

Optimization Bounds from Binary Decision Diagrams

J. N. Hooker

Joint work with

David Bergman, André Ciré, Willem van Hoeve

Carnegie Mellon University

ICS 2013

Binary Decision Diagrams

- **BDDs** historically used for circuit design and verification.
 - Lee 1959, Akers 1978, Bryant 1986.

Binary Decision Diagrams

- **BDDs** historically used for circuit design and verification.
 - Lee 1959, Akers 1978, Bryant 1986.
- **Compact** graphical representation of **boolean** function.
 - Can also represent **feasible set** of problem with binary variables.
 - Slight generalization (MDDs) represents finite domain variables.

Binary Decision Diagrams

- BDD is result of superimposing isomorphic subtrees in a search tree.
 - Unique reduced BDD for given variable ordering.
 - “Caching” is a popular theme in knowledge representation.
- Constraints need not have an inequality representation.

The 0-1 inequality

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

has 117,520 minimal
solutions

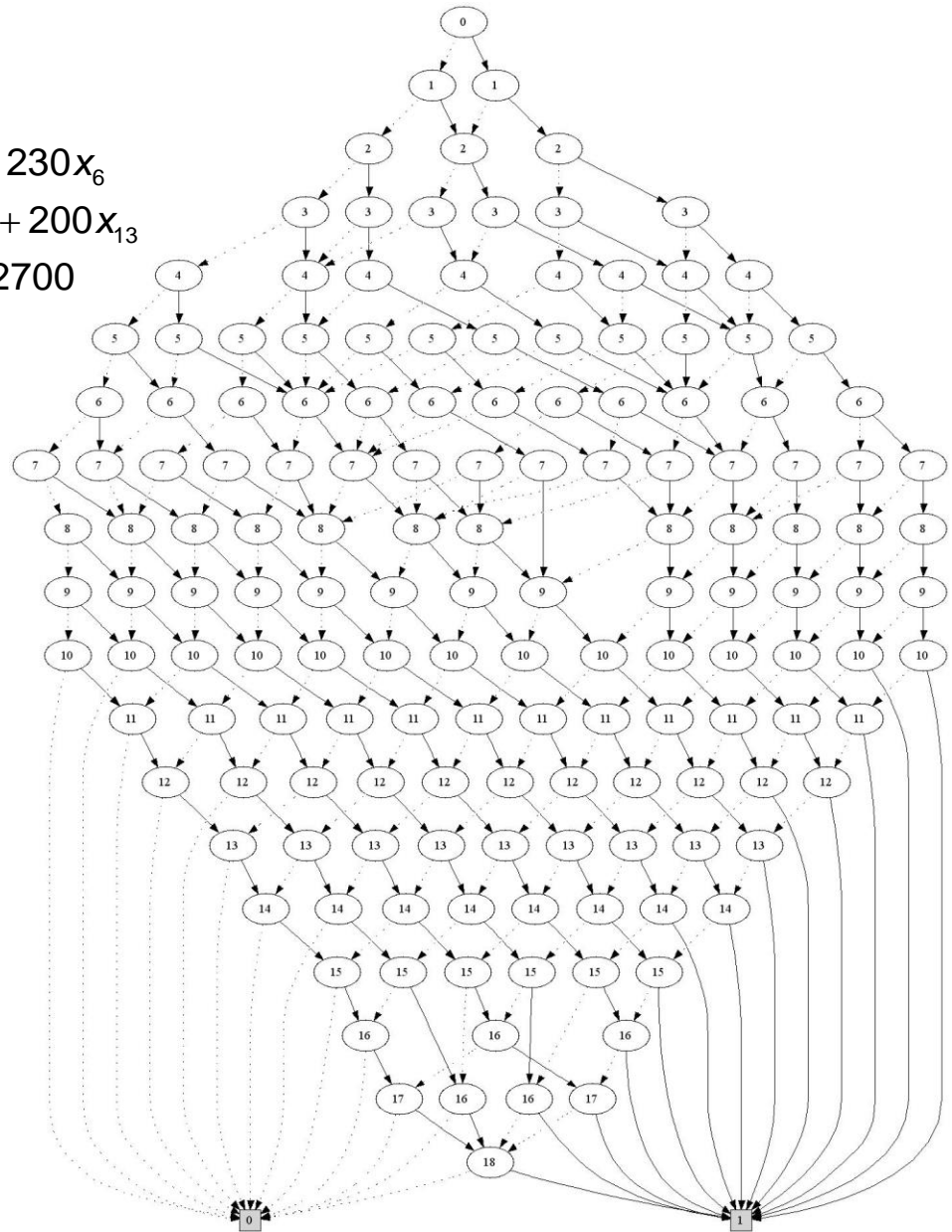
The 0-1 inequality

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

has 117,520 minimal
solutions

The BDD has only 152
nodes.

Paths from top to
bottom right
correspond to feasible
solutions



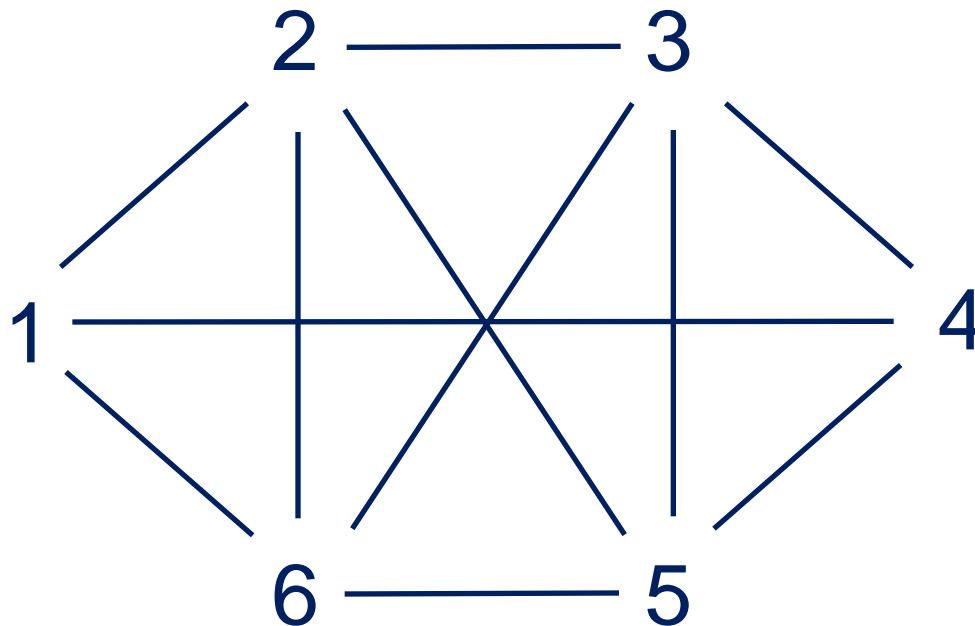
Binary Decision Diagrams

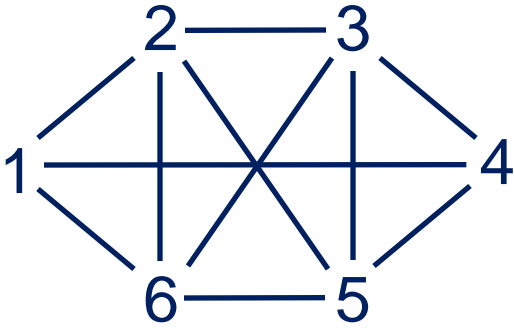
- BDD can grow exponentially with problem size.
 - So we use a smaller, **relaxed** BDD that represents **superset** of feasible set.
 - Andersen, Hadzic, Hooker, Tiedemann 2007.
 - For alldiff systems, reduced search tree from >1 million nodes to 1 node.
 - Subsequent papers with Hadzic, Hoda, van Hove, O'Sullivan.
- We focus on **independent set problem** on a graph...

Independent Set Problem

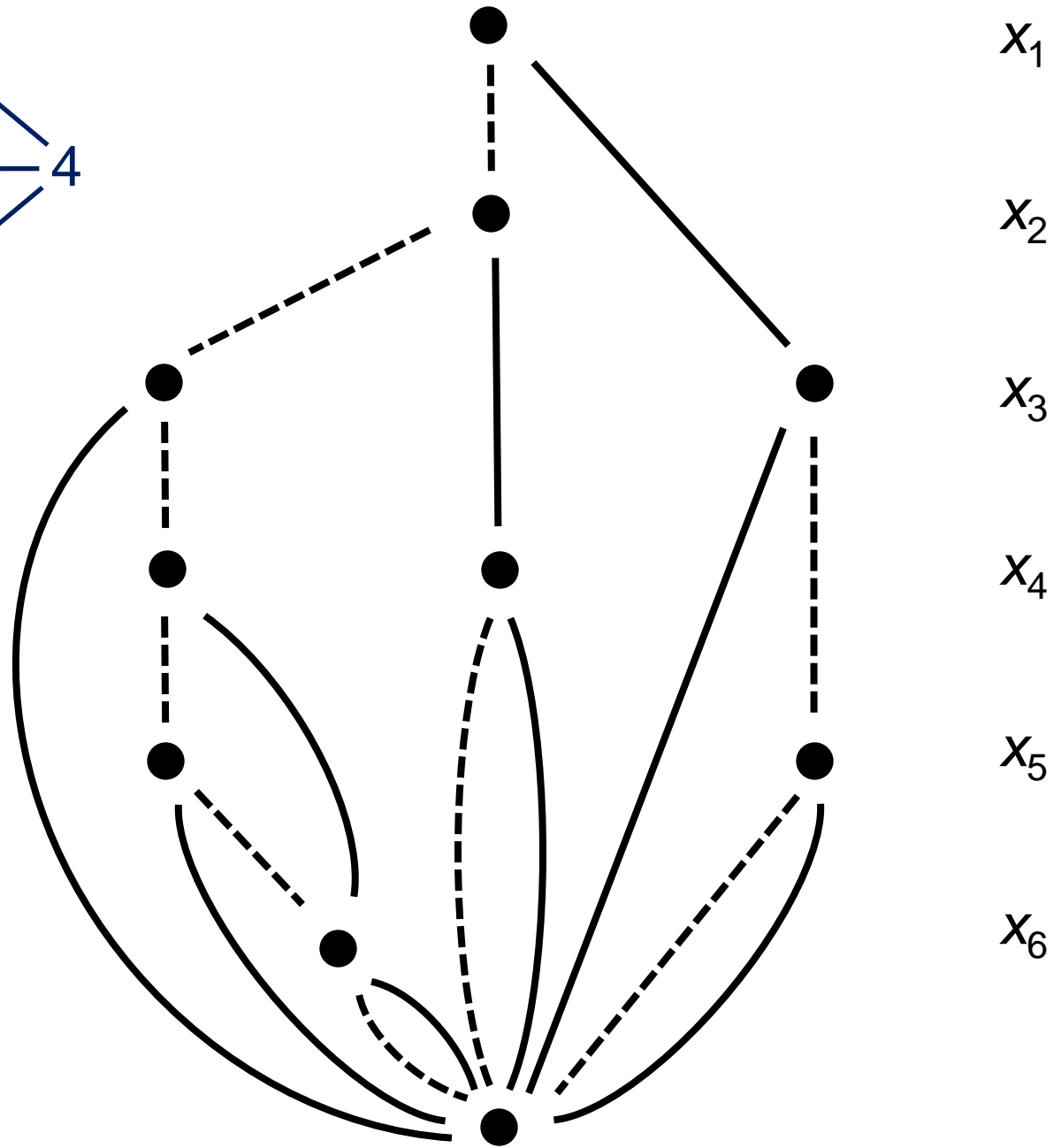
Let each vertex have weight w_i

Select nonadjacent vertices to maximize $\sum_i w_i x_i$

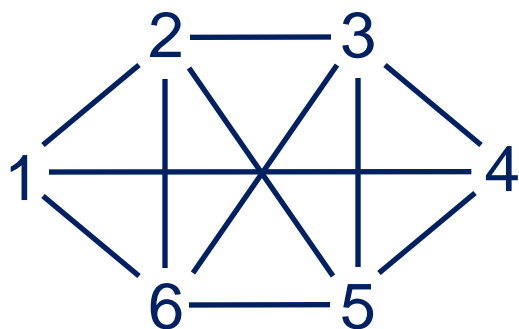




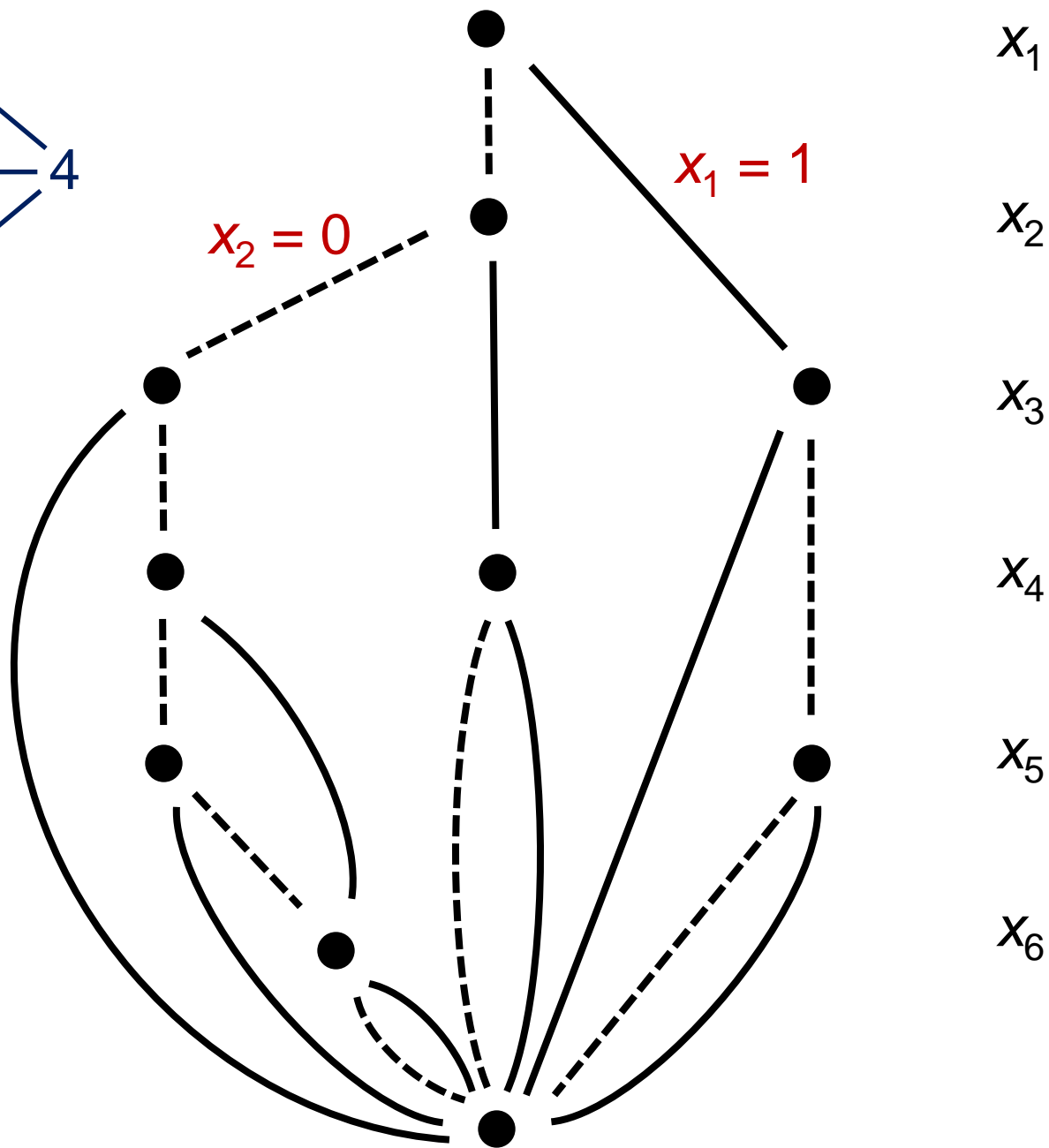
Exact BDD
for
independent
set problem
“zero-suppressed”
BDD

