What Decision Diagrams Can Do for You

John Hooker
Carnegie Mellon University

INFORMS Optimization Society
March 2018

Decision Diagrams

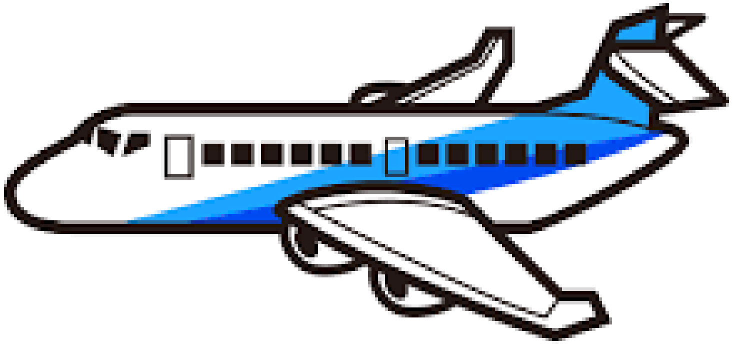
- Used in **computer science** and **AI** for decades
  - Logic circuit design
  - Product configuration
- A **new perspective** on optimization
  - An alternative **data structure**
  - A **new tool** to do many of the things we do in optimization.

Decision Diagrams

- Some advantages:
  - No need for **inequality** formulations.
  - No need for **linear** or **convex** relaxations.
  - Exploits **recursive structure** in the problem, but…
  - Solves **dynamic programming** models **without state space enumeration**.
  - Effective **parallel** computation.
  - Ideal for **postoptimality** analysis

Decision Diagrams

- This is a high-level **overview**.
  - No need to follow the details.



Some Contributors to This Work


Henrik Reif
Andersen


David
Bergman


André
Ciré


Tarik
Hadžić


Samid
Hoda


Willem
van Hoeve

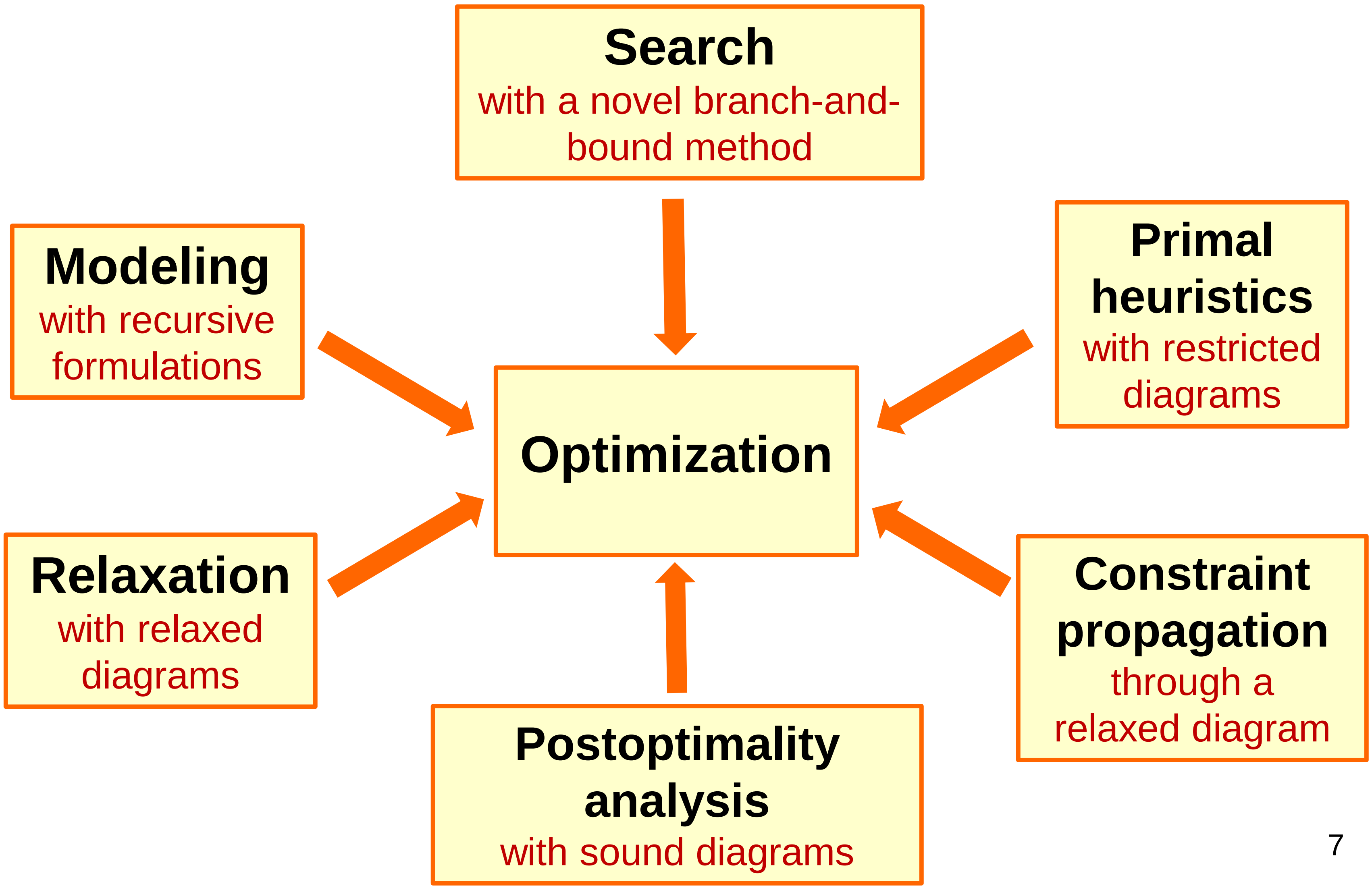

Thiago
Serra


Tallys
Yunes

Outline

- Decision diagram **basics**
- Optimization with **exact** decision diagrams
- Providing the basic **elements of optimization**
  - **Modeling, relaxation, primal heuristics, constraint propagation, search, postoptimality**
- Research **frontiers**
  - Separation
  - Radical reduction of state space in DP
  - Nonlinear optimization
  - Nonserial recursion
- References

Elements of Optimization



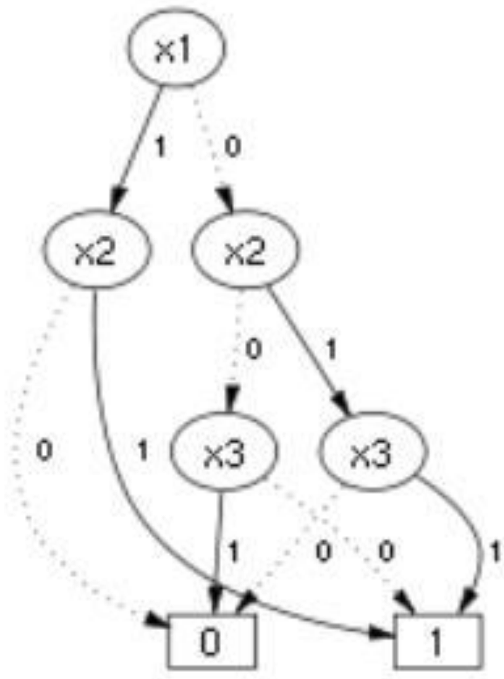
Outline

- Decision diagram **basics**
- Optimization with **exact** decision diagrams
- Providing the basic **elements of optimization**
  - **Modeling, relaxation, primal heuristics, constraint propagation, search, postoptimality**
- Research **frontiers**
  - Separation
  - Radical reduction of state space in DP
  - Nonlinear programming
  - Nonserial recursion
- References

Decision Diagram Basics

- Binary decision diagrams encode Boolean functions

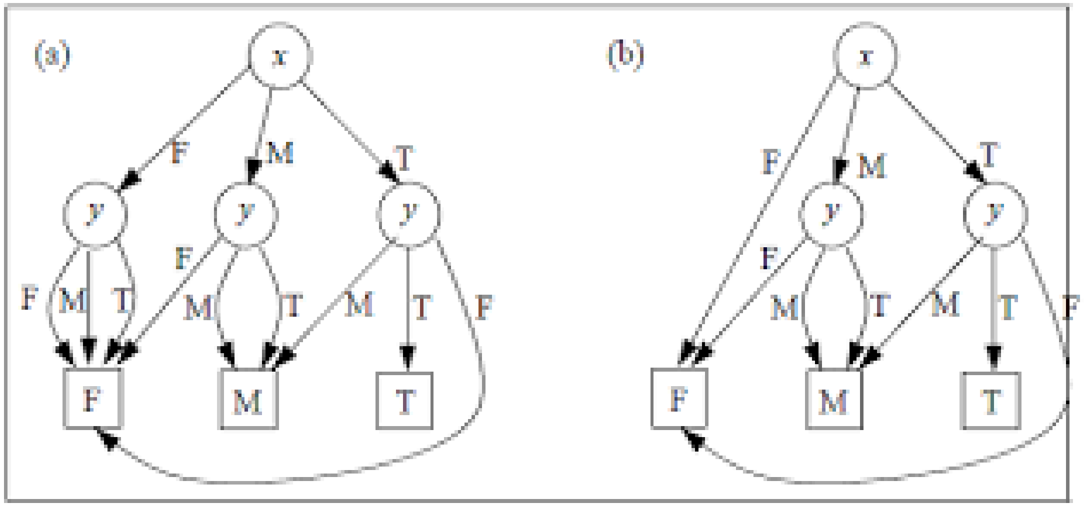
x_1	x_2	x_3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Boole (1847), Shannon (1937), Lee (1959), Akers (1978), Bryant (1986)

Decision Diagram Basics

- Binary decision diagrams encode Boolean functions
  - Easily generalized to **multivalued** decision diagrams



Reduced Decision Diagrams

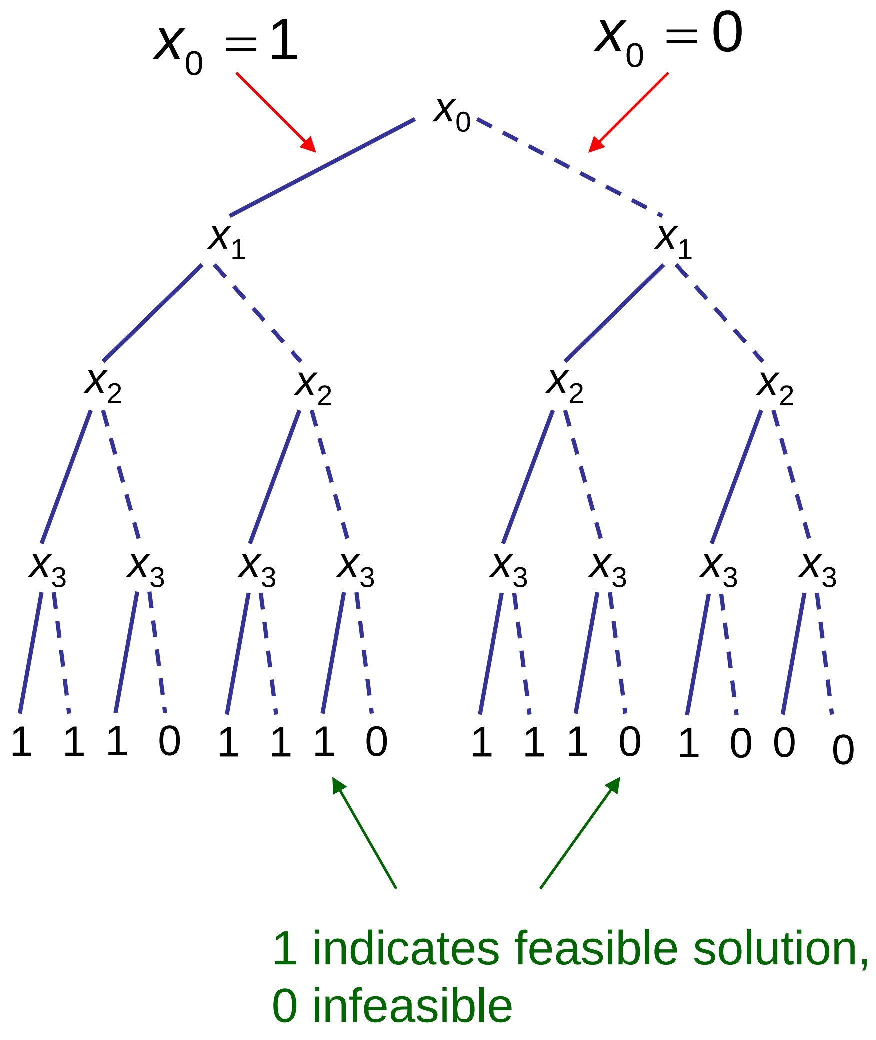
- There is a **unique reduced** DD representing any given function.
  - Once the variable ordering is specified.

Bryant (1986)

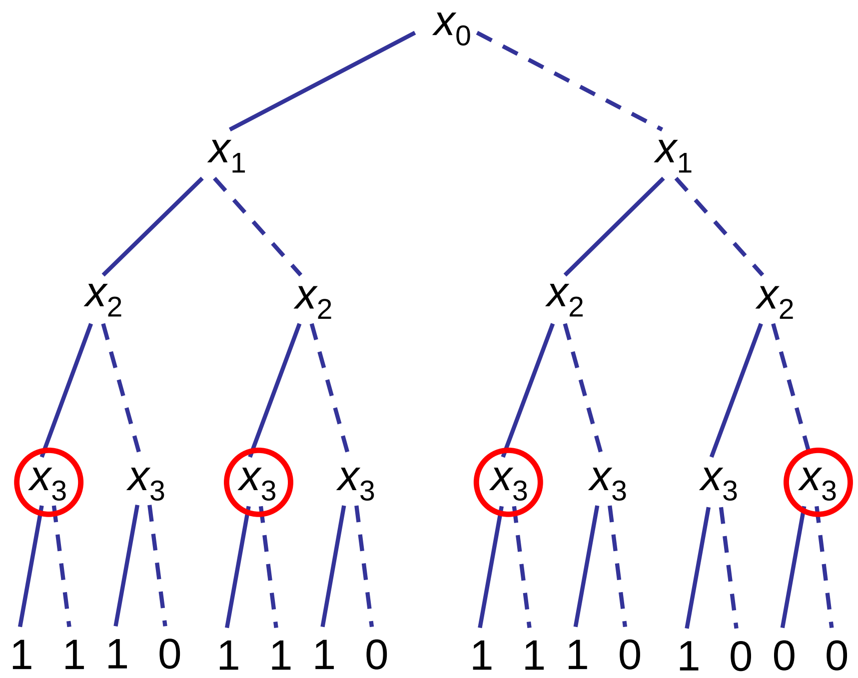
- The reduced DD can be viewed as a branching tree with **redundancy** removed.
  - Superimpose isomorphic subtrees.
  - Remove redundant nodes.

Branching tree for 0-1 inequality

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



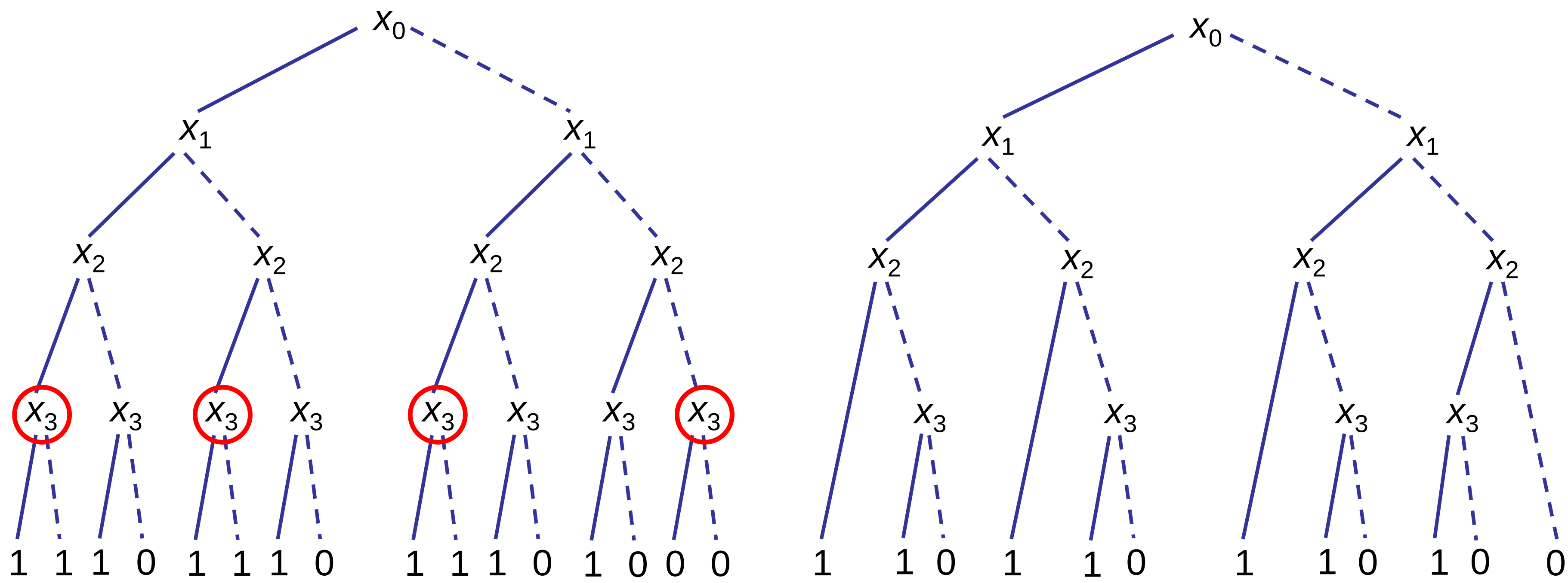
1



Branching tree for 0-1 inequality

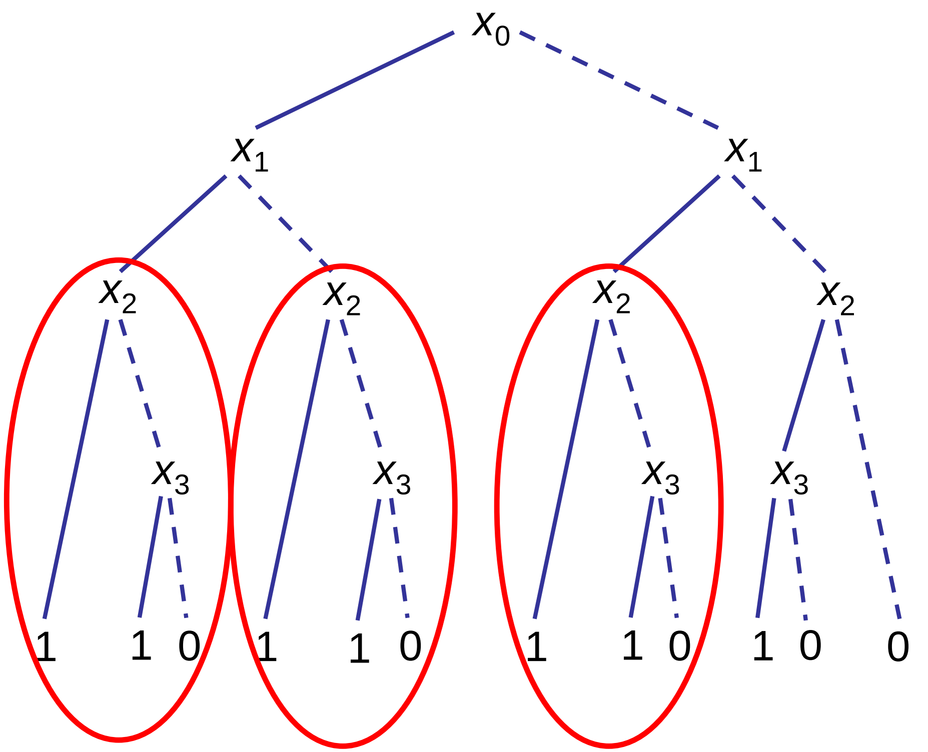
$$2x_0 + 3x_1 + 5x_2 + 5x_3 \geq 7$$

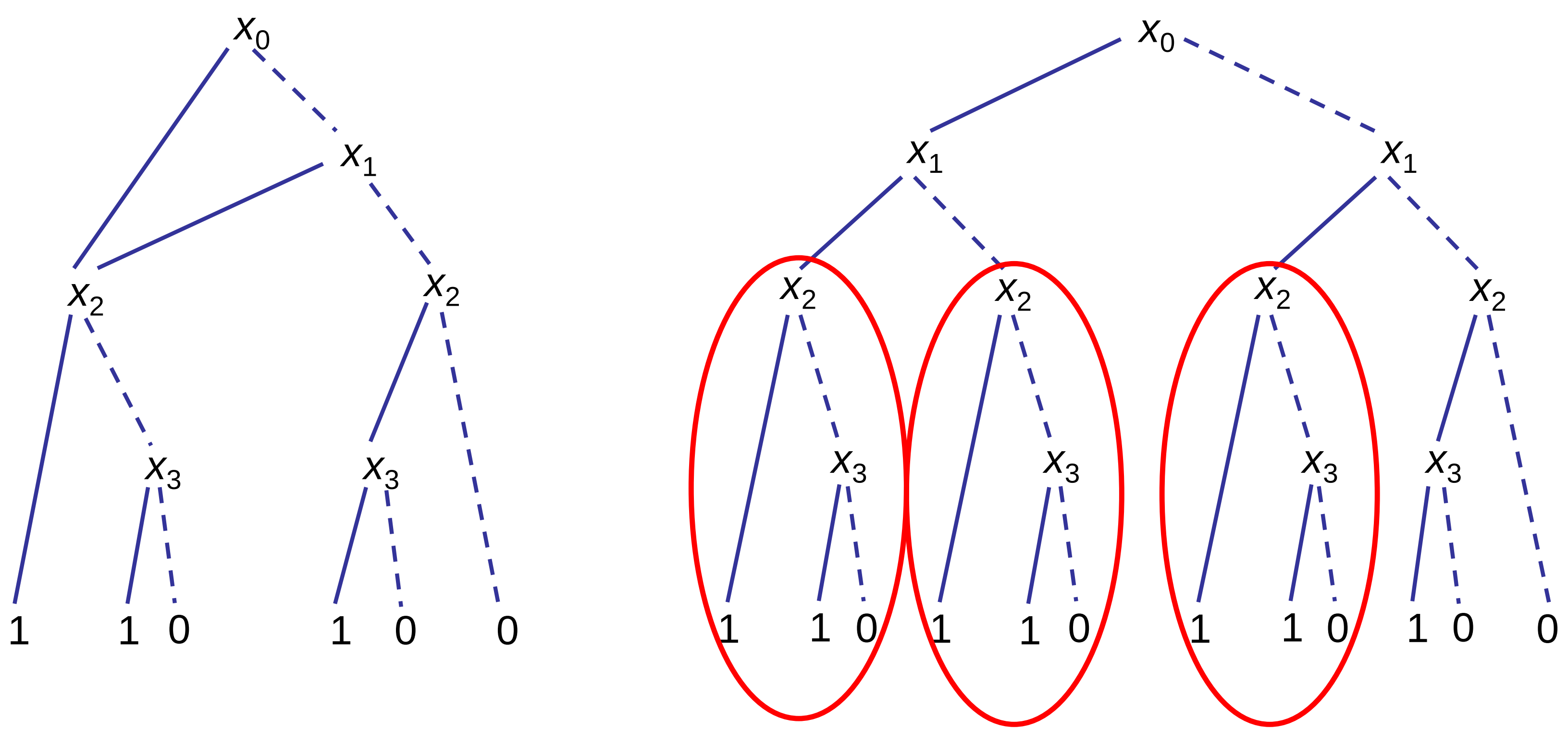
Remove redundant nodes…



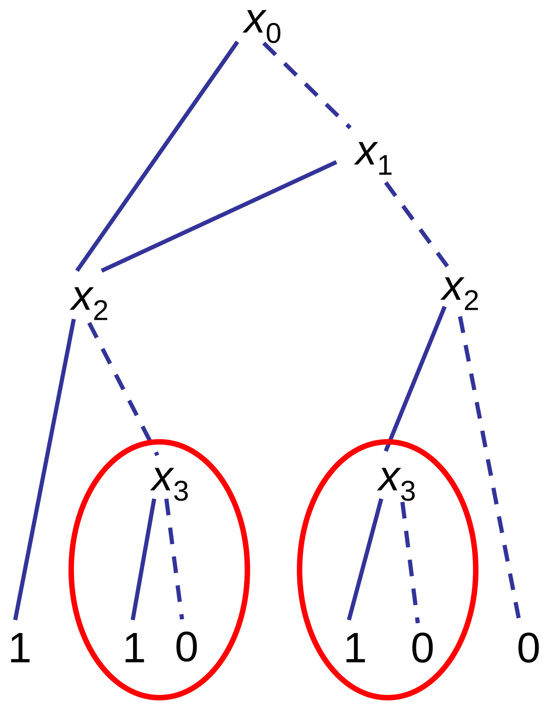
x_0
x_1 x_1
x_2 x_2 x_2 x_2
x_3 x_3 x_3 x_3 x_3 x_3 x_3 x_3
1 1 1 0 1 1 1 0 1 1 1 0 1 0 0 0
x_0
x_1 x_1
x_2 x_2 x_2 x_2
x_3 x_3 x_3 x_3
1 1 0 1 1 0 1 1 0 1 0 0

