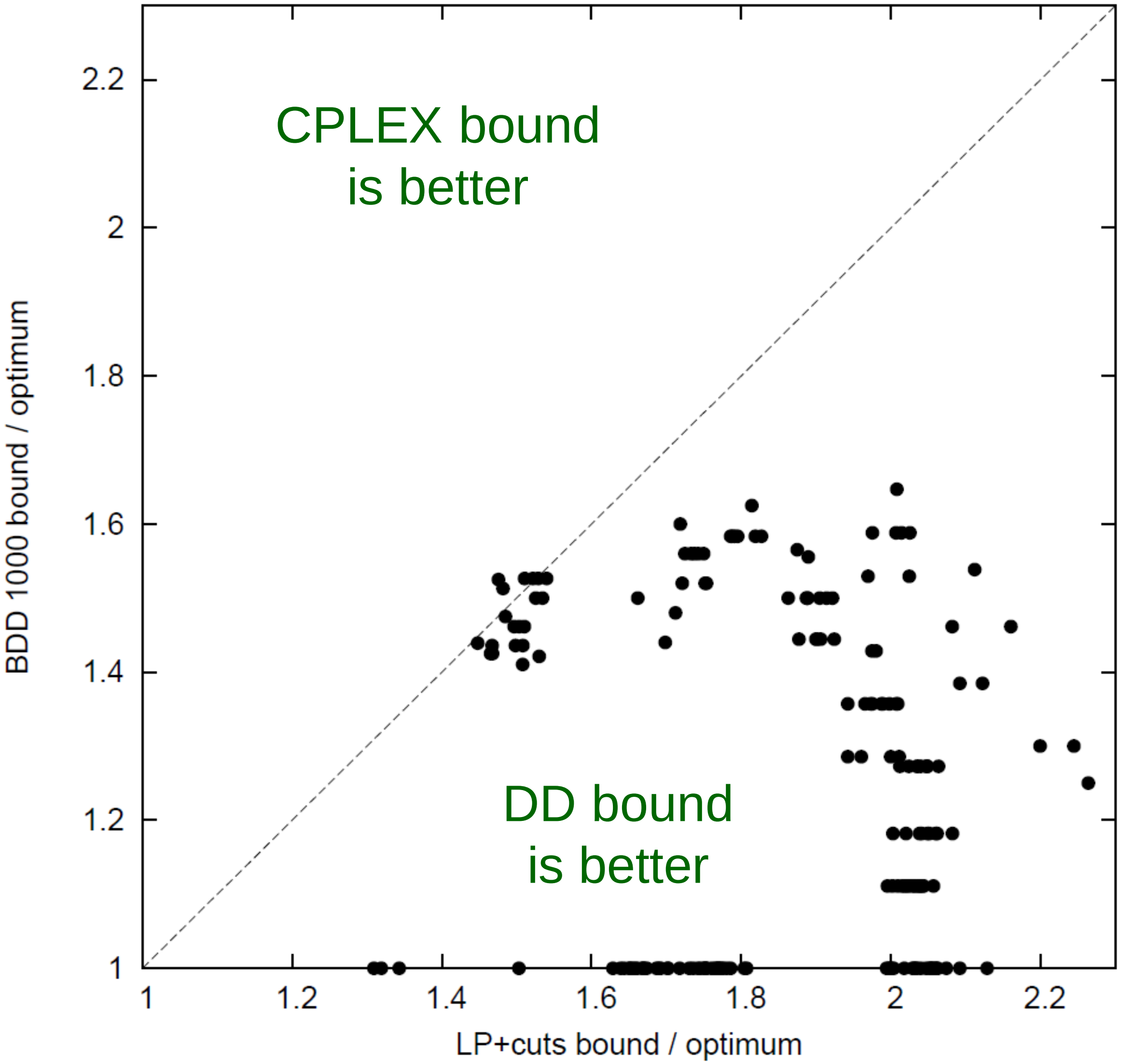
- Wider diagrams yield tighter bounds
  - But take longer to build.
  - Adjust width dynamically.



Bergman, Ciré, van Hoeve, JH (2013)

Relaxed Decision Diagrams

- DDs vs. CPLEX bound at root node for max stable set problem
  - Using CPLEX default cut generation
  - DD max width of 1000.
  - DDs require about 5% the time of CPLEX



Bergman, Ciré, van Hoeve, JH (2013)

Propagation in Relaxed DDs

- Propagate through **relaxed DD** rather than **domain store**.
  - DD conveys more information.
- This was first application of relaxed DDs.
  - Applied to multiple alldiffs (graph coloring).

Andersen, Hadžić, JH, Tiedemann (2007)

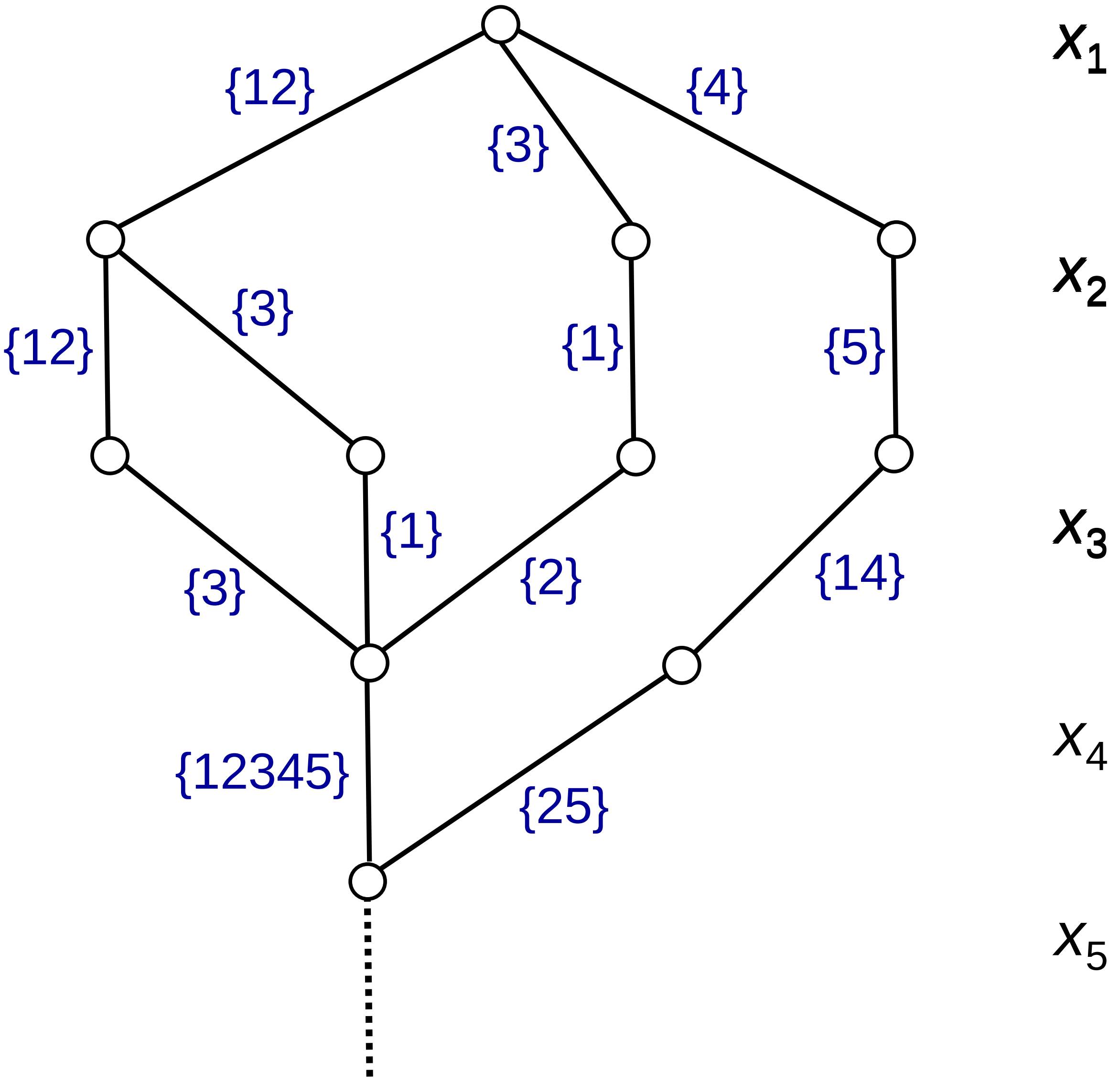
Propagation in Relaxed DDs

- Example 1: multiple alldiffs
  - Propagate `alldiff`(x_1, …, x_4)
  - Through a given DD relaxation.
- Example 2: single-machine scheduling with time windows.
  - Propagate alldiff + time windows.

Suppose this is a relaxed DD for the problem.

Indicate multiple arcs with arc domains

Propagate `alldiff`(x_1,...,x_4)



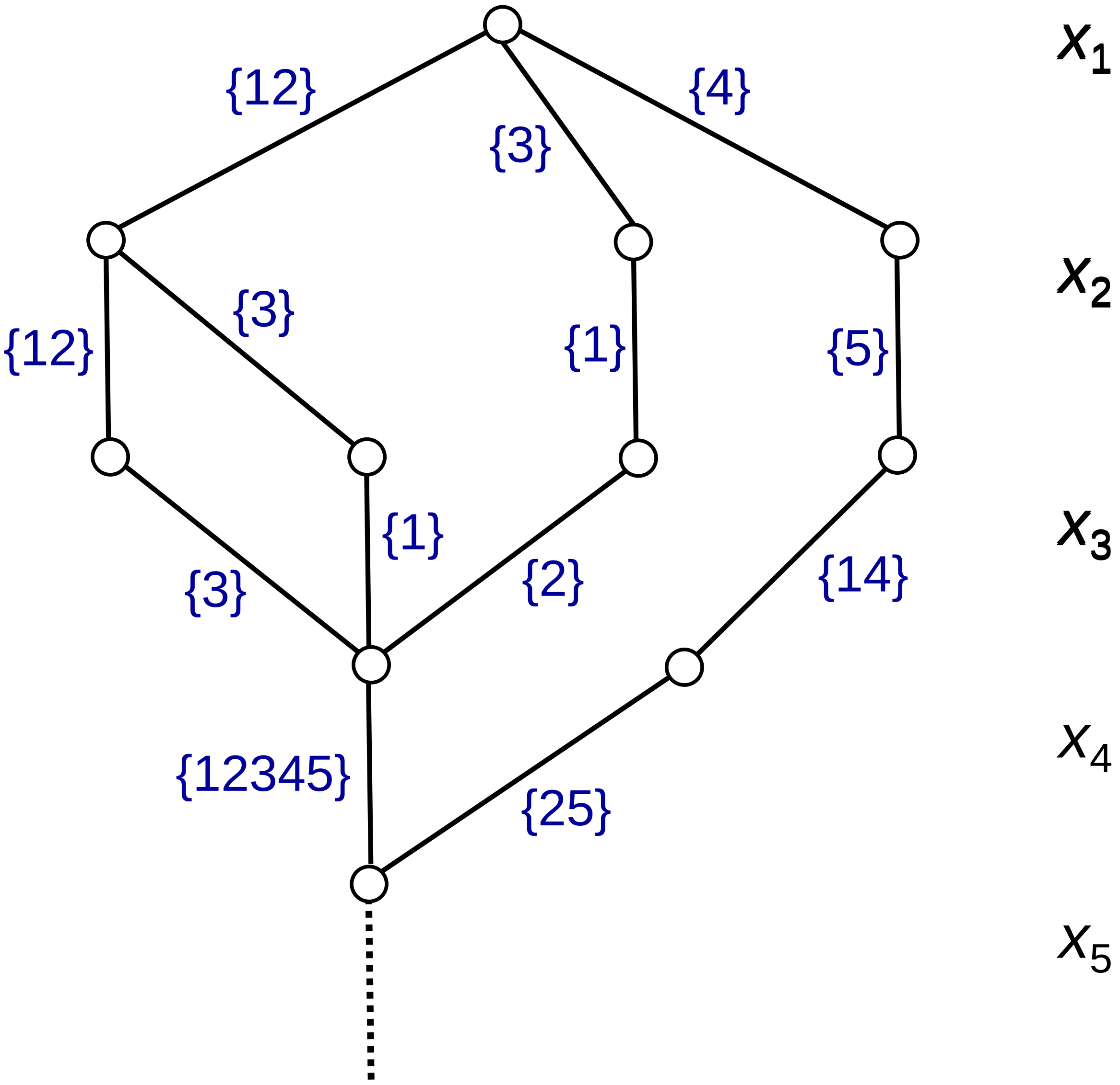
`alldiff` provides no filtering for domain store

{1234}

{12345}

{1234}

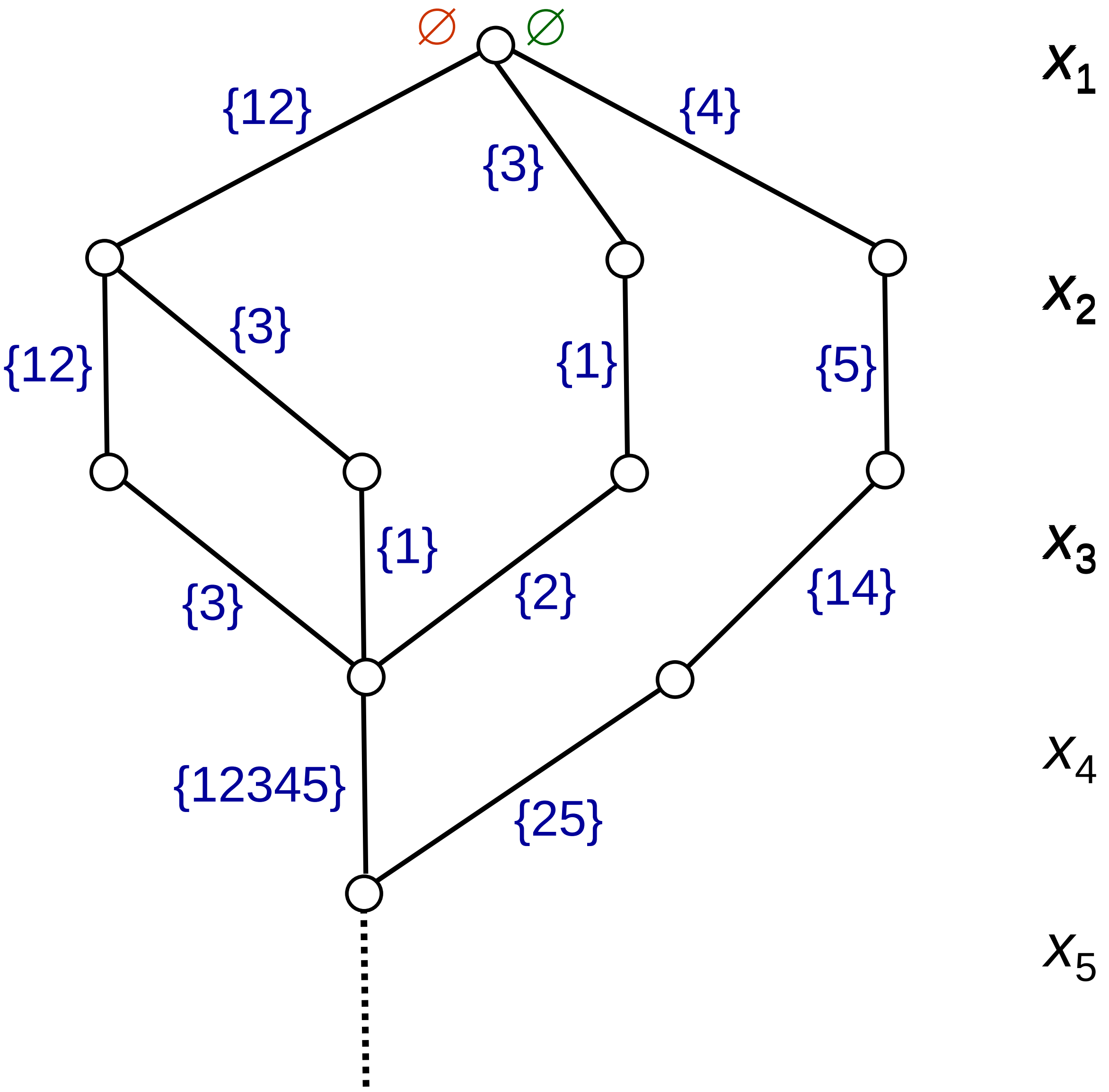
{12345}



For purposes of filtering alldiff, introduce state (*A*,*S*)

A = {jobs on **all** paths to node}

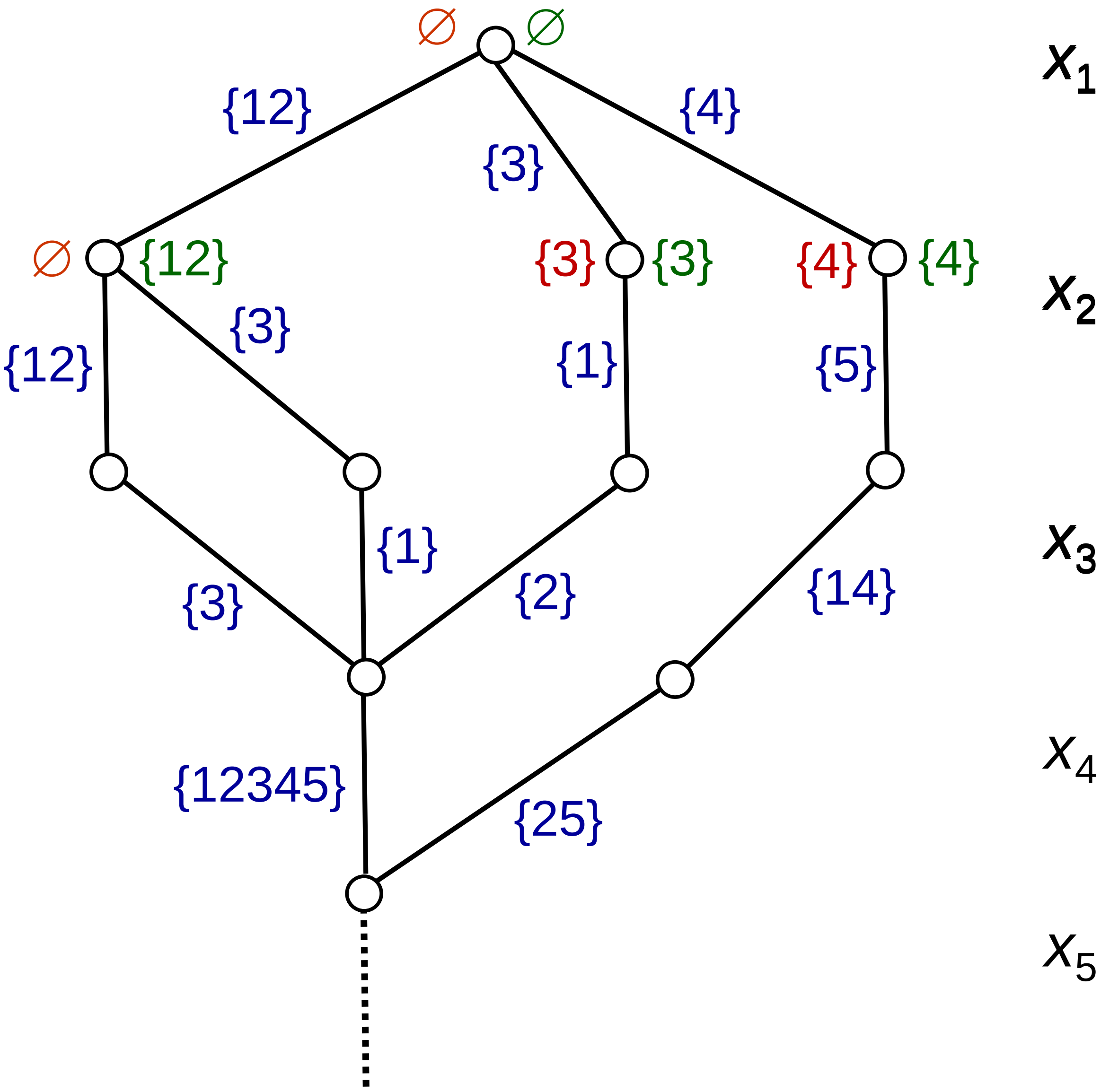
S = {jobs on **some** path to node}



For purposes of filtering alldiff, introduce state (*A*,*S*)

A = {jobs on **all** paths to node}

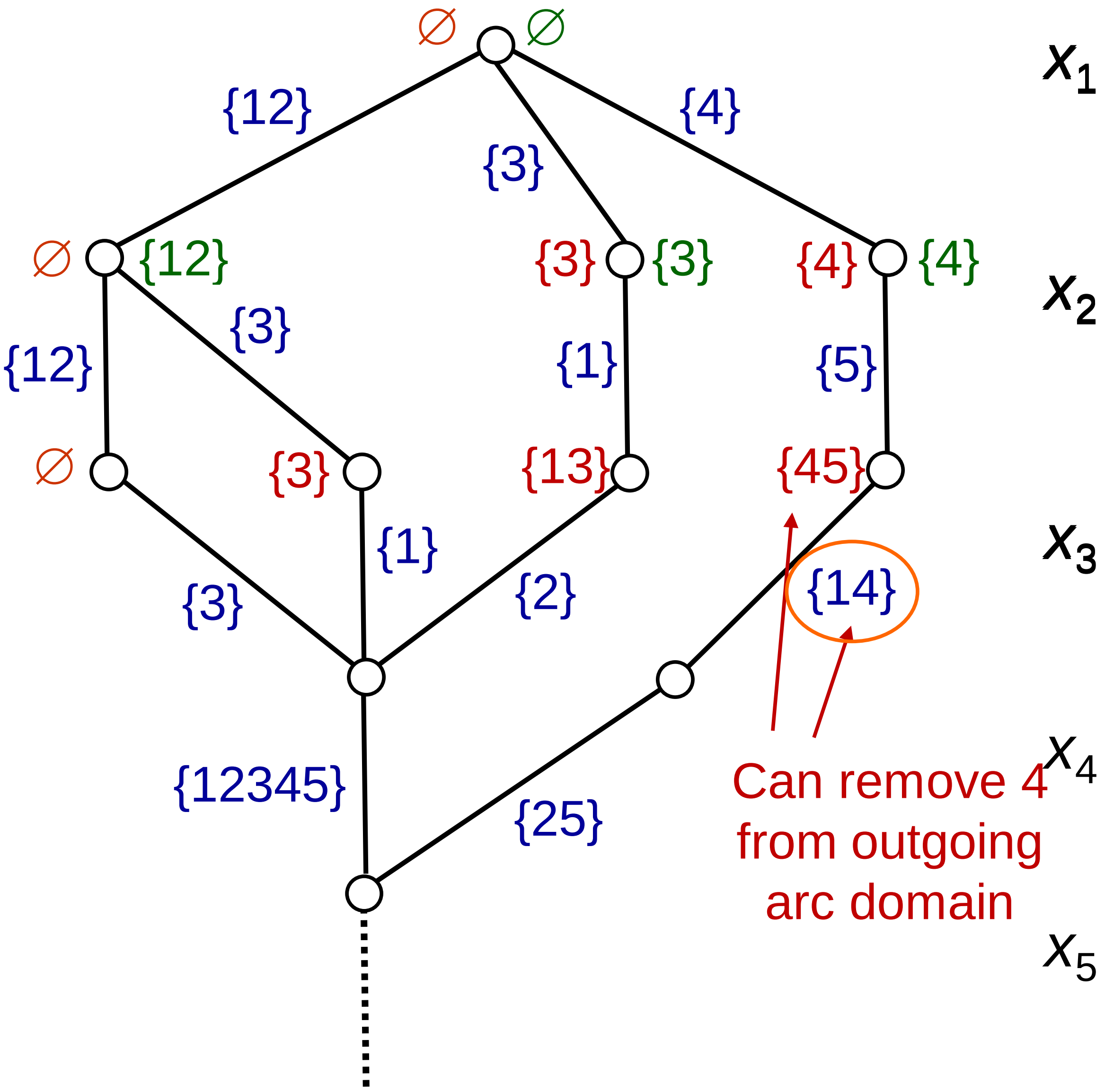
S = {jobs on **some** path to node}



For purposes of filtering alldiff, introduce state (A,S)

A = {jobs on **all** paths to node}

S = {jobs on **some** path to node}

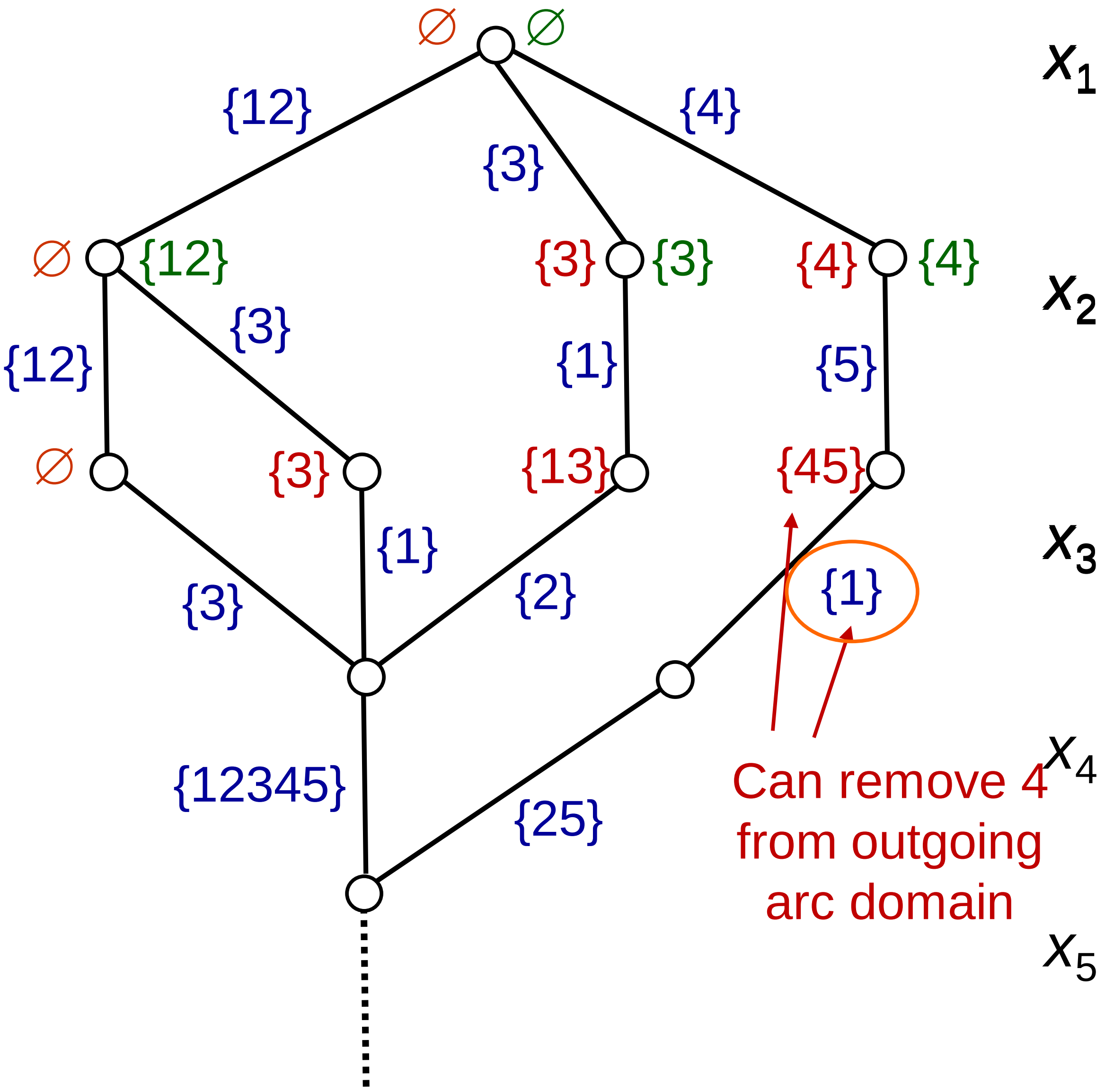


Can remove 4 from outgoing arc domain

For purposes of filtering alldiff, introduce state (*A*,*S*)

A = {jobs on **all** paths to node}

S = {jobs on **some** path to node}

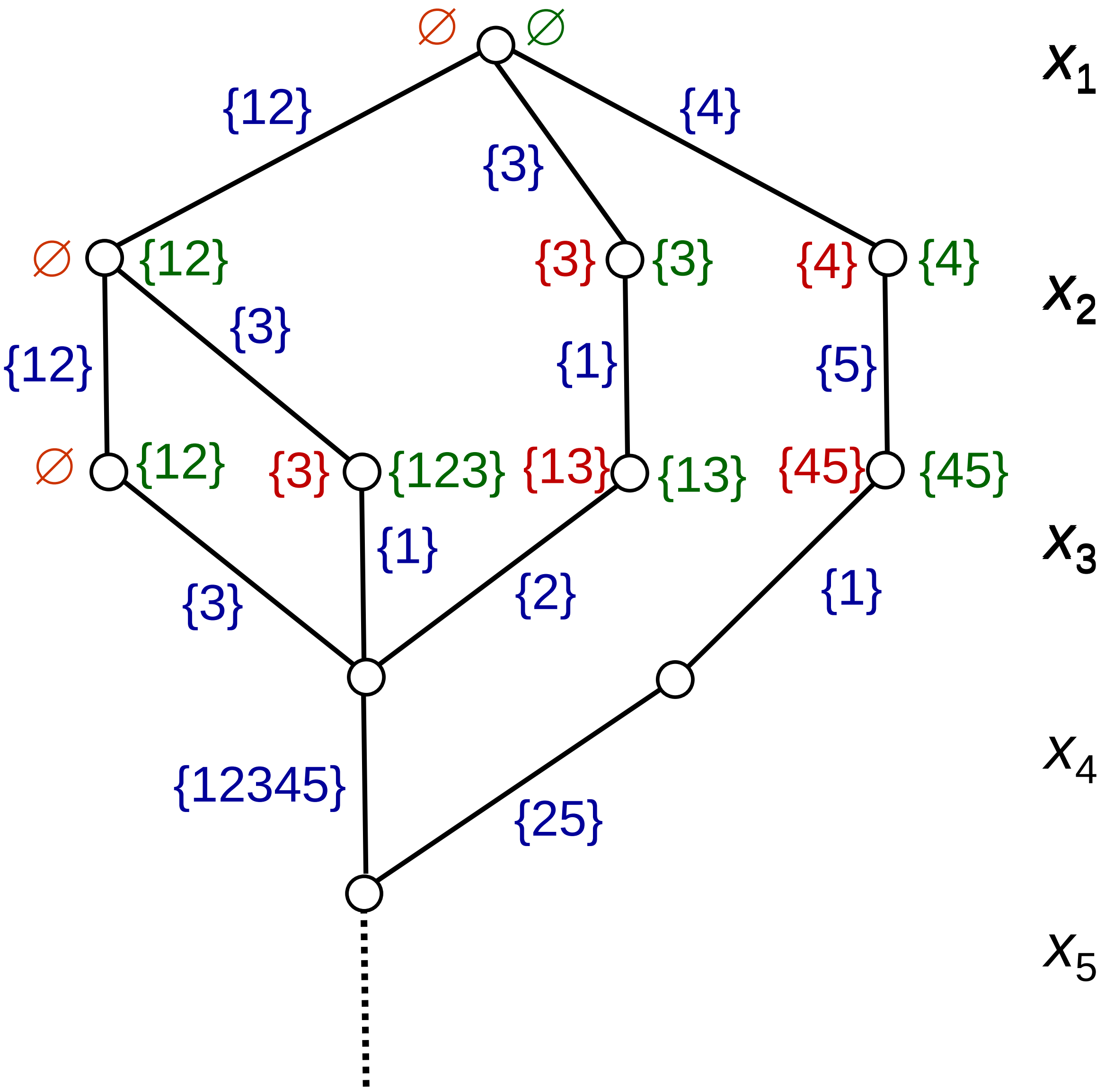


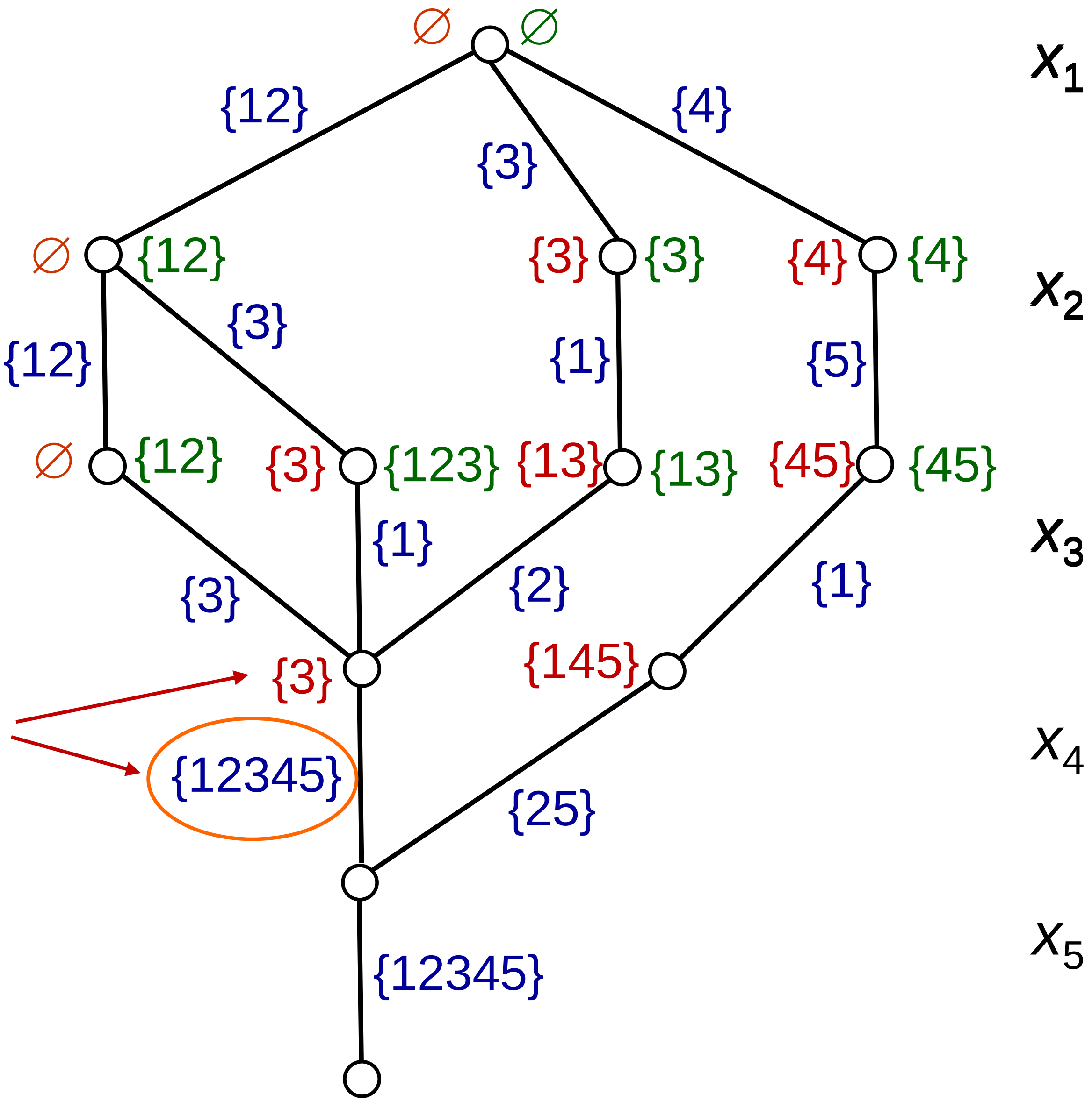
Can remove 4 from outgoing arc domain

For purposes of filtering alldiff, introduce state (A,S)

