The Separation Problem for Binary Decision Diagrams

J. N. Hooker

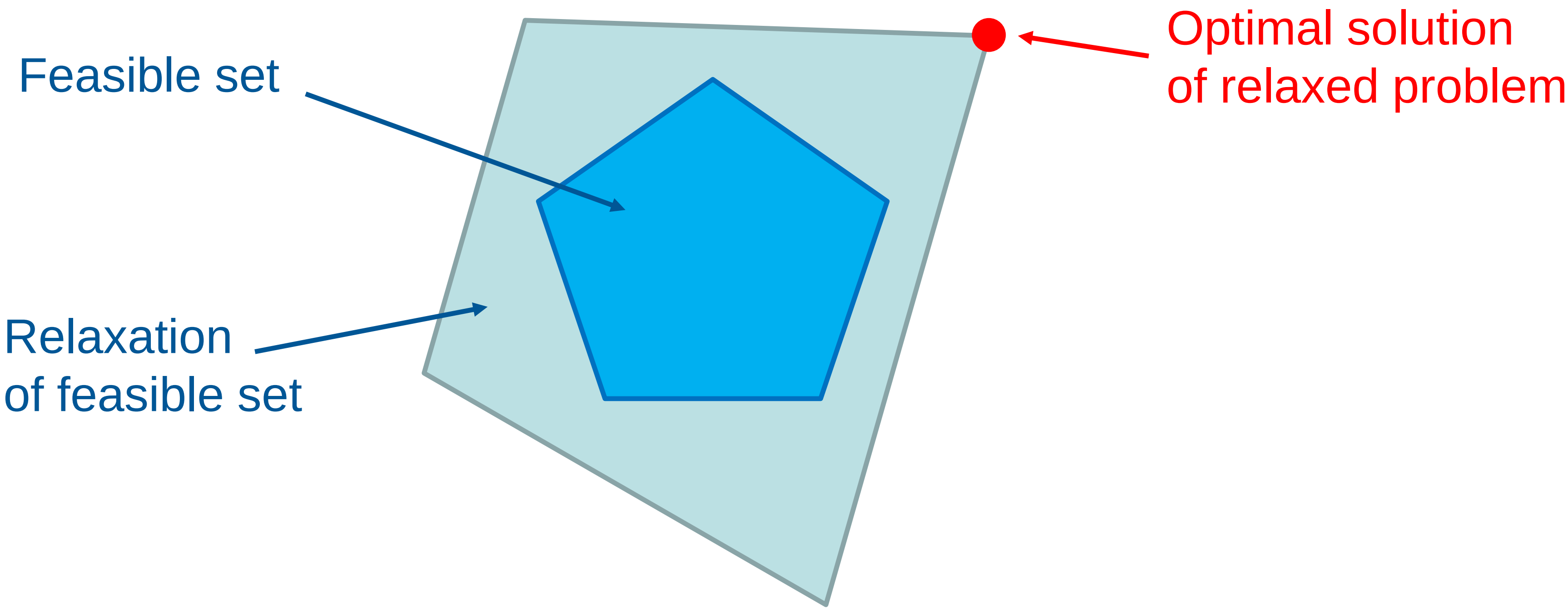
Joint work with André Ciré

Carnegie Mellon University

ISAIM 2014

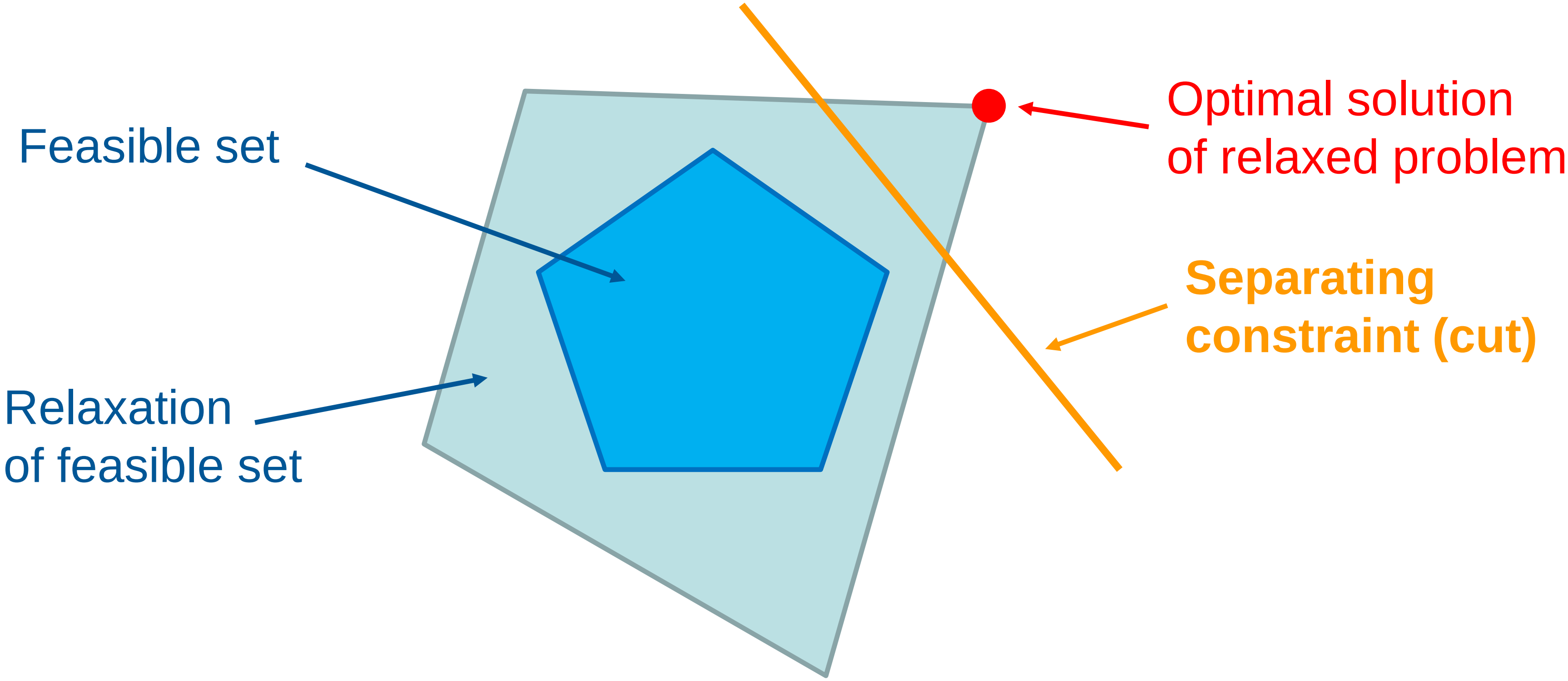
Separation Problem in Optimization

- Given a relaxation of an optimization problem…
- Find a constraint that **separates** solution of the relaxation from the feasible set



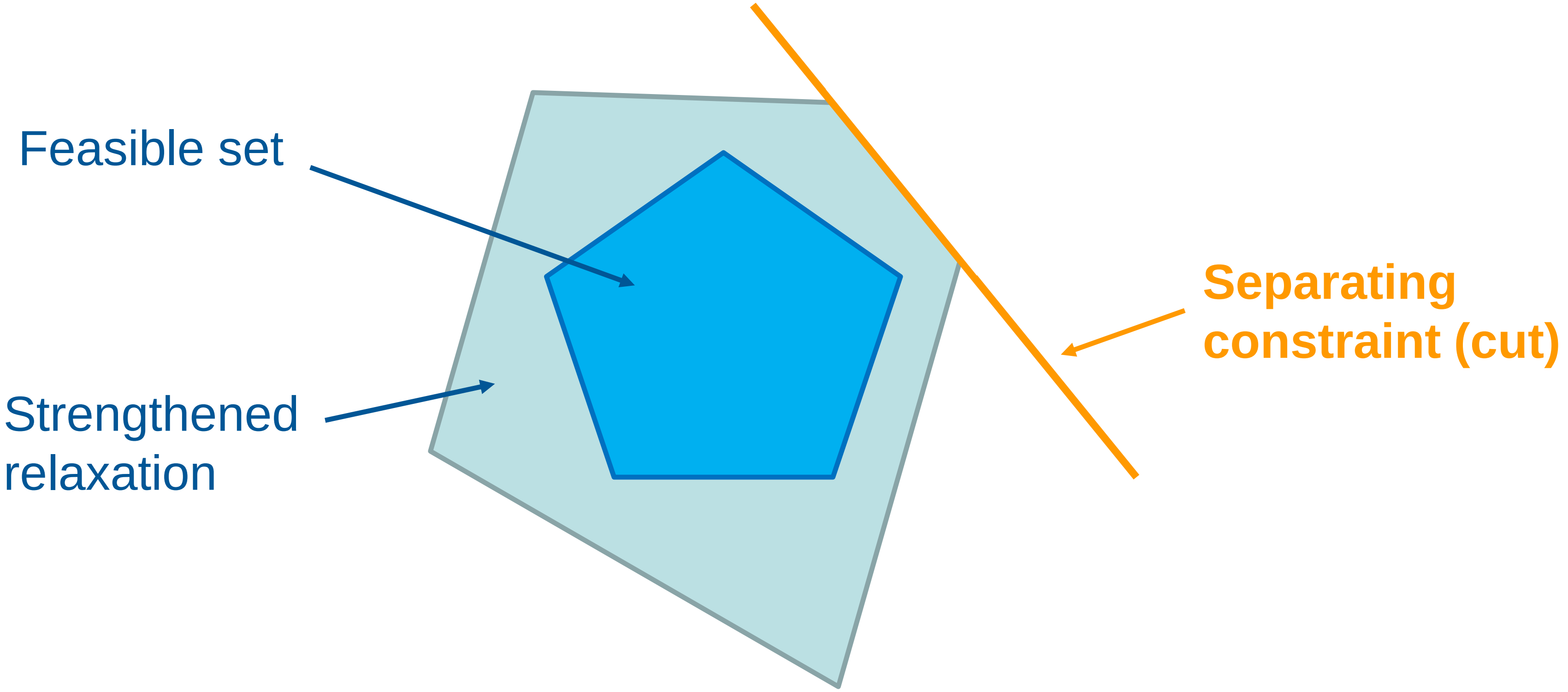
Separation Problem in Optimization

- Given a relaxation of an optimization problem…
- Find a constraint that **separates** solution of the relaxation from the feasible set



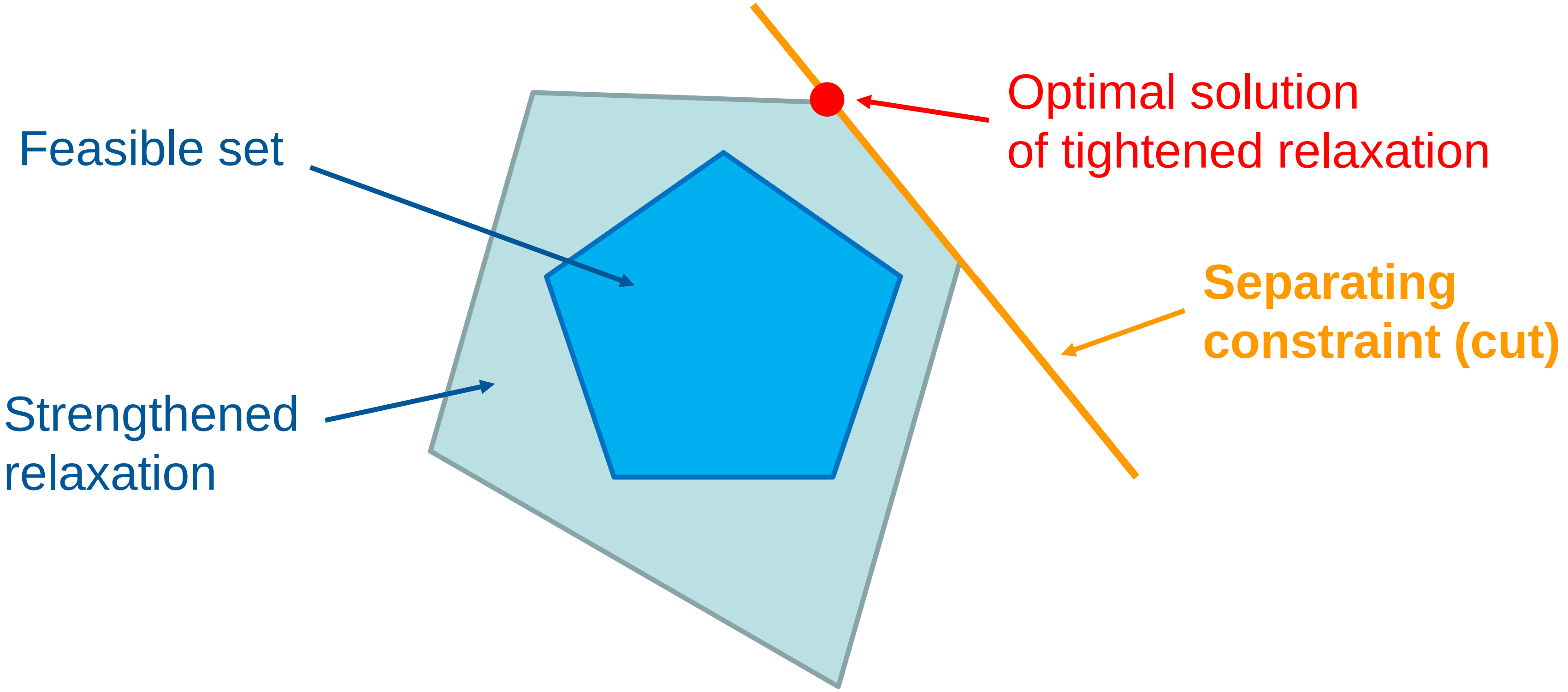
Separation Problem in Optimization

- Now **strengthen** the relaxation with the separating cut.
  - Cuts are usually linear inequalities.



Separation Problem in Optimization

- Now **strengthen** the relaxation with the separating cut.
  - Cuts are usually linear inequalities.
  - Re-solve relaxation and repeat.



Separation Problem in Optimization

- Separation is a **workhorse** in integer and nonlinear programming.
  - Relaxations are usually polyhedral.
  - *Typical problem:* in a **large** known family of cuts, find one that separates.
  - Cannot publish a cutting plane paper without a separation algorithm.

Separation Problem in Optimization

- Example: Integer programming
  - **Gomory** cuts
  - **Mixed integer rounding** cuts
  - Separating **knapsack** cuts
  - Separating **cover** inequalities
  - Separating cuts in special families
    - Subtour elimination, combs for TSP
    - Separating flow cuts for fixed-charge network flow
    - etc. (**huge** literature)

Separation Problem for BDDs

- Is there a role for separation in **discrete** relaxations?
- We will look at separation for **binary decision diagrams** (**BDD**s).
  - Recently used for **discrete optimization** and **constraint programming**.

Separation Problem for BDDs

- Is there a role for separation in **discrete relaxations**?
- We will look at separation for **binary decision diagrams** (**BDD**s).
  - Recently used for **discrete optimization** and **constraint programming**.
- Key idea: A **relaxed BDD** provides A discrete relaxation of the problem as a.
  - How to separate a solution (or family of solutions) from the relaxed BDD?
  - We will focus on separating **Benders cuts**.

Decision Diagrams

- **Binary decision diagrams (BDD**s**)** historically used for circuit design and verification.
  - Lee 1959, Akers 1978, Bryant 1986.

Decision Diagrams

- **Binary decision diagrams (BDD**s**)** 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.
  - Easy generalization to **multivalued** decision diagrams (MDDs) for finite domain variables.

Decision Diagrams

- **Binary decision diagrams (BDD**s**)** 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.
  - Easy generalization to **multivalued** decision diagrams (MDDs) for finite domain variables.
- BDD is result of superimposing isomorphic subtrees in a search tree.
  - Unique **reduced** BDD for given variable ordering.
  - A type of “caching.”

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 Hoeve, O'Sullivan.

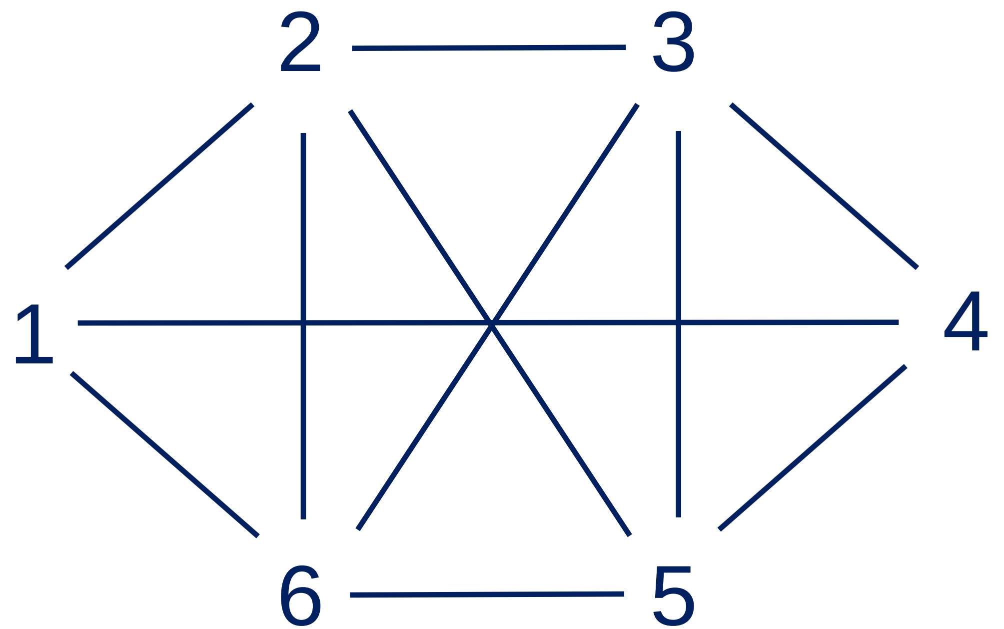
- Example: **independent set problem** on a graph…