Superimpose identical subtrees…

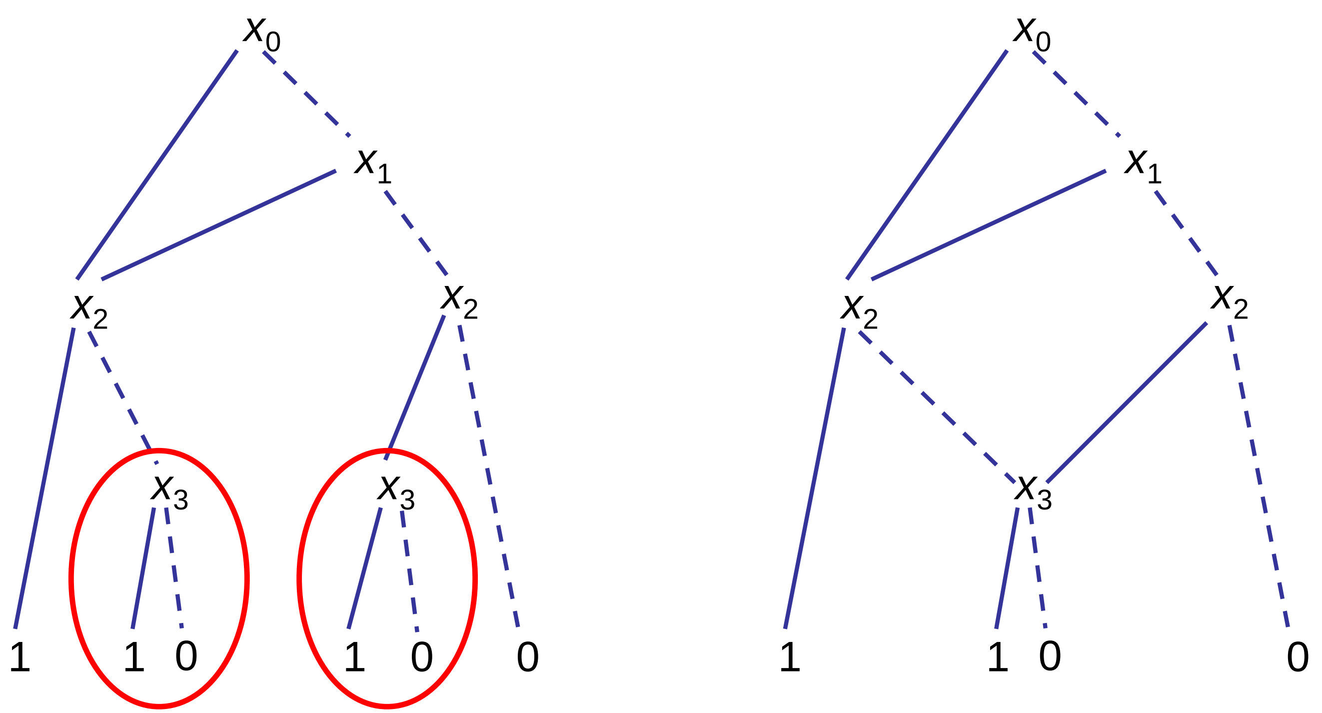




x_0
x_1
x_2
x_2
x_3
x_3
1
1 0
1 0
0
x_0
x_1
x_1
x_2
x_2
x_2
x_2
x_3
x_3
x_3
x_3
1
1 0
1
1 0
1
1 0
1 0
0

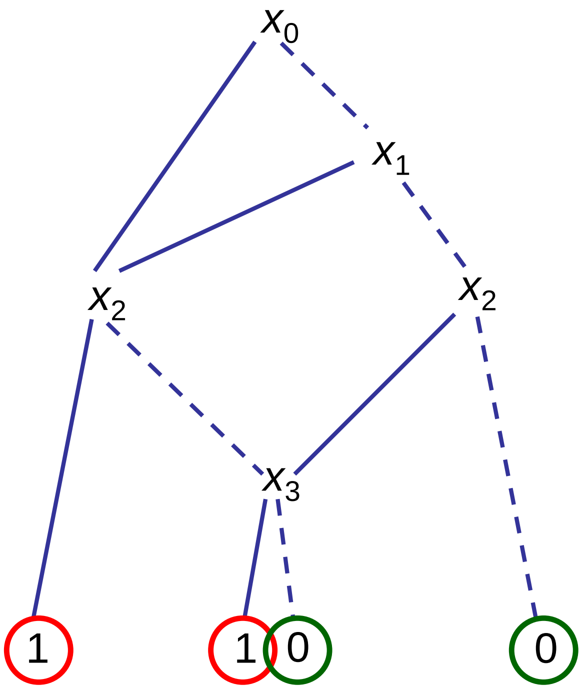


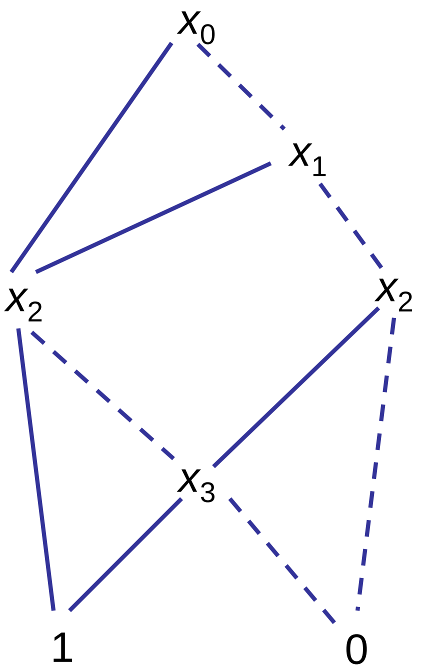
Superimpose identical subtrees…



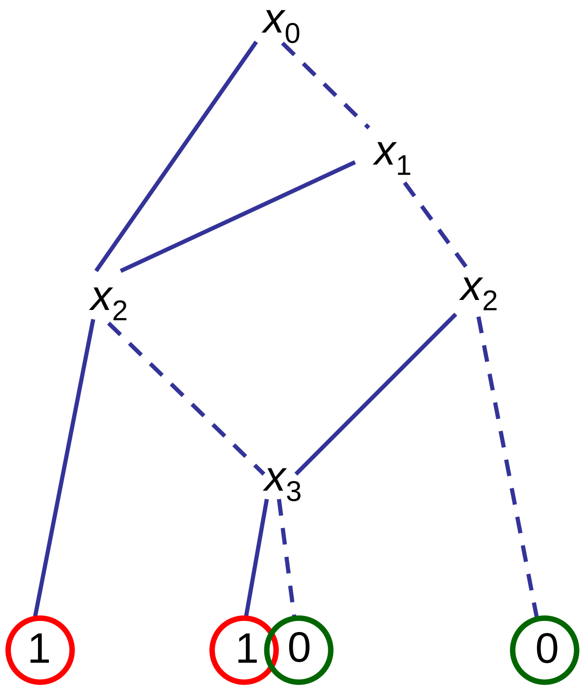
x_0
x_1
x_2
x_2
x_3
x_3
1
1
0
1
0
0
x_0
x_1
x_2
x_2
x_3
1
1
0
0

Superimpose identical leaf nodes…

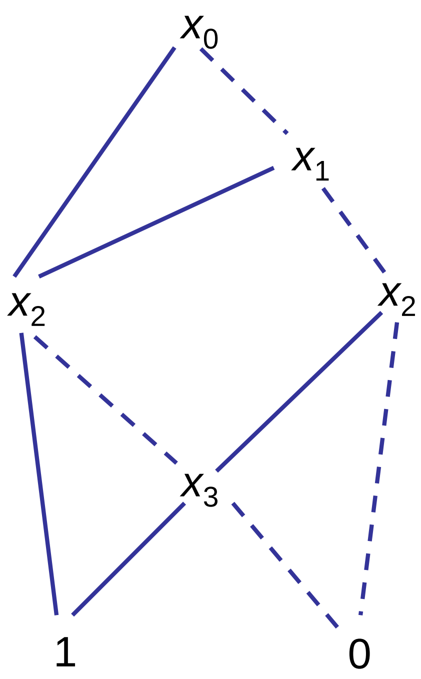


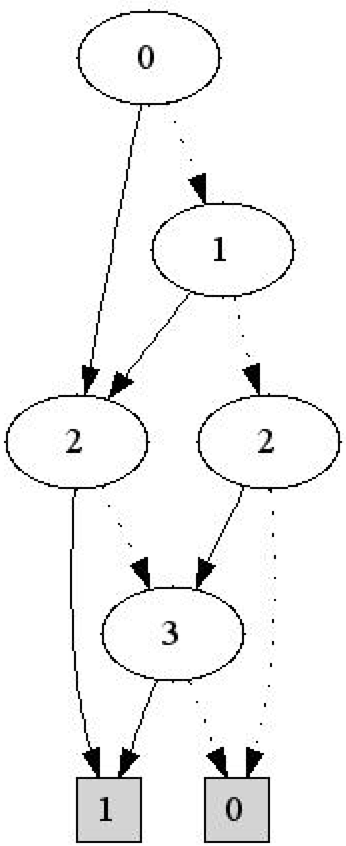


x_0
x_1
x_2
x_2
x_3
1
0



x_0
x_1
x_2
x_2
x_3
1
1
0
0





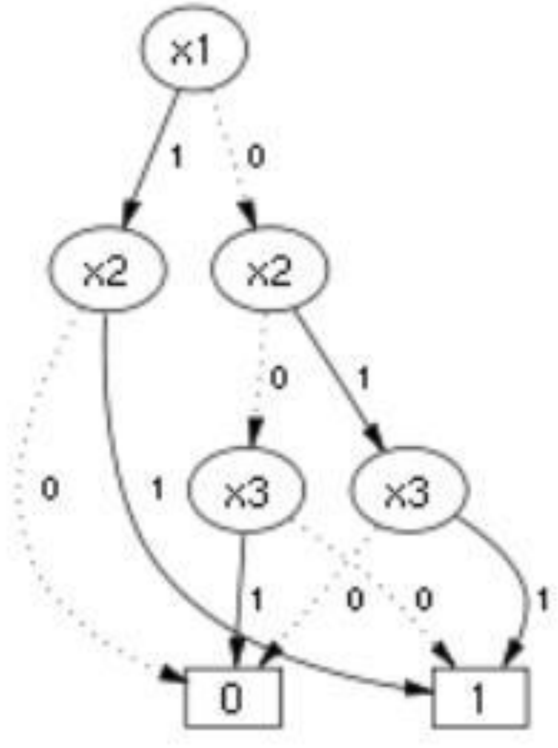
as generated by software

Outline

- Decision diagram **basics**
- Optimization with **exact** decision diagrams
- Providing the basic **elements of optimization**
  - **Modeling, relaxation, primal heuristics, constraint propagation, search, postoptimality**
- Research **frontiers**
  - Radical reduction of state space in DP
  - Nonlinear optimization
  - Nonserial recursion
- References

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.
  - Reduces **optimization** to a **shortest path** problem

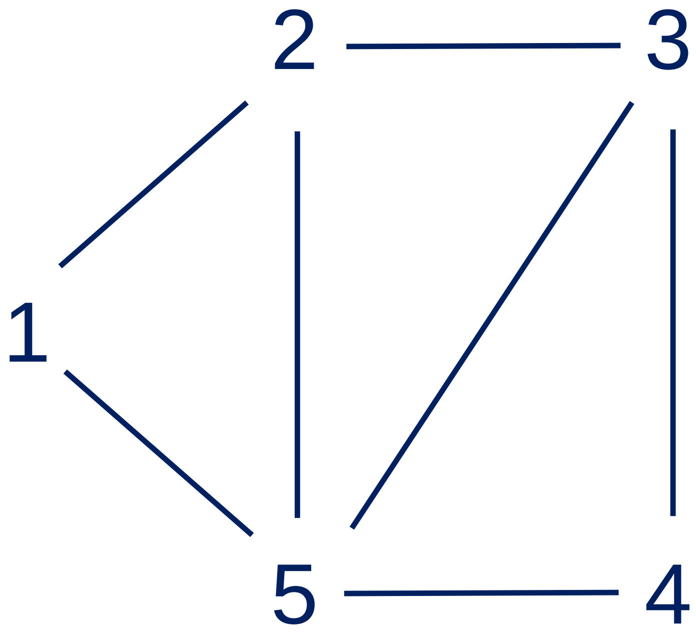


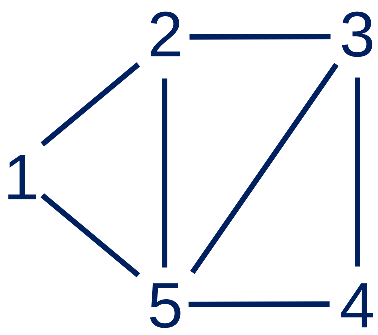
Hadžić and JH (2006, 2007)

Stable Set Problem

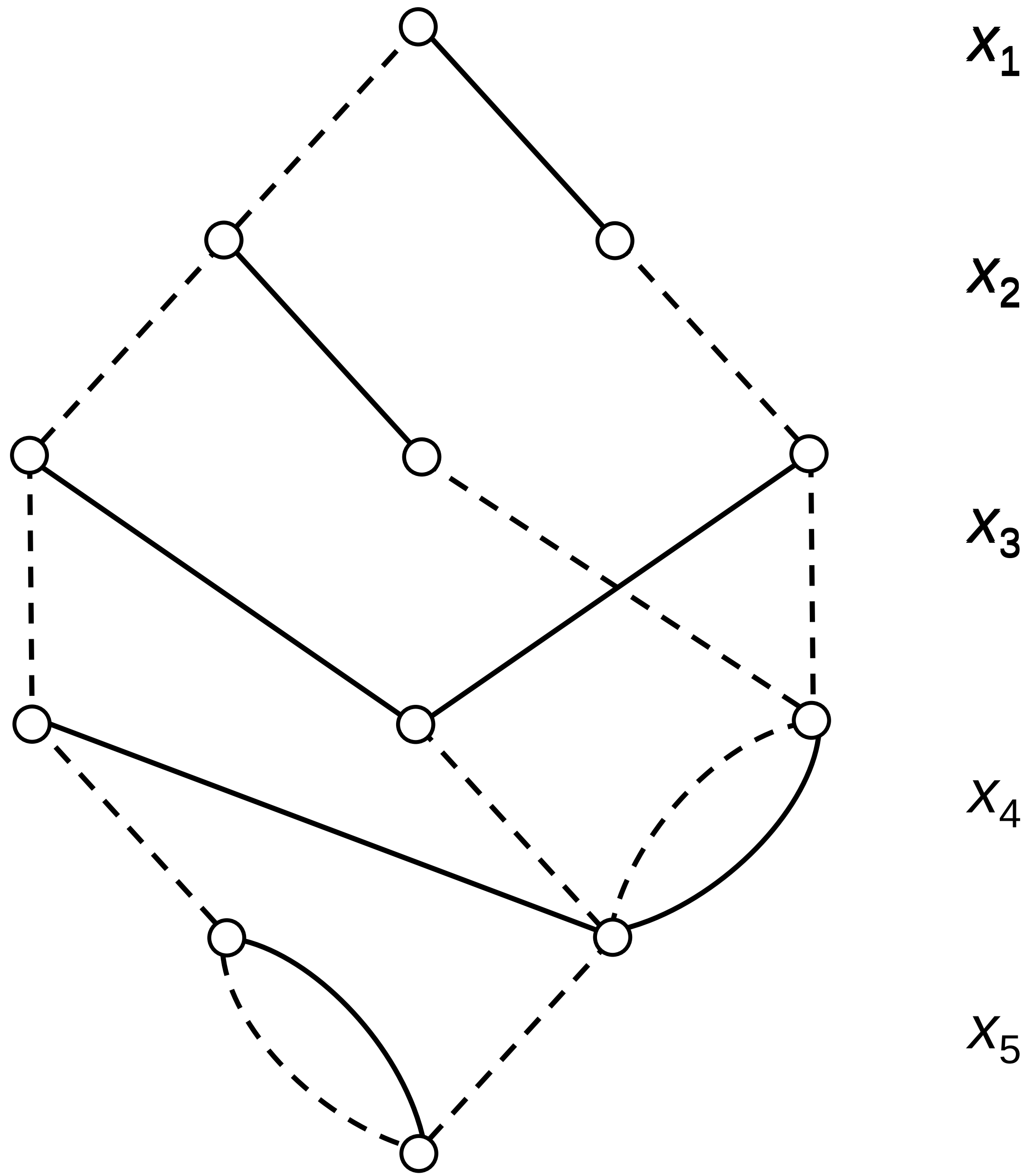
Let each vertex have weight w_i

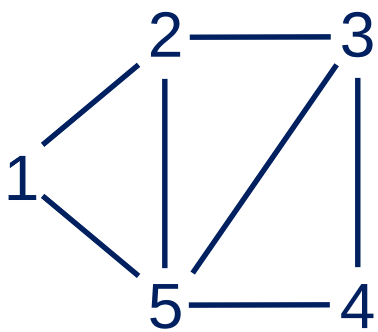
Select nonadjacent vertices to maximize $\sum_i w_i x_i$