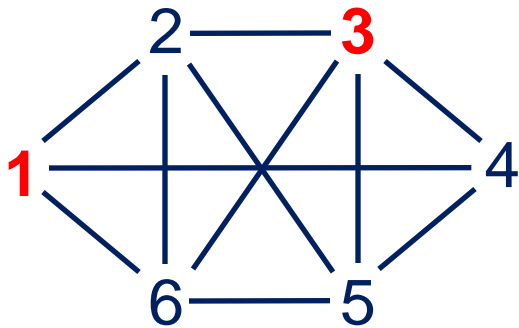


x_1
 x_2
 x_3
 x_4
 x_5
 x_6

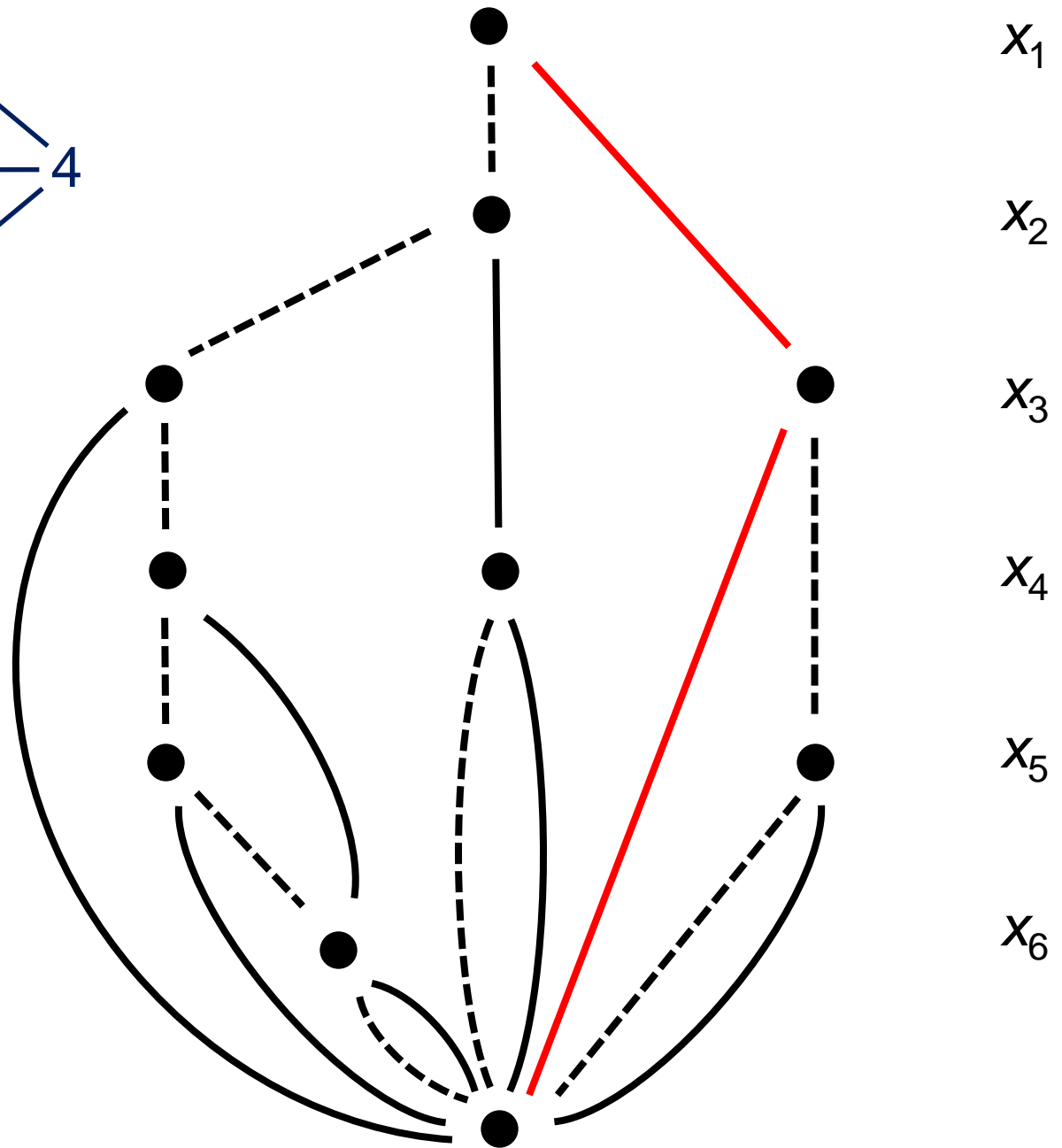


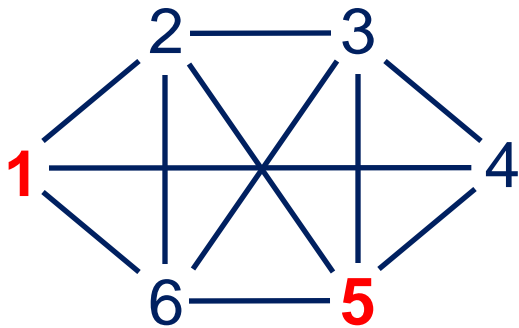
Exact BDD
for
independent
set problem
“zero-suppressed”
BDD



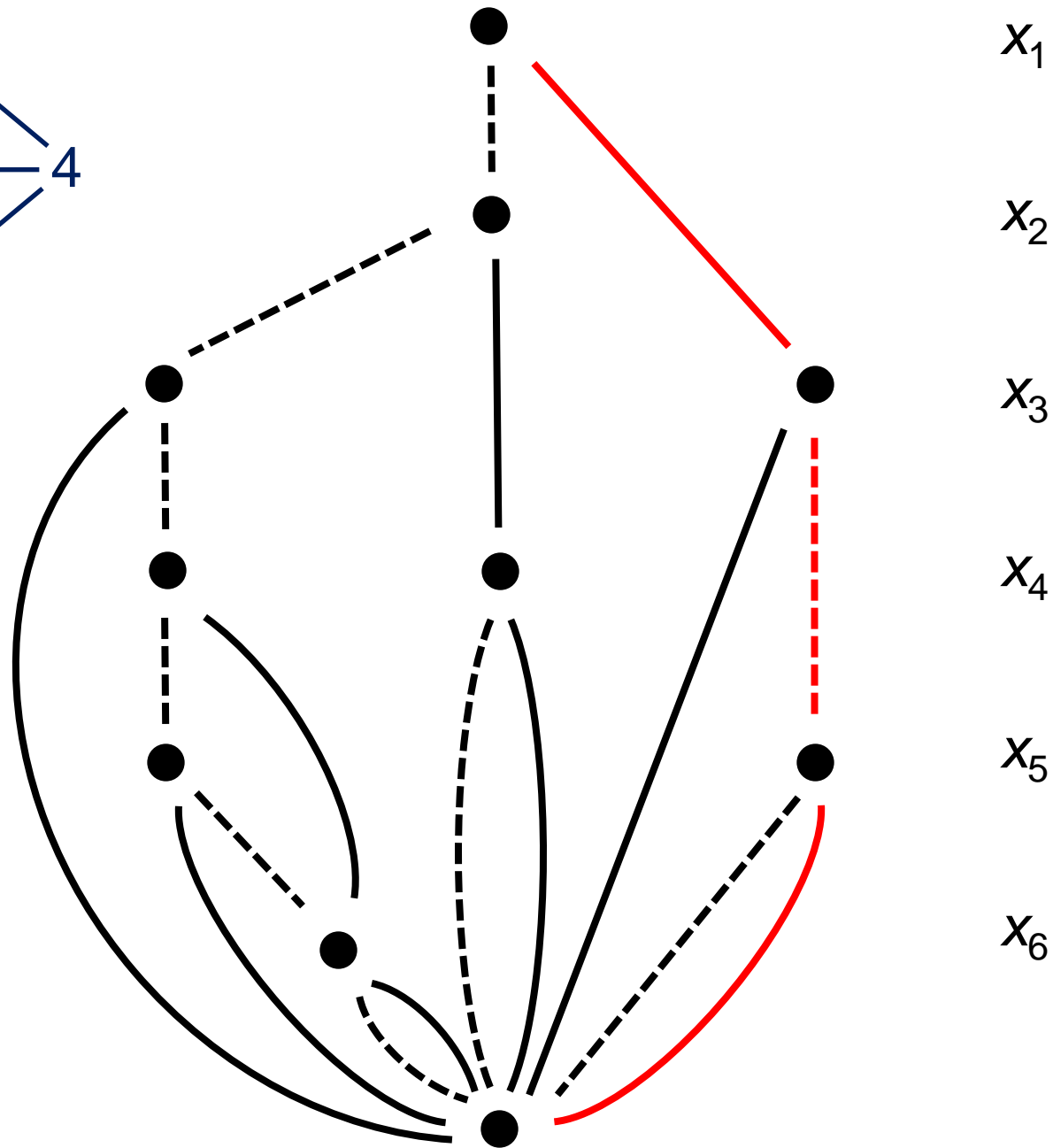


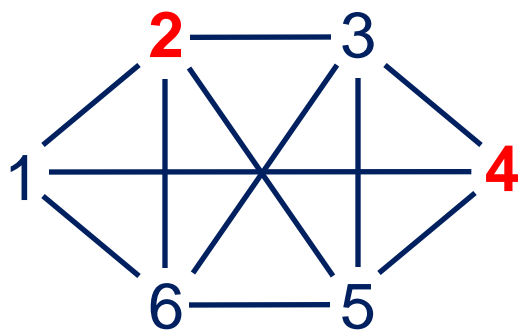
Paths from top to bottom correspond to the 11 feasible solutions



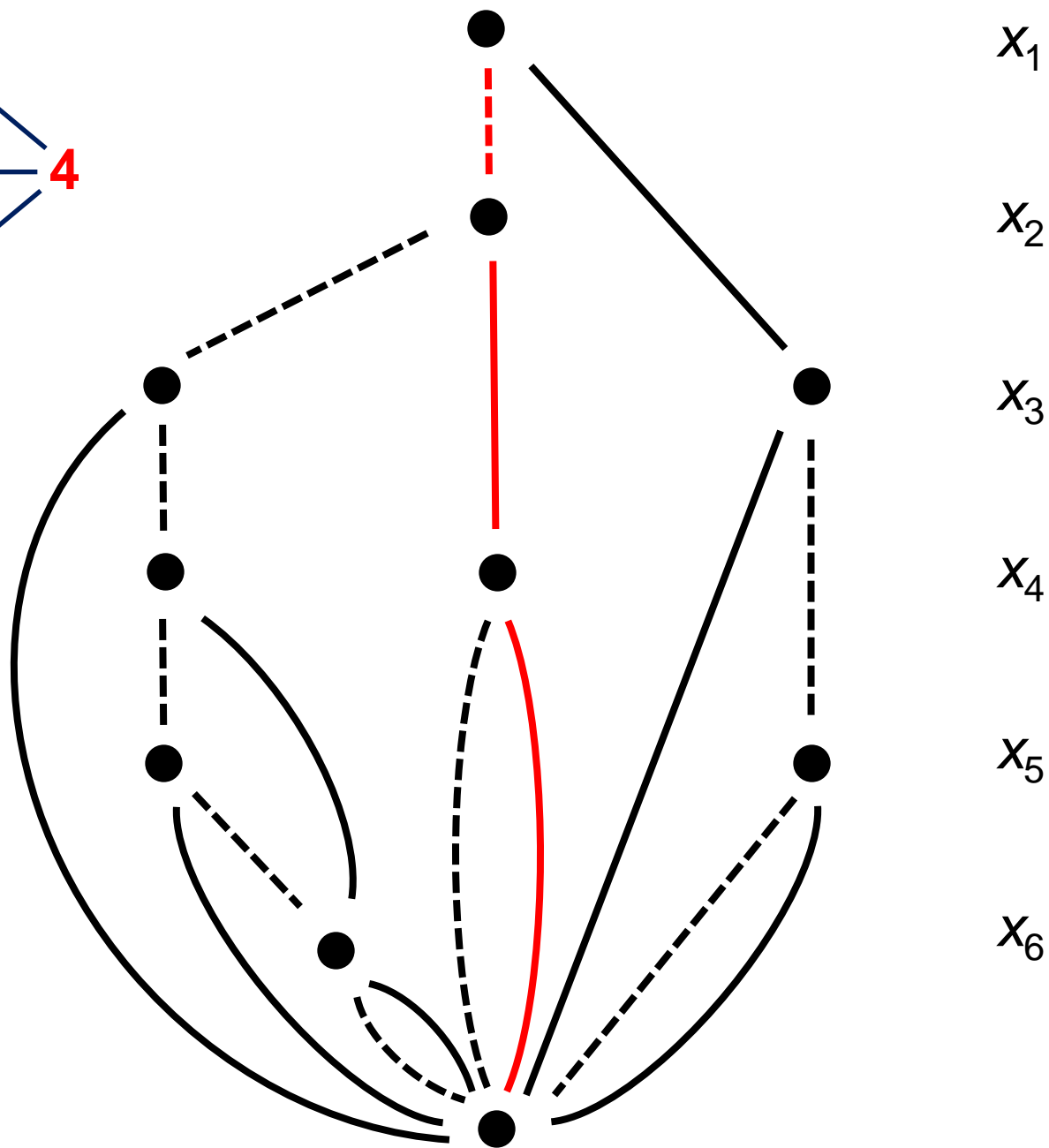


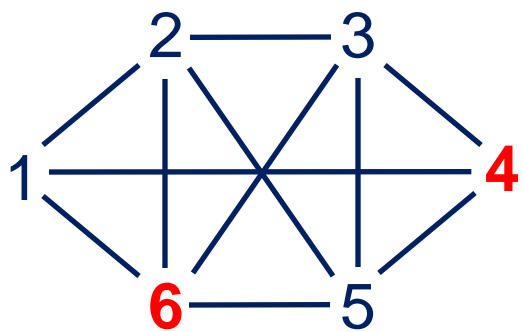
Paths from top to bottom correspond to the 11 feasible solutions



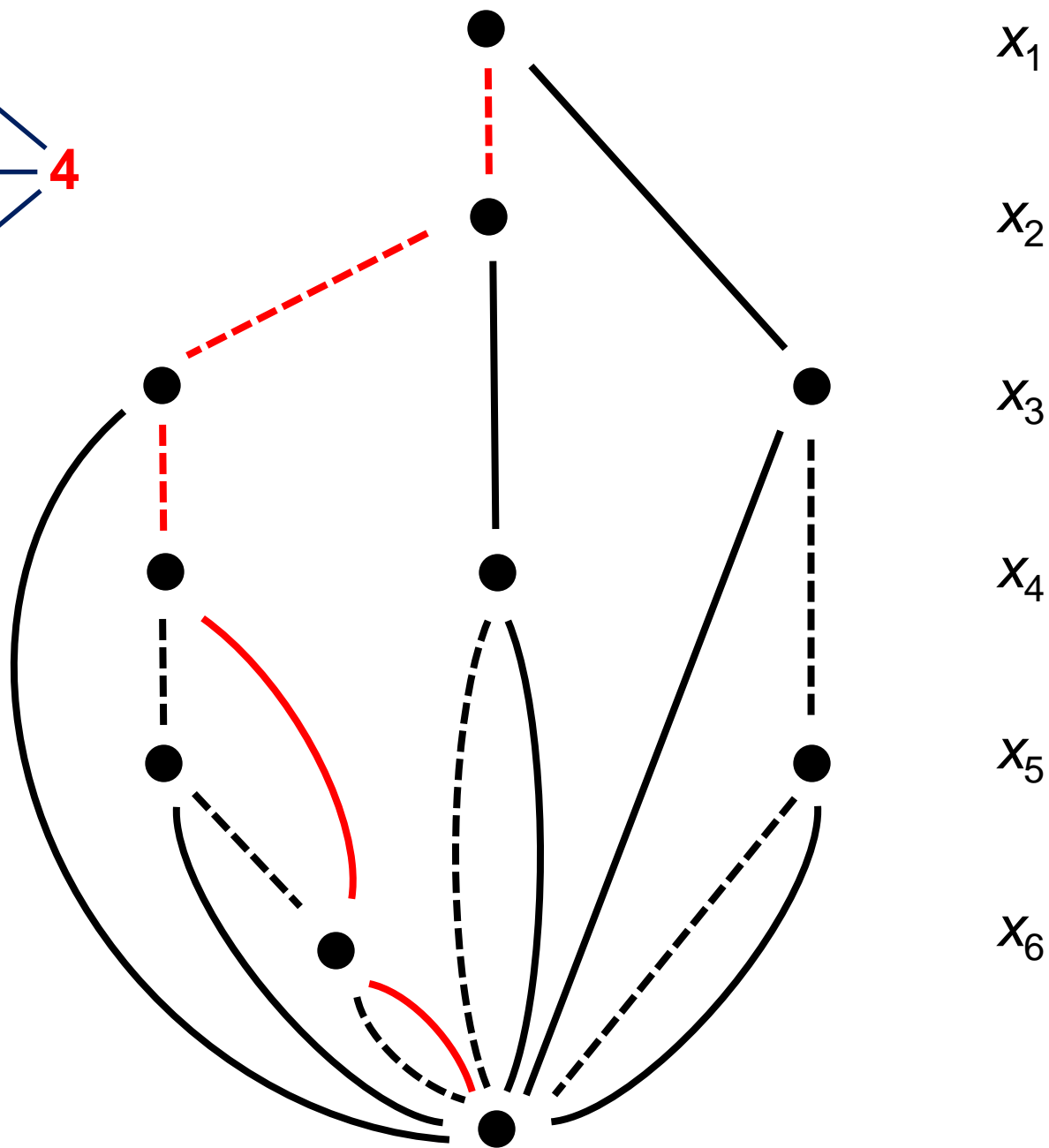


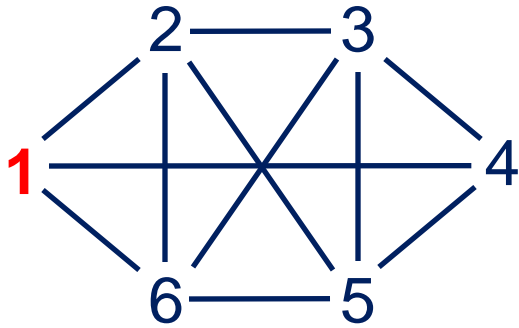
Paths from top to bottom correspond to the 11 feasible solutions