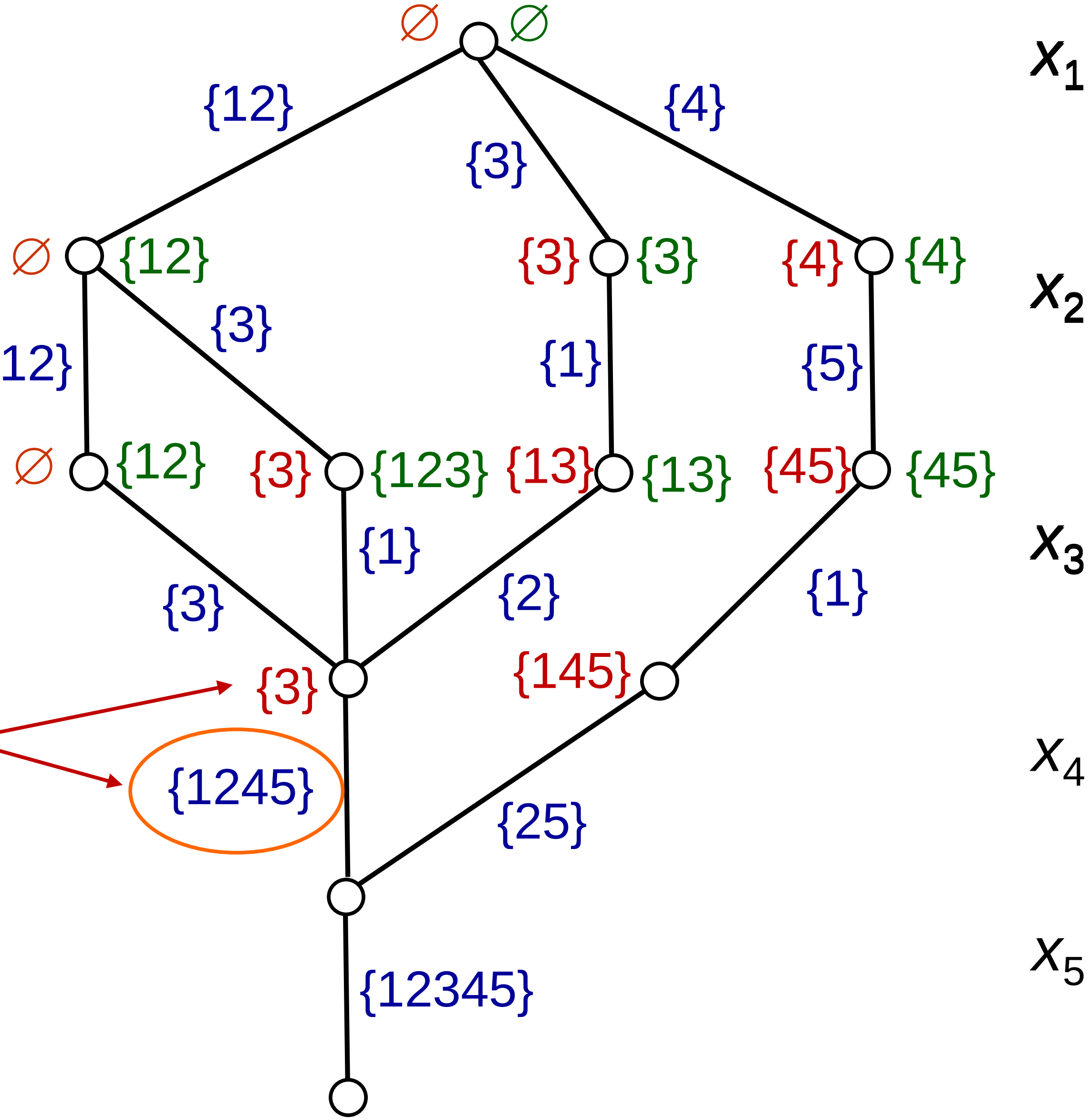
A = {jobs on **all** paths to node}

S = {jobs on **some** path to node}

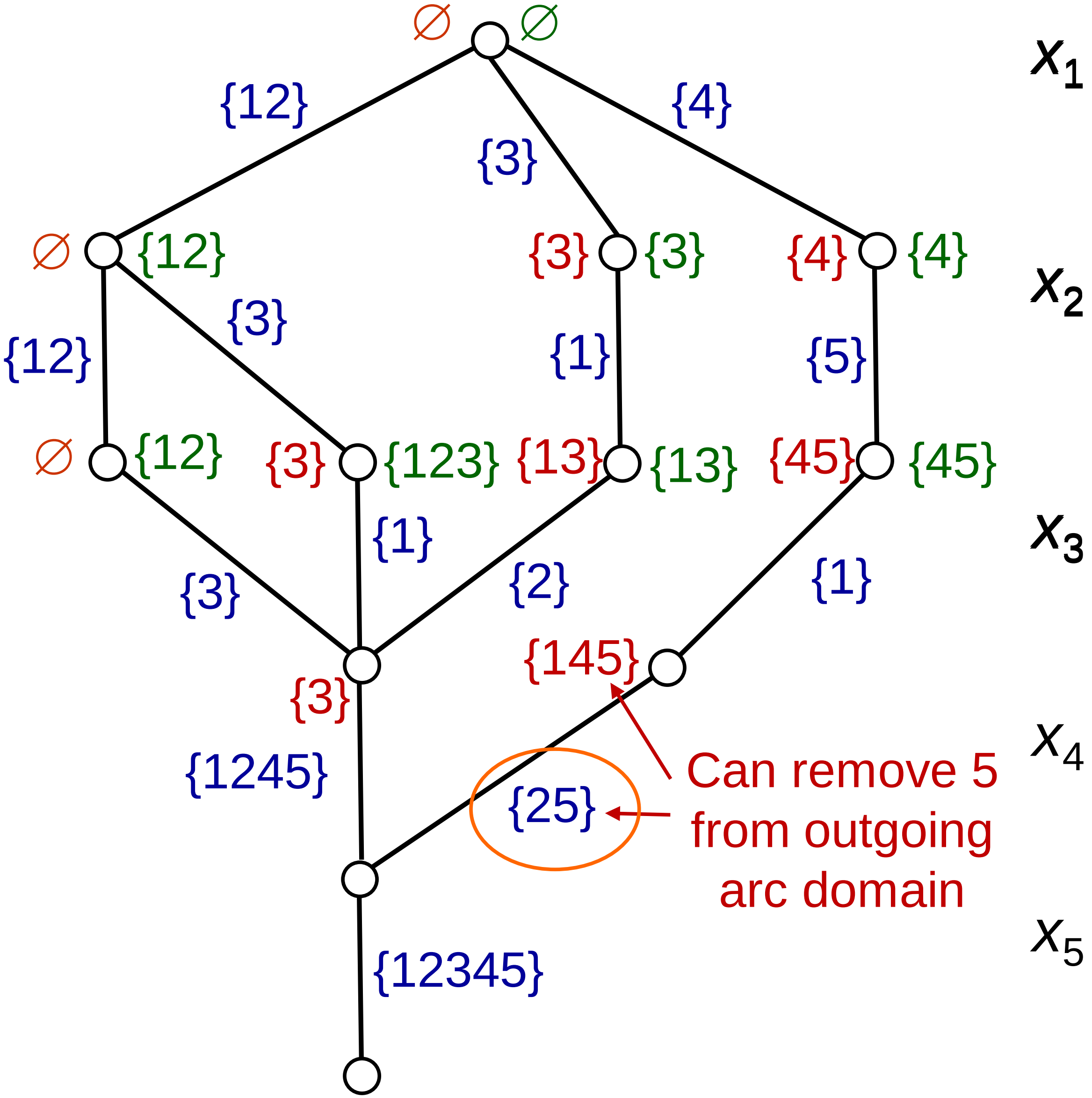




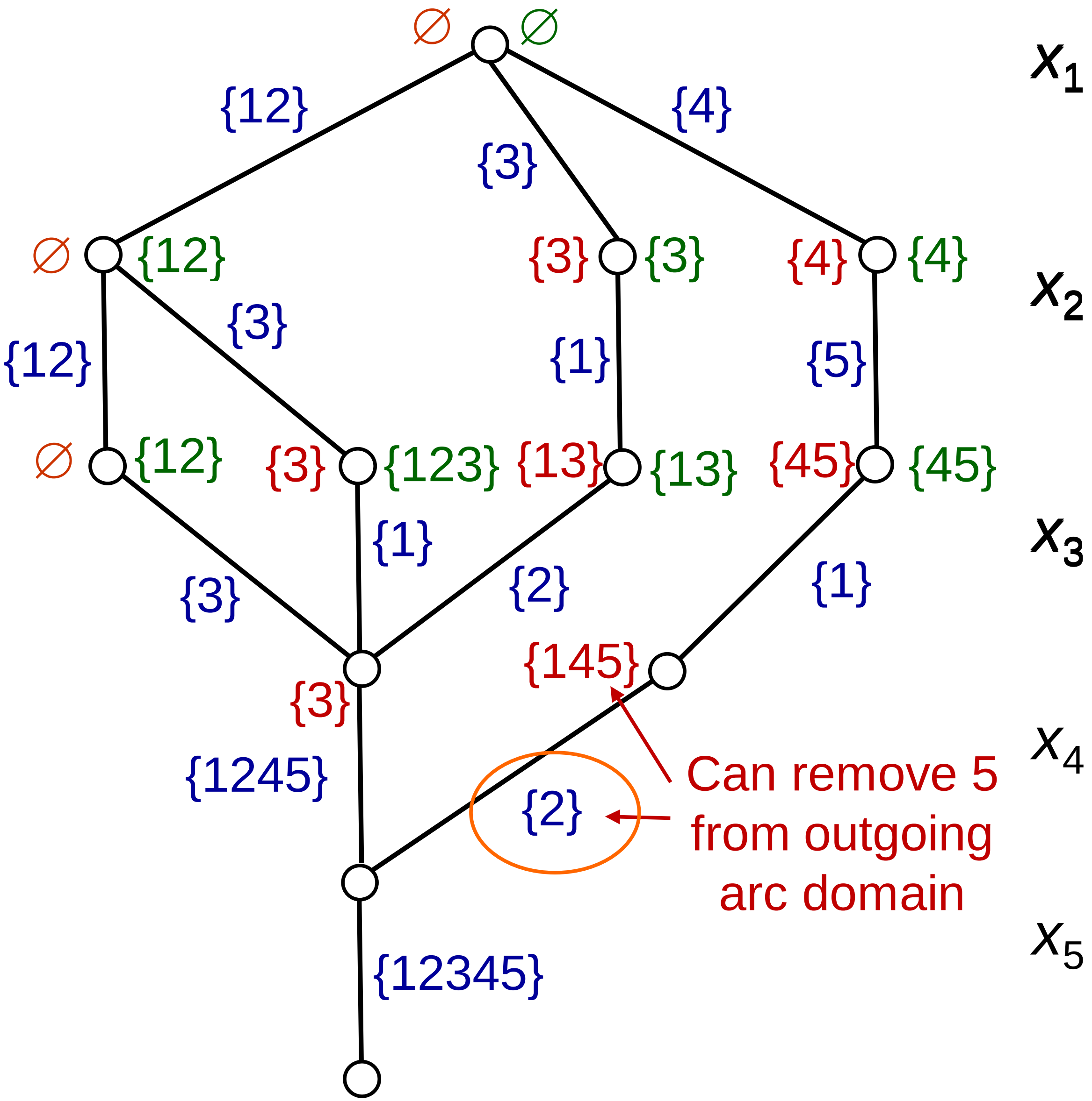
Can remove 3 from outgoing arc domain



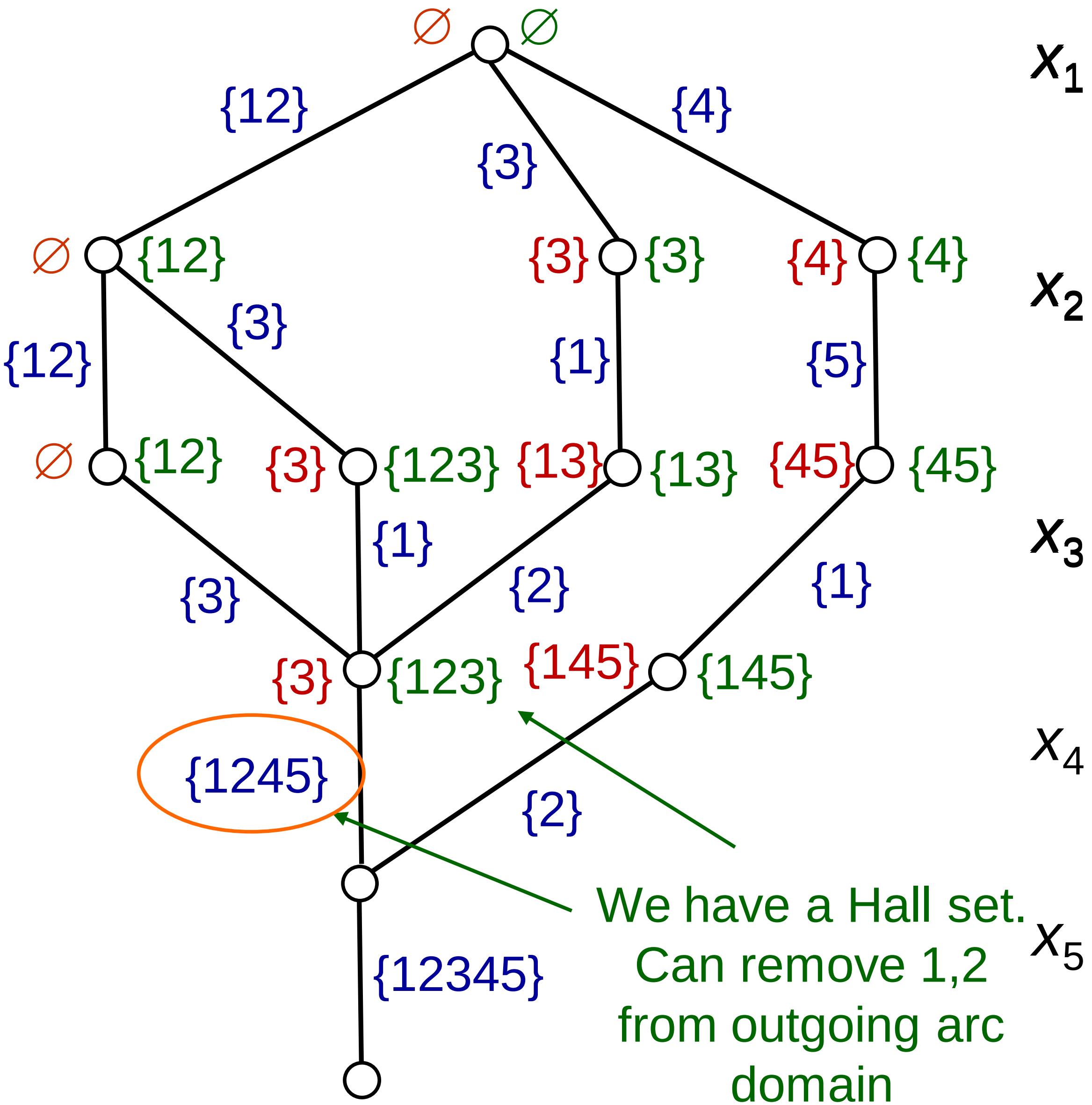
Can remove 3
from outgoing
arc domain



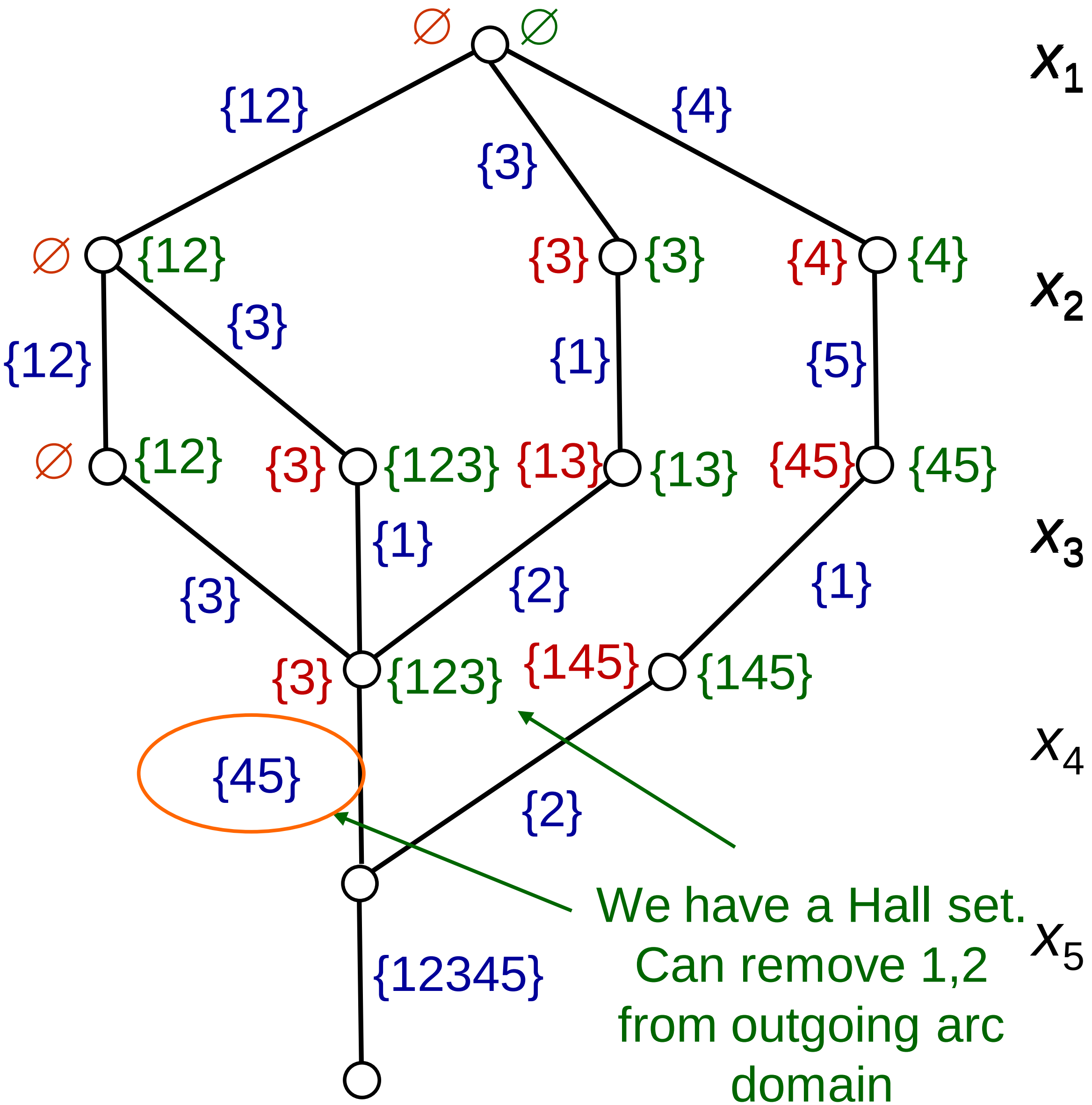
Can remove 5 from outgoing arc domain



Can remove 5 from outgoing arc domain

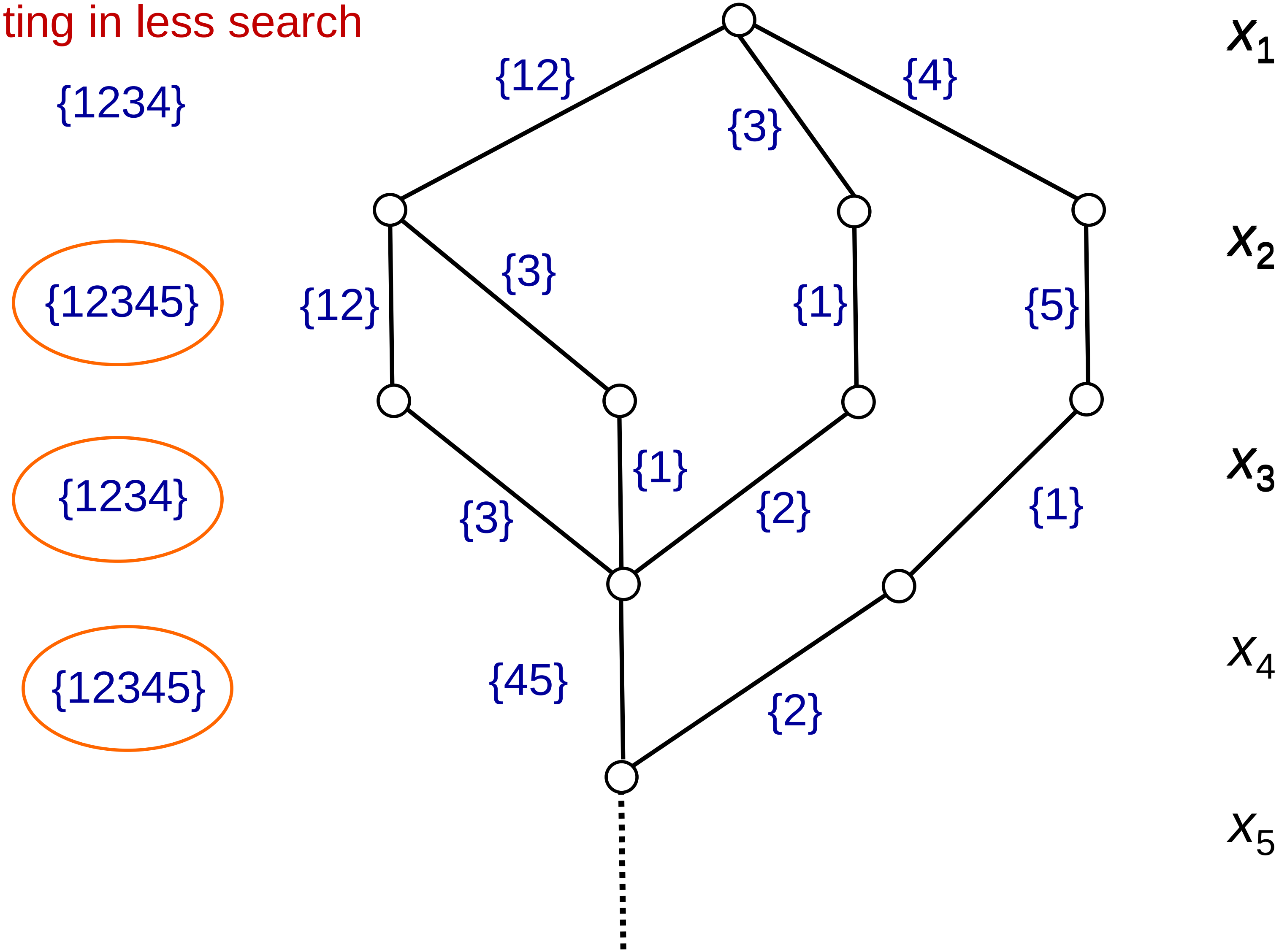


We have a Hall set.
Can remove 1,2
from outgoing arc
domain



We have a Hall set.
Can remove 1,2
from outgoing arc
domain

Now some domains can be reduced, resulting in less search



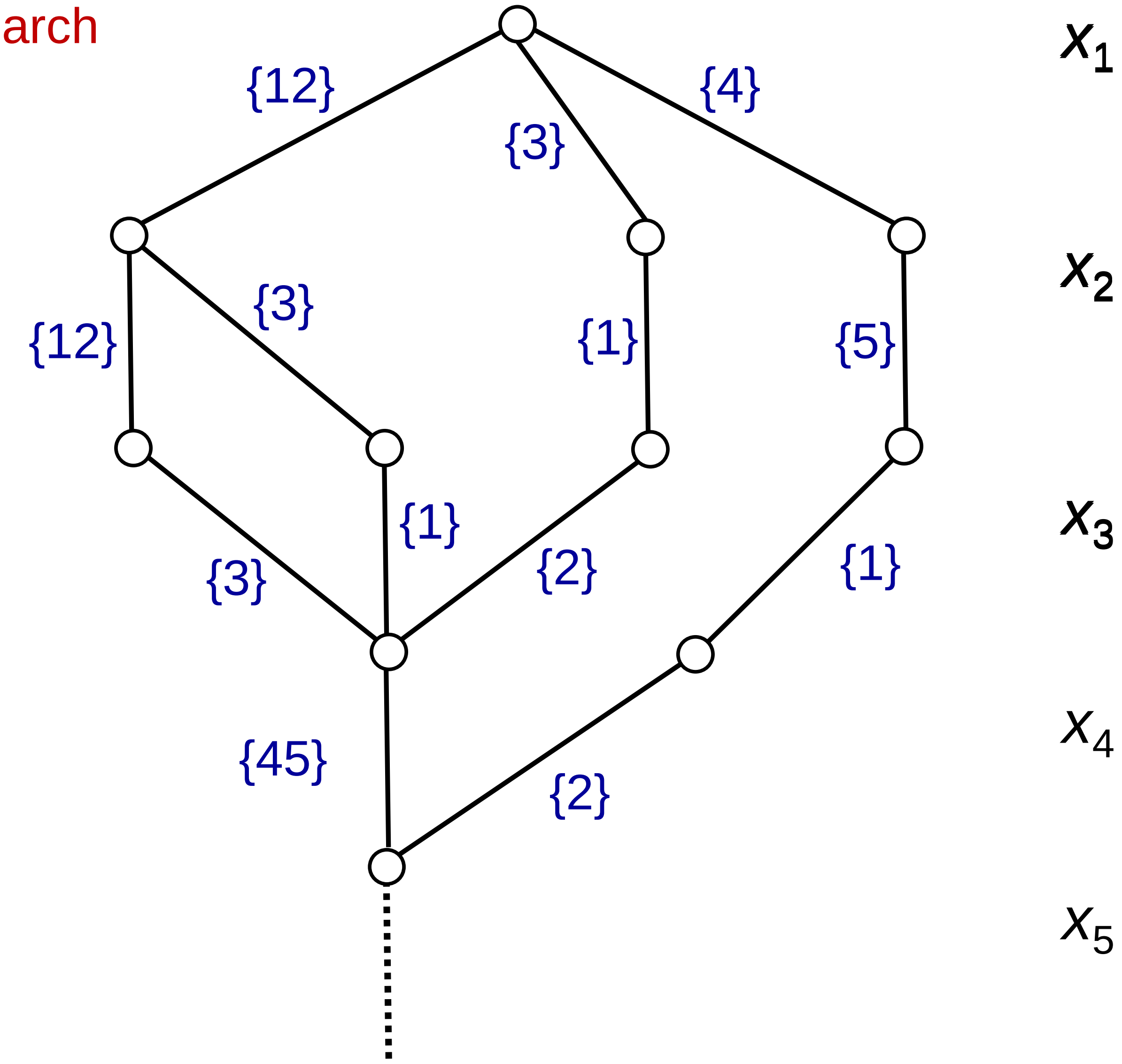
Now some domains
can be reduced,
resulting in less search

{1234}

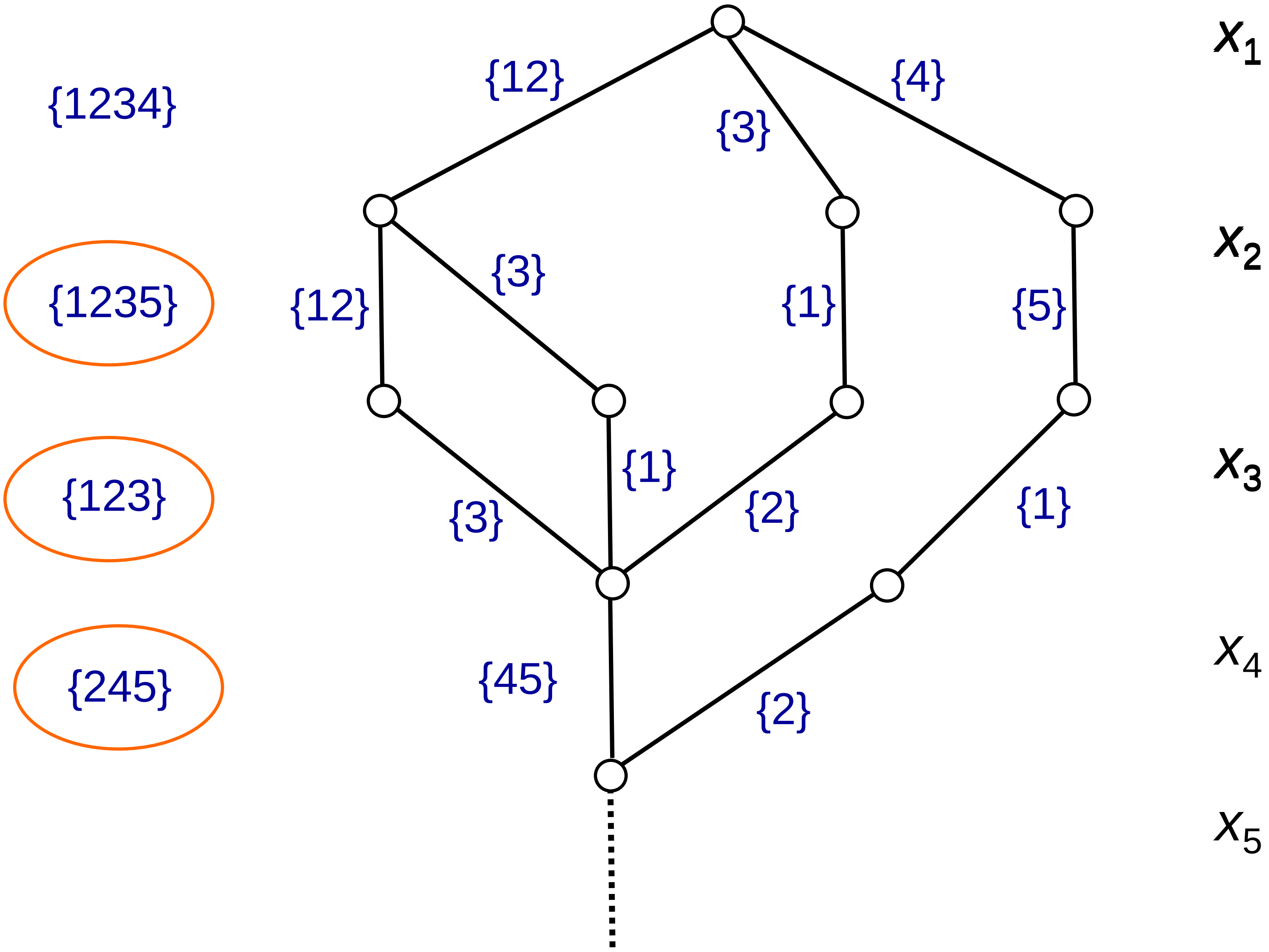
{1235}

{123}

{245}



Can follow this with a **bottom-up** pass.



Propagation in Relaxed DDs

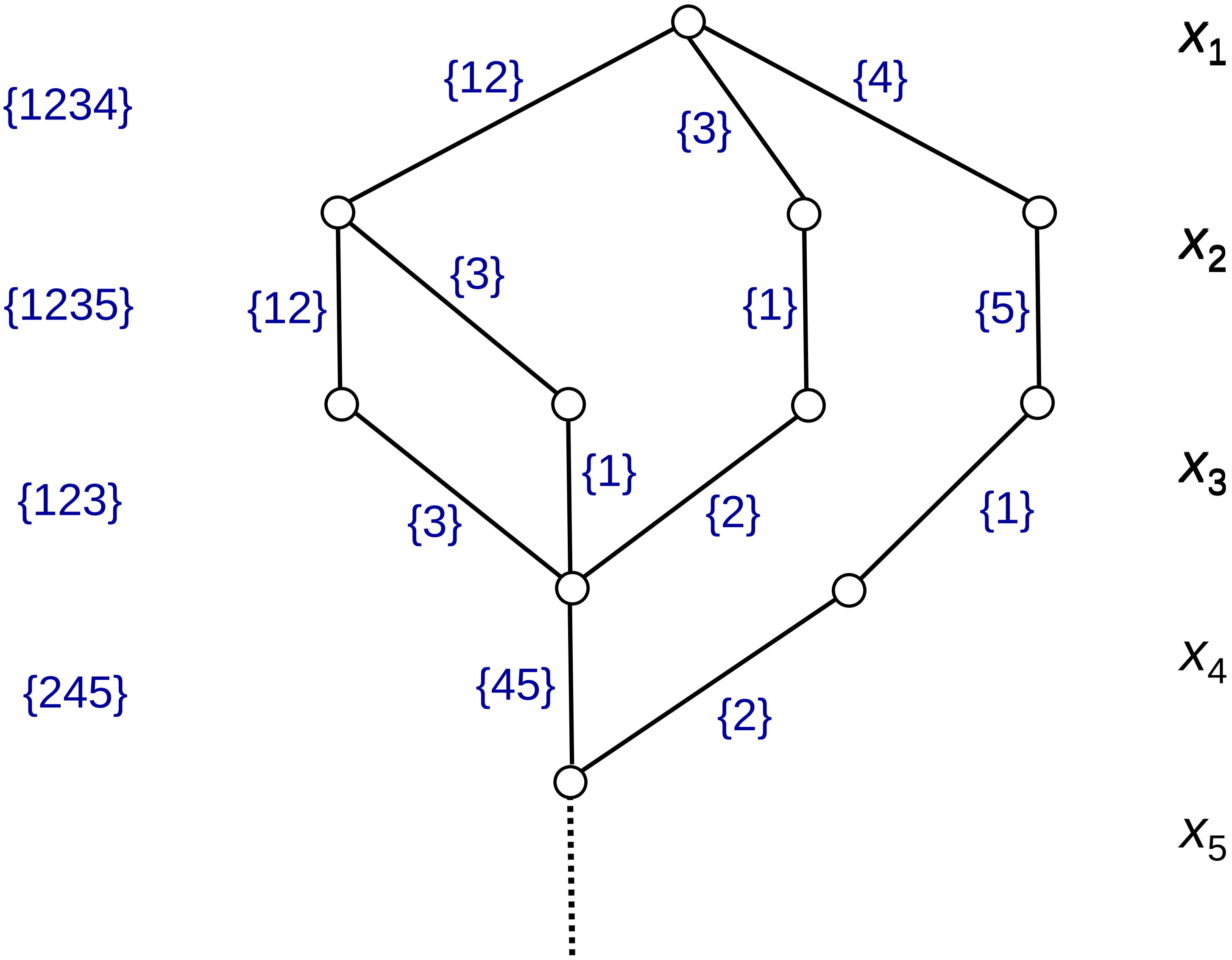
- Computational results
  - Reduced search trees from 1+ million nodes to 1 node.
  - Reduced computation time by one order of magnitude.

Andersen, Hadžić, JH, Tiedemann (2007)

Propagation in Relaxed DDs

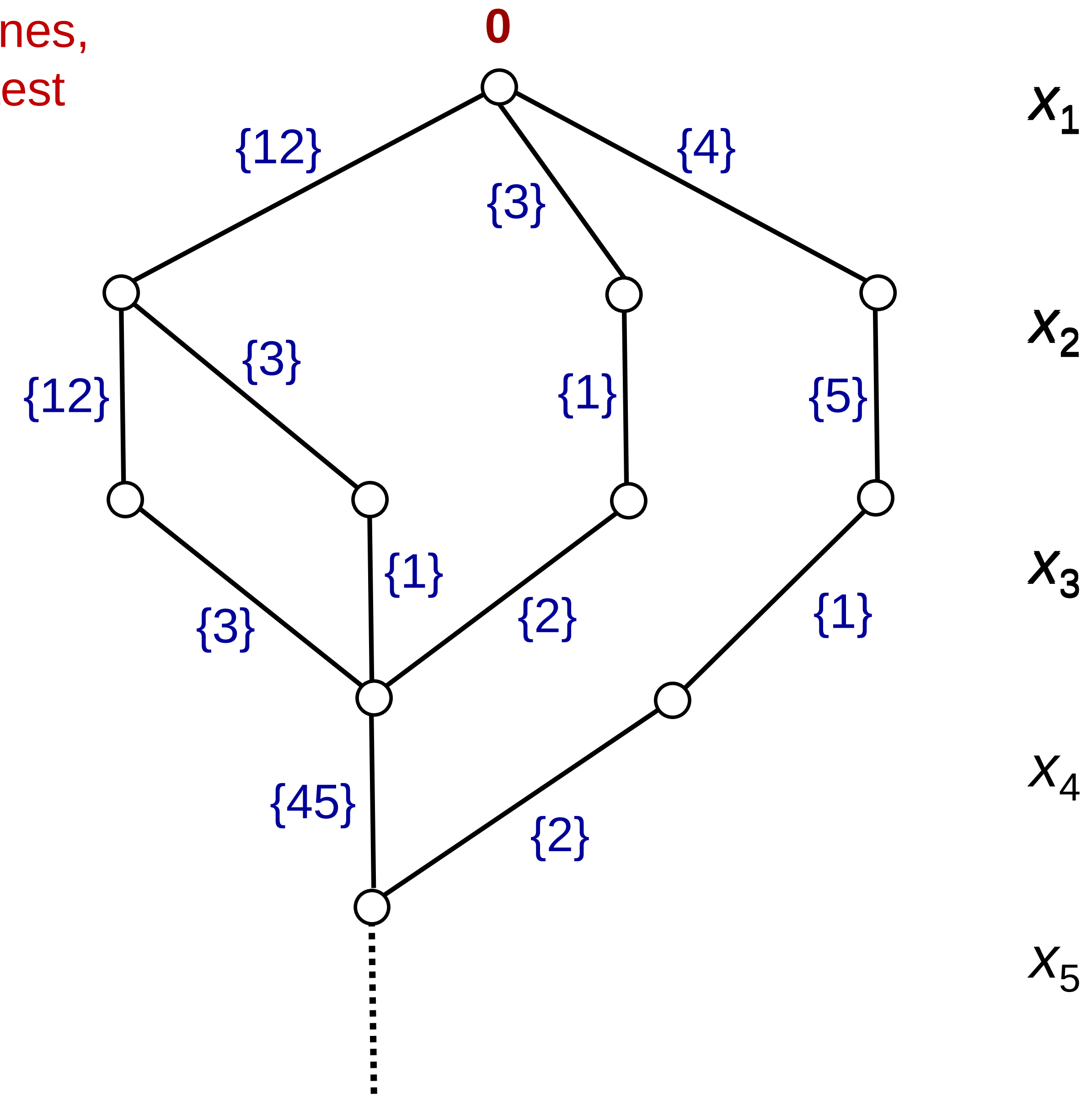
- Example 2: single-machine scheduling with time windows.
  - Schedule jobs sequentially, no overlap.
  - Each has given processing time and deadline.
  - Other constraints.
  - x_i = *i* th job in sequence
- Use same relaxed DD as before.
  - Suppose we have already propagated `alldiff`($x_1, \ldots, x_n$).
  - Now propagate **time windows**.

Current relaxed DD

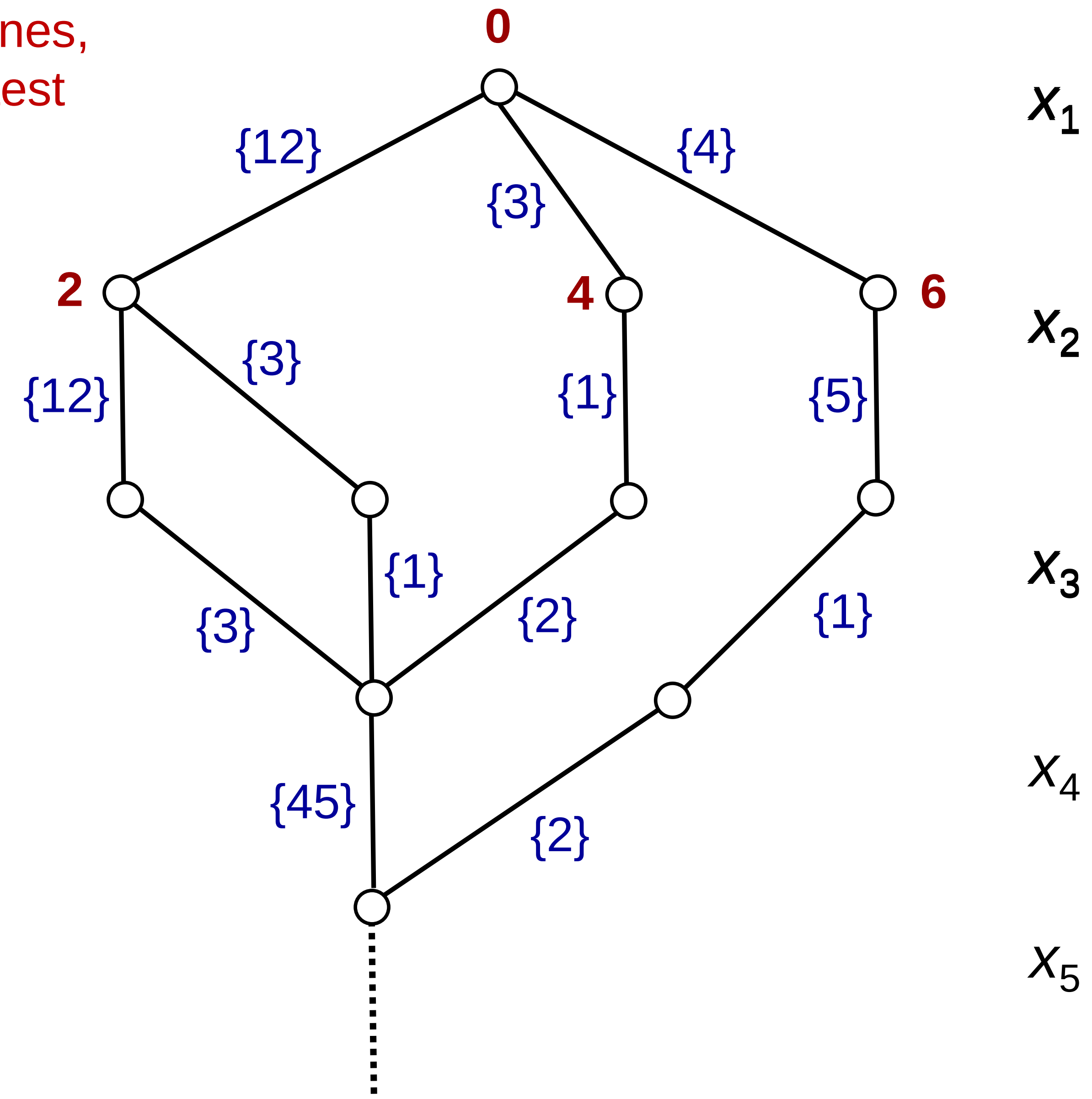


For purposes of propagating deadlines, let **state** = min latest finish time

job	Win-dow	Proc time
1	[0,4]	2
2	[3,7]	3
3	[1,8]	3
4	[5,7]	1
5	[2,10]	3
etc.		



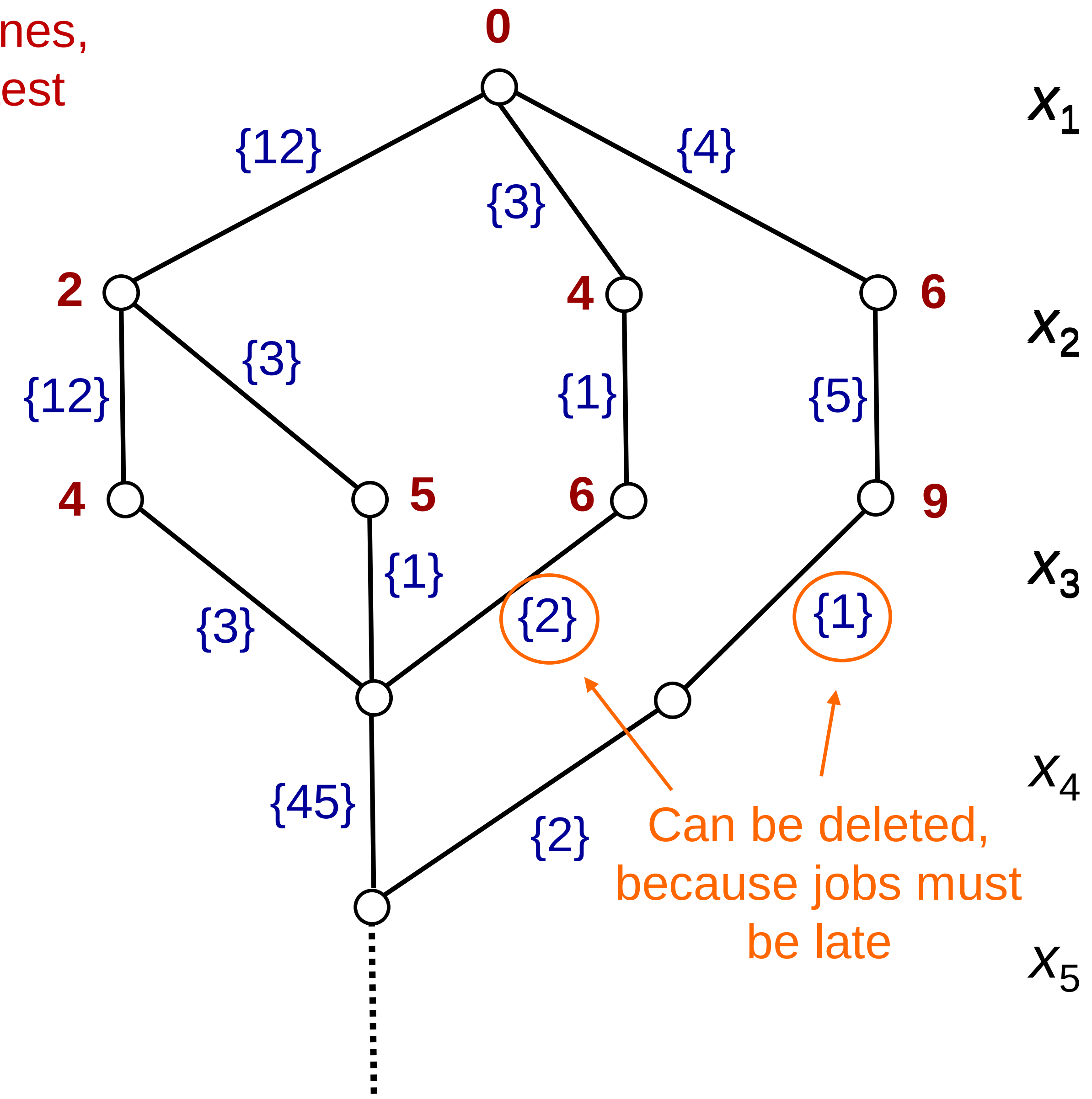
For purposes of propagating deadlines, let **state** = min latest finish time



job	Win-dow	Proc time
1	[0,4]	2
2	[3,7]	3
3	[1,8]	3
4	[5,7]	1
5	[2,10]	3
etc.		

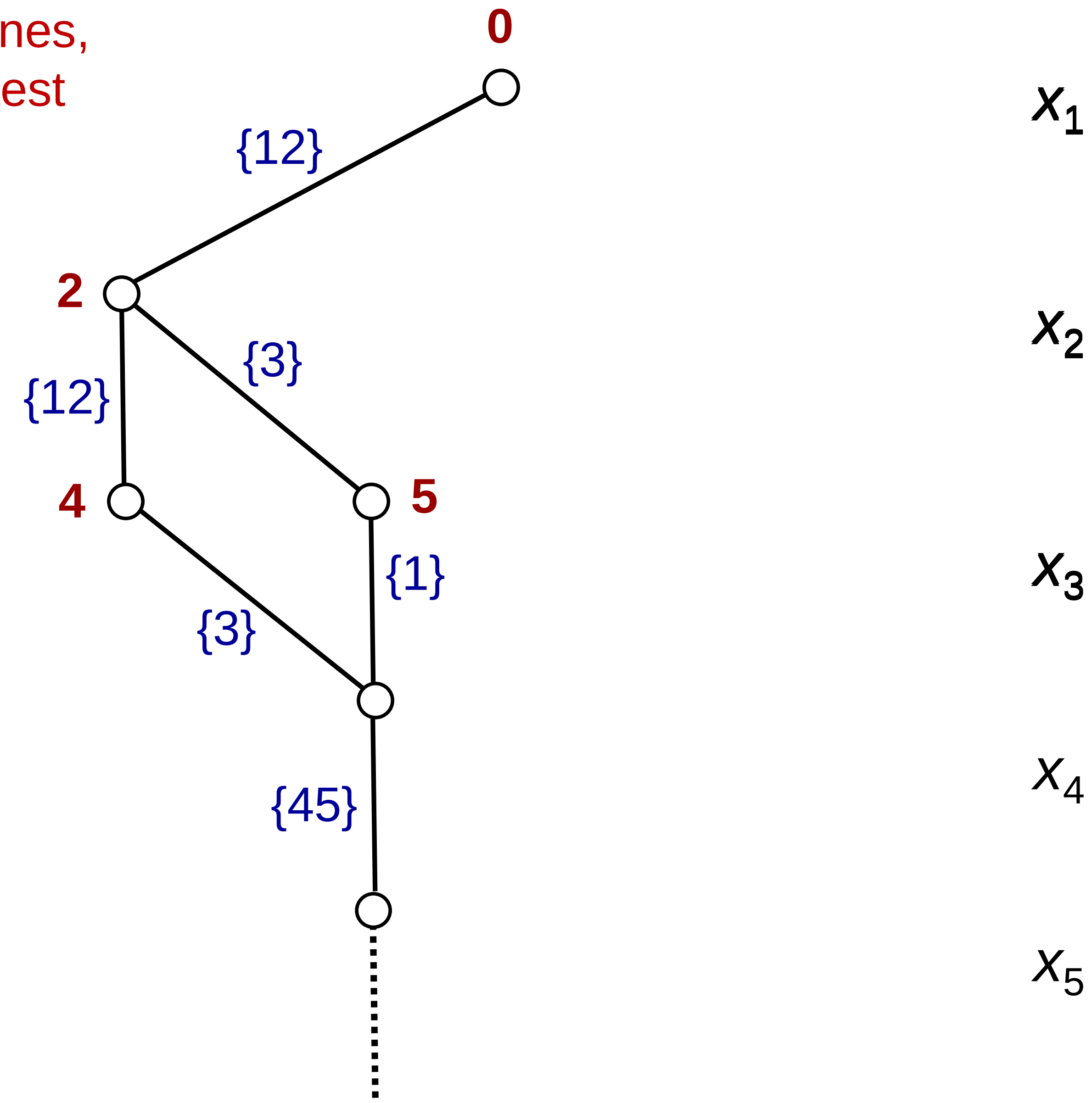
For purposes of propagating deadlines, let **state** = min latest finish time

job	Win-dow	Proc time
1	[0,4]	2
2	[3,7]	3
3	[1,8]	3
4	[5,7]	1
5	[2,10]	3
etc.		



Can be deleted, because jobs must be late

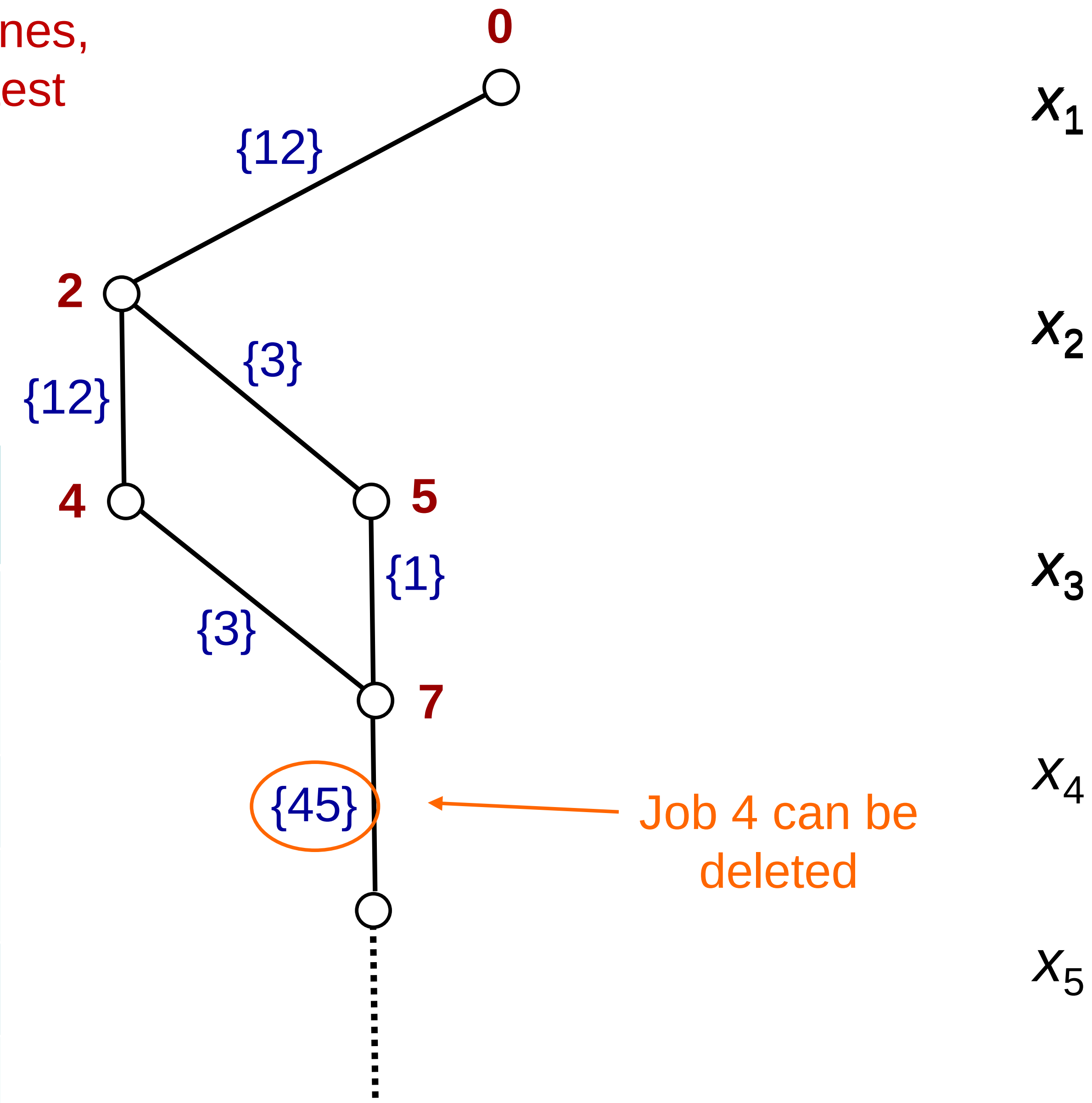
For purposes of propagating deadlines, let **state** = min latest finish time



job	Win-dow	Proc time
1	[0,4]	2
2	[3,7]	3
3	[1,8]	3
4	[5,7]	1
5	[2,10]	3
etc.		