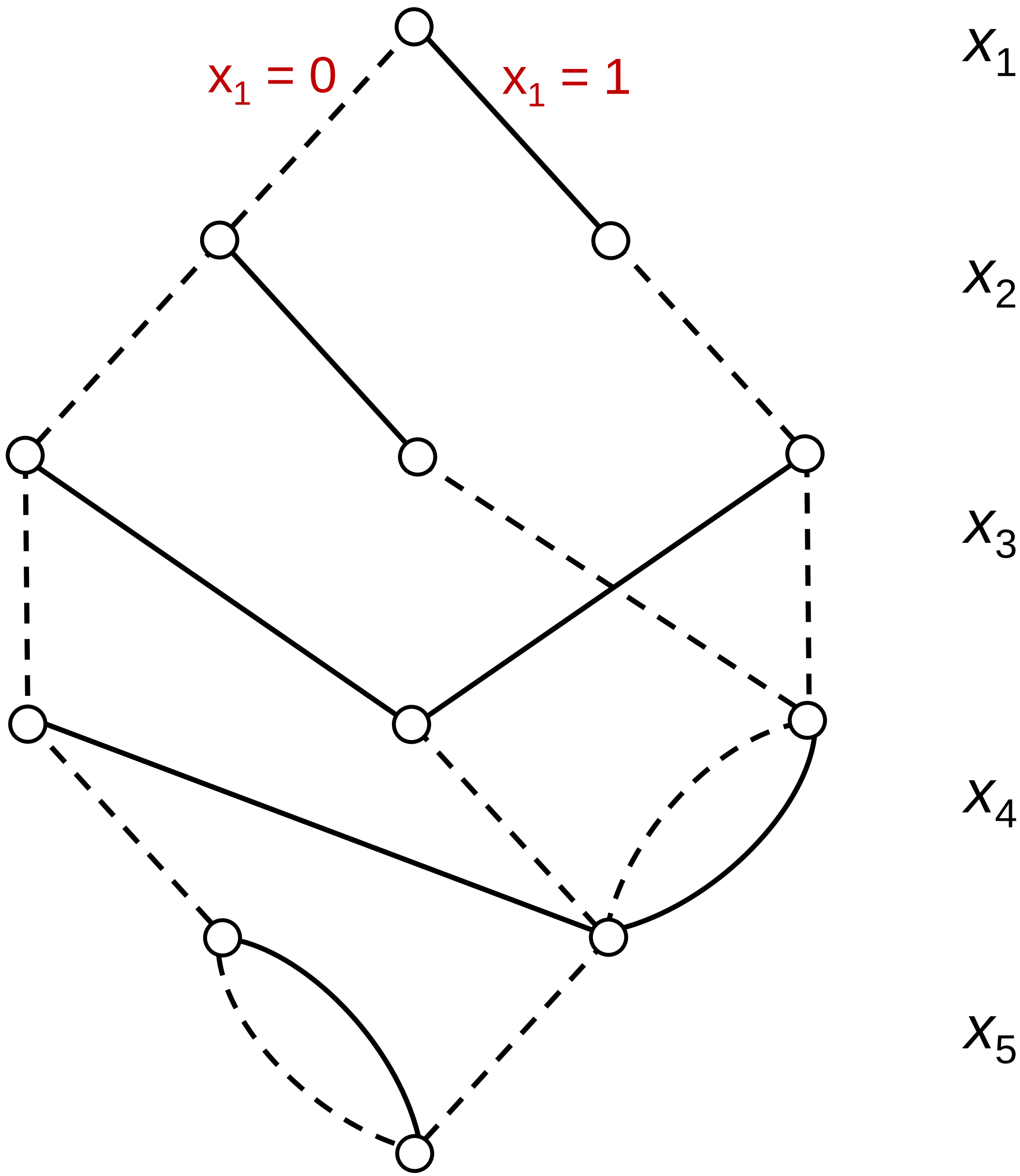


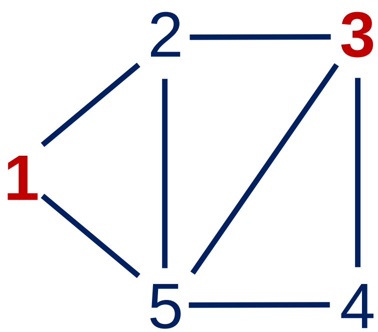
Exact DD for stable set problem



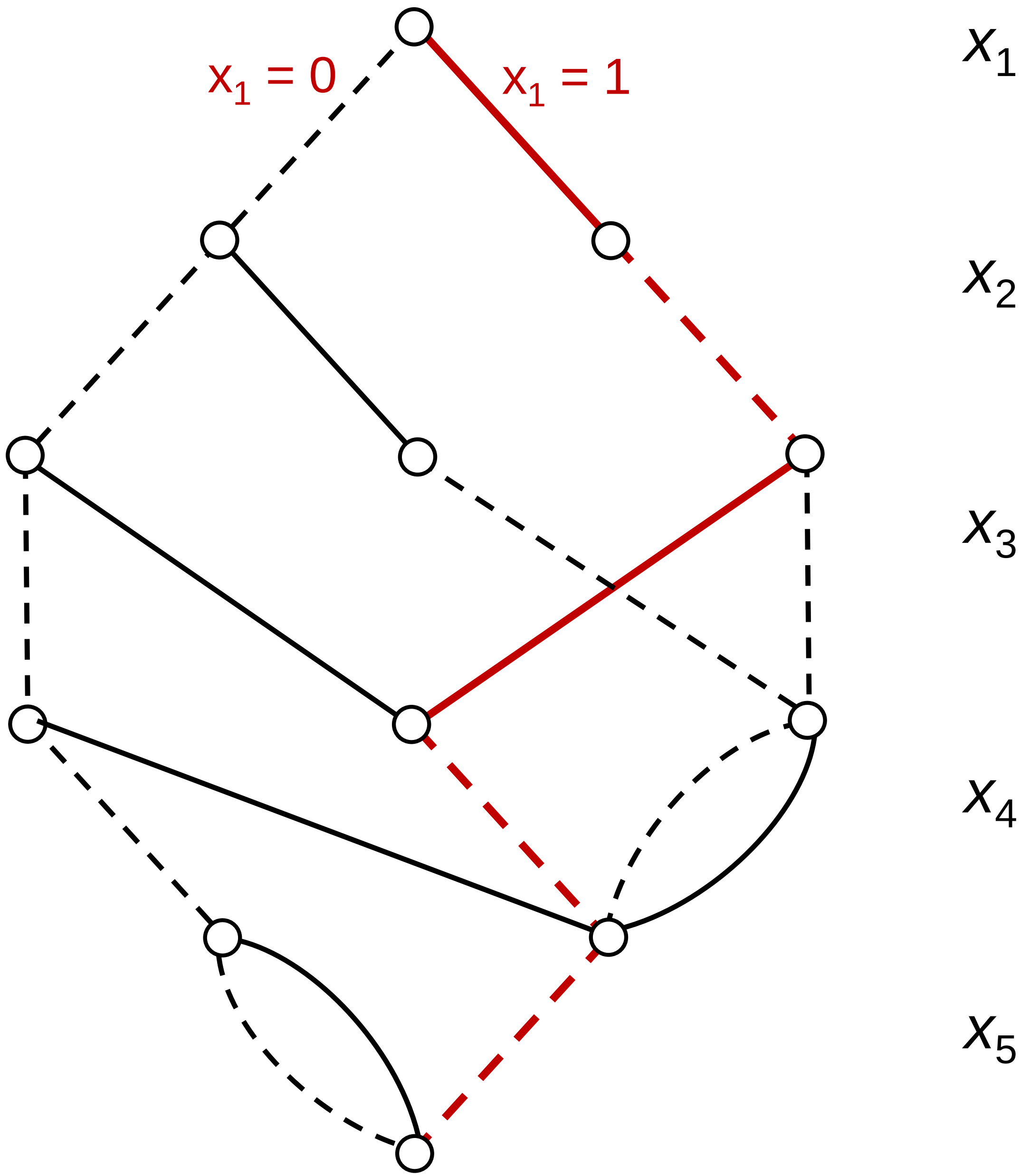


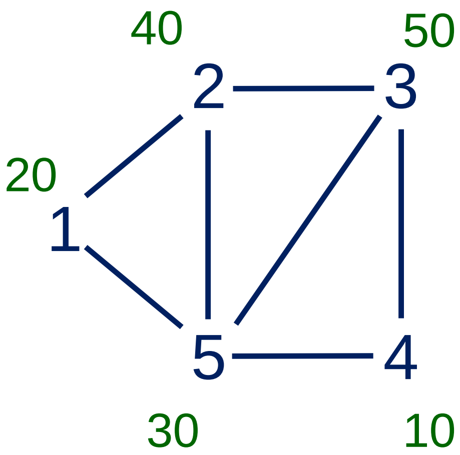
Exact DD for stable set problem



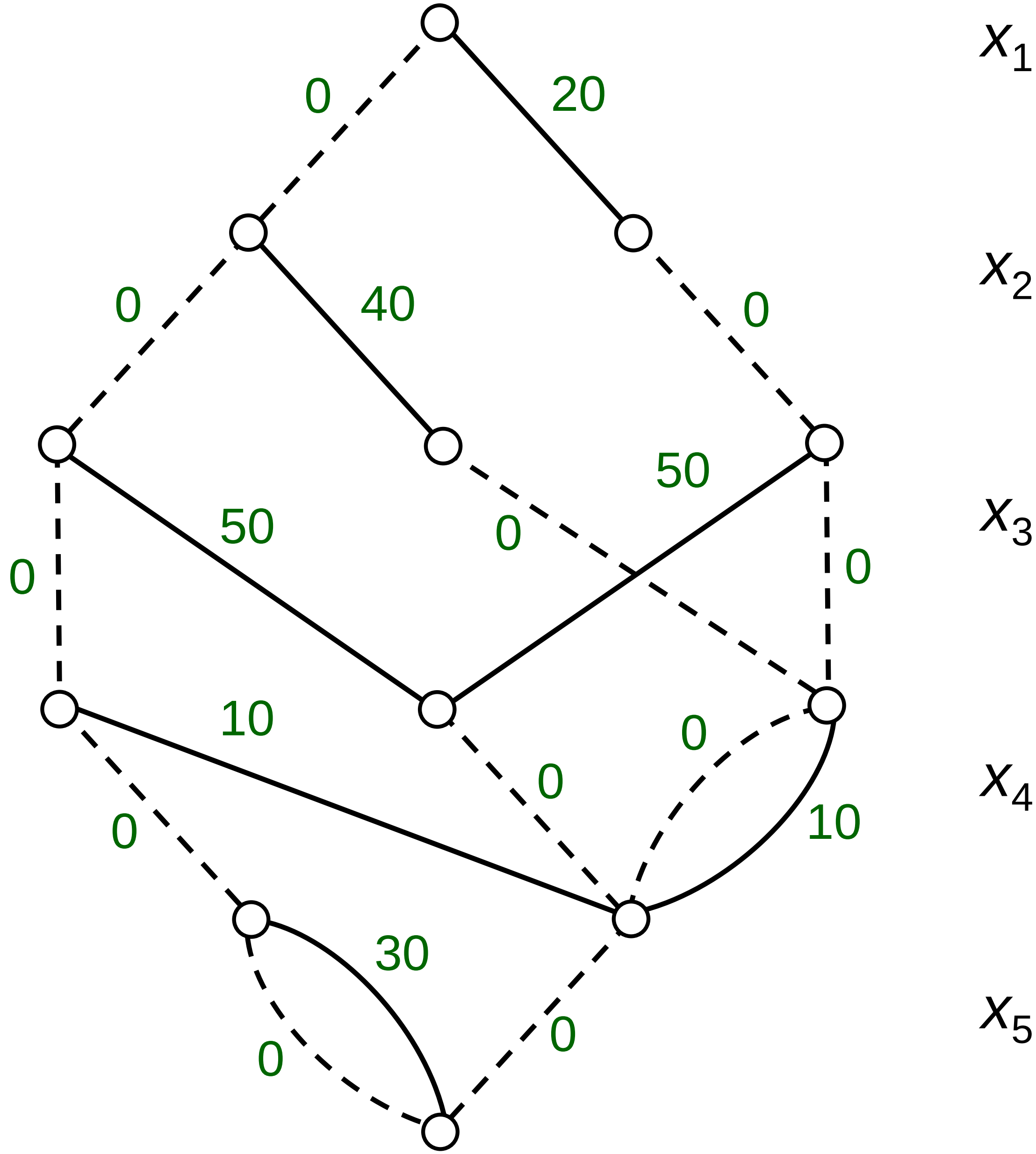


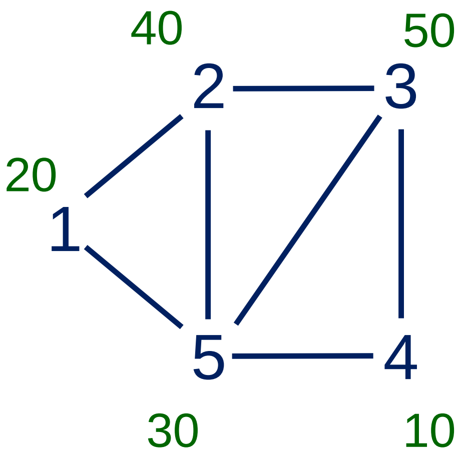
Paths from top to bottom correspond to the 9 feasible solutions



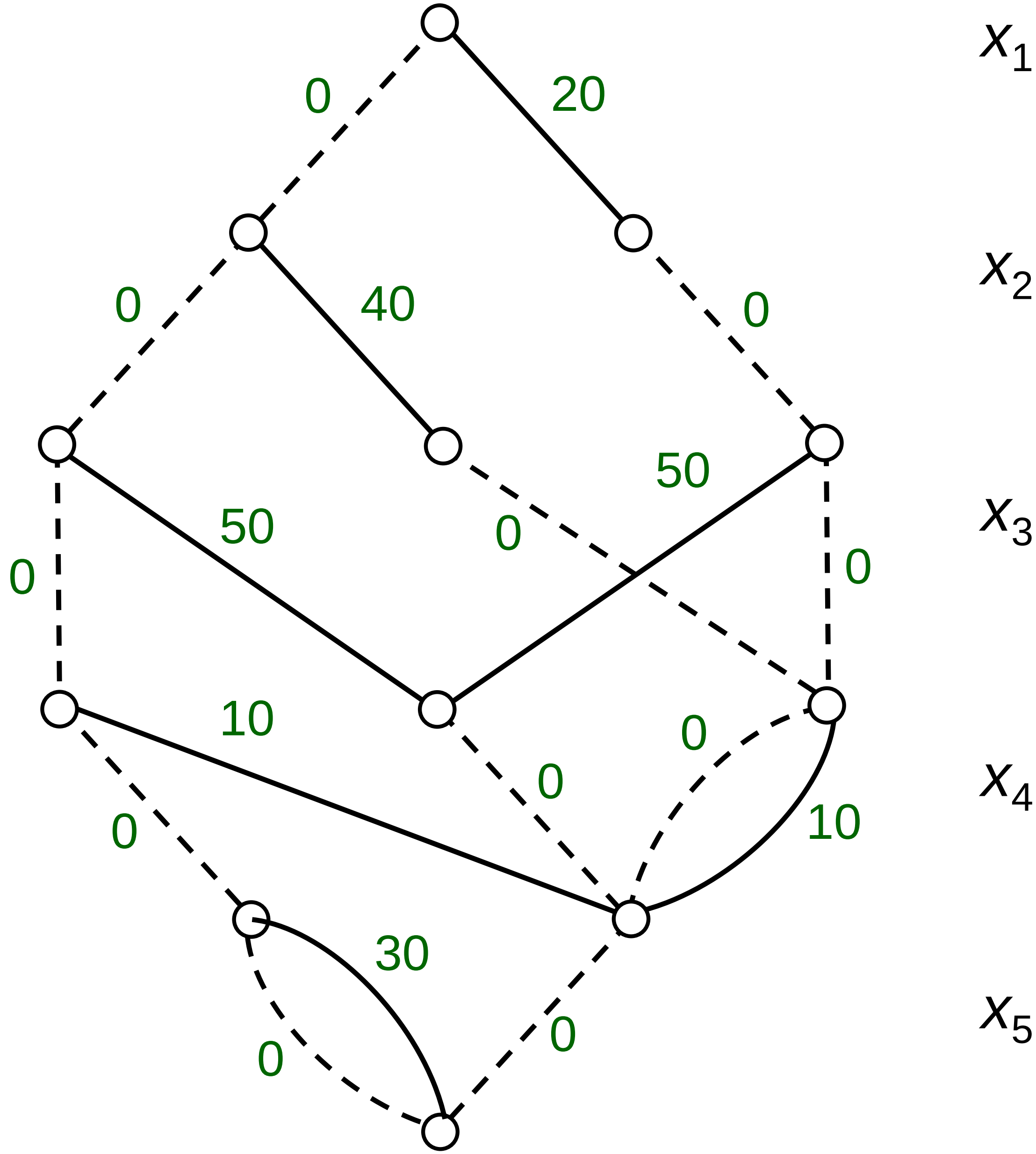


For objective function, associate weights with arcs

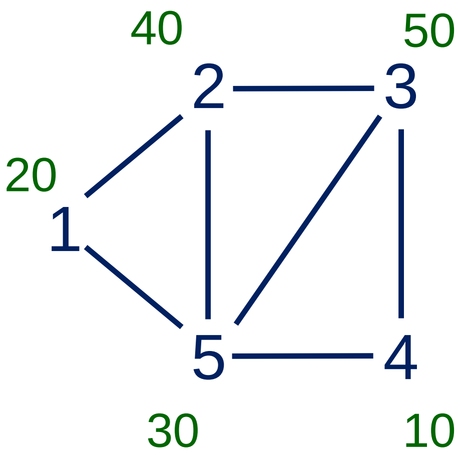




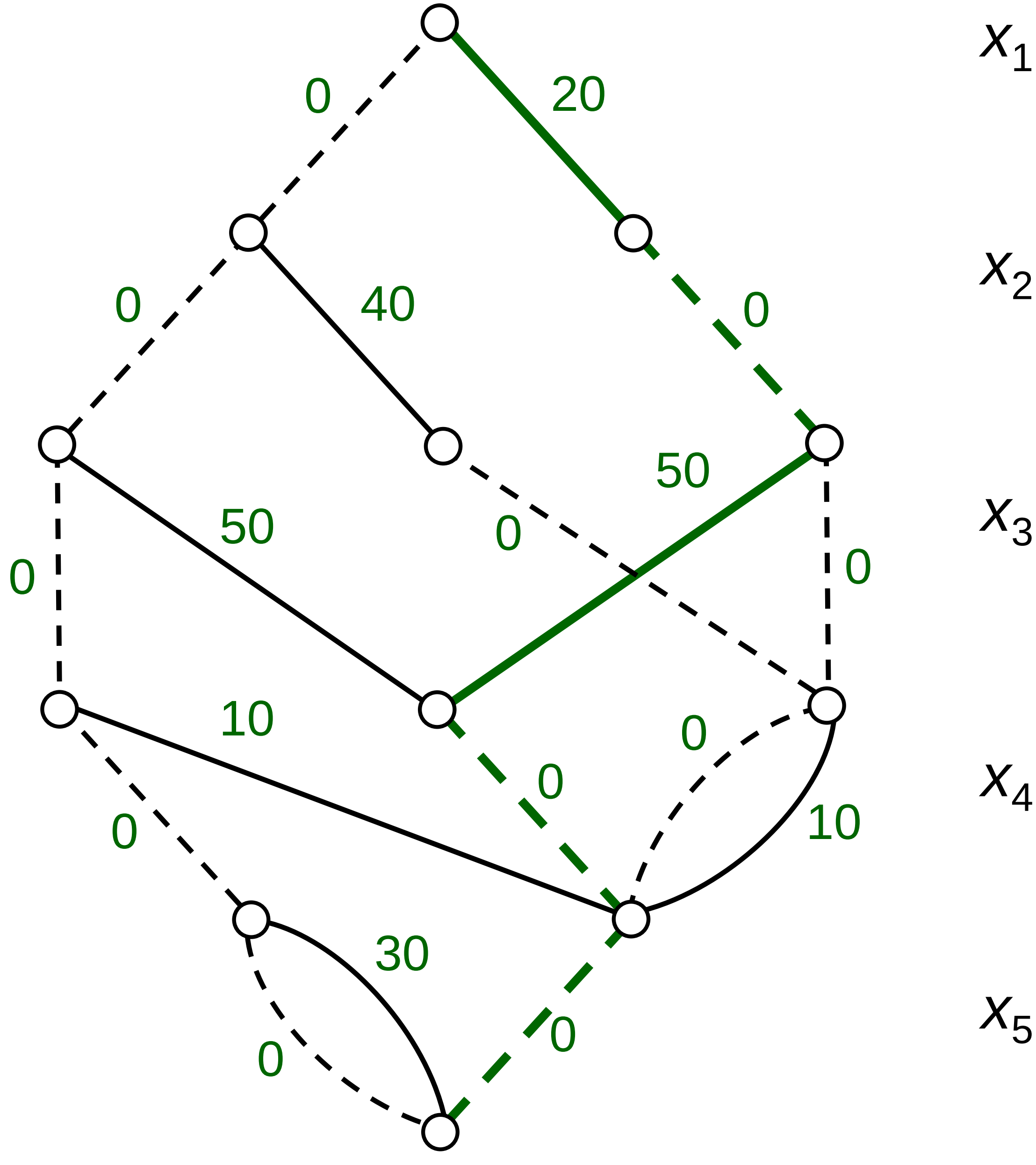
For objective function, associate weights with arcs



Optimal solution is **longest path**



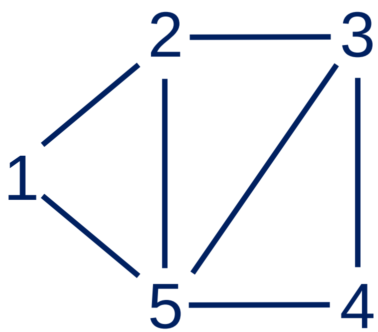
For objective function, associate weights with arcs



Optimal solution is **longest path**

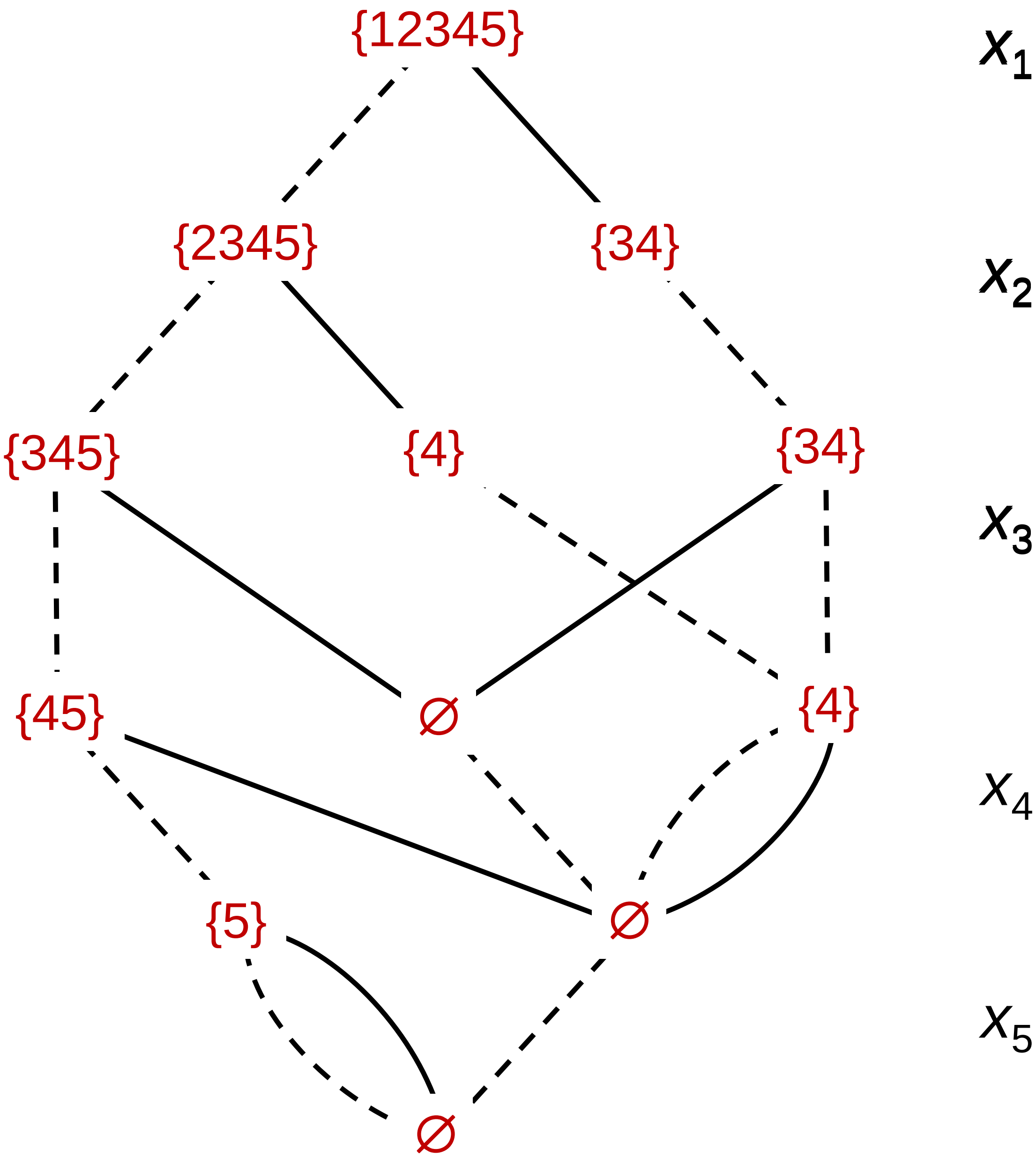
Exact DD Compilation

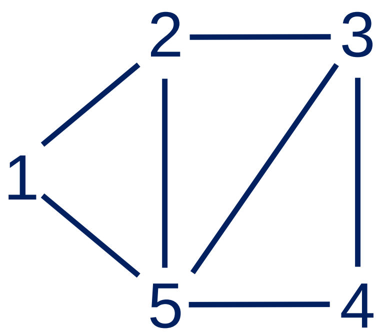
- Build an exact DD by associating a **state** with each node.
  - Merge nodes with **identical states**.