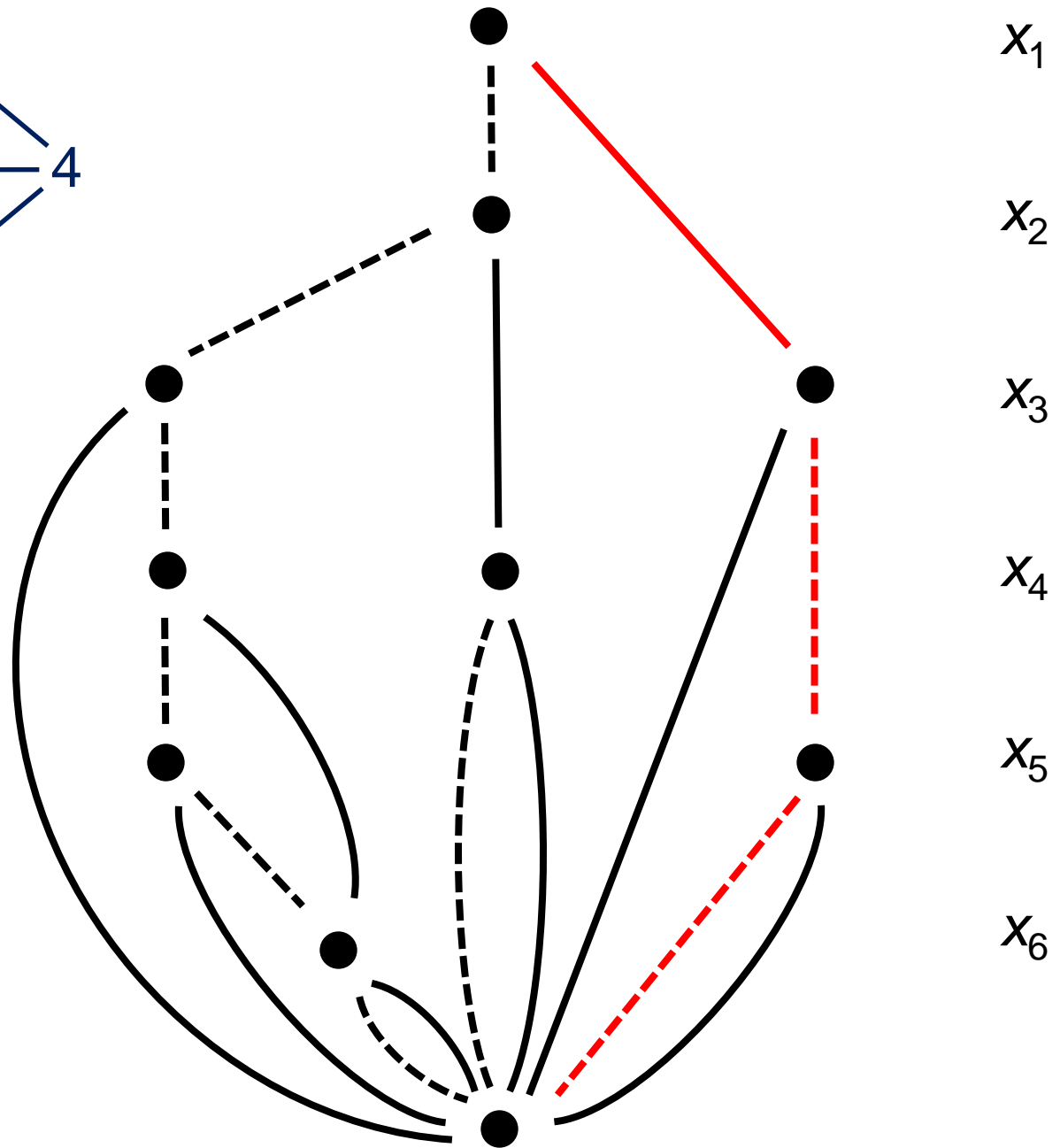
Paths from top to bottom correspond to the 11 feasible solutions

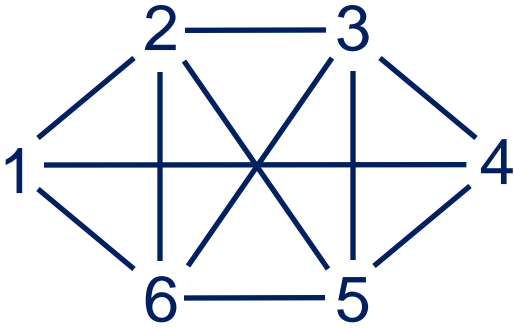




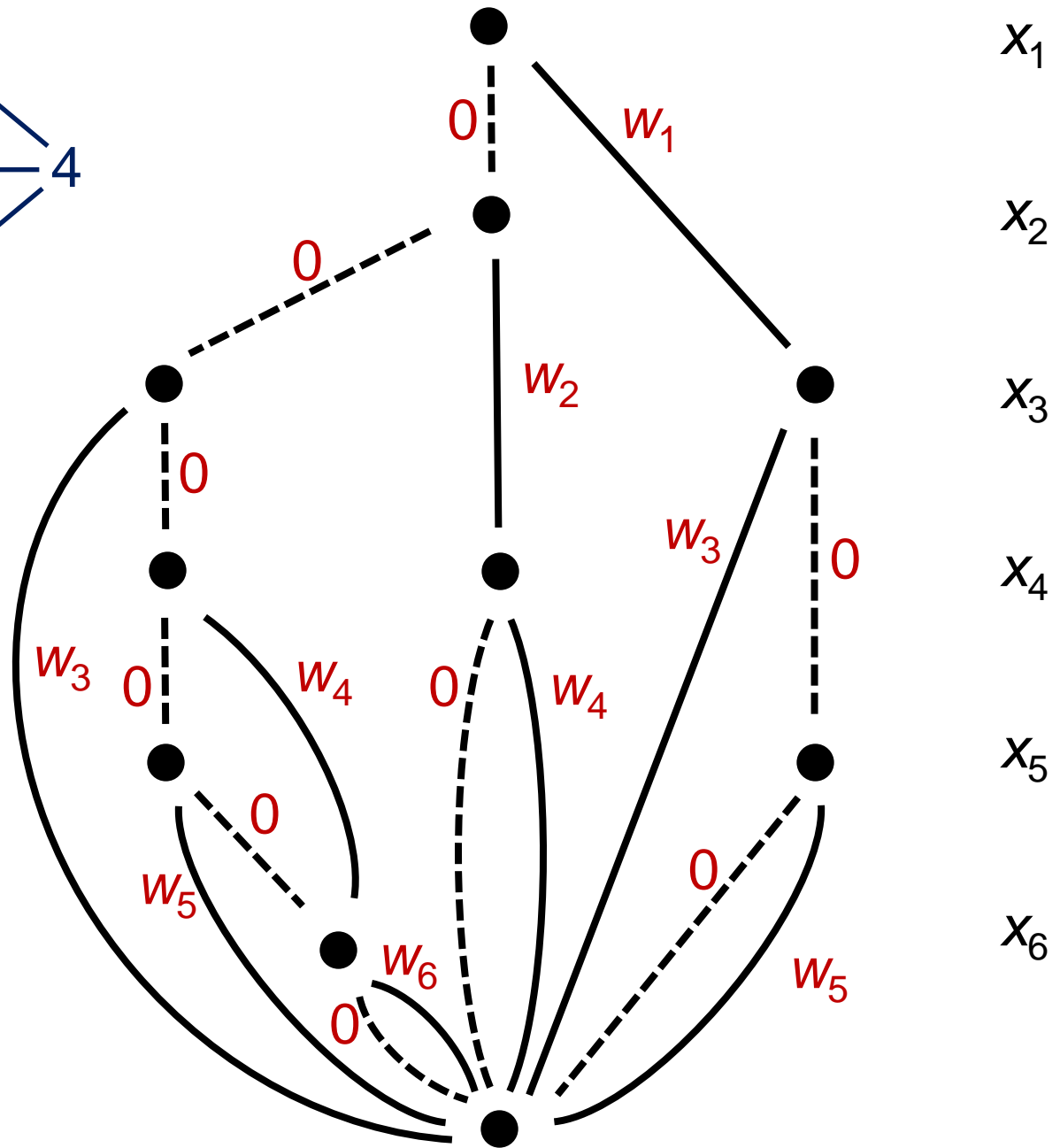
Paths from
top to bottom
correspond to
the 11
feasible
solutions

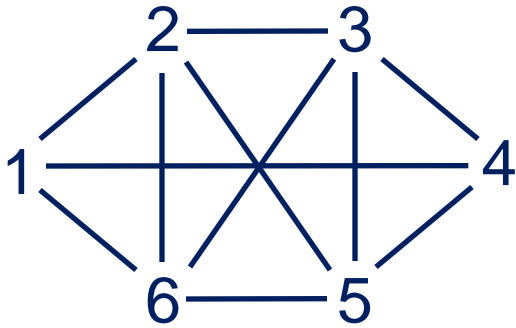
...and so
forth





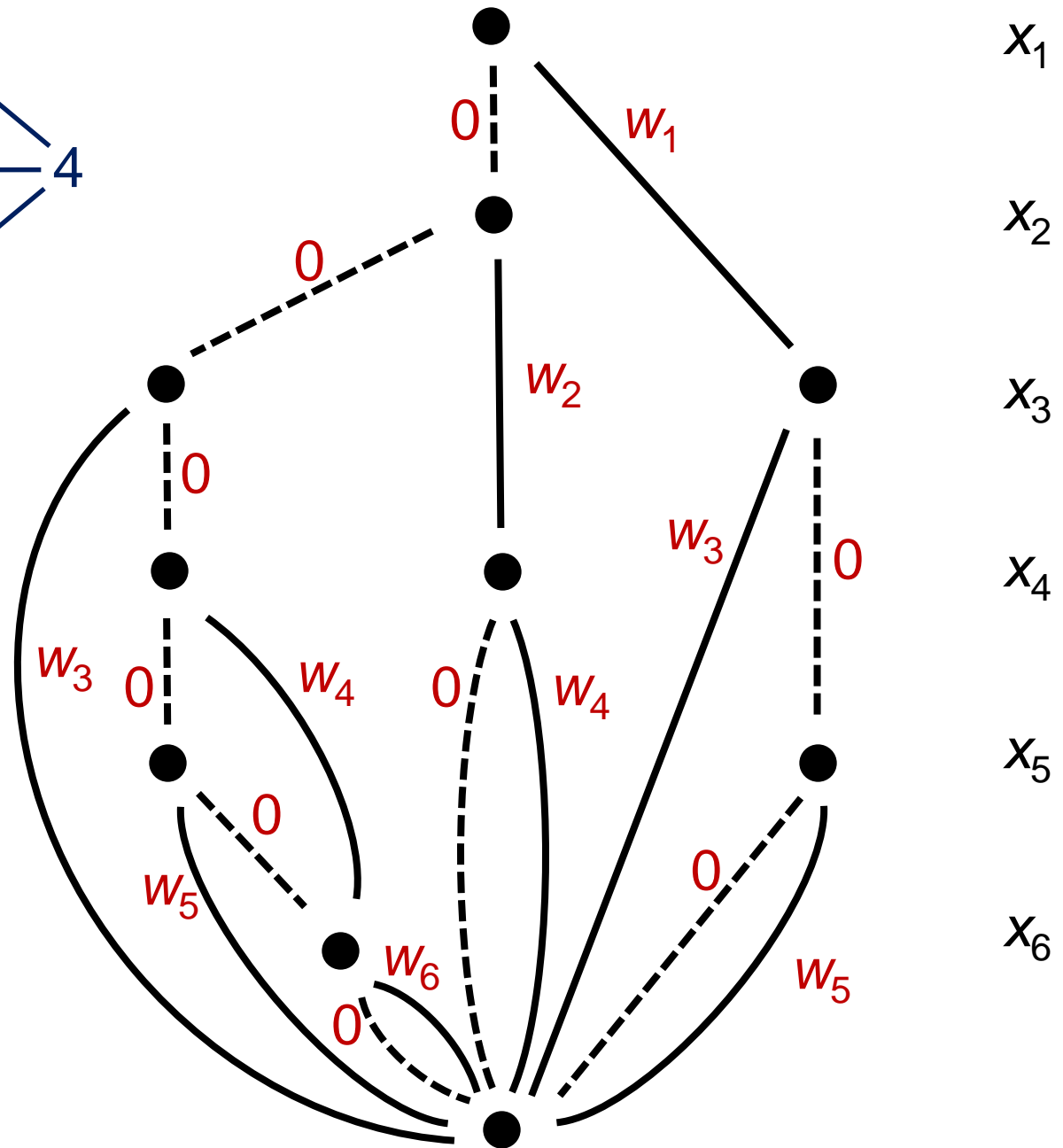
For objective function, associate weights with arcs