For purposes of propagating deadlines, let **state** = min latest finish time

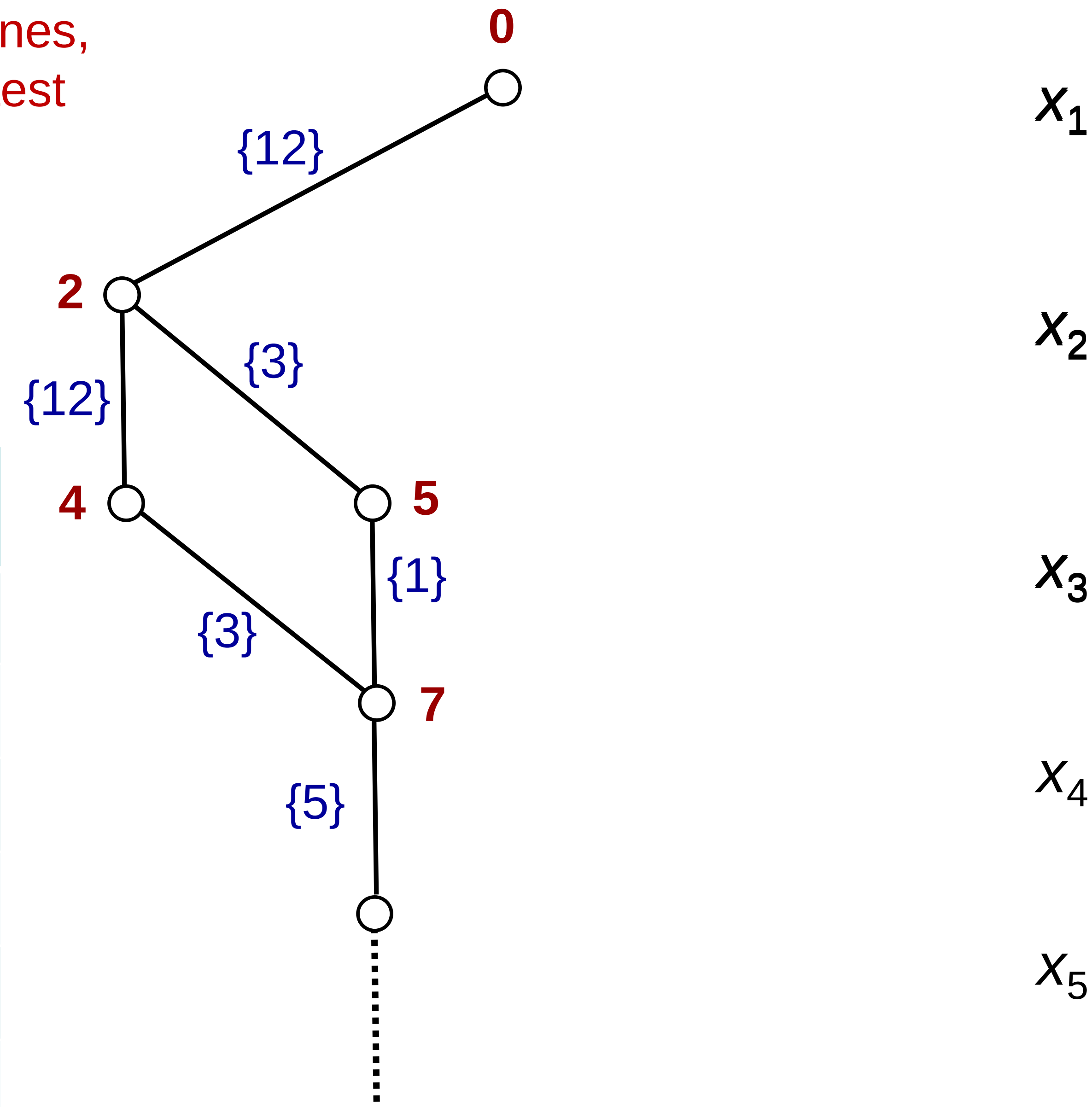
job	Win-dow	Proc time
1	[0,4]	2
2	[3,7]	3
3	[1,8]	3
4	[5,7]	1
5	[2,10]	3
etc.		



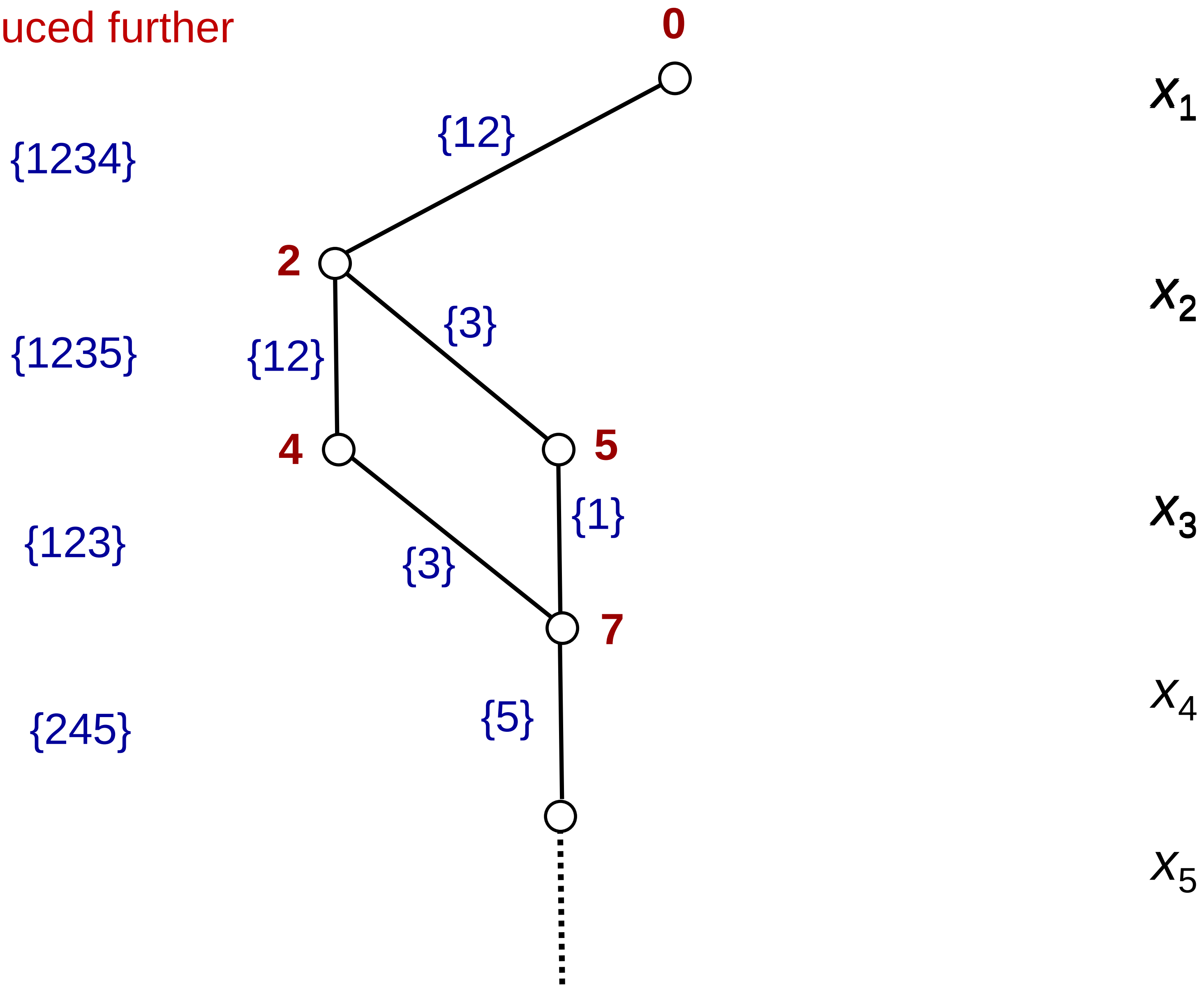
Job 4 can be deleted

For purposes of propagating deadlines, let **state** = min latest finish time

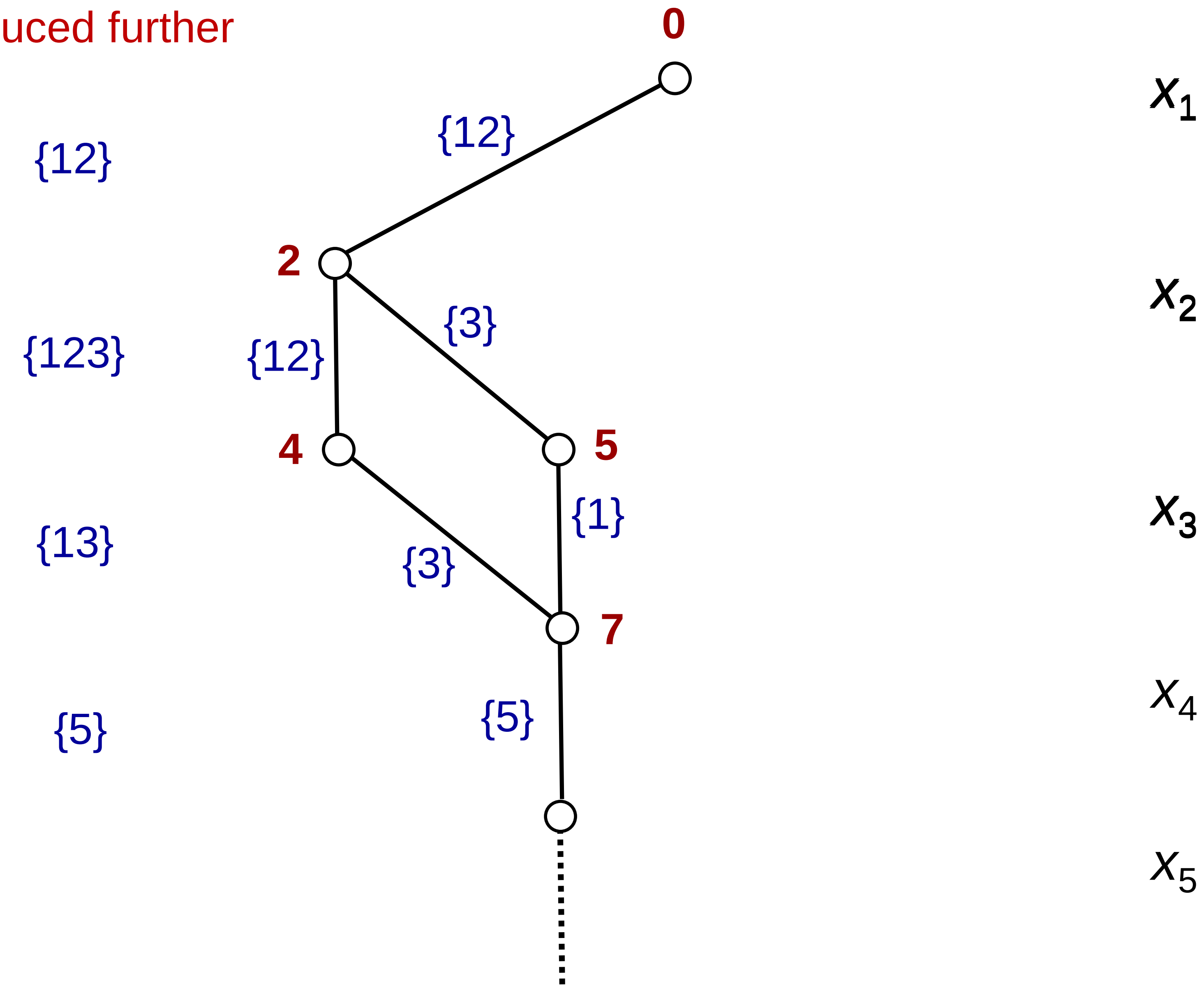
job	Window	Proc time
1	[0,4]	2
2	[3,7]	3
3	[1,8]	3
4	[5,7]	1
5	[2,10]	3
etc.		



Domains can be reduced further



Domains can be reduced further



CP Solver

- **Enhance** existing solver with DD-based **propagation**.
  - DD serves as **enhanced** domain store.
  - Can use **one or more** DDs.
    - Different **subsets of variables**
    - Different **variable orderings**
    - Propagate each constraint through suitable DD(s).
  - **Plug in** each DD as a new global constraint.

Ciré, van Hoeve (2013)

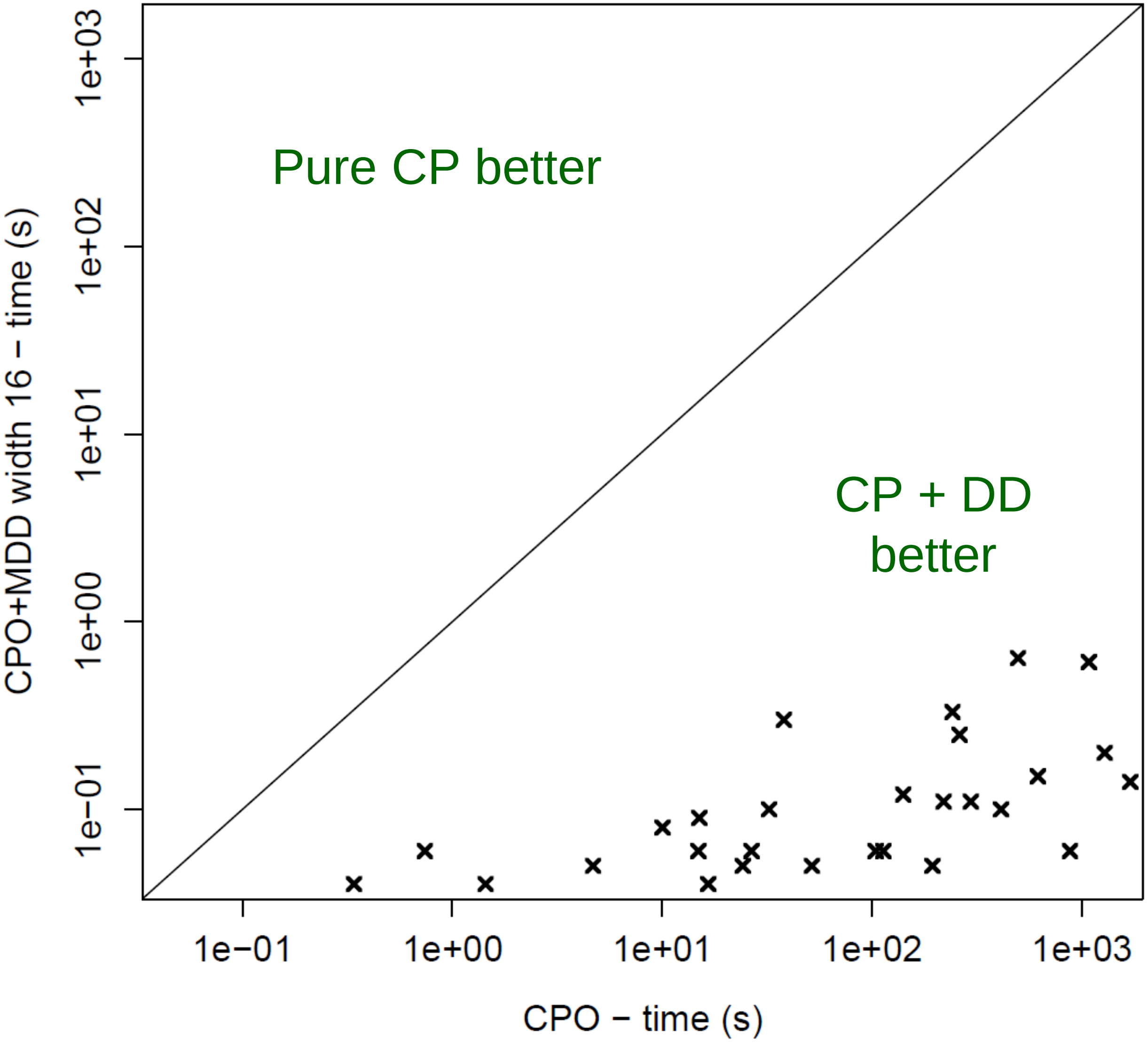
CP Solver

- Computational results.
  - Traveling salesman problem with time windows.
    - That is, single-machine scheduling with time windows **and** sequence-dependent setup times.
  - Dumas/Anscheuer instances.

Ciré, van Hoeve (2013)

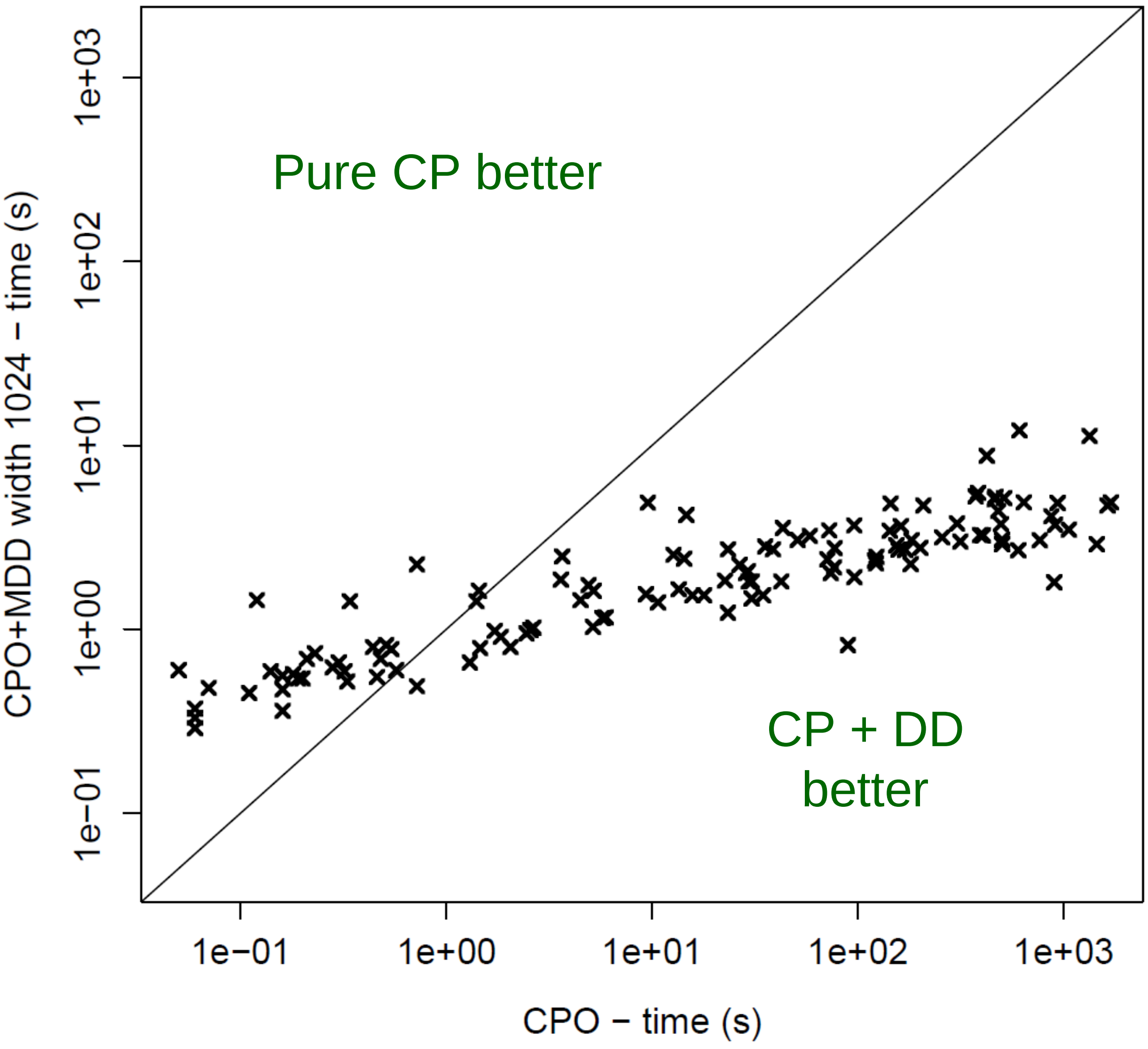
Computation time scatter plot, lex search

CPO =
CP Optimizer



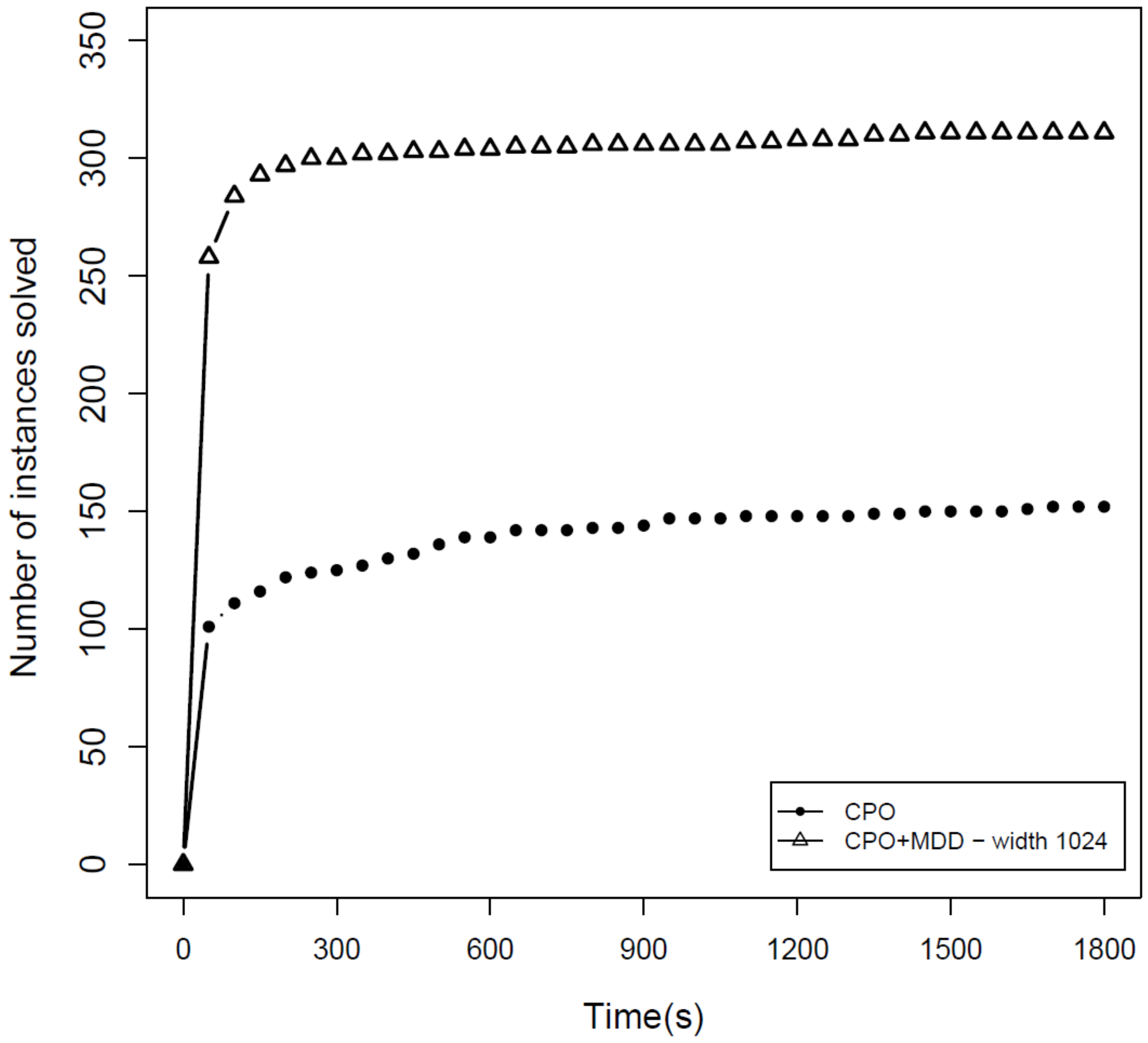
Computation time scatter plot, depth-first search

CPO =
CP Optimizer



Performance profile, depth-first search

CPO =
CP Optimizer



Restricted Decision Diagrams

- A **restricted** DD represents a **subset** of the feasible set.
- Restricted DDs provide a basis for a **primal heuristic**.
  - Shortest (longest) paths in the restricted DD provide good feasible solutions.
  - Generate a **limited-width** restricted DD by deleting nodes that appear unpromising.

Bergman, Ciré, van Hoeve, Yunes (2014)

Set covering problem

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

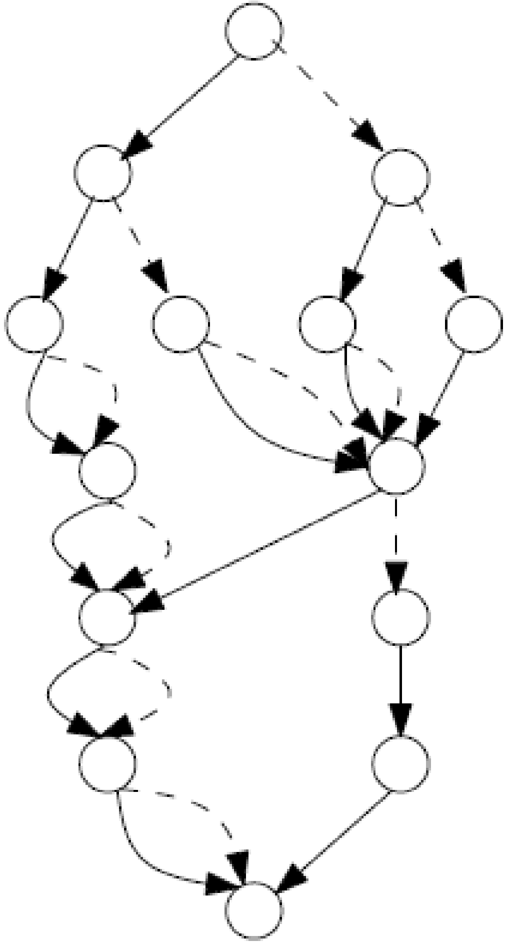
Sets

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

52 feasible solutions.

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

Restricted DD of width 4

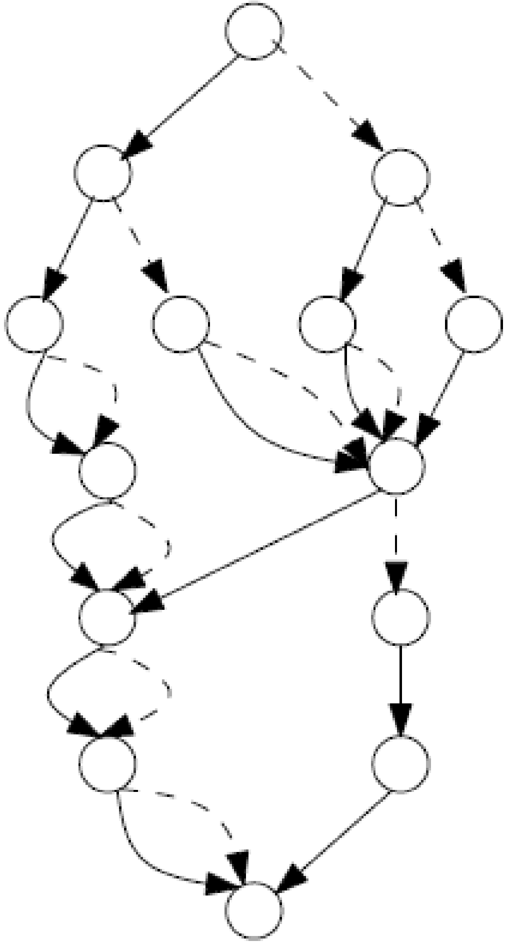


Several shortest paths have length 2.

All are minimum covers.

41 paths (< 52 feasible solutions)

Restricted DD of width 4



Several shortest paths have length 2.

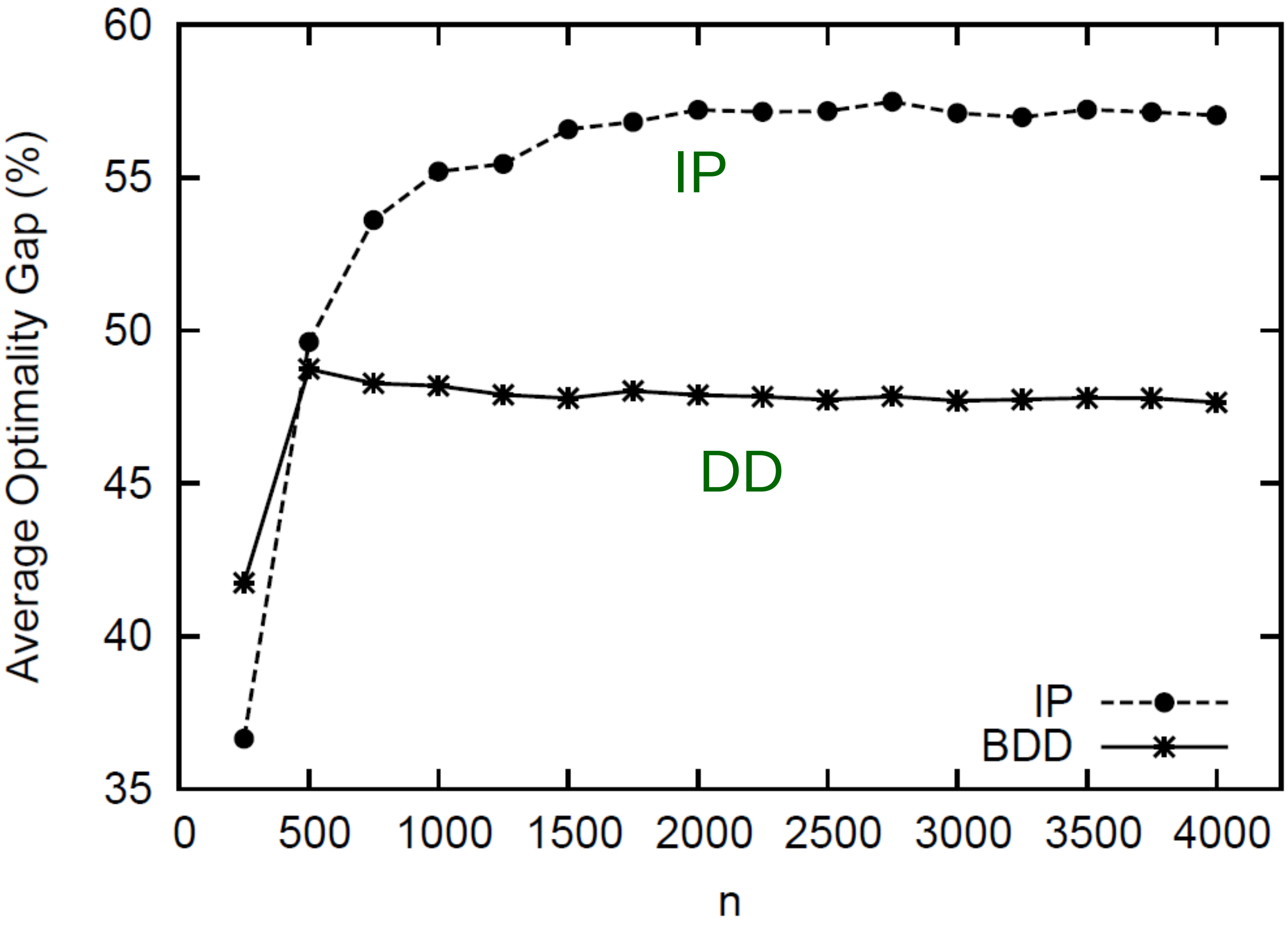
All are minimum covers.

In this case, restricted DD delivers optimal solutions.

41 paths (< 52 feasible solutions)

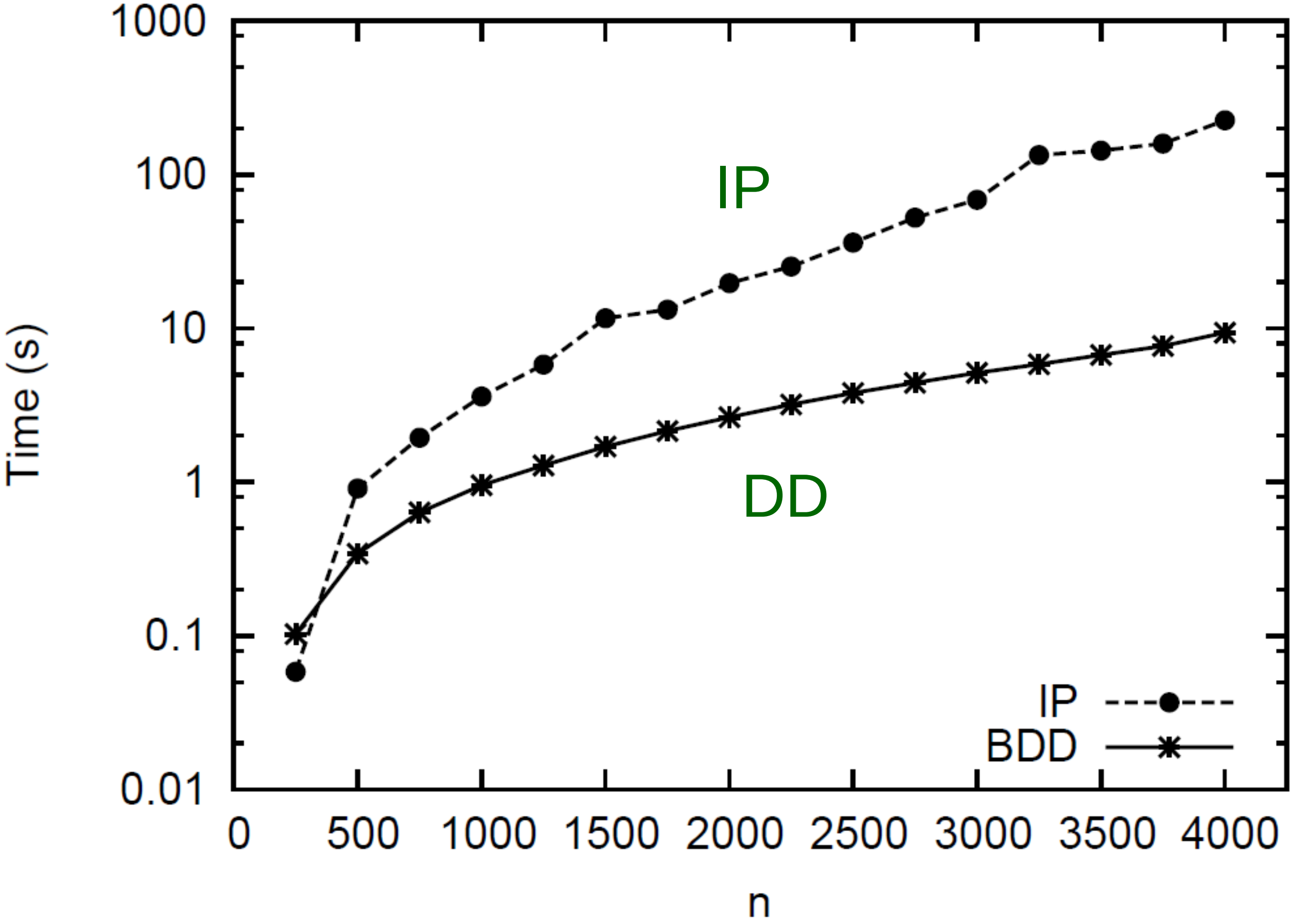
Optimality gap for set covering, *n* variables

Restricted DDs vs
Primal heuristic at root node of CPLEX



Computation time

Restricted DDs vs
Primal heuristic at root node of CPLEX (cuts turned off)



Dynamic Programming Model

- Formulate problem with **dynamic programming** model.
    - Rather than constraint set.
    - Problem must have **recursive** structure
    - But there is great **flexibility** to represent constraints and objective function.
    - Any function of **current state** is permissible.
    - We **don’t care** if state space is **exponential**, because we don’t solve the problem by dynamic programming.

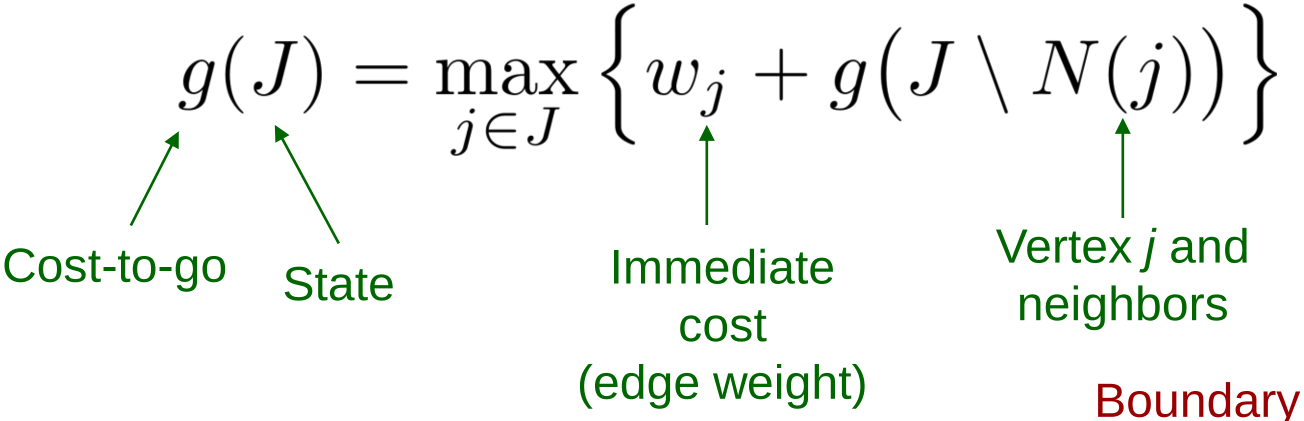
Dynamic Programming Model

- Formulate problem with **dynamic programming** model.
  - Rather than constraint set.
  - Problem must have **recursive** structure
  - But there is great **flexibility** to represent constraints and objective function.
  - Any function of **current state** is permissible.
  - We **don’t care** if state space is **exponential**, because we don’t solve the problem by dynamic programming.
- State variables are the same as in relaxed DD.
  - Must also specify **state merger** rule.

Dynamic Programming Model

- Max stable set problem on a graph.
  - **State** = set of vertices that can be added to stable set.

Recursion:



$$g(J) = \max_{j \in J} \left\{ w_j + g\big(J \setminus N(j)\big) \right\}$$

Cost-to-go

State

Immediate
cost
(edge weight)

Vertex *j* and
neighbors

Boundary condition:

$$g(\emptyset) = 0$$

Optimal value:

$$g(\{1, \ldots, n\})$$

Dynamic Programming Model

- Max stable set problem on a graph.
  - **State** = set of vertices that can be added to stable set.
  - **State merger** = union

Recursion:

$$g(J) = \max_{j \in J} \left\{ w_j + g\big(J \setminus N(j)\big) \right\}$$

Cost-to-go

State

Immediate cost (edge weight)

Vertex *j* and neighbors

Boundary condition:

$$g(\emptyset) = 0$$

Merger of states in *M* = $\bigcup_{J \in M} J$

Optimal value:

$$g(\{1, \ldots, n\})$$

Dynamic Programming Model

- Single-machine scheduling with due dates
  - Minimize total tardiness.
  - **State** = (set of jobs not yet processed,
    latest finish time of jobs processed so far)

$$g_i(J_i, f_i) = \max_{j \in J_i} \left\{ (f_i + p_j - d_j)^+ + g_{i+1}(J_i \setminus \{j\}, f_i + p_j) \right\}$$

Cost-to-go

Jobs remaining

Last finish time

Tardiness of job *j*

Boundary condition:
$g_{n+1}(\emptyset, f_{n+1}) = 0$

Optimal value:
$g_1(\{1, \ldots, n\}, 0)$

Dynamic Programming Model

- Single-machine scheduling with due dates
  - Minimize total tardiness.
  - **State** = (set of jobs not yet processed, latest finish time of jobs processed so far)
  - **State merger** = union, min

$$g_i(J_i, f_i) = \max_{j\in J_i}\left\{(f_i+p_j-d_j)^+ + g_{i+1}(J_i\setminus\{j\}, f_i+p_j)\right\}$$

Cost-to-go

Jobs remaining

Last finish time

Tardiness of job *j*

Boundary condition:

$$g_{n+1}(\emptyset, f_{n+1}) = 0$$

Optimal value:

$$g_1(\{1,\ldots,n\},0)$$

Merger of states in *M* = $\left(\bigcup_{(J_i,f_i)\in M} J_i,\ \min_{(J_i,f_i)\in M}\{f_i\}\right)$

Dynamic Programming Model

- Single machine scheduling with due dates
  - Easy to **add constraints** that are functions of current state
    - Release times
    - Shutdown periods
    - Precedence constraints on jobs
  - Easy to use **more complicated cost function** that is a function of current state
    - Step functions, etc.
    - Cost that depends on which jobs have been processed.

Dynamic Programming Model

- Scheduling with sequence-dependent setup times
  - **State** = (J_i, last job processed, f_i)
  - **State merger** requires modification of states

$$g_i(J_i, \ell_i, f_i) = \max_{j \in J_i} \left\{ (f_i + p_{\ell_i j} - d_j)^+ + g_{i+1}\big(J_i \setminus \{j\}, j, f_i + p_{\ell_i j}\big) \right\}$$

Last job processed

Tardiness of job *j*

Processing + setup time

Dynamic Programming Model

- Scheduling with sequence-dependent setup times
  - To allow for **state merger:**
  - **State** = (J_i, set L_i of pairs (ℓ_i, f_i), representing jobs that could have been the last processed)

$$g_i(J_i, L_i) = \max_{j \in J_i} \left\{ \left(\min_{(\ell_i, f_i) \in L_i} \{f_i + p_{\ell_i j}\} - d_j \right)^+ + g_{i+1}\left(J_i \setminus \{j\}, \left\{ \left(j, \min_{(\ell_i, f_i) \in L_i} \{f_i + p_{\ell_i j}\} \right) \right\} \right) \right\}$$

Merger of states in *M* = $\left(\bigcup_{(J_i, L_i) \in M} J_i, \bigcup_{(J_i, L_i) \in M} L_i, \right)$

Dynamic Programming Model

- Max cut problem on a graph.
  - Partition nodes into 2 sets so as to maximize total weight of connecting edges.
  - **State** = marginal benefit of placing each remaining vertex on left side of cut.
  - **State merger** =
    - Componentwise min if all components ≥ 0 or all ≤ 0; 0 otherwise
    - Adjust incoming arc weights
- Max 2-SAT.
  - Similar to max cut.

Branching Algorithm

- Solve optimization problem using a novel **branch-and-bound** algorithm.
  - Branch on nodes in **last exact layer** of relaxed decision diagram.
    - …rather than branch on variables.
    - Create a new **relaxed DD rooted** at each branching node.
    - Prune search tree using bounds from relaxed DD.

Bergman, Ciré, van Hoeve, JH (2016)

Branching Algorithm

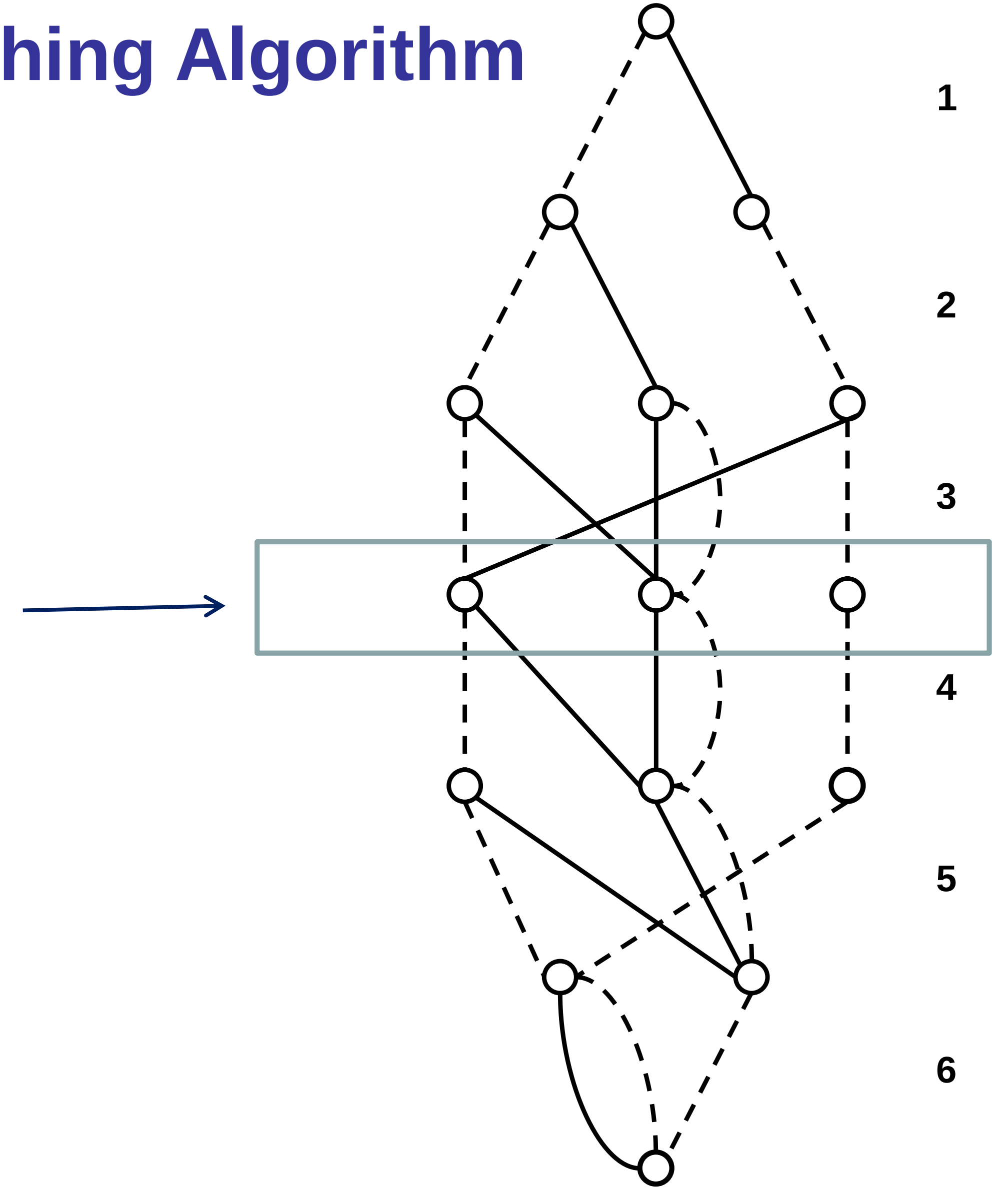
- Solve optimization problem using a novel **branch-and-bound** algorithm.
  - Branch on nodes in **last exact layer** of relaxed decision diagram.
    - …rather than branch on variables.
    - Create a new **relaxed DD rooted** at each branching node.
    - Prune search tree using bounds from relaxed DD.
  - Advantage: a manageable number states may be reachable in first few layers.
    - …even if the state space is **exponential**.
    - Alternative way of dealing with **curse of dimensionality**.