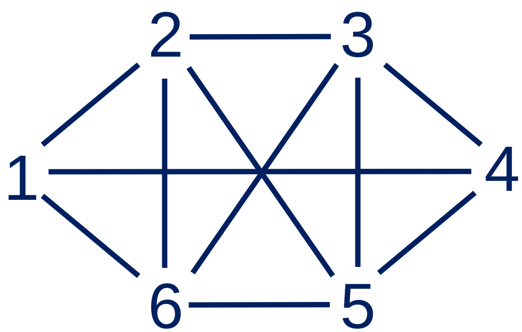
Independent Set Problem

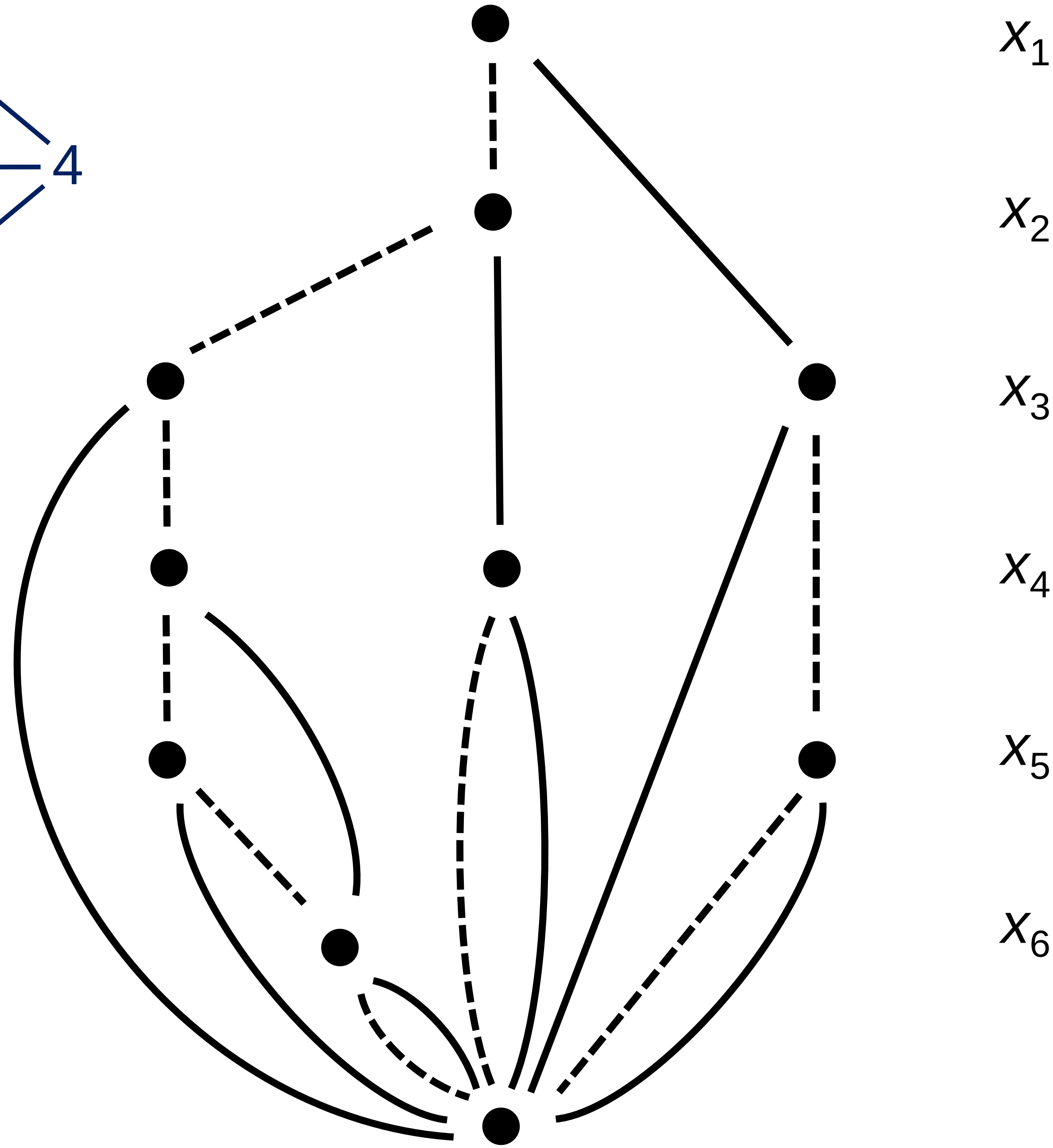
Let each vertex have weight w_i

Select nonadjacent vertices to maximize $\sum_i w_i x_i$

x_i =1 if vertex i selected

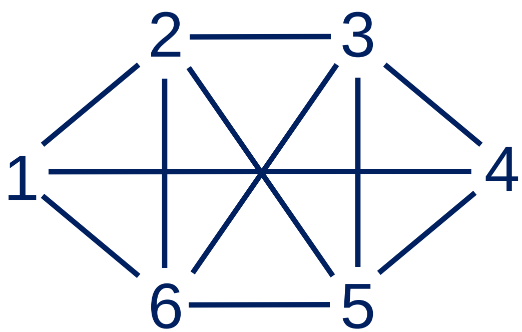






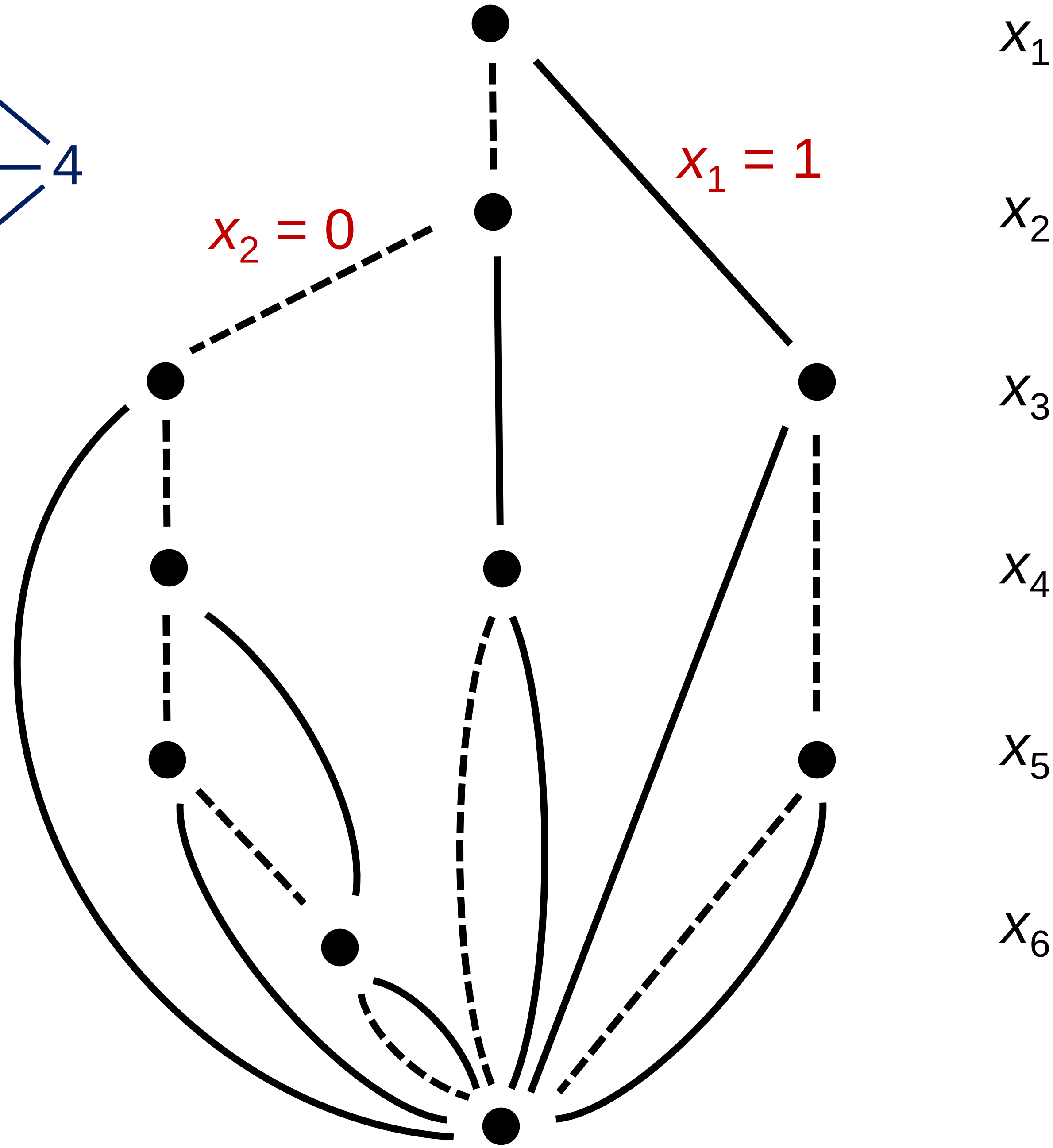
Exact BDD for independent set problem

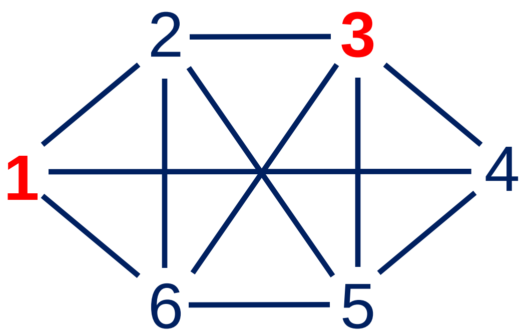
“zero-suppressed” BDD



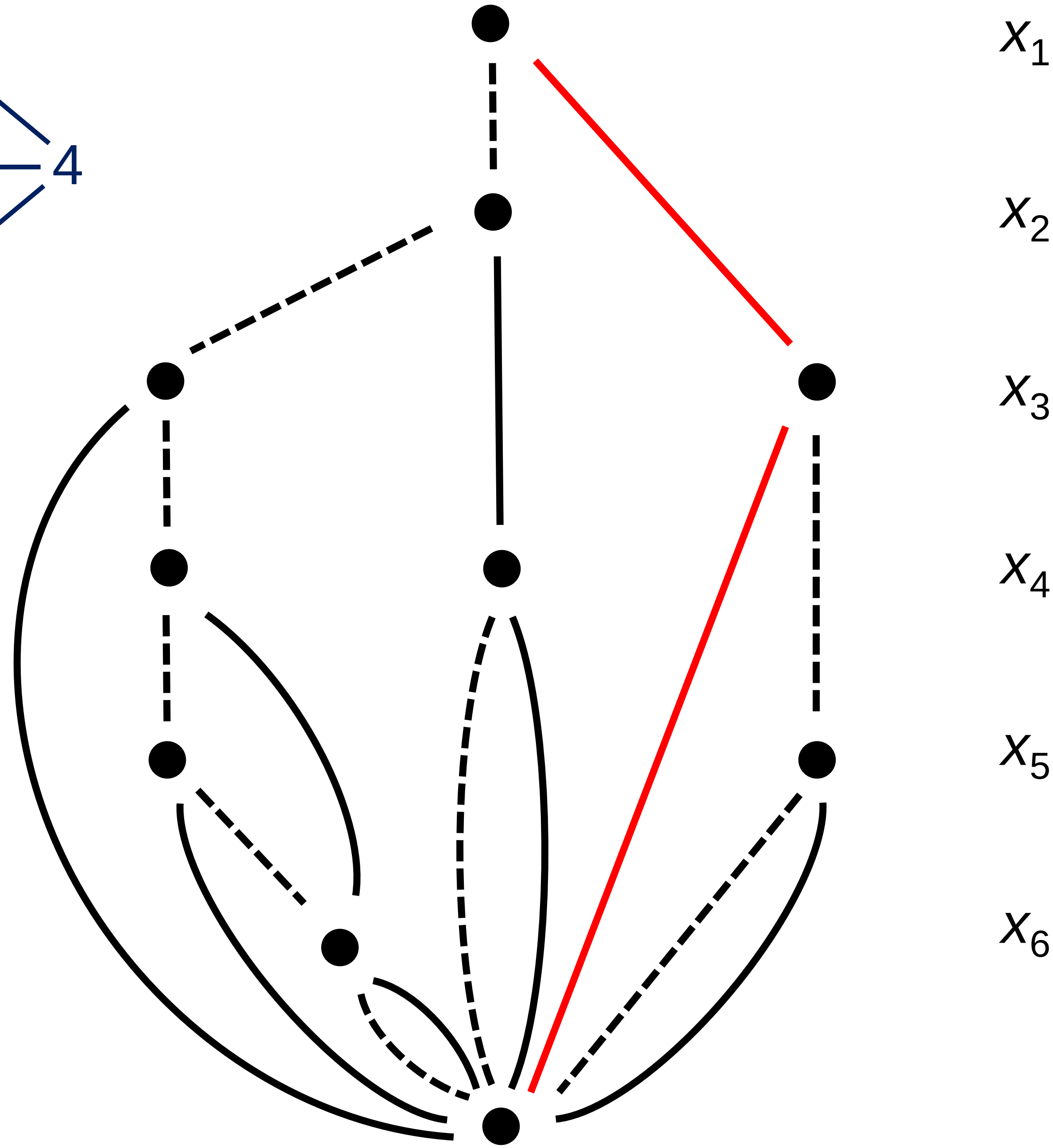
Exact BDD for independent set problem

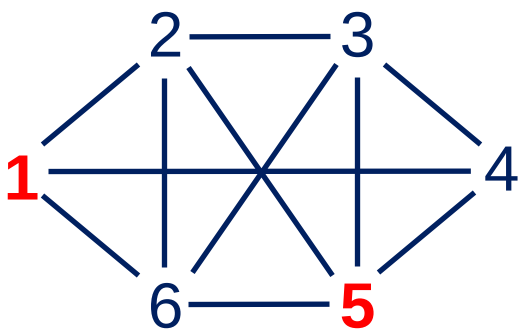
“zero-suppressed” BDD



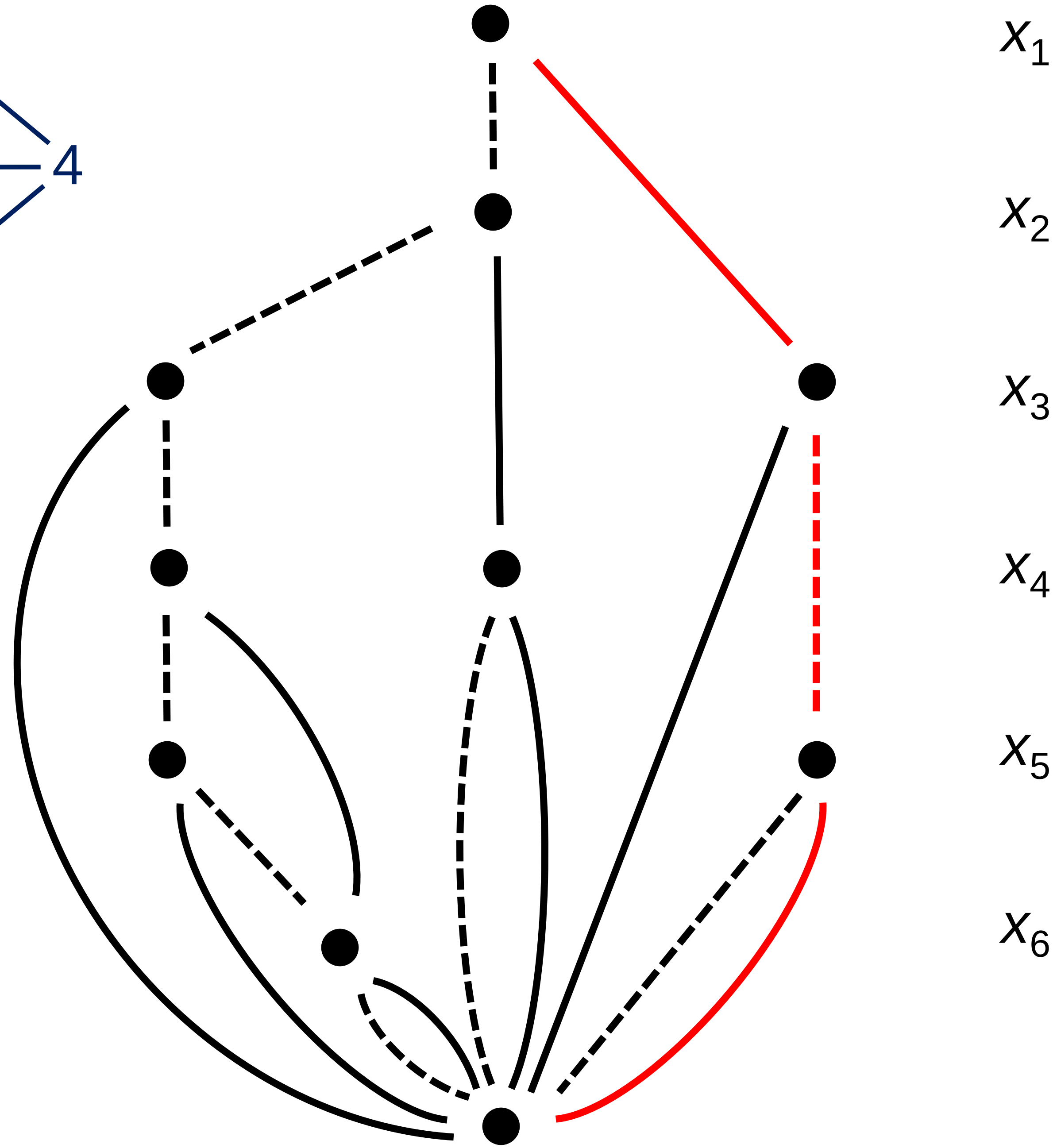


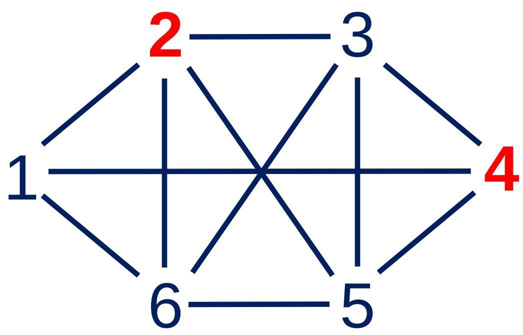
Paths from top
to bottom
correspond to
the 11 feasible
solutions



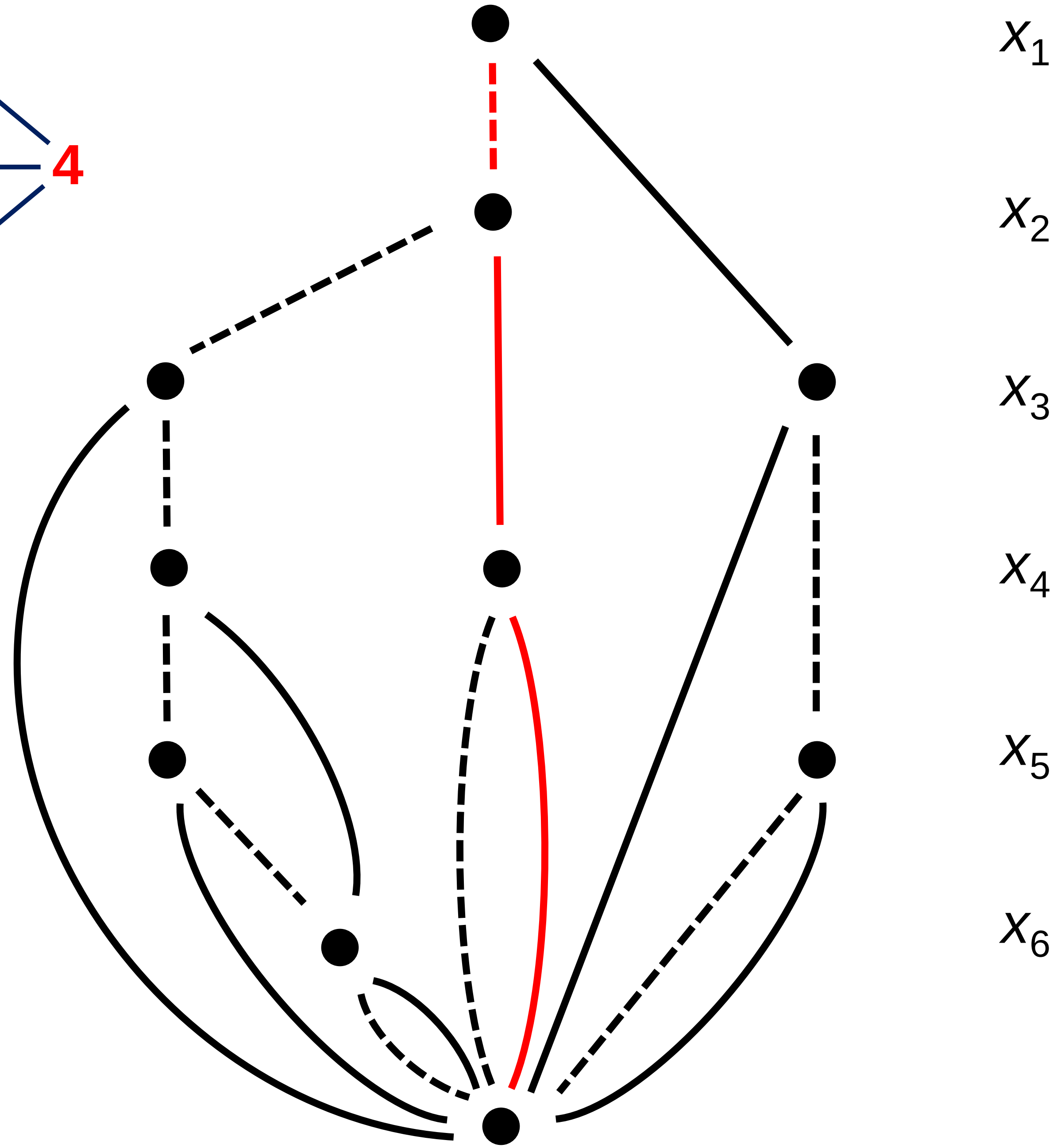


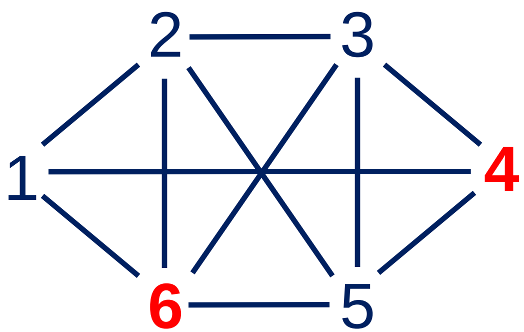
Paths from top to bottom correspond to the 11 feasible solutions



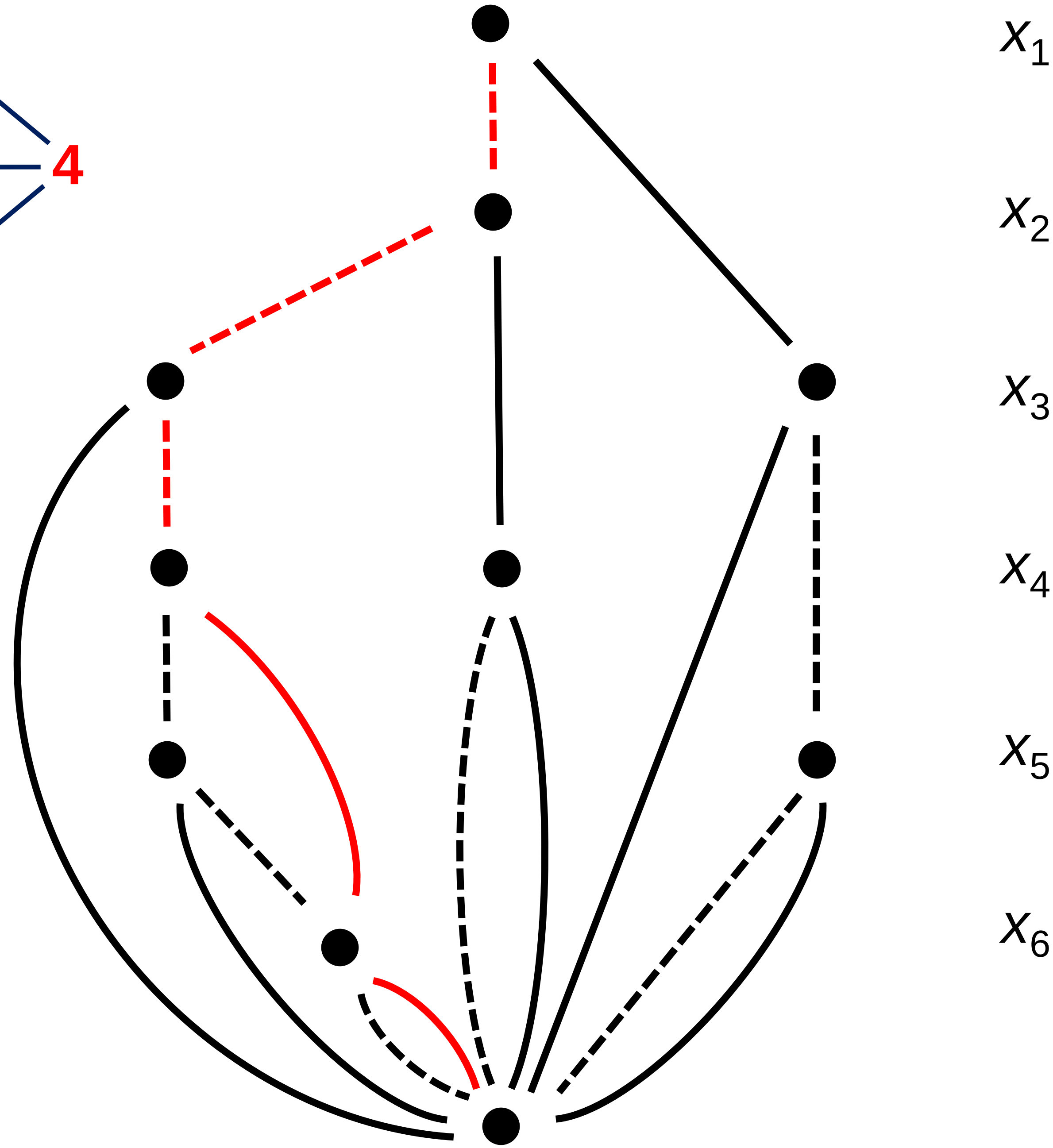


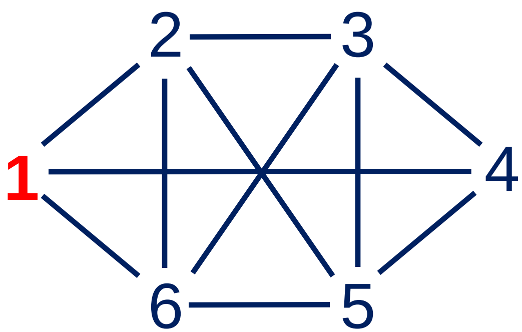
Paths from top to bottom correspond to the 11 feasible solutions





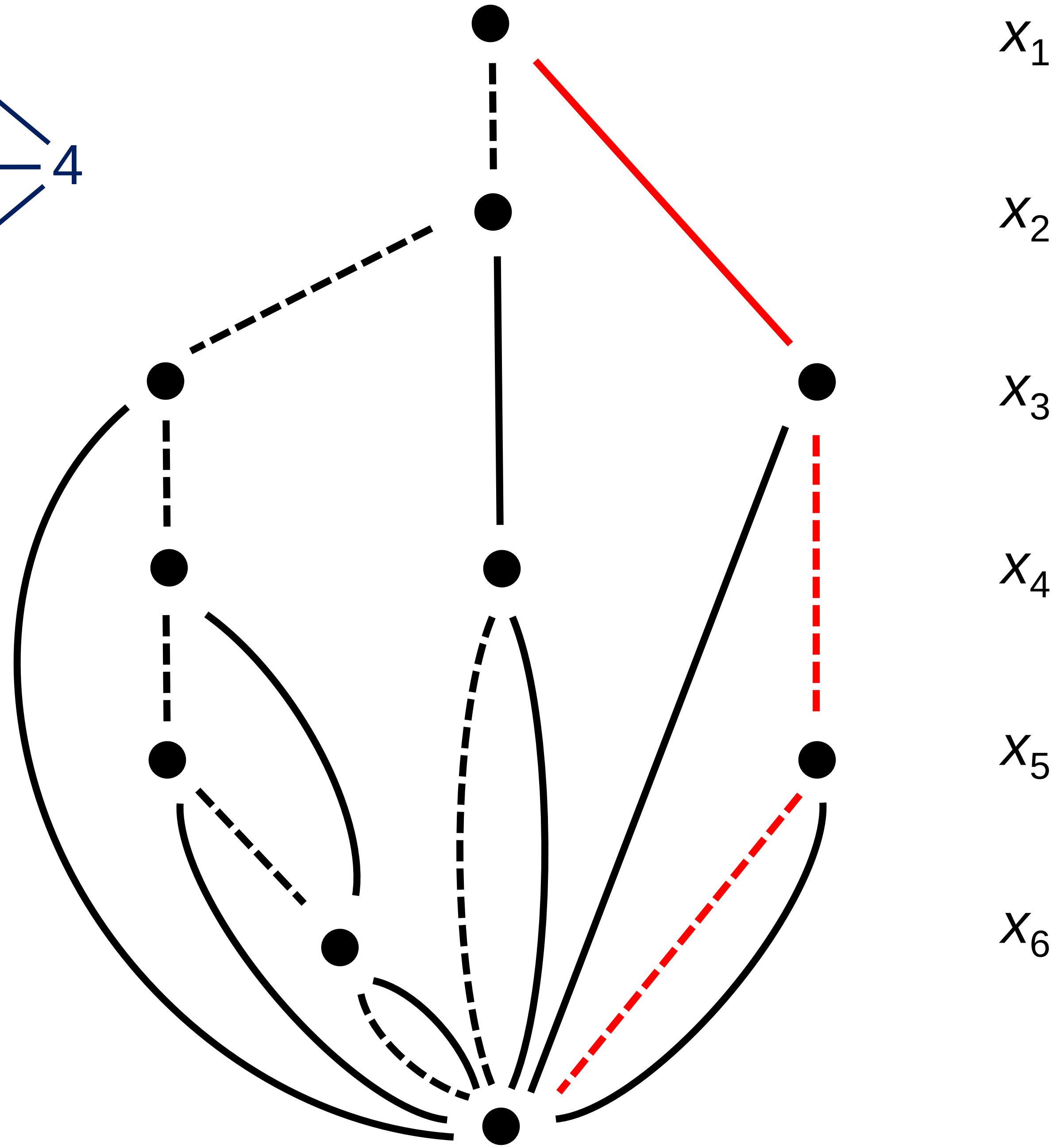
Paths from top to bottom correspond to the 11 feasible solutions

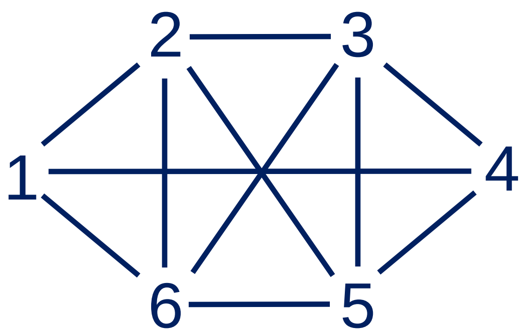




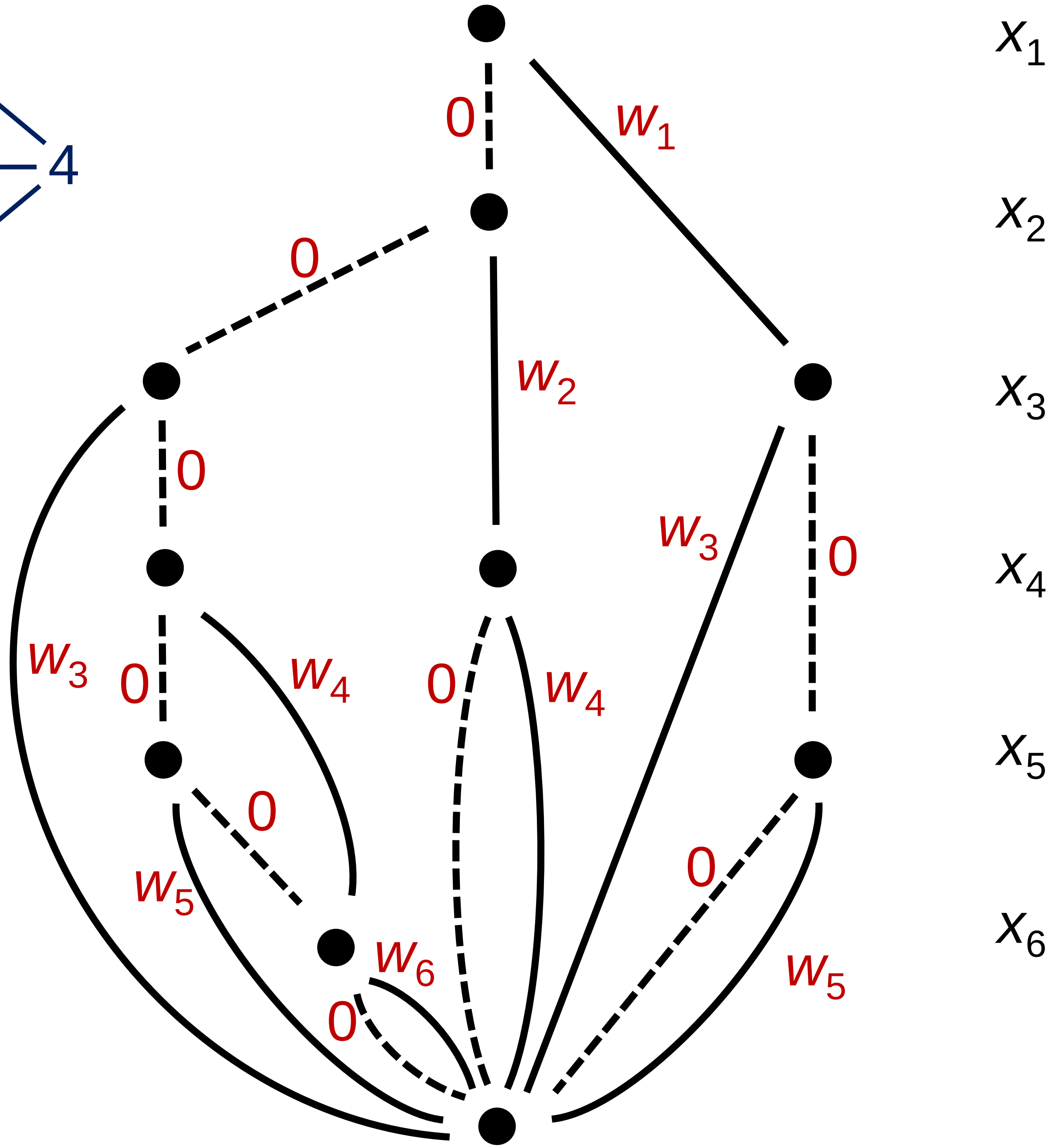
Paths from top to bottom correspond to the 11 feasible solutions