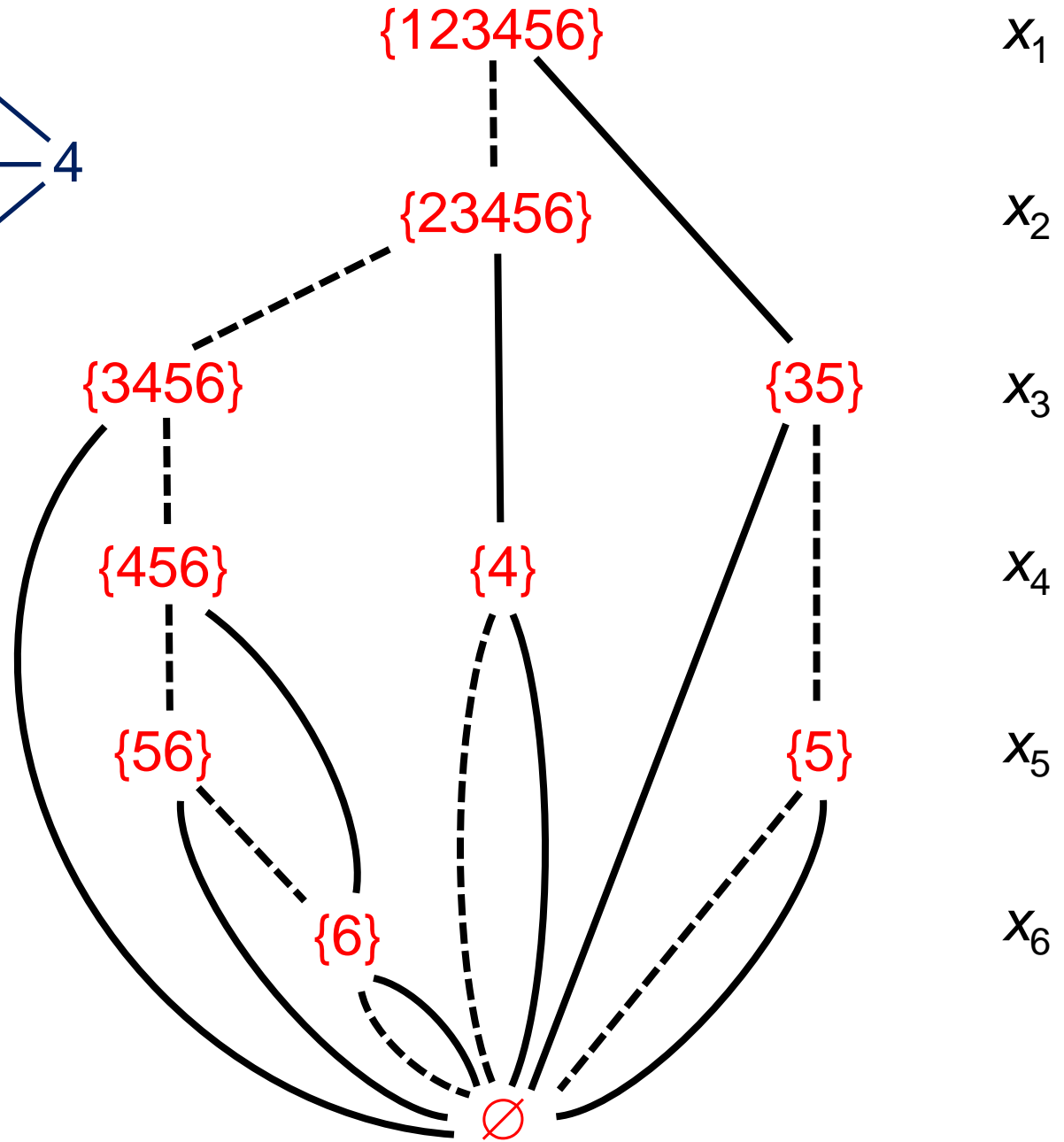
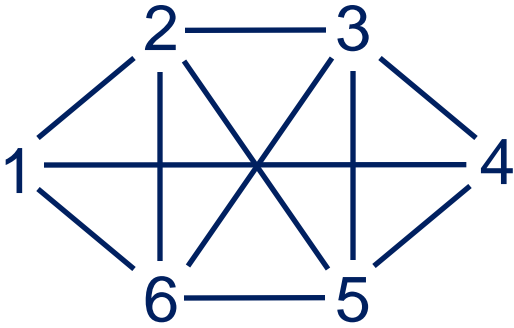
For objective function, associate weights with arcs

Optimal solution is longest path

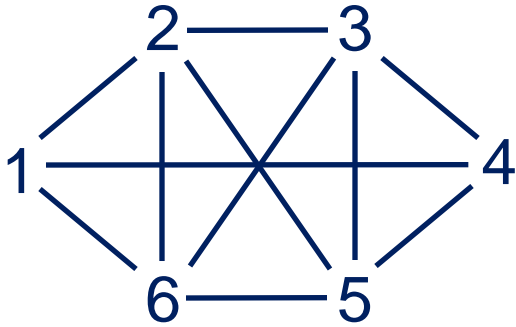


Objective Function

- In general, objective function can be any **separable function**.
 - Linear or nonlinear, convex or nonconvex
- BDDs can be generalized to **nonseparable** objective functions.
 - There is a unique reduced BDD with **canonical** edge costs.



To build BDD,
associate
state with
each node



{123456}

x_1

x_2

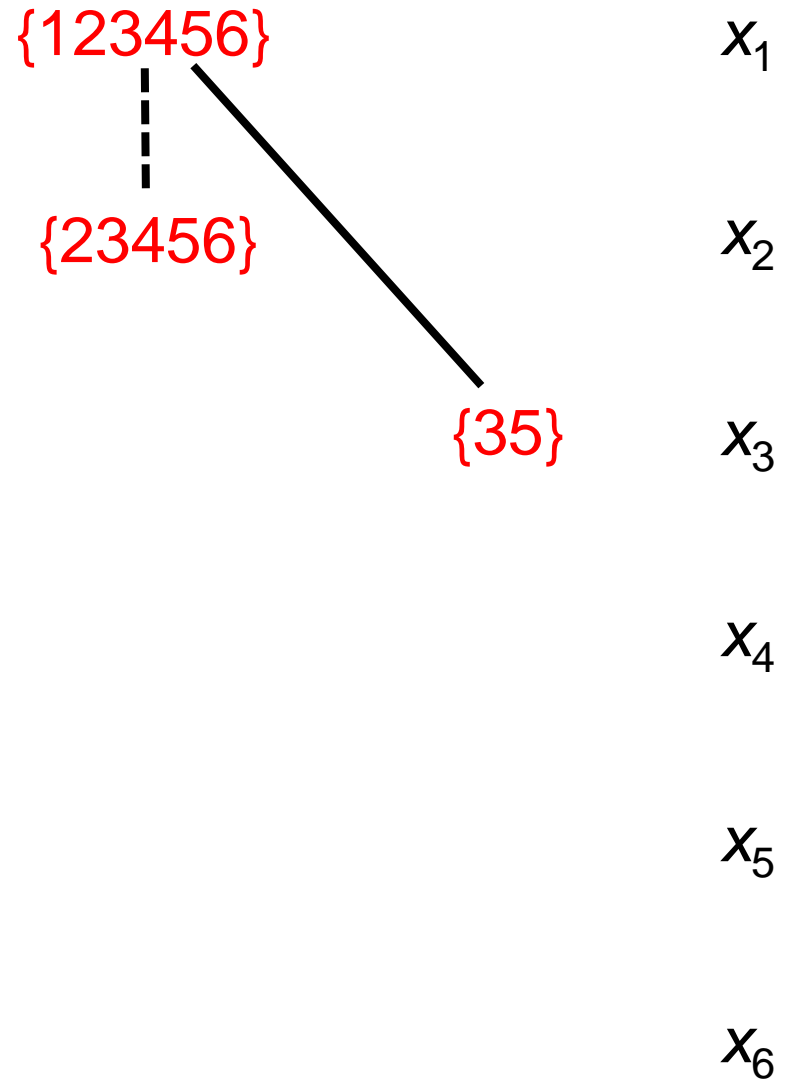
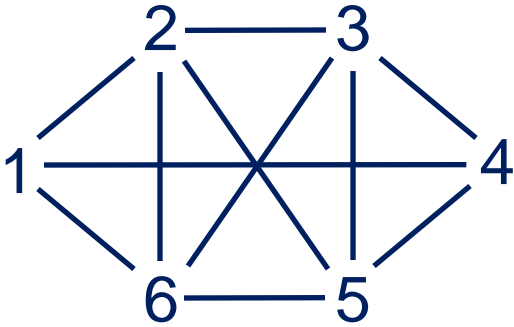
x_3

x_4

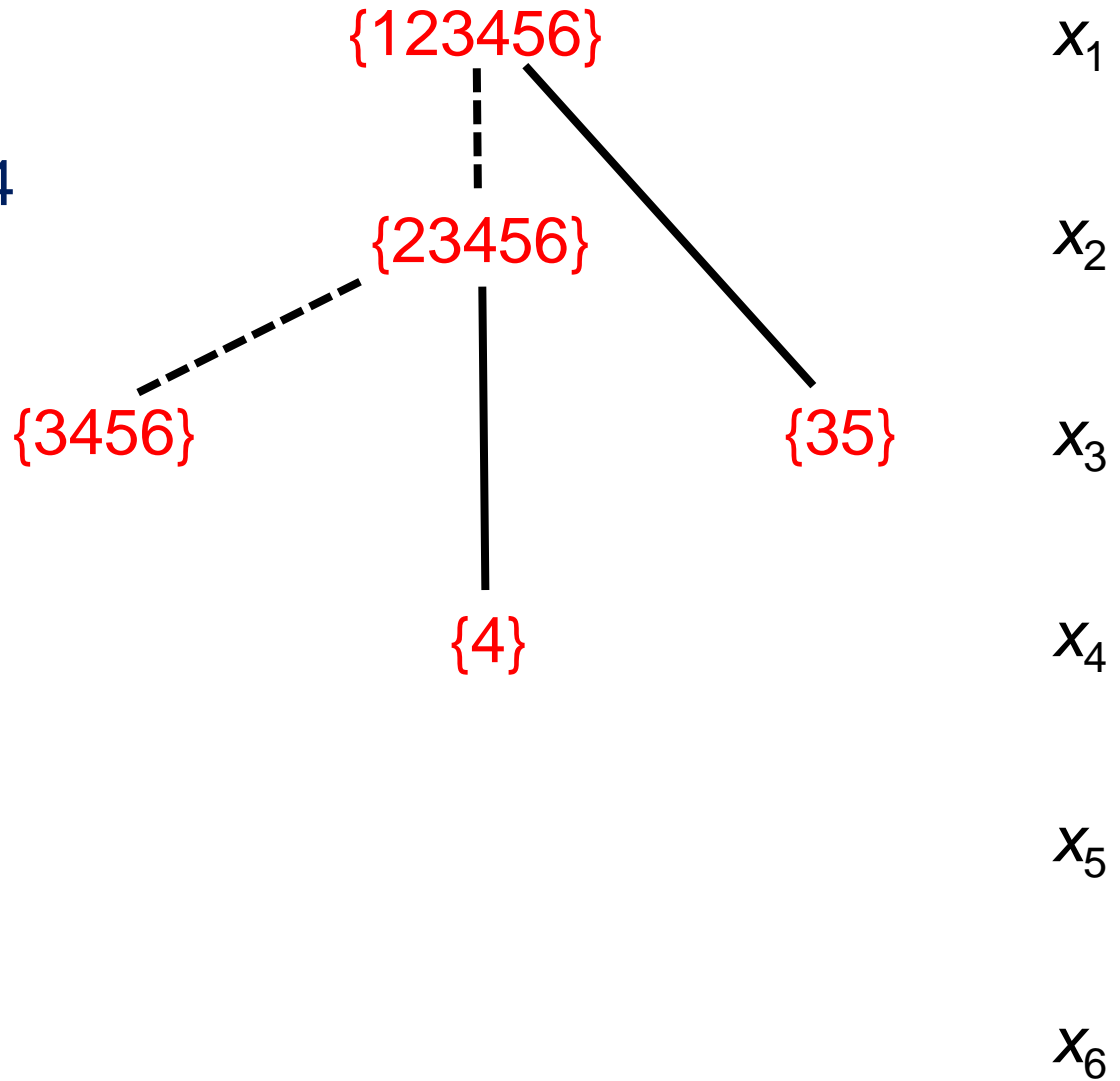
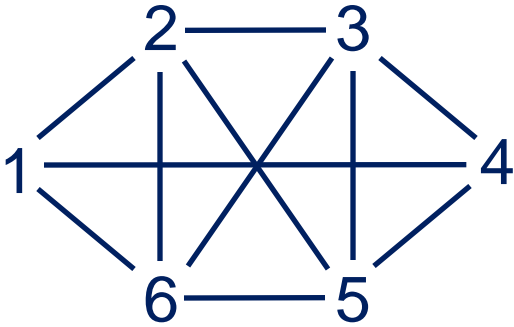
x_5

x_6

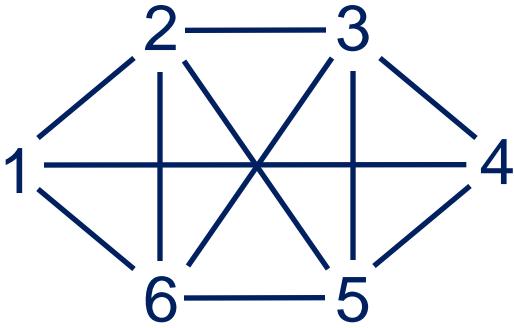
To build BDD,
associate
state with
each node



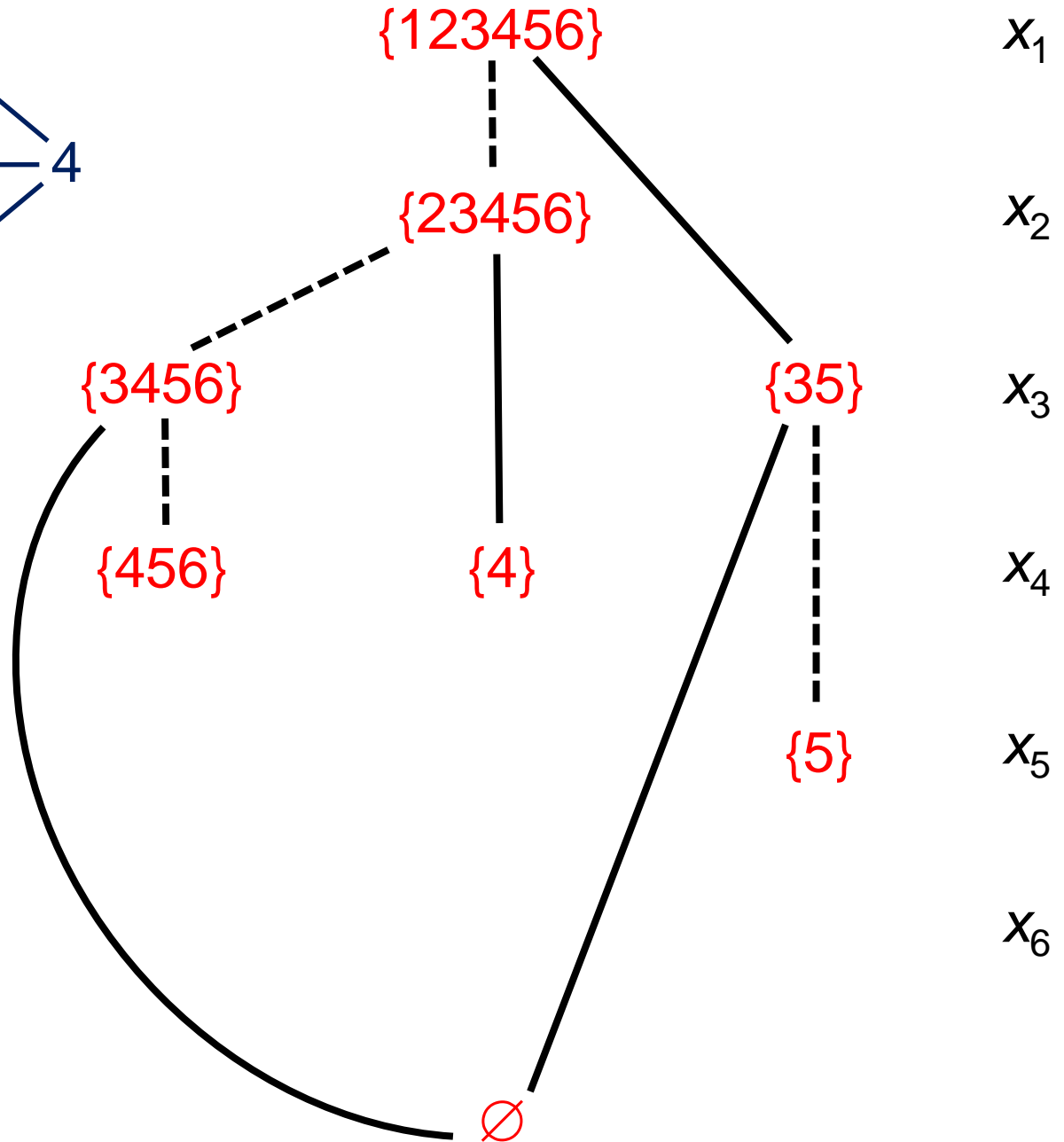
To build BDD,
 associate
state with
 each node