Bergman, Ciré, van Hoeve, JH (2016)

Branching Algorithm

Branching in a relaxed decision diagram

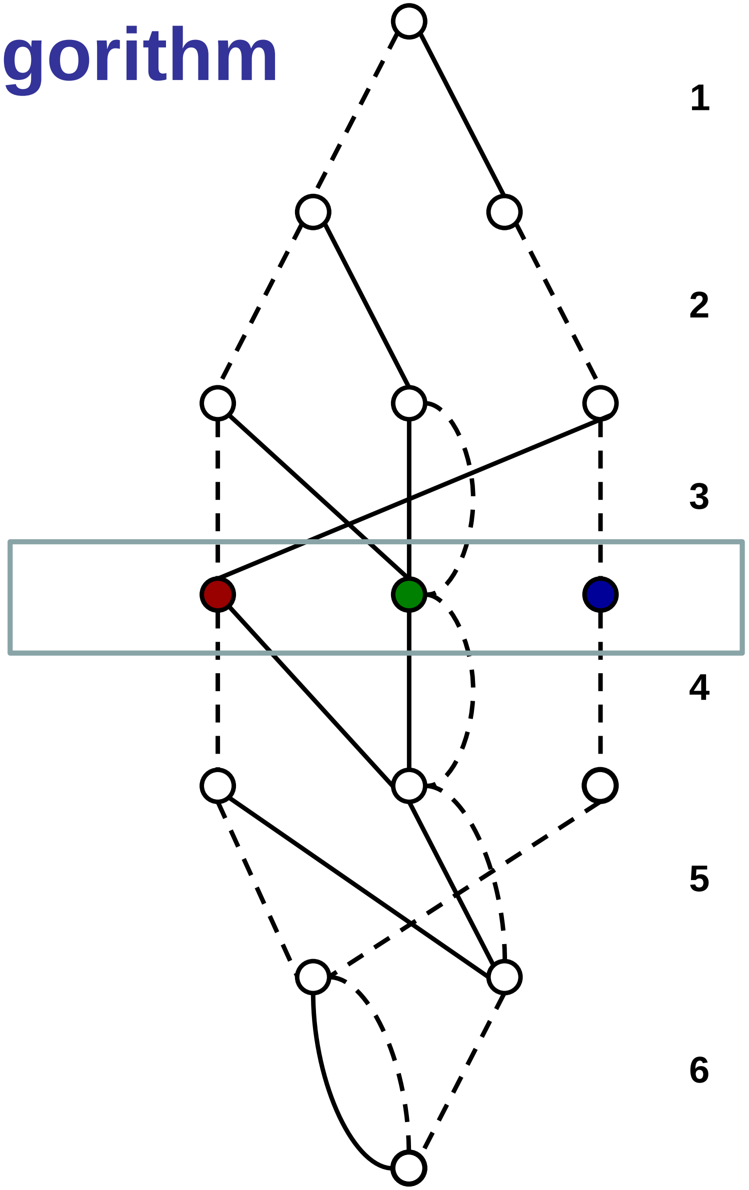
Diagram is exact down to here



Branching Algorithm

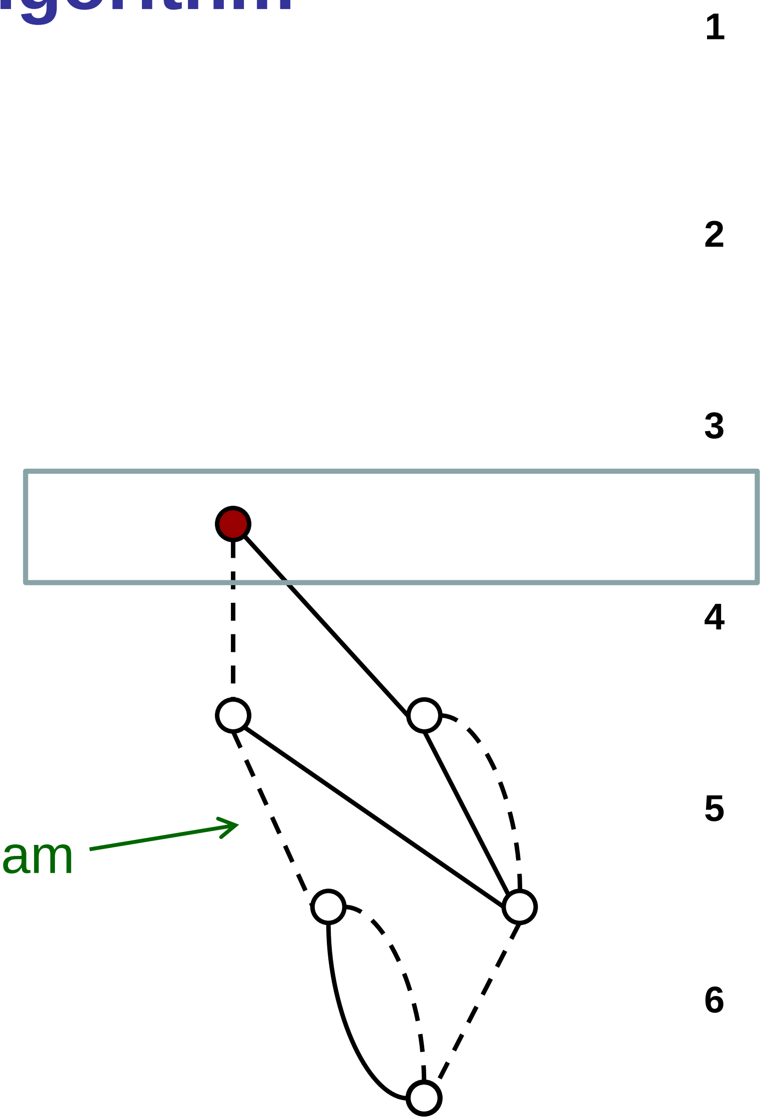
Branching in a relaxed decision diagram

Branch on nodes in this layer



Branching Algorithm

Branching in a relaxed decision diagram

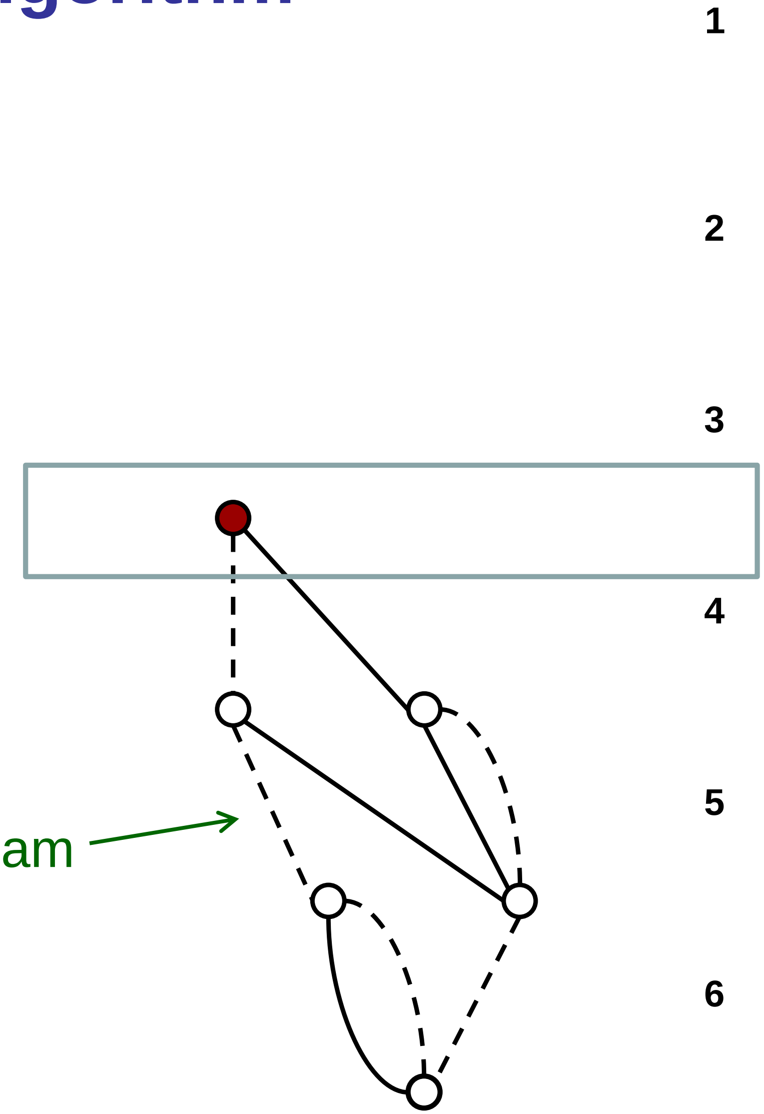


First branch

New relaxed decision diagram

Branching Algorithm

Branching in a relaxed decision diagram



First branch

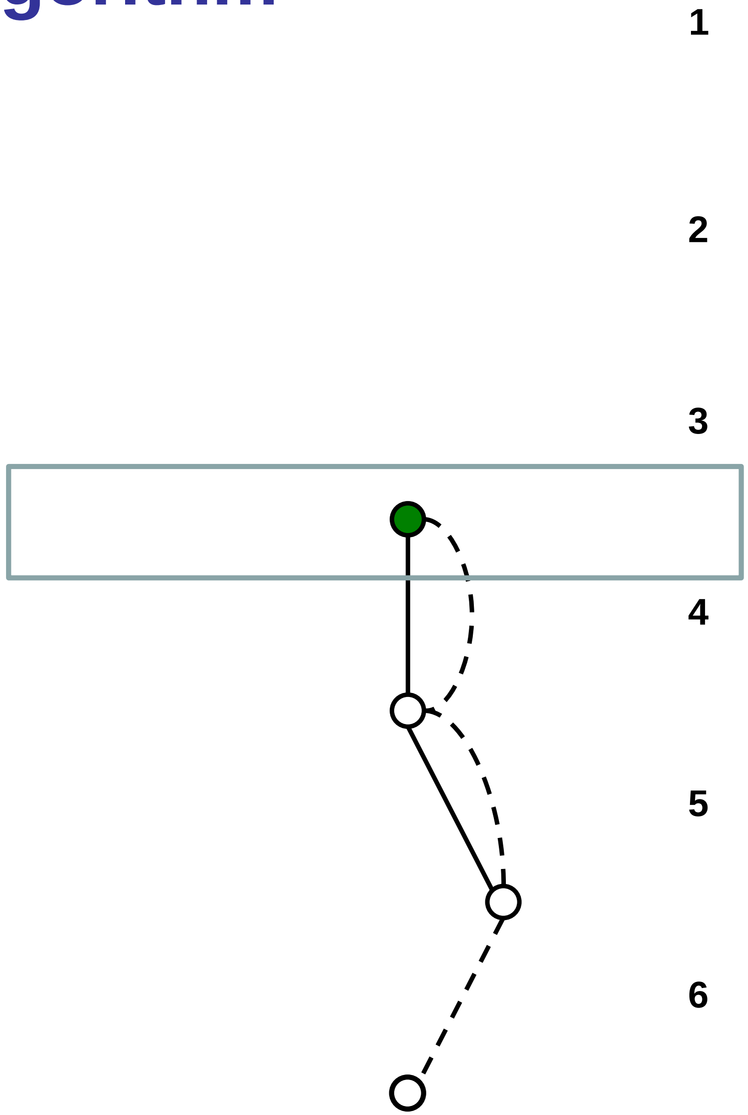
New relaxed decision diagram

Prune this branch if **cost bound** from relaxed DD is **no better** than cost of best feasible solution found so far (**branch and bound**).

Branching Algorithm

Branching in a relaxed decision diagram

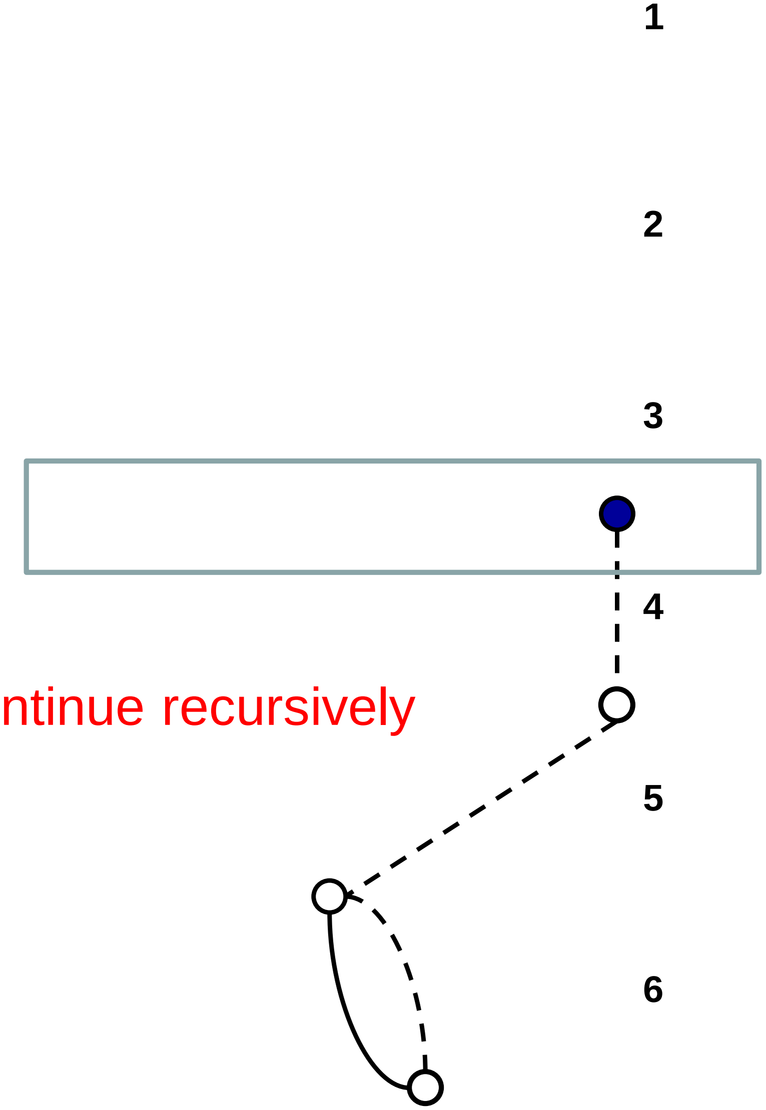
Second branch



Prune this branch if **cost bound** from relaxed DD is **no better** than cost of best feasible solution found so far (**branch and bound**).

Branching Algorithm

Branching in a relaxed decision diagram



Third branch

Continue recursively

Prune this branch if **cost bound** from relaxed DD is **no better** than cost of best feasible solution found so far (**branch and bound**).

State Space Relaxation?

- This is **very different** from state space relaxation.
  - Problem is **not solved by dynamic programming**.
  - Relaxation created by **merging nodes of DD**
    - …rather than mapping into smaller state space.
  - Relaxation is **constructed dynamically**
    - …as relaxed DD is built.
  - Relaxation uses **same state variables** as exact formulation
    - …which allows branching in relaxed DD

Christofides, Mingozzi, Toth (1981)

Discrete Optimization Solver

- Enhance existing solver with DDs
  - Better **bounds** from **relaxed** DDs.
  - Better **primal heuristic** using **restricted** DDs.
  - Add to existing LP relaxation and primal heuristics.
- Use stand-alone DD-based solver
  - Obtain **bounds** from **relaxed** DDs.
  - Use **restricted** DDs for **primal heuristic**.
  - Use **dynamic programming** formulation of problem.
  - **Branch** inside relaxed DD.

Computational performance

- Computational results…
  - Applied to stable set, max cut, max 2-SAT.
    - Superior to commercial MIP solver (CPLEX) on most instances.
    - Obtained best known solution on some max cut instances.
  - Slightly slower than MIP on stable set with precomputed clique cover model, but…

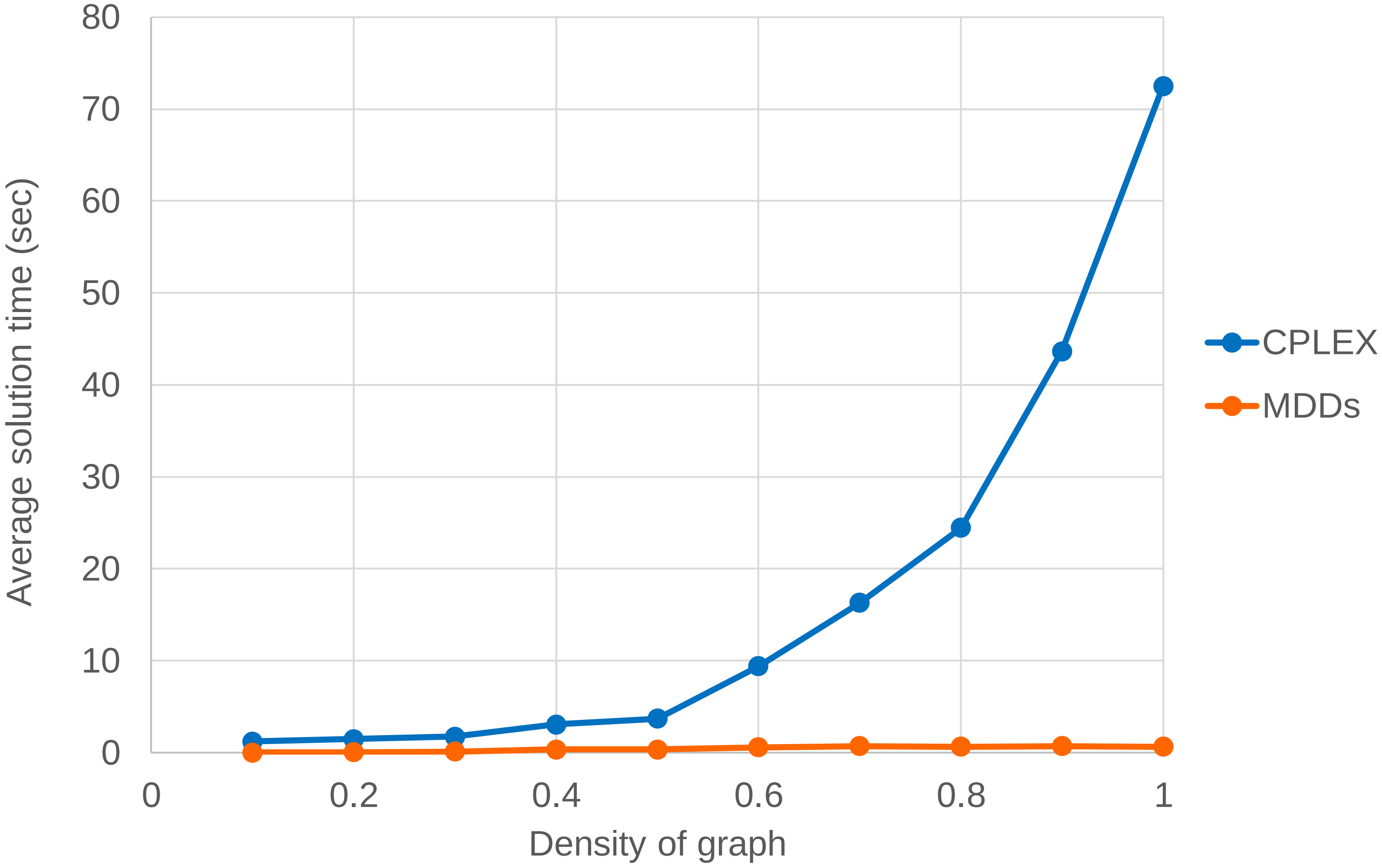
Bergman, Ciré, van Hoeve, JH (2016)

Computational performance

Max cut
on a graph

Avg. solution time
vs
graph density

30 vertices

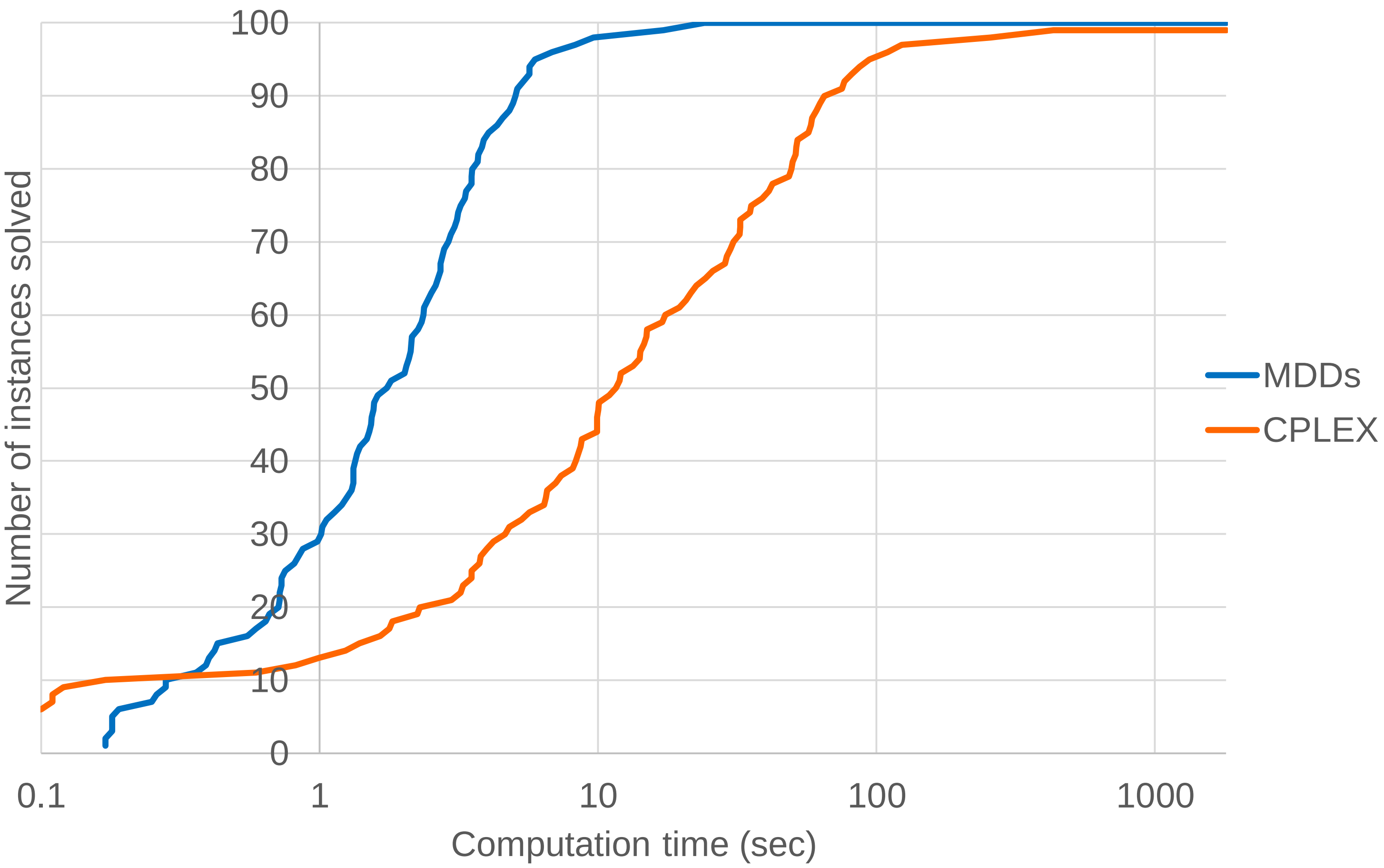


Computational performance

Max 2-SAT

Performance profile

30 variables

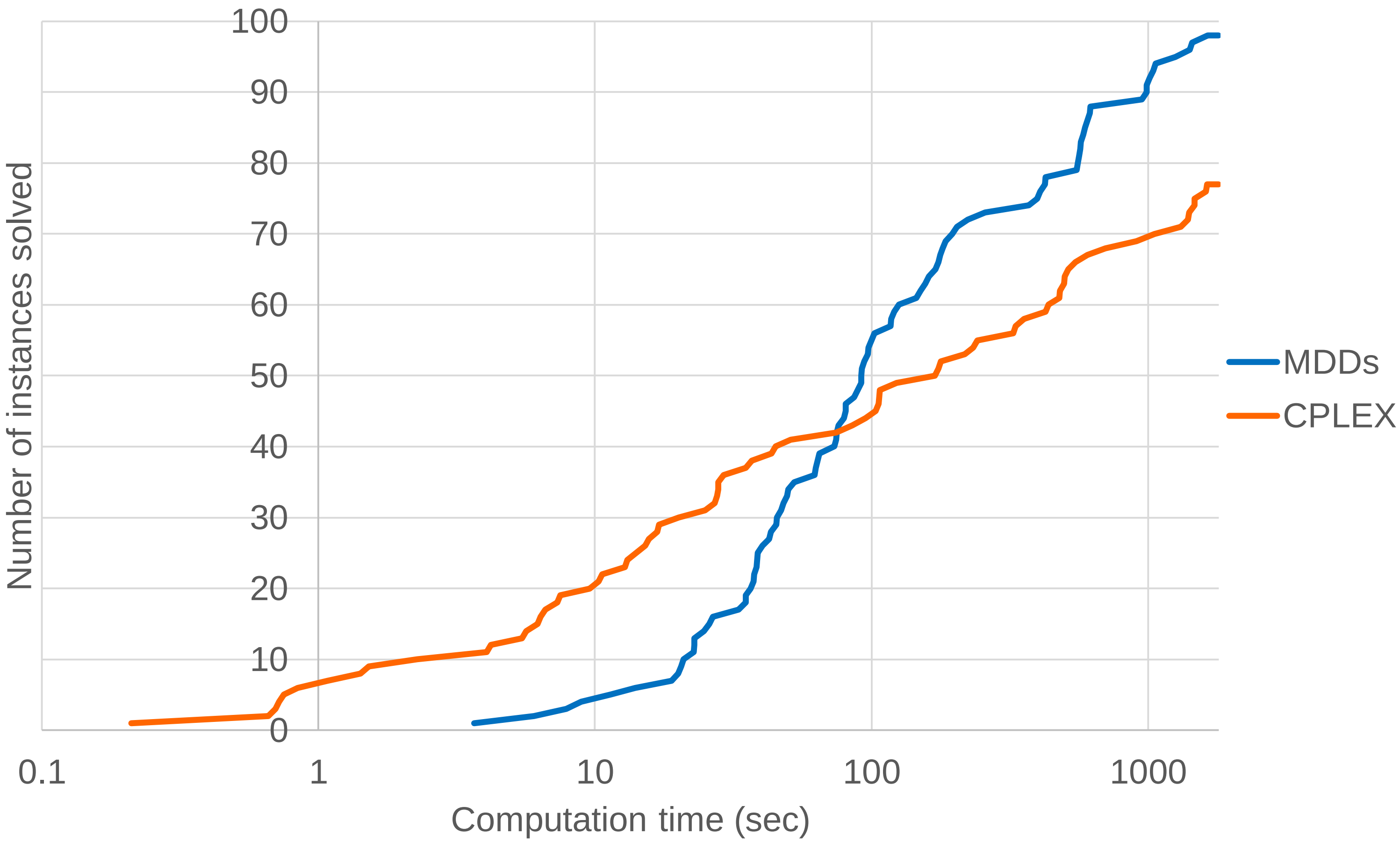


Computational performance

Max 2-SAT

Performance profile

40 variables



Computational performance

- Potential to scale up
  - No need to load large inequality model into solver.
  - **Parallelizes** very effectively
    - **Near-linear** speedup.
    - Much better than mixed integer programming.

Computational performance

- In all computational comparisons so far…
  - Problem is **easily formulated for IP**.
- DD-based optimization is most competitive when…
  - Problem has a recursive **dynamic programming** model…
  - and no convenient IP model.

Computational performance

- In all computational comparisons so far…
  - Problem is **easily formulated for IP**.
- DD-based optimization is most competitive when…
  - Problem has a recursive **dynamic programming** model…
  - and no convenient IP model.
- Such as…
  - Sequencing and scheduling problems
  - DP problems with exponential state space
    - New approach to "curse of dimensionality"
  - Problems with nonconvex, nonseparable objective function…

Modeling the Objective Function

- Weighted DD can represent **any** objective function
  - Separable functions are the easiest, but any nonseparable function is possible.
  - Can be nonlinear, nonconvex, etc.
  - The issue is complexity of resulting DD

Modeling the Objective Function

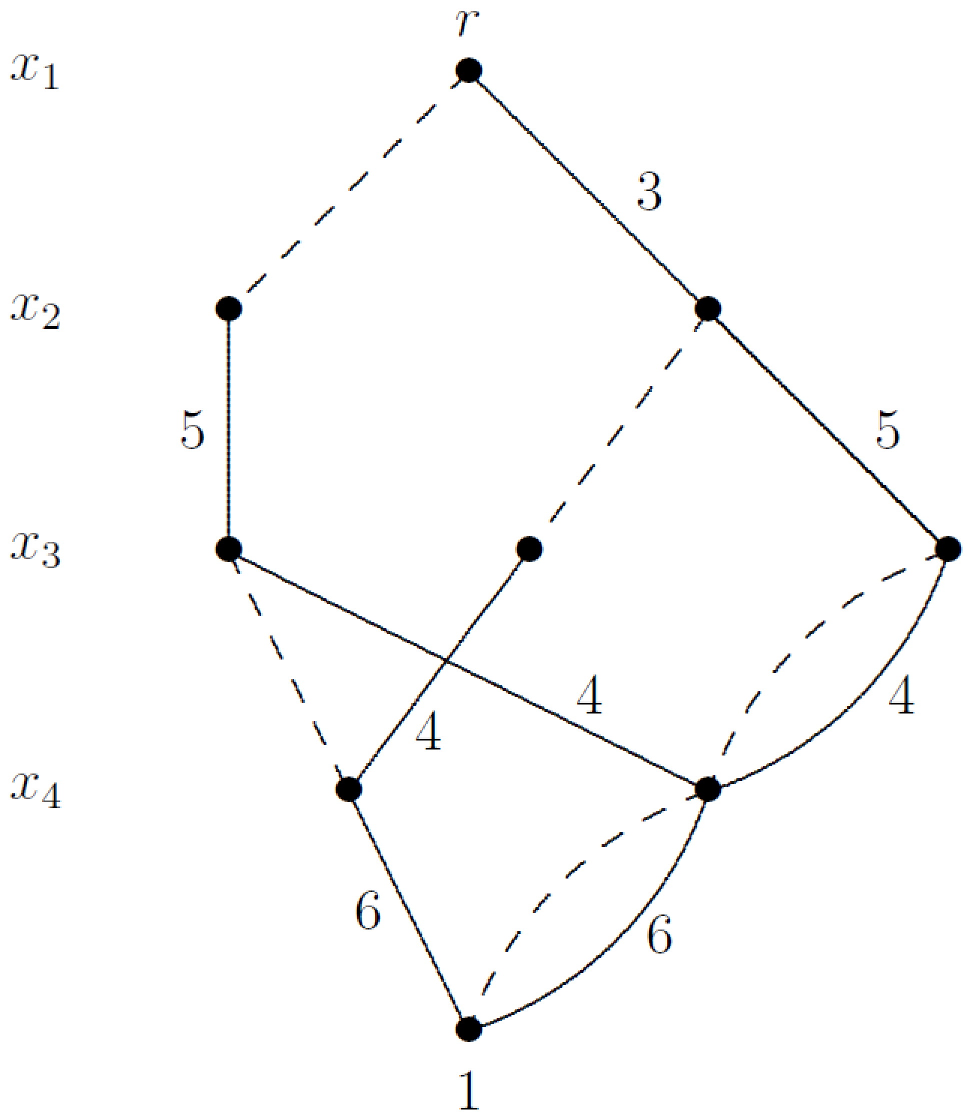
- Weighted DD can represent **any** objective function
  - Separable functions are the easiest, but any nonseparable function is possible.
  - Can be nonlinear, nonconvex, etc.
  - The issue is complexity of resulting DD
- Multiple encodings
  - A given objective function can be encoded by **multiple** assignments of costs to arcs.
  - There is a **unique canonical** arc cost assignment.
    - Which can **reduce size** of exact DD.
  - Design state variables accordingly

Modeling the Objective Function

Set covering with separable cost function

Easy. Just label arcs with weights.

	Set i			
	1	2	3	4
A	●	●		
B	●		●	●
C		●	●	
D		●		●
Weight	3	5	4	6



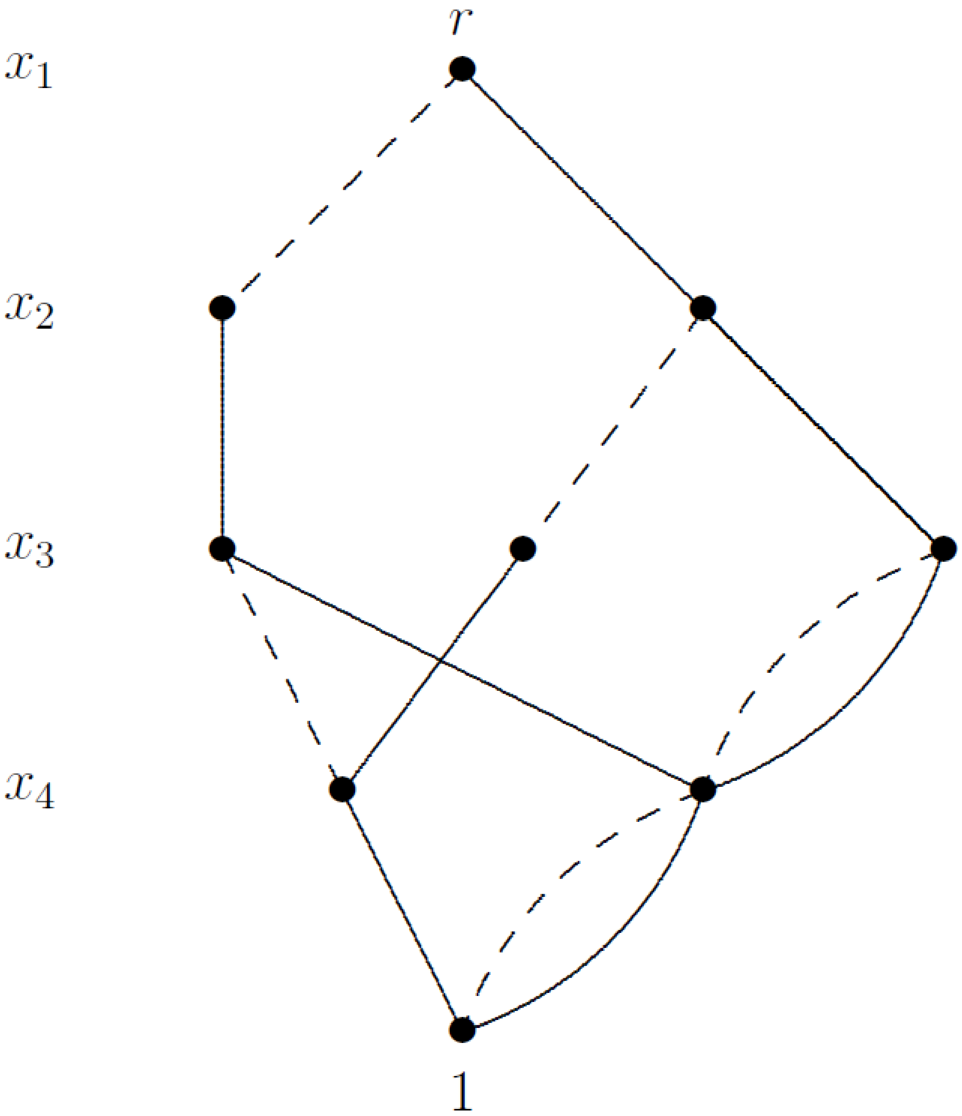
$x_i = 1$ when we select set i

Modeling the Objective Function

Nonseparable cost function

Now what?

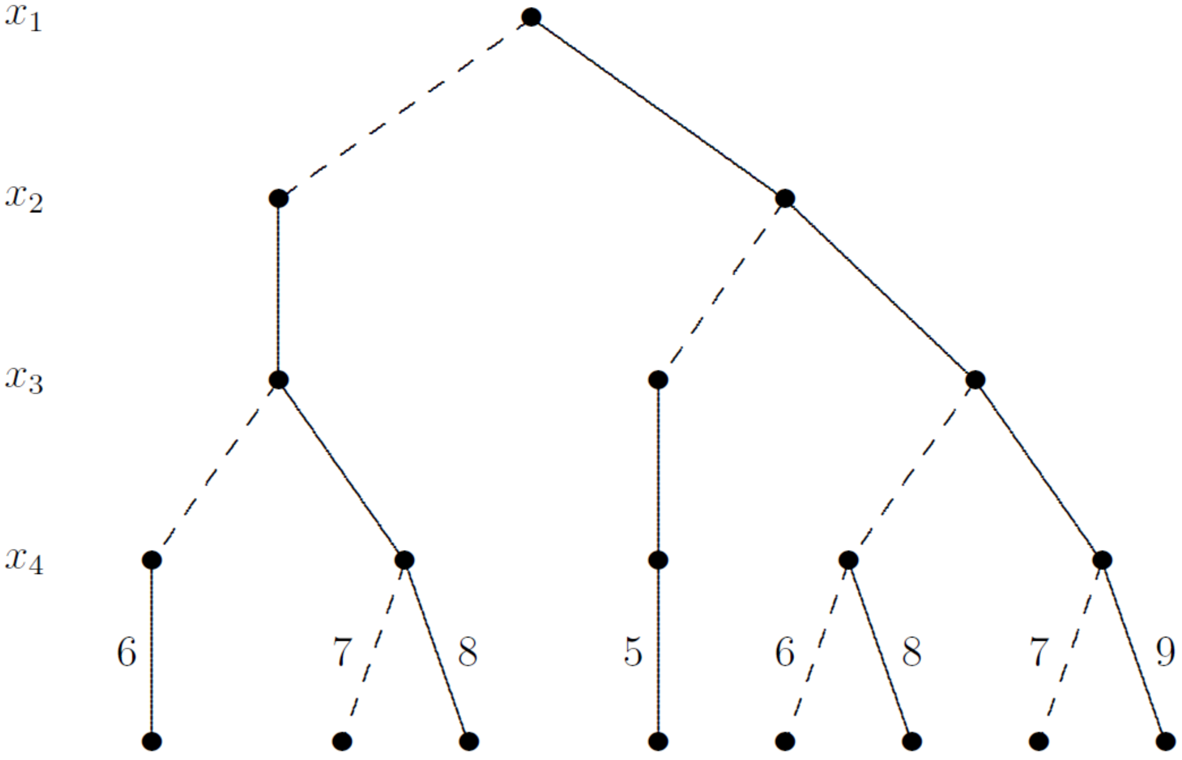
x	$f(x)$
(0,1,0,1)	6
(0,1,1,0)	7
(0,1,1,1)	8
(1,0,1,1)	5
(1,1,0,0)	6
(1,1,0,1)	8
(1,1,1,0)	7
(1,1,1,1)	9



Modeling the Objective Function

Nonseparable cost function

Put costs on leaves
of branching tree.



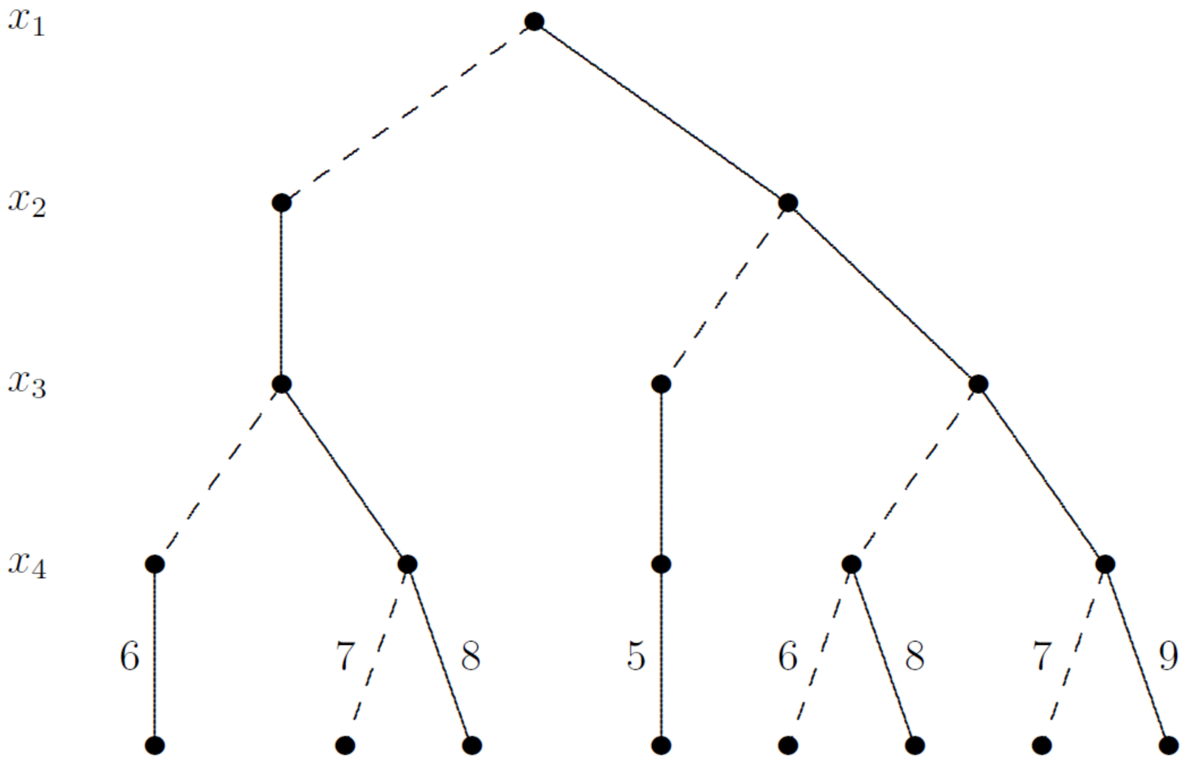
x	$f(x)$
(0,1,0,1)	6
(0,1,1,0)	7
(0,1,1,1)	8
(1,0,1,1)	5
(1,1,0,0)	6
(1,1,0,1)	8
(1,1,1,0)	7
(1,1,1,1)	9

Modeling the Objective Function

Nonseparable cost function

Put costs on leaves of branching tree.

But now we can't reduce the tree to an efficient decision diagram.



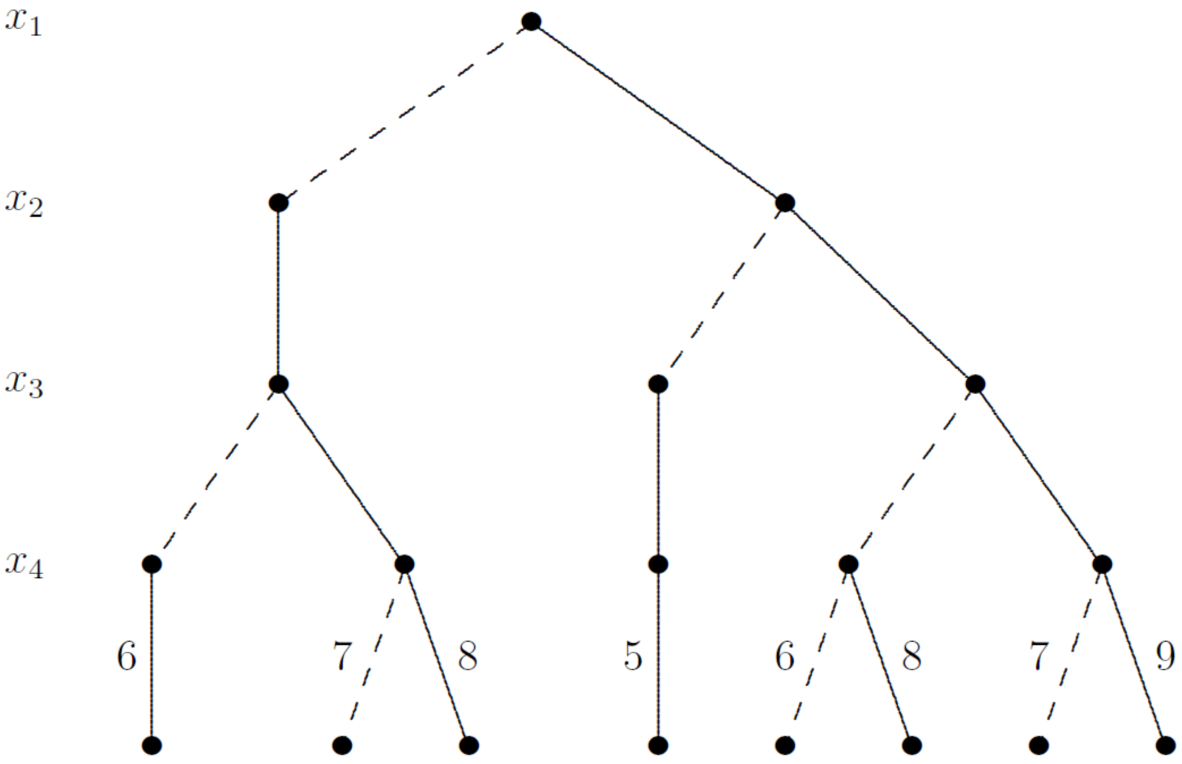
Modeling the Objective Function

Nonseparable cost function

Put costs on leaves of branching tree.