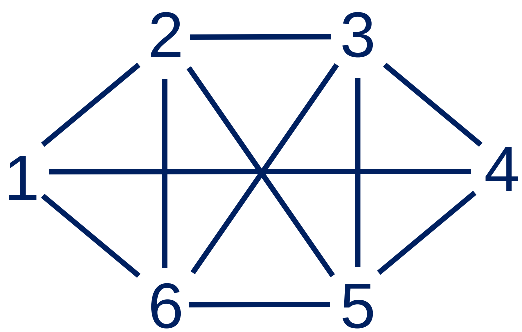
…and so forth

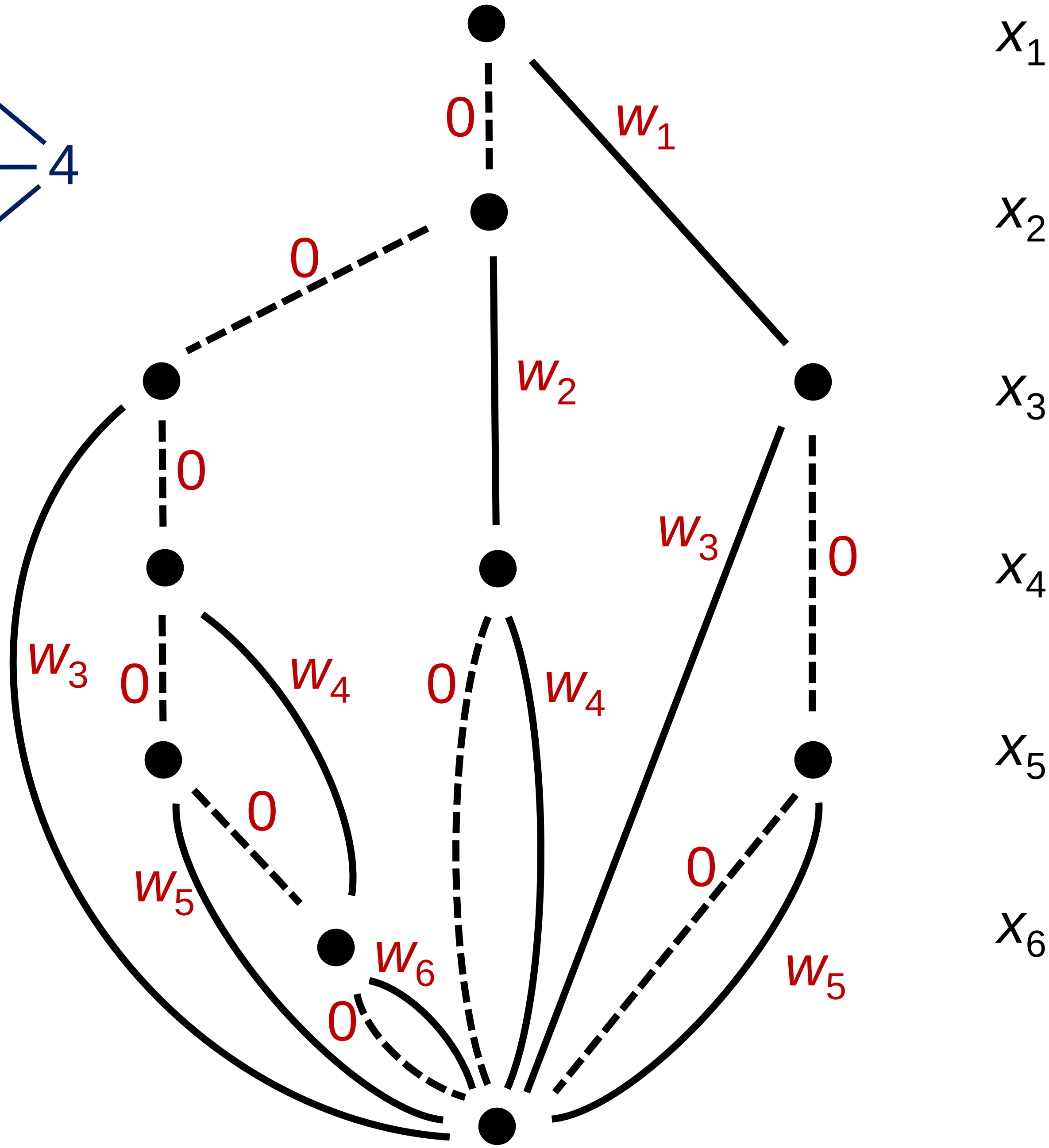




For objective function, associate weights with arcs





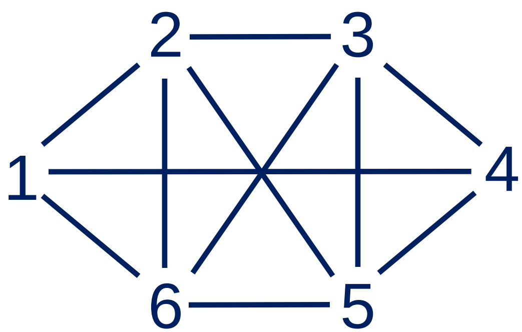


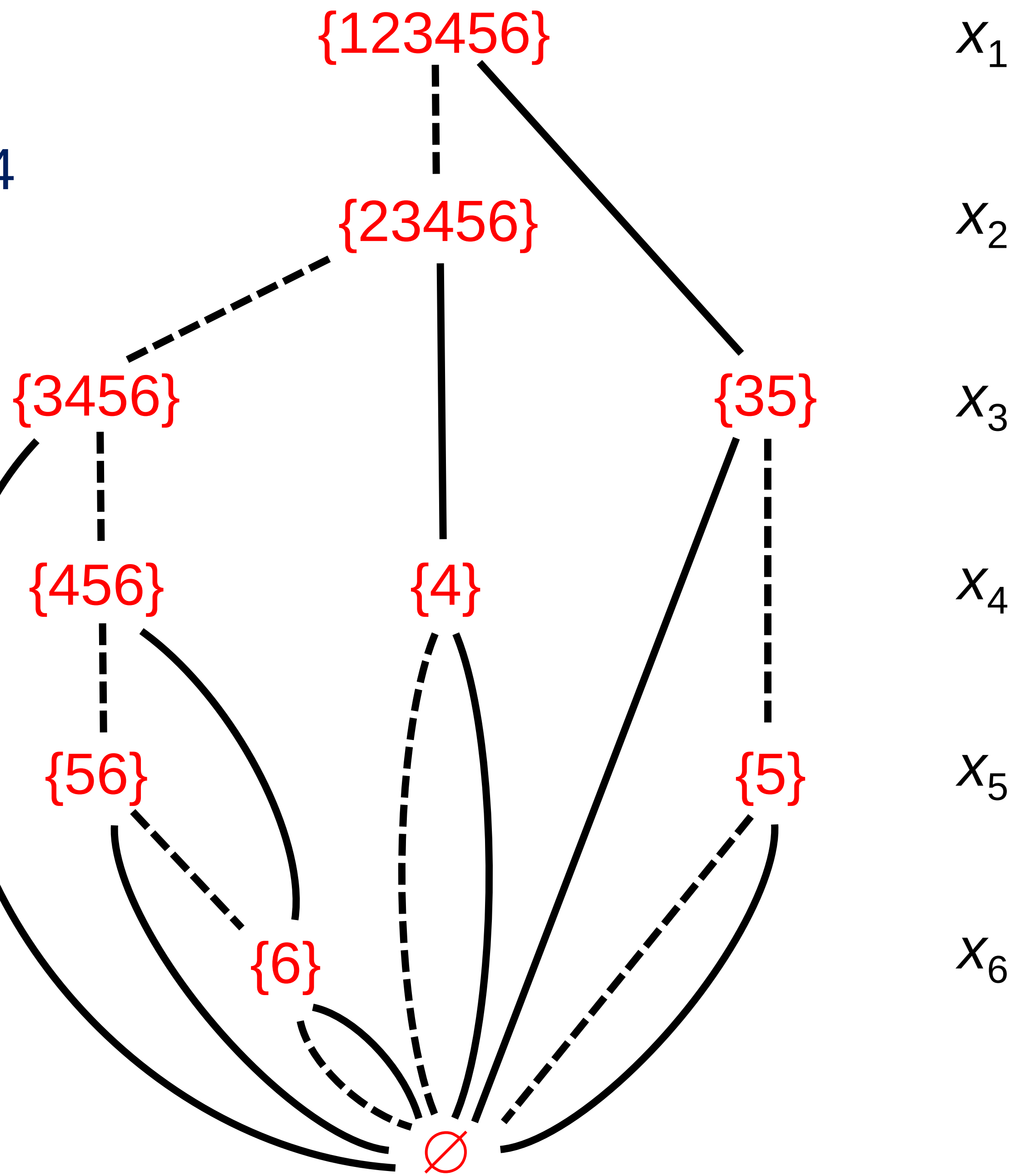
For objective function, associate weights with arcs

Optimal solution is **longest path**

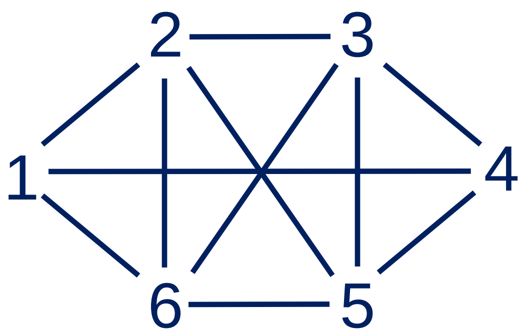
Objective Function

- 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

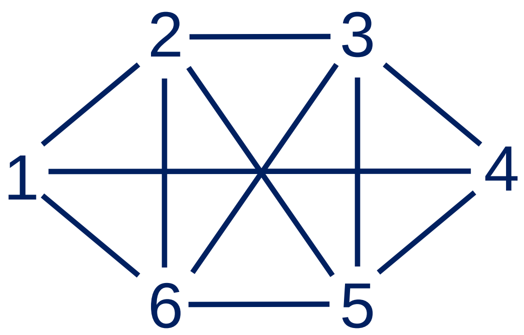
x_3

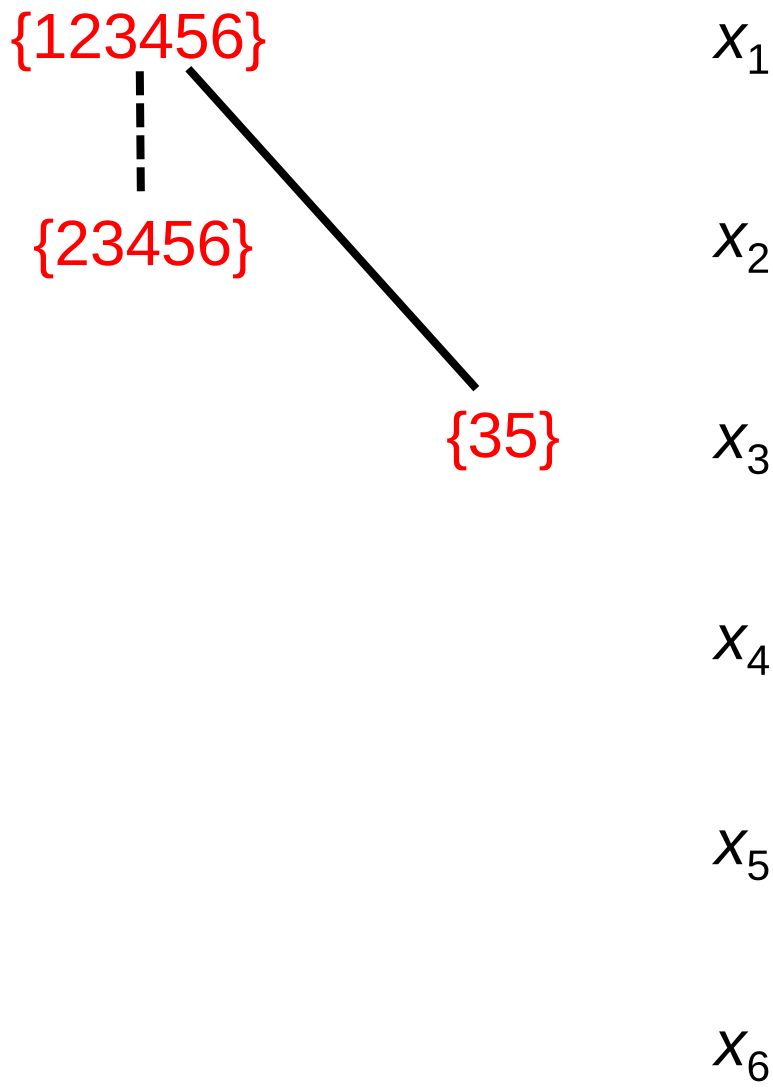
x_4

x_5

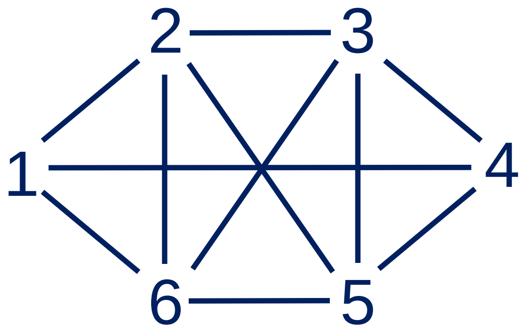
To build BDD,
associate **state**
with each node

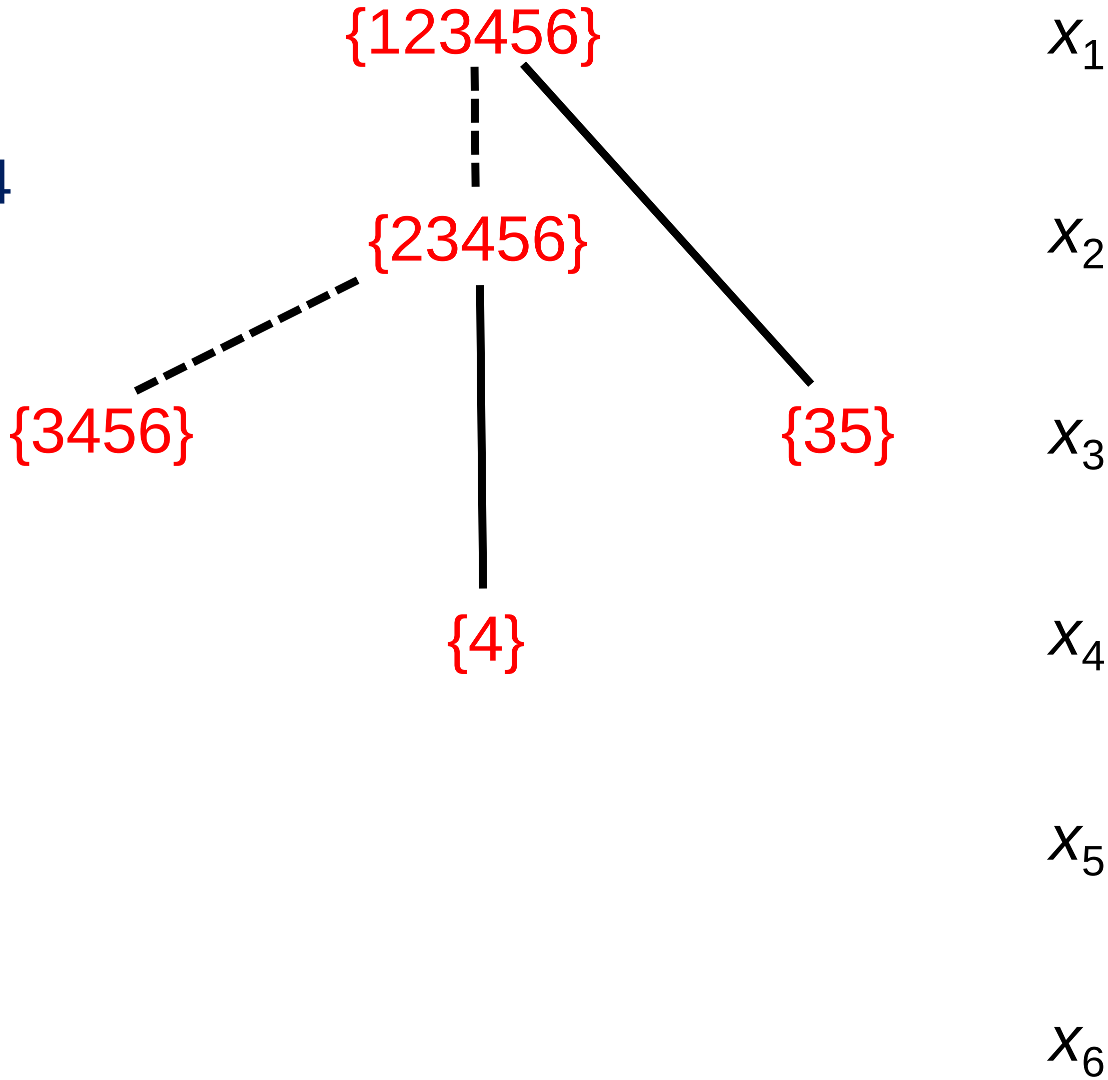
x_6



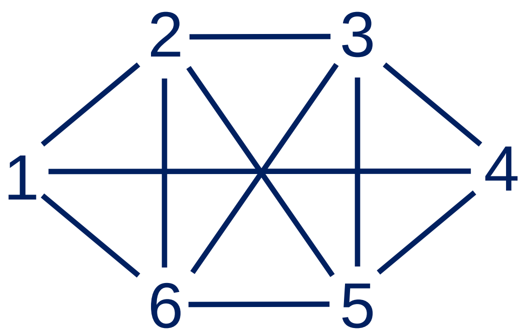


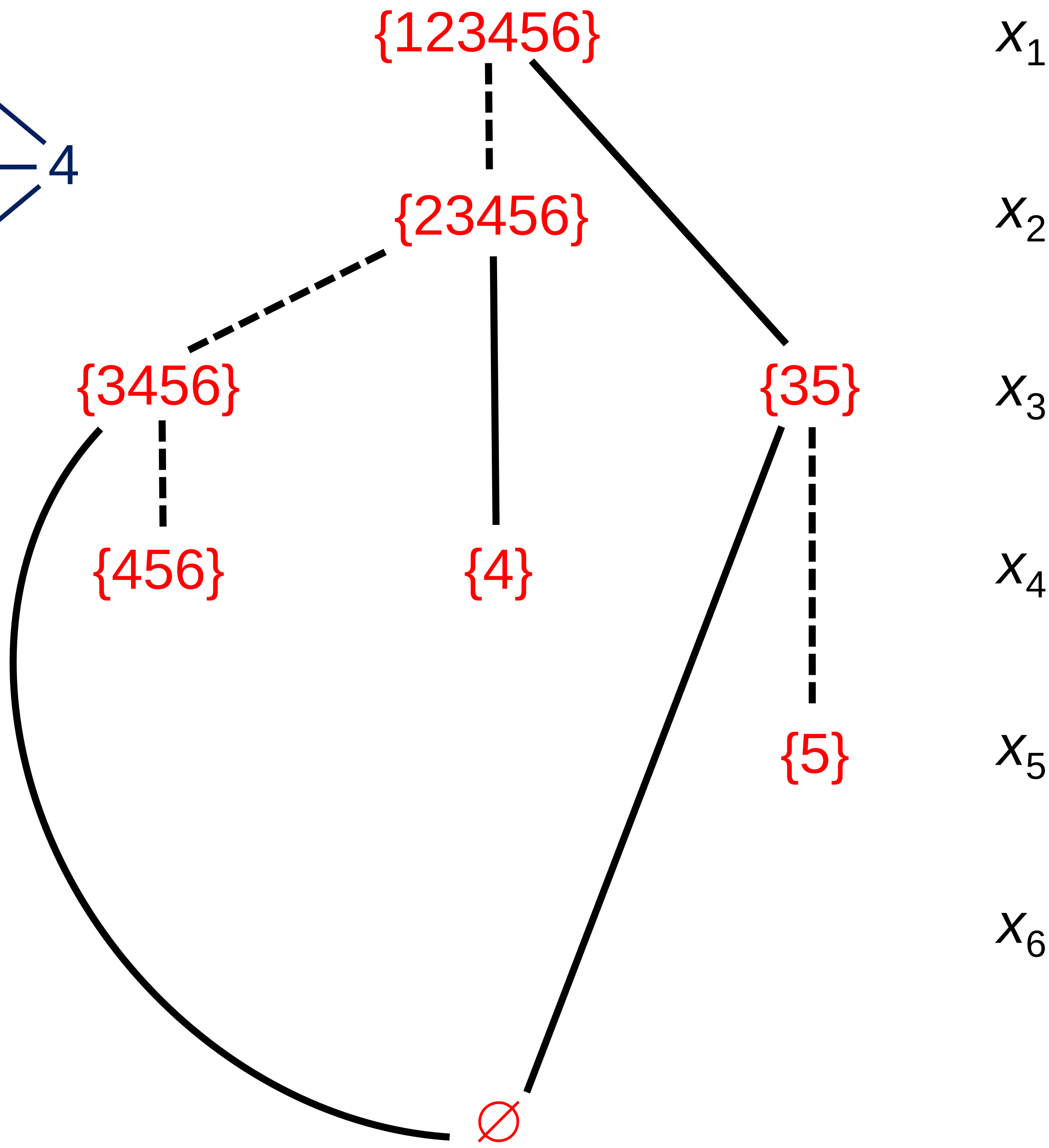
To build BDD, associate **state** with each node



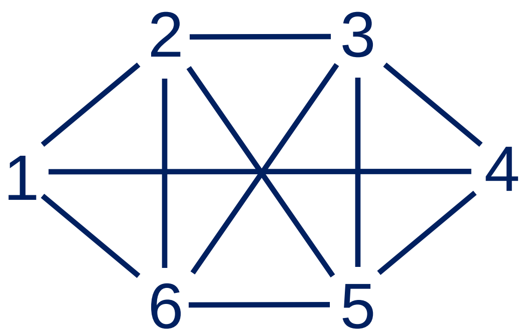


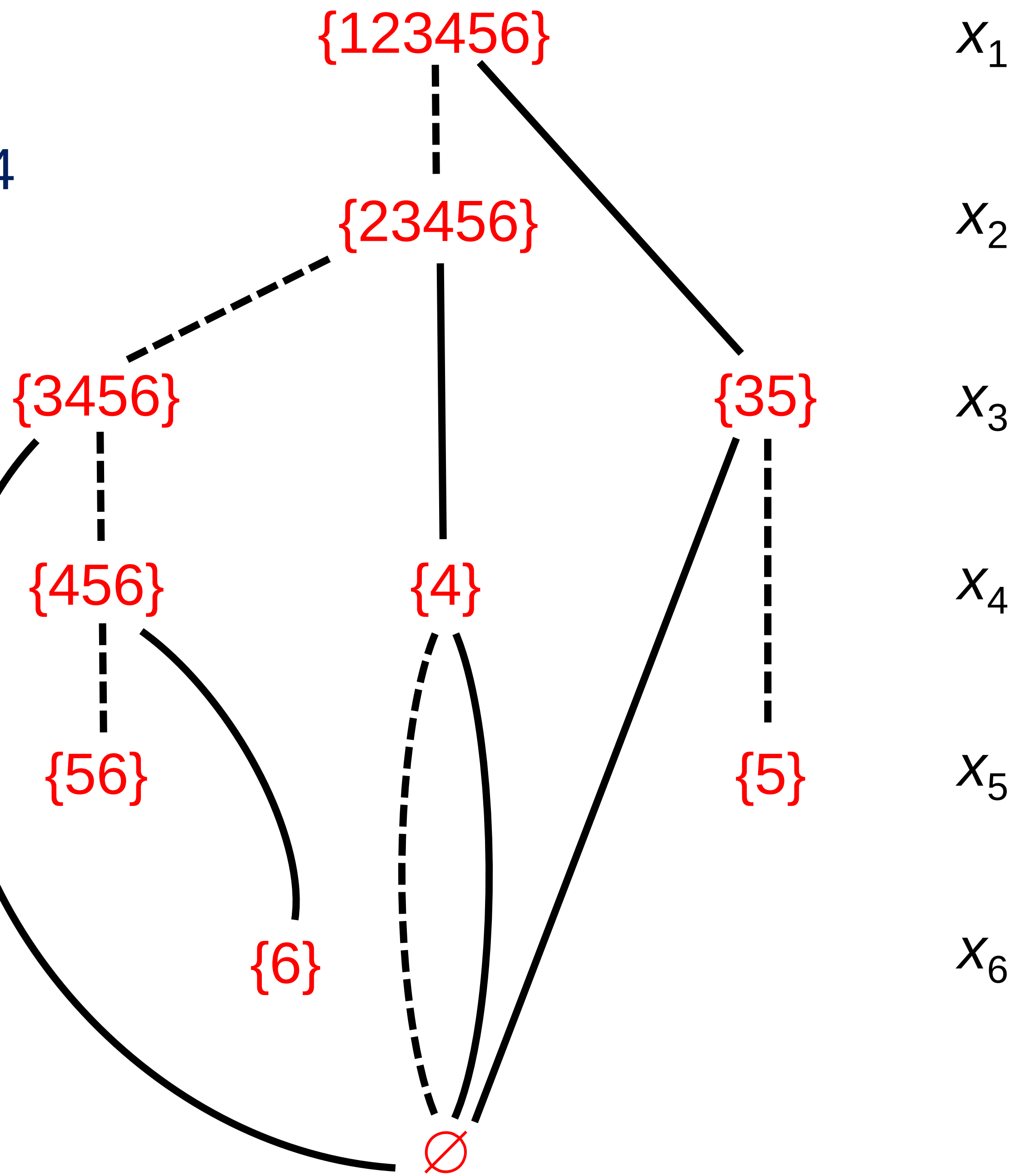
To build BDD,
associate **state**
with each node



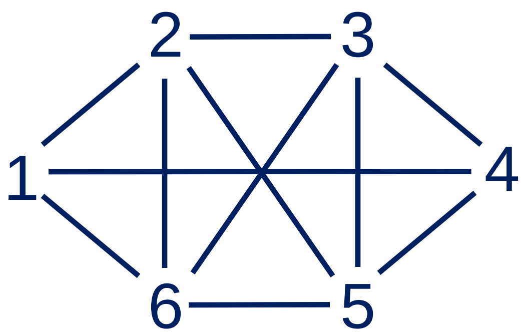


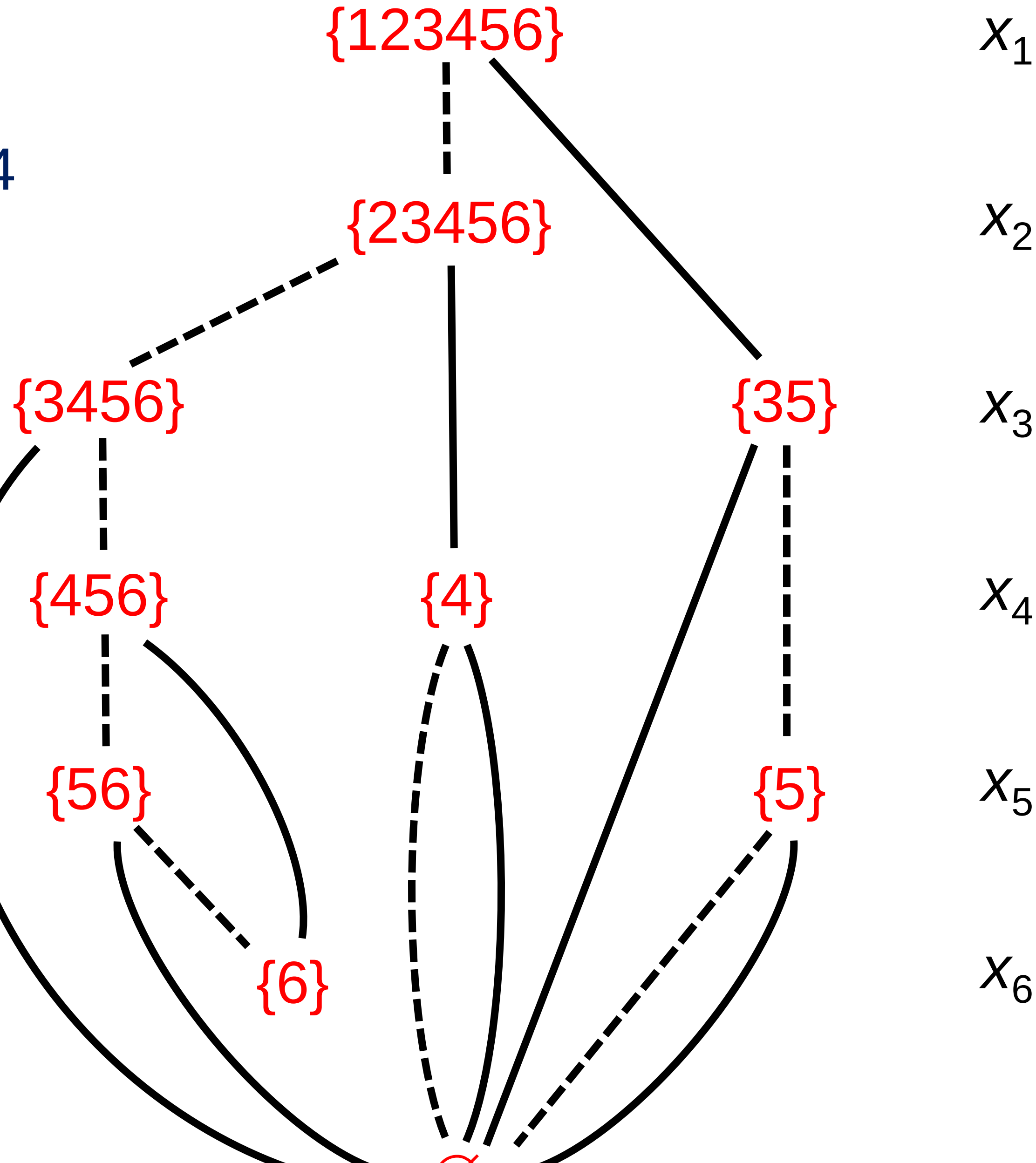
Merge nodes that correspond to the same state



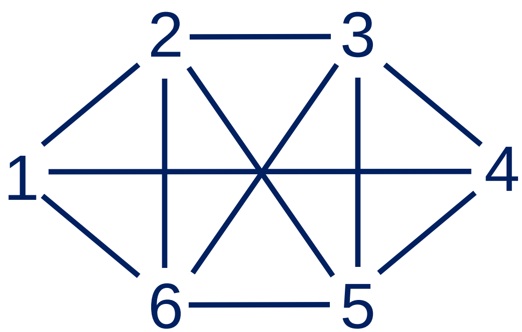


Merge nodes
that correspond
to the same
state

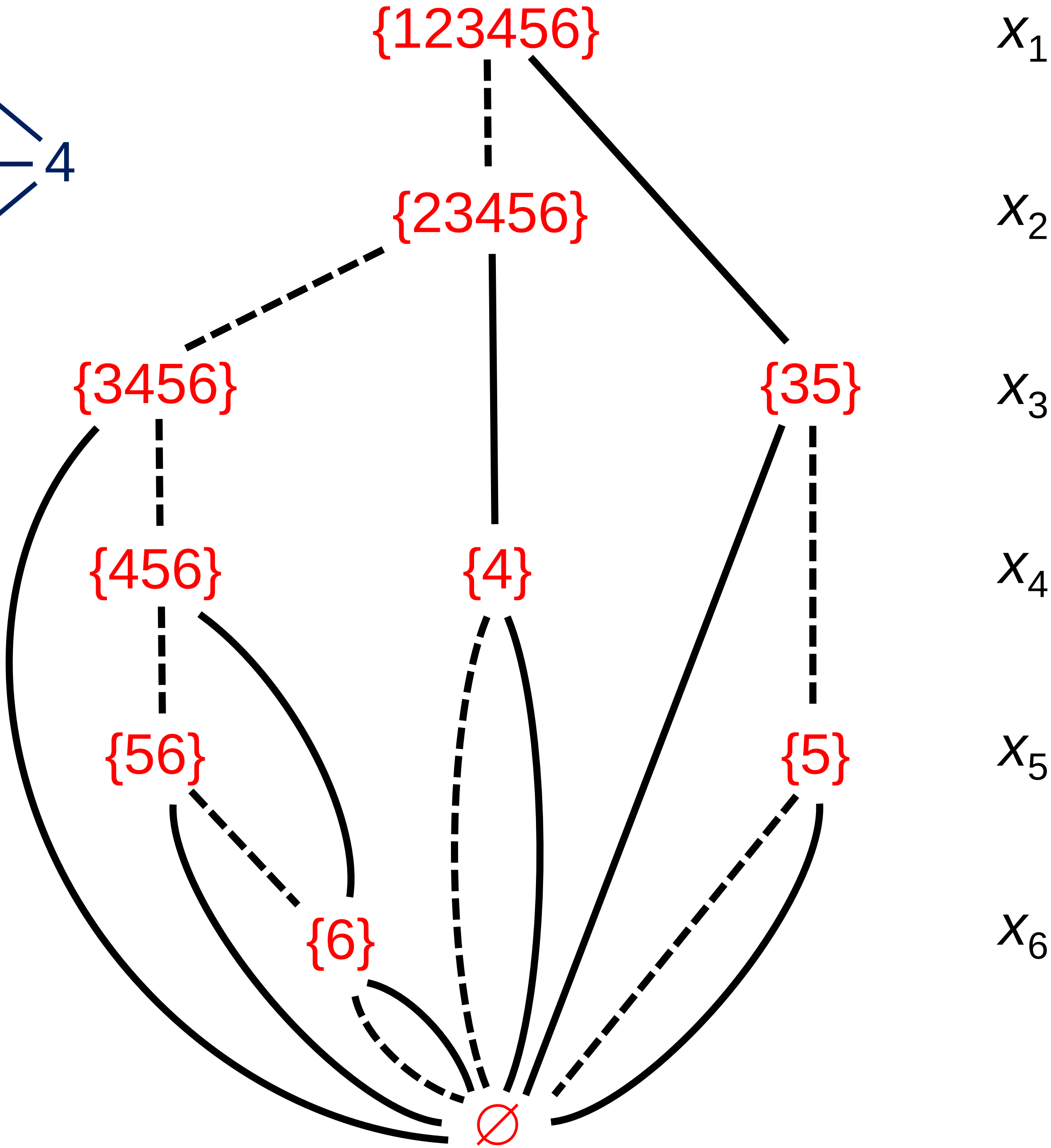




Merge nodes that correspond to the same state



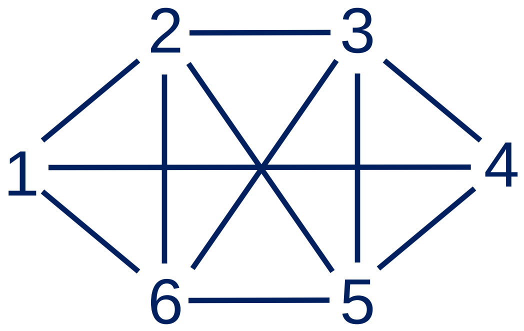
Width = 2



Merge nodes
that correspond
to the same
state

Relaxation Bounding

- To obtain a bound on the objective function:
  - Use a **relaxed** decision diagram
  - Analogous to linear programming relaxation in MIP
  - This relaxation is **discrete**.
  - Doesn't require the linear inequality formulation of MIP.