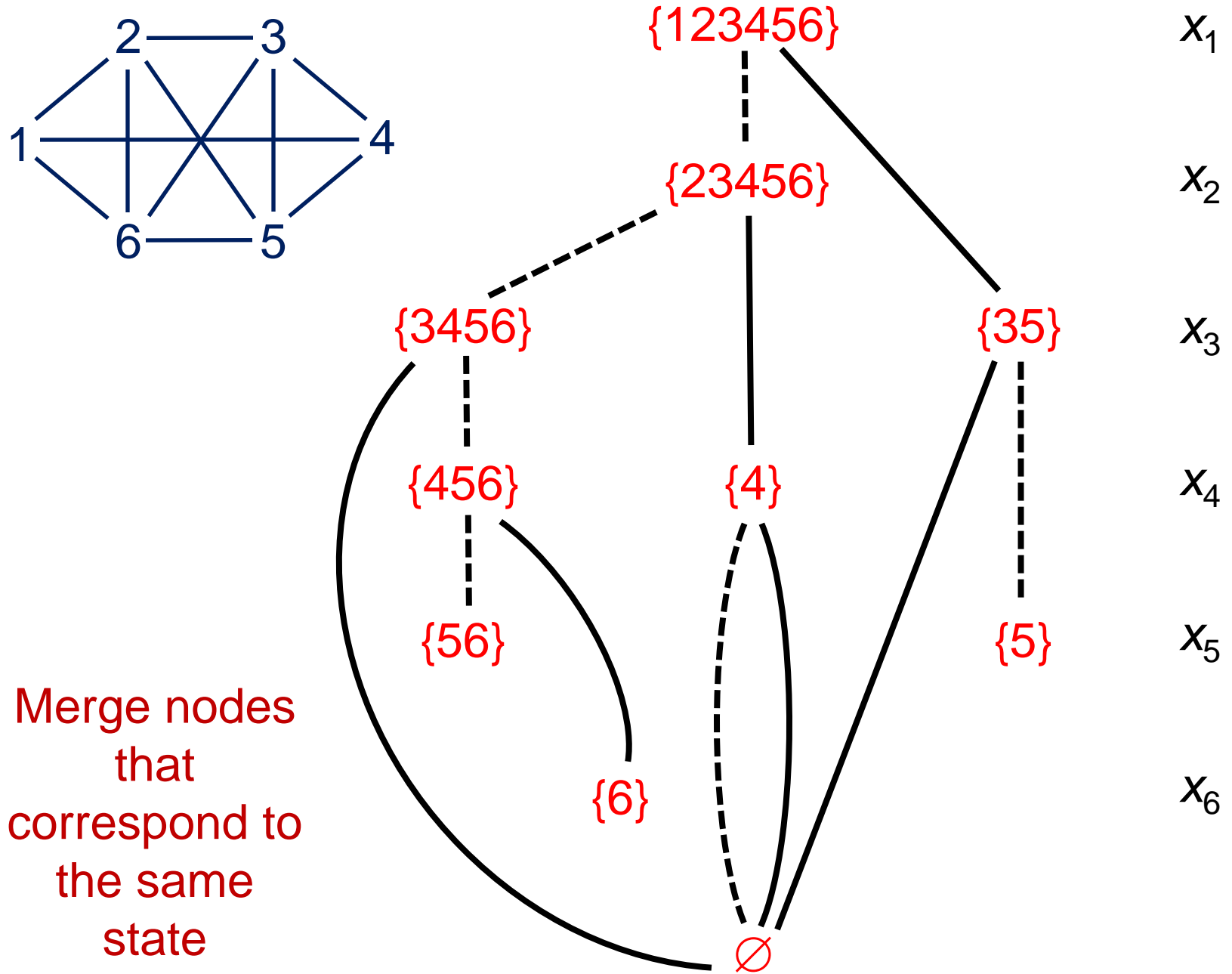
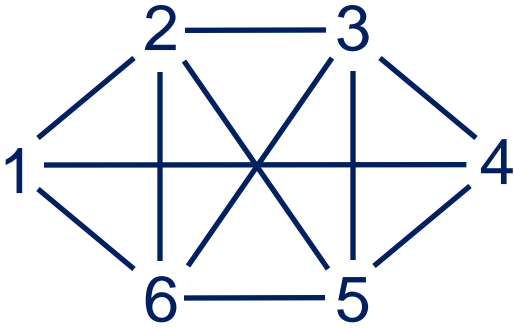


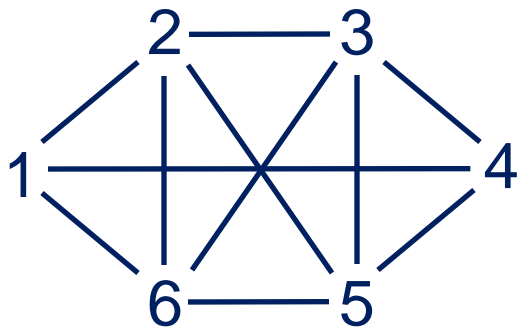
To build BDD,
 associate
state with
 each node



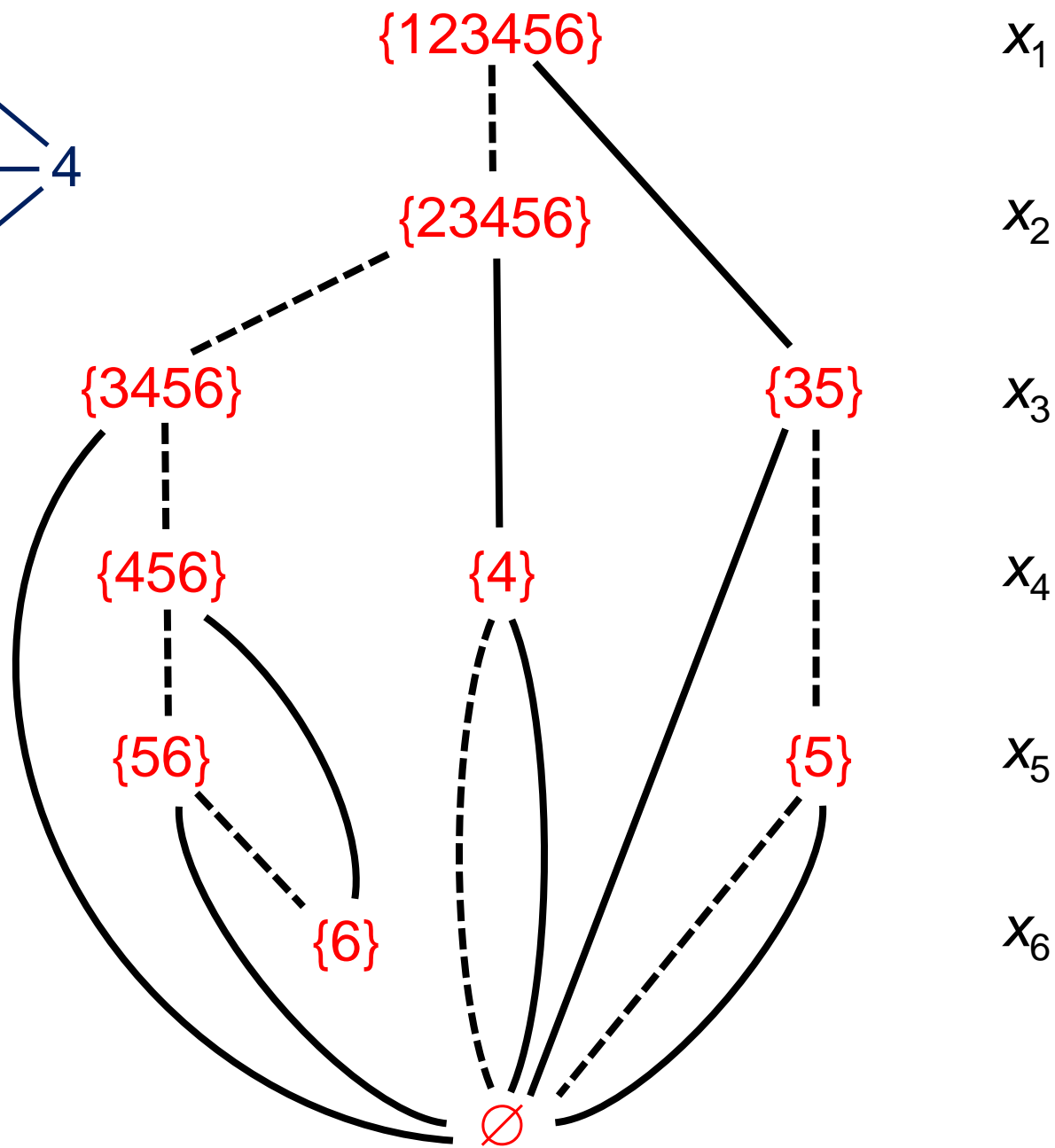
Merge nodes
that
correspond to
the same
state

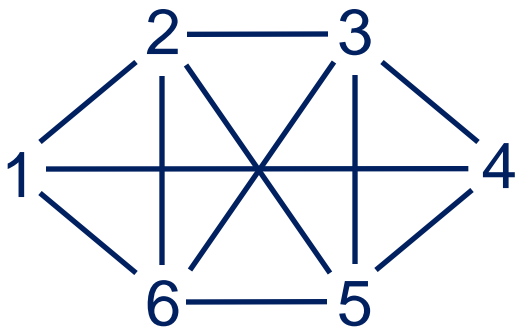






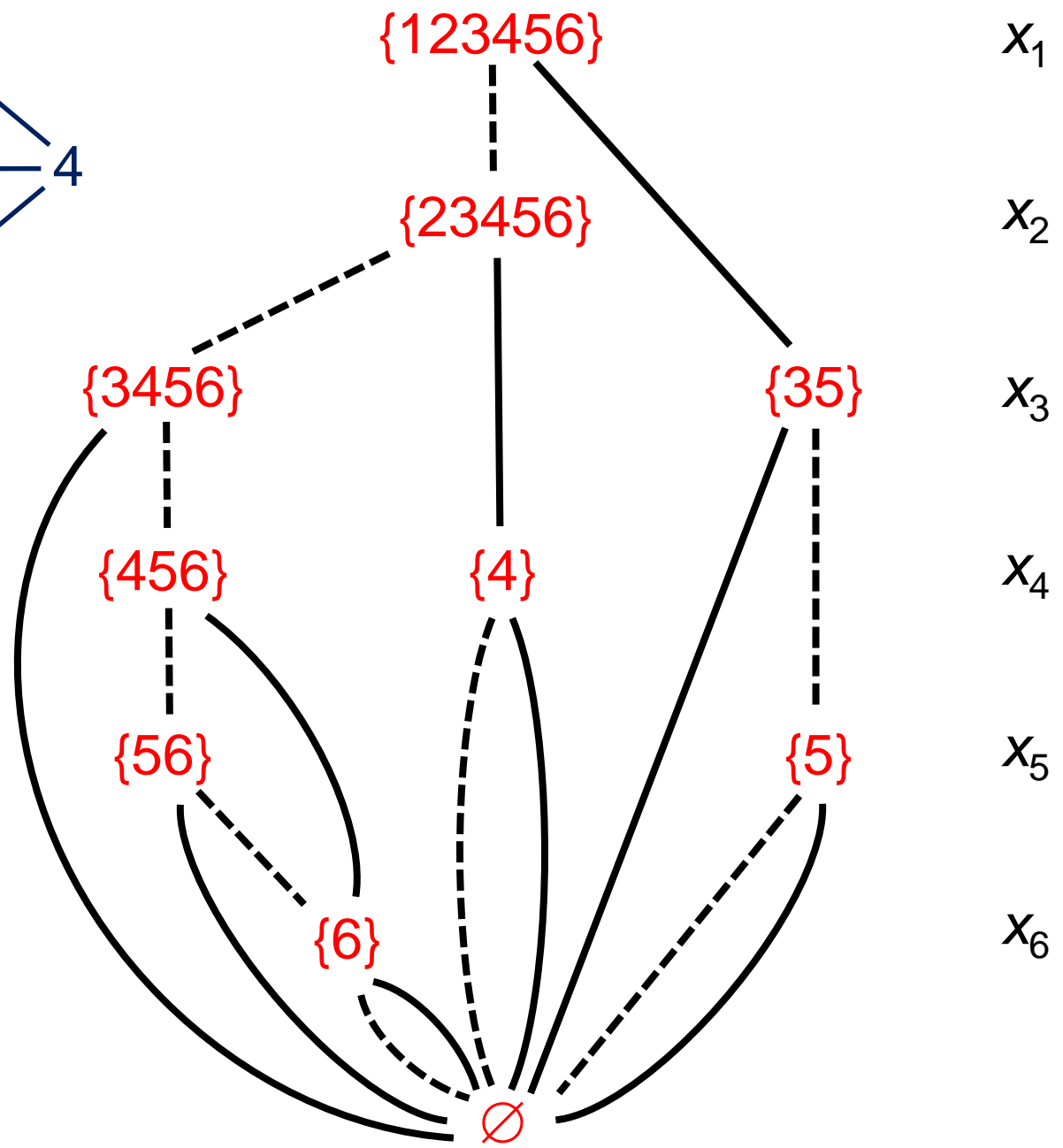
Merge nodes
that
correspond to
the same
state

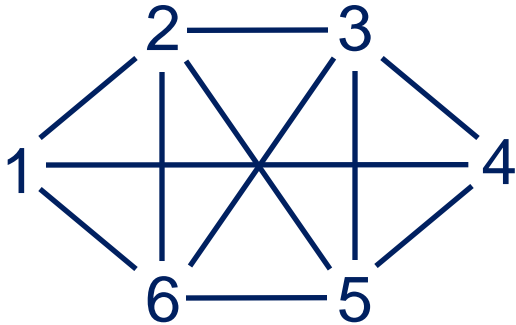




Width = 2

Merge nodes
that
correspond to
the same
state





{123456}

x_1

x_2

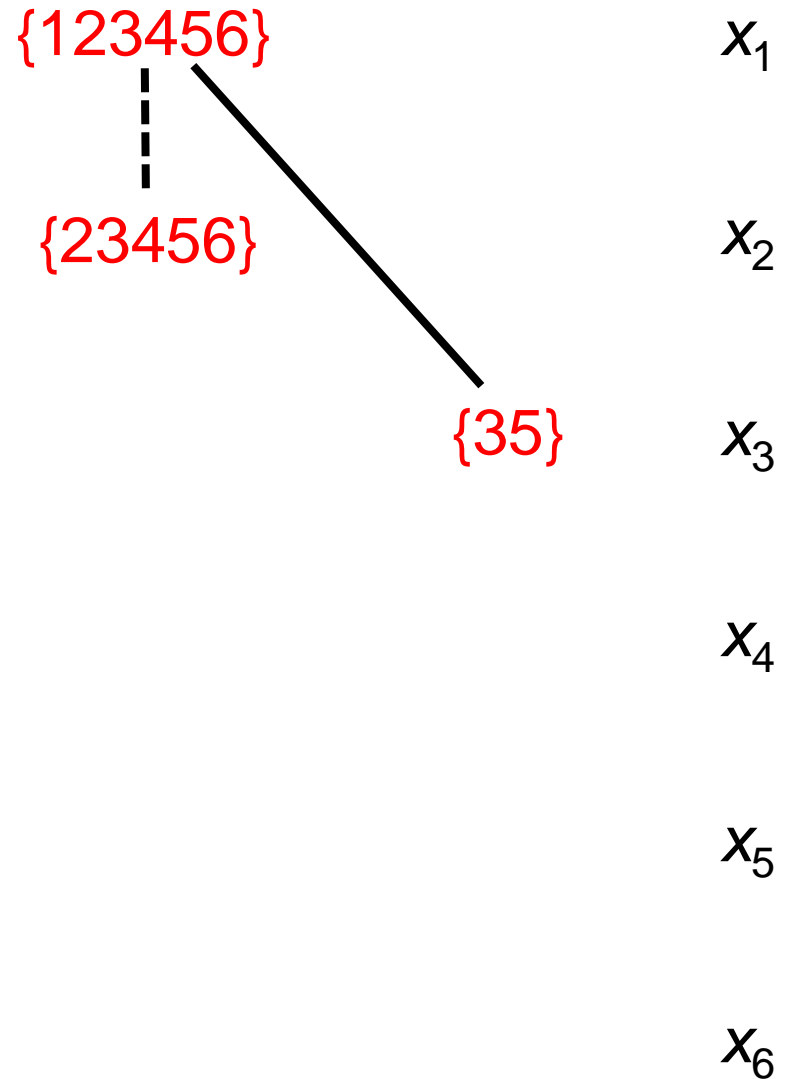
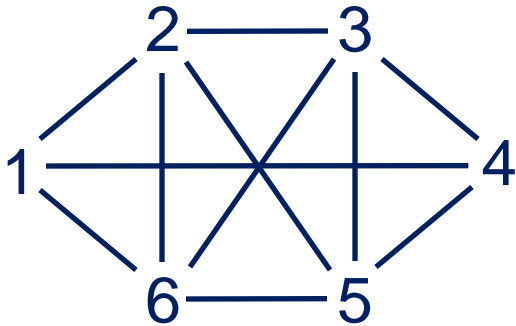
x_3