But now we can't reduce the tree to an efficient decision diagram.

We will rearrange costs to obtain **canonical costs**.



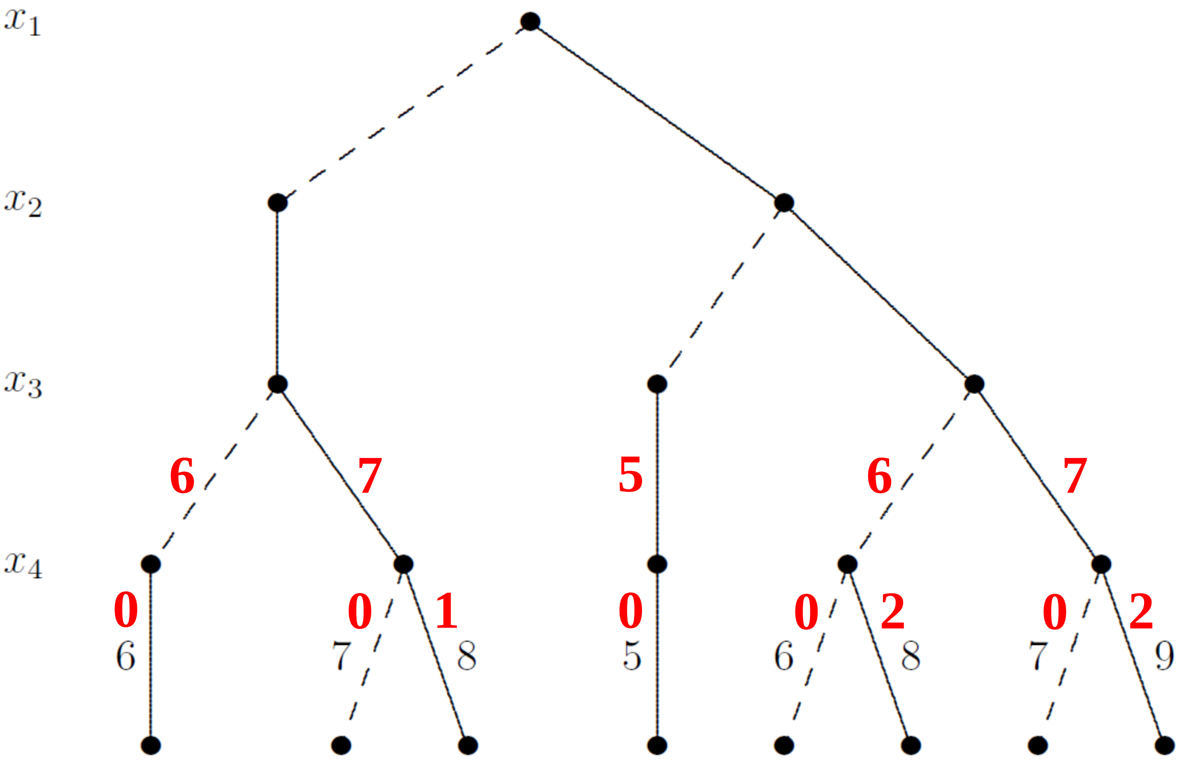
Modeling the Objective Function

Nonseparable cost function

Put costs on leaves of branching tree.

But now we can't reduce the tree to an efficient decision diagram.

We will rearrange costs to obtain **canonical costs**.



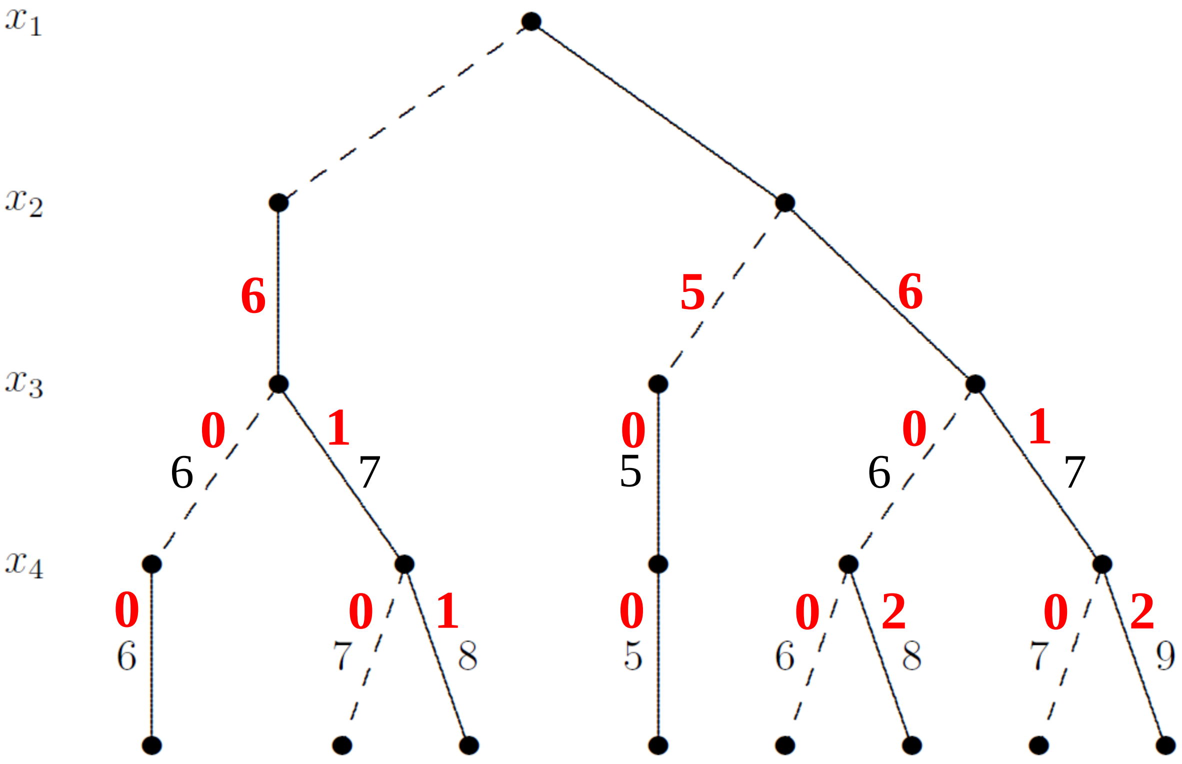
Modeling the Objective Function

Nonseparable cost function

Put costs on leaves of branching tree.

But now we can't reduce the tree to an efficient decision diagram.

We will rearrange costs to obtain **canonical costs**.



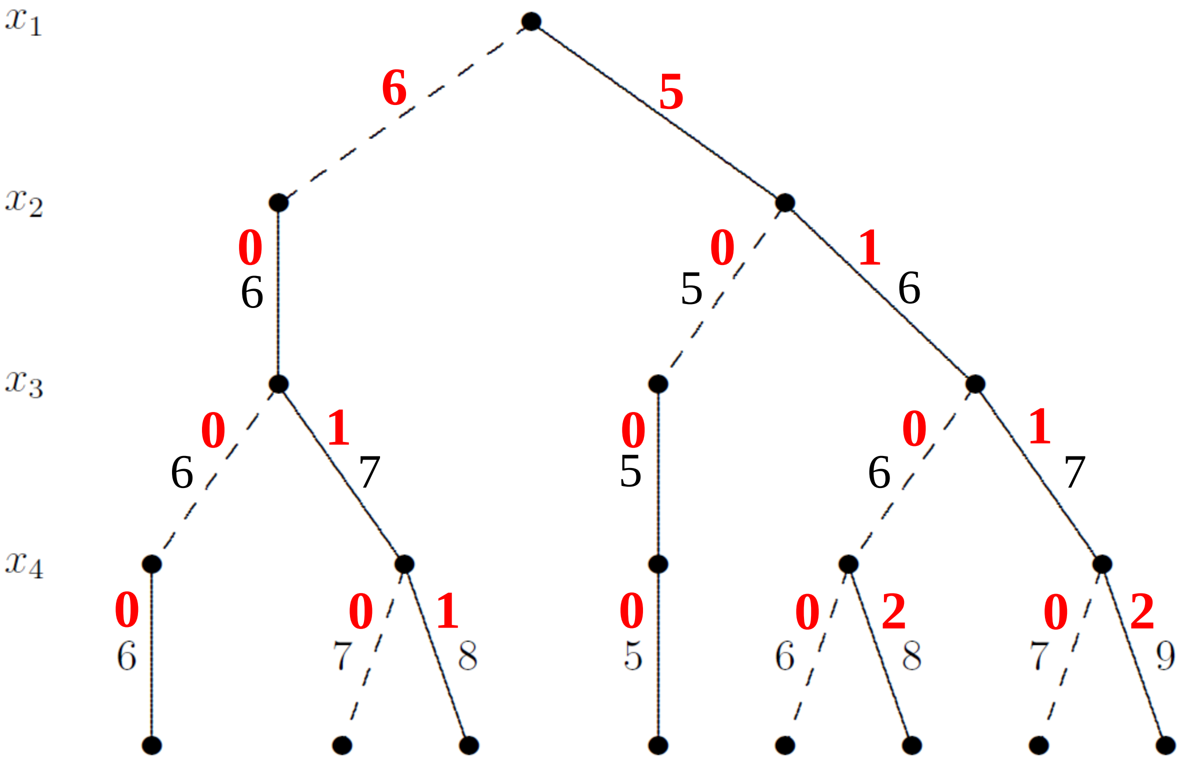
Modeling the Objective Function

Nonseparable cost function

Put costs on leaves of branching tree.

But now we can't reduce the tree to an efficient decision diagram.

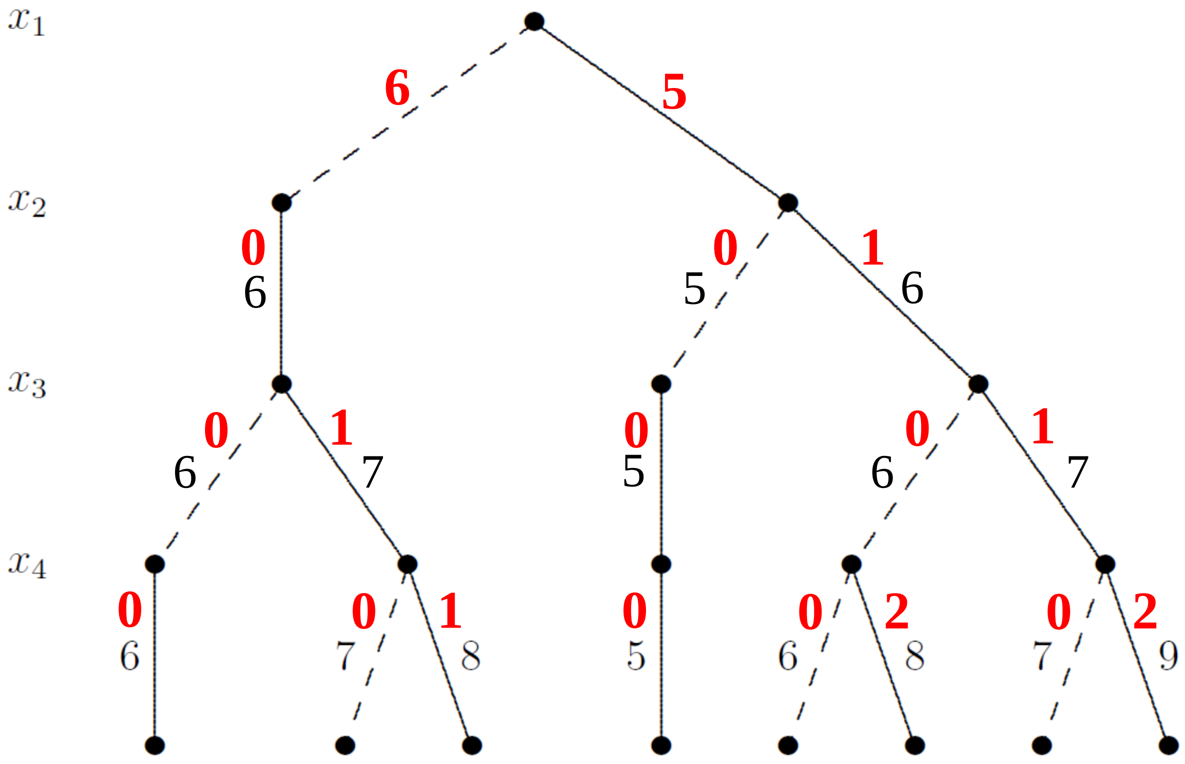
We will rearrange costs to obtain **canonical costs**.



Modeling the Objective Function

Nonseparable cost function

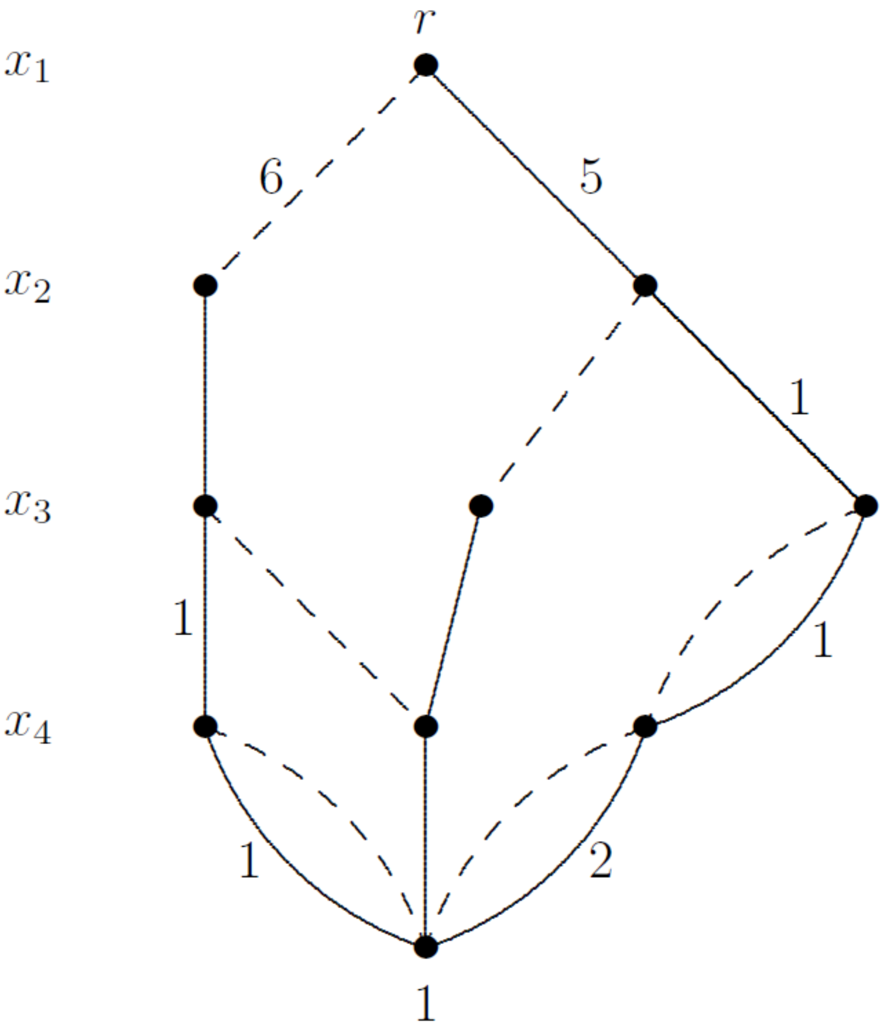
Now the tree can be reduced.



Modeling the Objective Function

Nonseparable cost function

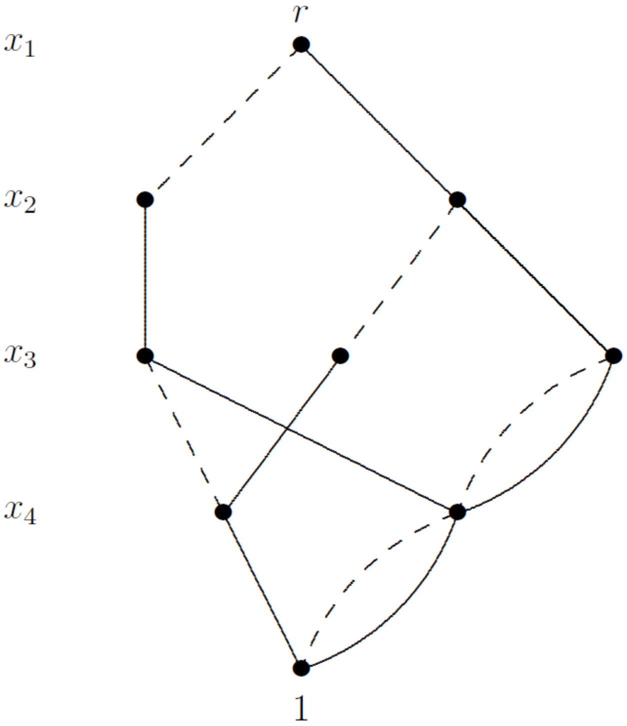
Now the tree can be reduced.

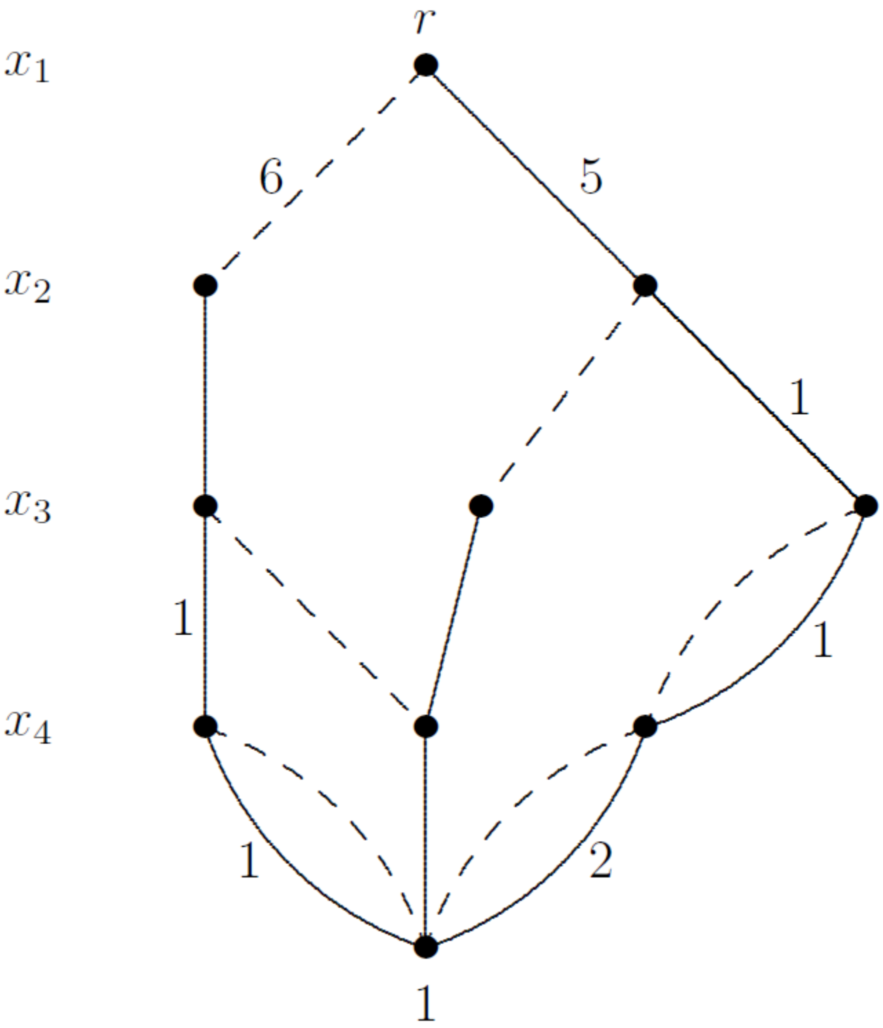


Modeling the Objective Function

Nonseparable cost function

DD is larger than
reduced unweighted DD,
but still compact.





Modeling the Objective Function

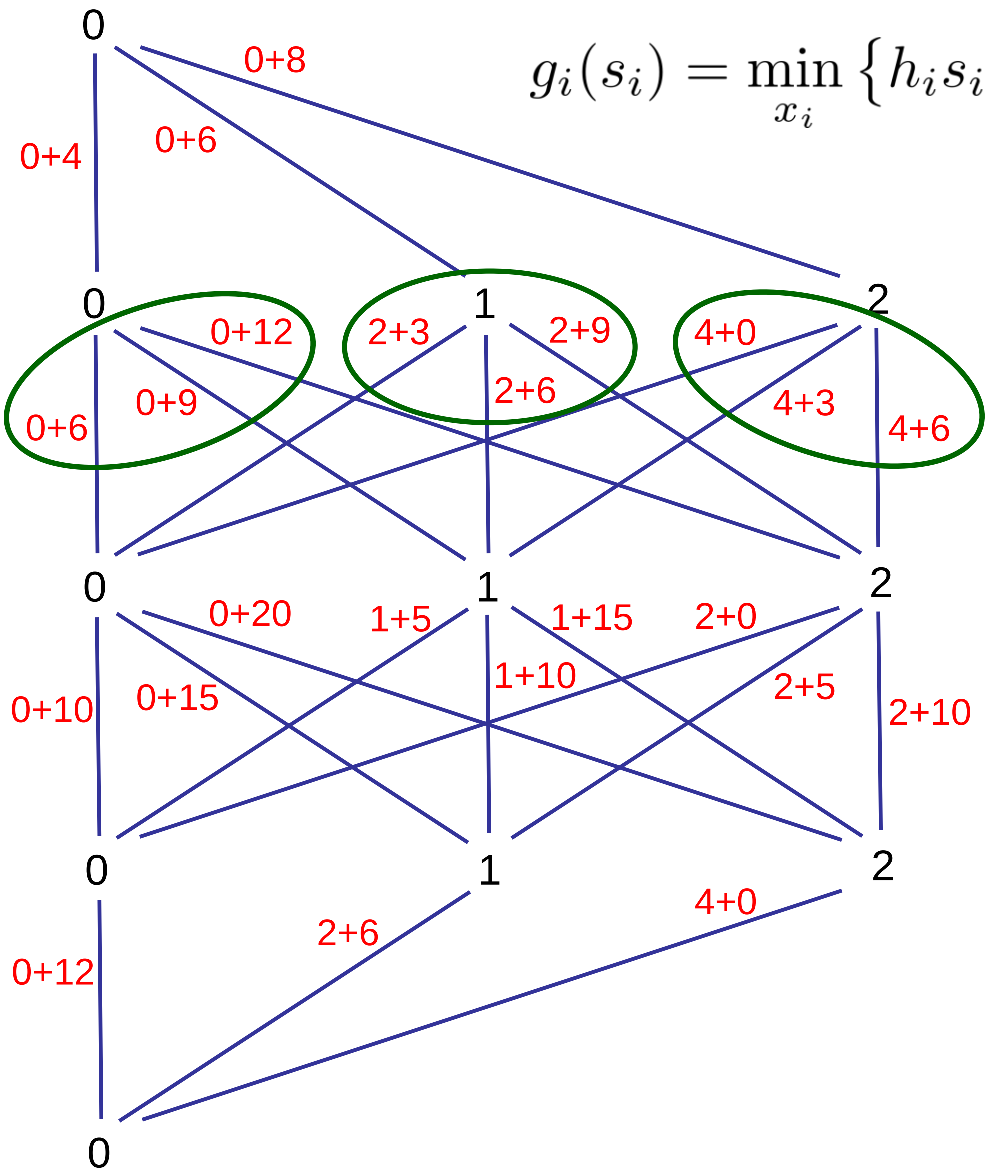
Theorem. For a given variable ordering, a given objective function is represented by a **unique** weighted decision diagram with canonical costs.

JH (2013),
Similar result for AADDs:
Sanner & McAllester (2005)

Inventory Management Example

- In each period i, we have:
  - Demand d_i
  - Unit production cost c_i
  - Warehouse space m
  - Unit holding cost h_i
- In each period, we decide:
  - Production level x_i
  - Stock level s_i
- Objective:
  - Meet demand each period while minimizing production and holding costs.

Reducing the Transition Graph

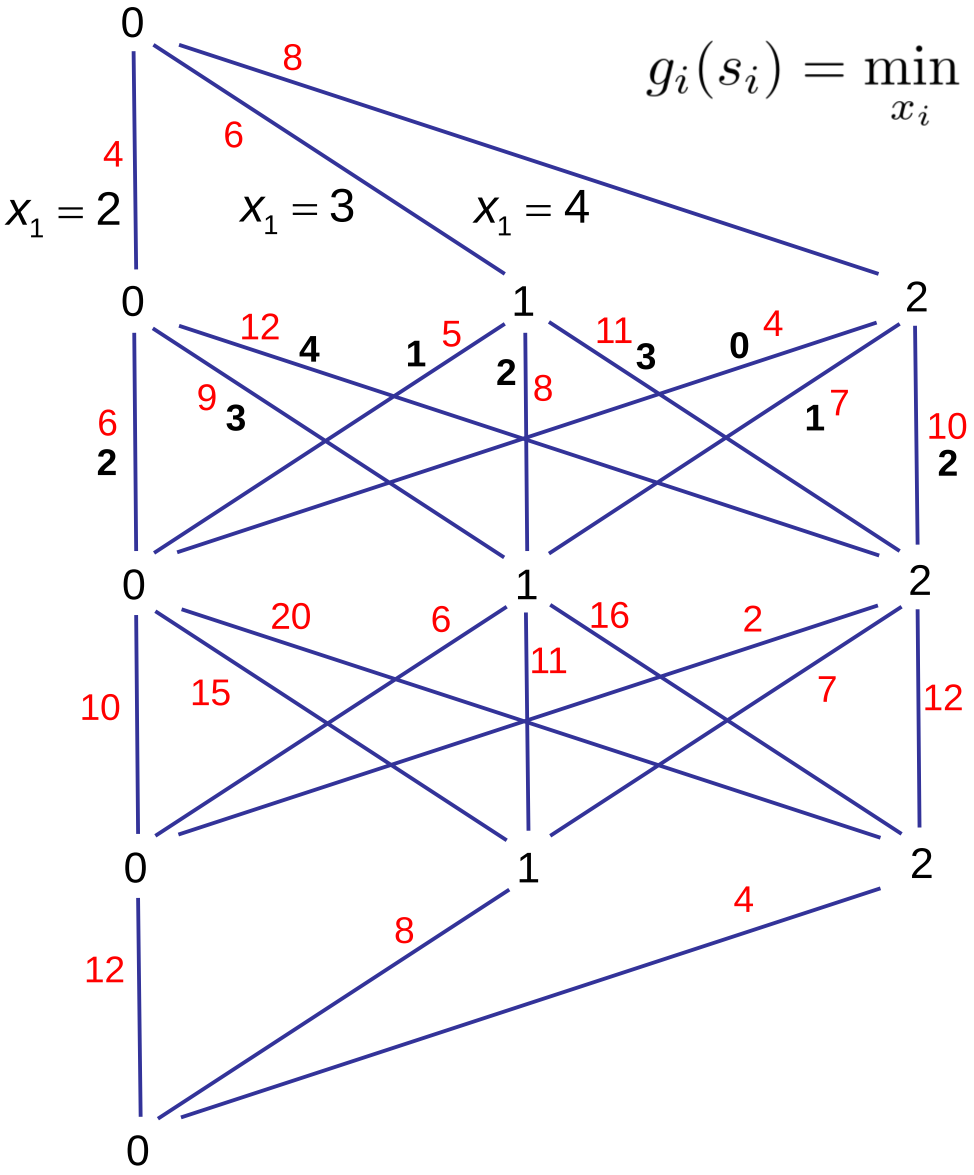


$$g_i(s_i) = \min_{x_i}\left\{h_i s_i + c_i x_i + g_{i+1}(s_i + x_i - d_i)\right\}$$

Arcs leaving each node are very similar.

- Transition to the same states.
- Have the same costs, up to an offset.

Inventory Problem



$$g_i(s_i) = \min_{x_i} \left\{ h_i s_i + c_i x_i + g_{i+1}(s_i + x_i - d_i) \right\}$$

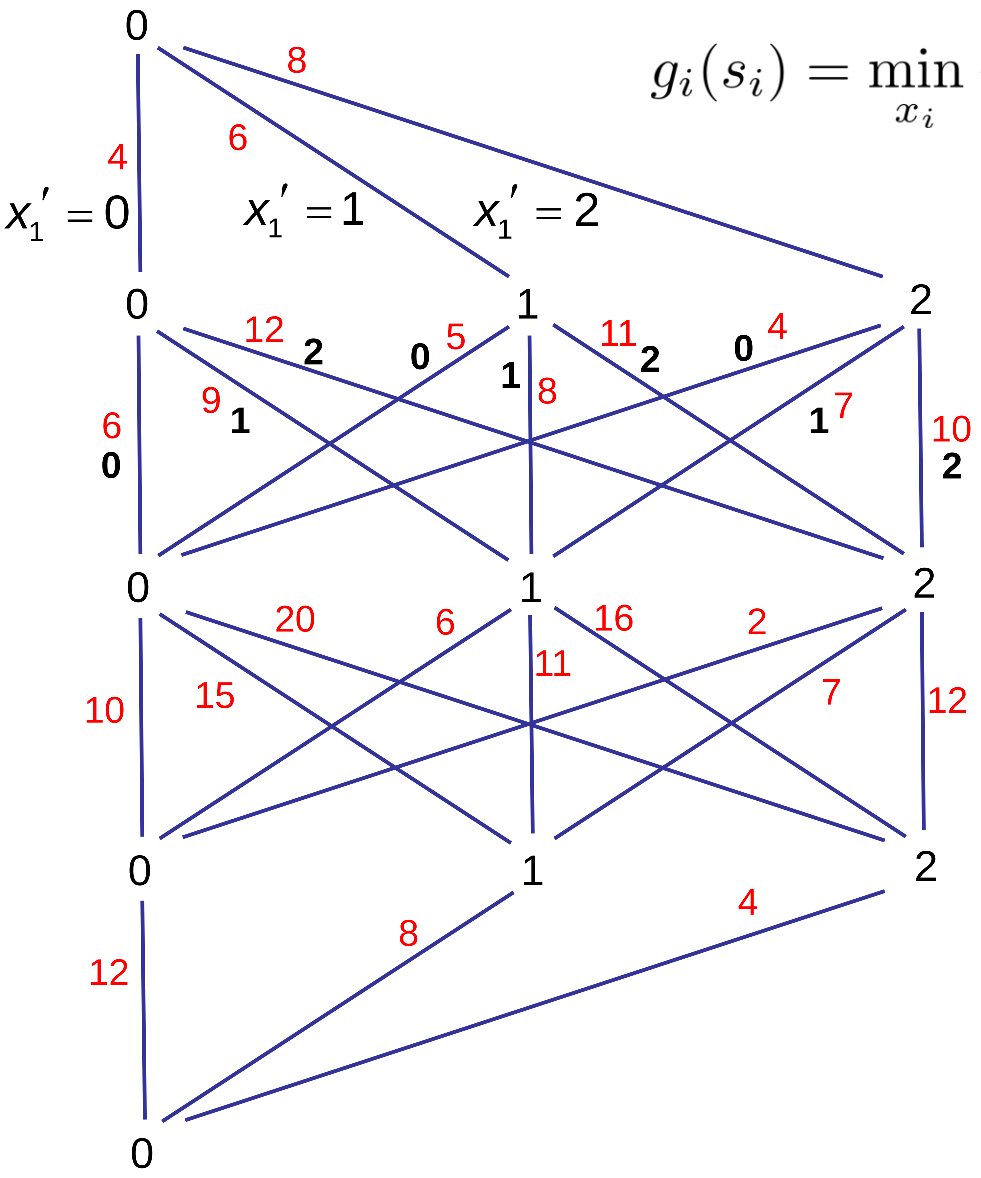
To equalize controls, let

$$x_i' = s_i + x_i - d_i$$

be the stock level in next period.

Inventory Problem

$$g_i(s_i) = \min_{x_i} \left\{ h_i s_i + c_i x_i + g_{i+1}(s_i + x_i - d_i) \right\}$$

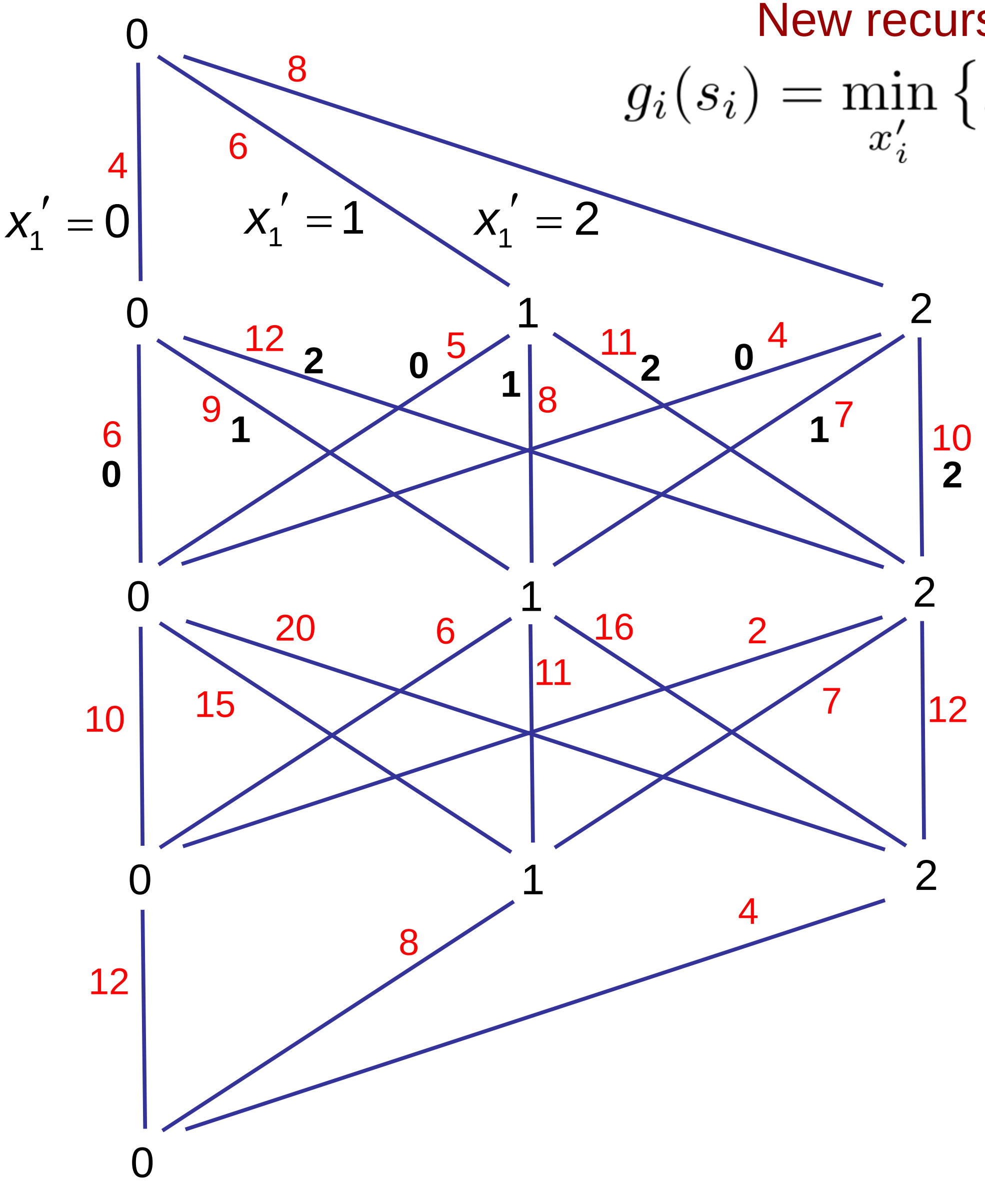


To equalize controls, let

$$x_i' = s_i + x_i - d_i$$

Be the stock level in next period.

Inventory Problem



New recursion:

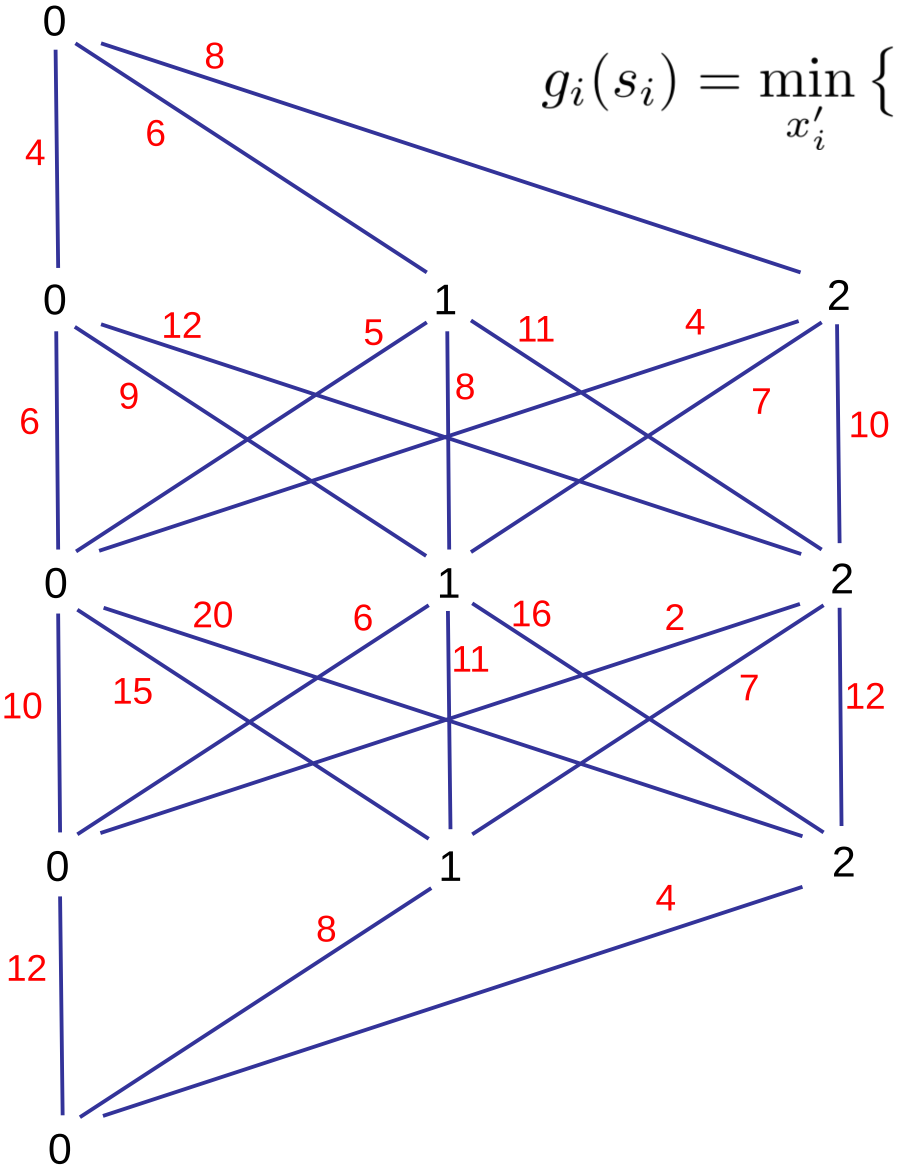
$$g_i(s_i) = \min_{x_i'} \left\{ h_i s_i + c_i(x_i' - s_i + d_i) + g_{i+1}(x_i') \right\}$$

To equalize controls, let

$$x_i' = s_i + x_i - d_i$$

Be the stock level in next period.

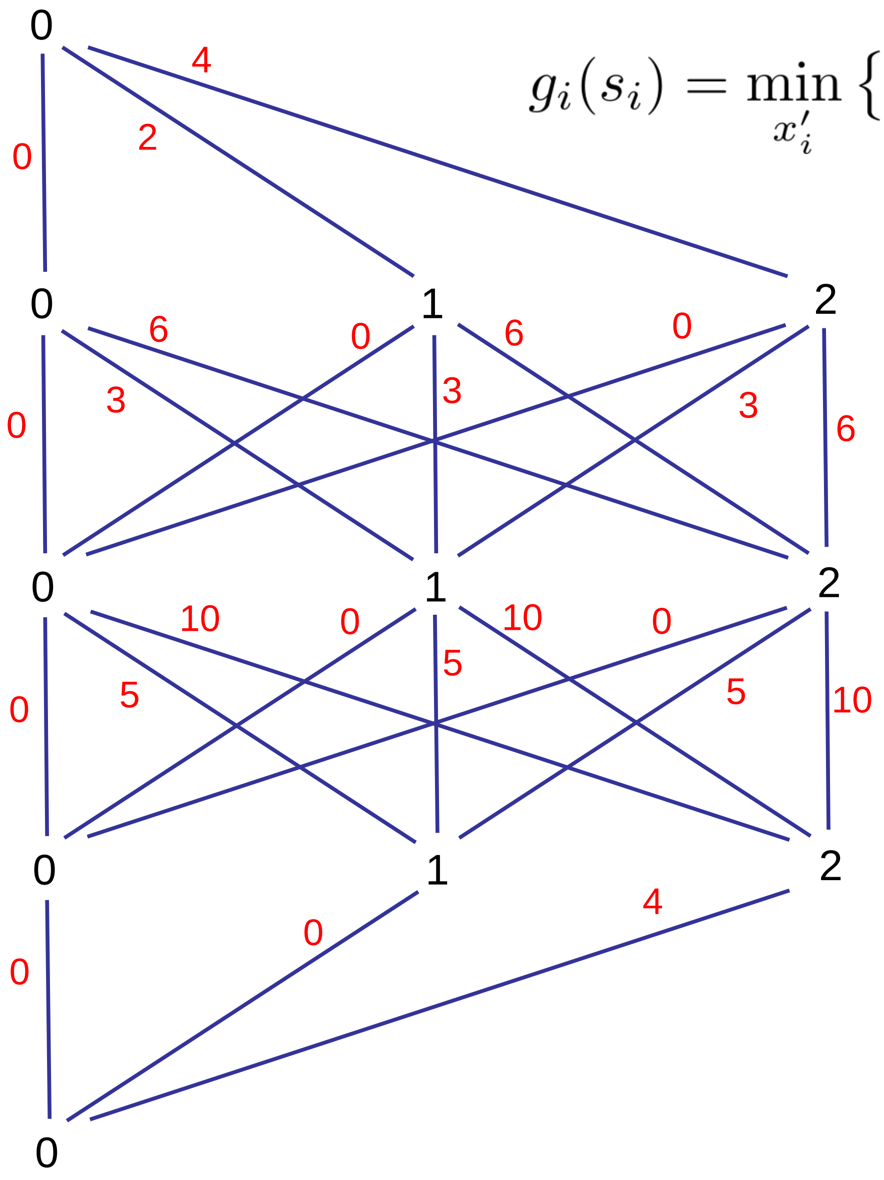
Inventory Problem



$$g_i(s_i) = \min_{x'_i} \left\{ h_i s_i + c_i(x'_i - s_i + d_i) + g_{i+1}(x'_i) \right\}$$

To obtain canonical costs, subtract $c_i(m - s_i) + h_i s_i$ from cost on each arc (s_i, s_{i+1}).