{123456}

x_1

x_2

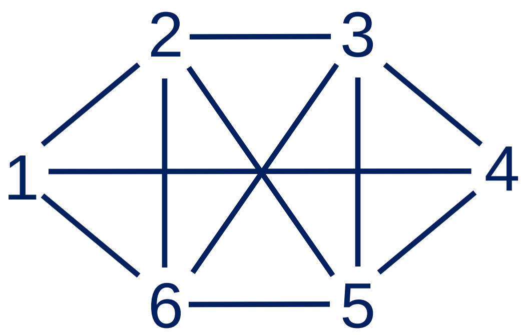
x_3

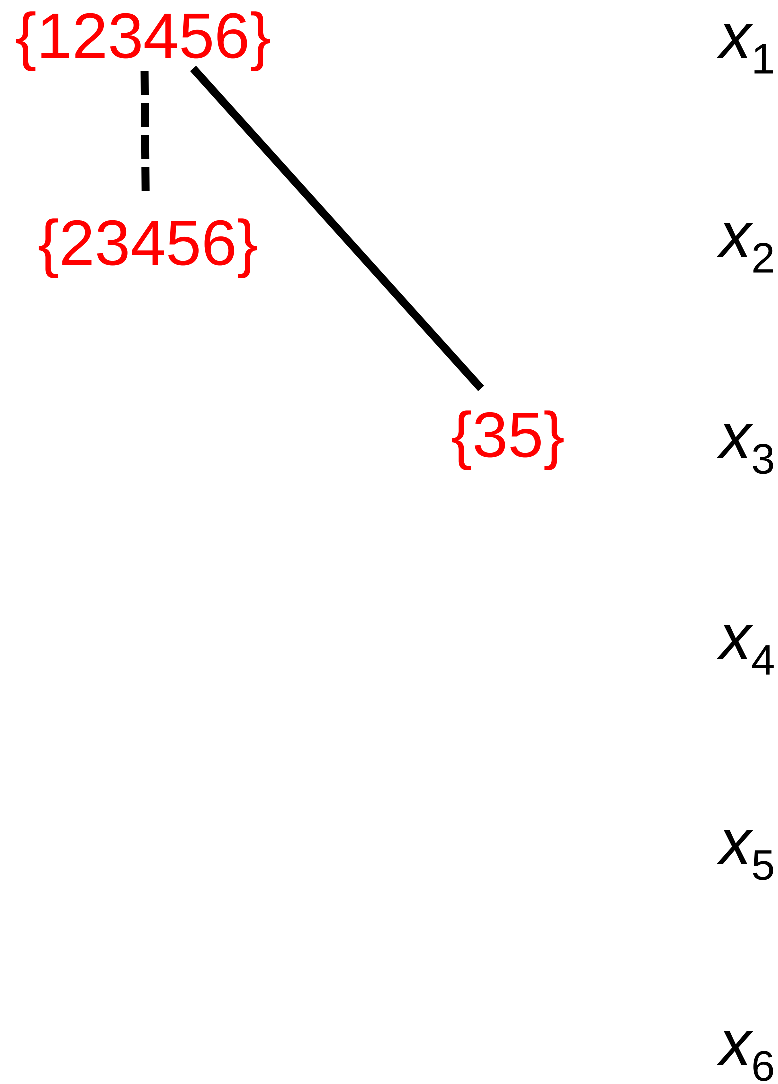
x_4

x_5

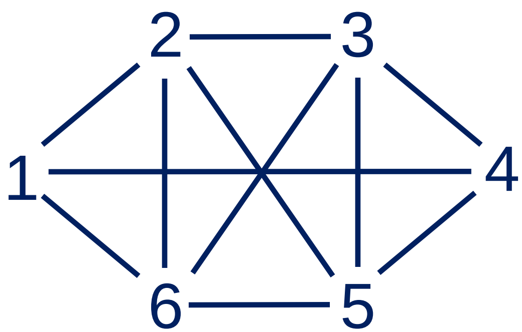
x_6

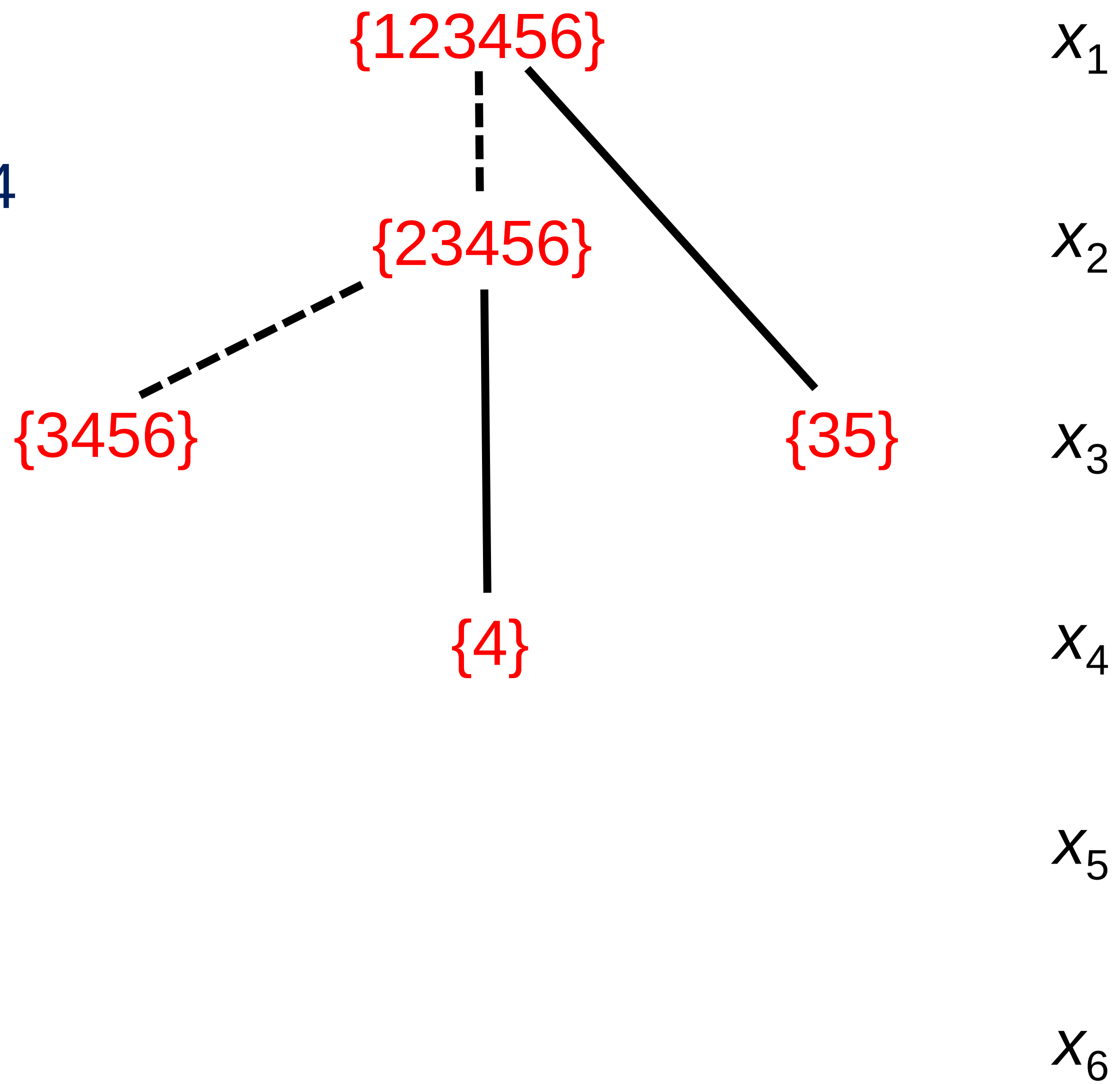
To build **relaxed** BDD, merge some additional nodes as we go along



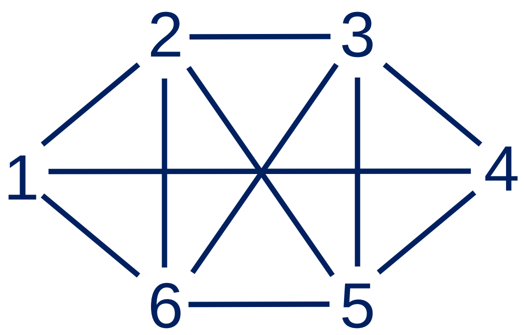


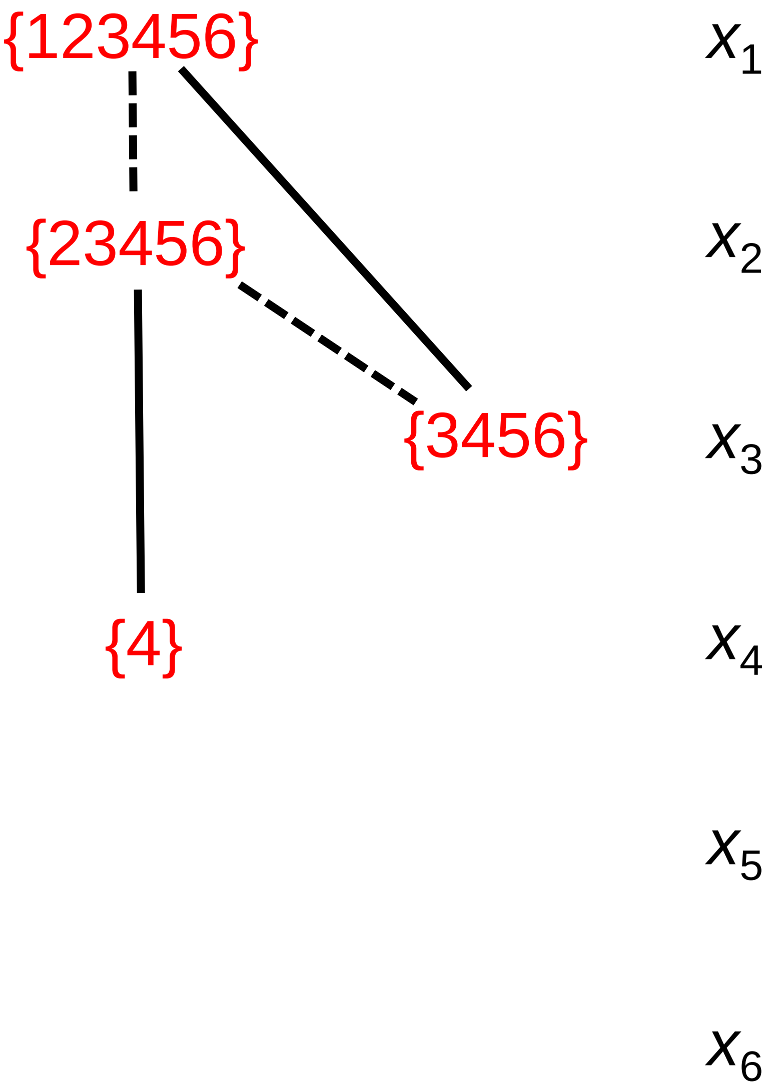
To build **relaxed** BDD, merge some additional nodes as we go along



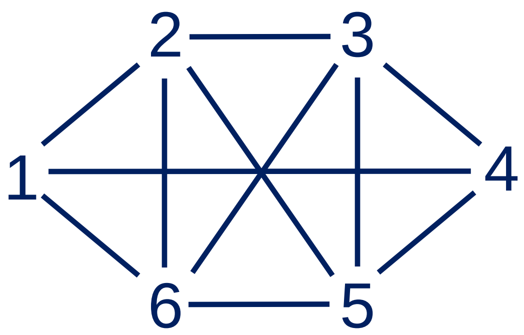


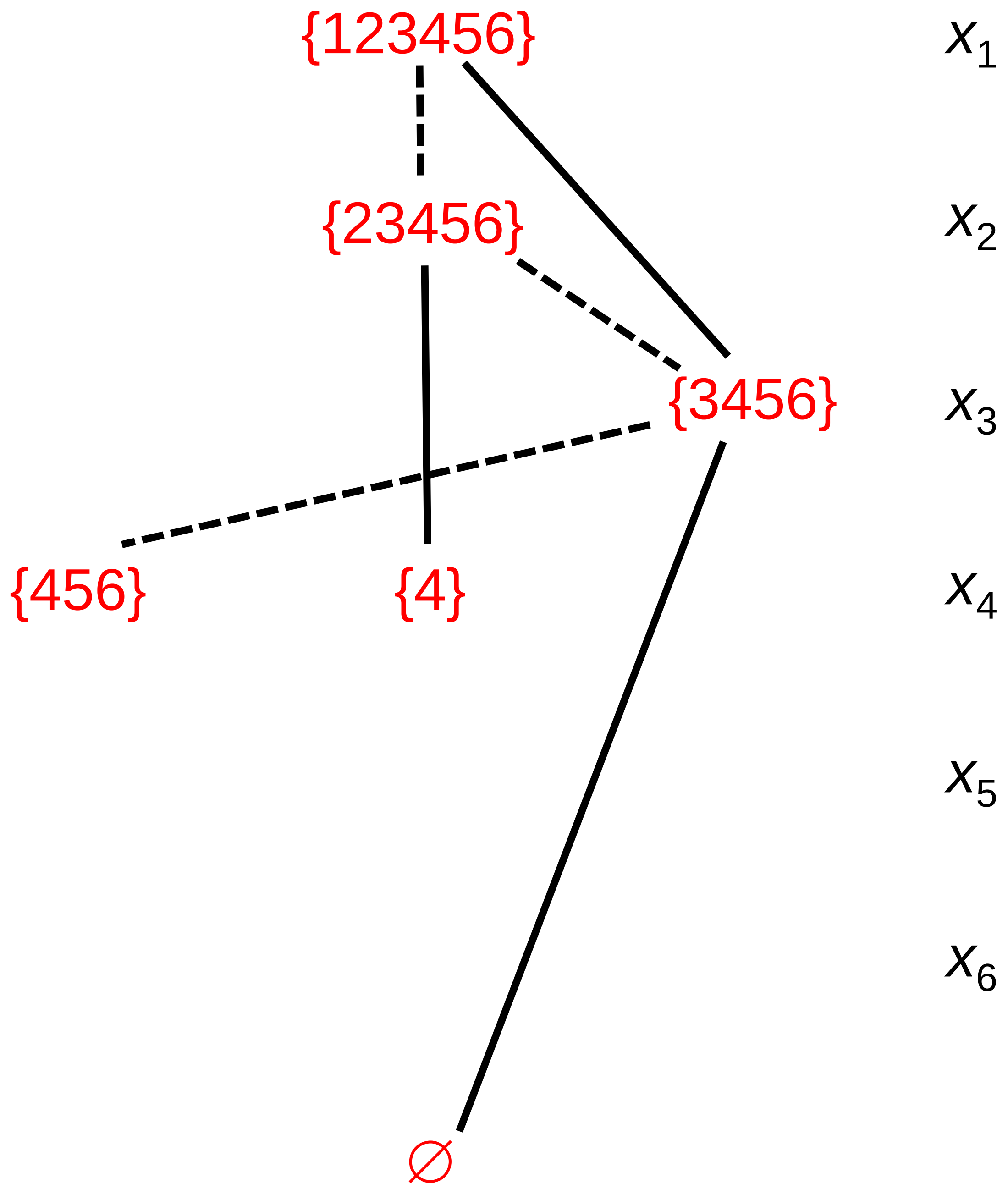
To build **relaxed** BDD, merge some additional nodes as we go along



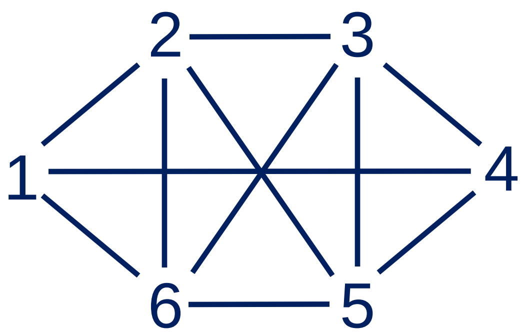


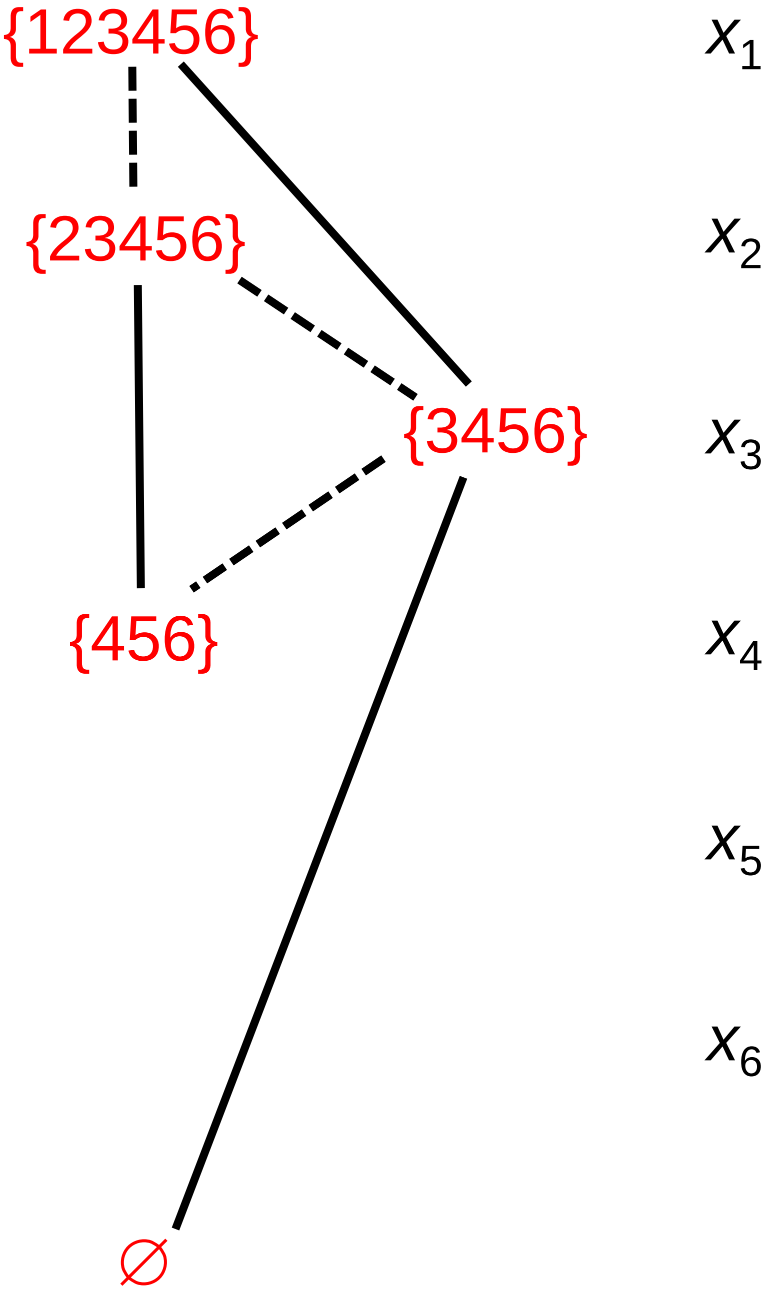
To build **relaxed** BDD, merge some additional nodes as we go along



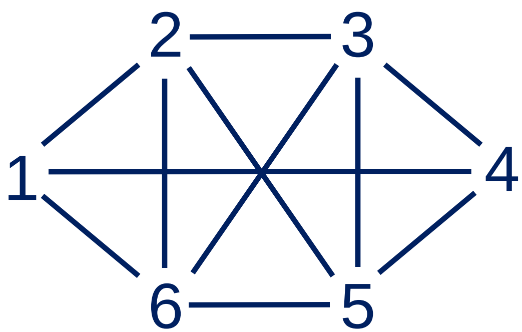


To build **relaxed** BDD, merge some additional nodes as we go along

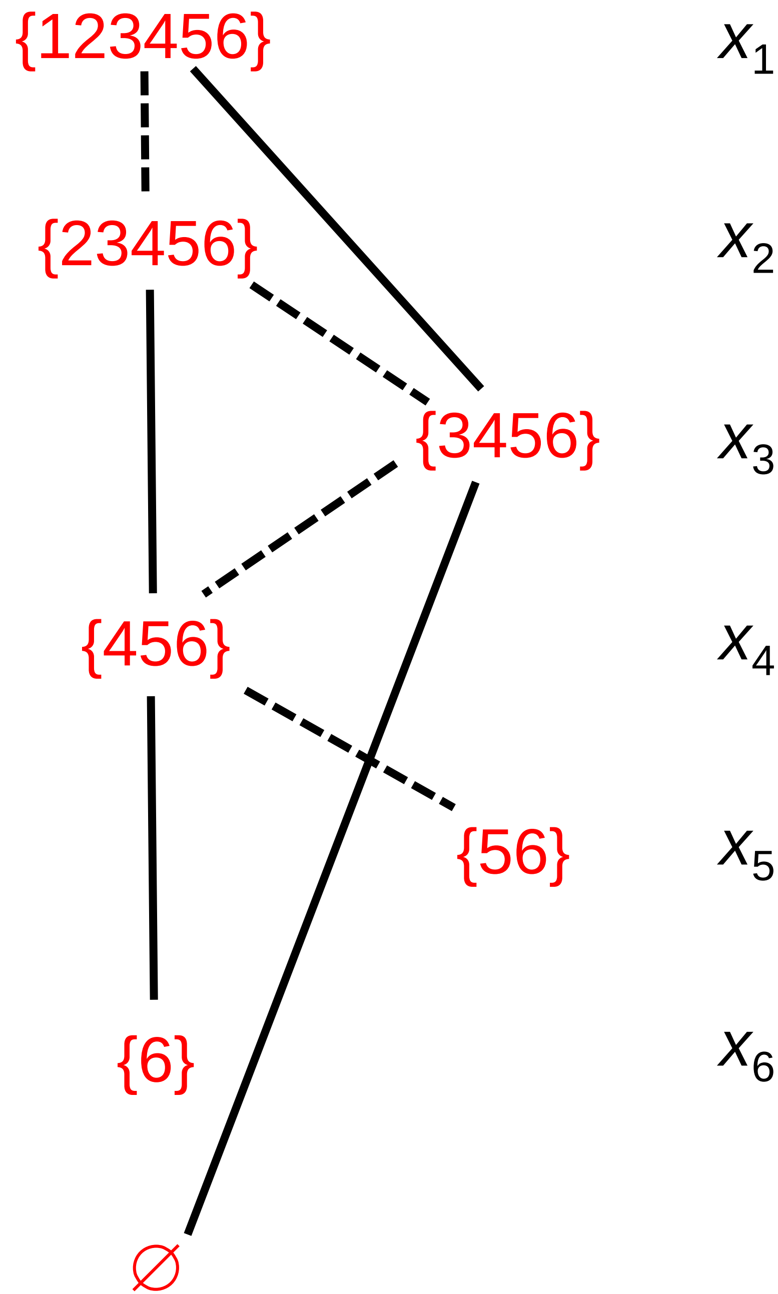


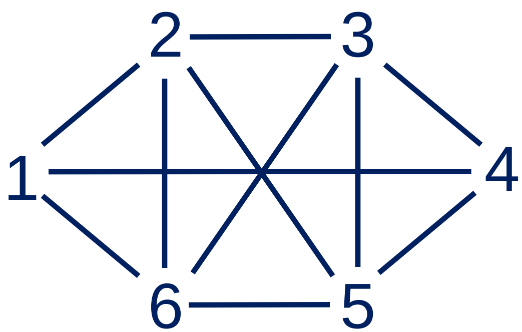


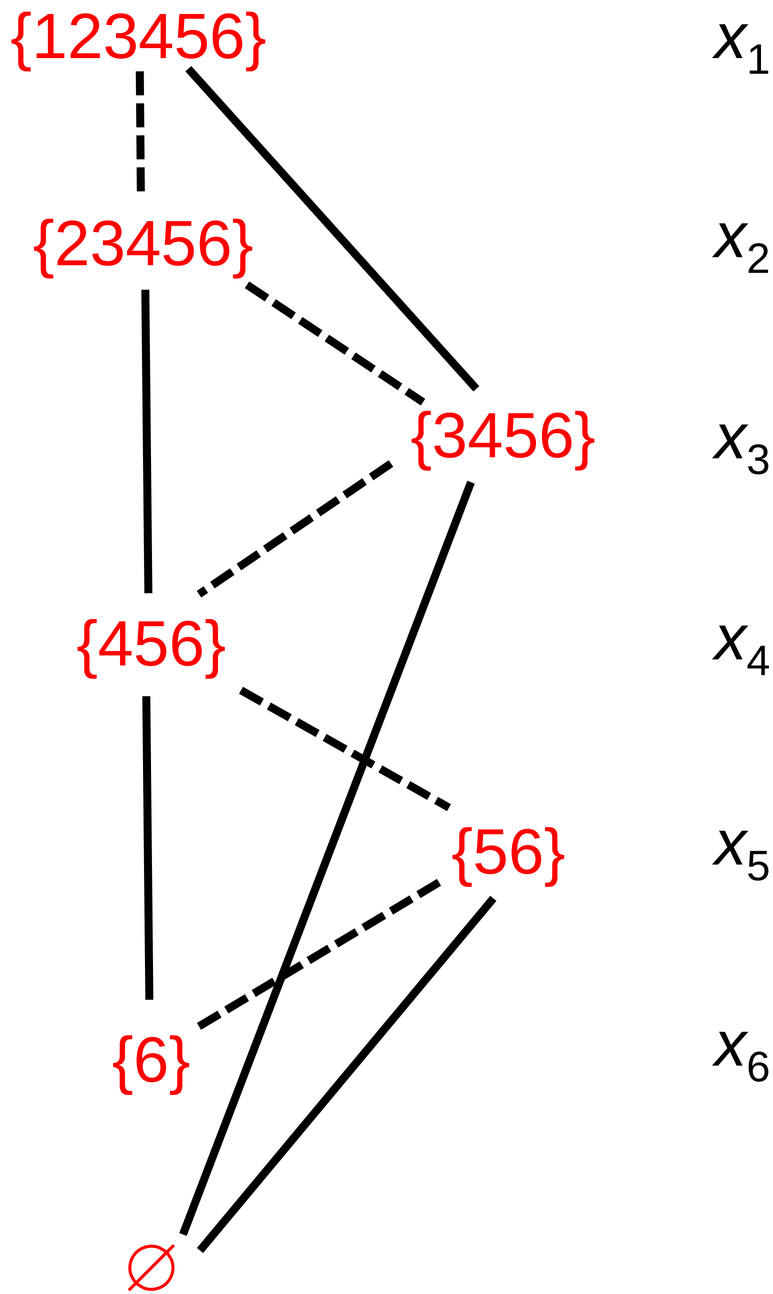
To build **relaxed** BDD, merge some additional nodes as we go along