x_4

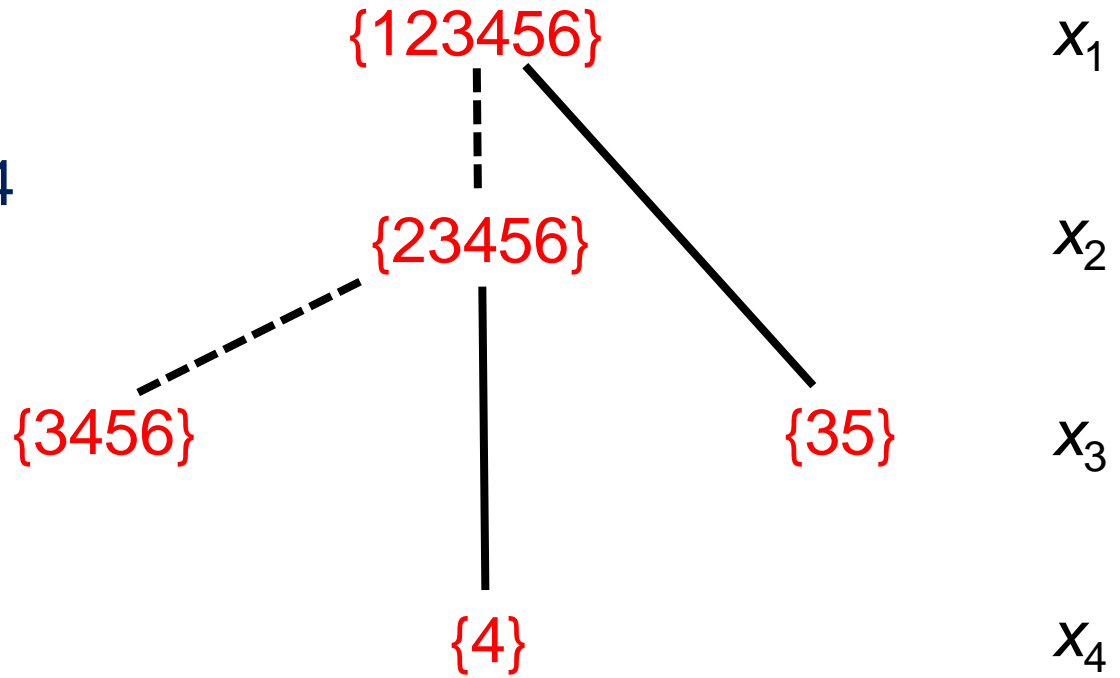
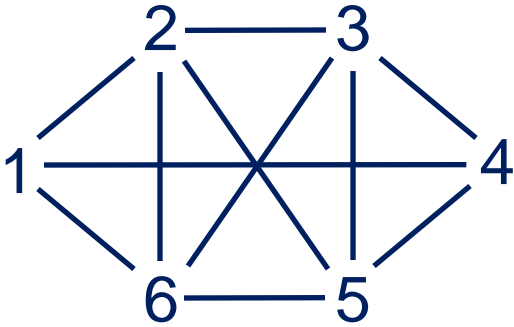
x_5

x_6

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

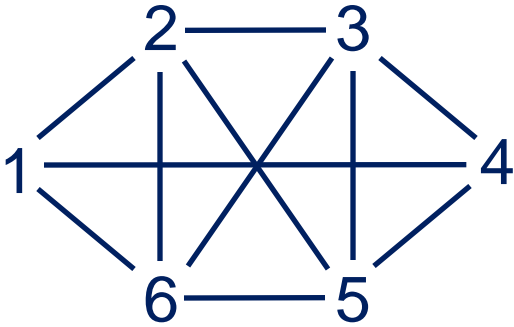


To build
relaxed
 BDD, merge
 some
 additional
 nodes as we
 go along

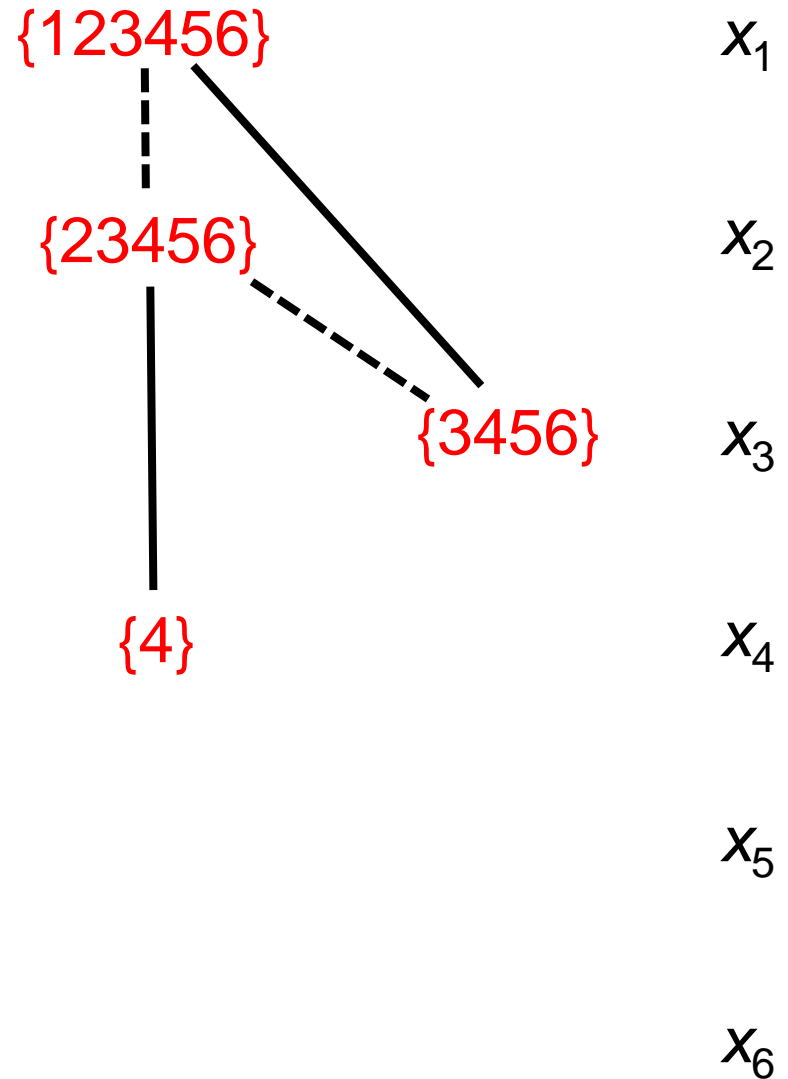


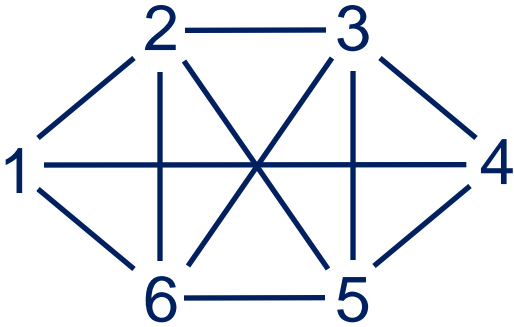
To build
relaxed
 BDD, merge
 some
 additional
 nodes as we
 go along

x_1
 x_2
 x_3
 x_4
 x_5
 x_6

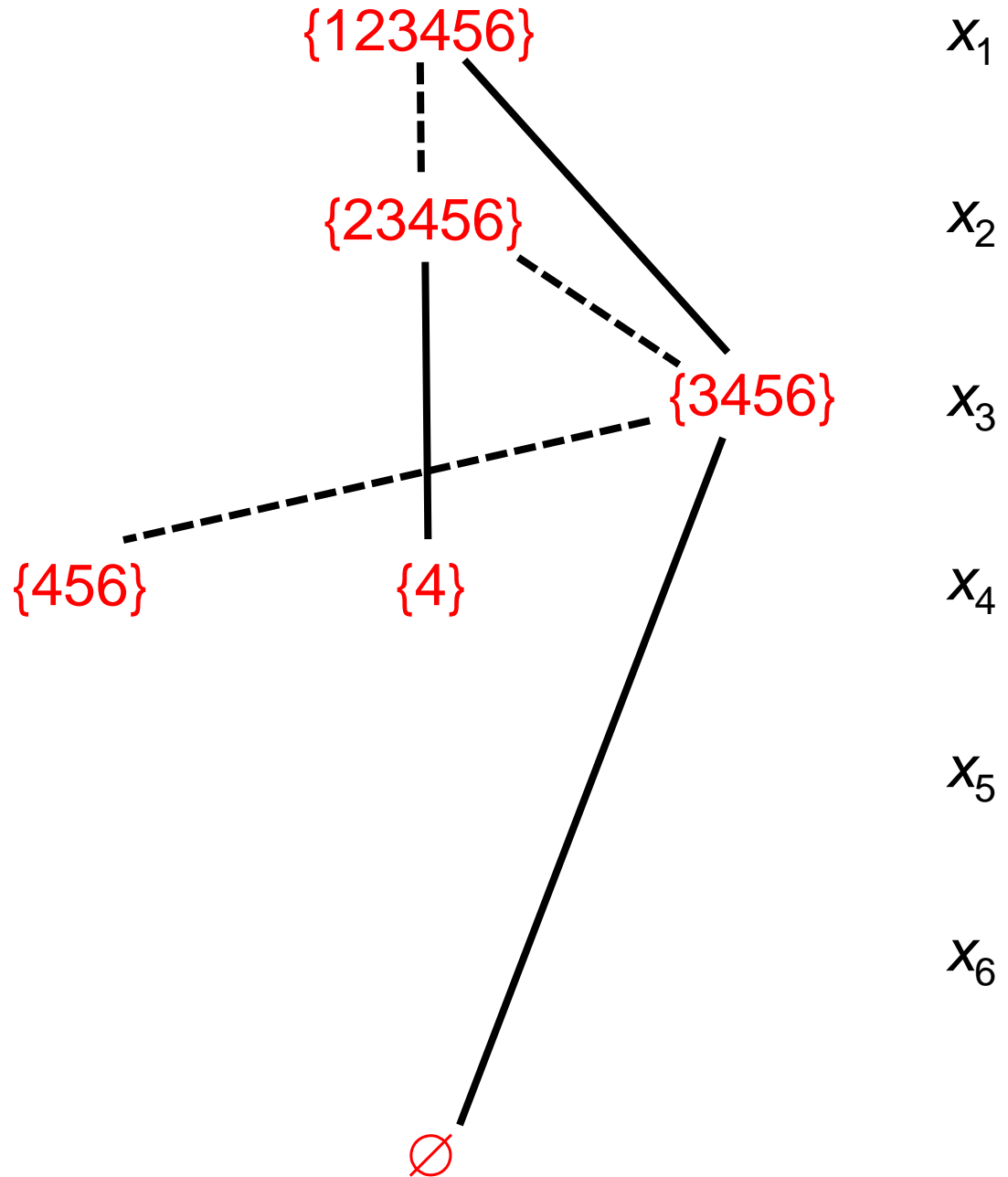


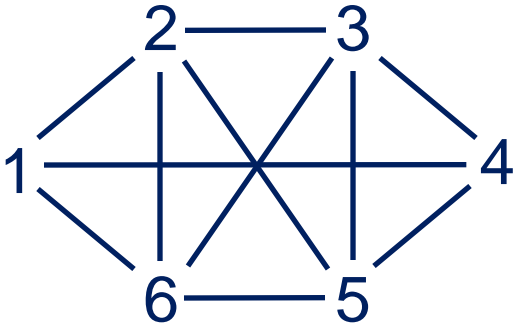
To build
relaxed
 BDD, merge
 some
 additional
 nodes as we
 go along



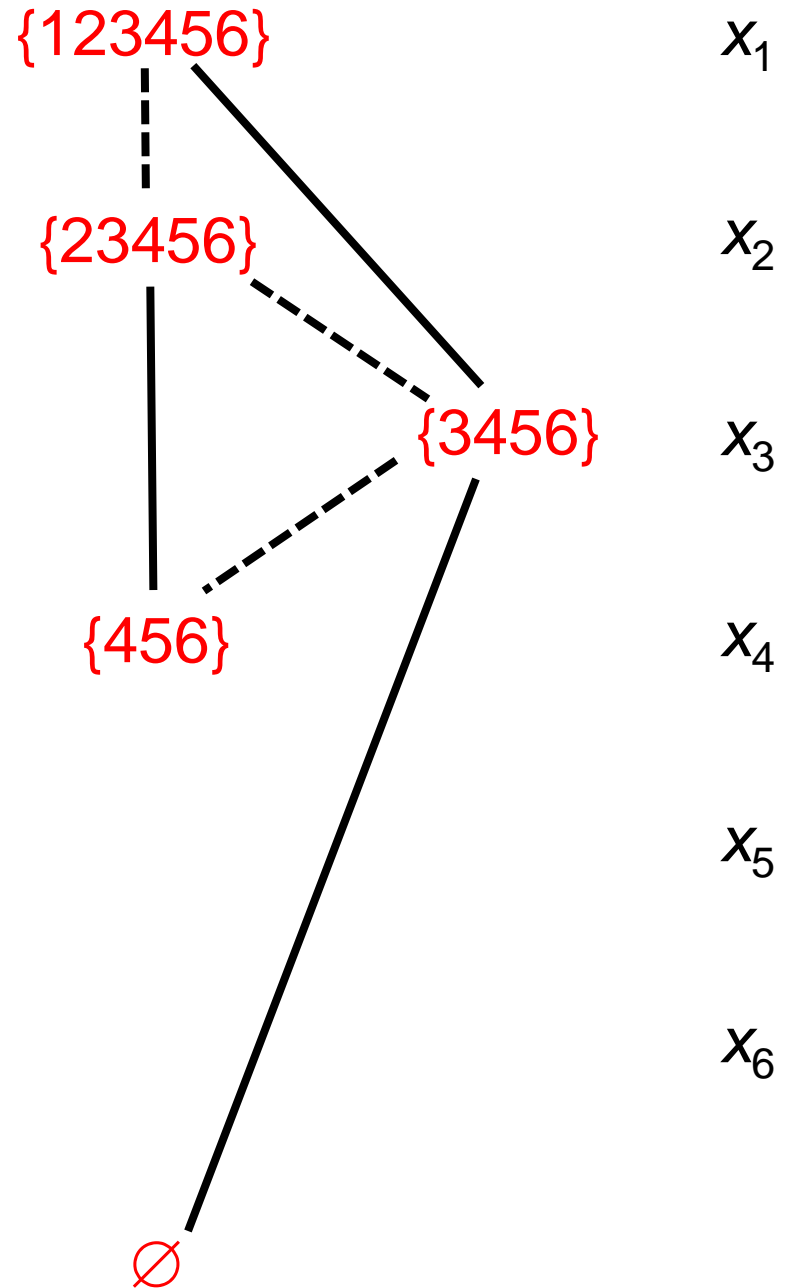


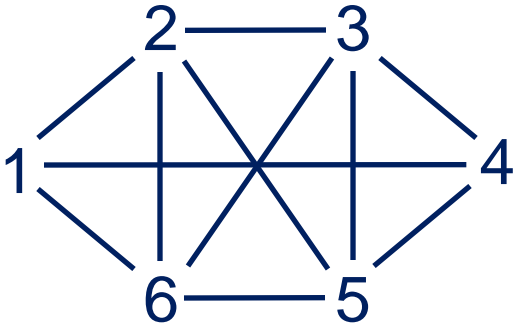
To build
relaxed
 BDD, merge
 some
 additional
 nodes as we
 go along