Inventory Problem

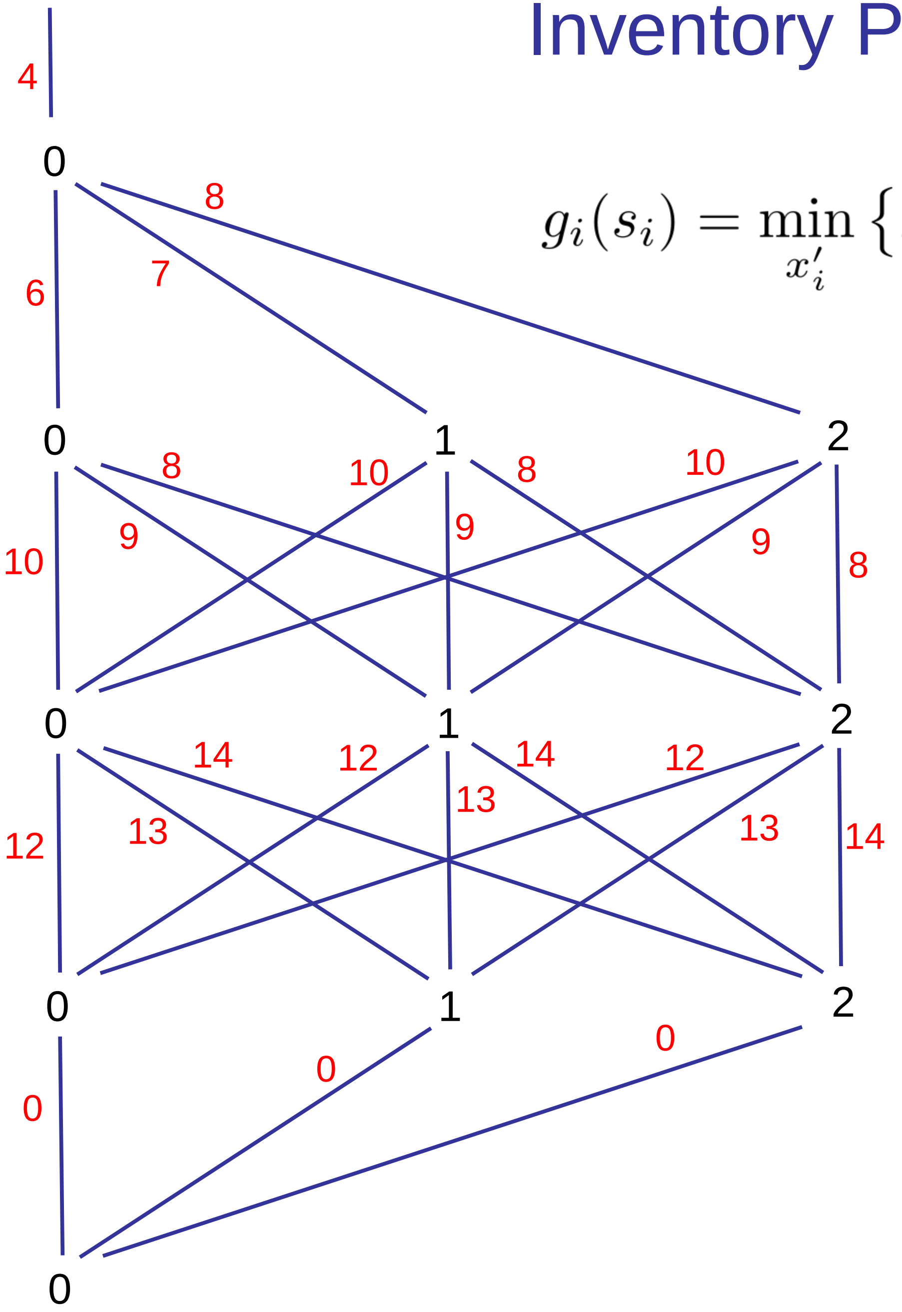


$$g_i(s_i) = \min_{x'_i} \left\{ h_i s_i + c_i(x'_i - s_i + d_i) + g_{i+1}(x'_i) \right\}$$

To obtain canonical costs, subtract $c_i(m - s_i) + h_i s_i$ from cost on each arc (s_i, s_{i+1}).

Add these offsets to incoming arcs.

Inventory Problem

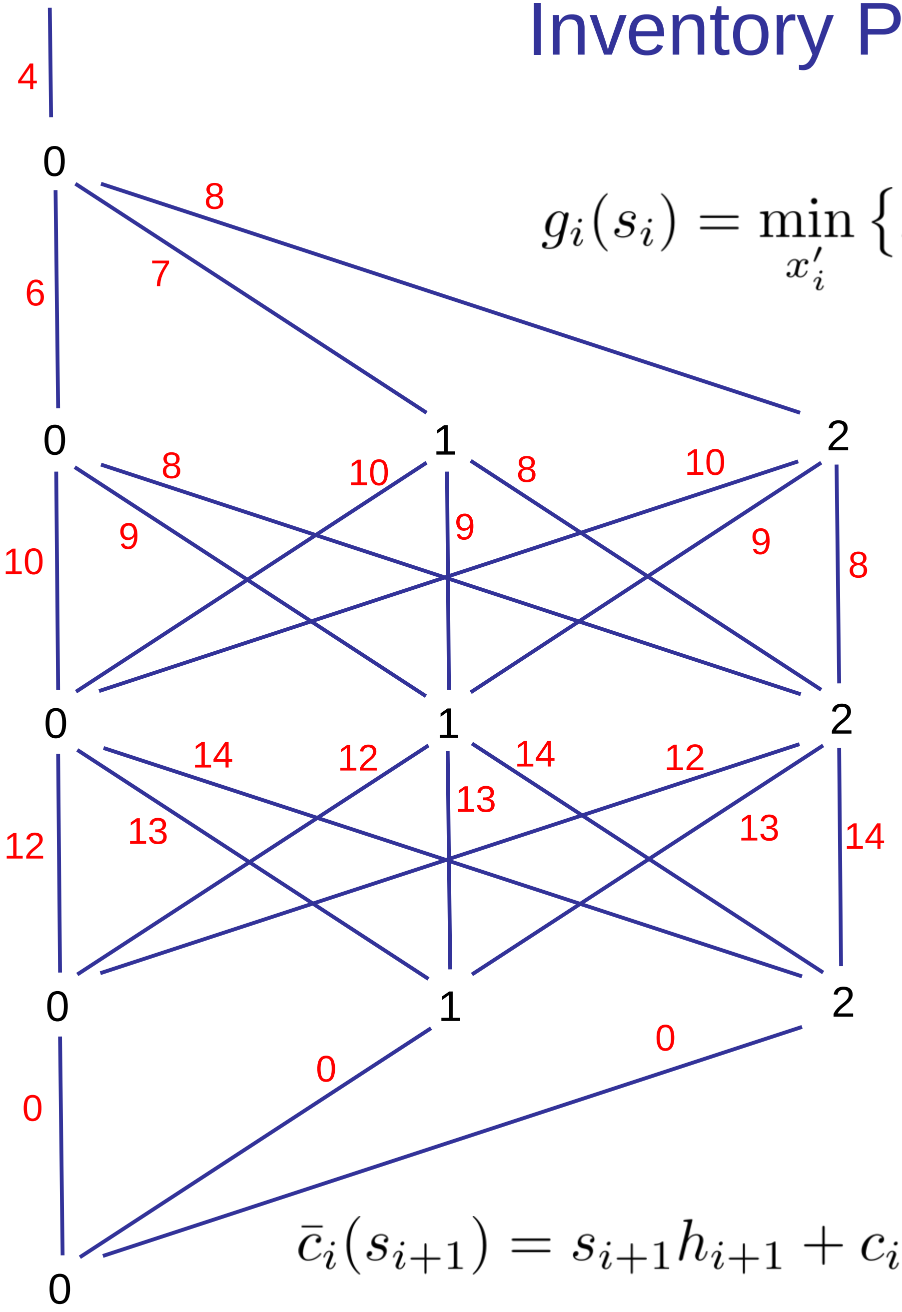


$$g_i(s_i) = \min_{x_i'} \left\{ h_i s_i + c_i(x_i' - s_i + d_i) + g_{i+1}(x_i') \right\}$$

To obtain canonical costs, subtract $c_i(m - s_i) + h_i s_i$ from cost on each arc (s_i,s_{i+1}).

Add these offsets to incoming arcs.

Inventory Problem



$$g_i(s_i) = \min_{x_i'} \left\{h_i s_i + c_i(x_i' - s_i + d_i) + g_{i+1}(x_i')\right\}$$

To obtain canonical costs, subtract $c_i(m - s_i) + h_i s_i$ from cost on each arc (s_i, s_{i+1}).

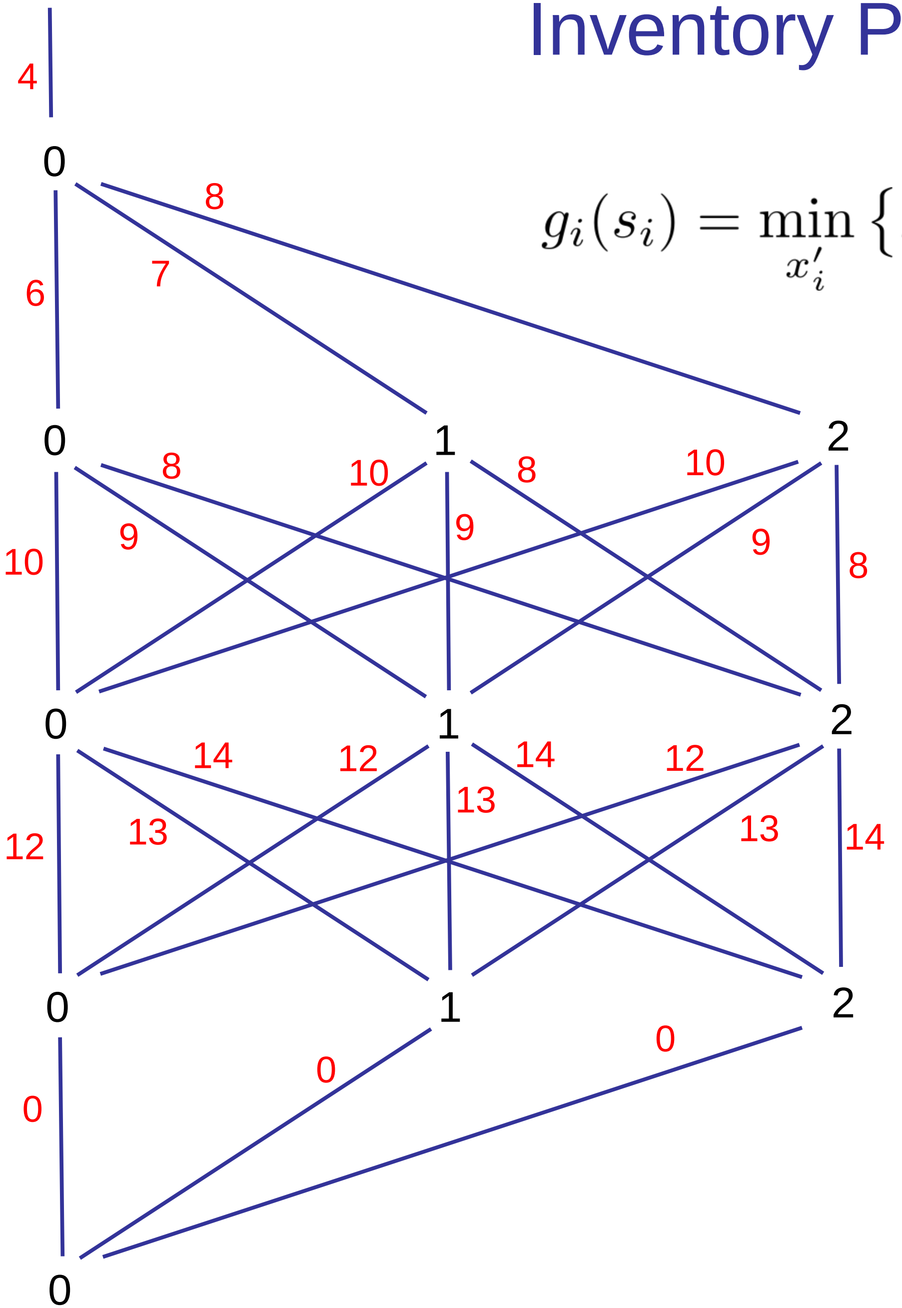
Add these offsets to incoming arcs.

Now outgoing arcs look alike.

And all arcs into state s_i have the same cost

$$\bar{c}_i(s_{i+1}) = s_{i+1}h_{i+1} + c_i(d_i - s_{i+1} - m) + c_{i+1}(m - s_{i+1})$$

Inventory Problem



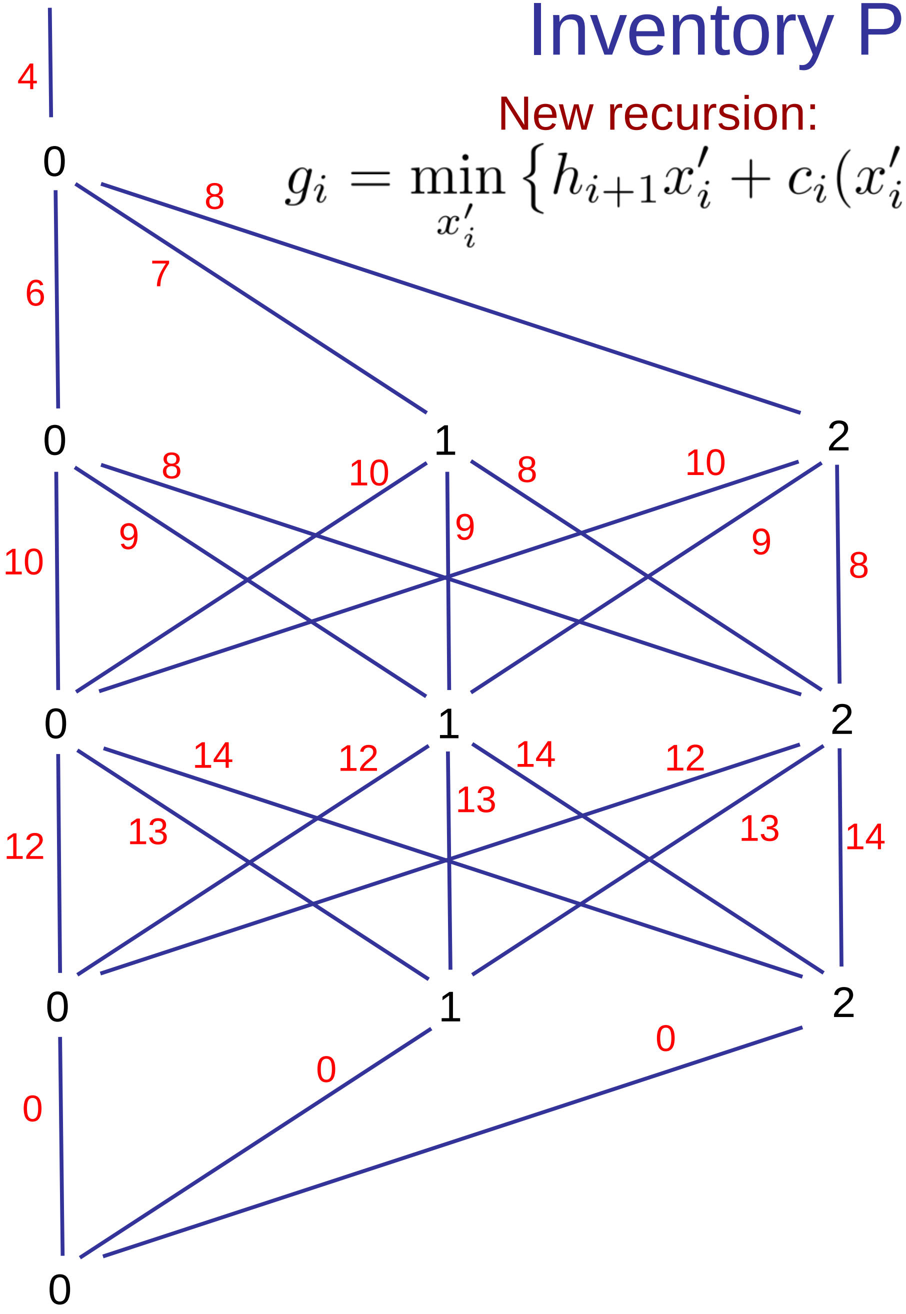
$$g_i(s_i) = \min_{x_i'} \left\{ h_i s_i + c_i(x_i' - s_i + d_i) + g_{i+1}(x_i') \right\}$$

These are canonical costs with offset $\min_{s_{i+1}} \left\{ \bar{c}_i(s_{i+1}) \right\}$

Inventory Problem

New recursion:

$$g_i = \min_{x_i'} \left\{ h_{i+1} x_i' + c_i(x_i' - m + d_i) + c_{i+1}(m - x_i') + g_{i+1} \right\}$$

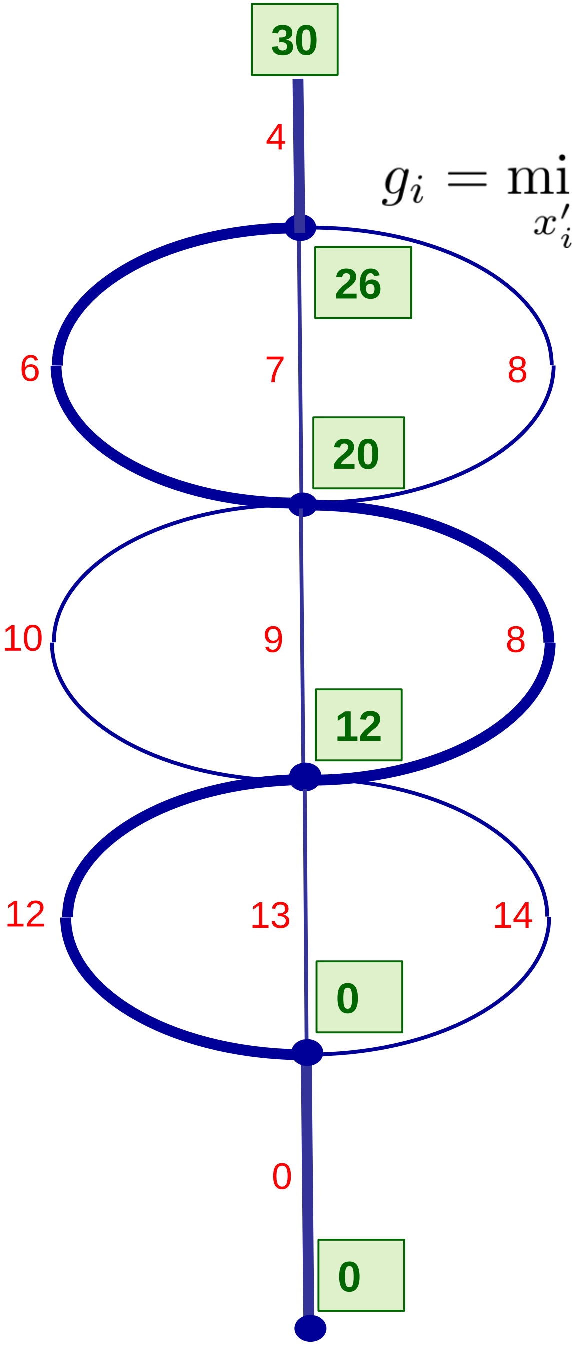


These are canonical costs with offset $\min_{s_{i+1}} \left\{ \bar{c}_i(s_{i+1}) \right\}$

Inventory Problem

New recursion:

$$g_i = \min_{x_i'} \left\{ h_{i+1}x_i' + c_i(x_i' - m + d_i) + c_{i+1}(m - x_i') + g_{i+1} \right\}$$



Now there is only one state per period.

JH (2013)

Nonserial Decision Diagrams

- Analogous to **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

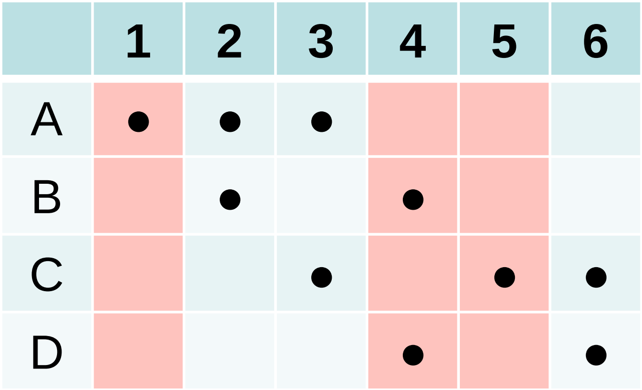
	1	2	3	4	5	6
A	●	●	●			
B		●		●		
C			●		●	●
D				●		●

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				●		●

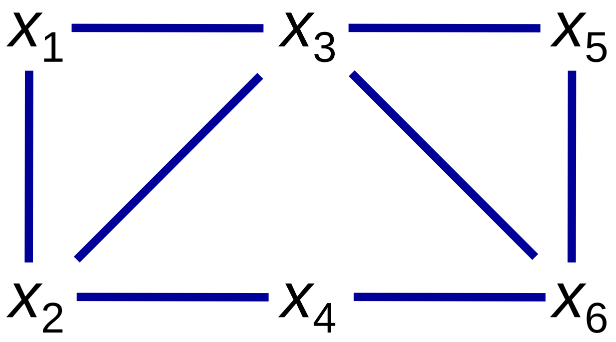
0-1 formulation

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

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

Set Partitioning example

Dependency graph



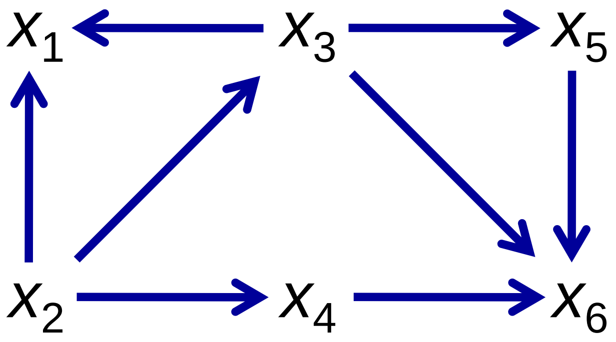
0-1 formulation

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

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

Set Partitioning example

Dependency graph



Enumeration order

x_2

x_3

x_4

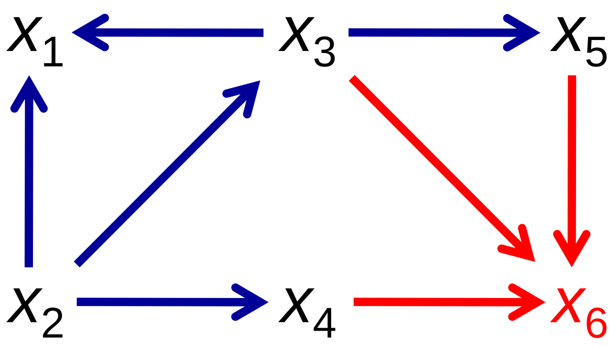
x_1

x_5

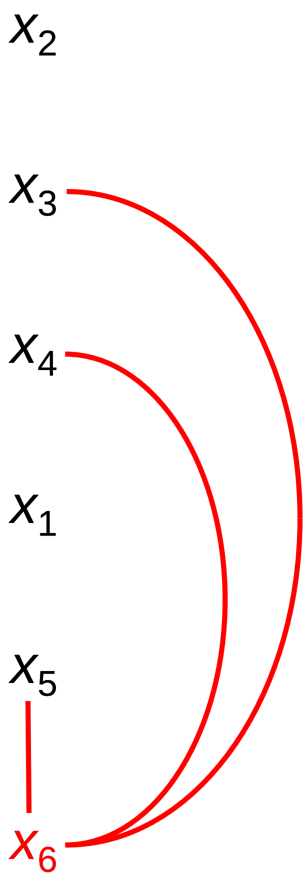
x_6

Set Partitioning example

Dependency graph

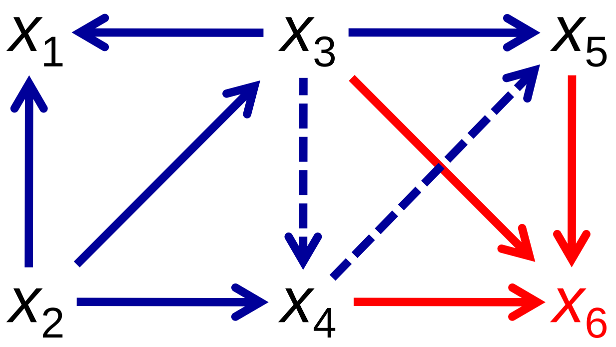


Enumeration order

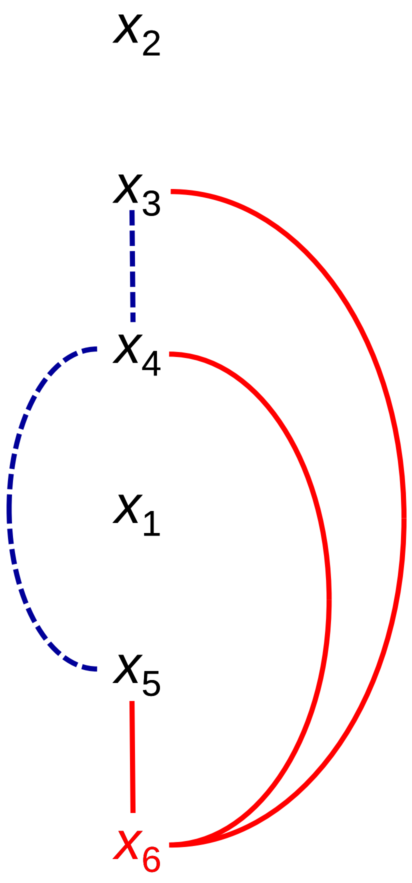


Set Partitioning example

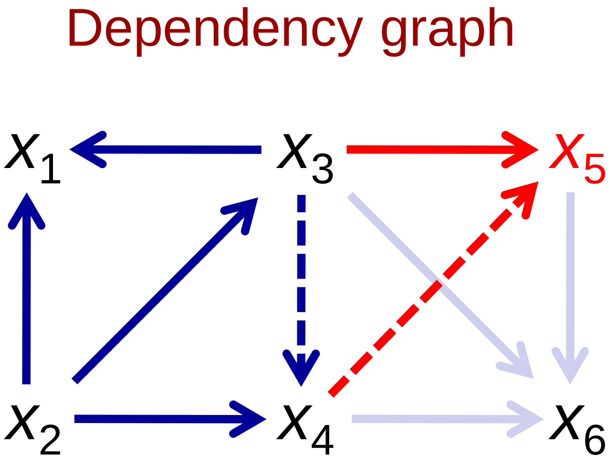
Dependency graph

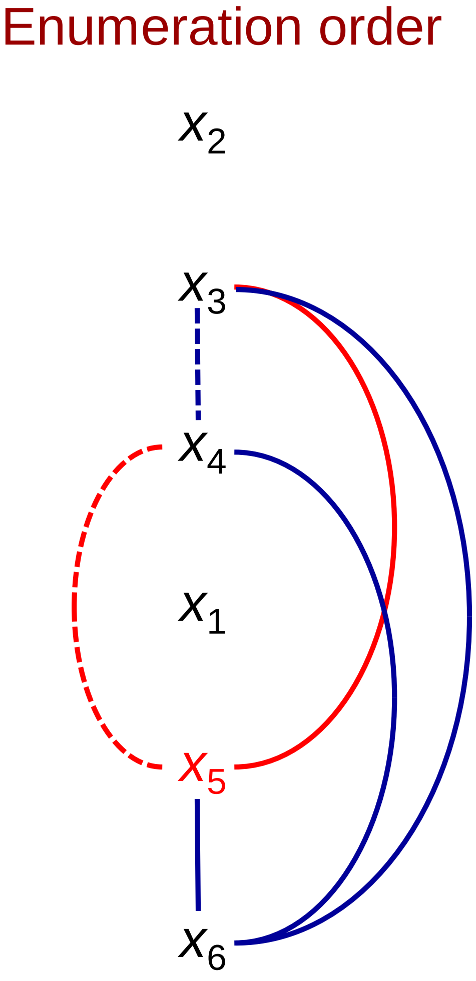


Enumeration order

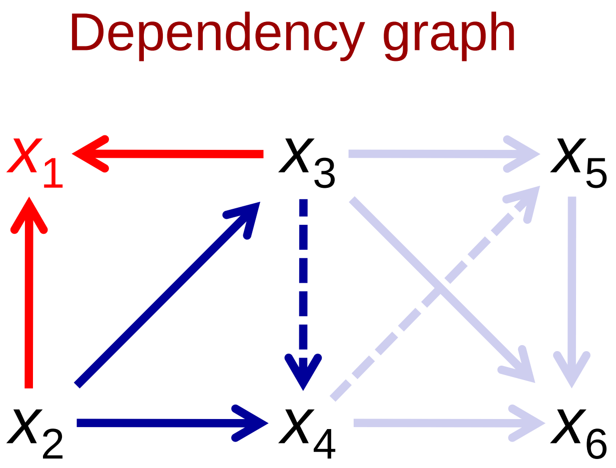


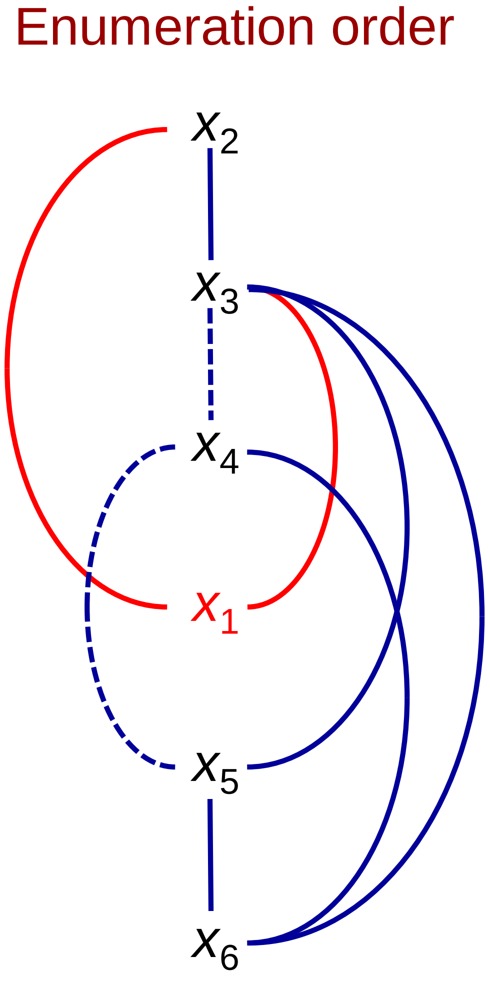
Set Partitioning example



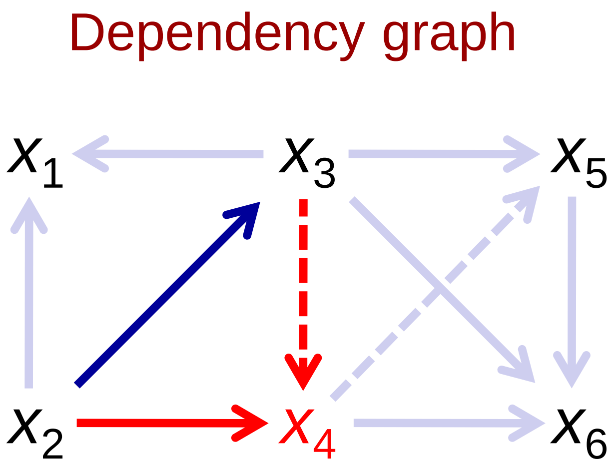


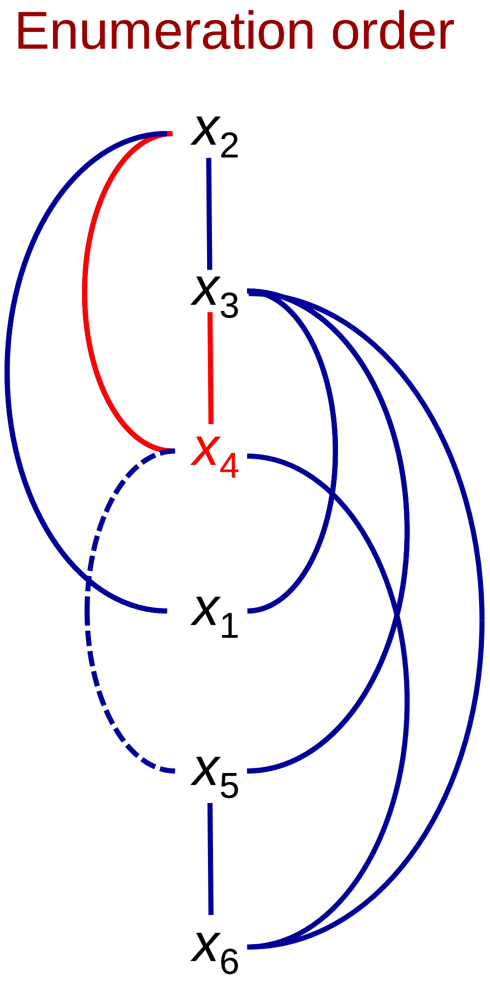
Set Partitioning example





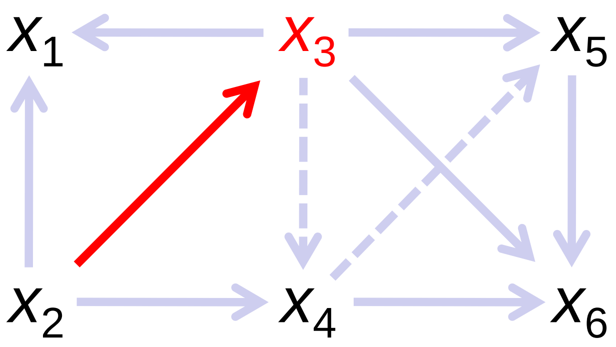
Set Partitioning example



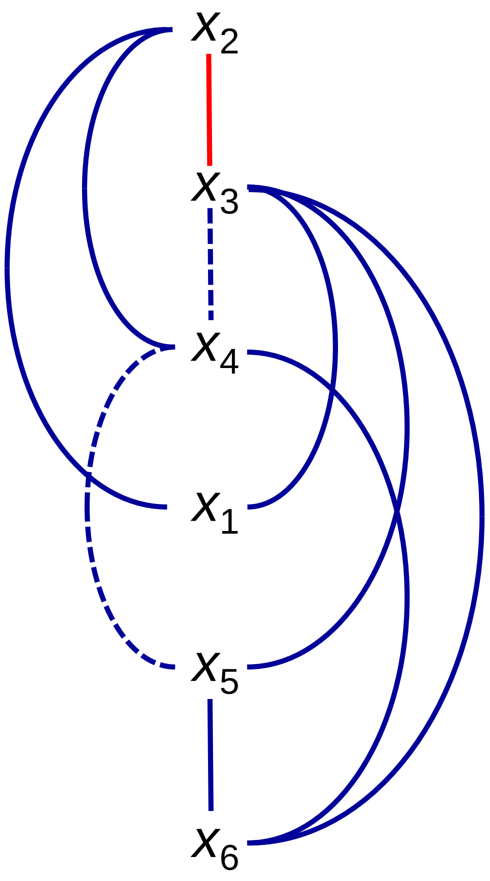


Set Partitioning example

Dependency graph

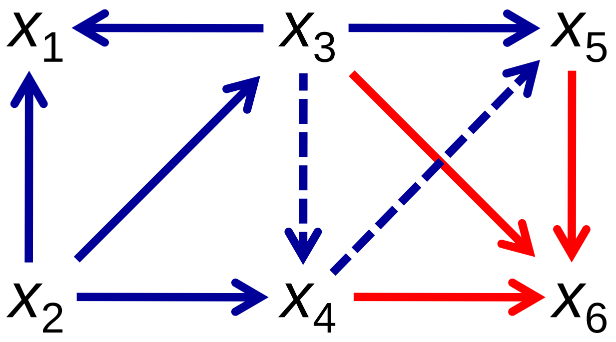


Enumeration order



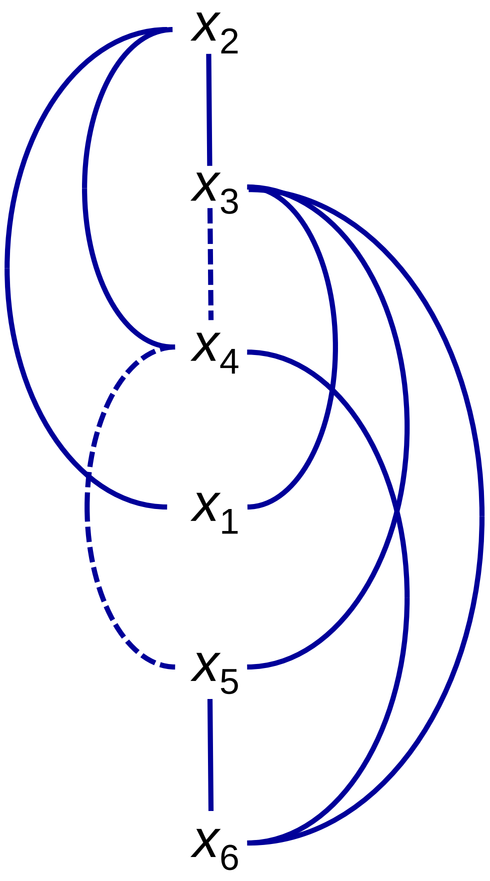
Set Partitioning example

Dependency graph



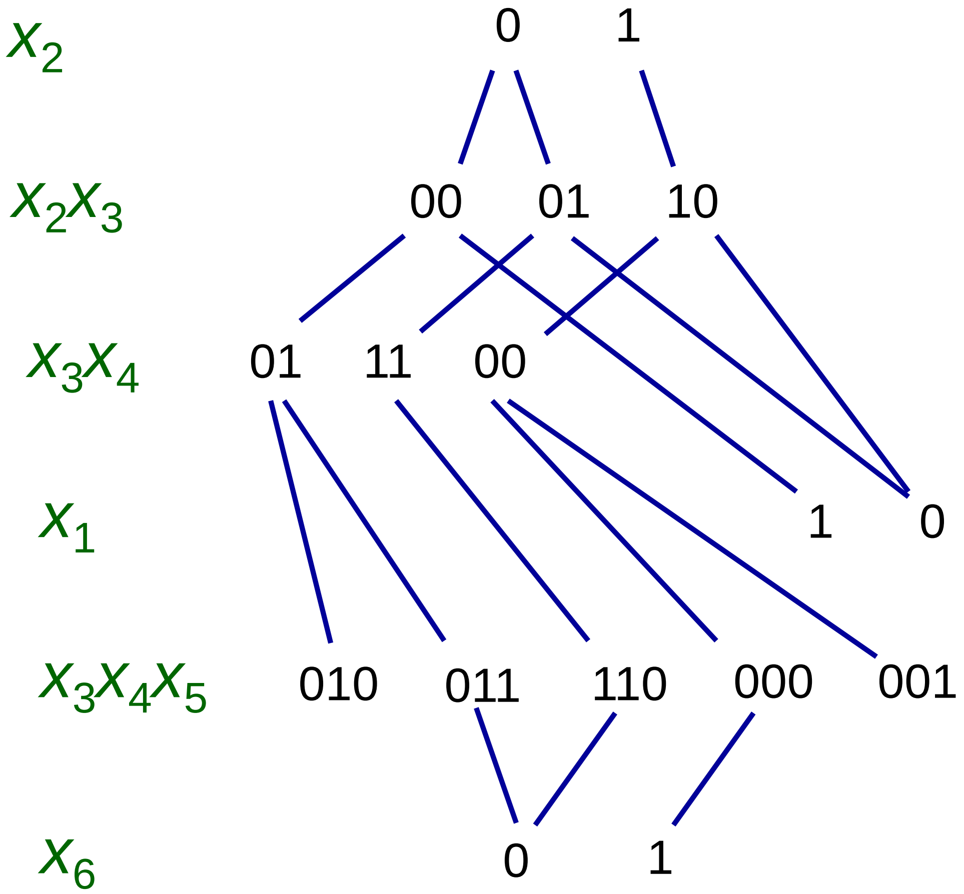
Induced width = 3
(max in-degree)