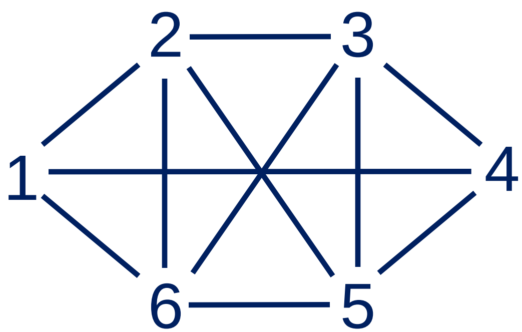
To build **relaxed** BDD, merge some additional nodes as we go along





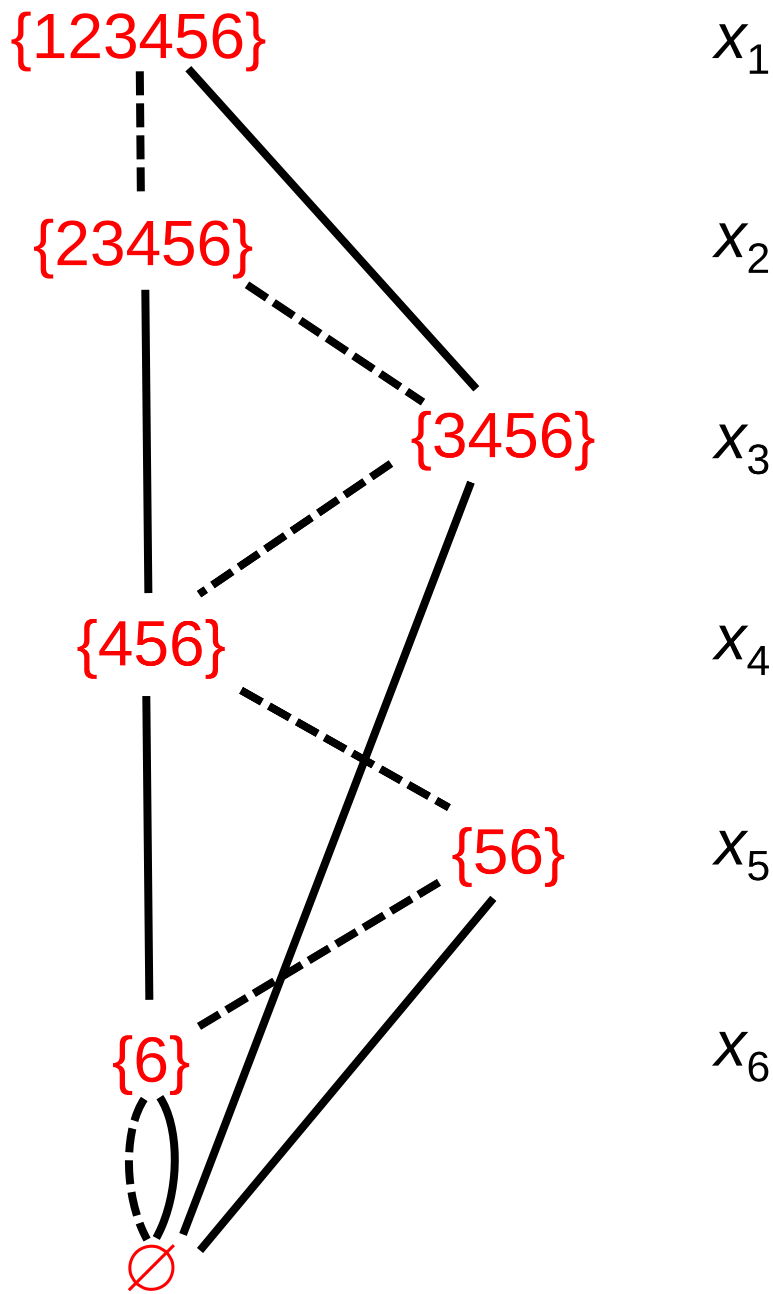


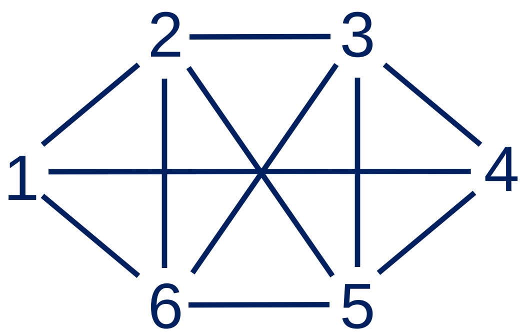
To build **relaxed** BDD, merge some additional nodes as we go along



Width = 1

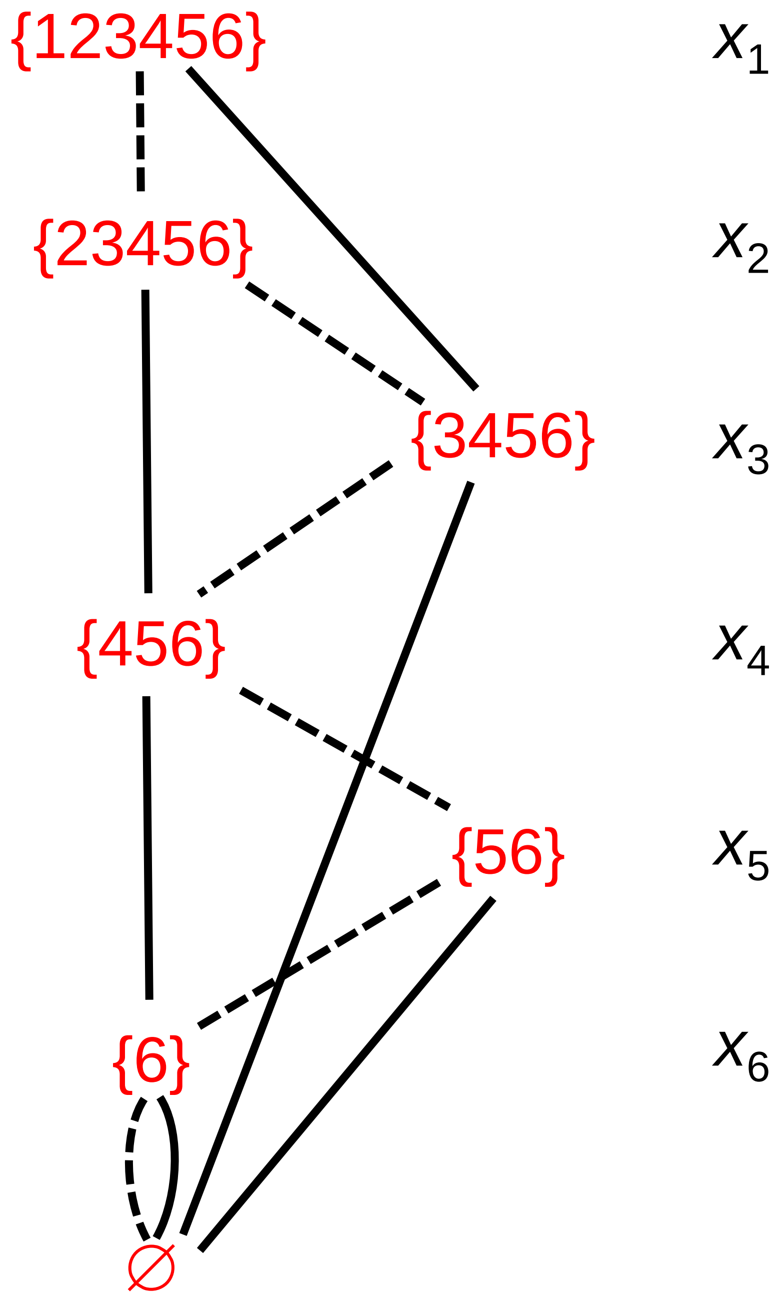
To build **relaxed** BDD, merge some additional nodes as we go along

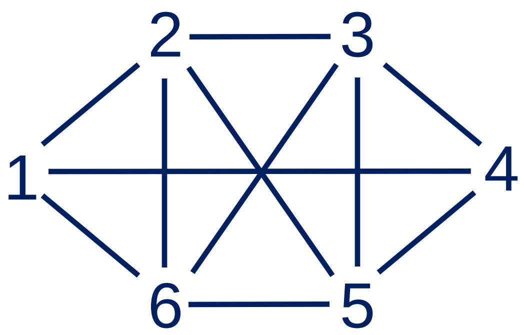


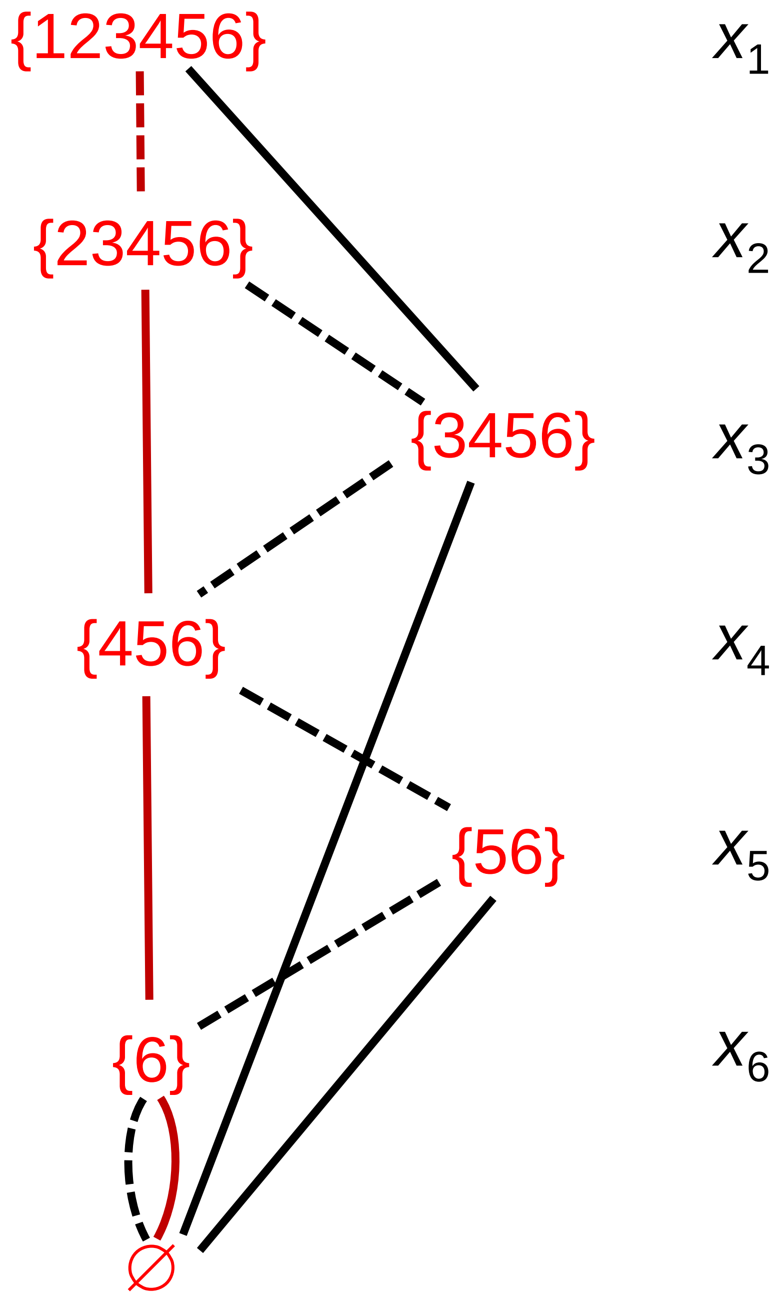


Width = 1

Represents 18 solutions, including 11 feasible solutions





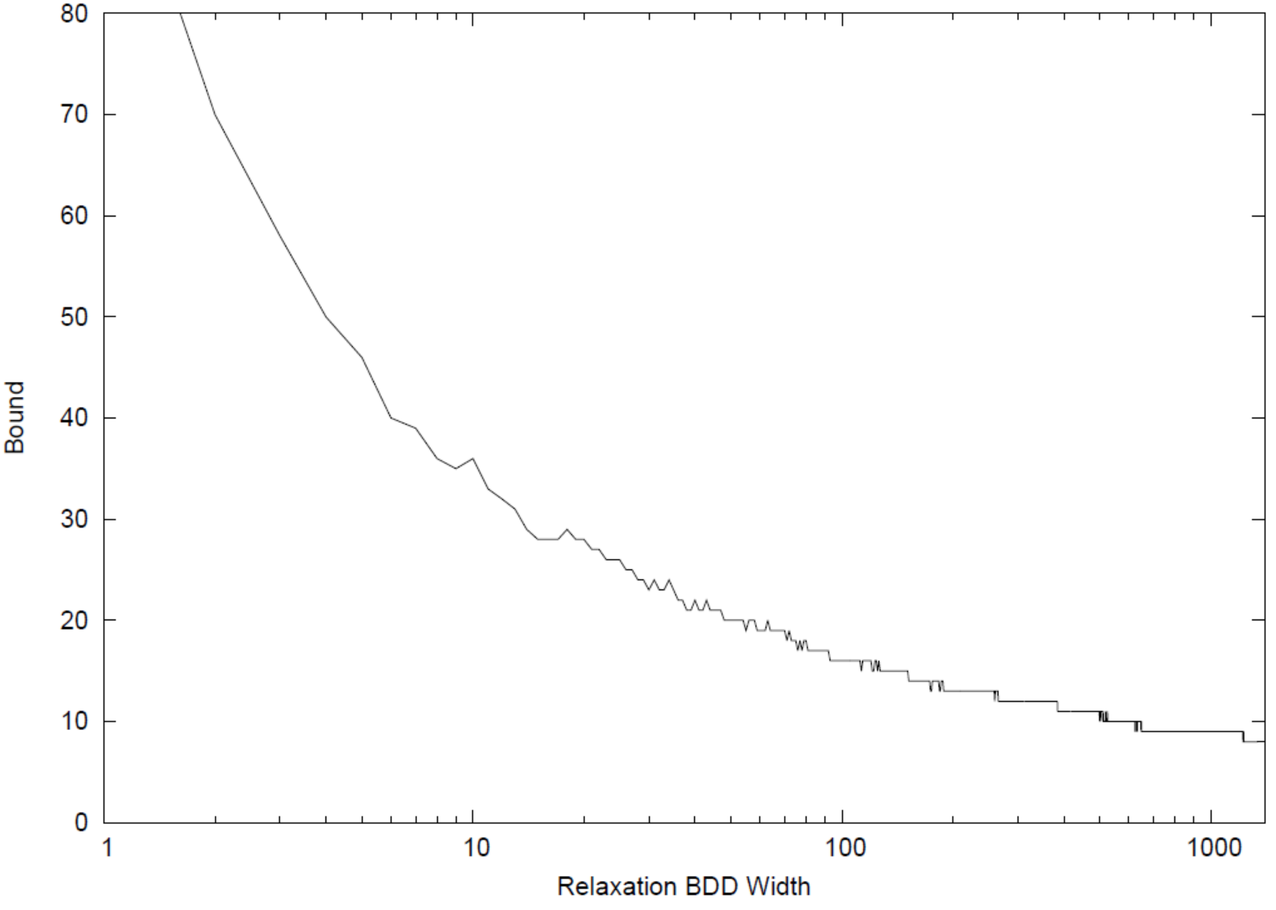


Width = 1

Longest path
gives bound
of 3 on optimal
value of 2

Bound vs. Width

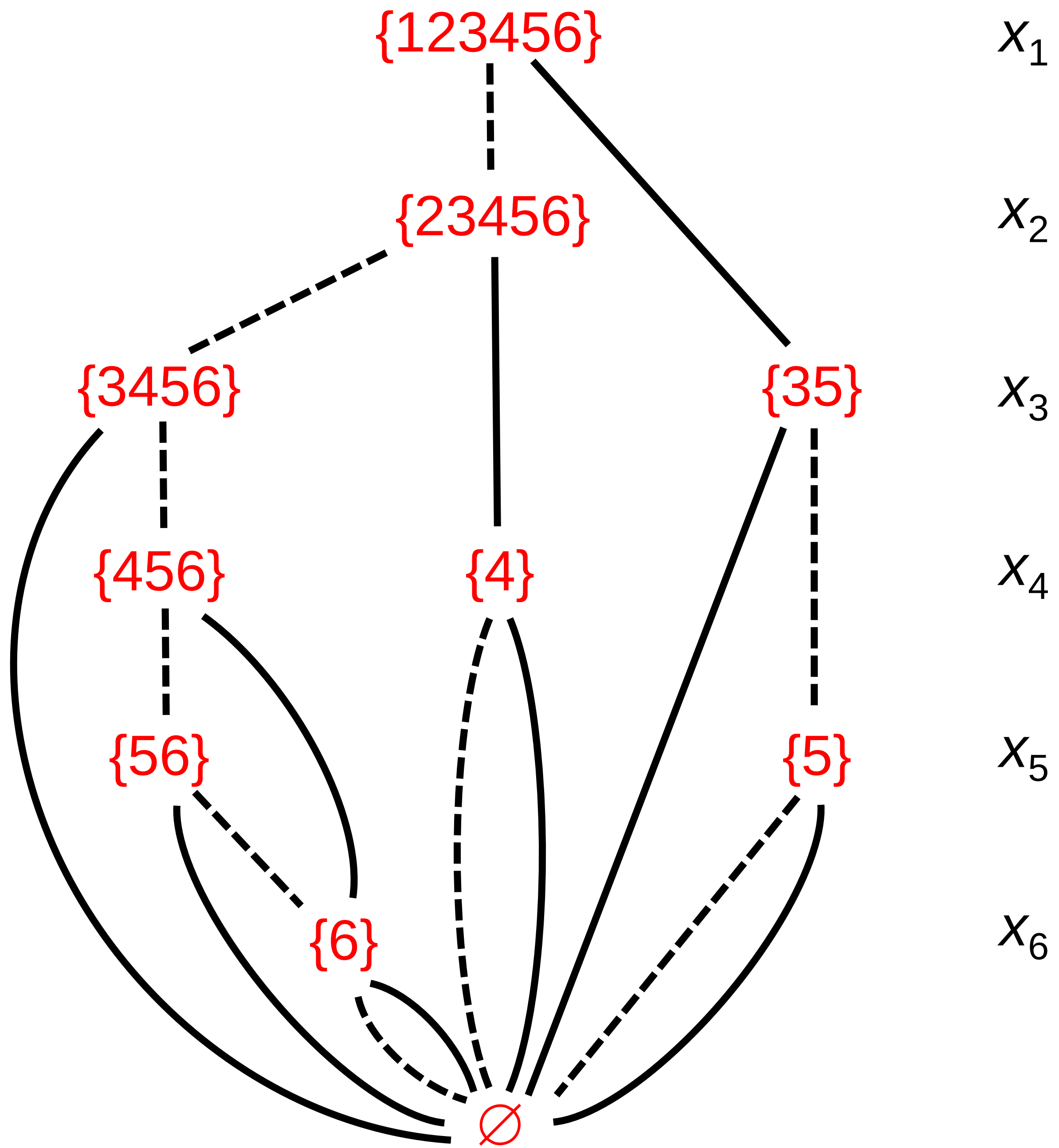
- Wider BDDs yield tighter bounds.
  - But take longer to build.



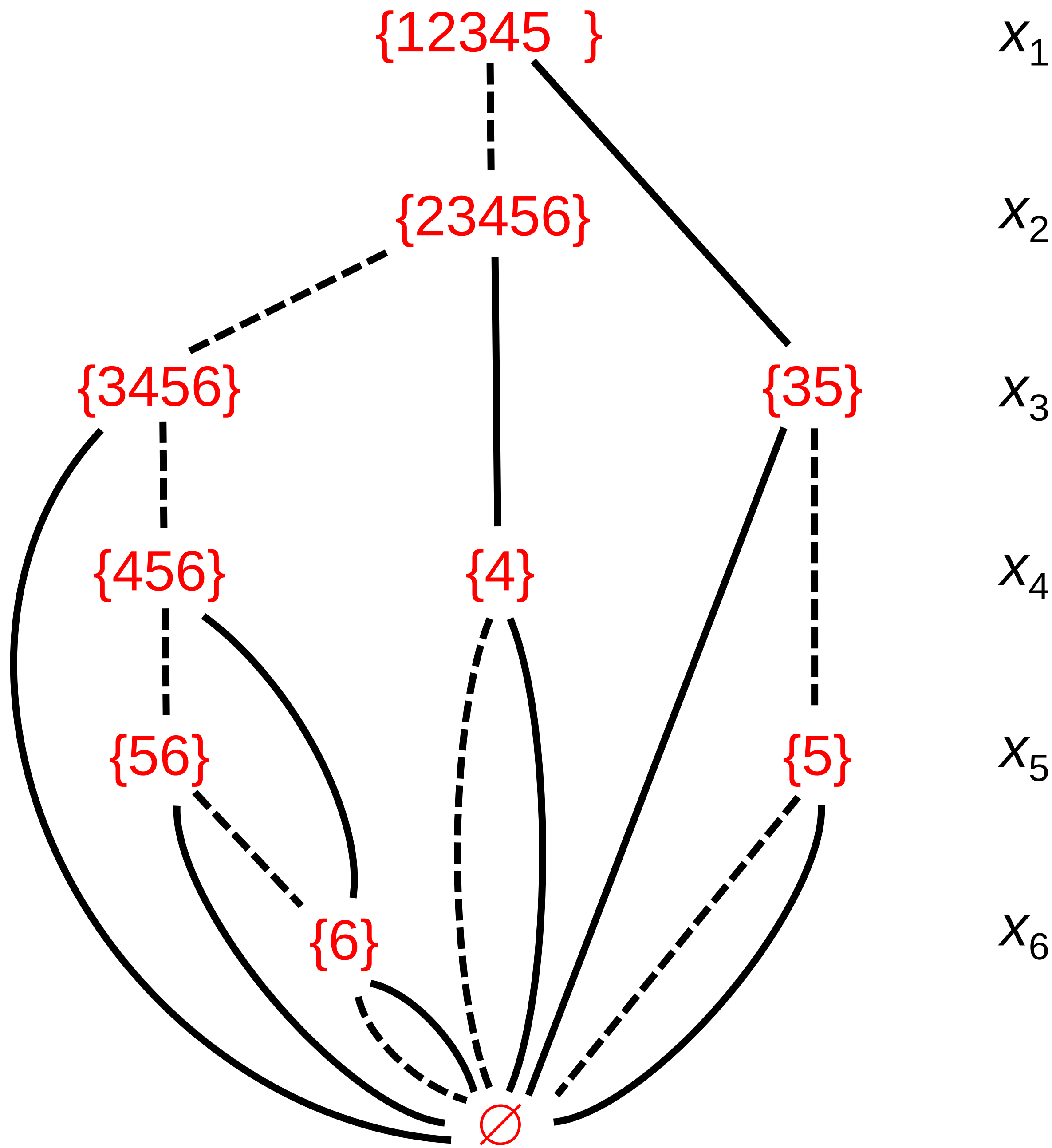
Propagation

- We can propagate by removing arcs from the decision diagram.
  - Rather than removing elements from variable domains.
  - More effective than traditional domain filtering.
  - More information propagated from one constraint to the next.