Exact DD for stable set problem

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





{12345}

x_1

x_2

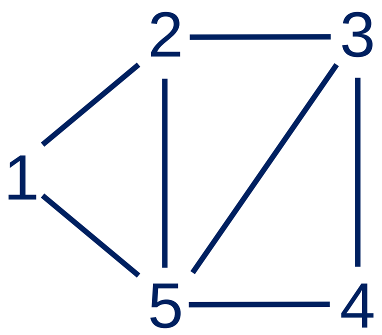
Exact DD for stable set problem

x_3

x_4

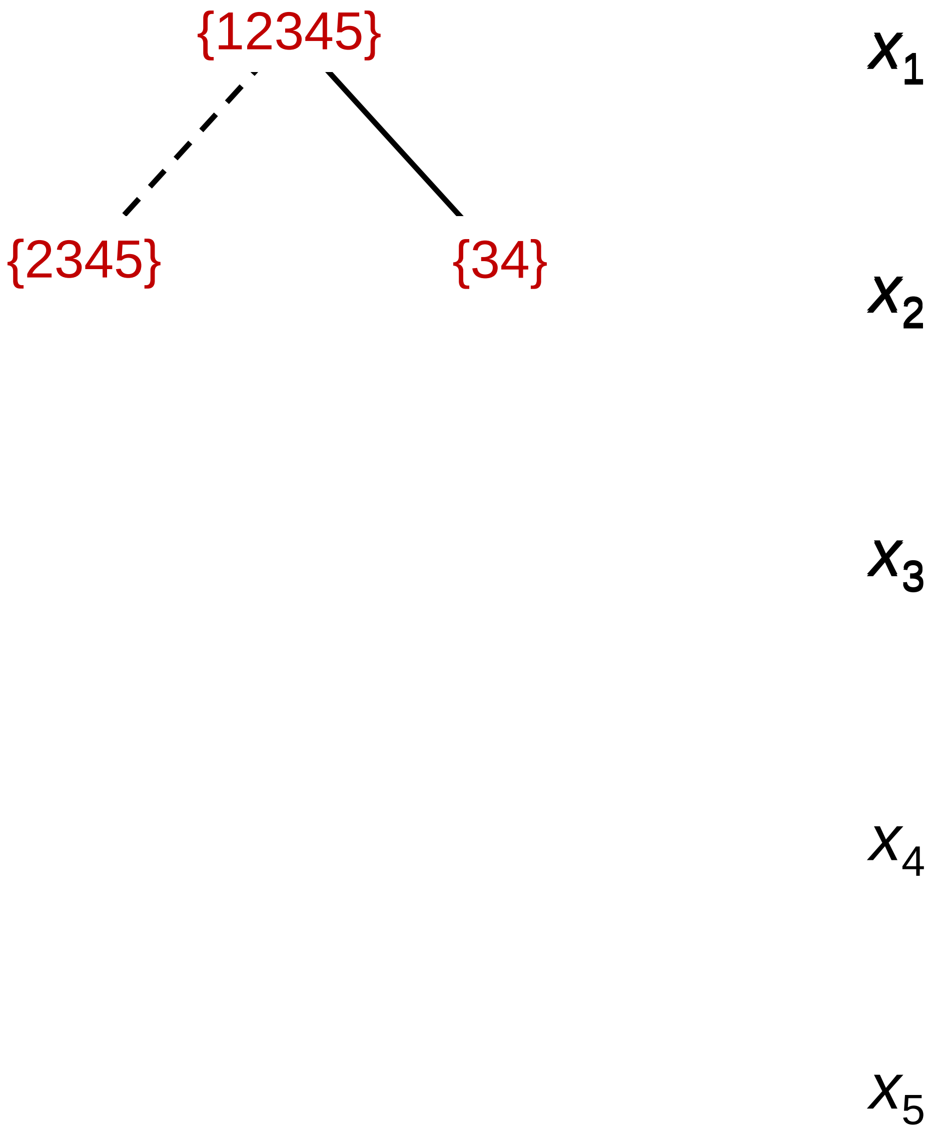
To build DD, associate **state** with each node

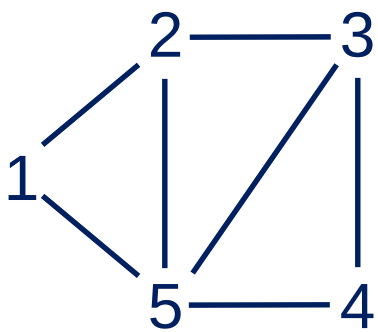
x_5

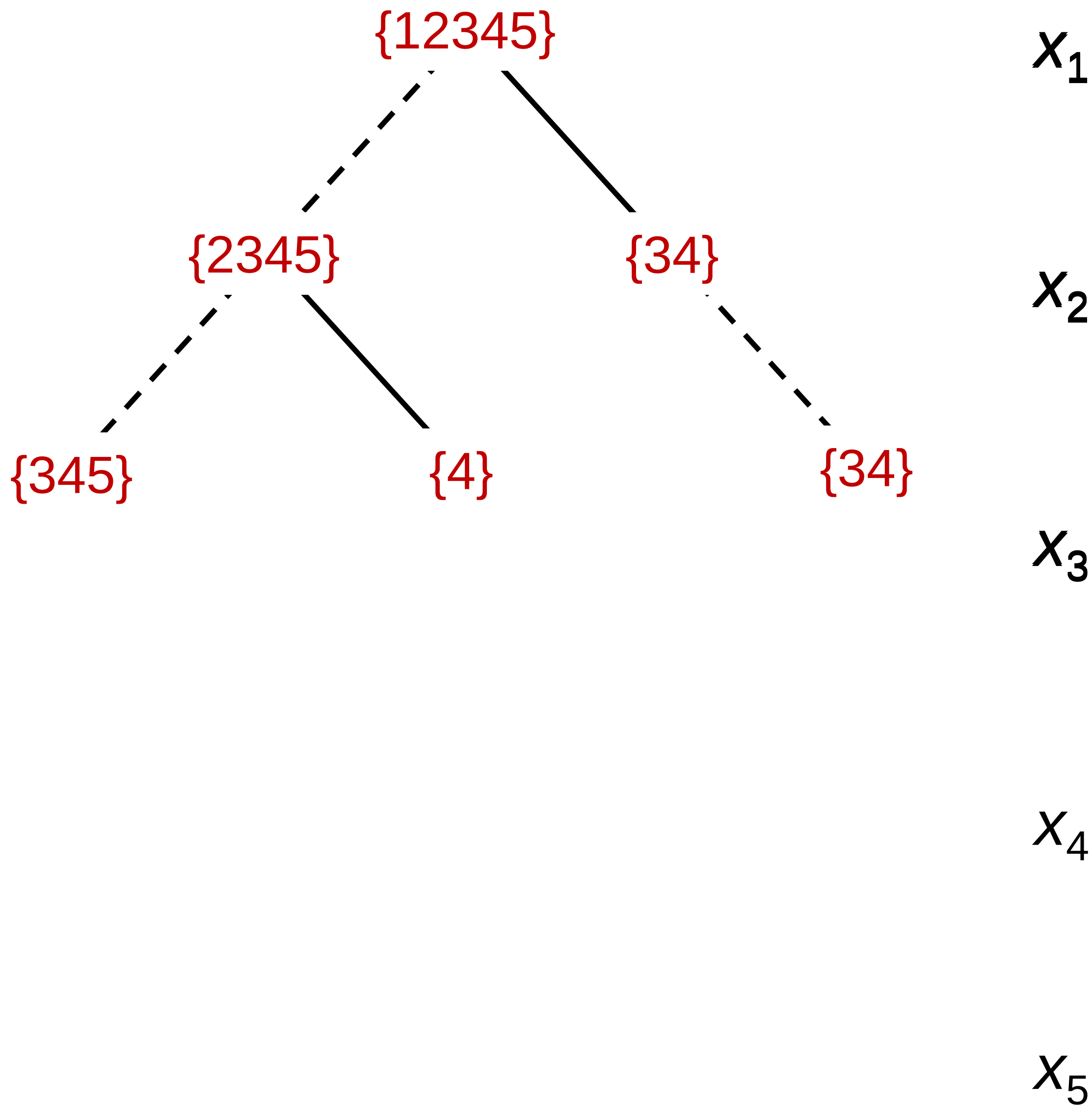


Exact DD for stable set problem

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

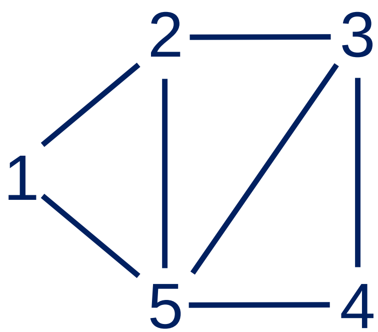


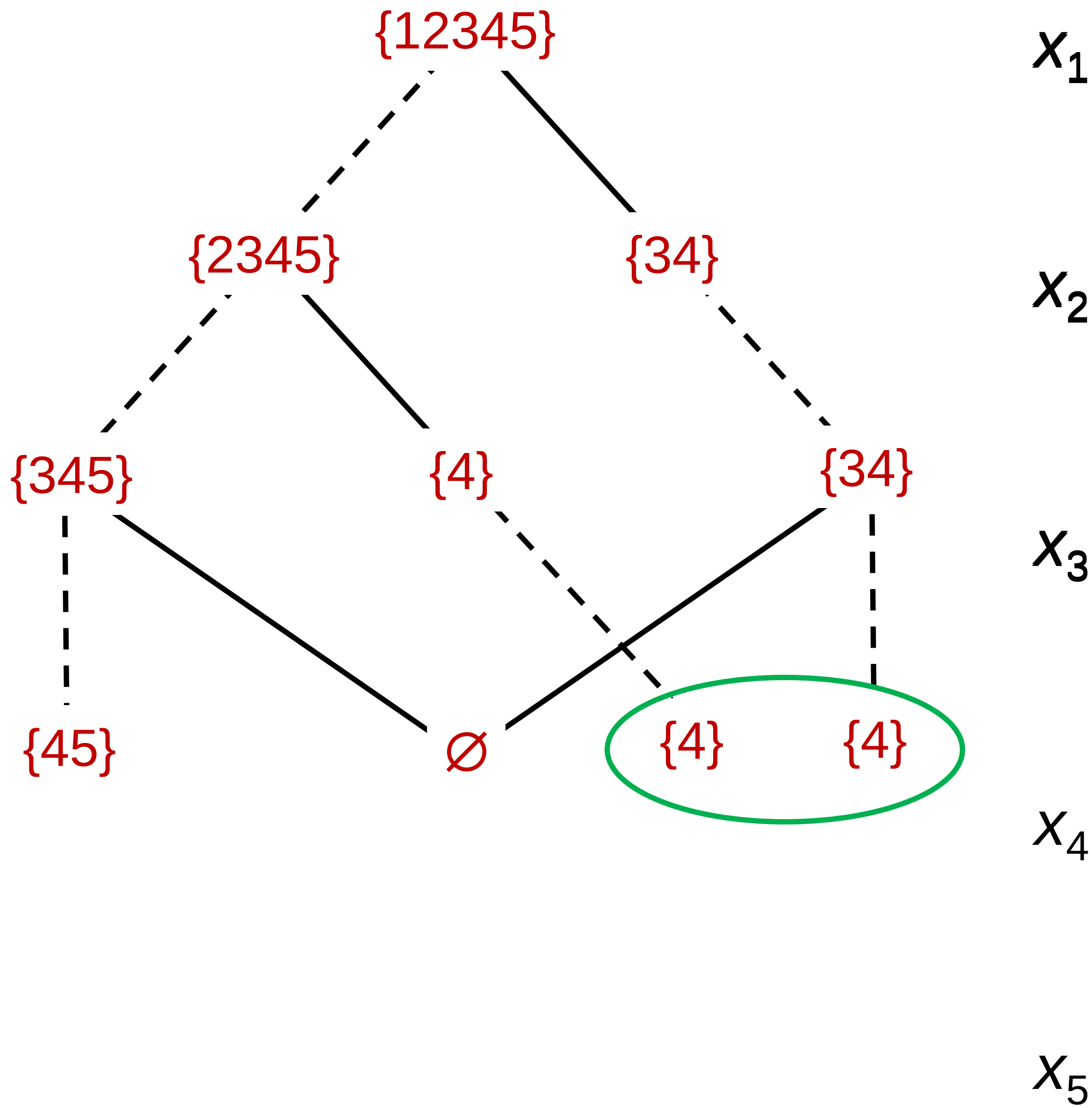




Exact DD for stable set problem

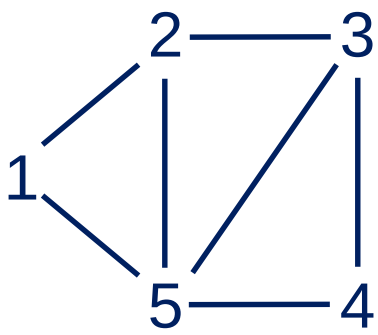
To build DD, associate **state** with each node

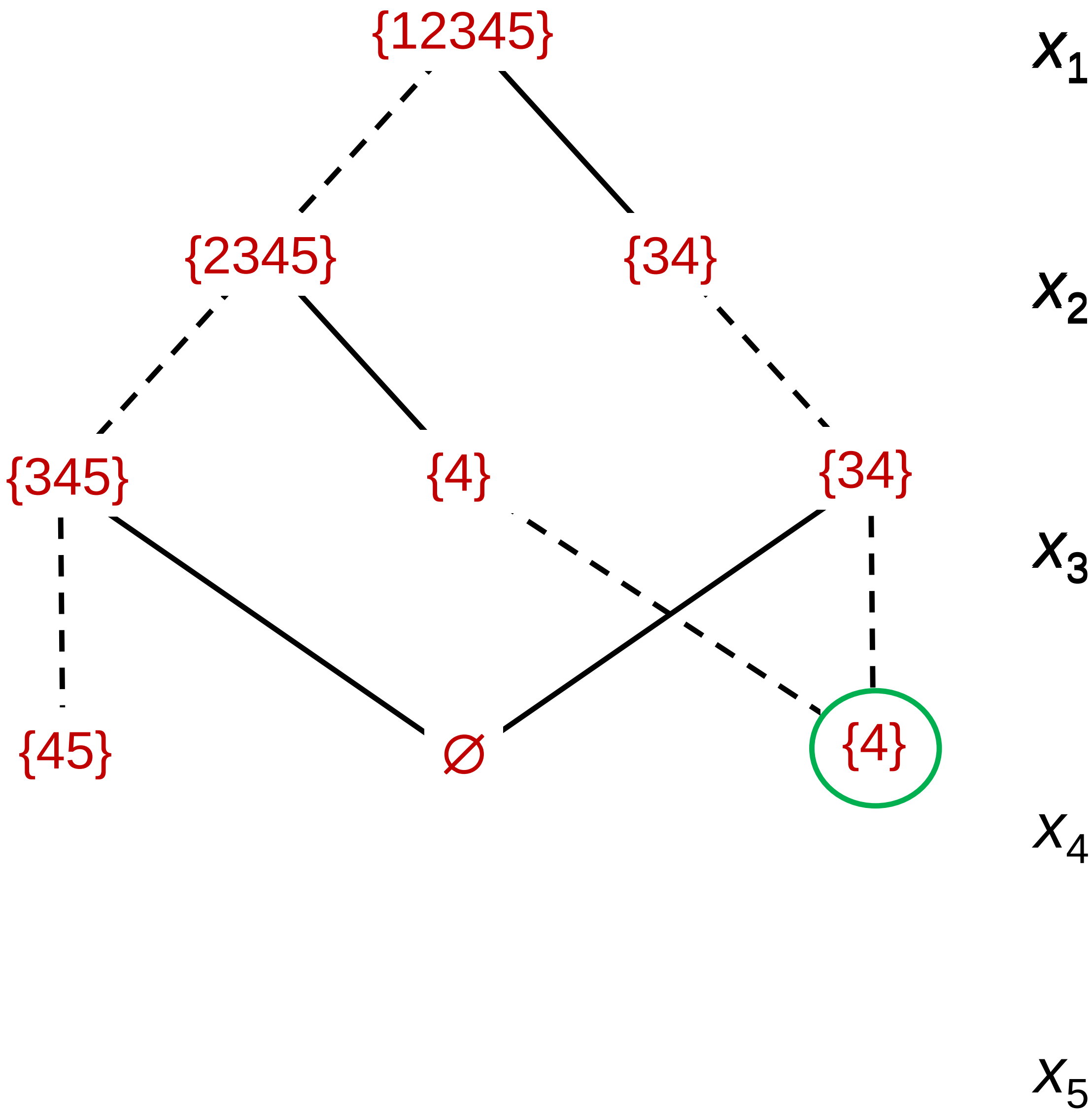




Exact DD for stable set problem

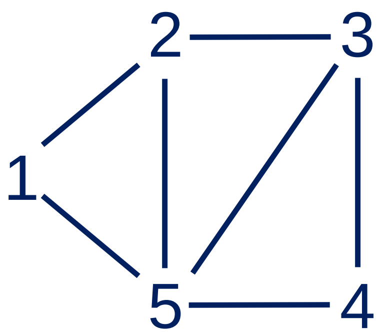
Merge nodes that correspond to the same state

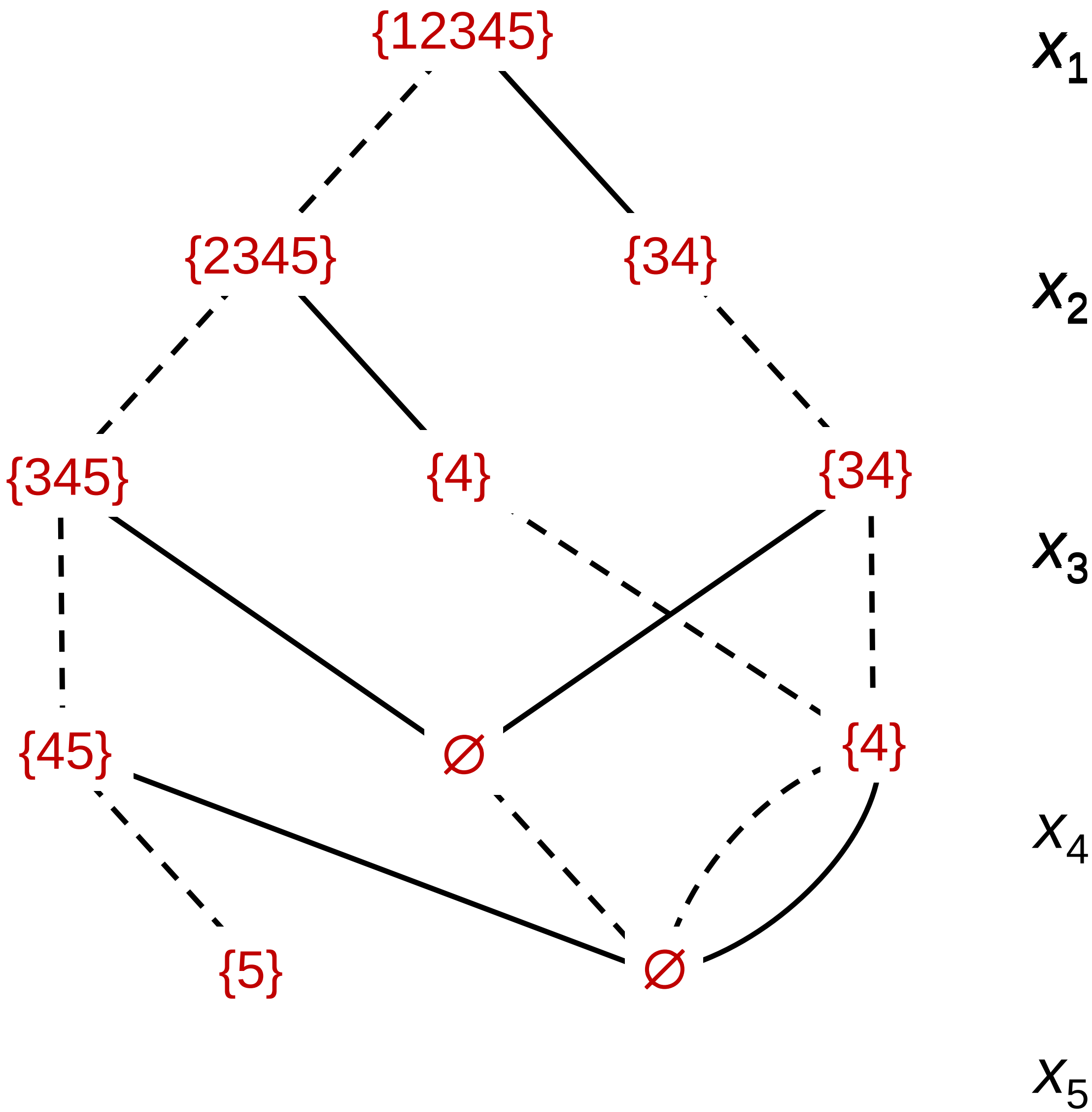




Exact DD for stable set problem

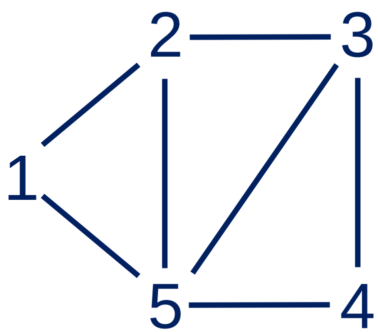
Merge nodes that correspond to the same state