Enumeration order

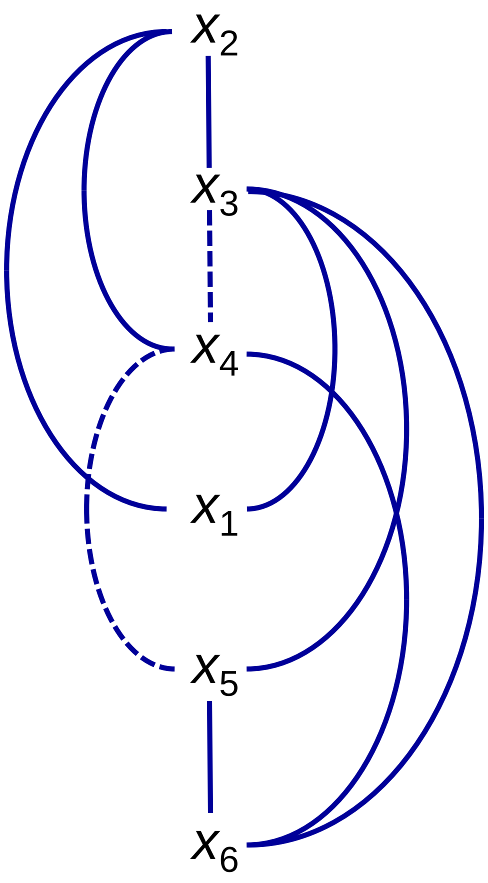


Set Partitioning example

Solution by nonserial DP

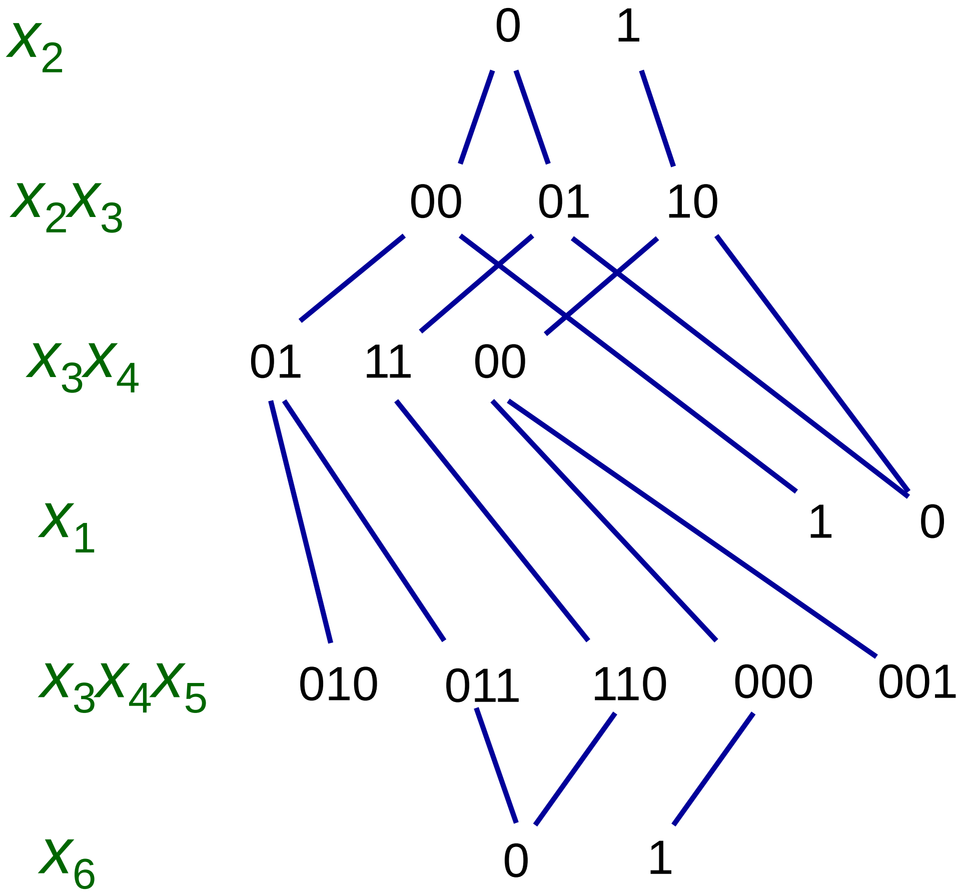


Enumeration order



Set Partitioning example

Solution by nonserial DP

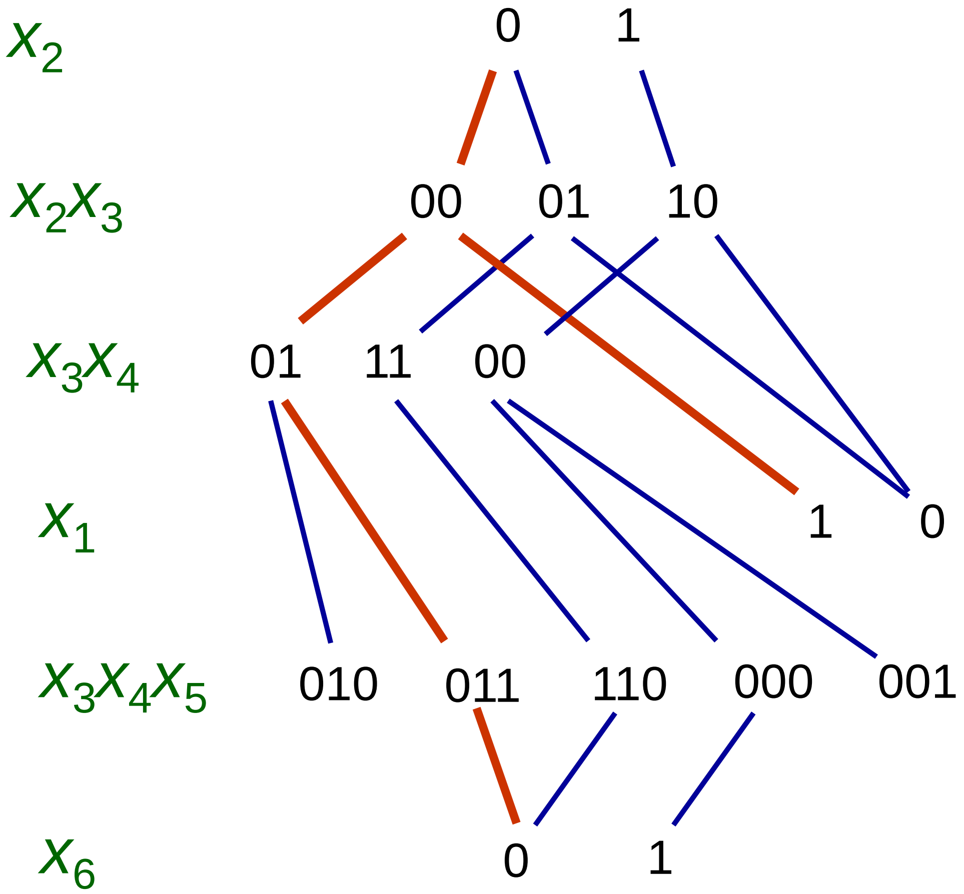


Sets

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

Set Partitioning example

Feasible solution



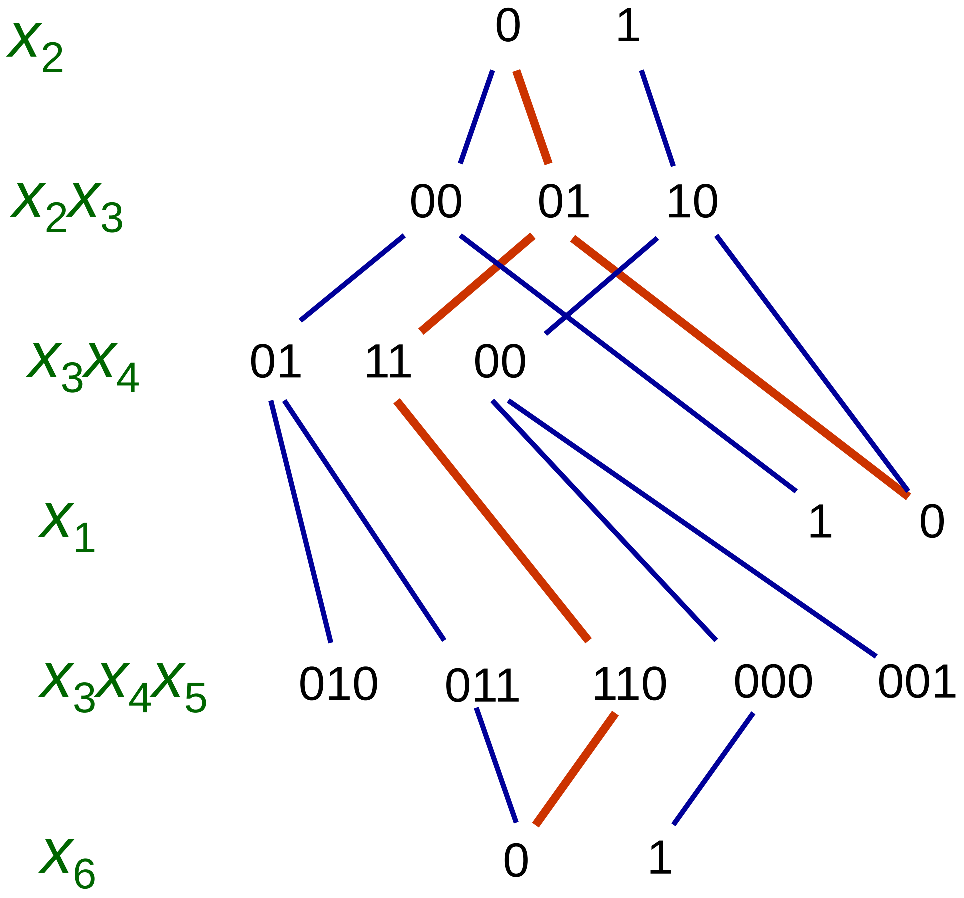
Sets

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

1 0 0 1 1 0

Set Partitioning example

Feasible solution



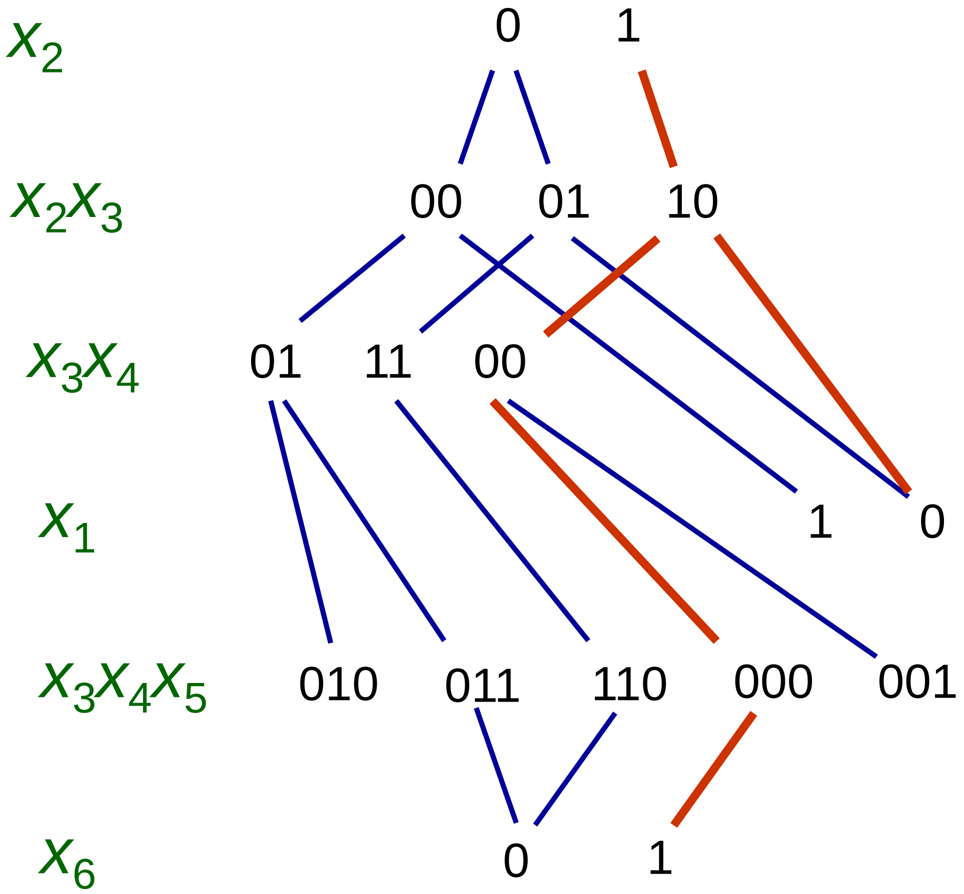
Sets

	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

Feasible solution



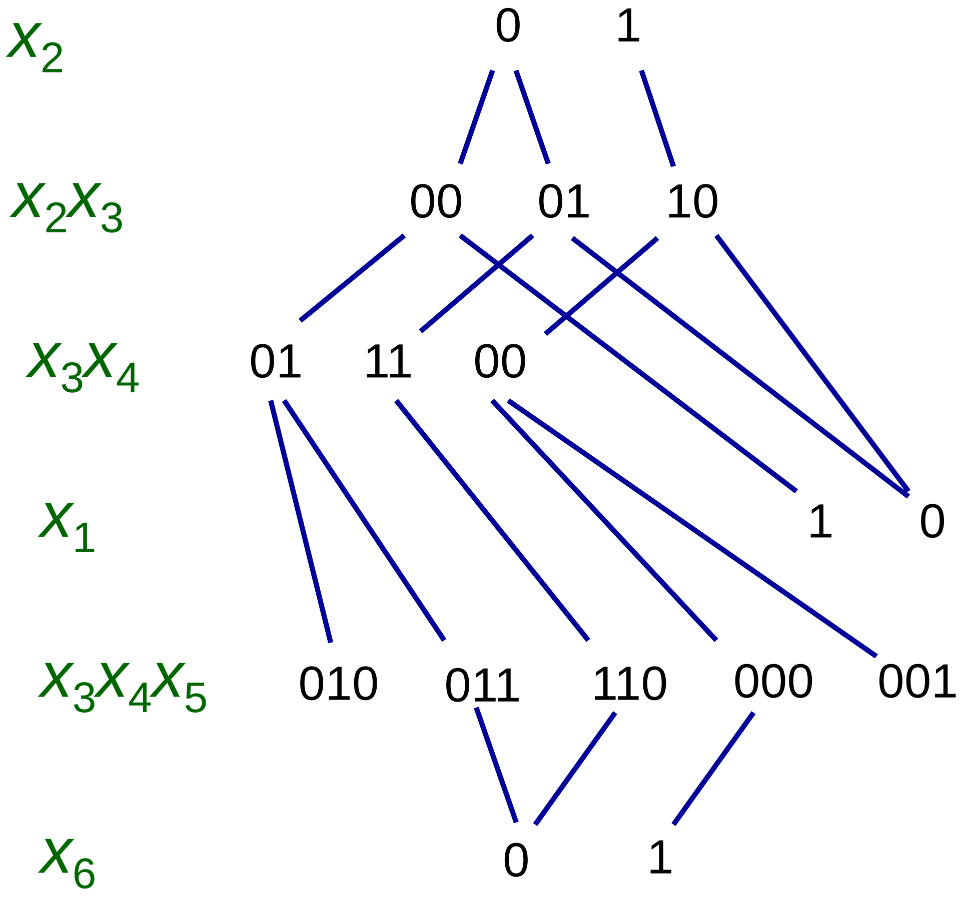
Sets

	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

Solution by nonserial DP



x_2 0 1

x_2x_3 00 01 10

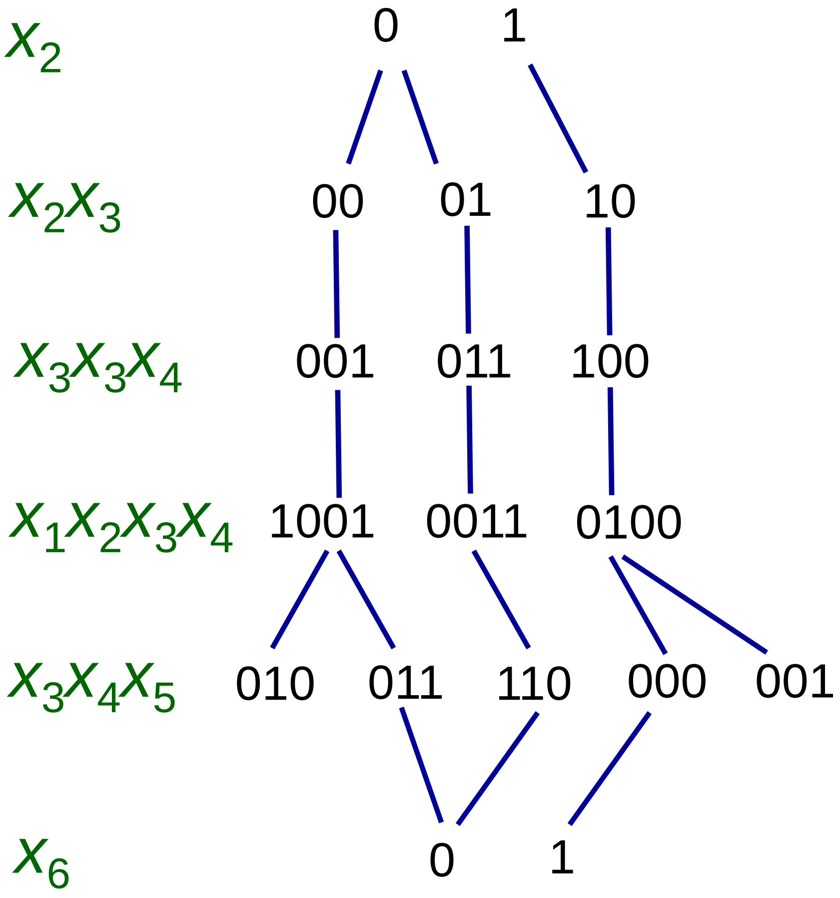
x_3x_4 01 11 00

x_1 1 0

$x_3x_4x_5$ 010 011 110 000 001

x_6 0 1

Serialized DP



x_2 0 1

x_2x_3 00 01 10

$x_3x_3x_4$ 001 011 100

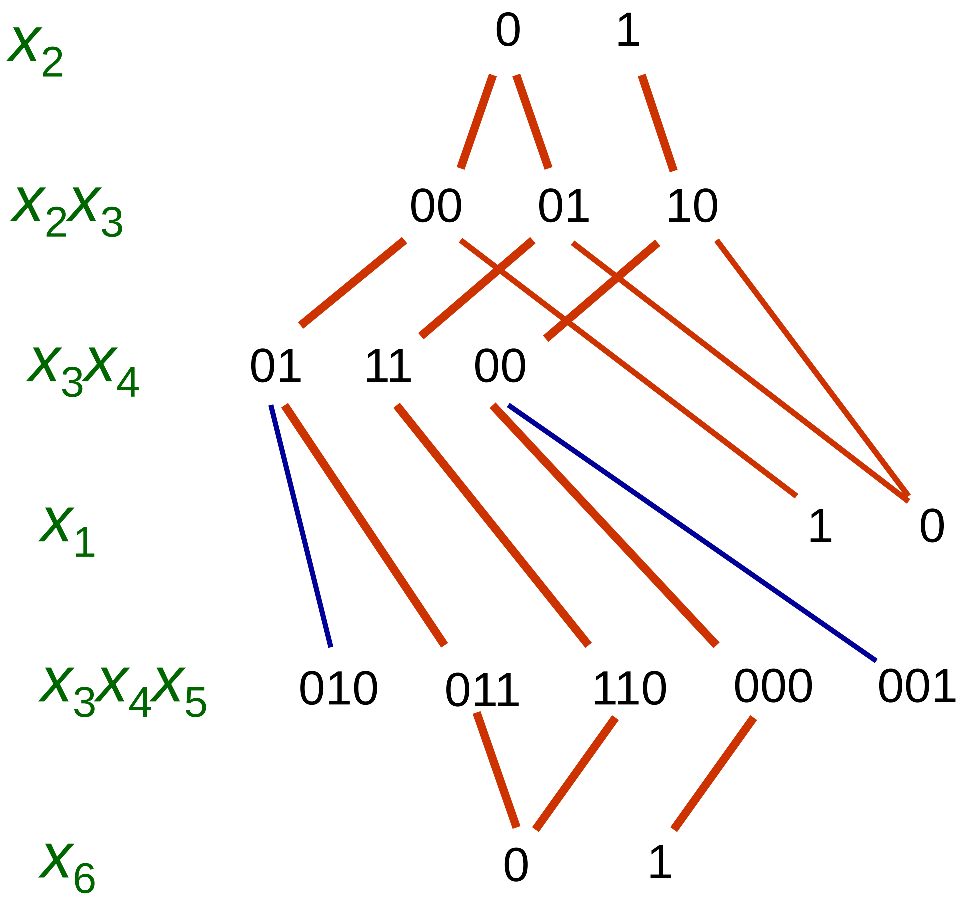
$x_1x_2x_3x_4$ 1001 0011 0100

$x_3x_4x_5$ 010 011 110 000 001

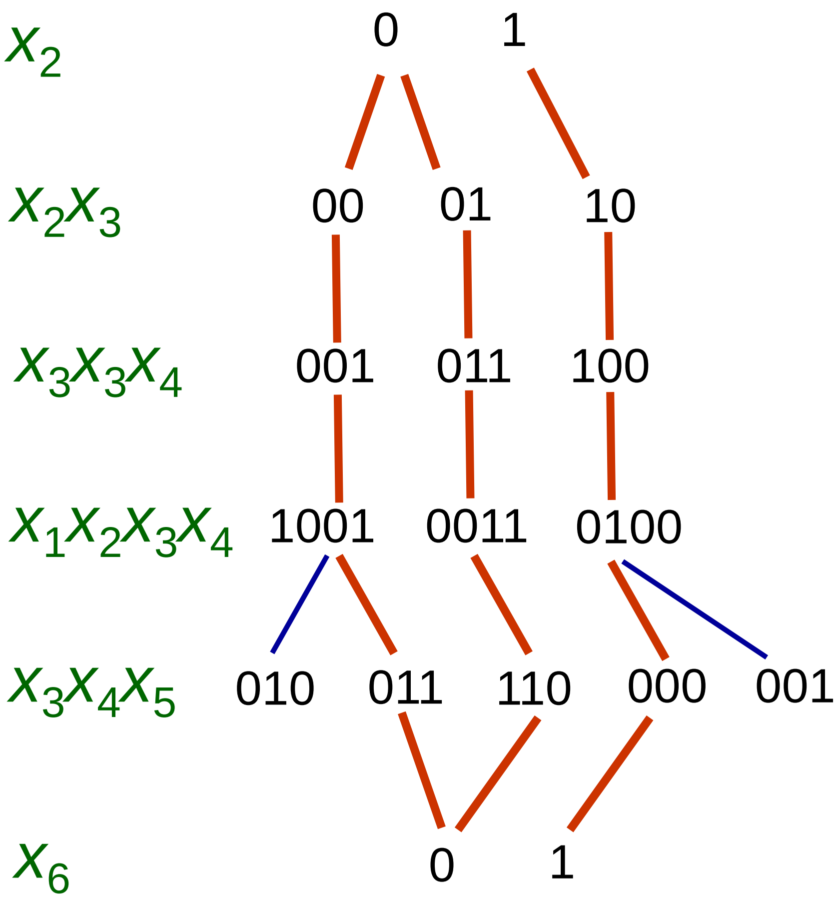
x_6 0 1

Set Partitioning example

Feasible solutions

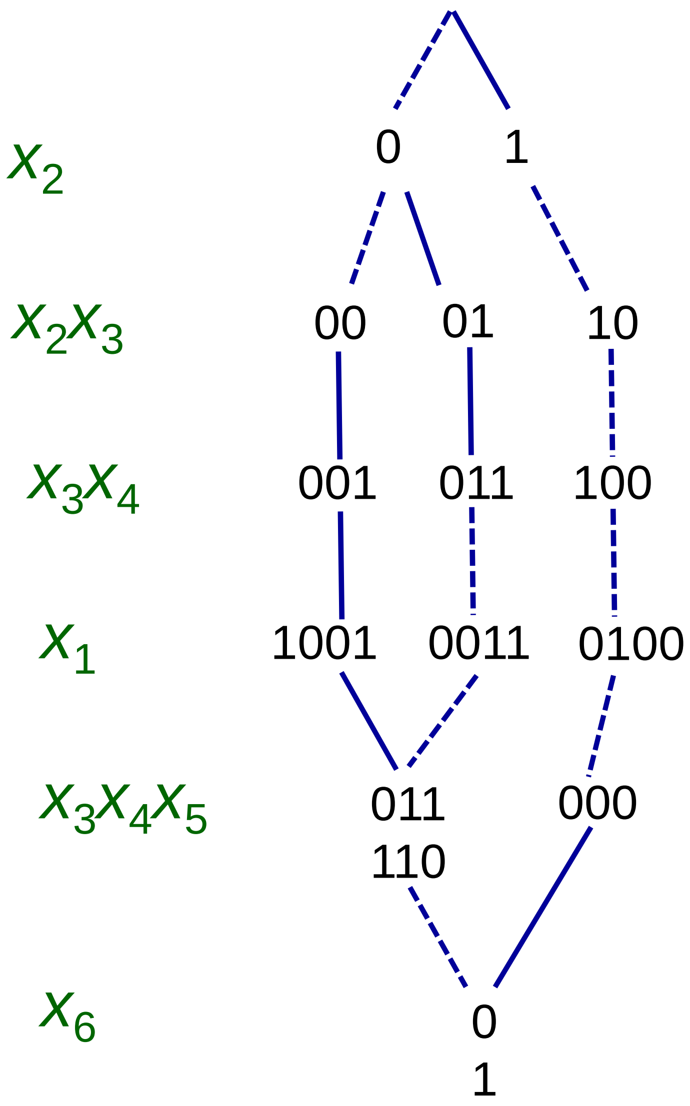


Feasible solutions



BDD vs. DP Solution

BDD



x_2 0 1

x_2x_3 00 01 10

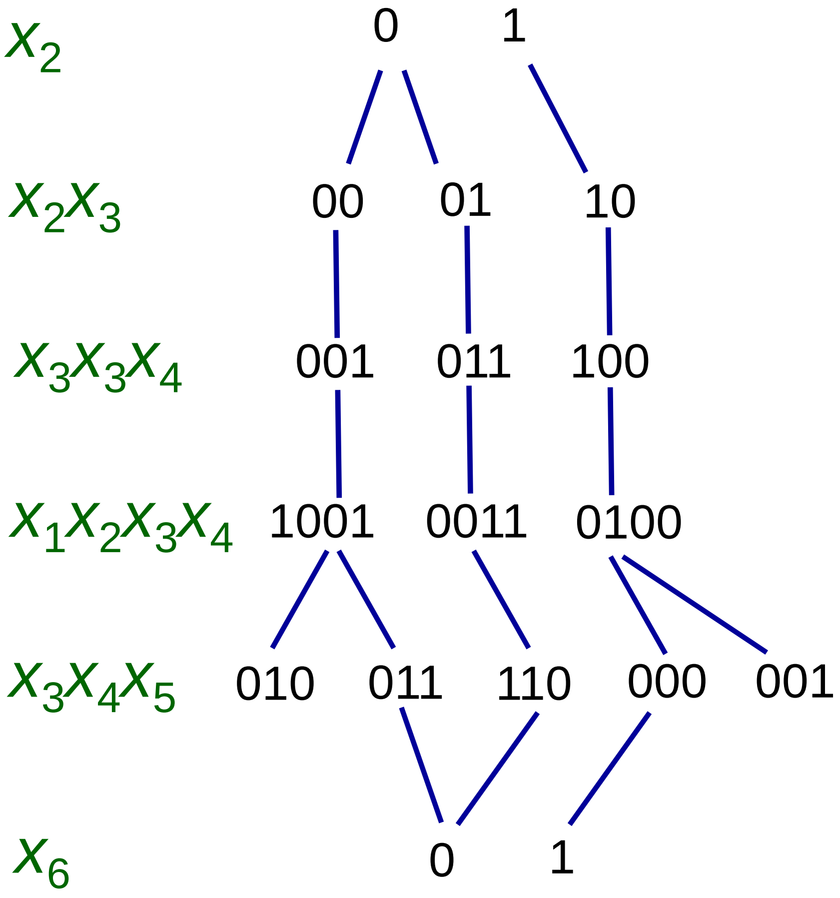
x_3x_4 001 011 100

x_1 1001 0011 0100

$x_3x_4x_5$ 011 110 000

x_6 0 1

Serialized DP



x_2 0 1

x_2x_3 00 01 10

$x_3x_3x_4$ 001 011 100

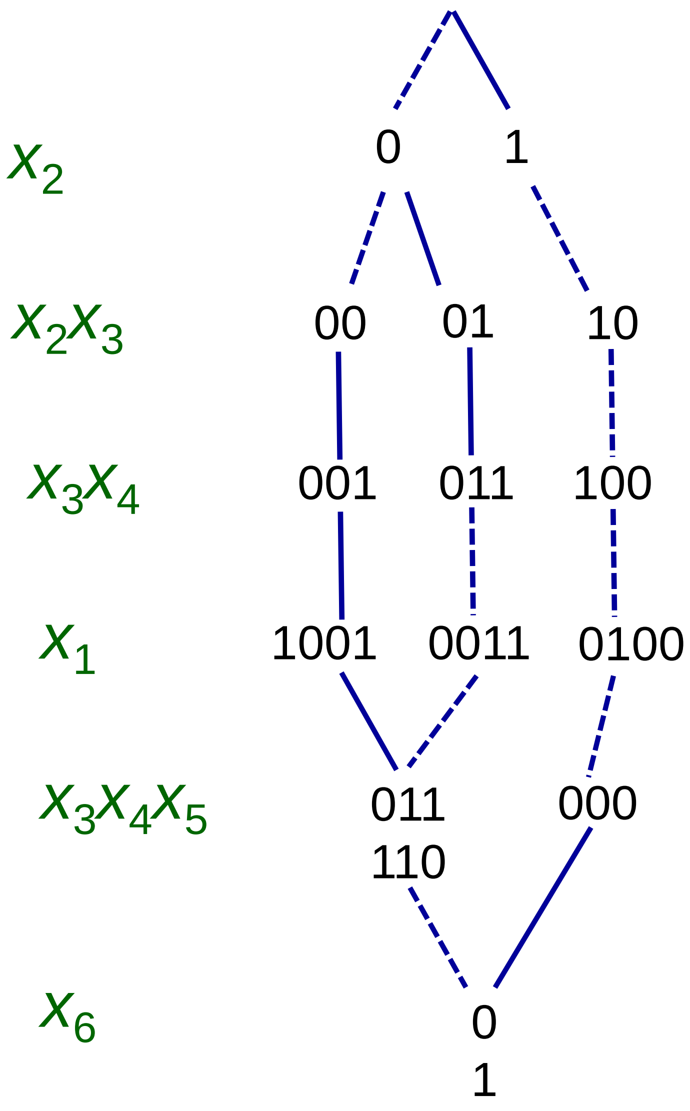
$x_1x_2x_3x_4$ 1001 0011 0100

$x_3x_4x_5$ 010 011 110 000 001

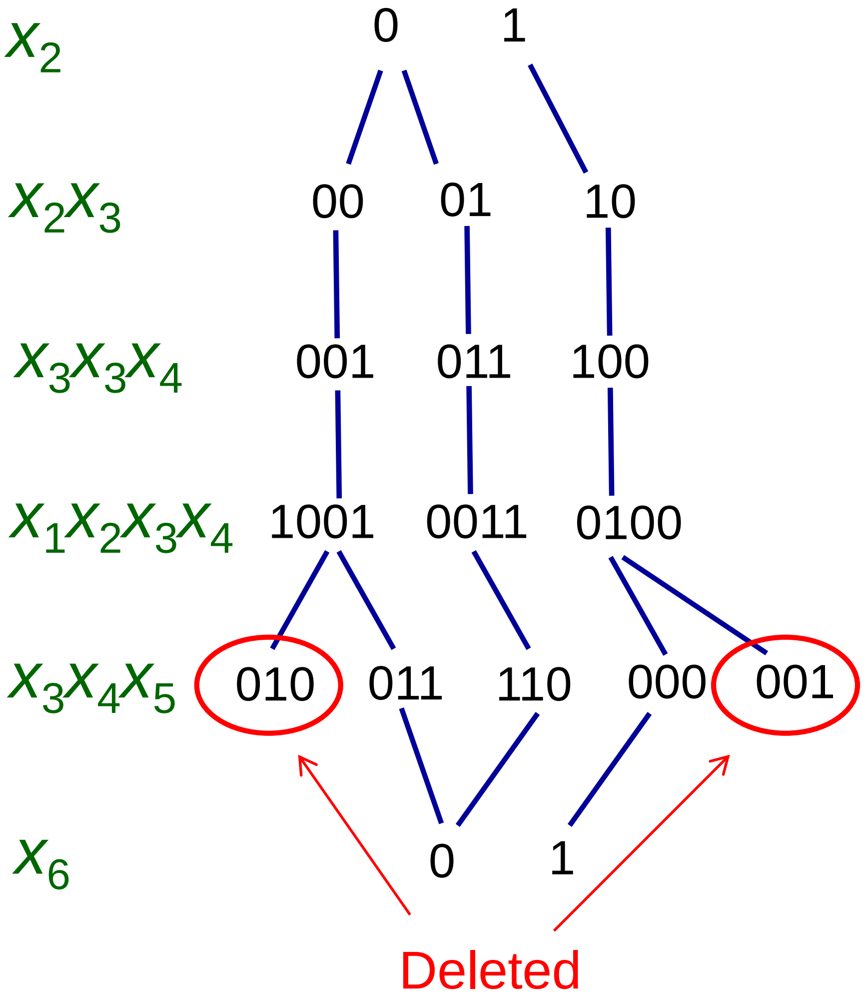
x_6 0 1

BDD vs. DP Solution

BDD

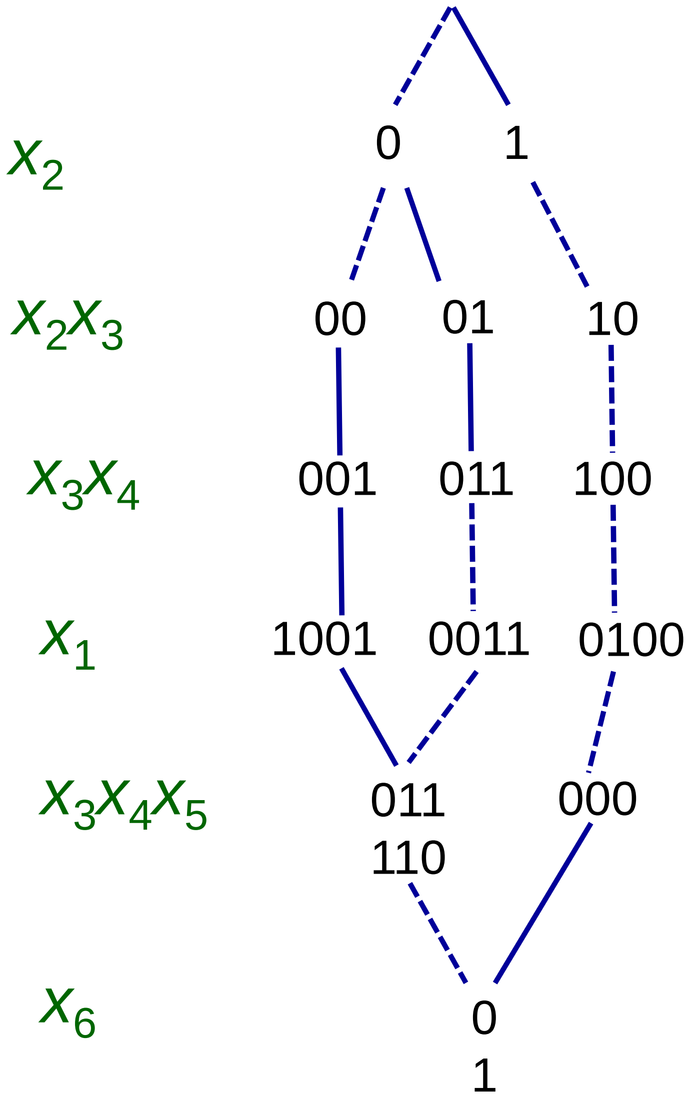


Serialized DP

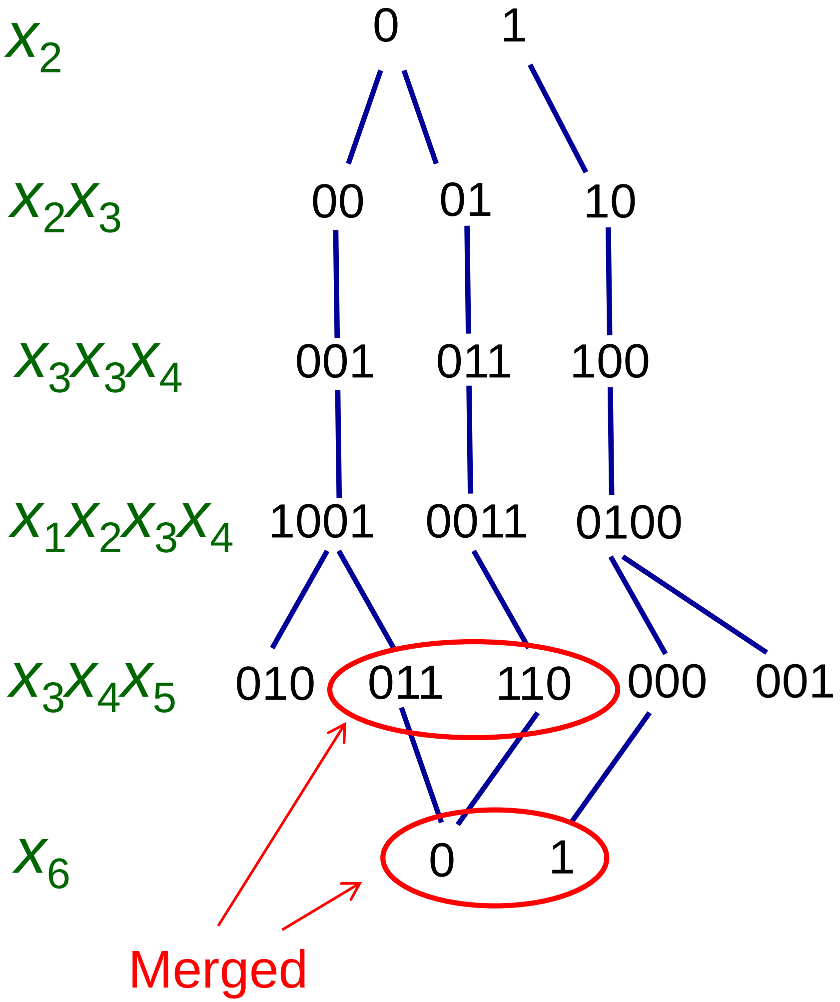


BDD vs. DP Solution

BDD

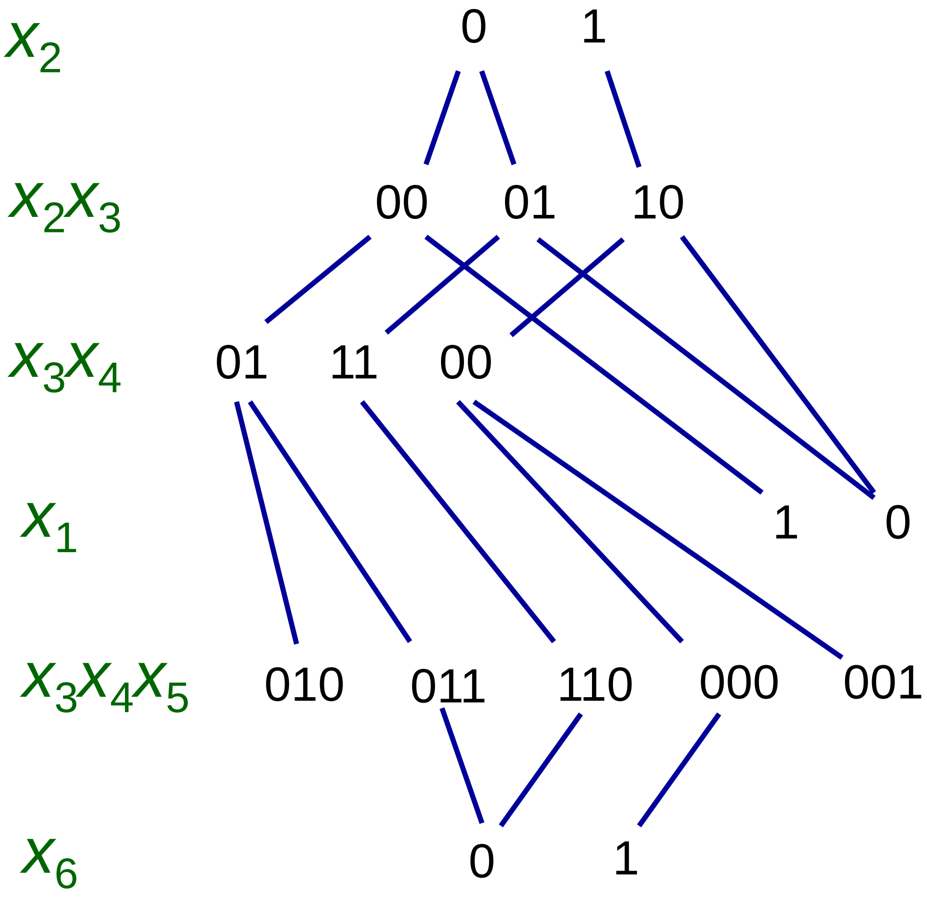


Serialized DP



Set Partitioning example

Solution by nonserial DP



x_2 0 1

x_2x_3 00 01 10

x_3x_4 01 11 00

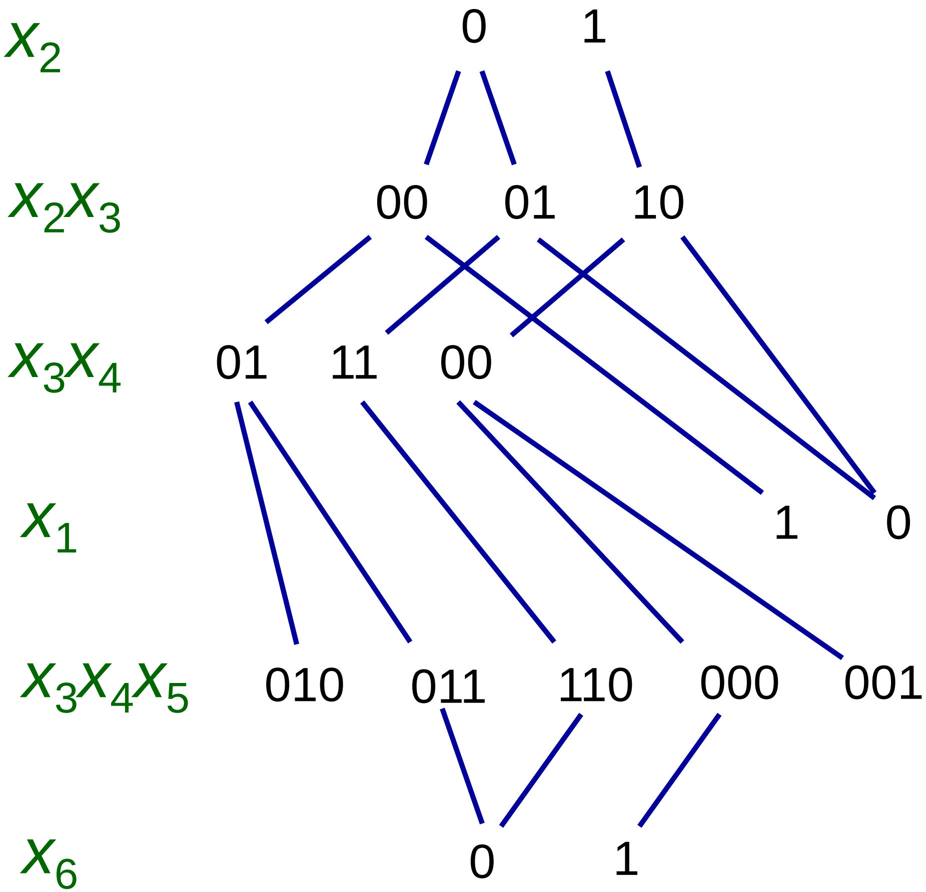
x_1 1 0

$x_3x_4x_5$ 010 011 110 000 001

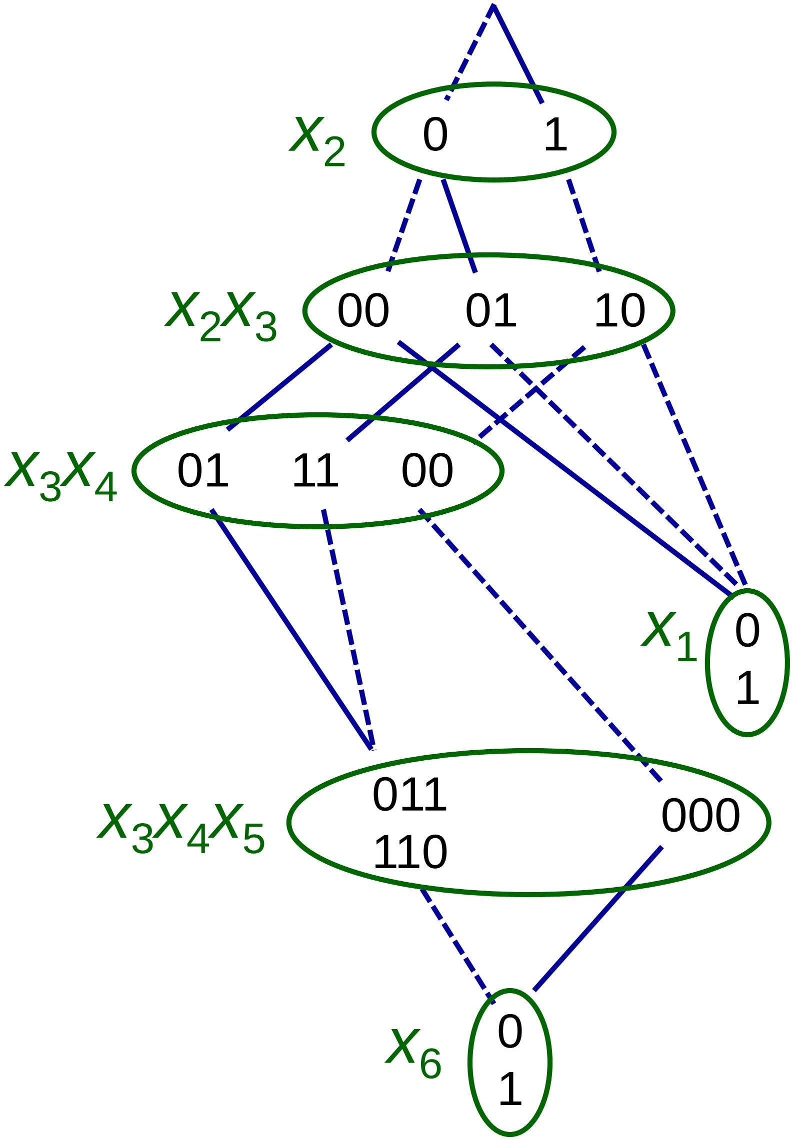
x_6 0 1

Set Partitioning example

Solution by nonserial DP



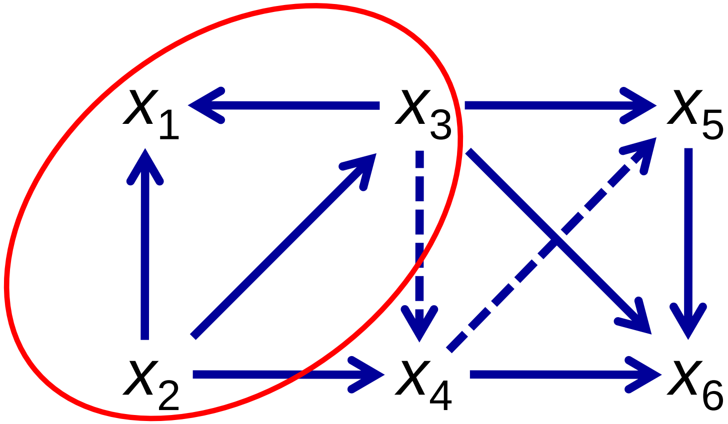
Nonserial BDD



Constructing the Join Tree

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

Clique in the dependency graph



$X_1X_2X_3$

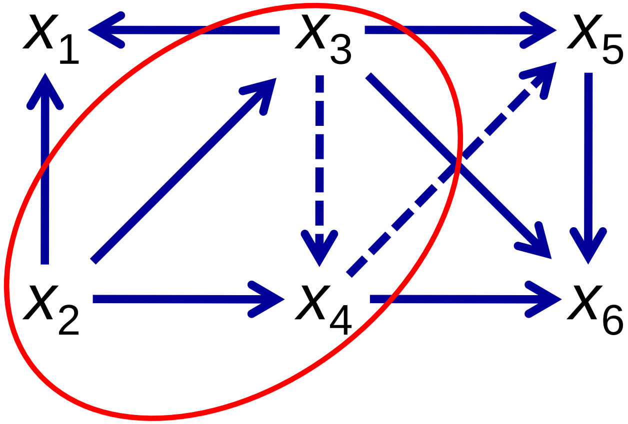
Constructing the Join Tree

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

Clique in the dependency graph

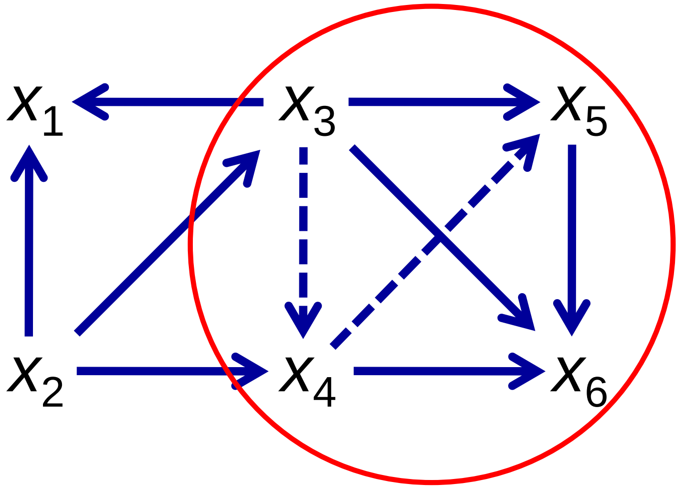
$x_1 x_2 x_3$

$x_2 x_3 x_4$



Constructing the Join Tree

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



Clique in the dependency graph

$X_1X_2X_3$

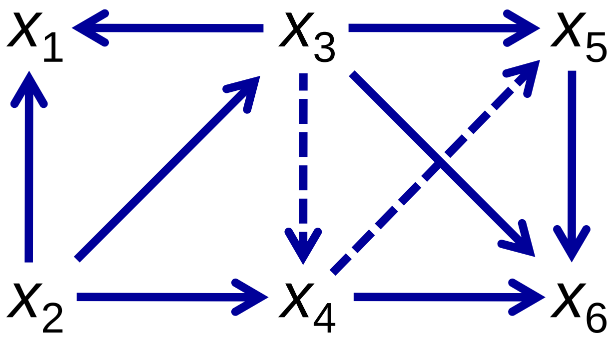
$X_2X_3X_4$

$X_3X_4X_5X_6$

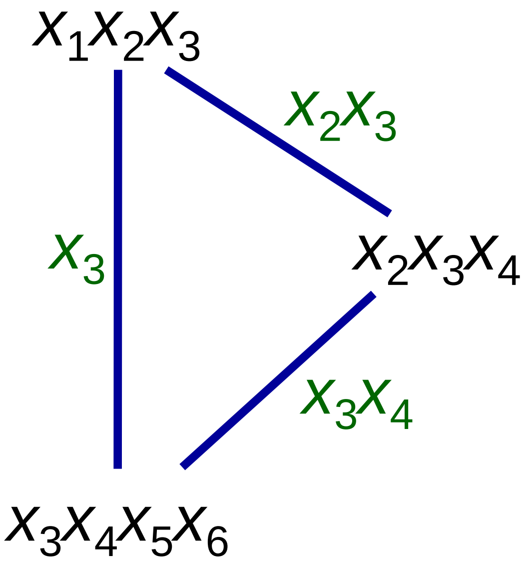
Constructing the Join Tree

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

Dependency graph



Join graph

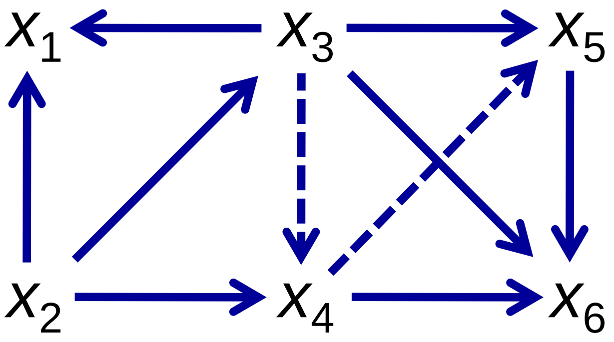


Connect nodes with common variables

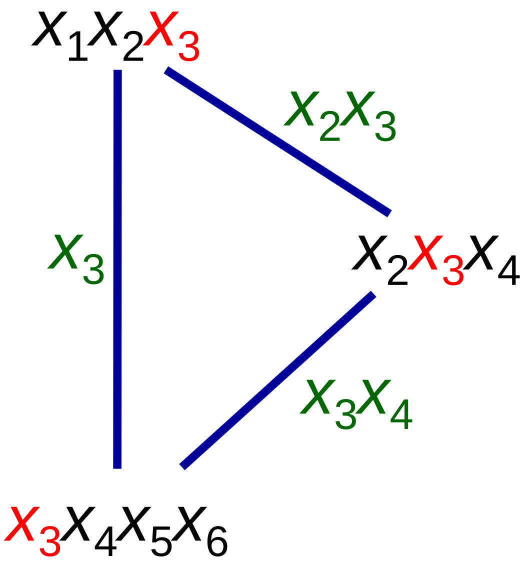
Constructing the Join Tree

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

Dependency graph



Join graph

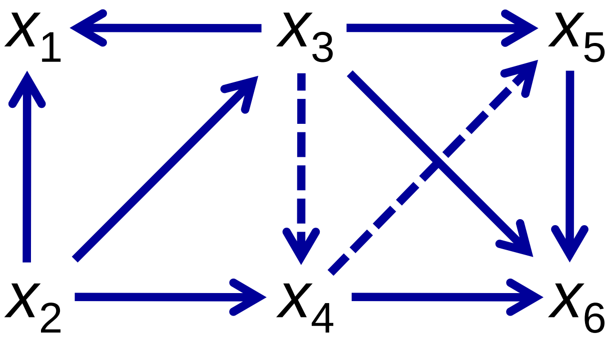


x_j occurs along every path connecting x_j with x_j

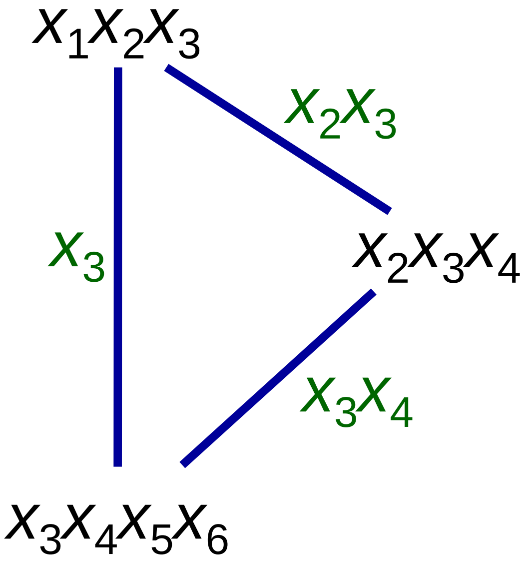
Constructing the Join Tree

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

Dependency graph



Join graph



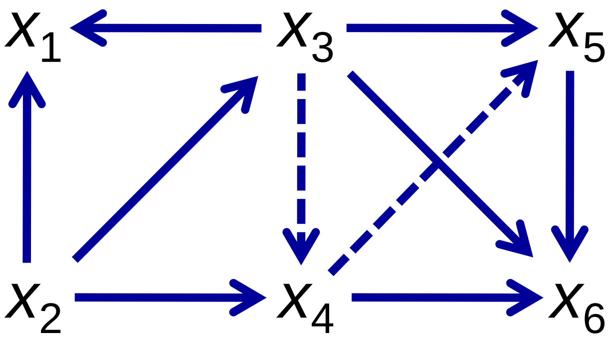
This can be viewed as the **constraint dual**