Suppose this is the relaxed decision diagram

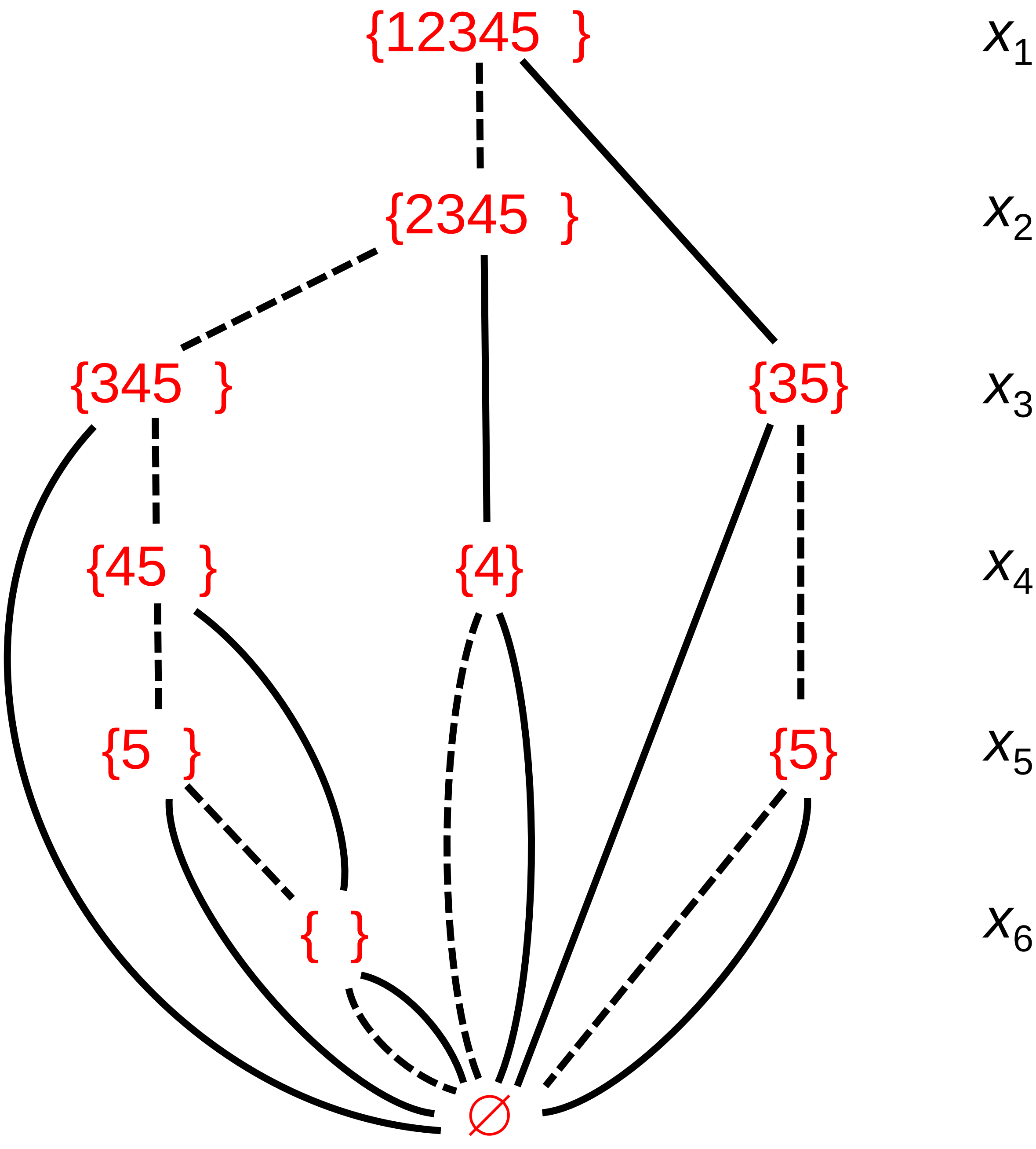


Suppose this is the relaxed decision diagram



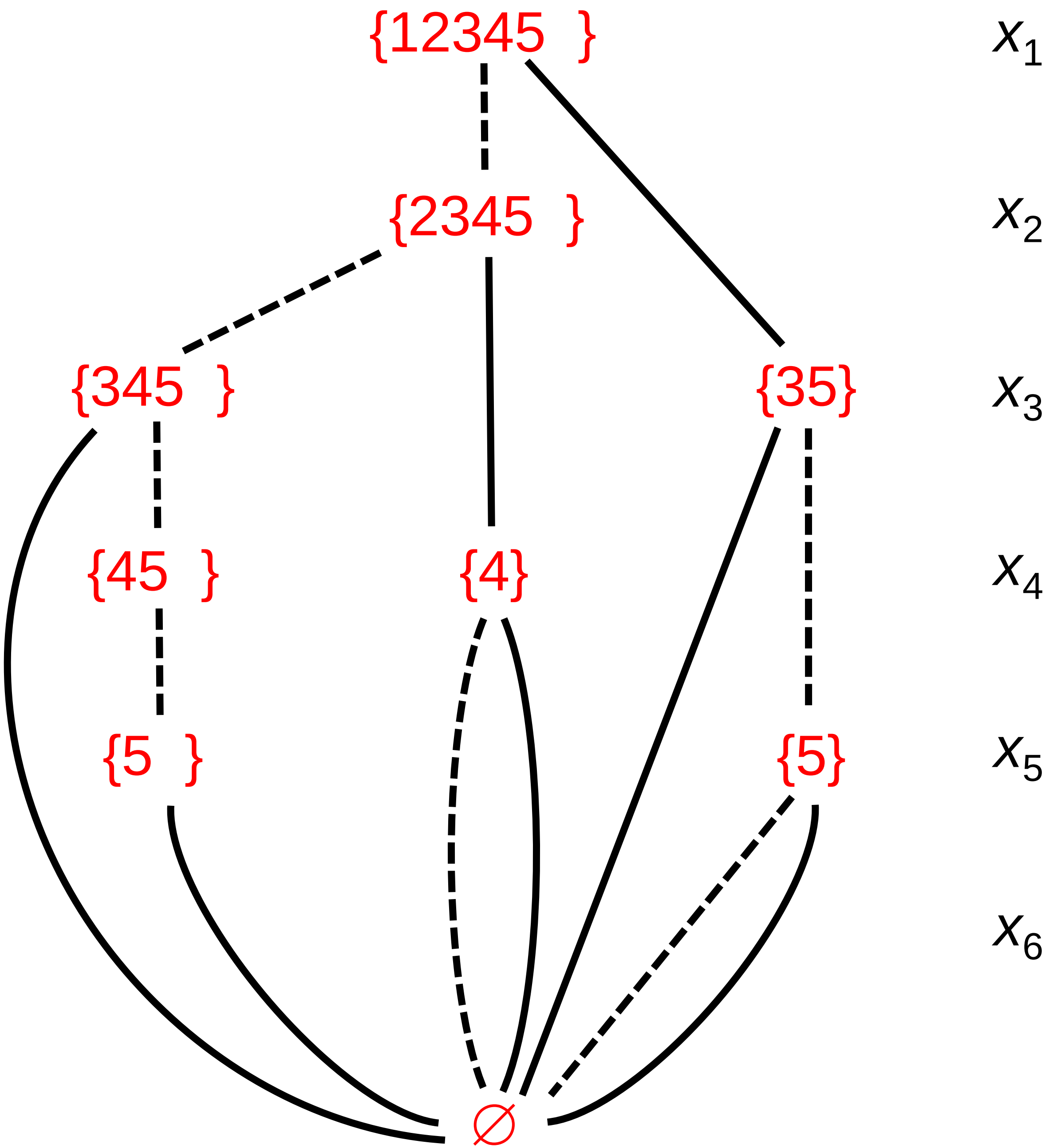
Suppose other constraints remove 6 from domain of x_1

Suppose this is the relaxed decision diagram



This propagates through the states and removes some arcs.

Suppose this is the relaxed decision diagram

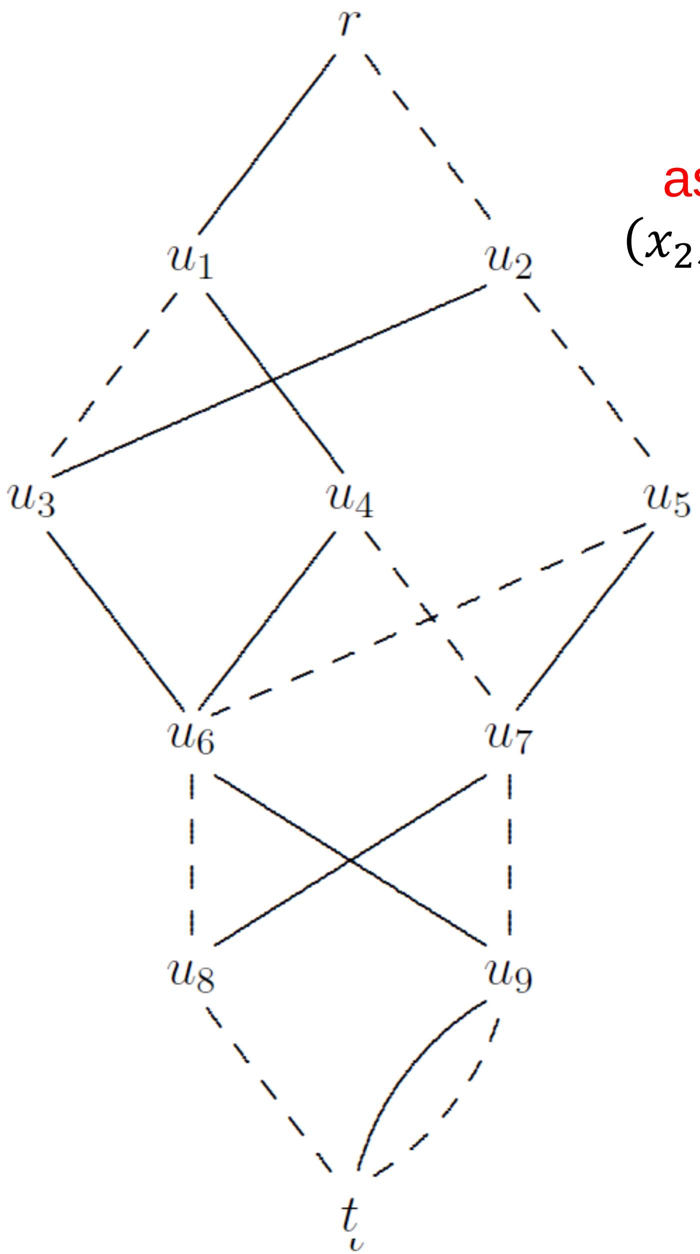


This propagates through the states and removes some arcs.

Separation Problem for BDDs

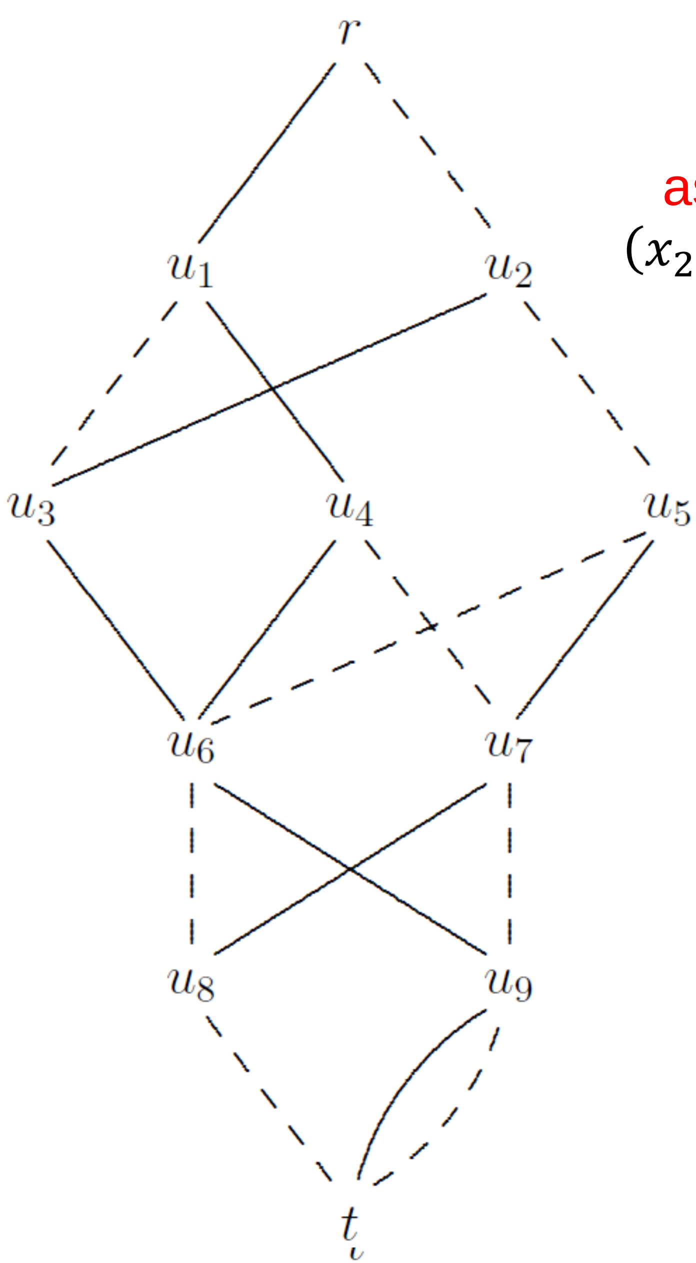
- Exclude a given partial assignment $x_i = \bar{x}_i$ for $i \in I$.
  - That is, remove all paths in which $x_i = \bar{x}_i$ for $i \in I$.
- Example…

Original BDD



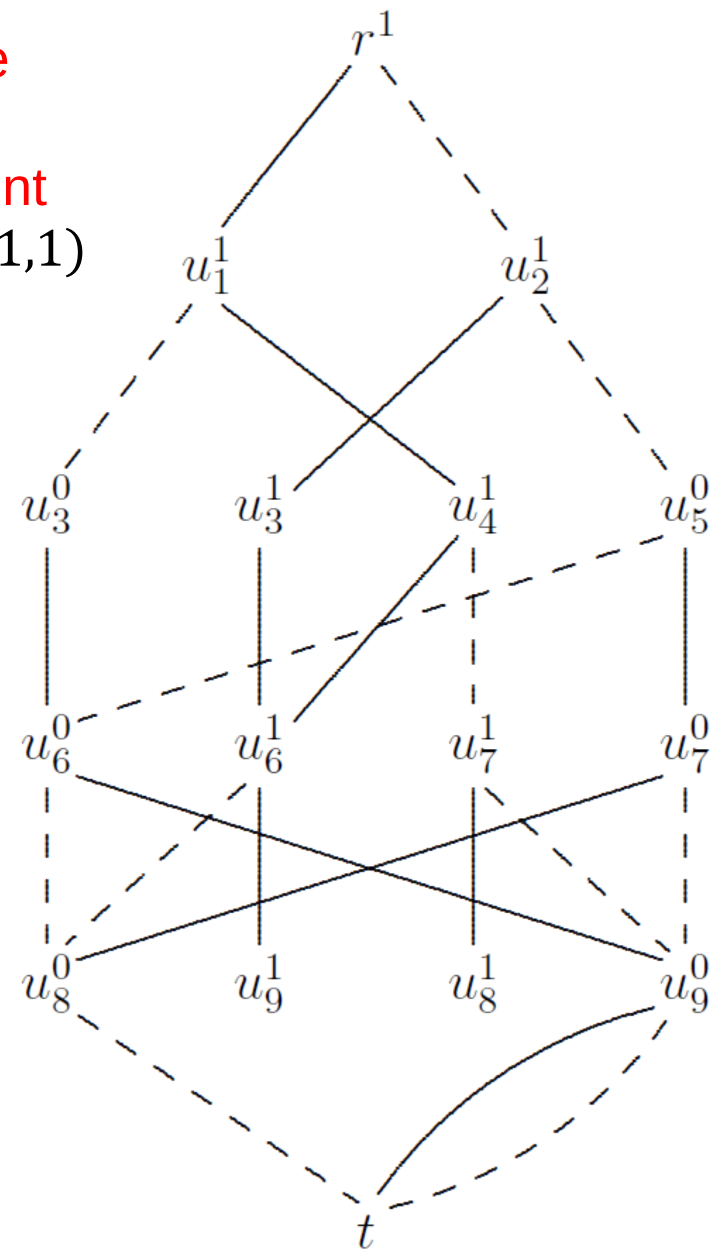
Remove partial assignment $(x_2, x_4) = (1,1)$

Original BDD



Remove partial assignment $(x_2, x_4) = (1,1)$

Separating BDD



Separation Algorithm

- In principle, a partial assignment can be separated by conjoining two BDDs.

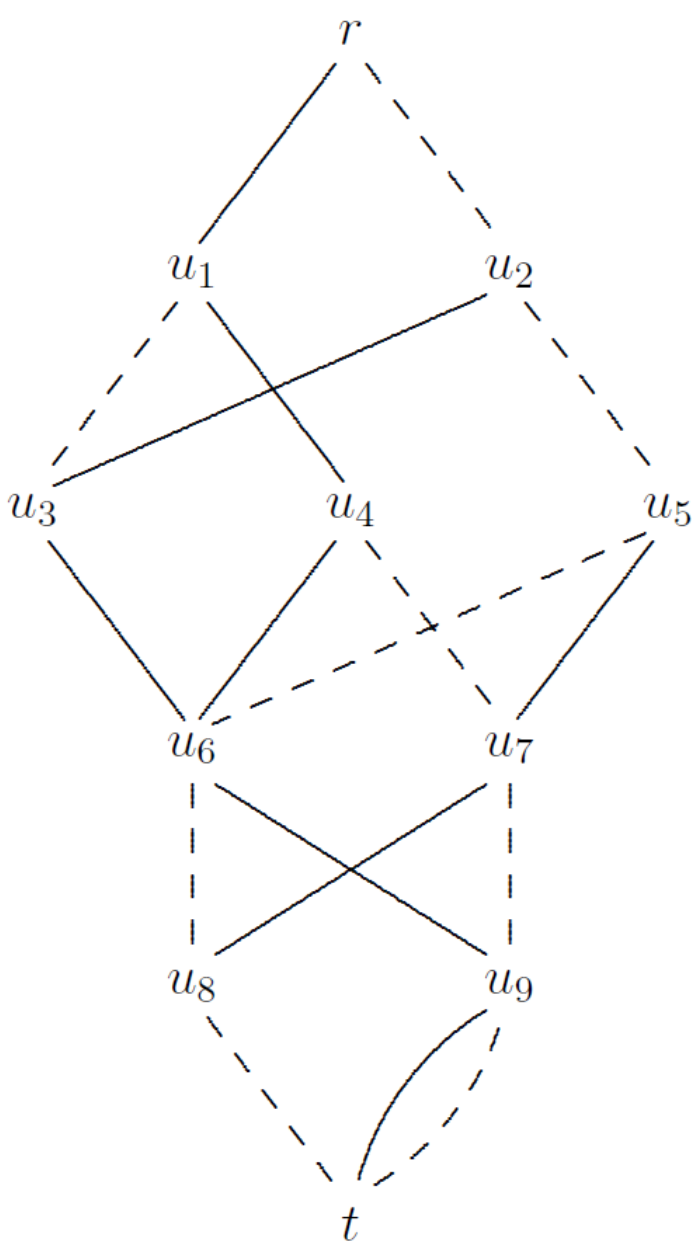
Original BDD $\wedge$ **Width-2 BDD** representing negation of partial assignment

  - However, this introduces an unnecessary data structure.

Separation Algorithm

- We will propose an algorithm specifically for BDD separation.
  - Exposes **essential logic** of separation.
  - Operates on **original data structure**.
  - Allows proof of **tighter bounds** on growth of the separating BDD as cuts are added.

Original BDD



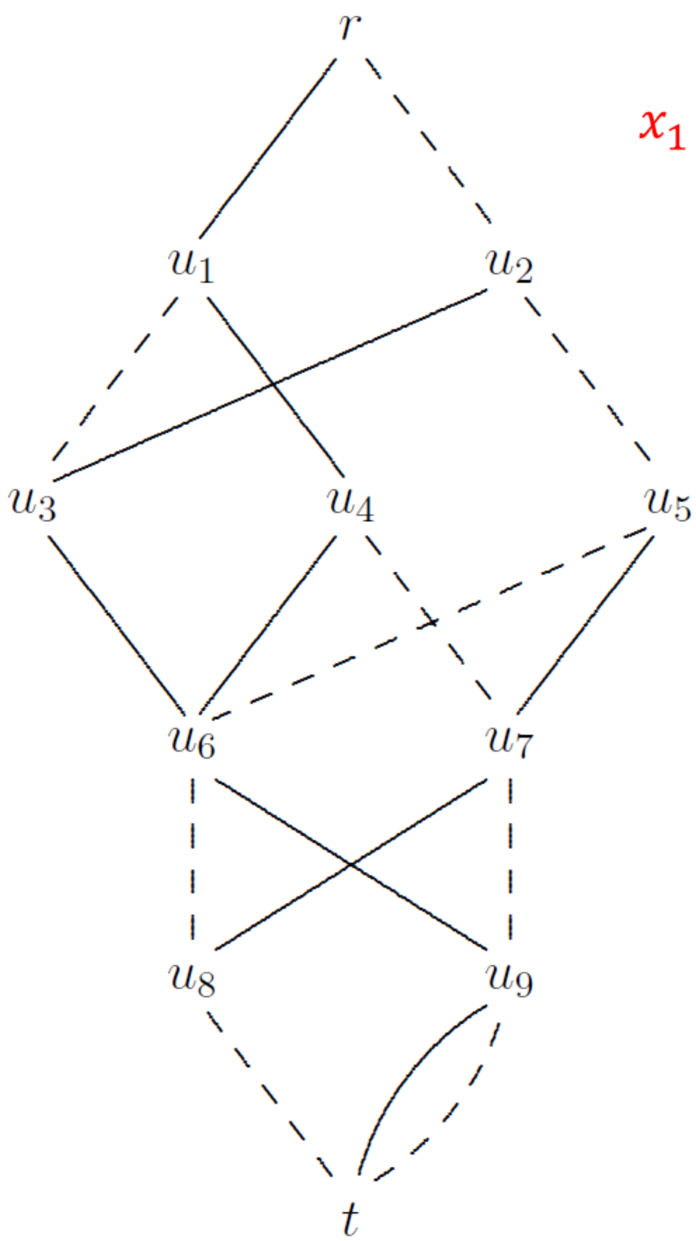
Separating BDD

A node has **state 1** when all incoming paths are excluded.

Otherwise **state 0**.

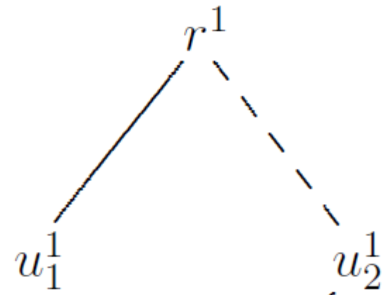
Assign state 1 to root node.

Original BDD



x_1 unrestricted

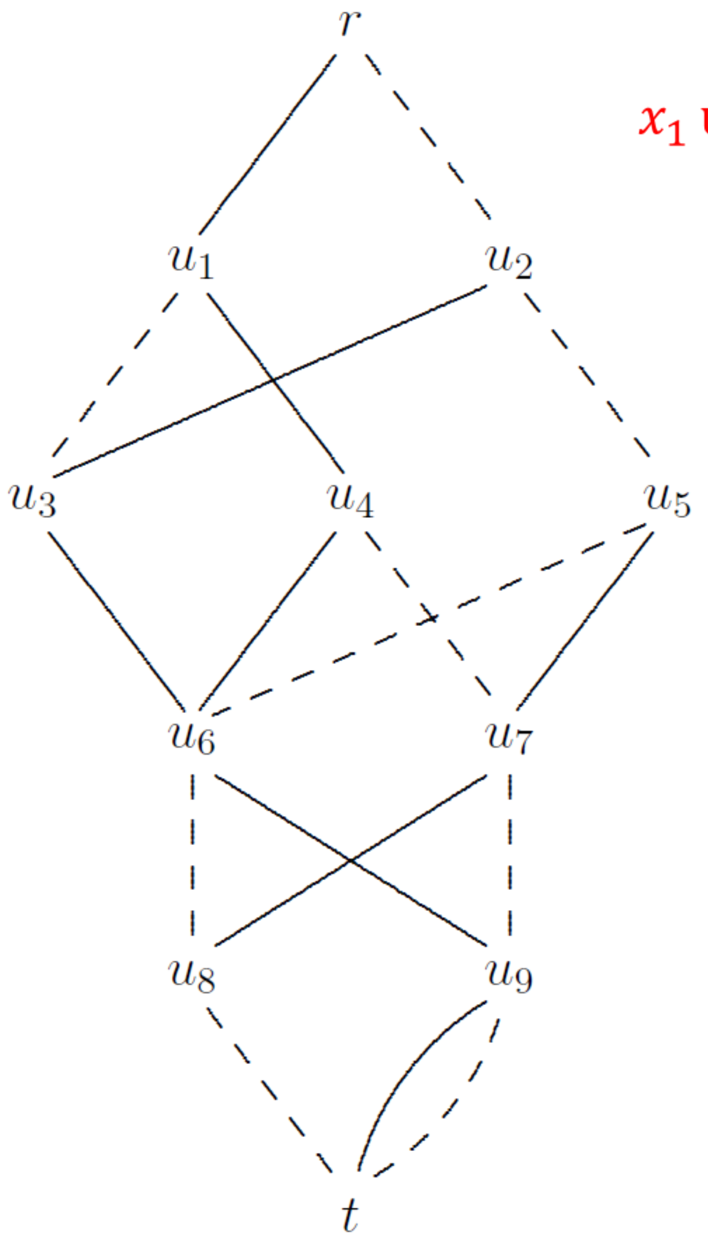
Separating BDD



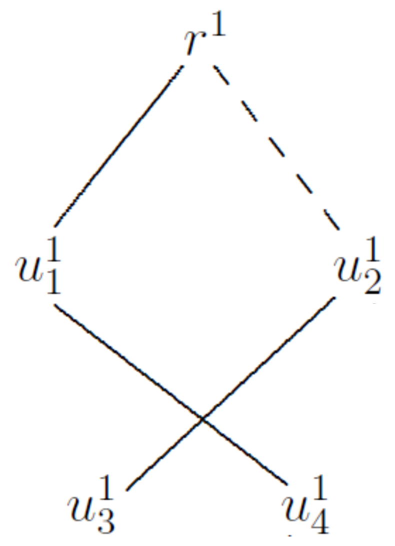
Duplicate arcs leaving *r* in original BDD.

Child nodes inherit state of parent node.