Exact DD for stable set problem

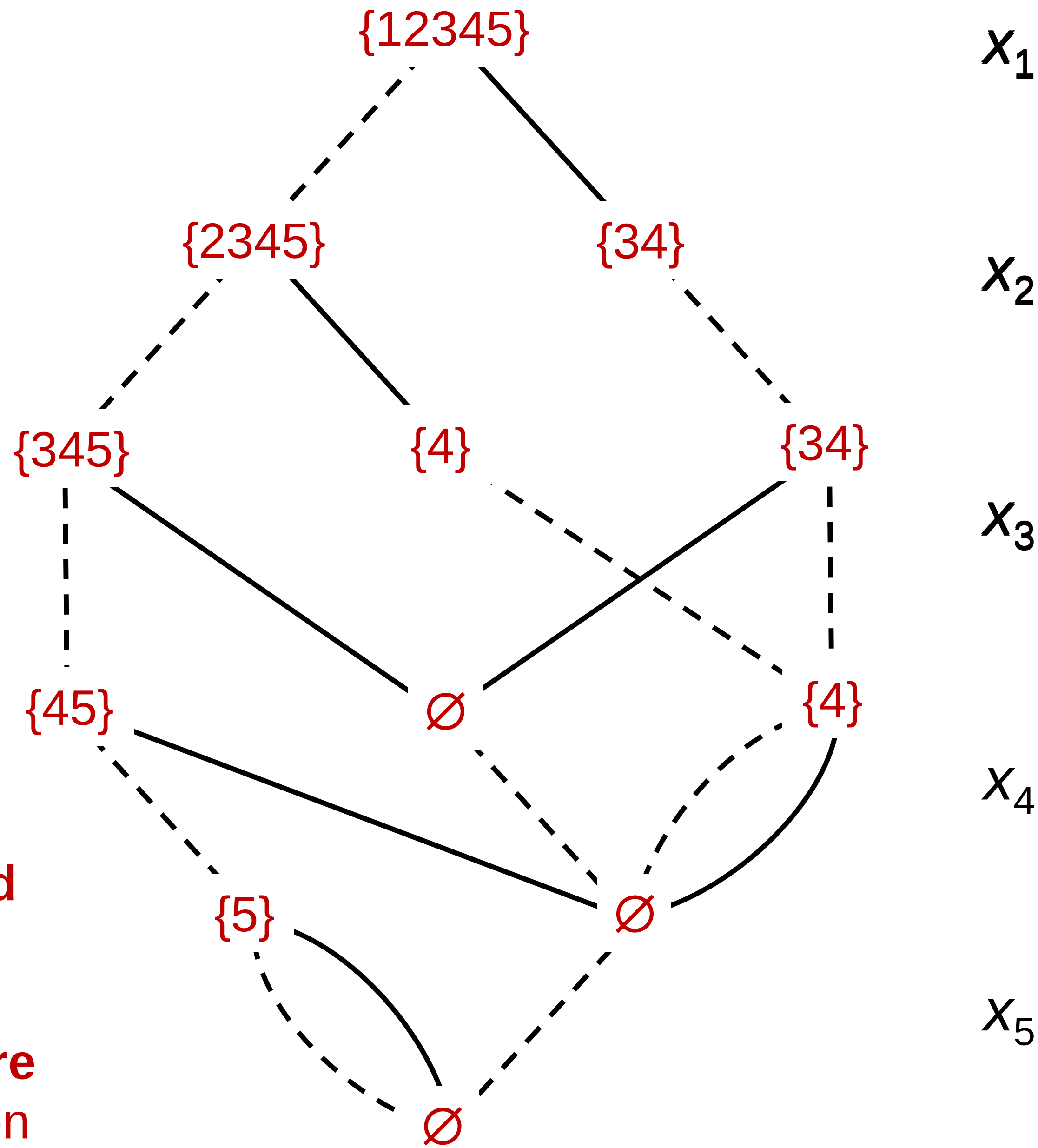
To build DD, associate **state** with each node



Exact DD for
stable set
problem

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

DD reduction is a **more powerful** simplification method than DP



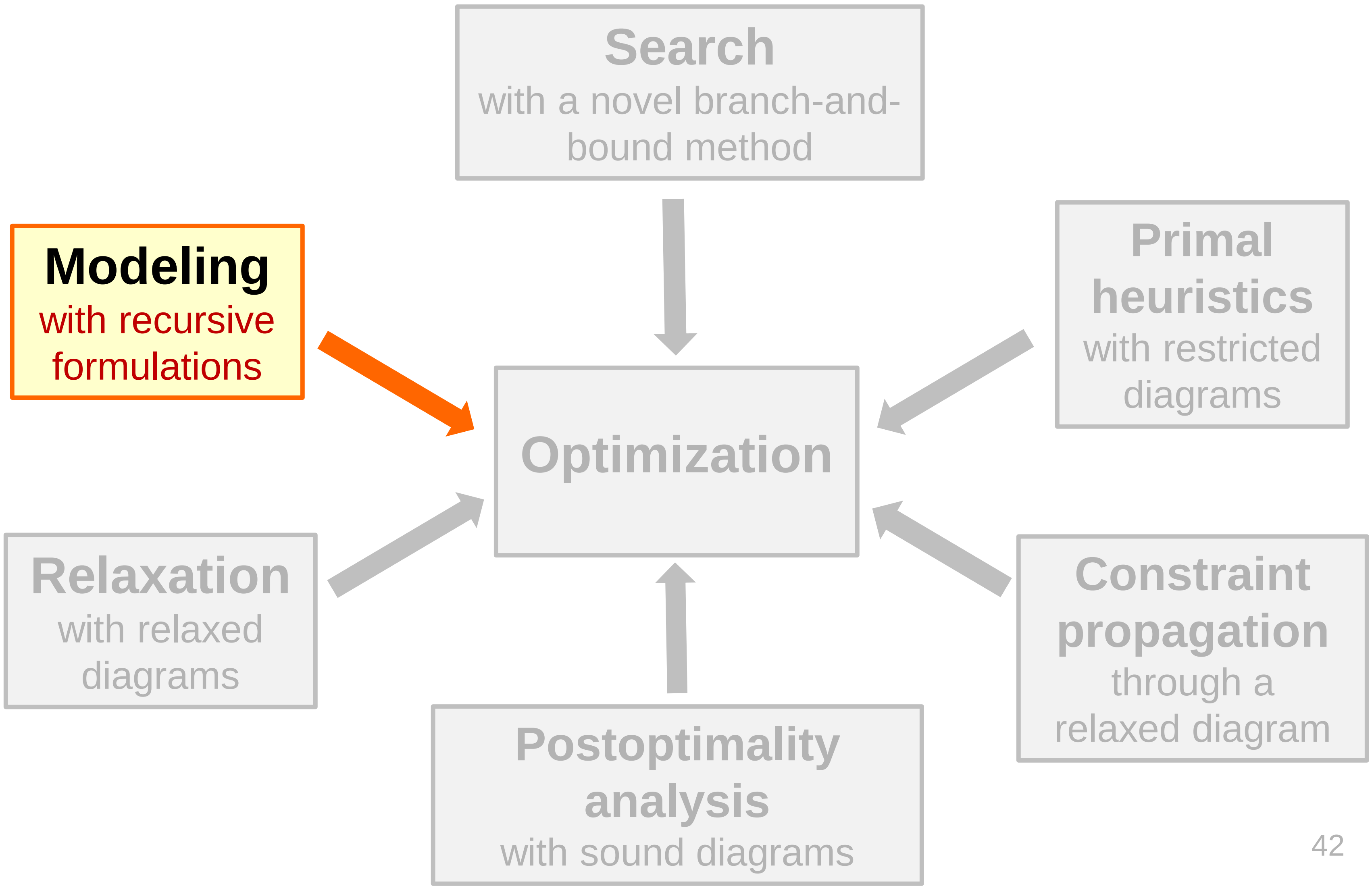
Outline

- Decision diagram **basics**
- Optimization with **exact** decision diagrams
- Providing the basic **elements of optimization**
  - **Modeling, relaxation, primal heuristics, constraint propagation, search, postoptimality**
- Research **frontiers**
  - Separation
  - Radical reduction of state space in DP
  - Nonlinear optimization
  - Nonserial recursion
- References

Toward a General-Purpose Solver

- The decision diagram tends to grow exponentially**.**
- To build a **practical solver:**
  - Use a **recursive dynamic programming model**.
  - Use limited-width **relaxed** decision diagrams to bound the objective value.
  - Use limited-width **restricted** decision diagrams for primal heuristic.
  - Use relaxed diagrams for **constraint propagation**.
  - Use **novel branching scheme** within relaxed decision diagrams.
  - Use **sound decision diagrams** for postoptimality analysis.

Elements of Optimization



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.
  - We **don’t care** if state space is **exponential**, because we don’t solve the problem by dynamic programming.

Dynamic Programming Model

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

Recursion:

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

Cost-to-go — g

State — J

Immediate cost (edge weight) — w_j

Vertex *j* and neighbors — $N(j)$

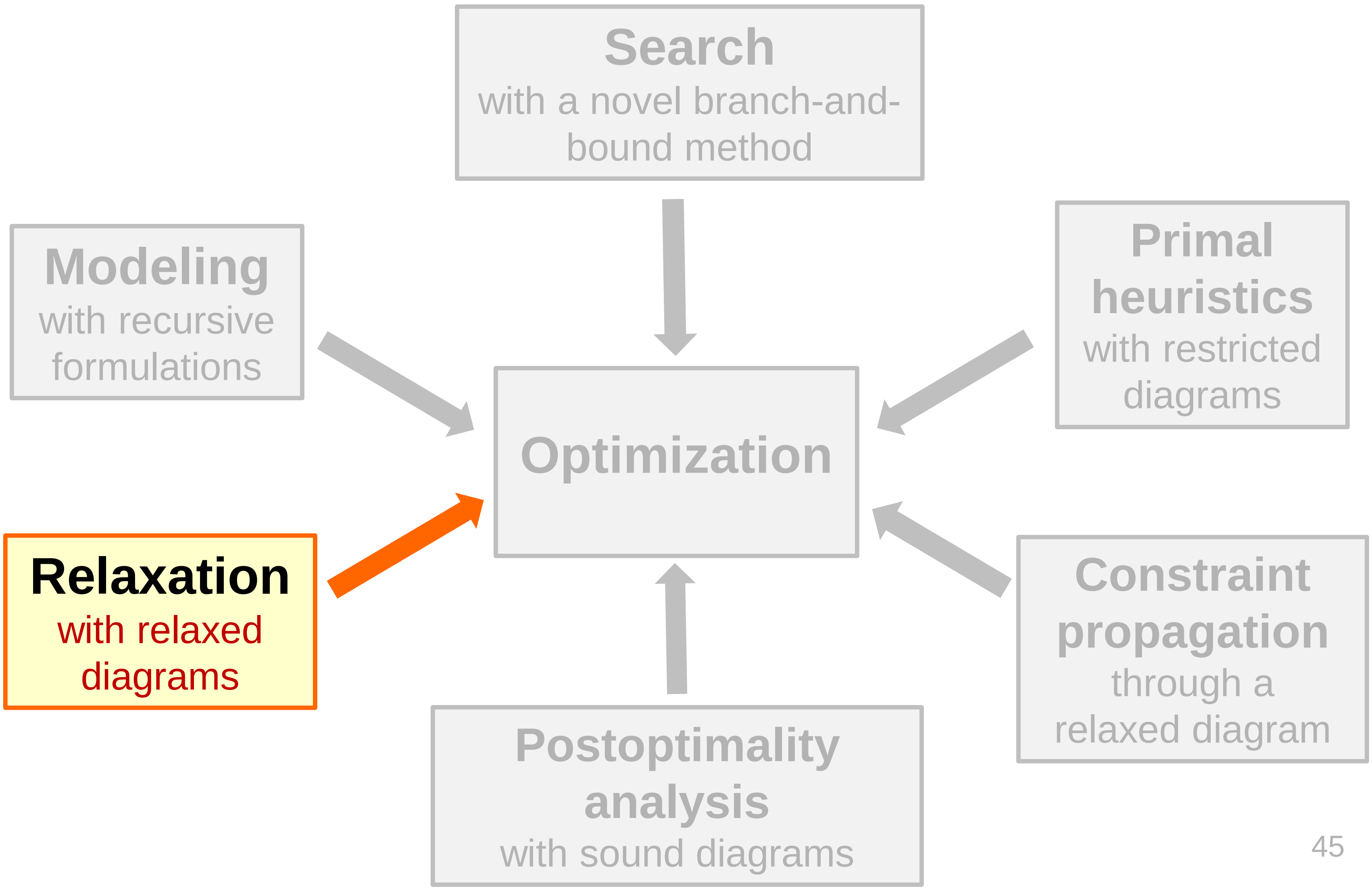
Boundary condition:

$$g(\emptyset) = 0$$

Optimal value:

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

Elements of Optimization



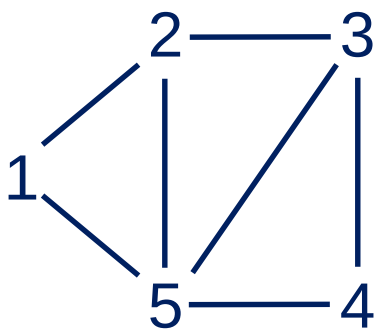
Relaxed Decision Diagrams

- An exact DD can grow **too large**.
  - So we use a smaller, **relaxed** DD

Relaxed Decision Diagrams

- An exact DD can grow **too large**.
  - So we use a smaller, **relaxed** DD
- 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)



{12345}

x_1

x_2

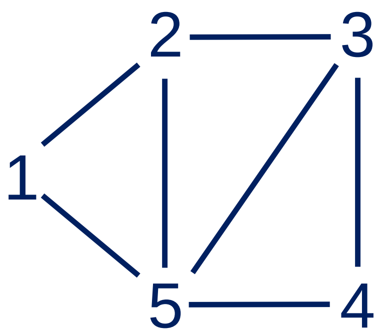
Stable set problem

x_3

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

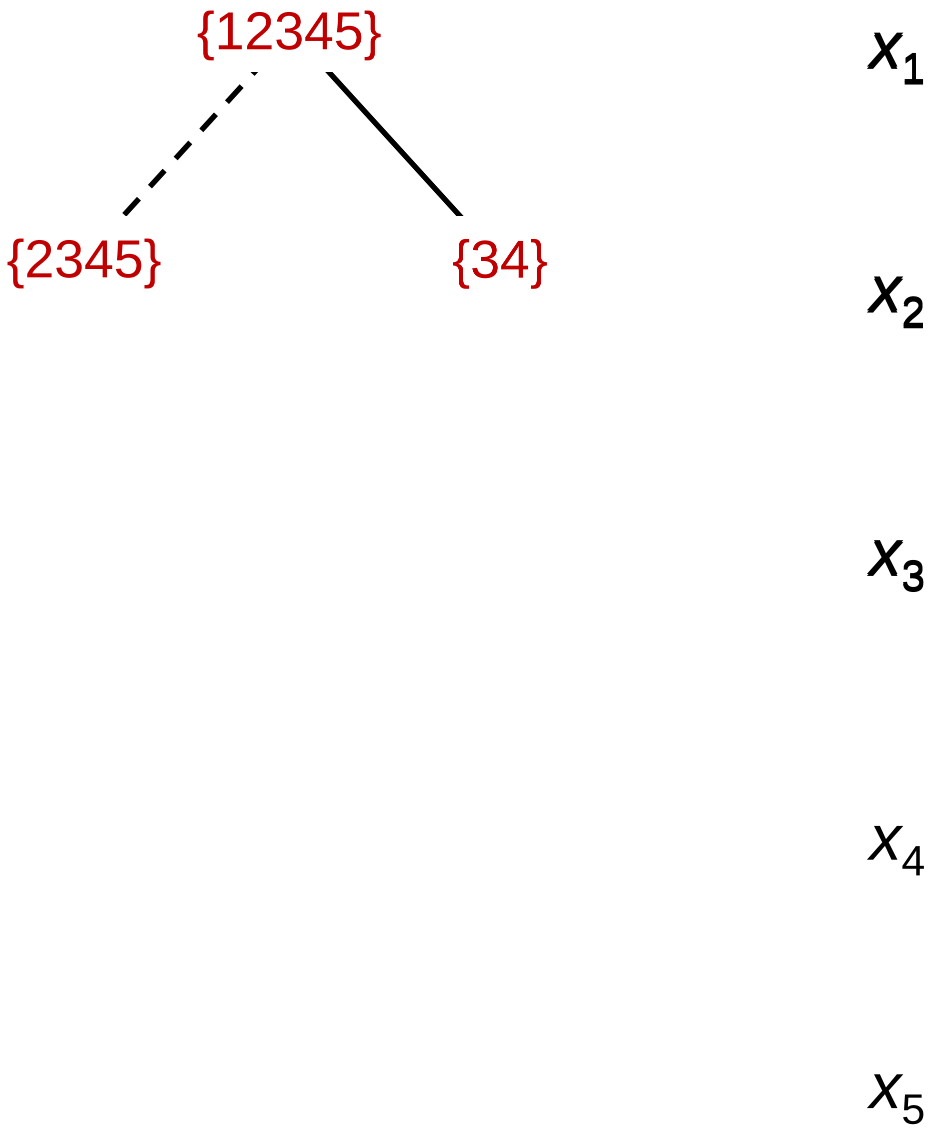
x_4

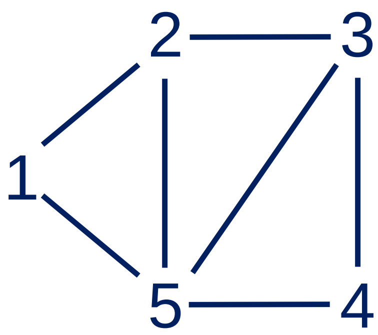
x_5



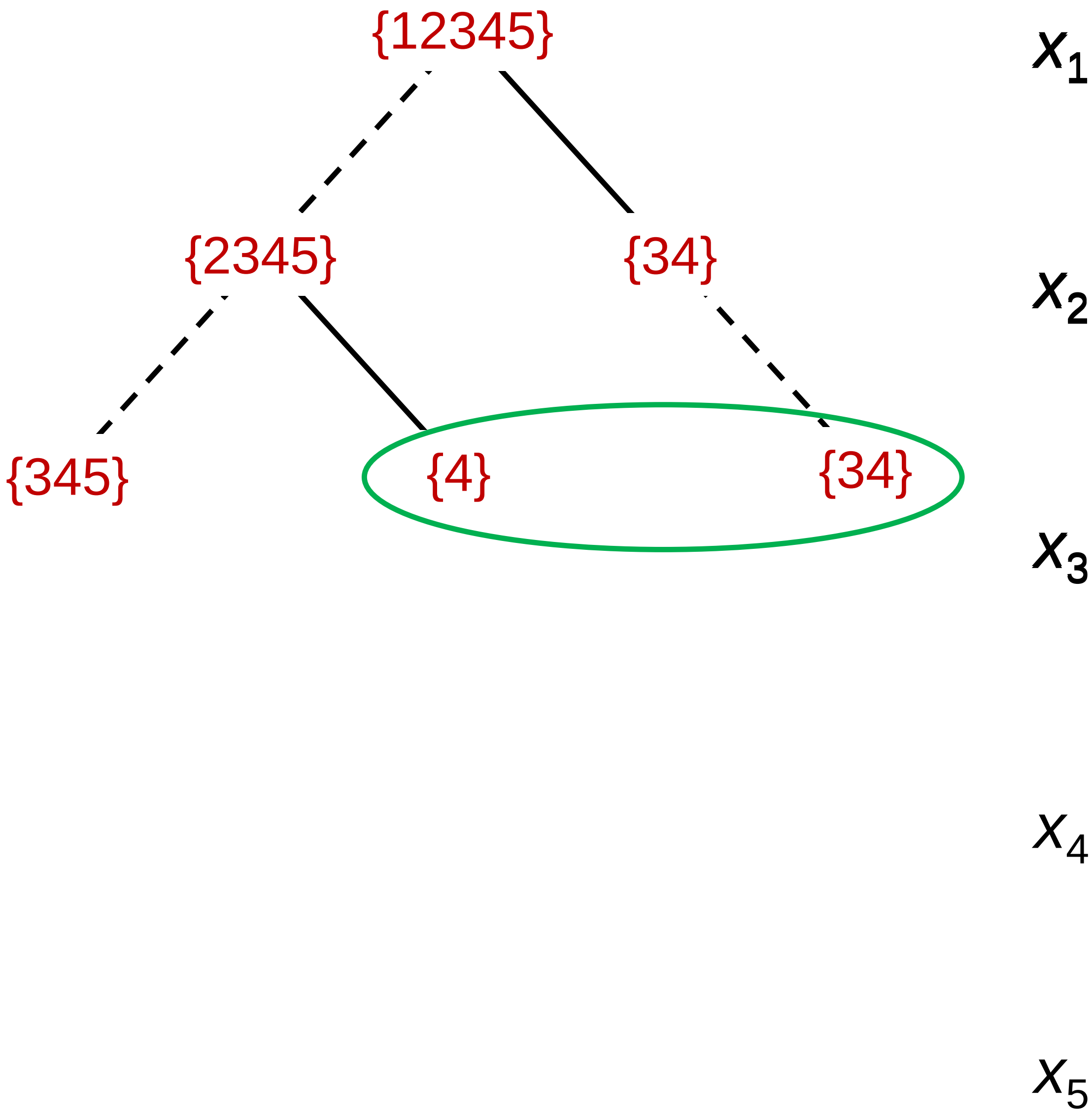
Stable set problem

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



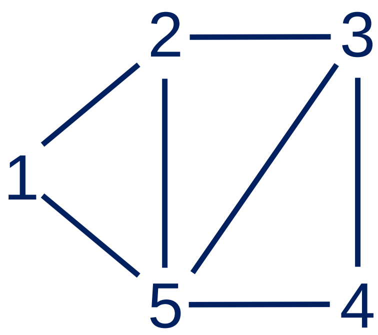


Stable set problem

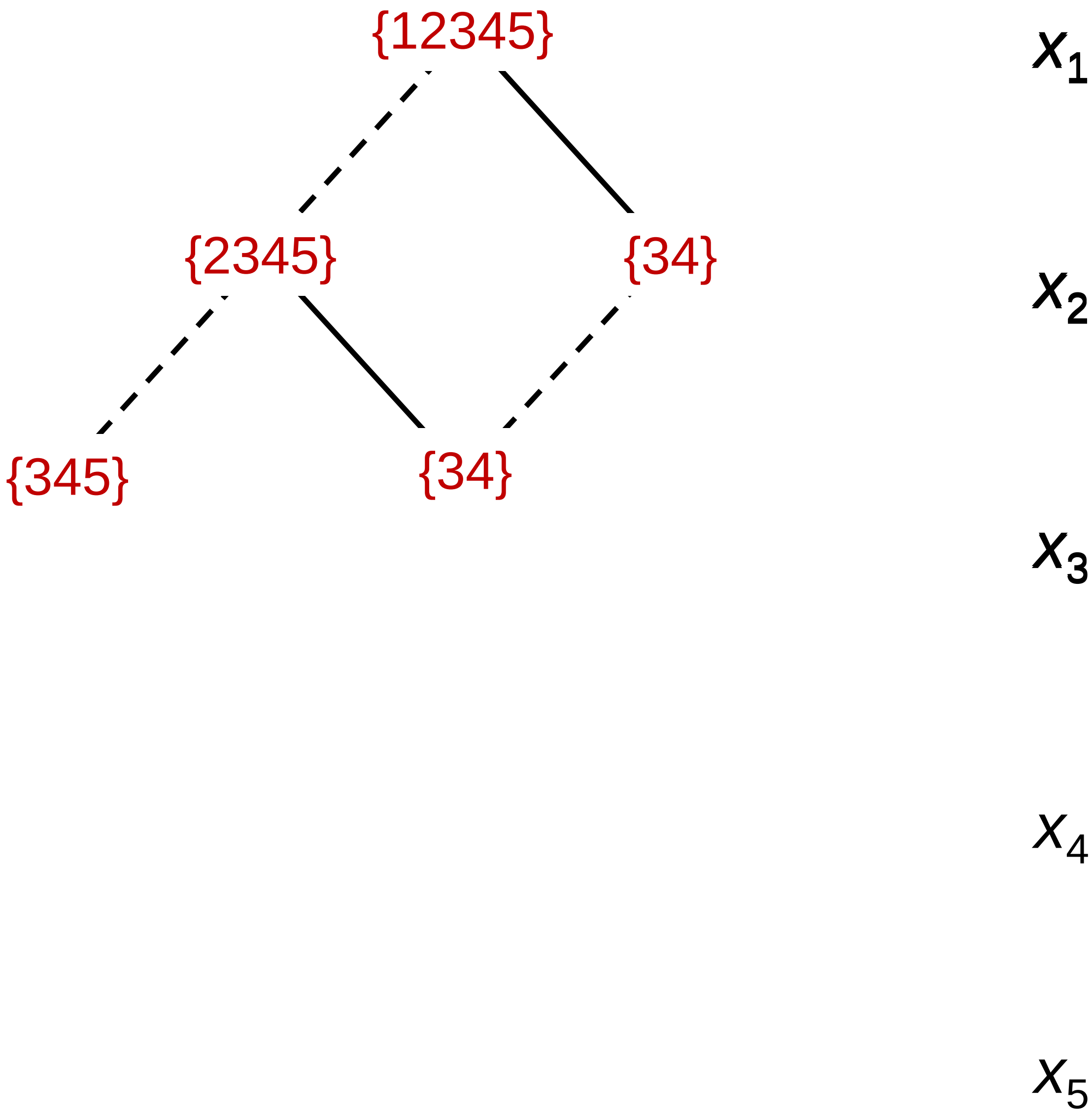


To build **relaxed** DD, merge some additional nodes as we go along.