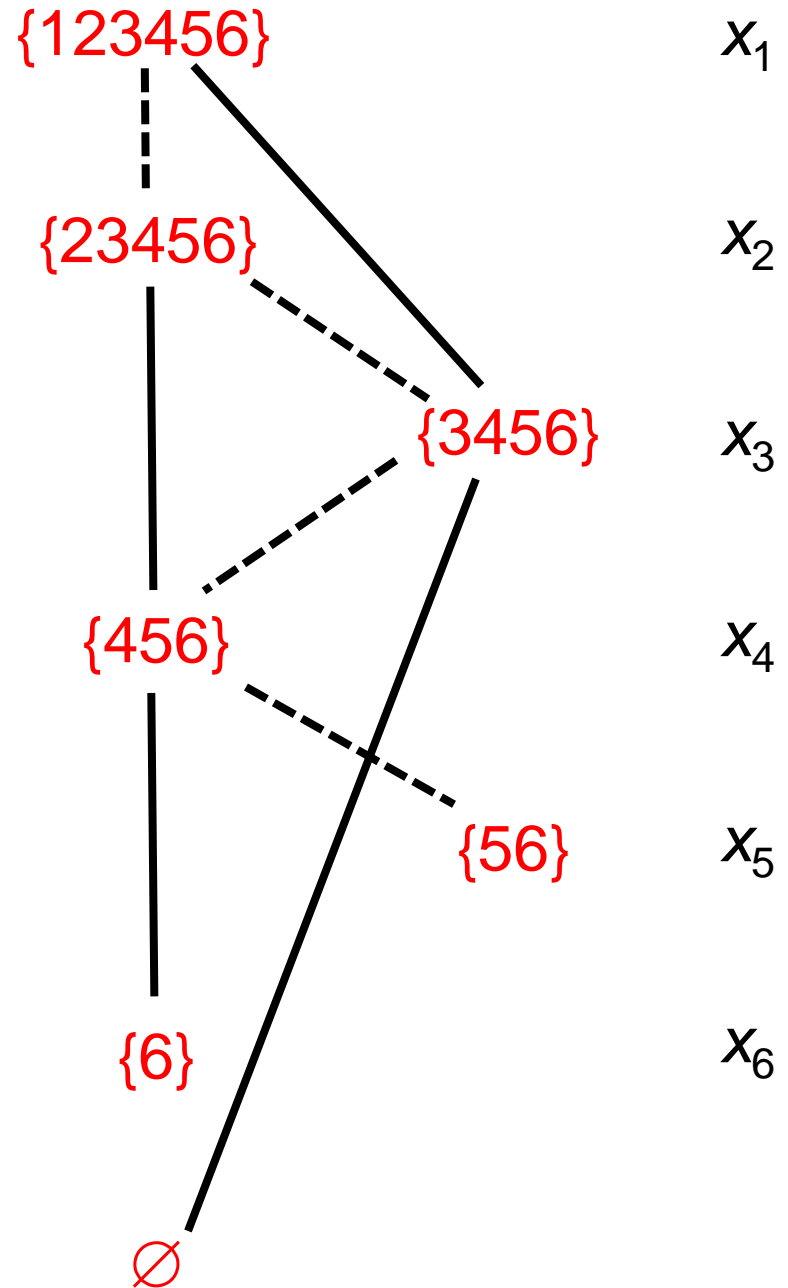


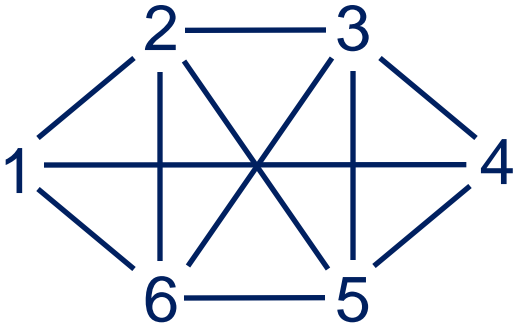
To build
relaxed
 BDD, merge
 some
 additional
 nodes as we
 go along



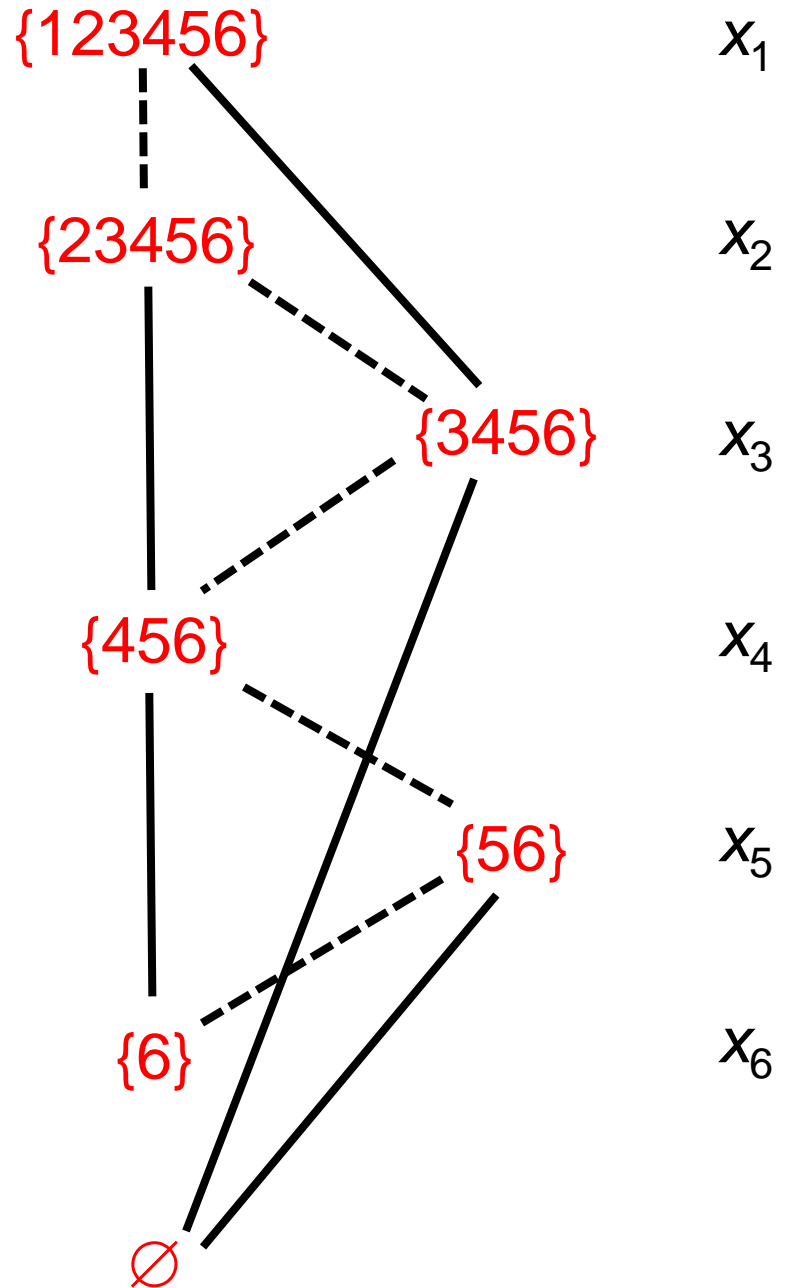


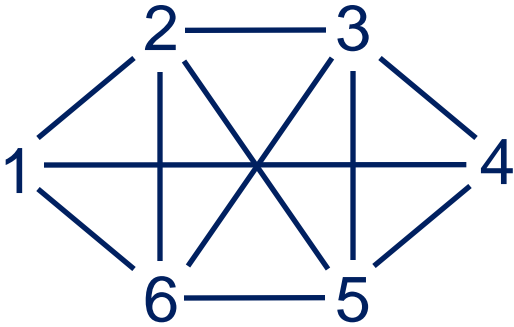
To build
relaxed
 BDD, merge
 some
 additional
 nodes as we
 go along





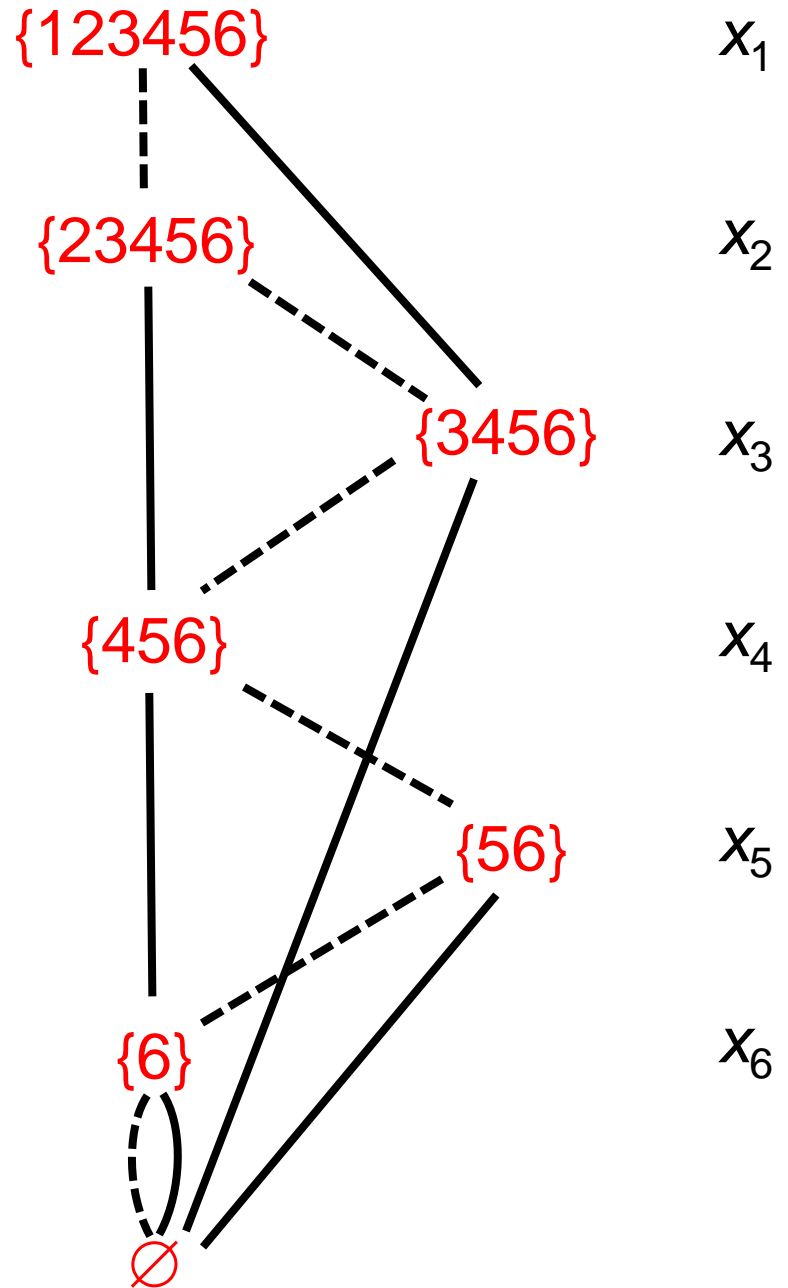
To build
relaxed
 BDD, merge
 some
 additional
 nodes as we
 go along

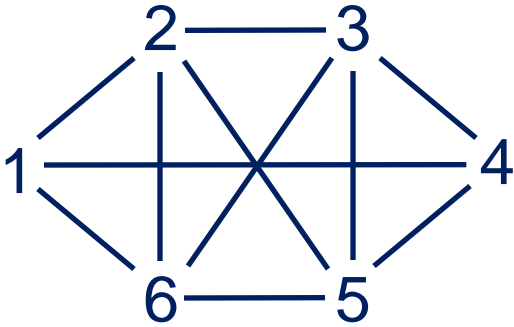




Width = 1

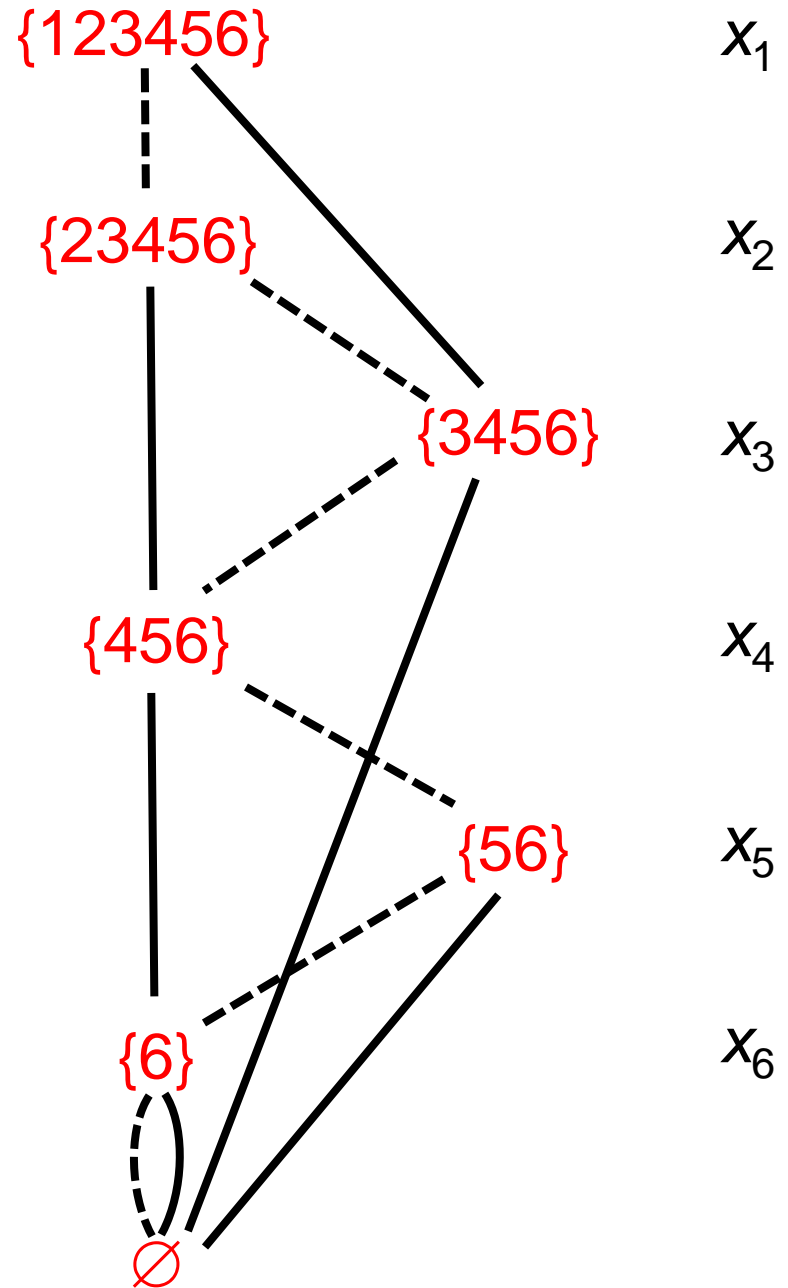
To build
relaxed
 BDD, merge
 some
 additional
 nodes as we
 go along

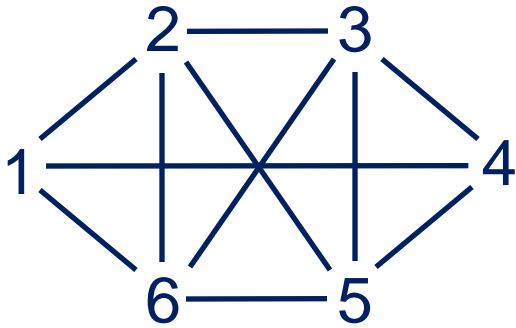




Width = 1

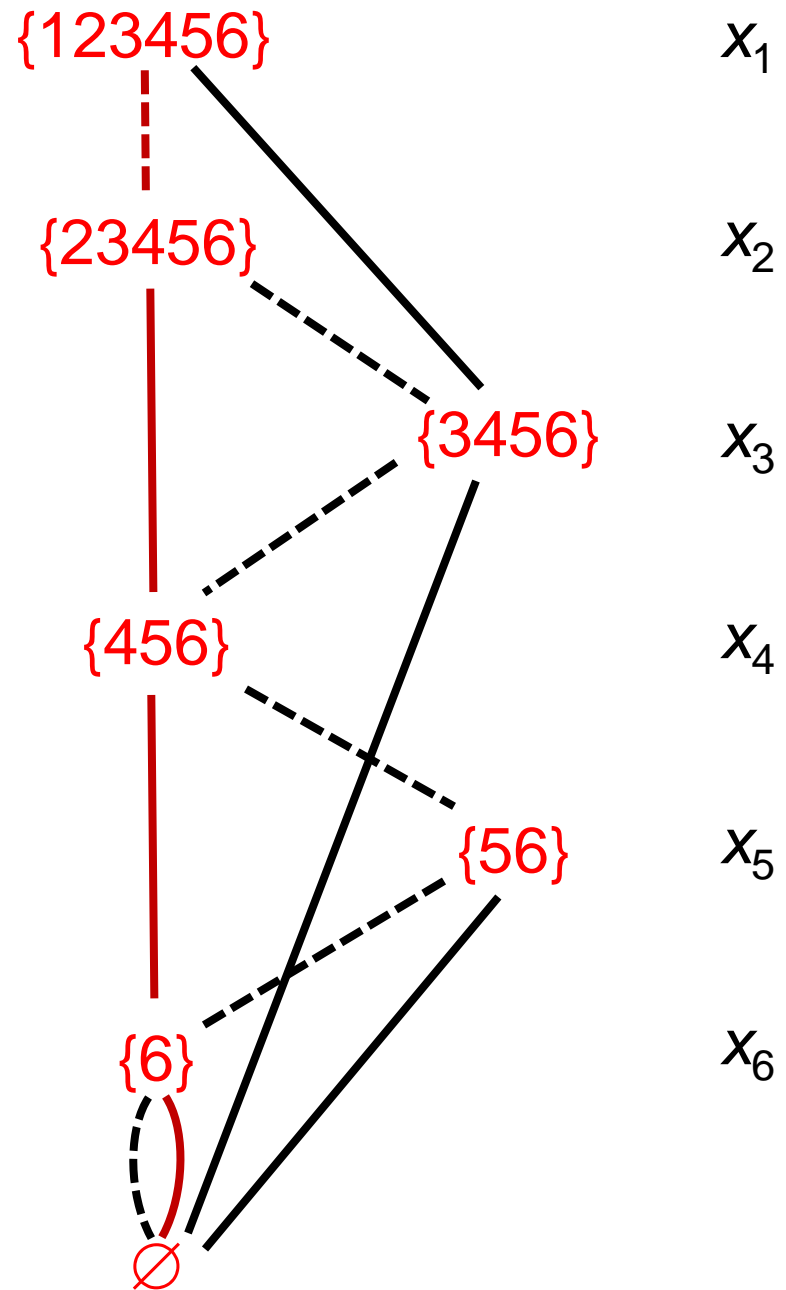
Represents
18 solutions,
including 11
feasible
solutions