Original BDD



Separating BDD

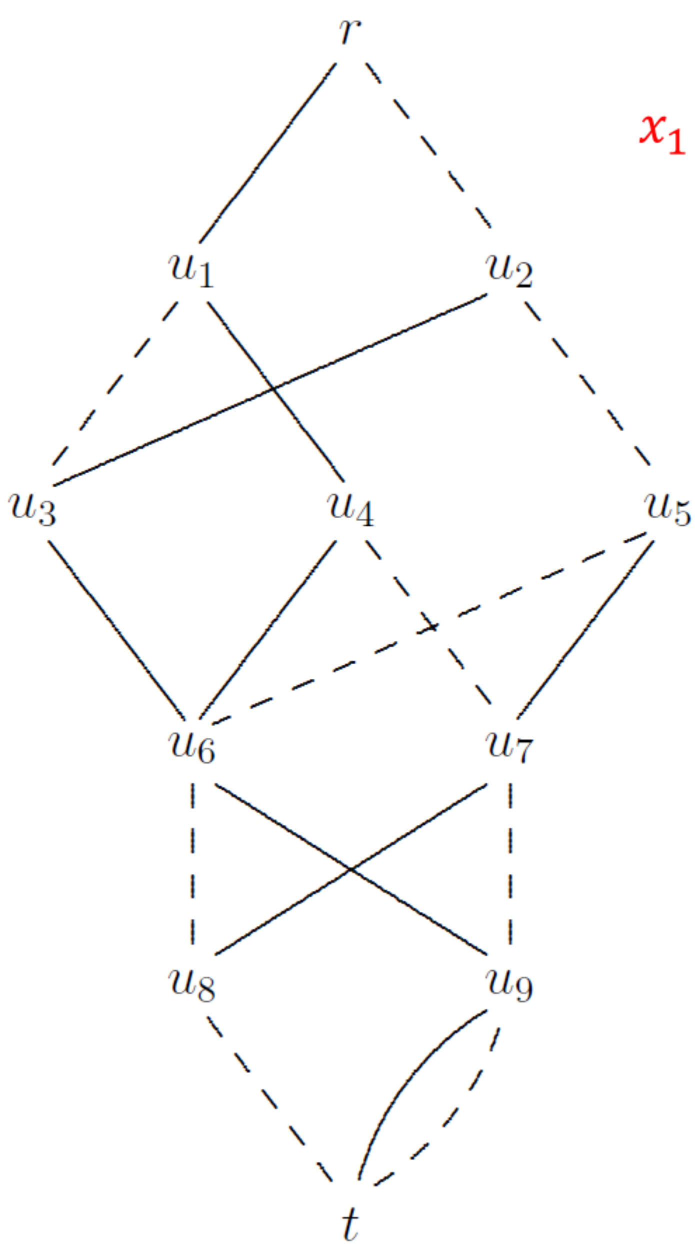


x_1 unrestricted

$x_2 \neq 1$

1-arcs from state 1 nodes preserve state 1

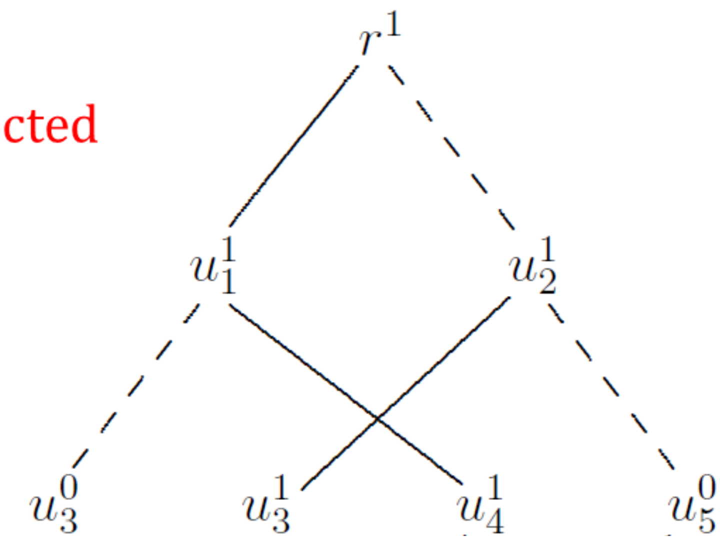
Original BDD



x_1 unrestricted

$x_2 \neq 1$

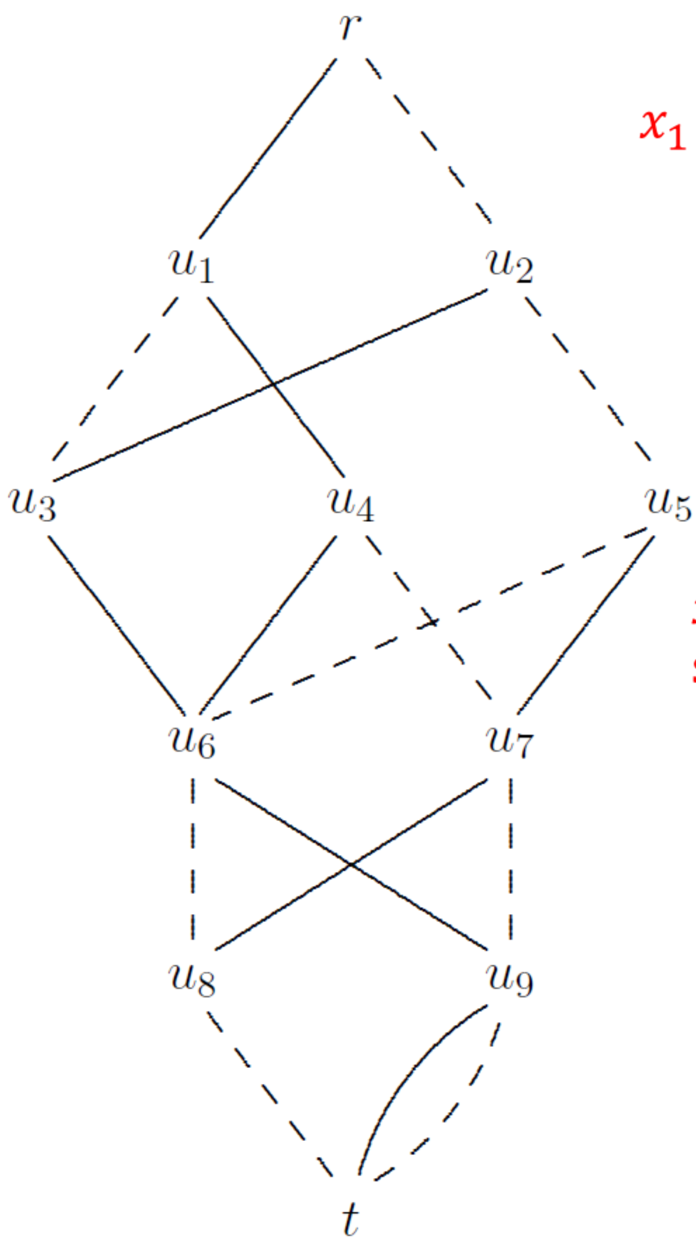
Separating BDD



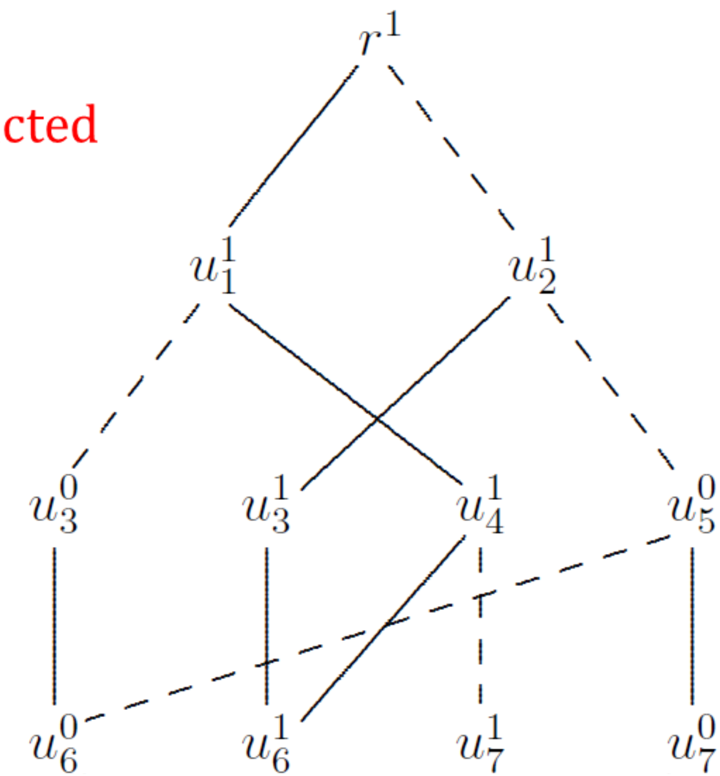
1-arcs from state 1 nodes preserve state 1

0-arcs from state 1 nodes switch to state 0

Original BDD



Separating BDD



x_1 unrestricted

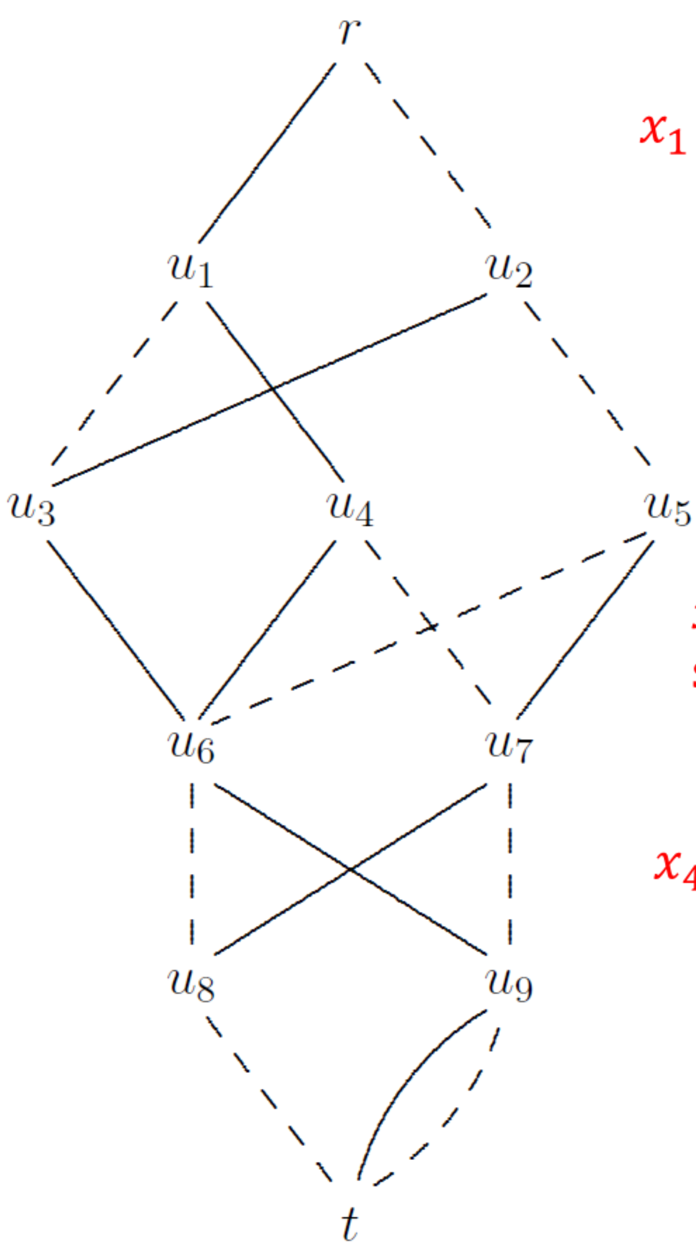
$x_2 \neq 1$

x_3 unre-
stricted

Duplicate arcs in original BDD.

Child nodes inherit state of parent node.

Original BDD



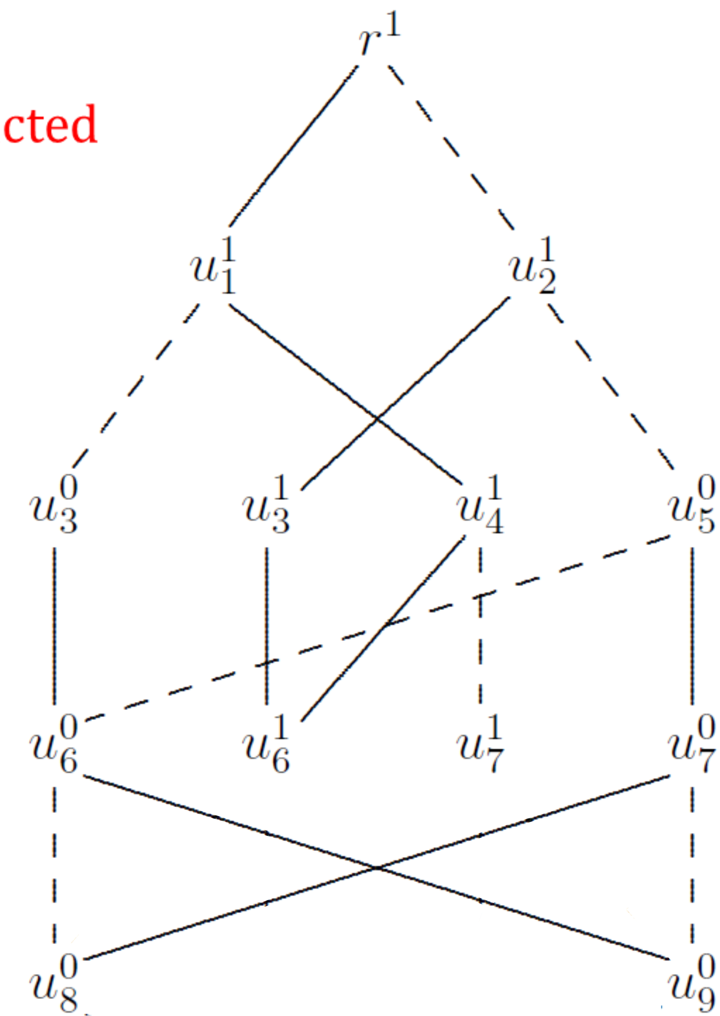
x_1 unrestricted

$x_2 \neq 1$

x_3 unre-
stricted

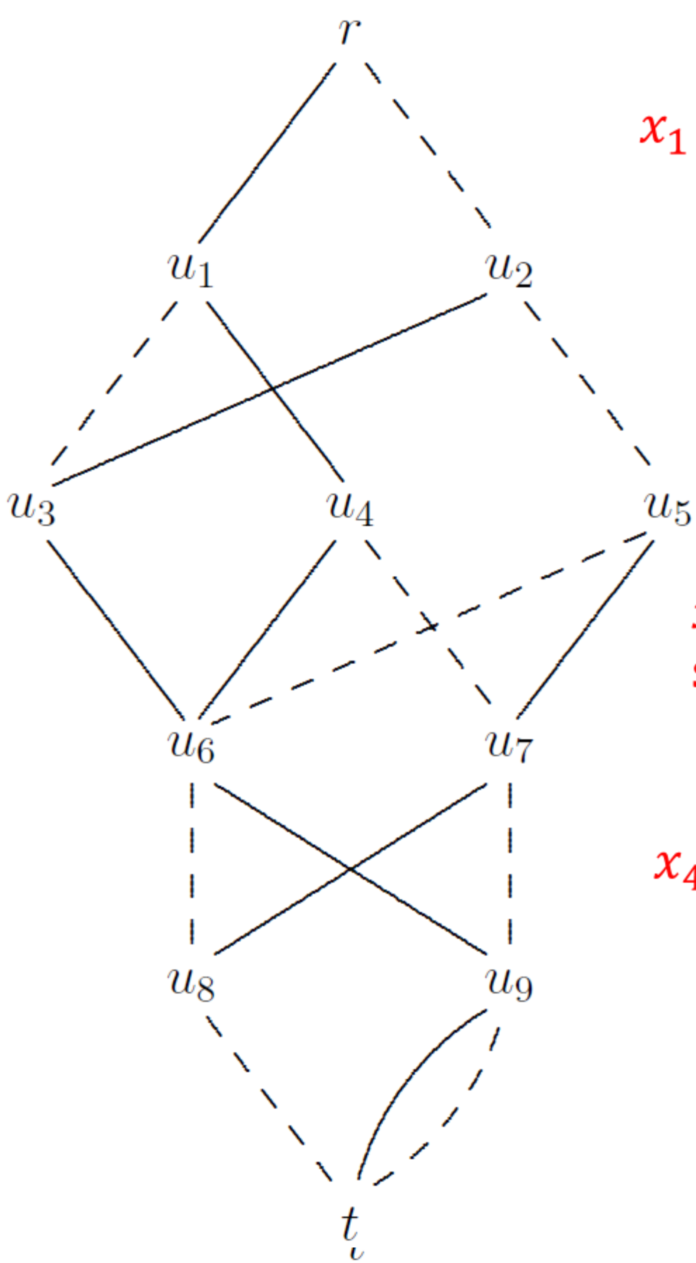
$x_4 \neq 1$

Separating BDD



Duplicate arcs from nodes
with state 0, preserving state.

Original BDD



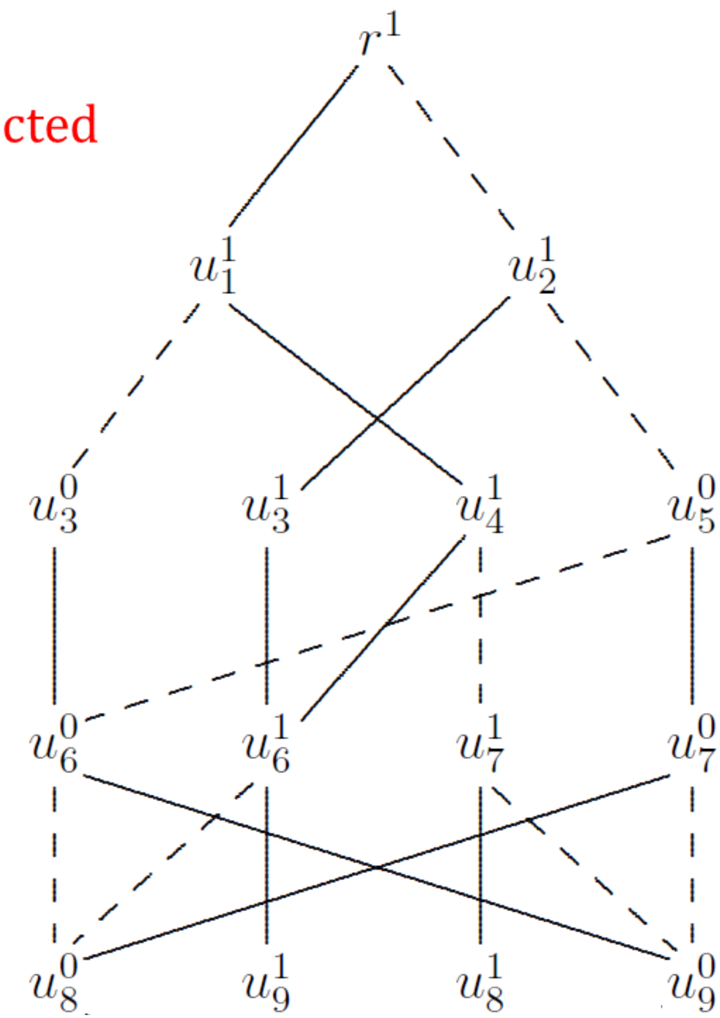
x_1 unrestricted

$x_2 \neq 1$

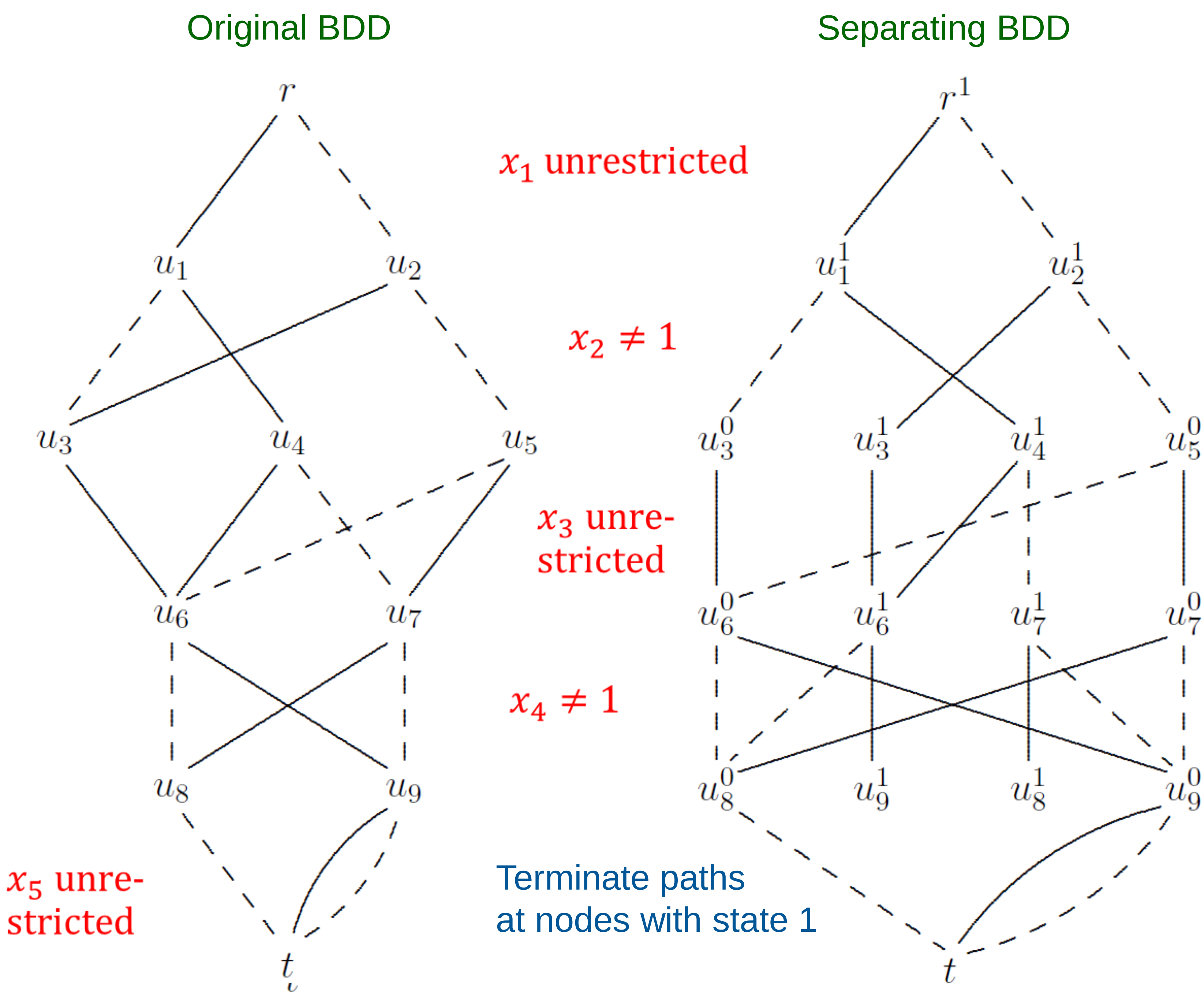
x_3 unre-
stricted

$x_4 \neq 1$

Separating BDD



1-arcs from state 1 nodes
preserve state
0-arcs switch state to 0.



Original BDD
Separating BDD
x_1 unrestricted
$x_2 \neq 1$
x_3 unrestricted
$x_4 \neq 1$
x_5 unrestricted
Terminate paths
at nodes with state 1
r
u_1
u_2
u_3
u_4
u_5
u_6
u_7
u_8
u_9
t
r^1
u_1^1
u_2^1
u_3^0
u_3^1
u_4^1
u_5^0
u_6^0
u_6^1
u_7^1
u_7^0
u_8^0
u_9^1
u_8^1
u_9^0
t

Size of Separating BDD

- We wish to separate from a given BDD all solutions *x* in which $x_i = \bar{x}_i$ for $i \in I$.

Theorem (obvious). The separating BDD is at most twice as large as the original BDD.

If only one solution is separated, the separating BDD has at most one additional node per layer.

- This refers to separating BDD created by the algorithm
  - Not necessarily a **reduced** (minimal) BDD.

Size of Separating BDD

- We wish to separate from a given BDD all solutions *x* in which $x_i = \bar{x}_i$ for $i \in I$.
- Let n_i be size of layer *i* of original BDD.
- Let *j*,*k* be smallest, largest indices in I.

Theorem (not so obvious).

Size of layer *i* of separating BDD $\leq \begin{Bmatrix} n_i + \varphi_i \text{ if } j \leq i \leq k \\ n_i \ \text{ otherwise} \end{Bmatrix}$

where $\varphi_i = \begin{Bmatrix} \min\{n_i, \varphi_{i-1}\} & \text{if } i-1 \in I \\ \min\{n_i, 2\varphi_{i-1}\} & \text{otherwise} \end{Bmatrix}$

Size of Separating BDD

- We wish to separate from a given BDD all solutions x in which $x_i = \bar{x}_i$ for $i \in I$.
- Let n_i be size of layer i of original BDD.
- Let j,k be smallest, largest indices in I.

Corollary Portion of BDD outside the range of indices in I is unaffected by separation.

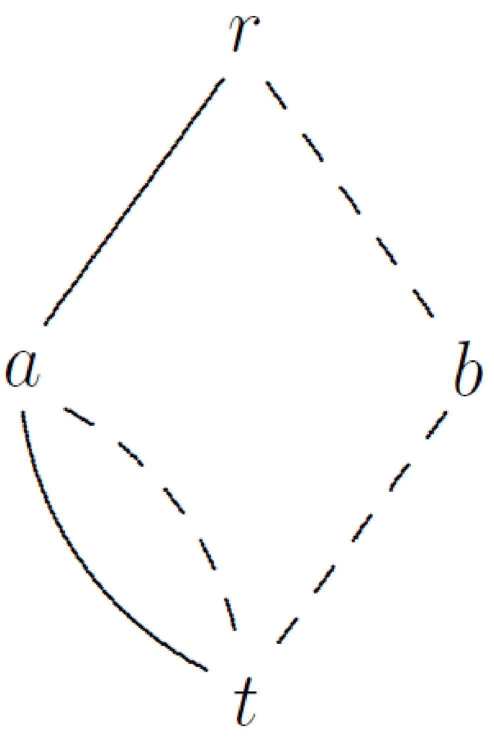
- This will be useful in decomposition methods.

Reduced Separating BDD

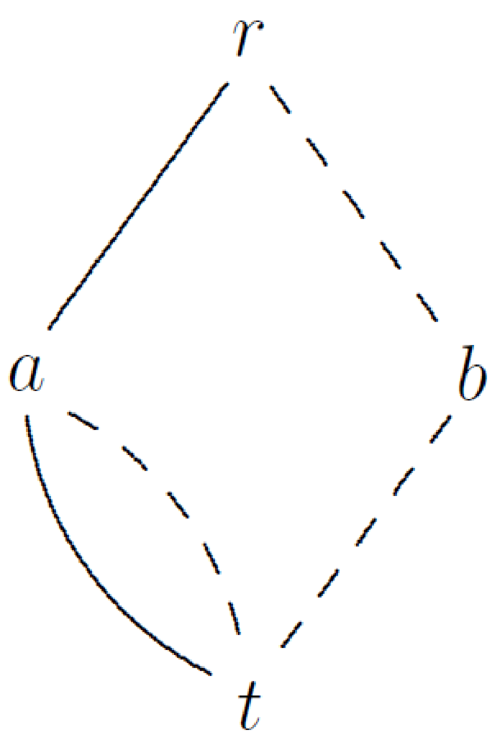
- Separating BDD generated by the algorithm need not be **reduced**.
  - The reduced BDD for a Boolean function is the smallest BDD that represents the function.
  - It is unique.

- For example…

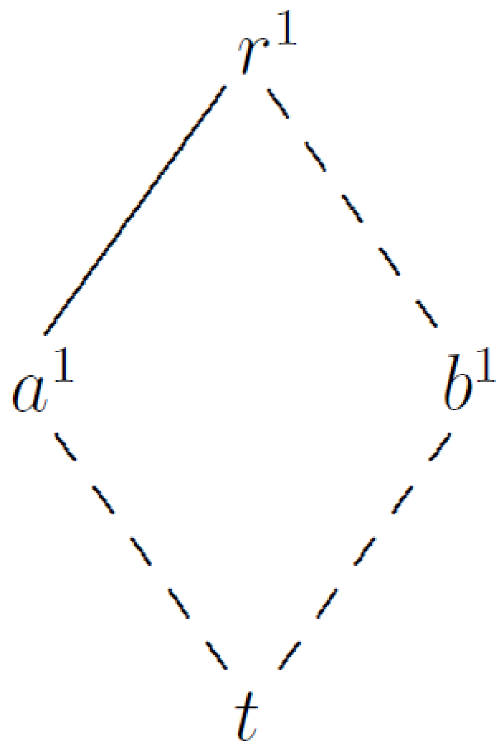
Original BDD



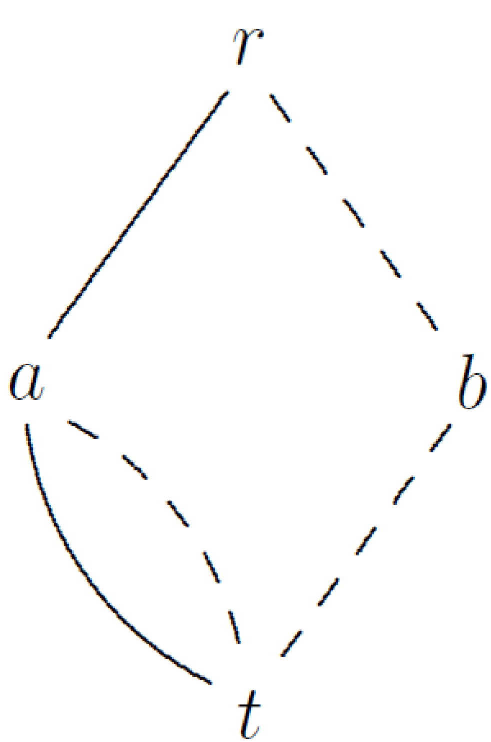
Original BDD



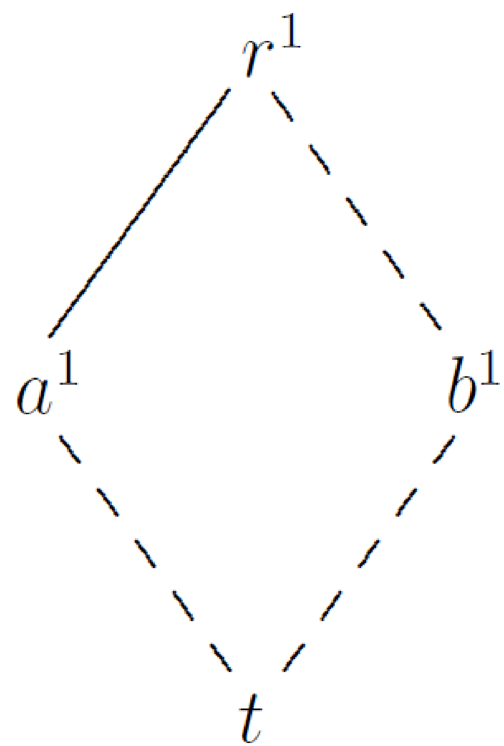
BDD that
separates x_2 = 1
as generated
by algorithm



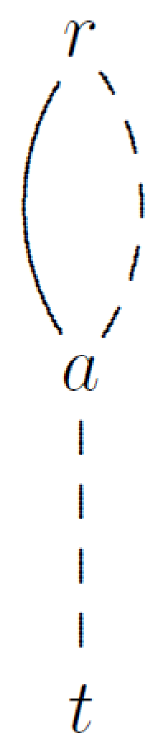
Original BDD



BDD that separates $x_2 = 1$ as generated by algorithm



Reduced form of separating BDD



Growth of BDD

- Key question: How fast does the separating BDD grow when a sequence of partial solutions are separated?
  - Traditional relaxation grows **linearly**.
  - One inequality constraint added per solution separated.
  - Will the separating BDD grow linearly?