Take the **union** of merged states

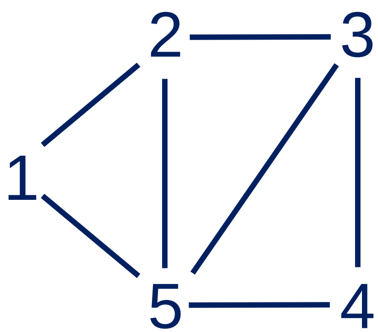


Stable set problem



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

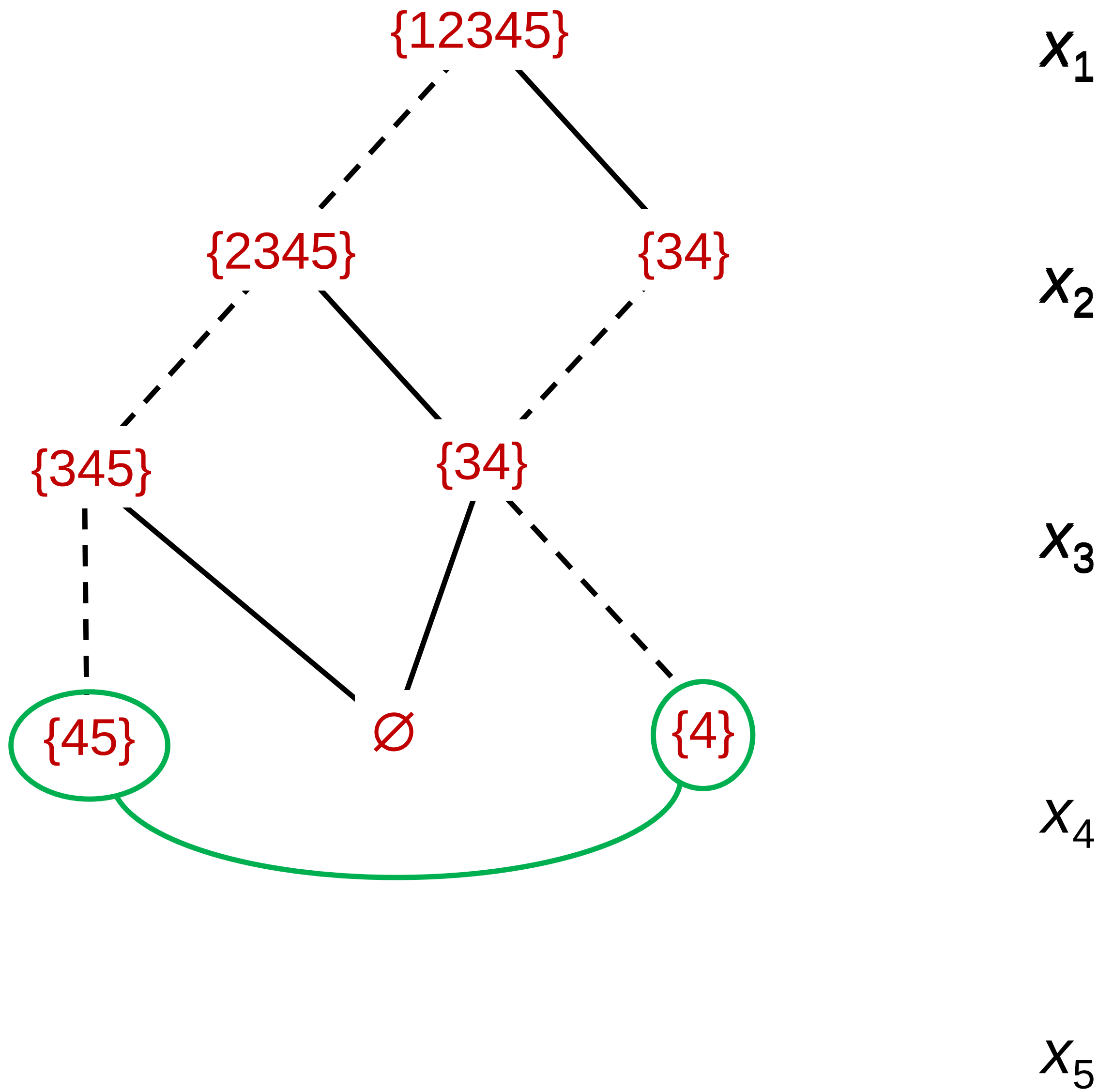
Take the **union** of merged states.

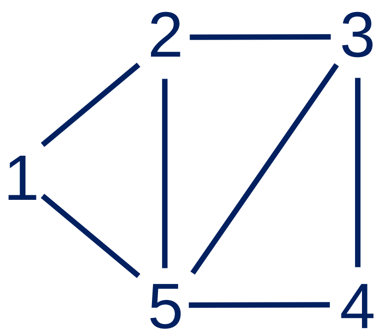


Stable set problem

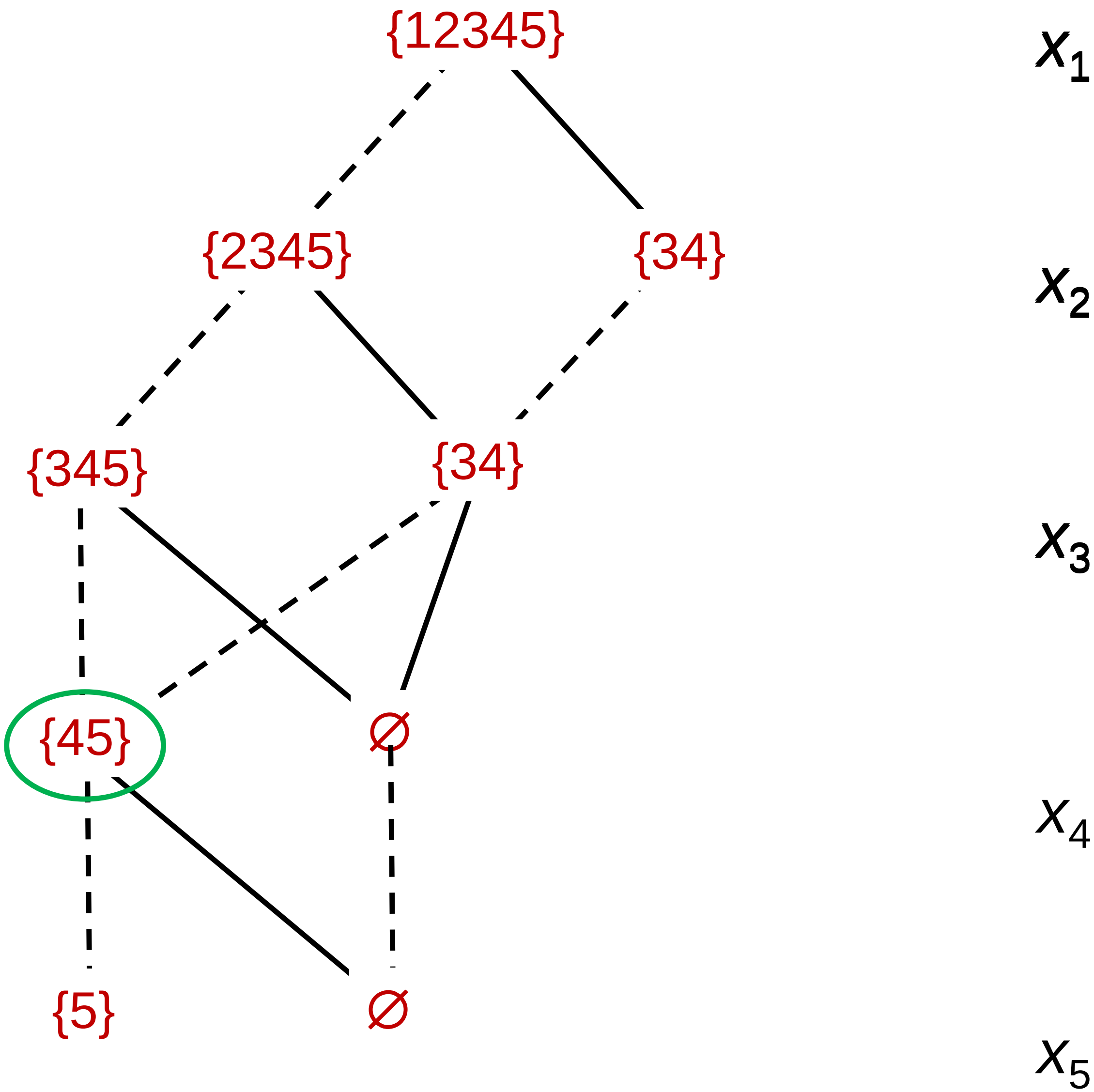
To build **relaxed** DD, merge some additional nodes as we go along.

Take the **union** of merged states.



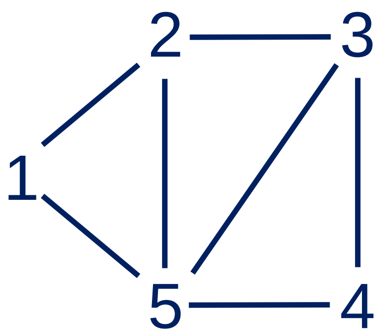


Stable set problem



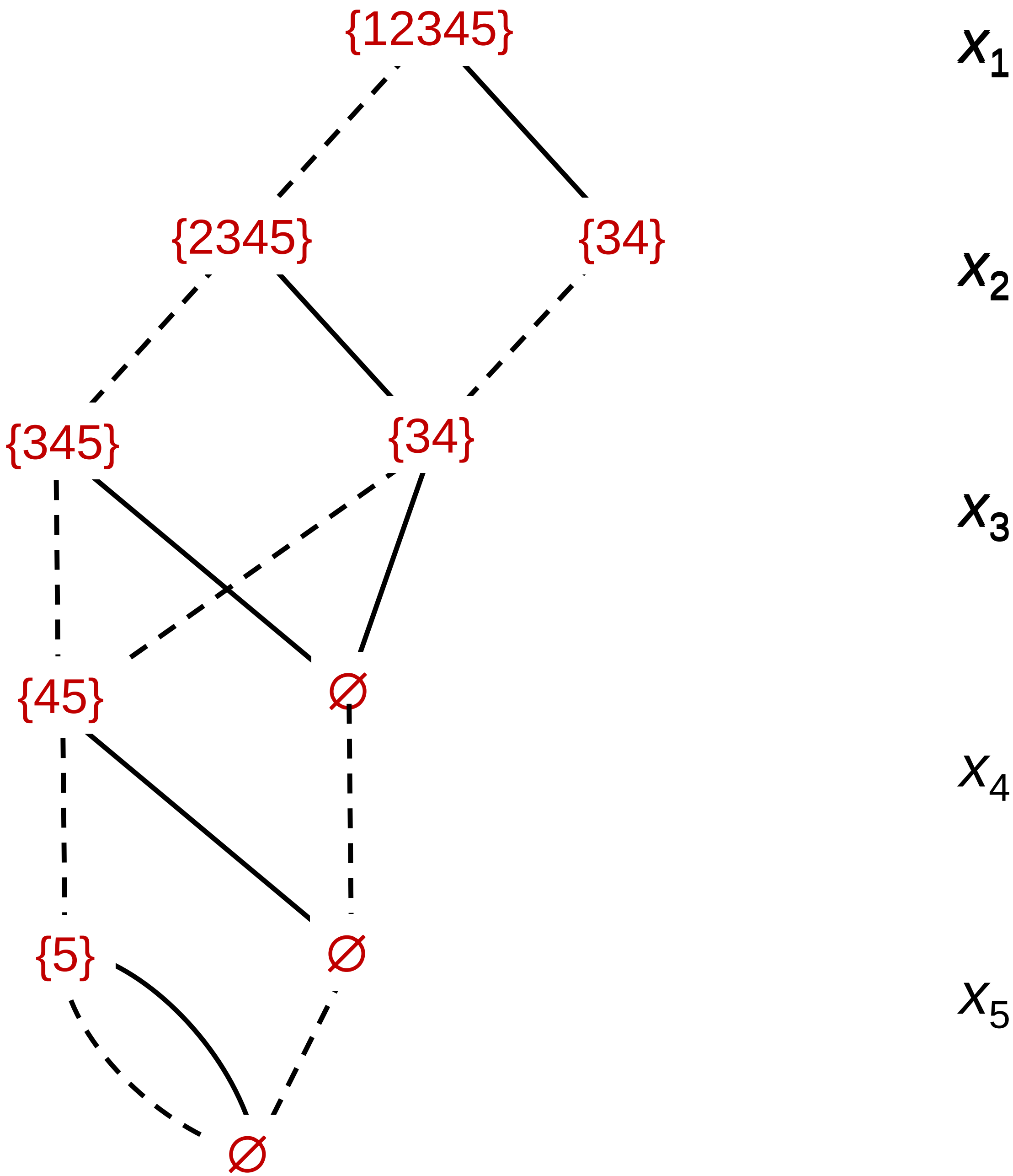
To build **relaxed** DD, merge some additional nodes as we go along.

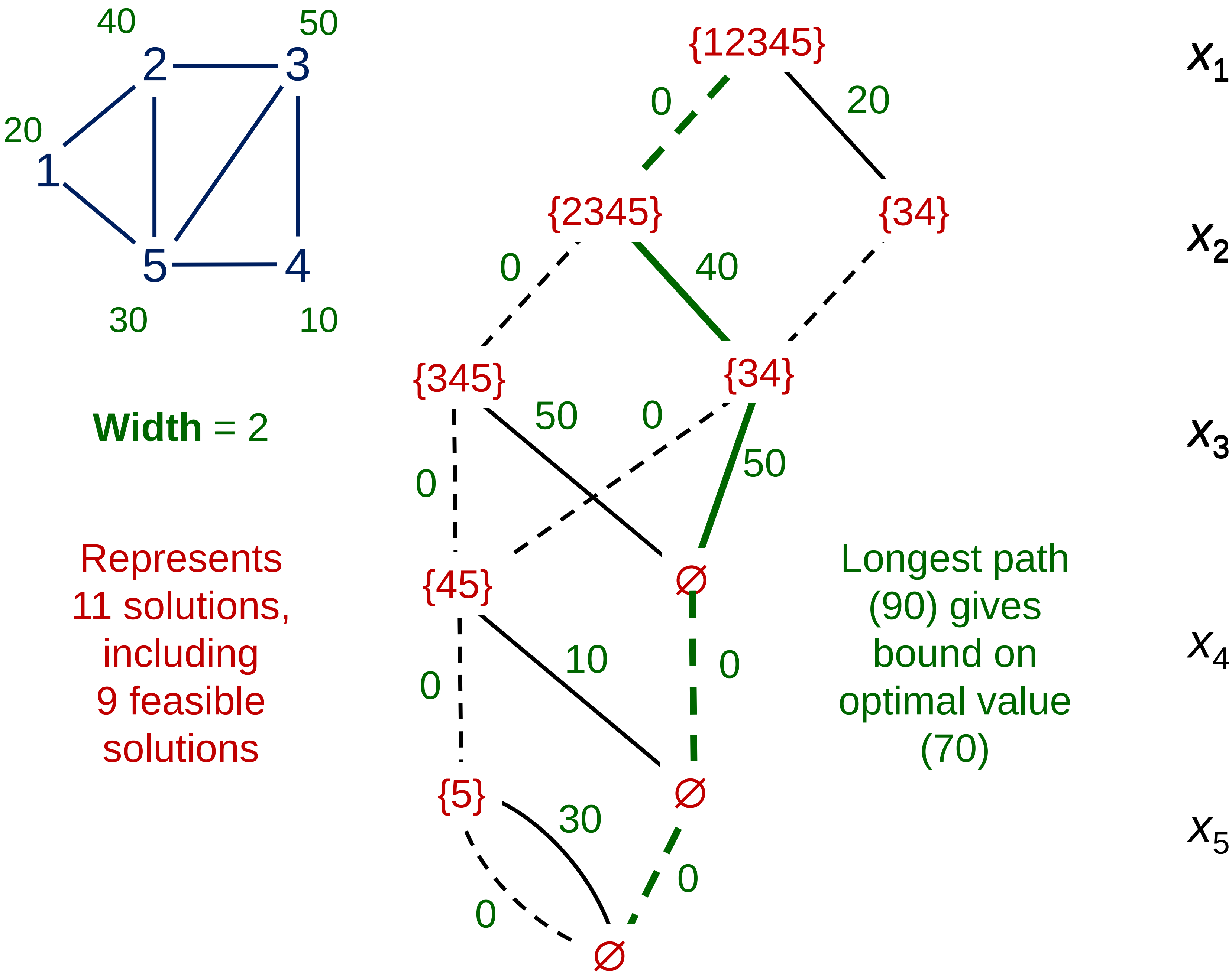
Take the **union** of merged states.



Width = 2

Represents 11 solutions, including 9 feasible solutions





40
50
2
3
20
1
5
4
30
10
Width = 2
Represents
11 solutions,
including
9 feasible
solutions
{12345}
{2345}
{34}
{345}
{34}
{45}
∅
{5}
∅
∅
0
20
0
40
50
0
0
50
0
10
0
30
0
0
Longest path
(90) gives
bound on
optimal value
(70)
x_1
x_2
x_3
x_4
x_5

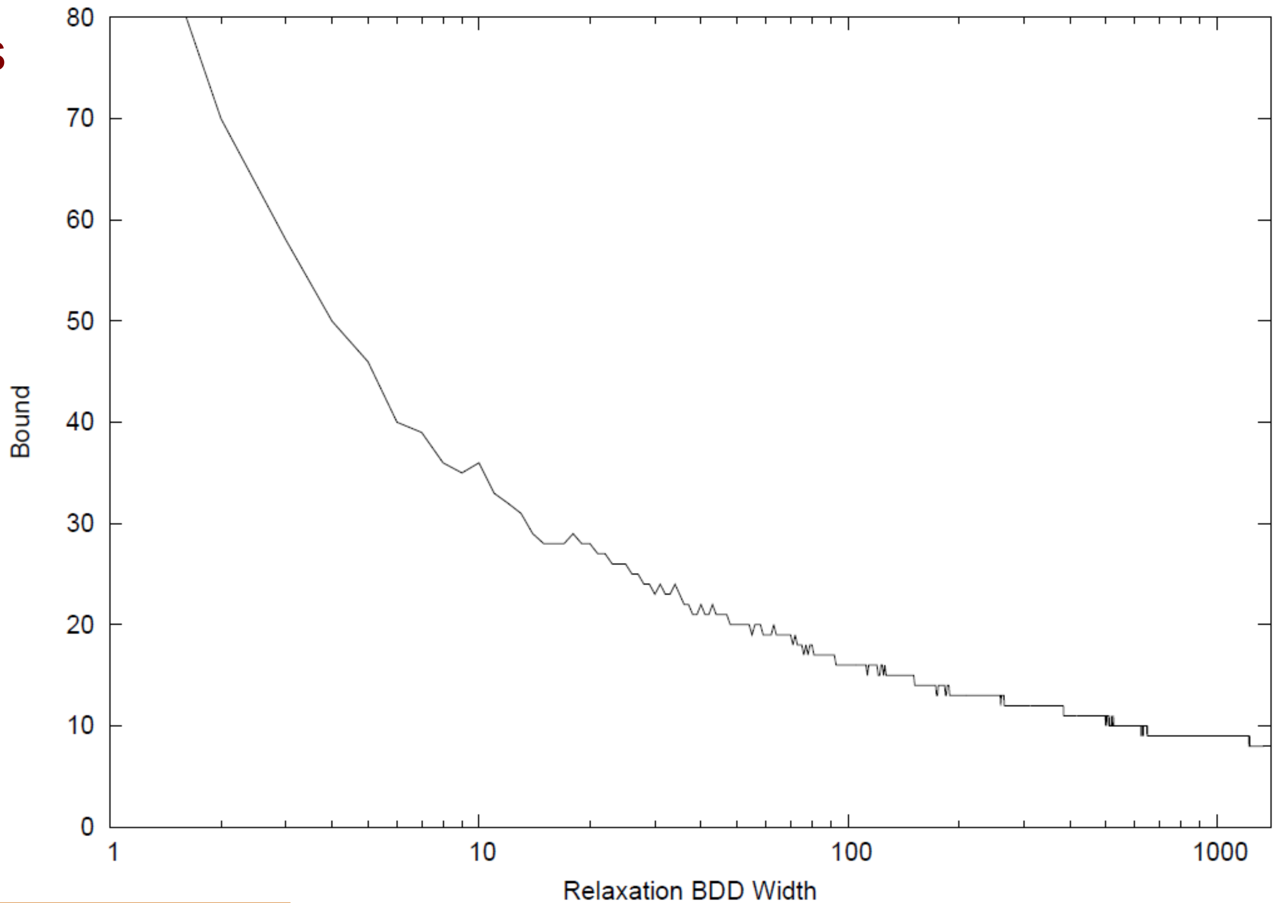
Relaxed Decision Diagrams

- Alternate relaxation method: **node refinement**.
  - Start with DD of width 1 representing Cartesian product of variable domains.
  - Split nodes so as to remove some infeasible paths.
  - Will be illustrated in **constraint propagation**.