Binary constraints equate common variables in subproblems

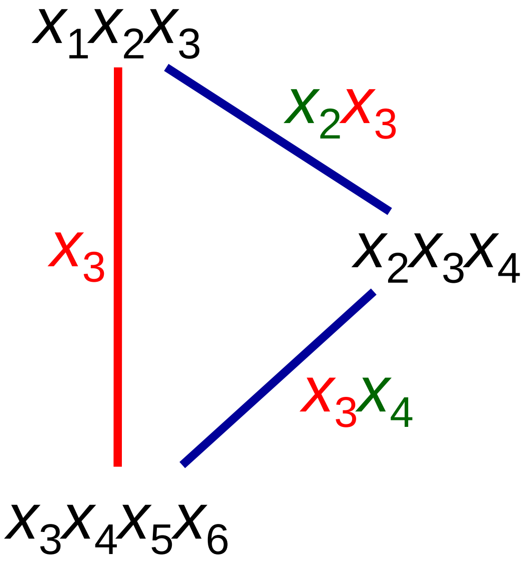
Constructing the Join Tree

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

Dependency graph



Join graph

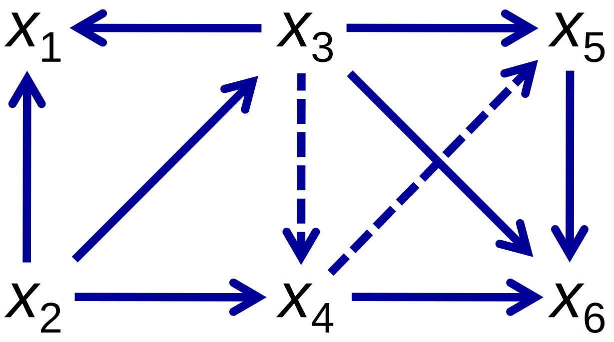


Some edges may be redundant when equating variables

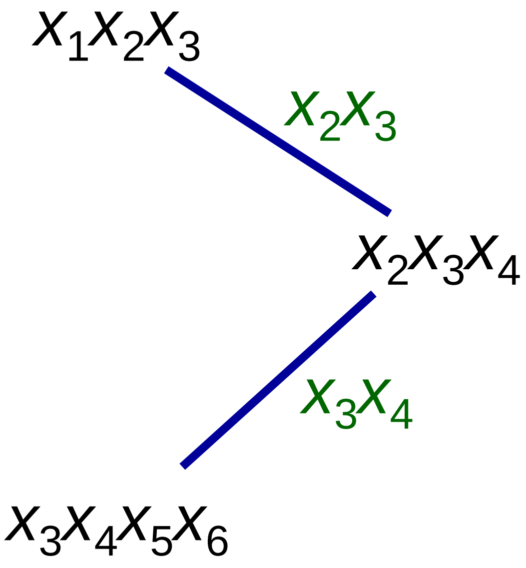
Constructing the Join Tree

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

Dependency graph



Join tree

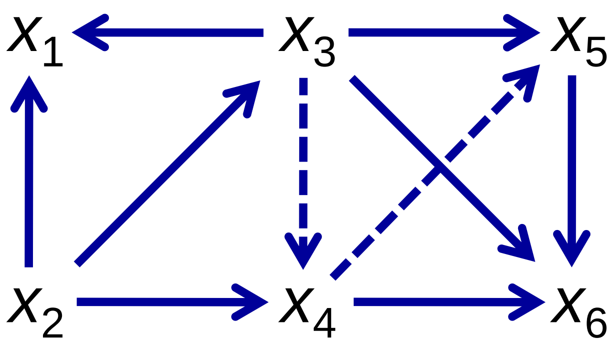


Removing redundant edges
yields **join tree**

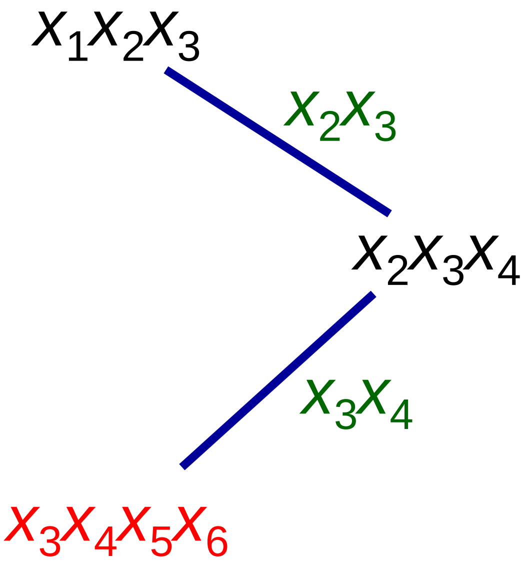
Constructing the Join Tree

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

Dependency graph



Join tree



$X_1X_2X_3$

X_2X_3

$X_2X_3X_4$

X_3X_4

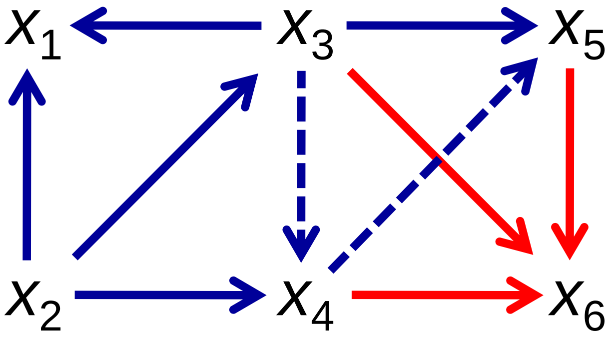
$X_3X_4X_5X_6$

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

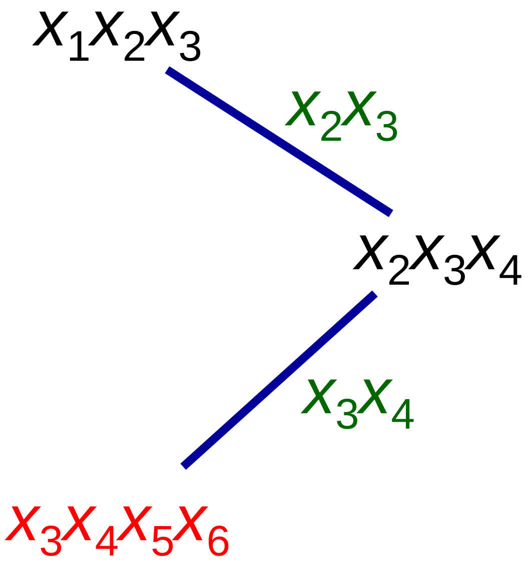
Constructing the Join Tree

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

Dependency graph



Join tree



Induced width = tree width = 3

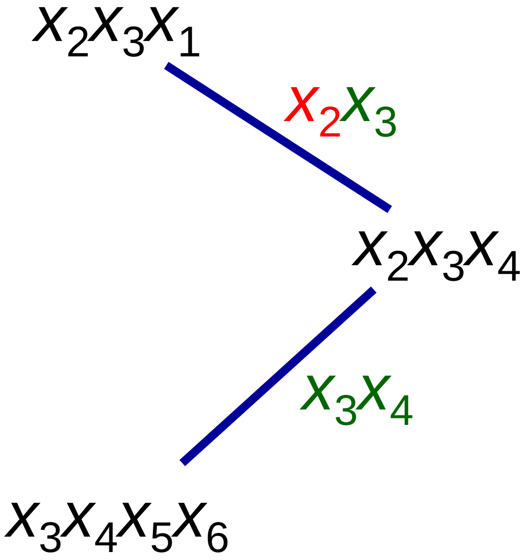
Designing the Nonserial BDD

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

BDD design

x_2

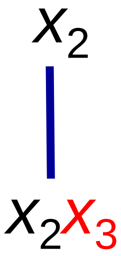
Join tree



Designing the Nonserial BDD

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

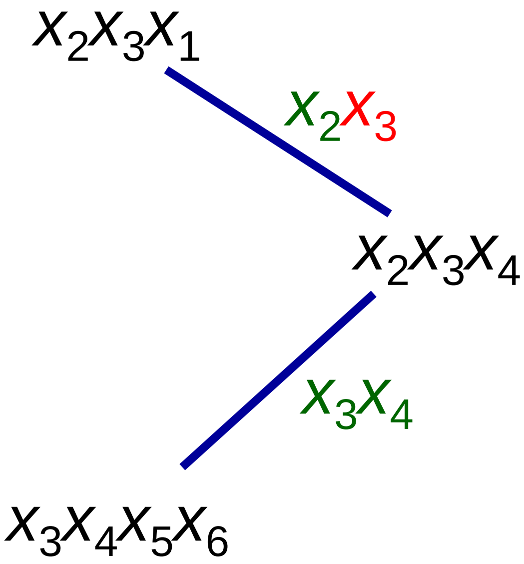
BDD design



x_2

x_2x_3

Join tree



$x_2x_3x_1$

x_2x_3

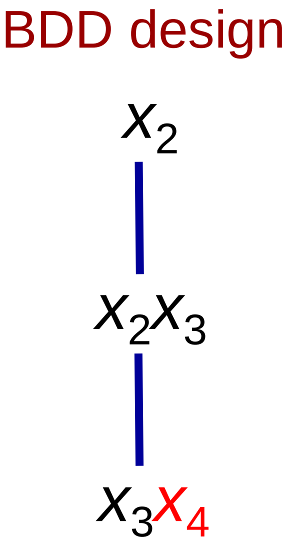
$x_2x_3x_4$

x_3x_4

$x_3x_4x_5x_6$

Designing the Nonserial BDD

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

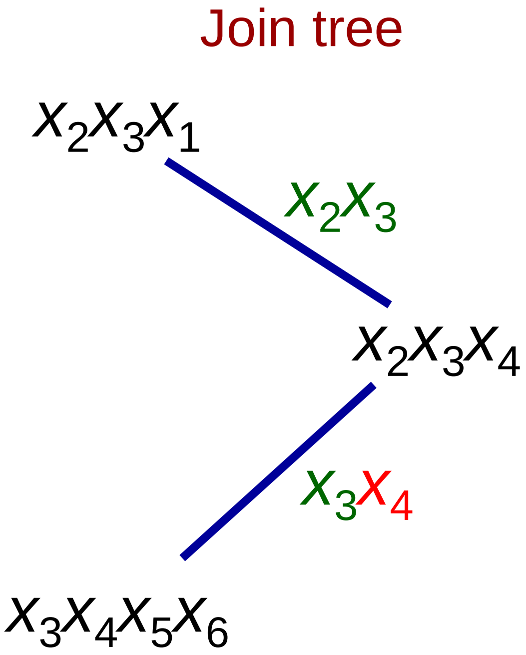


BDD design

X_2

X_2X_3

X_3X_4



Join tree

$X_2X_3X_1$

X_2X_3

$X_2X_3X_4$

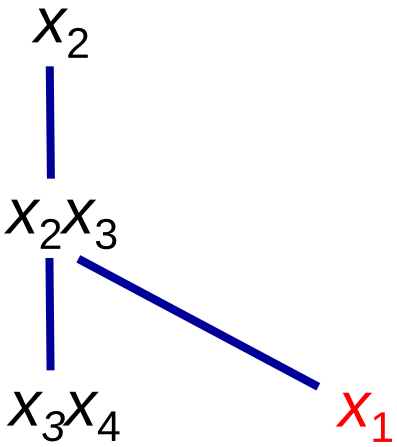
X_3X_4

$X_3X_4X_5X_6$

Designing the Nonserial BDD

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

BDD design



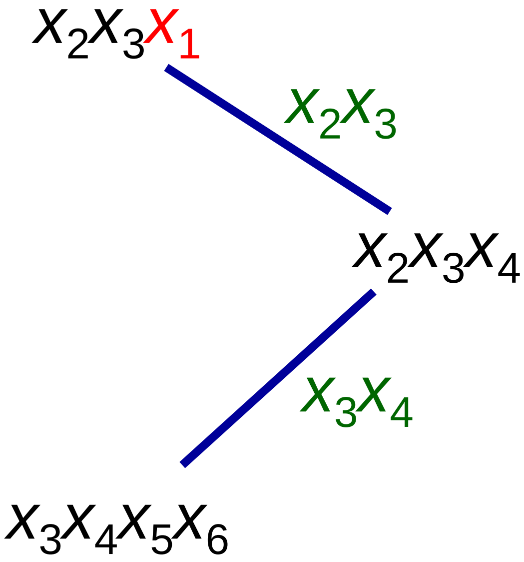
x_2

x_2x_3

x_3x_4

x_1

Join tree



$x_2x_3x_1$

x_2x_3

$x_2x_3x_4$

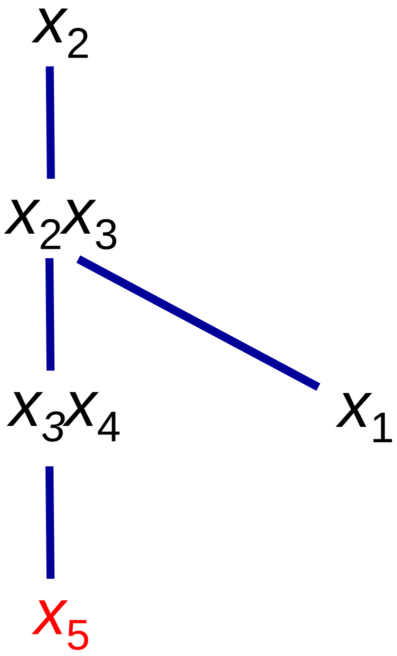
x_3x_4

$x_3x_4x_5x_6$

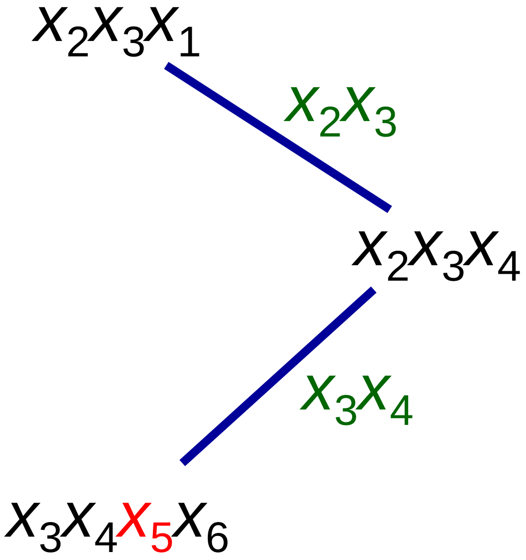
Designing the Nonserial BDD

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

BDD design



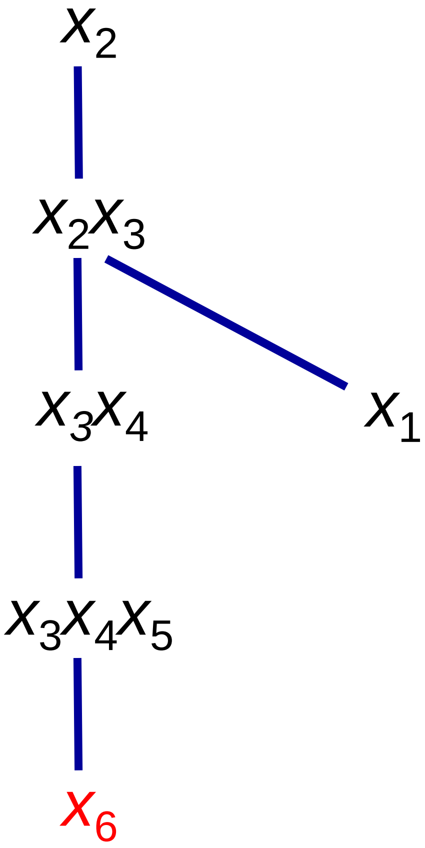
Join tree



Designing the Nonserial BDD

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

BDD design



x_2

x_2x_3

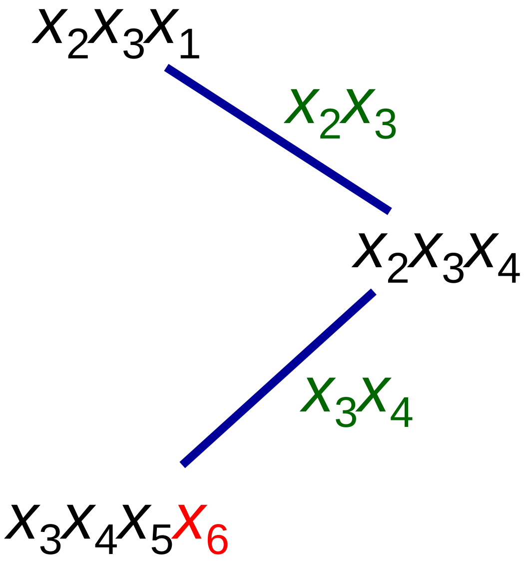
x_3x_4

x_1

$x_3x_4x_5$

x_6

Join tree



$x_2x_3x_1$

x_2x_3

$x_2x_3x_4$

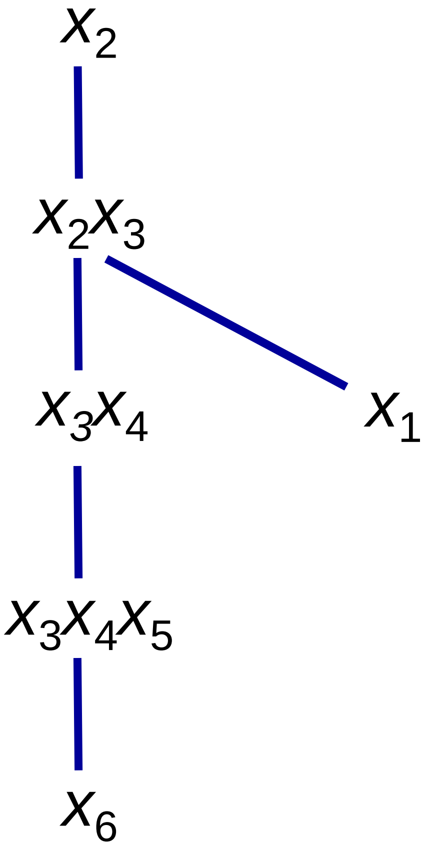
x_3x_4

$x_3x_4x_5x_6$

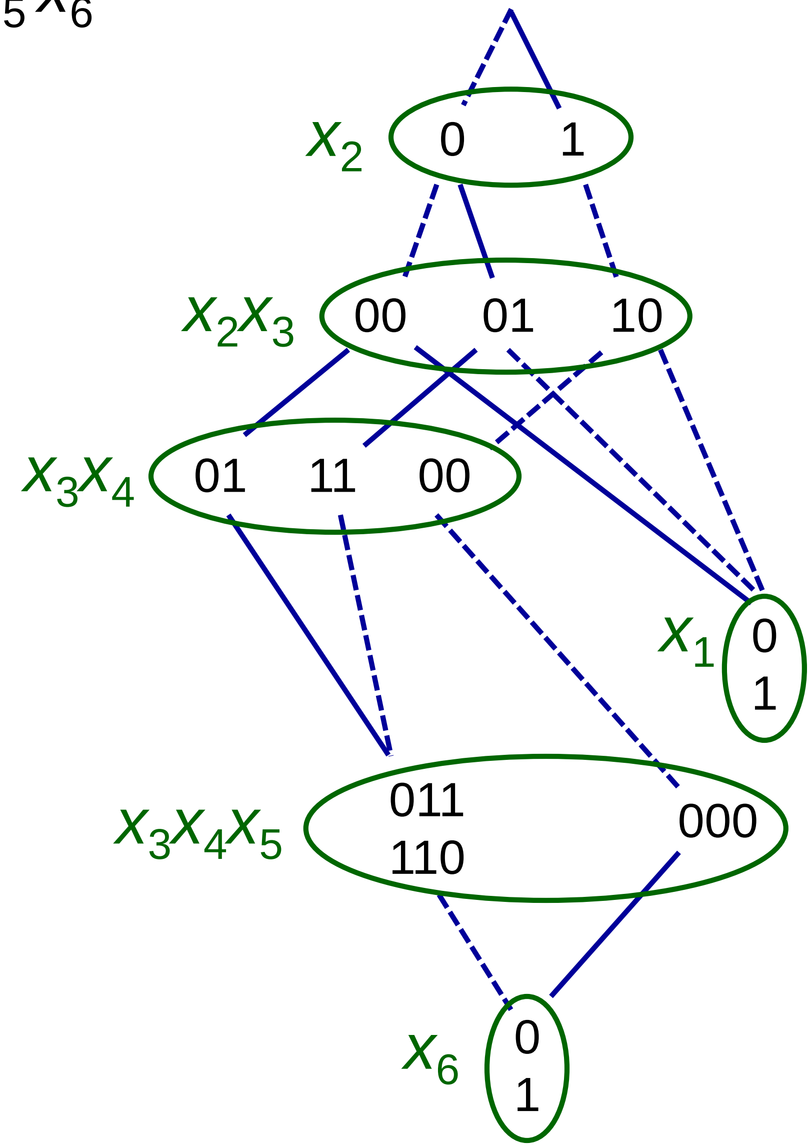
Designing the Nonserial BDD

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

BDD design



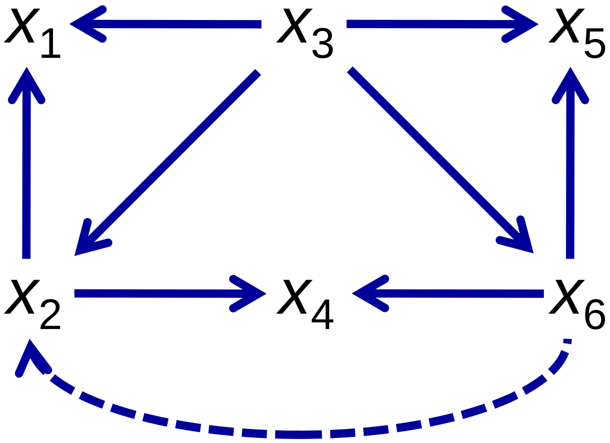
Nonserial BDD



Another Variable Ordering

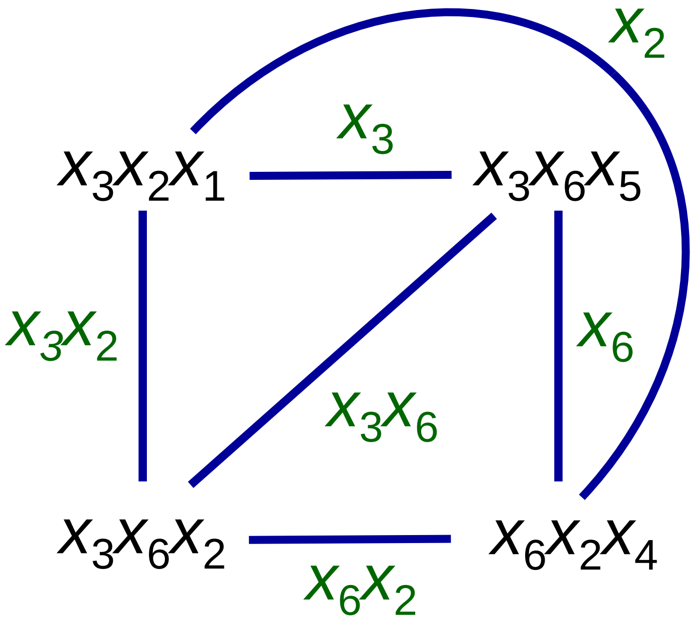
$X_3\ X_6\ X_2\ X_5\ X_1\ X_4$

Dependency graph



Induced width = 2

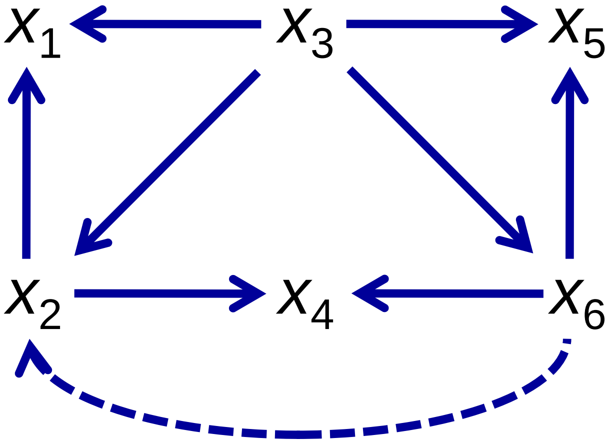
Join graph



Constructing the Join Tree

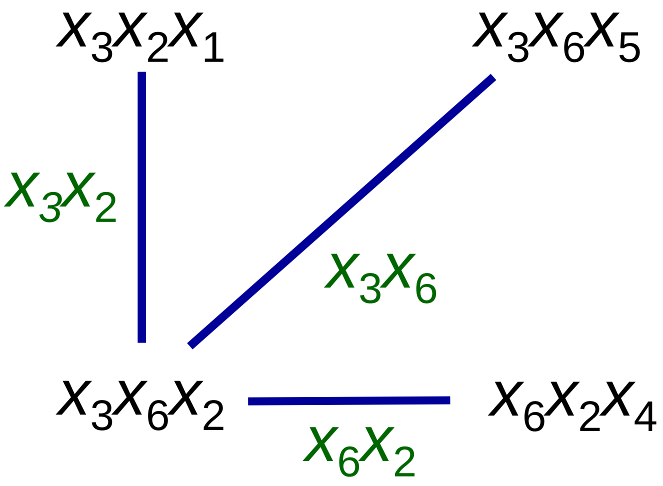
$X_3 \; X_6 \; X_2 \; X_5 \; X_1 \; X_4$

Dependency graph



Induced width = 2

Join tree



Tree width = 2

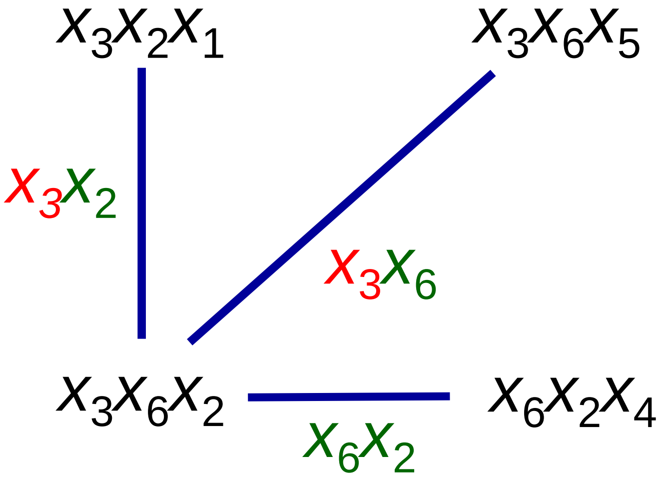
Designing the BDD

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

BDD design

x_3

Join tree


$x_3x_2x_1$

$x_3x_6x_5$

x_3x_2

x_3x_6

$x_3x_6x_2$

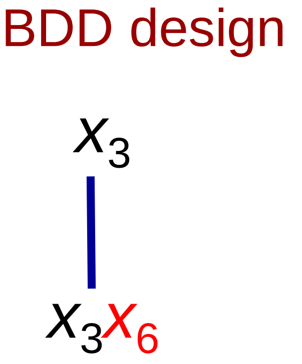
$x_6x_2x_4$

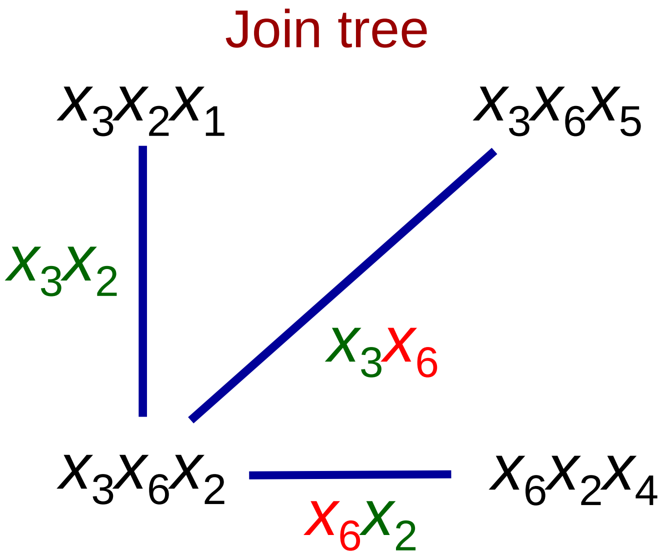
x_6x_2

Tree width = 2

Designing the BDD

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



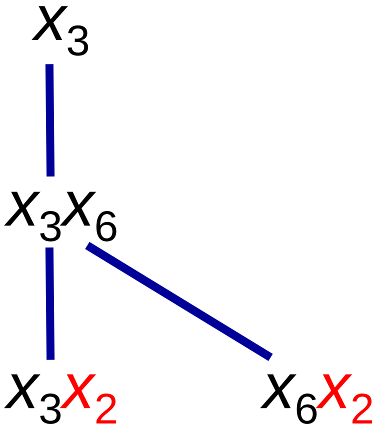


Tree width = 2

Designing the BDD

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

BDD design



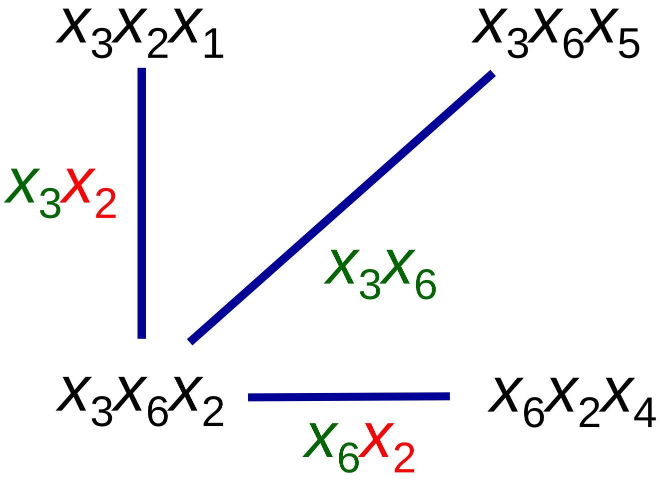
x_3

x_3x_6

x_3x_2

x_6x_2

Join tree



$x_3x_2x_1$

$x_3x_6x_5$

x_3x_2

x_3x_6

$x_3x_6x_2$

$x_6x_2x_4$

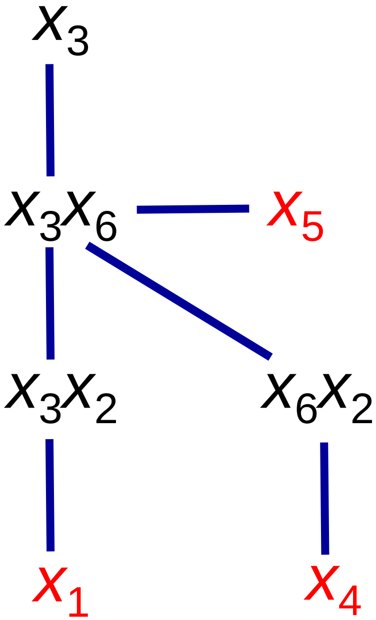
x_6x_2

Tree width = 2

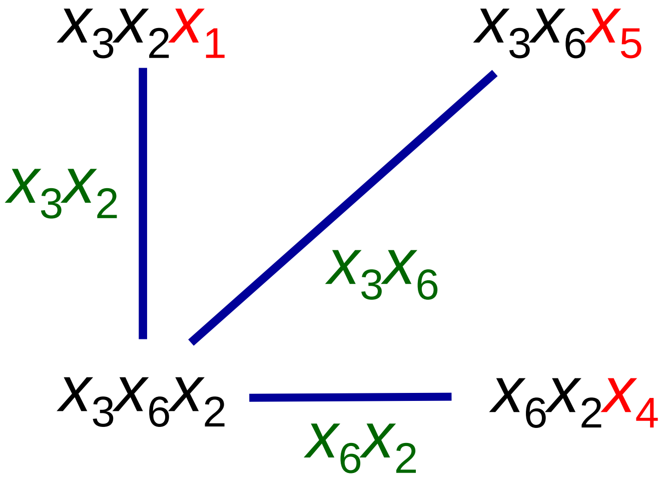
Designing the BDD

X_3 X_6 X_2 X_5 X_1 X_4

BDD design



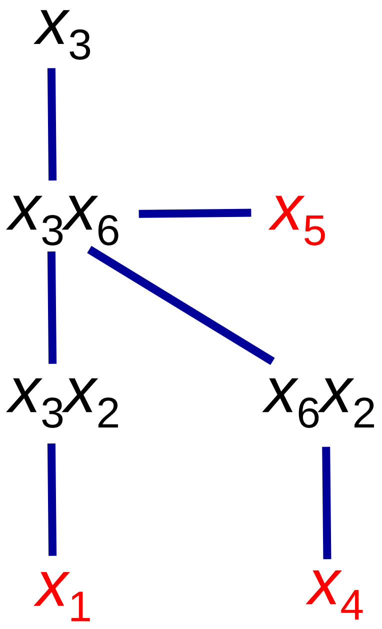
Join tree

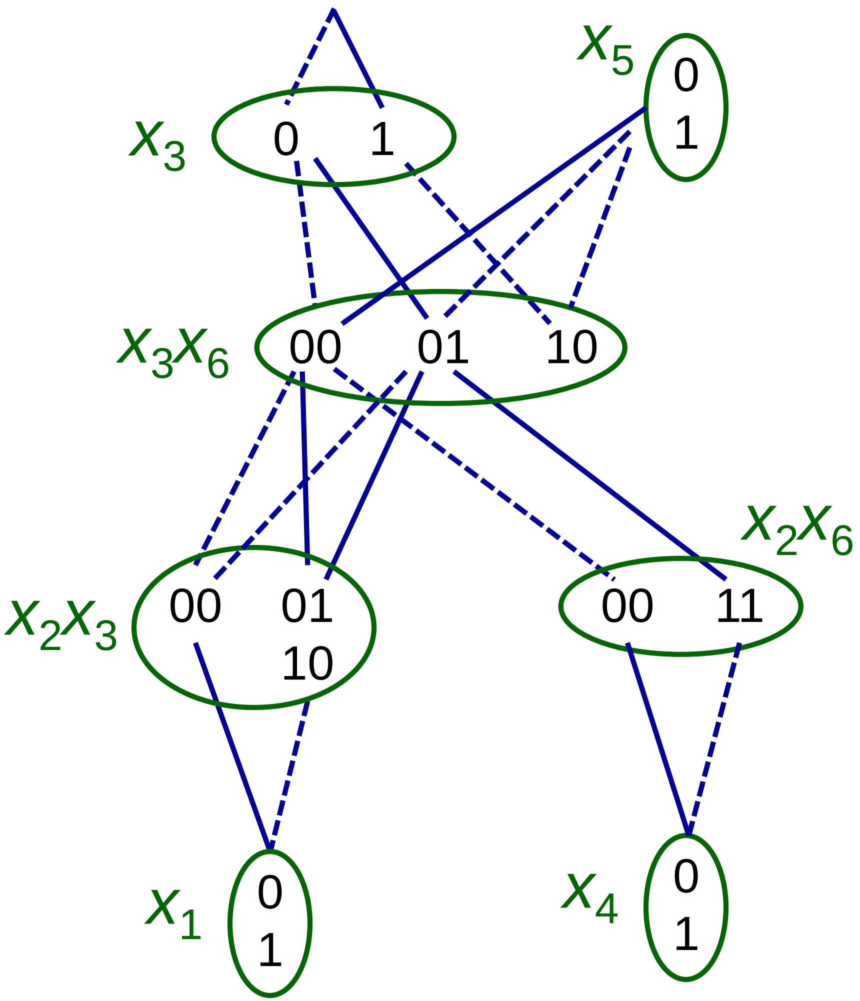


Tree width = 2

Nonserial BDD

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


BDD design


Nonserial BDD

Current Research

- Broader applicability
  - Stochastic dynamic programming
  - Continuous global optimization
- Combination with other techniques
  - Lagrangean relaxation.
  - Column generation
  - Logic-based Benders decomposition
    - Solve separation problem

References

2006

- T. Hadzic and J. N. Hooker. Discrete global optimization with binary decision diagrams. In *Workshop on Global Optimization: Integrating Convexity, Optimization, Logic Programming, and Computational Algebraic Geometry (GICOLAG)*, Vienna, 2006.

2007

- Tarik Hadzic and J. N. Hooker. Cost-bounded binary decision diagrams for 0-1 programming. In *Proceedings of CPAIOR*. LNCS 4510, pp. 84-98. Springer, 2007.
- Tarik Hadzic and J. N. Hooker. Postoptimality analysis for integer programming using binary decision diagrams. December 2007, revised April 2008 (not submitted).
- M. Behle. Binary Decision Diagrams and Integer Programming. PhD thesis, Max Planck Institute for Computer Science, 2007.
- H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann. A constraint store based on multivalued decision diagrams. In *Proceedings of CP*. LNCS 4741, pp. 118-132. Springer, 2007.

2008

- T. Hadzic, J. N. Hooker, B. O'Sullivan, and P. Tiedemann. Approximate compilation of constraints into multivalued decision diagrams. In *Proceedings of CP*. LNCS 5202, pp. 448-462. Springer, 2008.
- T. Hadzic, J. N. Hooker, and P. Tiedemann. Propagating separable equalities in an MDD store. In *Proceedings of CPAIOR*. LNCS 5015, pp. 318-322. Springer, 2008.

References

2010

- S. Hoda. Essays on Equilibrium Computation, MDD-based Constraint Programming and Scheduling. *PhD thesis*, Carnegie Mellon University, 2010.
- S. Hoda, W.-J. van Hoeve, and J. N. Hooker. A Systematic Approach to MDD-Based Constraint Programming. In *Proceedings of CP*. LNCS 6308, pp. 266-280. Springer, 2010.
- T. Hadzic, E. O'Mahony, B. O'Sullivan, and M. Sellmann. Enhanced inference for the market split problem. In *Proceedings, International Conference on Tools for AI (ICTAI)*, pages 716–723. IEEE, 2009.

2011

- D. Bergman, W.-J. van Hoeve, and J. N. Hooker. Manipulating MDD Relaxations for Combinatorial Optimization. In *Proceedings of CPAIOR*. LNCS 6697, pp. 20-35. Springer, 2011.

2012

- A. A. Cire and W.-J. van Hoeve. MDD Propagation for Disjunctive Scheduling. In *Proceedings of ICAPS*, pp. 11-19. AAAI Press, 2012.
- D. Bergman, A.A. Cire, W.-J. van Hoeve, and J.N. Hooker. Variable Ordering for the Application of BDDs to the Maximum Independent Set Problem. In *Proceedings of CPAIOR*. LNCS 7298, pp. 34-49. Springer, 2012.

References

2013

- A. A. Cire and W.-J. van Hoeve. Multivalued Decision Diagrams for Sequencing Problems. *Operations Research* 61(6): 1411-1428, 2013.
- D. Bergman. New Techniques for Discrete Optimization. *PhD thesis*, Carnegie Mellon University, 2013.
- J. N. Hooker. Decision Diagrams and Dynamic Programming. In *Proceedings of CPAIOR*. LNCS 7874, pp. 94-110. Springer, 2013.
- B. Kell and W.-J. van Hoeve. An MDD Approach to Multidimensional Bin Packing. In *Proceedings of CPAIOR*, LNCS 7874, pp. 128-143. Springer, 2013.

2014

- D. R. Morrison, E. C. Sewell, S. H. Jacobson , Characteristics of the maximal independent set ZDD, Journal of Combinatorial Optimization 28 (1) 121-139, 2014
- D. R. Morrison, E. C. Sewell, S. H. Jacobson, Solving the Pricing Problem in a Generic Branch-and-Price Algorithm using Zero-Suppressed Binary Decision Diagrams,
- D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker. Optimization Bounds from Binary Decision Diagrams. *INFORMS Journal on Computing* 26(2): 253-258, 2014.
- A. A. Cire. Decision Diagrams for Optimization. *PhD thesis*, Carnegie Mellon University, 2014.
- D. Bergman, A. A. Cire, and W.-J. van Hoeve. MDD Propagation for Sequence Constraints. *JAIR*, Volume 50, pages 697-722, 2014.
- D. Bergman, A. A. Cire, W.-J. van Hoeve, and T. Yunes. BDD-Based Heuristics for Binary Optimization. *Journal of Heuristics* 20(2): 211-234, 2014.

References

2014

- D. Bergman, A. A. Cire, A. Sabharwal, H. Samulowitz, V. Saraswat, and W.-J. van Hoeve. Parallel Combinatorial Optimization with Decision Diagrams. In *Proceedings of CPAIOR*, LNCS 8451, pp. 351-367. Springer, 2014.
- A. A. Cire and J. N. Hooker. The Separation Problem for Binary Decision Diagrams. In *Proceedings of the International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, 2014.]

2015

- D. Bergman, A. A. Cire, and W.-J. van Hoeve. Lagrangian Bounds from Decision Diagrams. *Constraints* 20(3): 346-361, 2015.
- B. Kell, A. Sabharwal, and W.-J. van Hoeve. BDD-Guided Clause Generation. In *Proceedings of CPAIOR*, 2015.

2016

- D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker, *Decision Diagrams for Optimization*, Springer, to appear.
- D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. N. Hooker. Discrete Optimization with Decision Diagrams. *INFORMS Journal on Computing* 28: 47-66, 2016.