Width = 1

Longest path
gives bound
of 3 on optimal
value of 2



Variable Ordering

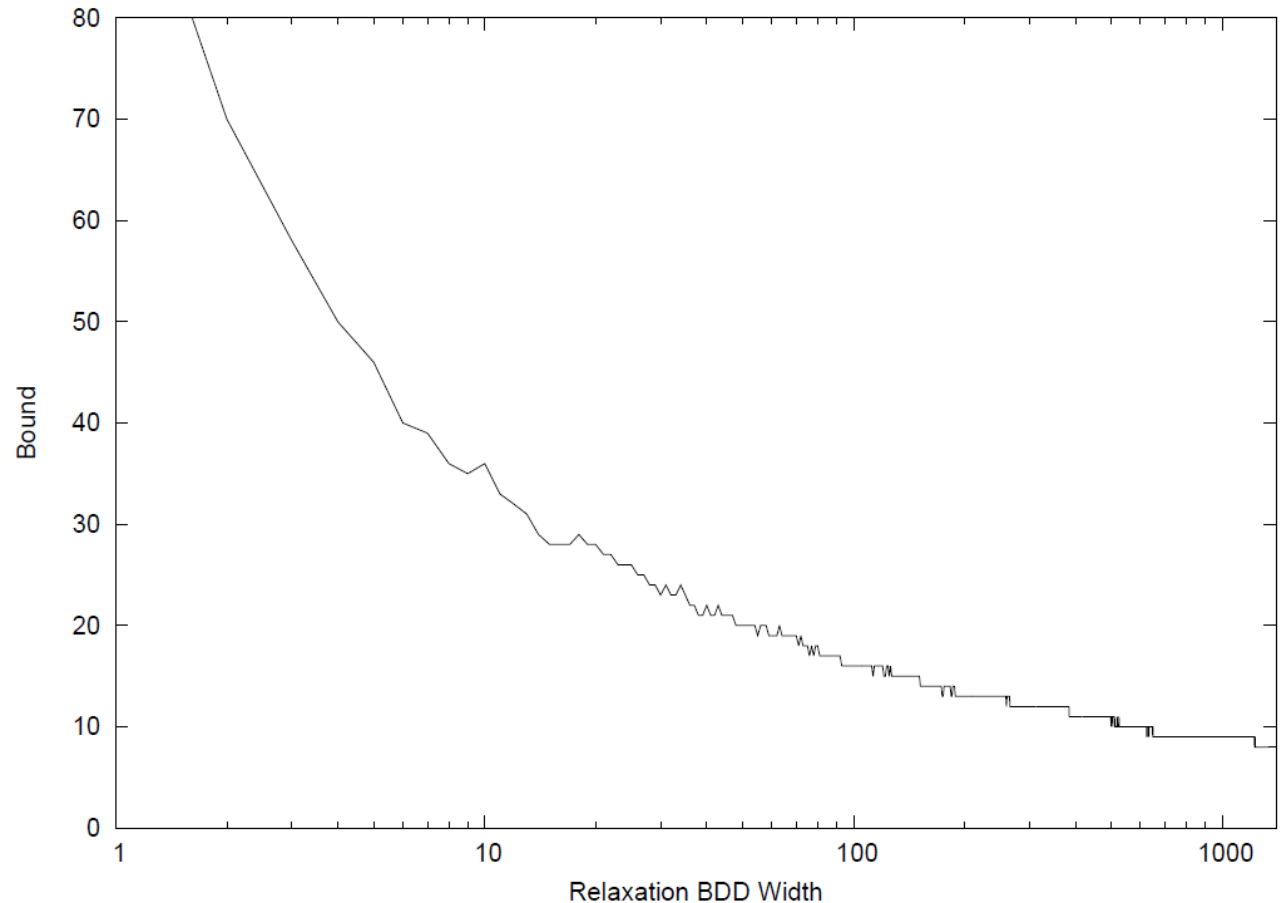
- Variable ordering is key.
 - Just as in branching methods.
 - Some orderings provide much tighter bounds.
- Width of exact BDD is bounded by Fibonacci numbers.
 - For an ordering induced by maximal path decomposition of the graph.
- We used a dynamic ordering heuristic.
 - Next variable is the one appearing in the smallest number of states on the current level.
 - Better than maximal path ordering.

Merging Heuristic

- Which nodes to merge when building relaxation?
- A longest path heuristic seems by far the best.
 - Order nodes on current layer by increasing length of longest path into each node.
 - Merge nodes in this order.
 - We lose information in areas not likely to be part of the solution.
- This is better than merging nodes with more vertices in the corresponding states.

Width of BDD

- Wider BDDs yield tighter bounds.
 - But take longer to build.

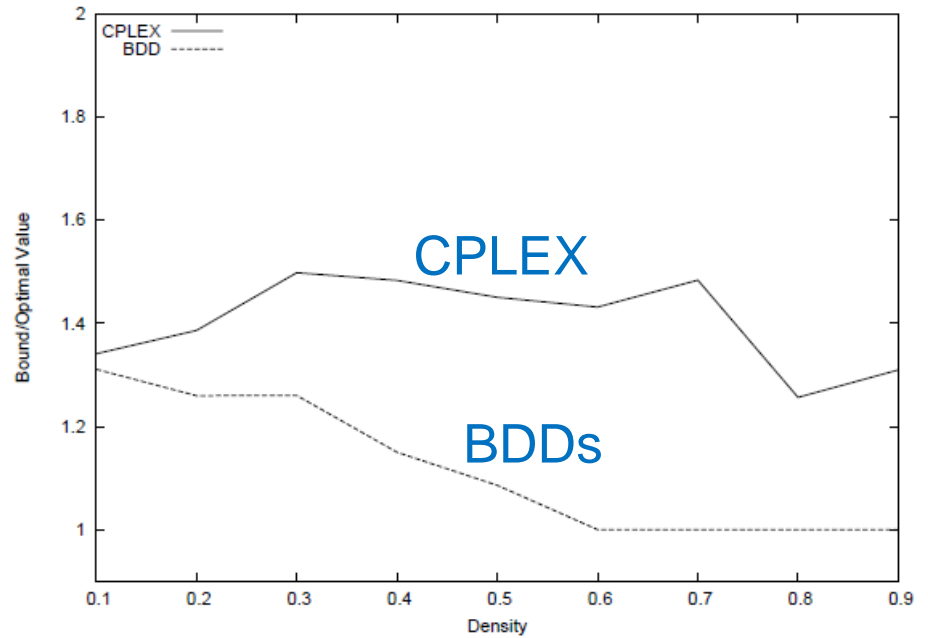


Comparison with LP Bound

- Random and benchmark instances
- Compare with LP bound at root node in CPLEX
 - Turn off presolve
 - We don't use it.
 - It makes CPLEX bound worse or at most 1 better.
 - Use only clique cuts
 - Other cuts improve bound at most 1
 - And require orders of magnitude more time

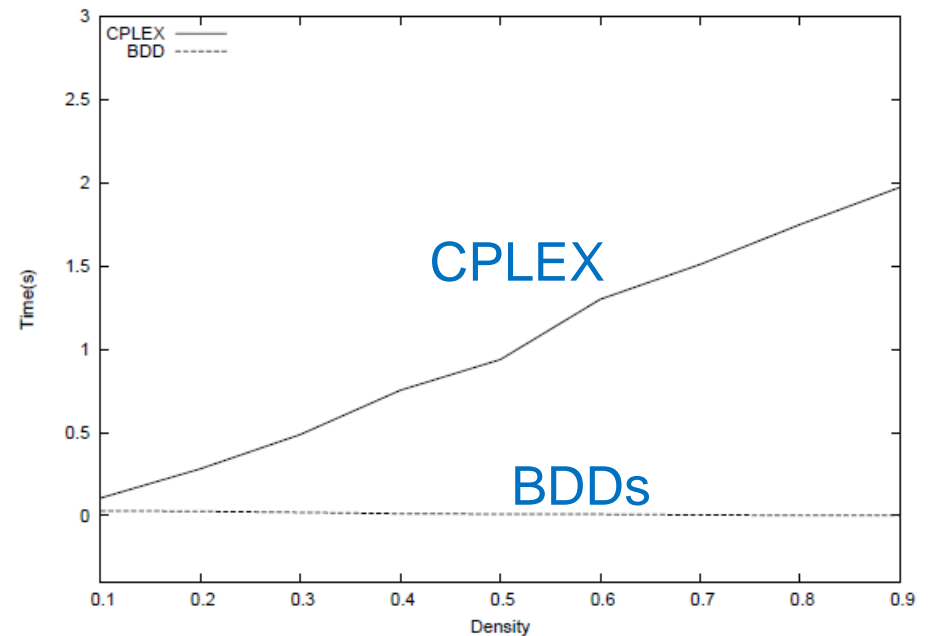
Random instances

Relative bound
vs. density



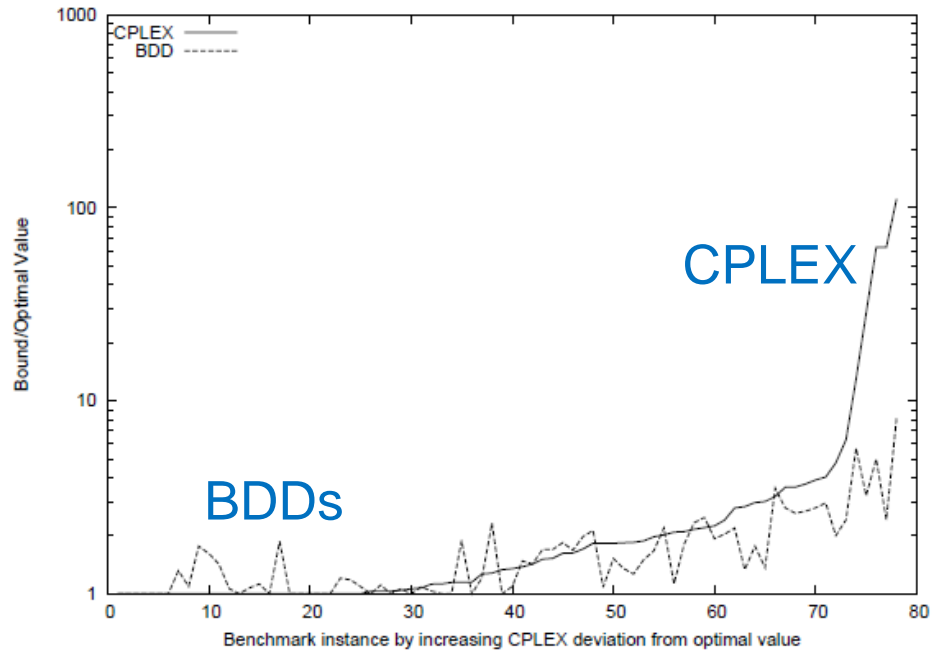
100 vertices
Max BDD width = 100

Time
vs. density



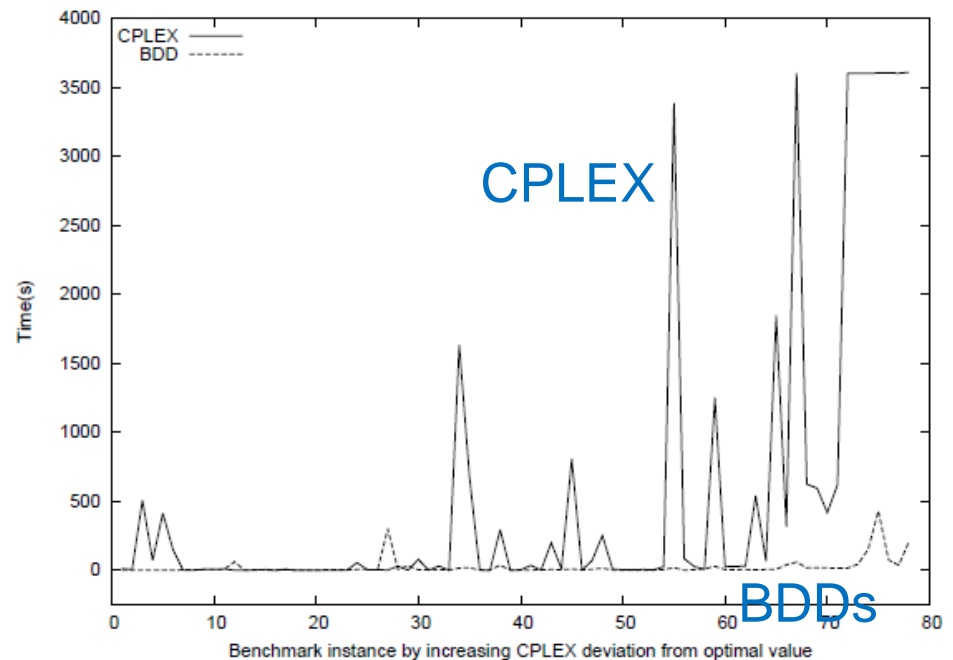
Benchmark instances (DIMACS)

Relative bound



Max BDD width = 100

Time



Alternate LP Formulation

- Try clique cover formulation for LP.
 - Faster than generating clique cuts in CPLEX.
 - Turn off all cuts in CPLEX.
- Requires computing clique cover before LP is formulated.
 - May yield tighter bounds than 100-width BDD for sparse instances.

Alternate LP Formulation

- Try clique cover formulation for LP.
 - Faster than generating clique cuts in CPLEX.
 - Turn off all cuts in CPLEX.
- Requires computing clique cover before LP is formulated.
 - May yield tighter bounds than 100-width BDD for sparse instances.
- Remedy: use wider BDDs to get tighter bounds.
 - We can process a **much** wider BDD in the same time CPLEX spends at the root node.
 - E.g., width 1 million

Incrementality

- Fast incremental calculation of bound.
 - Modify relaxed BDD after variable is fixed in branching tree (fast).
 - Recompute longest path (fast).
- However, may be better to rebuild relaxation from scratch.
 - Research issue.

Future Work

- “Separating cuts”
 - They have an analog for BDDs.
 - Modify BDD to remove solution of current relaxation.
- More problems
 - Assembly line sequencing
 - Vehicle routing with time windows

Future Work

- General BDD-based solver
 - Branch in the BDD
 - Combine with BDD-based propagation
 - BDD-based bounds, primal heuristic
 - No LP relaxation, cutting planes
 - Linearity, convexity irrelevant
 - Possibly nonseparable objective function

Dynamic Programming

- DP state transition graph is a weighted decision diagram.
 - Possibly nonreduced.
 - Objective function is in general nonseparable.
 - Conversion to canonical costs may allow reduction.

Dynamic Programming

- DP state transition graph is a weighted decision diagram.
 - Possibly nonreduced.
 - Objective function is in general nonseparable.
 - Conversion to canonical costs may allow reduction.
- DP state + BDD state
 - DP state at each node provides information in addition to BDD.
 - Can strengthen a relaxed BDD by associating a DP state with each node.
 - BDD relaxation procedure leads to dynamic state space relaxation for DP.