Andersen, Hadžić, JH, Tiedemann (2007)

Relaxed Decision Diagrams

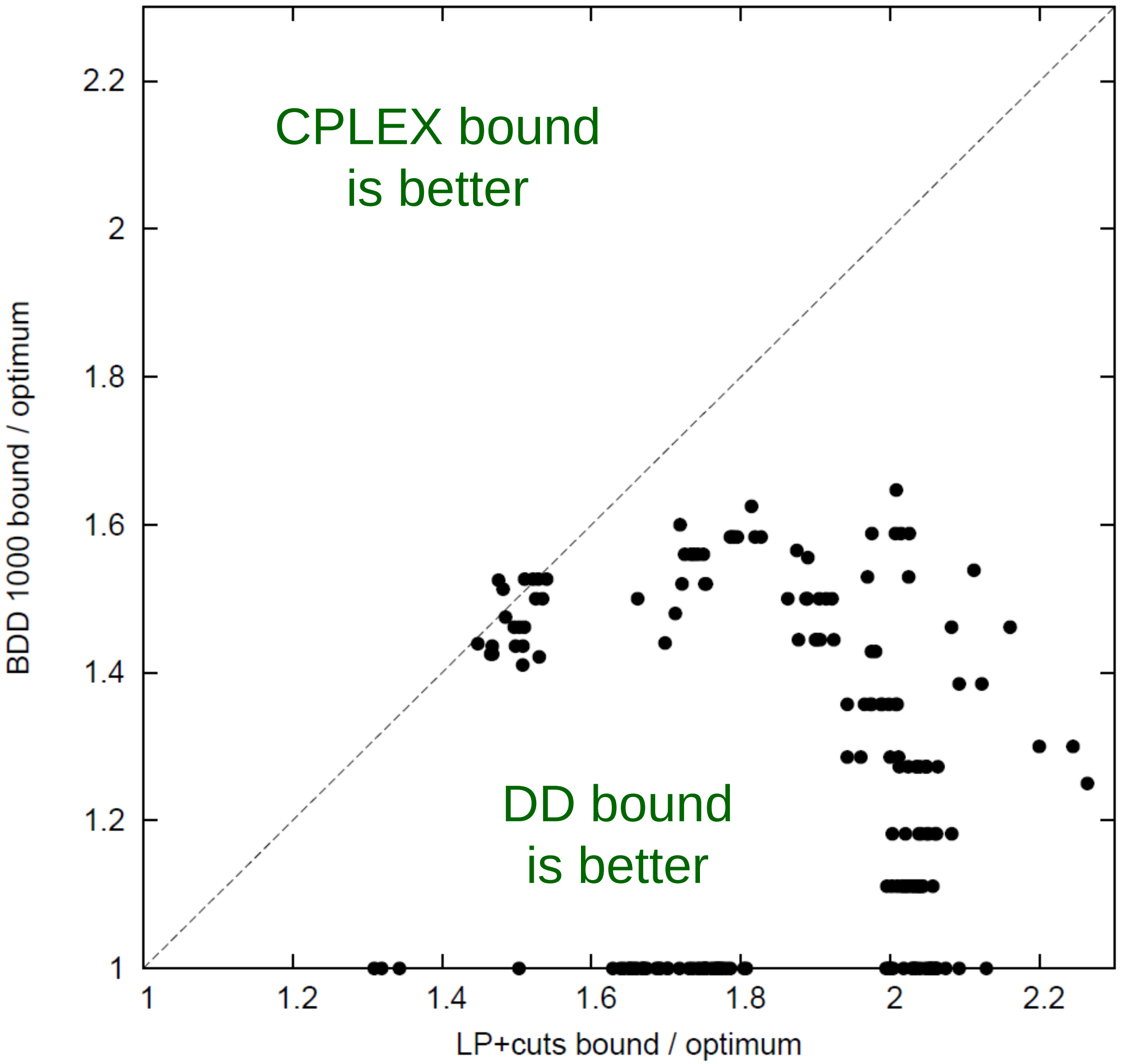
- Wider diagrams yield tighter bounds
  - But take longer to build.
  - Adjust width dynamically.



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 (2014)

Relaxing DP Models

- Decision diagrams provide **a general method** for relaxing dynamic programming models.
  - Including problem for which no practical relaxation exists.
- Example: job sequencing with **sequence-dependent setup times** and time windows.
  - Setup time is **less** when certain jobs have already been processed.
  - A **hard problem** to solve exactly.
  - **No useful relaxation**.

JH (2017)

DP Model for Job Sequencing

Set of jobs scheduled so far

State: $S = (V, f)$

Initial state = $(\emptyset, 0)$

Finish time of last job scheduled

Controls: $x_j(V, f) = \{1, \ldots, n\} \setminus V$

Immediate cost: $c_j\big((V, f), x_j\big) = \big(\max\{r_{x_j}, f\} + p_{x_j}(V) - d_{x_j}\big)^+$

State-dependent processing time

$$h_j(S) = \min_{x_j \in X_j(S)} \Big\{ c_j(S, x_j) + h_{j+1}\big((\phi_j(S, x_j)\big) \Big\}$$

State in stage *j*

Set of possible controls

Immediate cost

Cost to go

DP Model for Job Sequencing

State: $S = (V, f)$

- V: Set of jobs scheduled so far
- f: Finish time of last job scheduled

Initial state = $(\emptyset, 0)$

Controls: $x_j(V, f) = \{1, \ldots, n\} \setminus V$

Immediate cost: $c_j\big((V, f), x_j\big) = \big(\max\{r_{x_j}, f\} + p_{x_j}(V) - d_{x_j}\big)^+$

Transition: $\phi_j\big((V, f), x_j\big) = \big(V \cup \{x_j\}, \max\{r_{x_j}, f\} + p_{x_j}(V)\big)$

$$h_j(S) = \min_{x_j \in X_j(S)} \Big\{ c_j(S, x_j) + h_{j+1}\big((\phi_j(S, x_j)\big) \Big\}$$

- S: State in stage *j*
- $X_j(S)$: Set of possible controls
- $c_j(S, x_j)$: Immediate cost
- $h_{j+1}\big((\phi_j(S, x_j)\big)$: Cost to go

Relaxed DP Model

Set of jobs scheduled in **all** feasible solutions so far

$$S = (V, U, f)$$

Initial state = $(\emptyset, \emptyset, 0)$

New state variable: set of jobs scheduled in **some** feasible solution so far

Earliest possible finish time of immediately previous job

Transition:

$$\phi_j\big((V,U,f),x_j\big) = \big(V \cup \{x_j\}, U \cup \{x_j\}, \max\{r_{x_j}, f\} + p_{x_j}(U)\big)$$

Processing time depends on *U*, not *V*
(state variable *V* can be dropped if desired)

$$h_j(S) = \min_{x_j \in X_j(S)} \Big\{ c_j(S, x_j) + h_{j+1}\big((\phi_j(S, x_j))\big) \Big\}$$

Node Merger in Relaxation

- Merge nodes as the diagram is constructed
  - States *S*, *T* merge to form state $S \oplus T$
- Merger operation must yield valid relaxation
  - In state-dependent job sequencing,

$$(V, U, f) \oplus (V', U', f') = \big(V \cap V', U \cup U', \min\{f, f'\}\big)$$

Conditions for a Valid Relaxation

- There are two jointly sufficient conditions for obtaining a relaxed diagram from node merger.
  - A condition on the transition function
  - And a condition on the merger operation $S \oplus T$

Conditions for a Valid Relaxation

- There are two jointly sufficient conditions for obtaining a relaxed diagram from node merger.
  - A condition on the transition function
  - And a condition on the merger operation $S \oplus T$
- First we need a definition: state S' relaxes state S in the same stage if
  - Every control feasible in S is feasible in S'

$$X_j(S) \subseteq X_j(S')$$

  - The immediate cost of a control feasible in S is no greater in S'.

$$c_j(S, x_j) \geq c_j(S', x_j), \quad \text{all } x_j \in X_j(S)$$

Conditions for a Valid Relaxation

Theorem. The merger of states *S* and *T* in layer *j* of diagram *D* yields a relaxation of *D* if:

- S′ relaxes S implies that $\phi_j(S', x_j)$ relaxes $\phi_j(S, x_j)$ for any control x_j feasible in *S*.
- $S \oplus T$ relaxes both *S* and *T*.

Proof by induction.

This generalizes to **stochastic** decision diagrams, where the conditions are much more complicated.

JH (2017)

Conditions for a Valid Relaxation

It is easily checked that node merger for job sequencing satisfies these conditions.

- S' relaxes S implies that $\phi_j(S', x_j)$ relaxes $\phi_j(S, x_j)$ for any control x_j feasible in *S*.

- $S \oplus T$ relaxes both *S* and *T*.

Merger Heuristics

- Goal:
  - Merge nodes that are **not likely to lie on short paths**.
  - This reduces the likelihood of creating a superoptimal path, which would weaken the bound.
  - Keep merging nodes until desired width is obtained.

Merger Heuristics

- Goal:
  - Merge nodes that are **not likely to lie on short paths**.
  - This reduces the likelihood of creating a superoptimal path, which would weaken the bound.
  - Keep merging nodes until desired width is obtained.
- Underlying idea
  - Preserve accuracy in the region of the diagram that is likely to contain the best solutions.
  - Analogous to using denser finite elements in models of the atmosphere in regions with more activity.

Merger Heuristics

- Finish time heuristic
  - Merge nodes whose **last finish time** states are large.
  - Paths through these nodes are likely to be longer.

Merger Heuristics

- Finish time heuristic
  - Merge nodes whose **last finish time** states are large.
  - Paths through these nodes are likely to be longer.
- Shortest path heuristic
  - Merge nodes whose **shortest-path distances from root** are large.

Merger Heuristics

- Finish time heuristic
  - Merge nodes whose **last finish time** states are large.
  - Paths through these nodes are likely to be longer.
- Shortest path heuristic
  - Merge nodes whose **shortest-path distances from root** are large.

– Random heuristic
  - Randomly choose nodes for merger..

Merger Heuristics

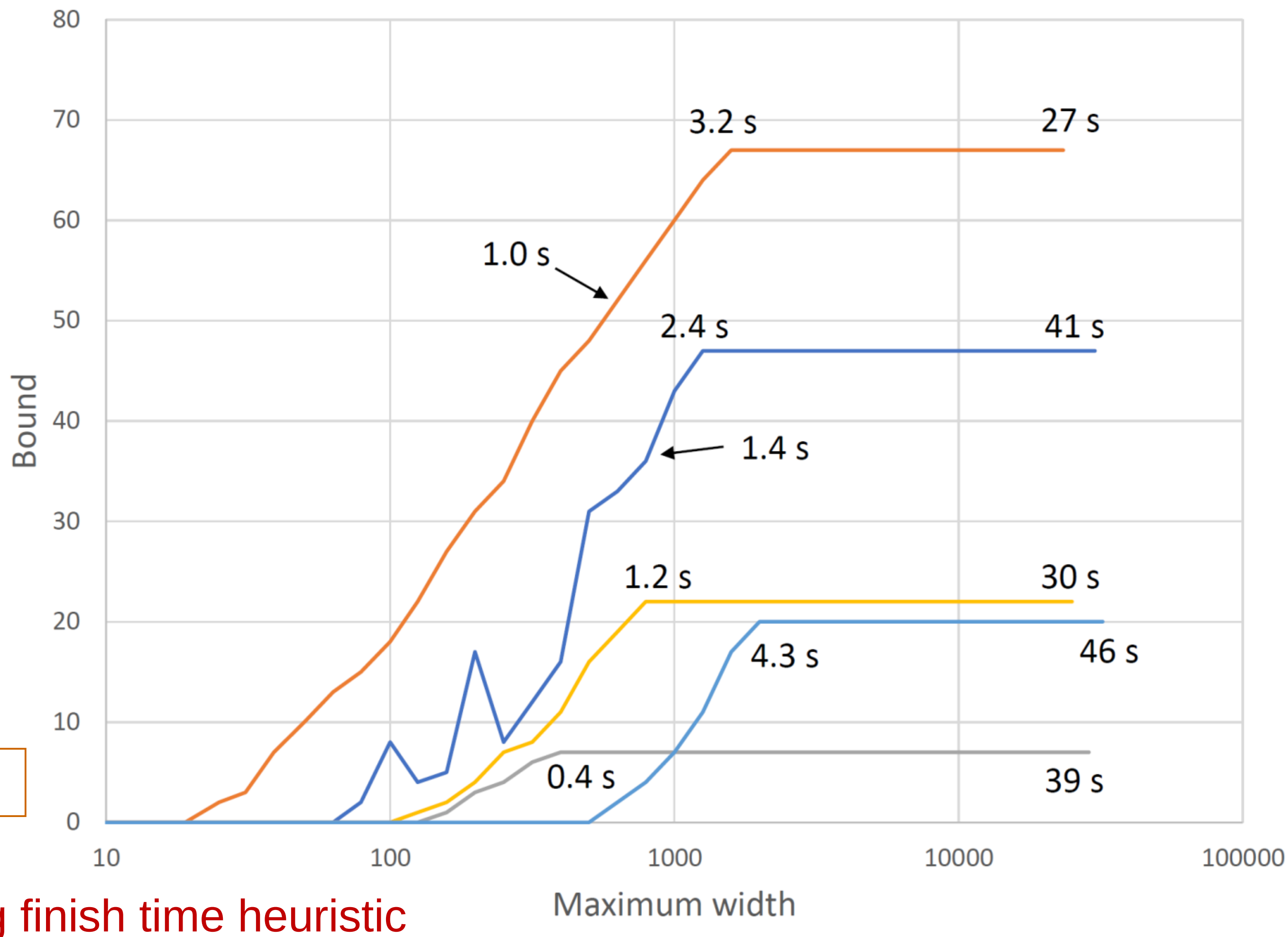
- Finish time heuristic – **By far the best**
  - Merge nodes whose **last finish time** states are large.
  - Paths through these nodes are likely to be longer.
- Shortest path heuristic
  - Merge nodes whose **shortest-path distances from root** are large.

– Random heuristic

  - Randomly choose nodes for merger..

Computational Results

12 jobs

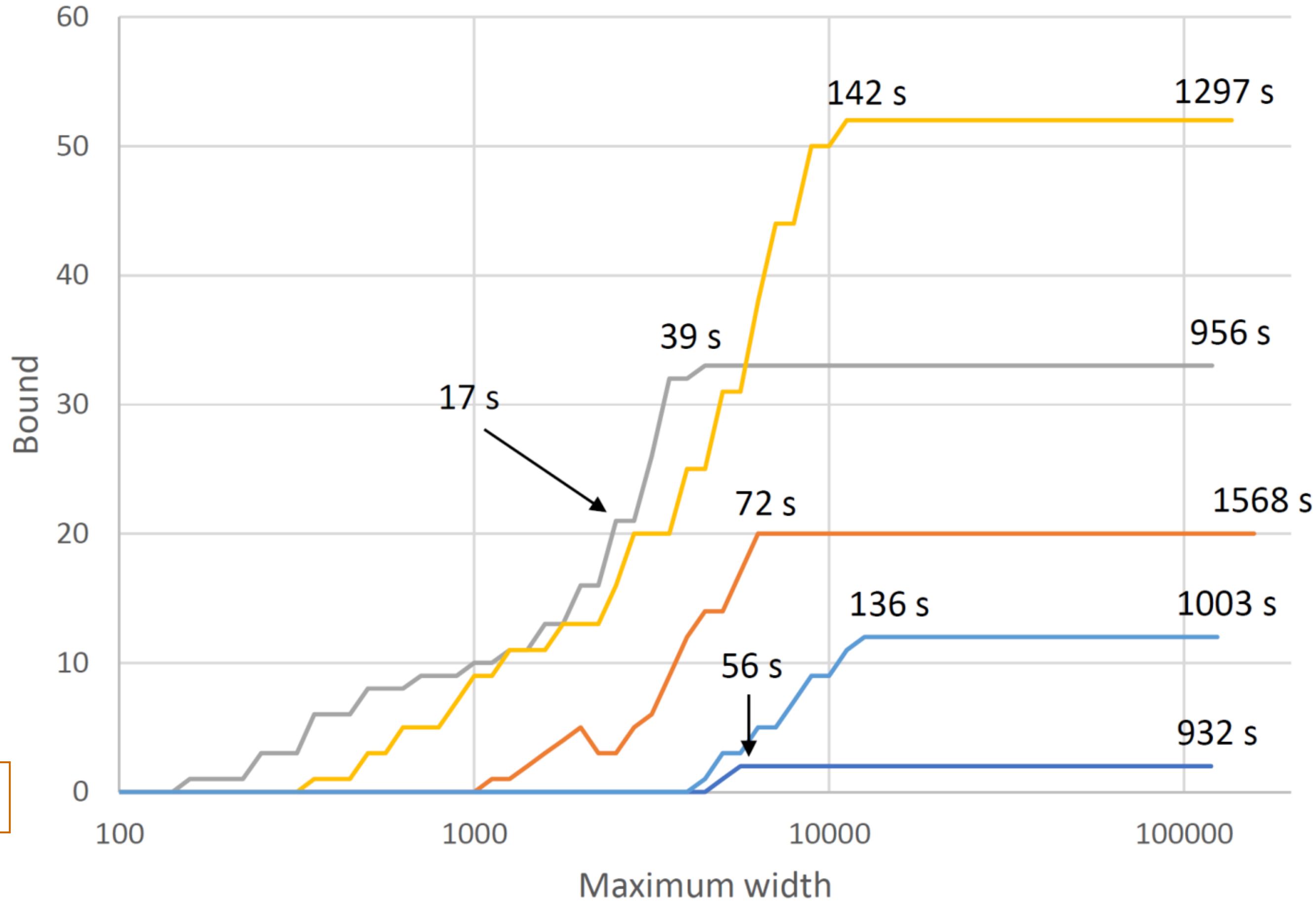


JH (2017)

Using finish time heuristic

Computational Results

14 jobs



JH (2017)

Using finish time heuristic

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)

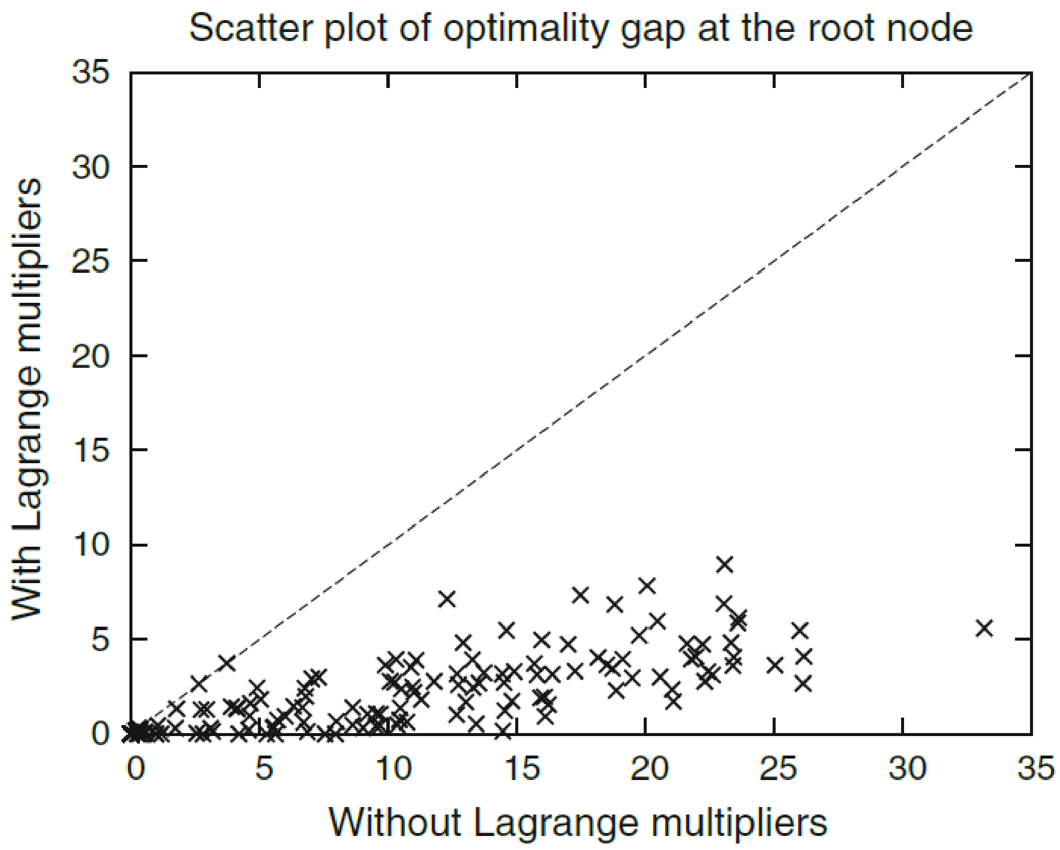
Improving the Bound

- A simple **Lagrangian** technique can improve the bound provided by a relaxed DD.
  - Increase costs on infeasible paths.
- This is effective on TSPTW, etc.
  - May also be useful for general DP models.