Worst-Case Growth

- Can **reduced** separating BDD grow exponentially?
  - Yes

- Example
  - Start with BDD that represents all Boolean vectors (width 1).
  - Separate: $(1,*,*,\ldots,*,*,1)$

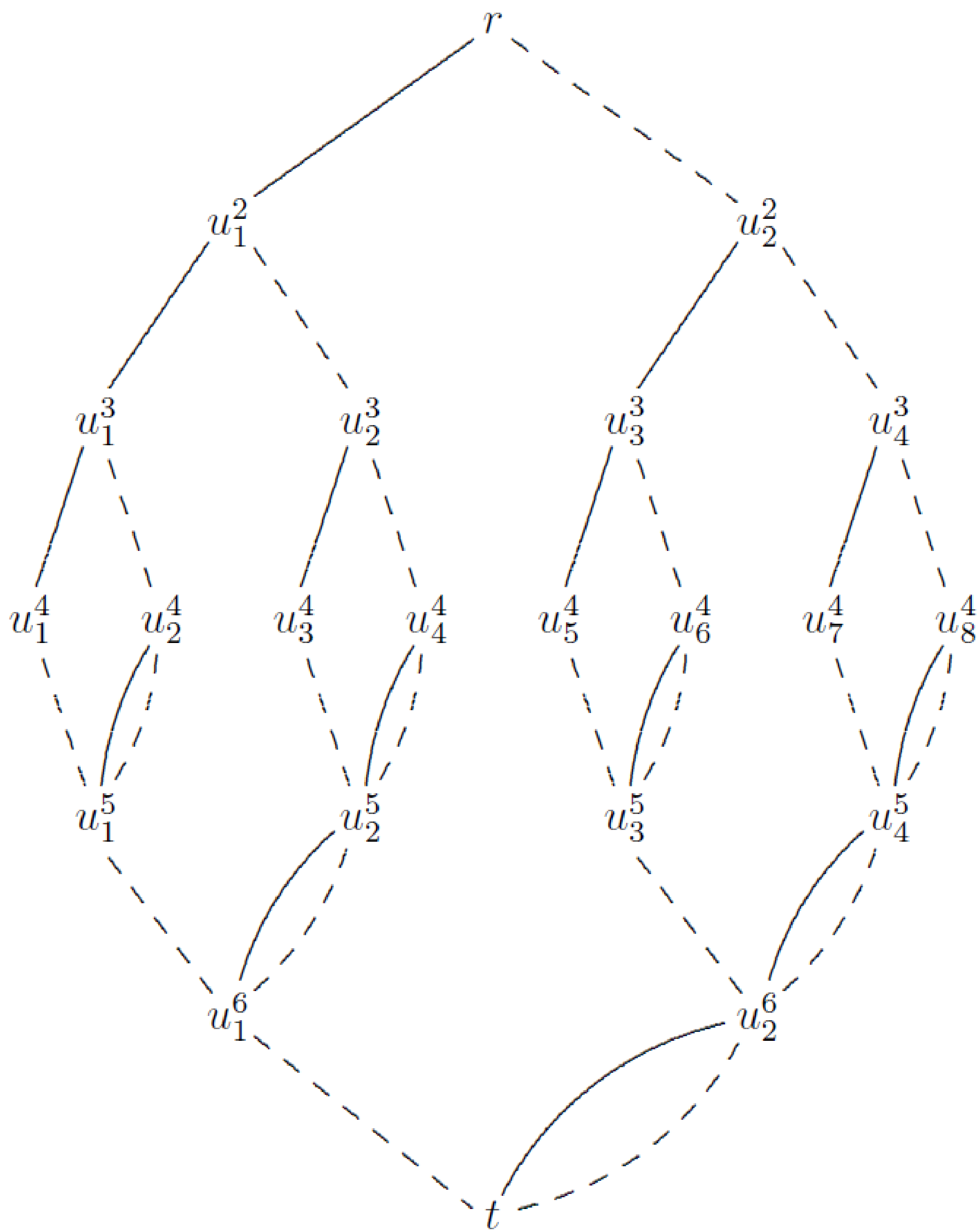
    $(*,1,*,\ldots,*,1,*)$

    $(*,*,1,\ldots,1,*,*)$

    etc.

Reduced BDD
for n = 6
variables.

It has width $2^{n/2}$



Empirical Growth

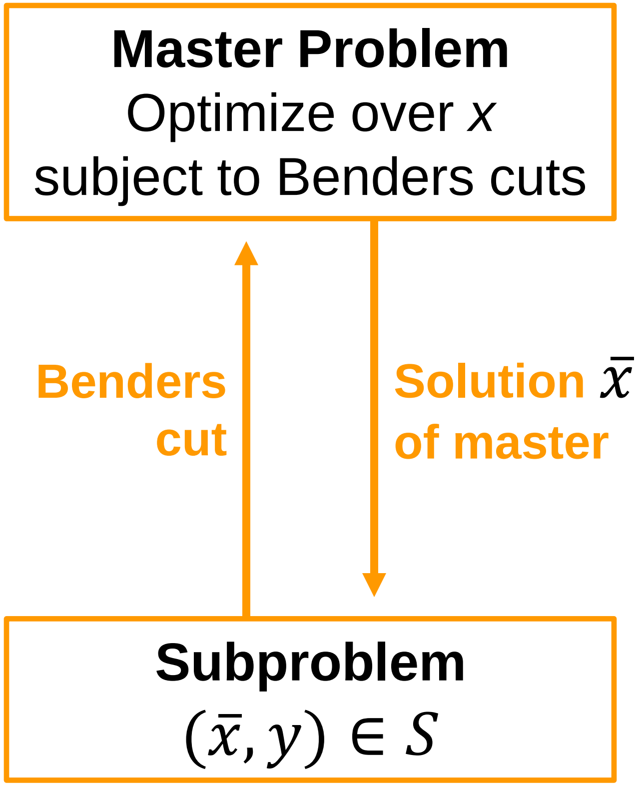
- How fast does the separating BDD grow in a realistic optimization algorithm?
  - We will look at a **logic-based Benders** algorithm
  - …for the **home health care delivery problem**

Benders Decomposition

- Logic-based Benders decomposition is a generalization of classical Benders.
  - Address a simplified problem:

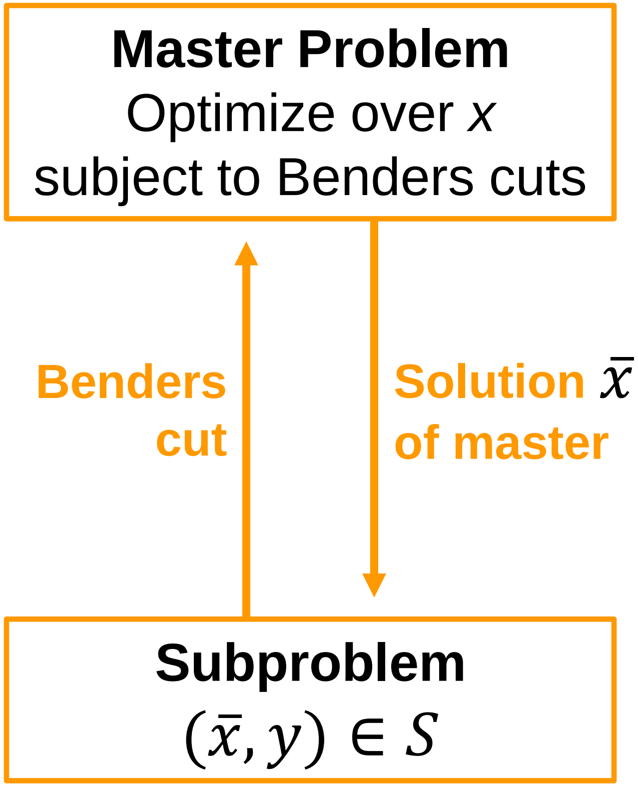
$$\min f(x)$$
$$(x, y) \in S$$

  - Benders cut excludes $\bar{x}$ (and perhaps similar solutions) if it is infeasible in the subproblem.
  - Algorithm terminates when $\bar{x}$ is feasible in the subproblem.



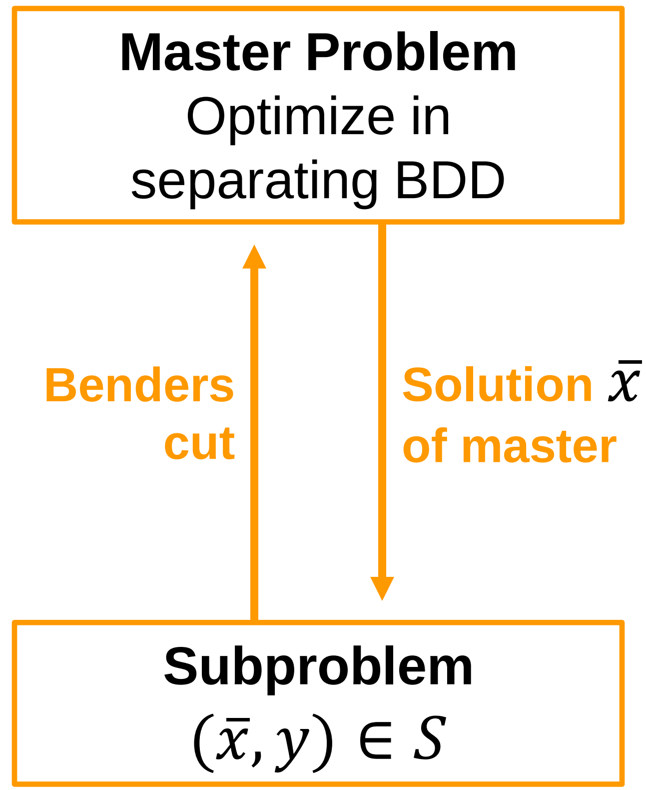
Benders Decomposition

- Logic-based Benders decomposition is a generalization of classical Benders.
  - Master problem is initially a **relaxation** of the original problem over *x* (warm start).
  - Relaxation becomes **tighter** as Benders cuts are added.



Benders Decomposition

- Using BDDs in Benders:
  - We will use a **relaxed BDD** as initial master problem (warm start).
  - Benders cuts exclude **partial assignments**.
  - Add a Benders cut by creating a **separating BDD**.

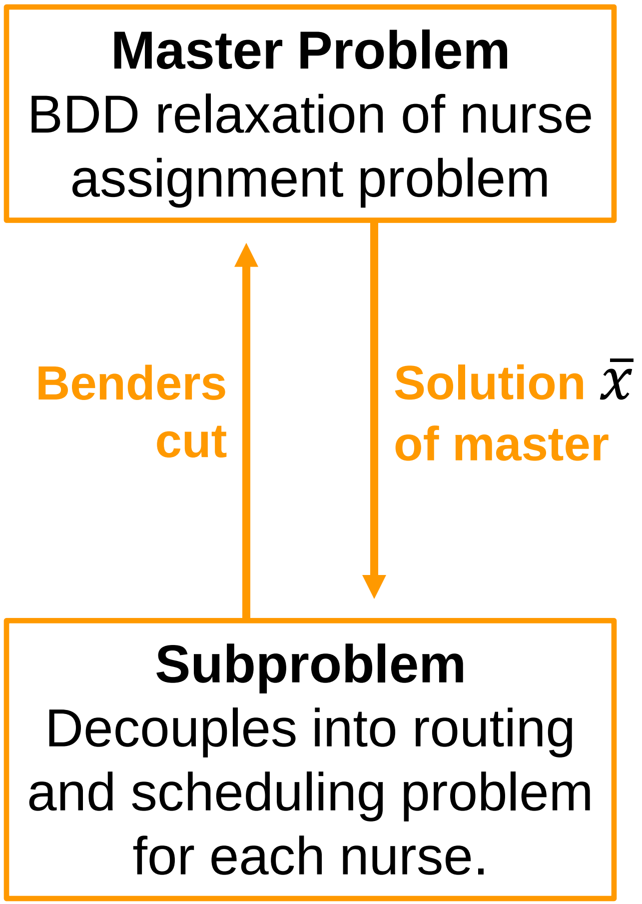


Home Health Care

- Home health care delivery problem.
  - Assign nurses to homebound patients.
  - …subject to constraints on nurse qualifications.
  - Route each nurse through assigned patients, observing time windows.
  - Nurse must take a break if day is long enough.
  - Minimize cost in some sense.

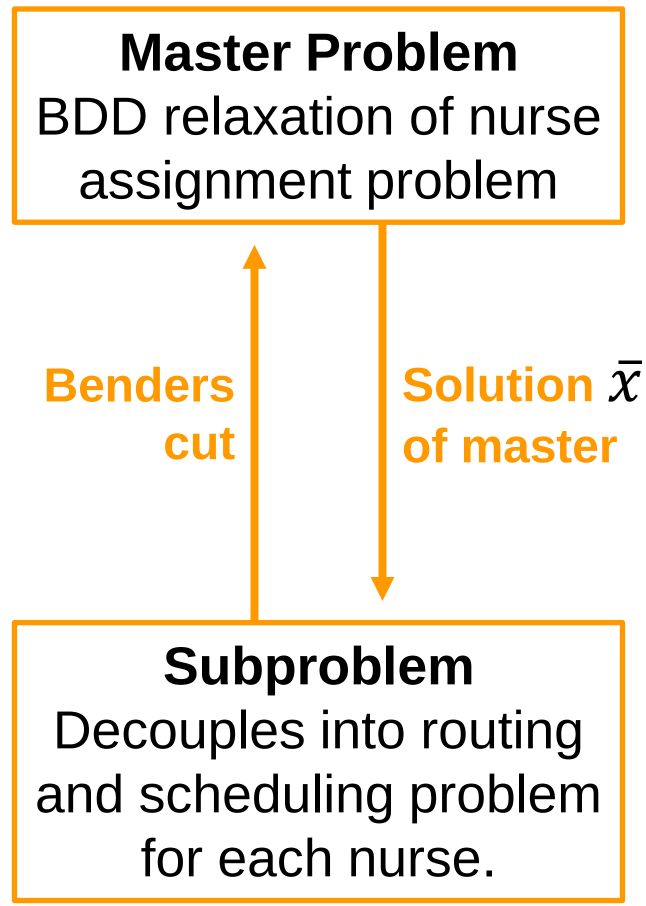
Home Health Care

- Solve with Benders decomposition.
  - Assignment problem in master.
  - Subproblem generates Benders cuts when there is no feasible schedule for one or more nurses.
  - Each cut excludes a partial assignment of nurses to patients.



Home Health Care

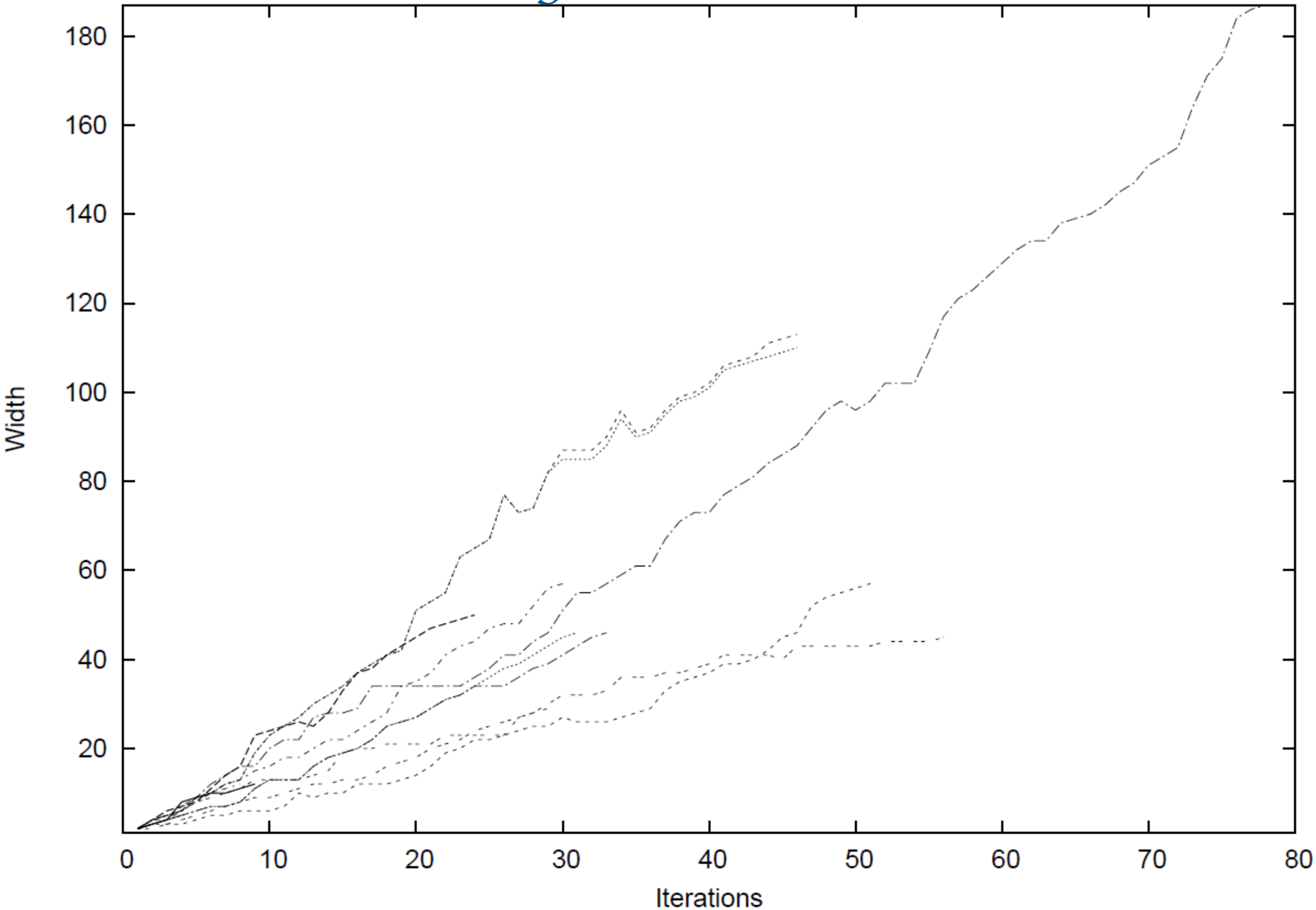
- Solve with Benders decomposition.
  - Assignment problem in master.
  - Subproblem generates Benders cuts when there is no feasible schedule for one or more nurses.
  - Each cut excludes a partial assignment of nurses to patients.
- How fast does the separating BDD grow in the master problem?



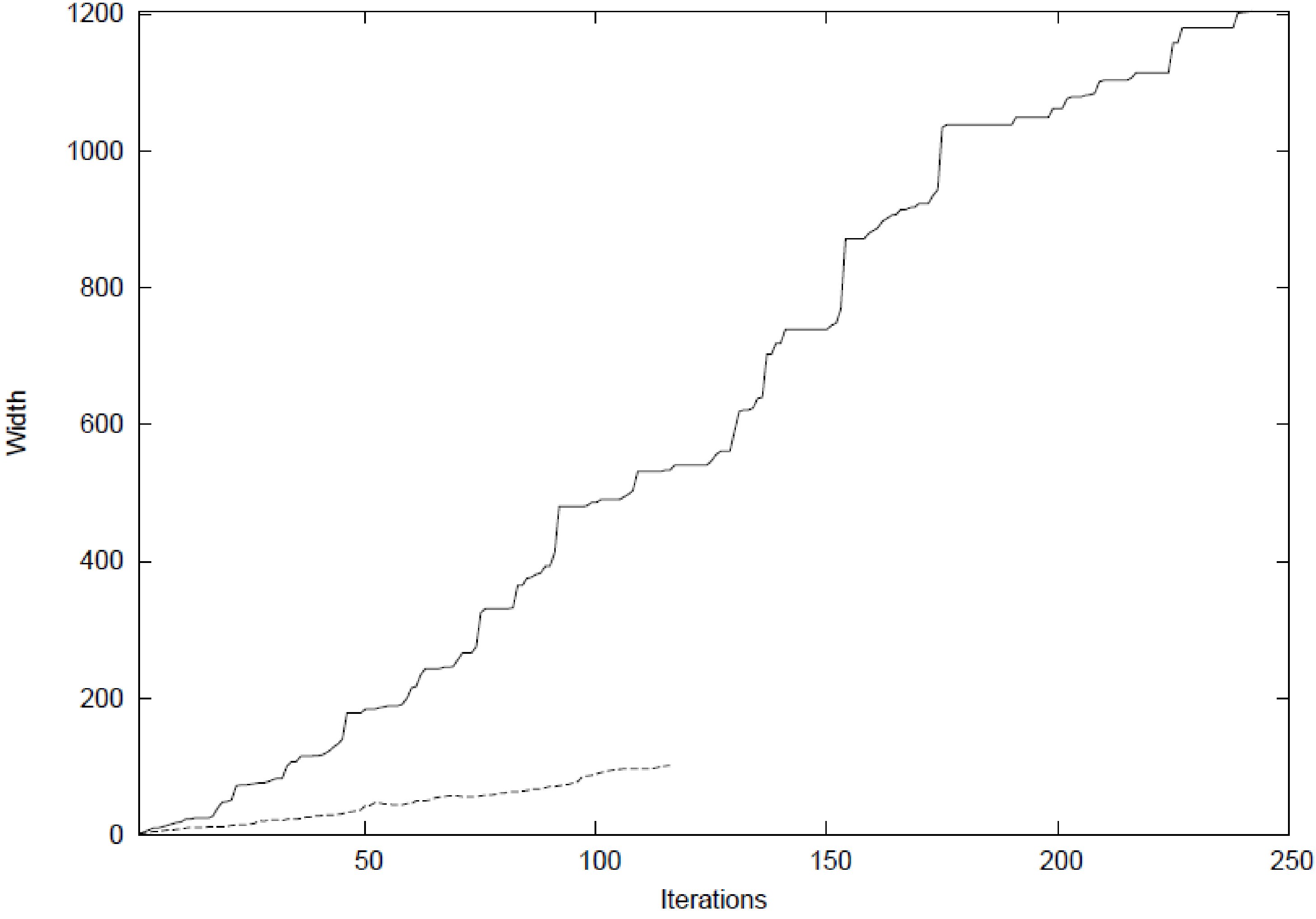
Results for 20 instances

Instance	Iterations	Time (sec)
set1-n30r0	9	7.5
set1-n30r1	24	24.4
set1-n30r2	116	69.7
set1-n30r3	46	40.1
set1-n30r4	31	19.3
set1-n30r5	78	64.3
set1-n30r6	30	29.6
set1-n30r7	29	18.0
set1-n30r8	1	10.2
set1-n30r9	2	11.3
set2-n30r0	9	8.5
set2-n30r1	24	23.8
set2-n30r2	51	31.7
set2-n30r3	46	39.4
set2-n30r4	33	22.1
set3-n30r0	8	3.1
set3-n30r1	56	9.4
set3-n30r2	4	0.7
set3-n30r3	242	80.3
set3-n30r4	820	568.6

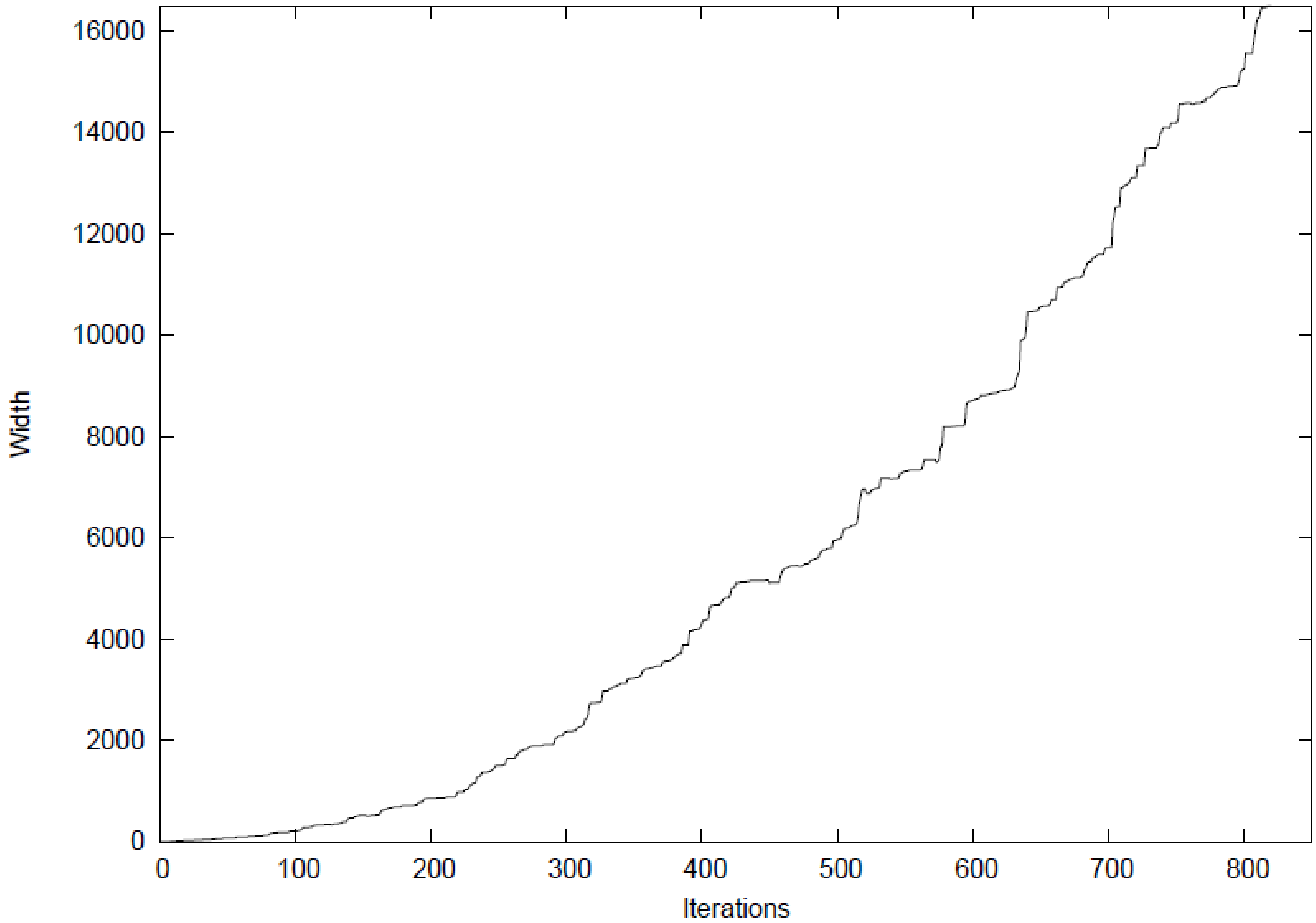
Growth of Separating BDD for All but 3 Instances



Growth of Separating BDD for 2 Harder Instances



Growth of Separating BDD for Hardest Instance



Empirical Growth

- Separating BDD grows more or less **linearly** in all but one instance.
  - Somewhat superlinear in hardest instance.
  - Most BDDs never exceeded width of 100.
  - A width-1000 BDD can be processed in small fraction of a second.
- Hardest instance:
  - Width 16,496.
  - 820 iterations.
  - Final iteration processed in 2.9 seconds, including solution of subproblem.

Exploiting structure

- Include relaxation of subproblem in the master problem
  - Common technique in logic-based Benders.
- How? Add routing and scheduling variables to **bottom** of BDD in the master.
  - The Benders cuts will still contain only assignment variables.
  - By previous corollary, the rest of the BDD will be **unchanged** after separation.

Conclusions

- General separation algorithm is straightforward for BDDs.
  - Reduced separating BDD grows **exponentially** in worst case.
    - Classical relaxation grows **linearly**.
  - Empirically, BDD may grow **linearly**
    - Almost always grows linearly in a representative application of logic-based Benders decomposition.

Conclusions

- So separation may have a useful role in BDD-based optimization.
  - But we must keep a close eye on **growth rate**.
  - Exploiting problem structure with **variable ordering** may reduce growth rate.