Bergman, Ciré, van Hoeve (2015)

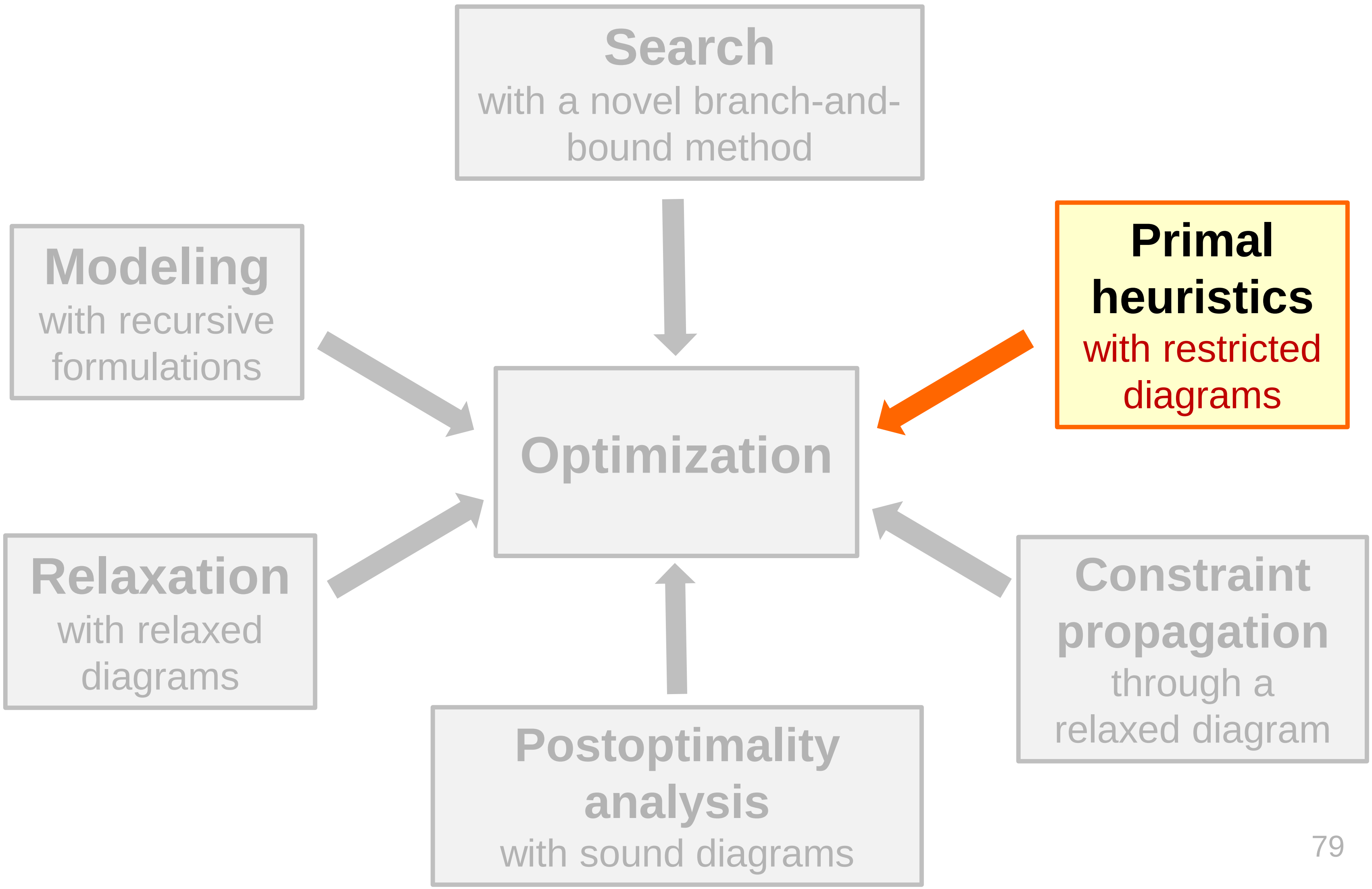
Improving the Bound

Effect of Lagrangian relaxation on quality of bound in TSPTW



Bergman, Ciré, van Hoeve (2015)

Elements of Optimization



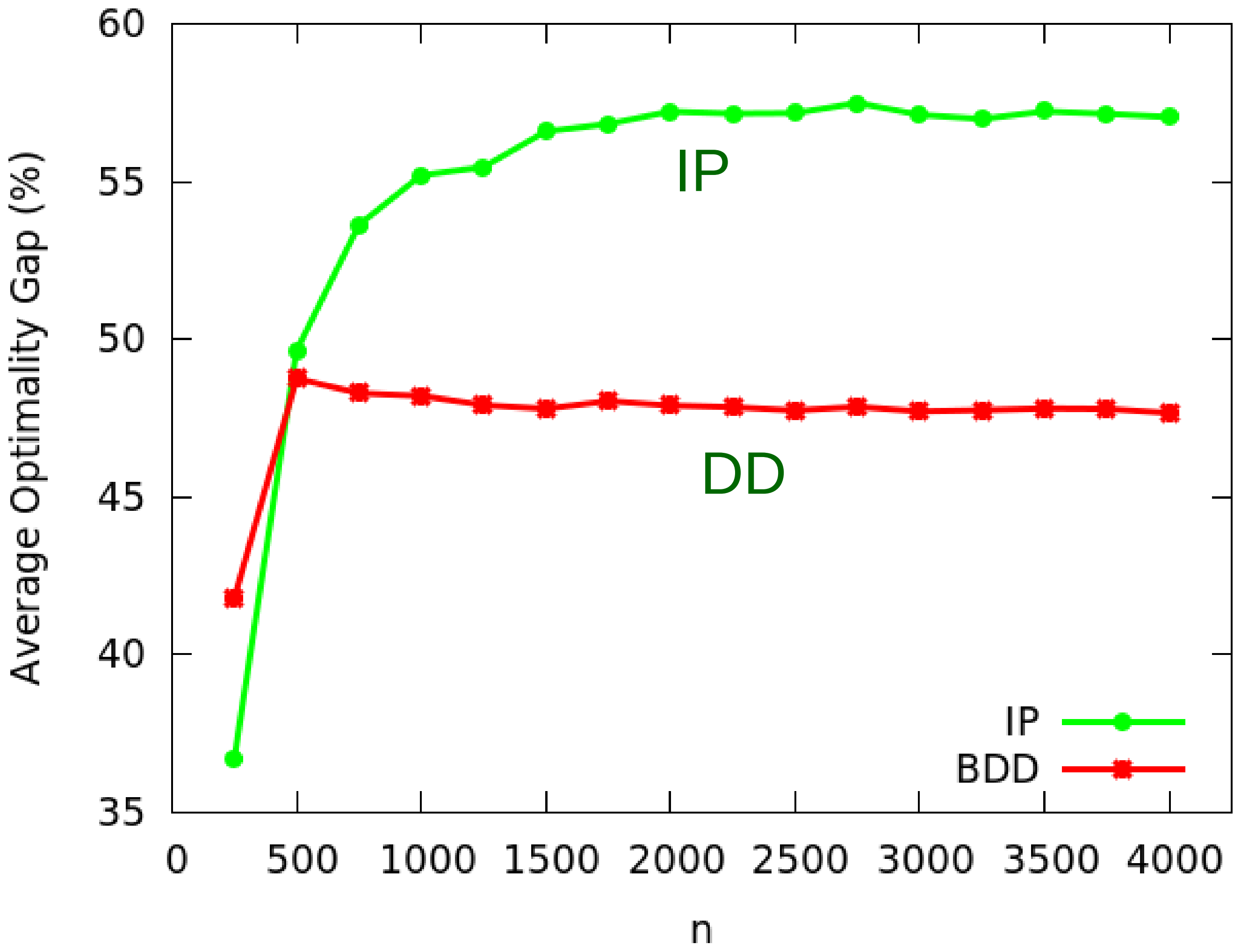
Restricted Decision Diagrams

- 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 unpromising nodes as diagram is constructed top-down

Bergman, Ciré, van Hoeve, Yunes (2014)

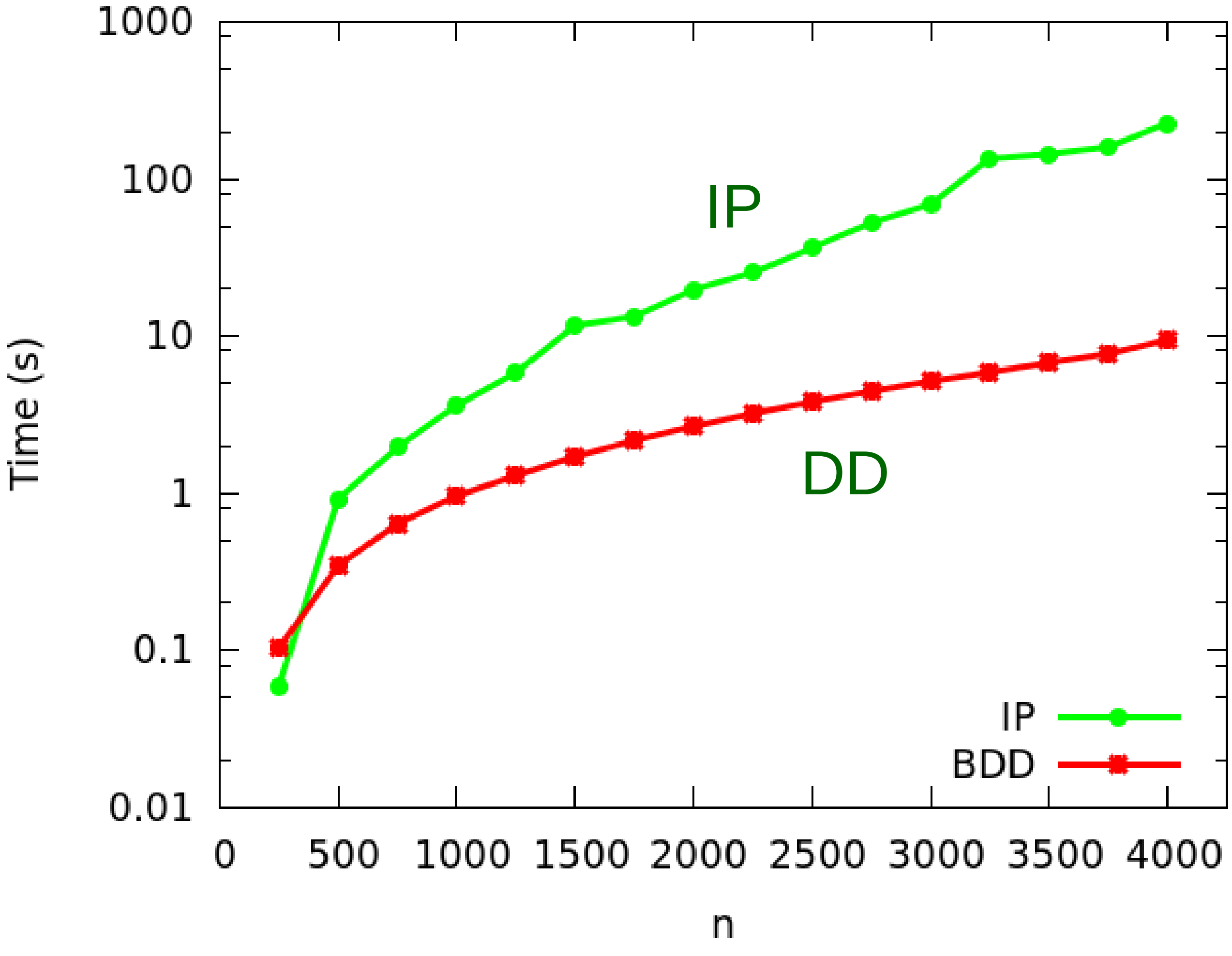
Optimality gap for **set covering**, *n* variables

Restricted DDs vs
Primal heuristic at root node of CPLEX

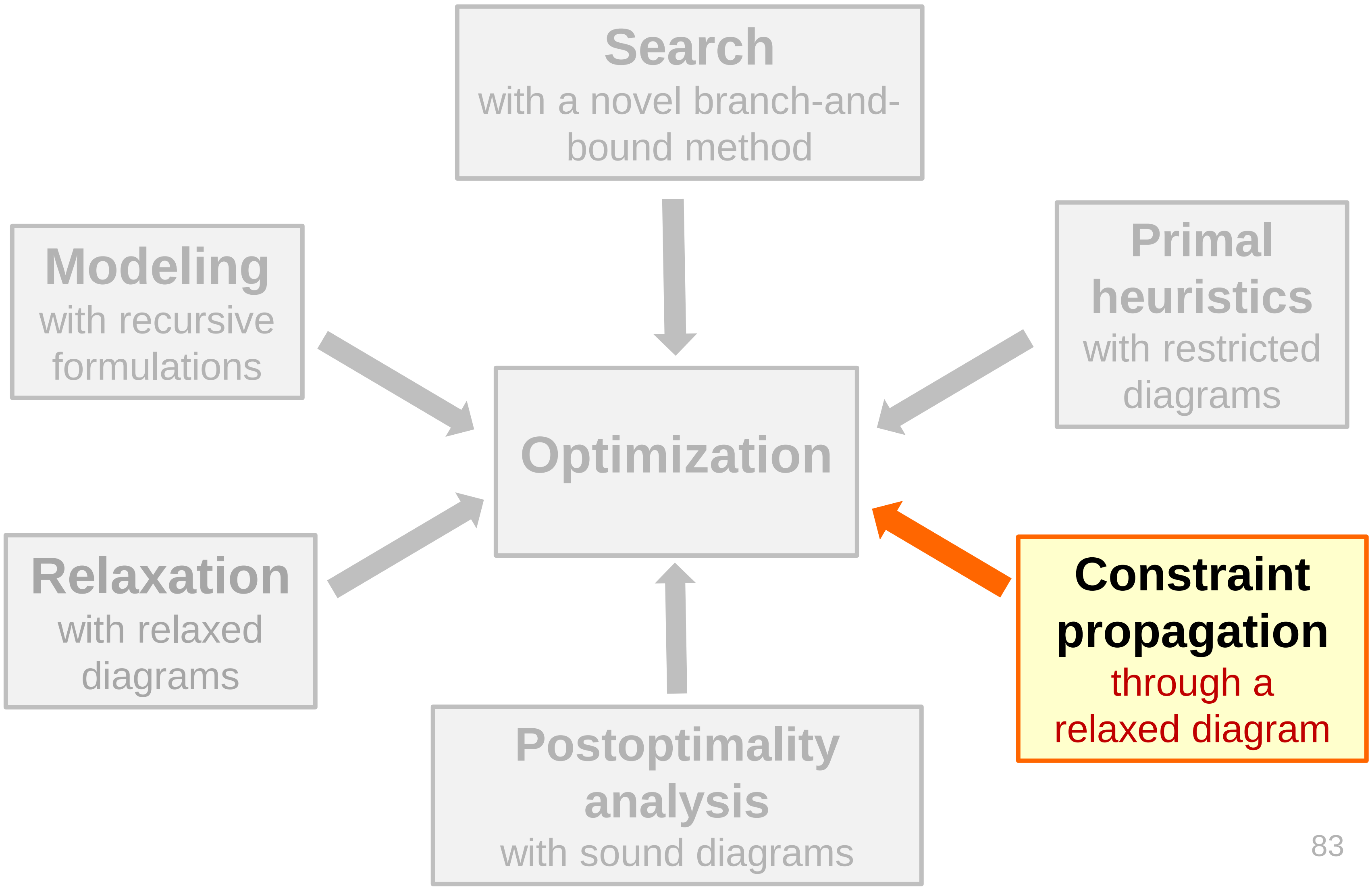


Computation time

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



Elements of Optimization



Constraint Propagation

- Original application: **graph coloring**
  - Use **node splitting** to create relaxed diagram.
  - **Propagate** through relaxed diagram by removing arcs that cannot be part of a feasible path.
  - In multiple alldiff problem (graph coloring), this reduced search tree from **1 million nodes to 1 node.**
  - Order of magnitude reduction in solution time.

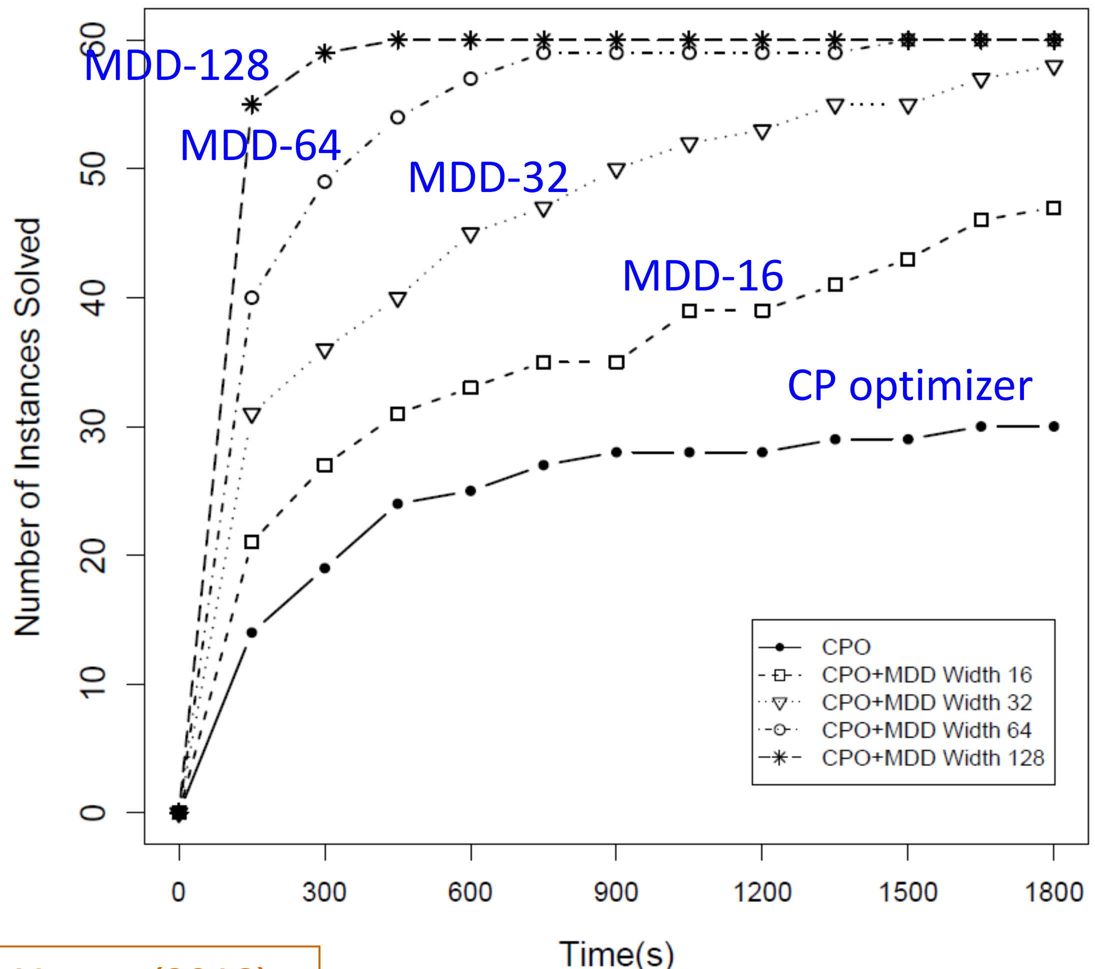
Andersen, Hadžić, JH, Tiedemann (2007)

Constraint Propagation

- Recent application: **TSP with time windows**.
- Decision diagram propagator becomes an **additional global constraint** in a constraint programming solver.
  - The CP solver conducts the search.
  - Substantial speedup
  - Closed 3 open problem instances in TSPLIB.

Ciré and van Hoeve (2013)

Effect of decision-diagram-based propagation in a constraint programming solver (ILOG CP Optimizer)



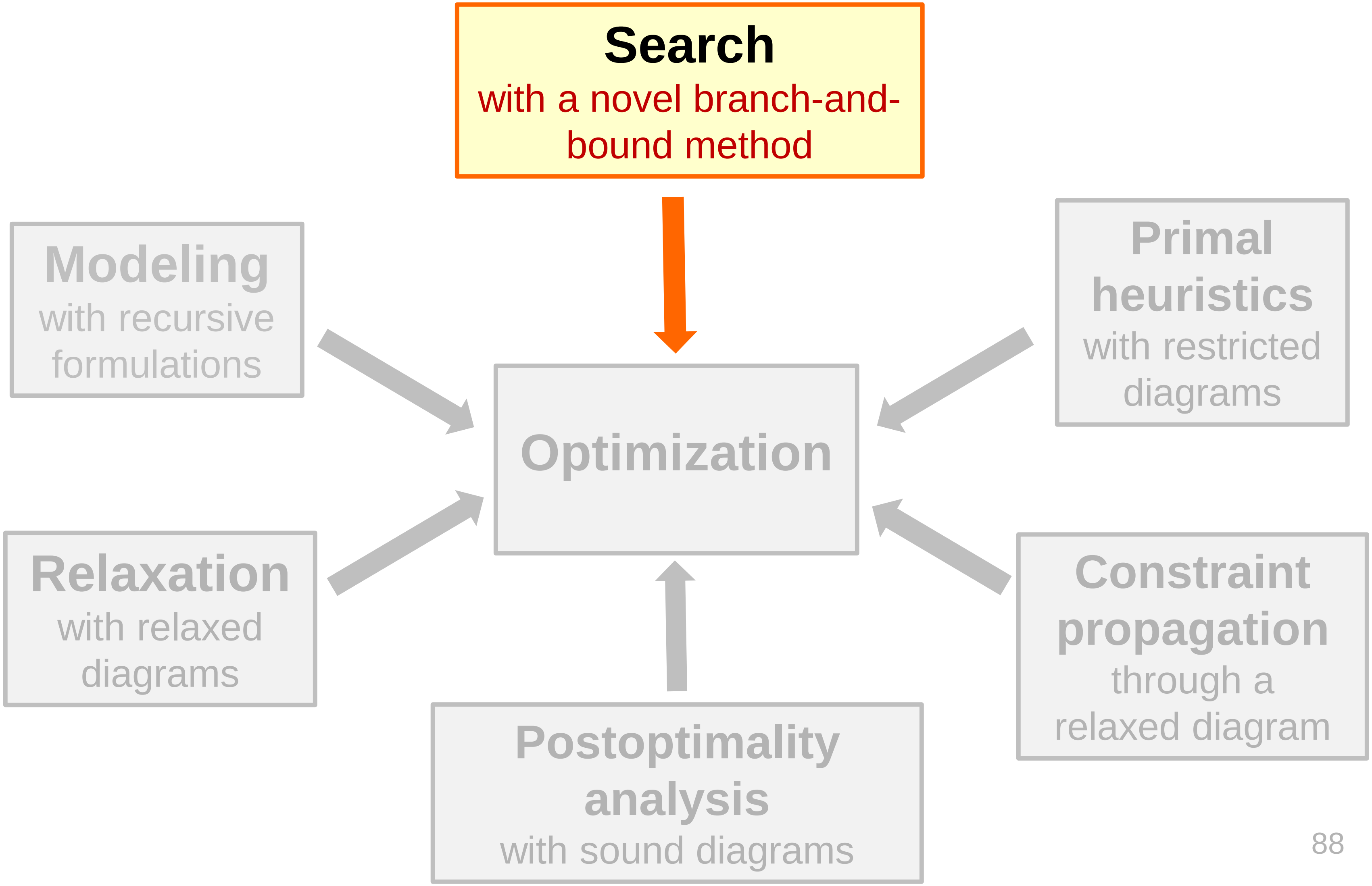
Ciré and van Hoeve (2013)

Three open instances solved

instance	vertices	bounds	CPO best	CPO time (s)	CPO+MDD, width 2048 best	CPO+MDD, width 2048 time (s)
br17.10	17	55	55	0.01	55	4.98
br17.12	17	55	55	0.01	55	4.56
ESC07	7	2125	2125	0.01	2125	0.07
ESC25	25	1681	1681	TL	1681	48.42
p43.1	43	28140	28205	TL	28140	287.57
p43.2	43	[28175, 28480]	28545	TL	**28480**	**279.18**
p43.3	43	[28366, 28835]	28930	TL	**28835**	**177.29**
p43.4	43	83005	83615	TL	83005	88.45
ry48p.1	48	[15220, 15805]	18209	TL	16561	TL
ry48p.2	48	[15524, 16666]	18649	TL	17680	TL
ry48p.3	48	[18156, 19894]	23268	TL	22311	TL
ry48p.4	48	[29967, 31446]	34502	TL	**31446**	**96.91**
ft53.1	53	[7438, 7531]	9716	TL	9216	TL
ft53.2	53	[7630, 8026]	11669	TL	11484	TL
ft53.3	53	[9473, 10262]	12343	TL	11937	TL
ft53.4	53	14425	16018	TL	14425	120.79

Ciré and van Hoeve (2013)

Elements of Optimization



Branching Algorithm

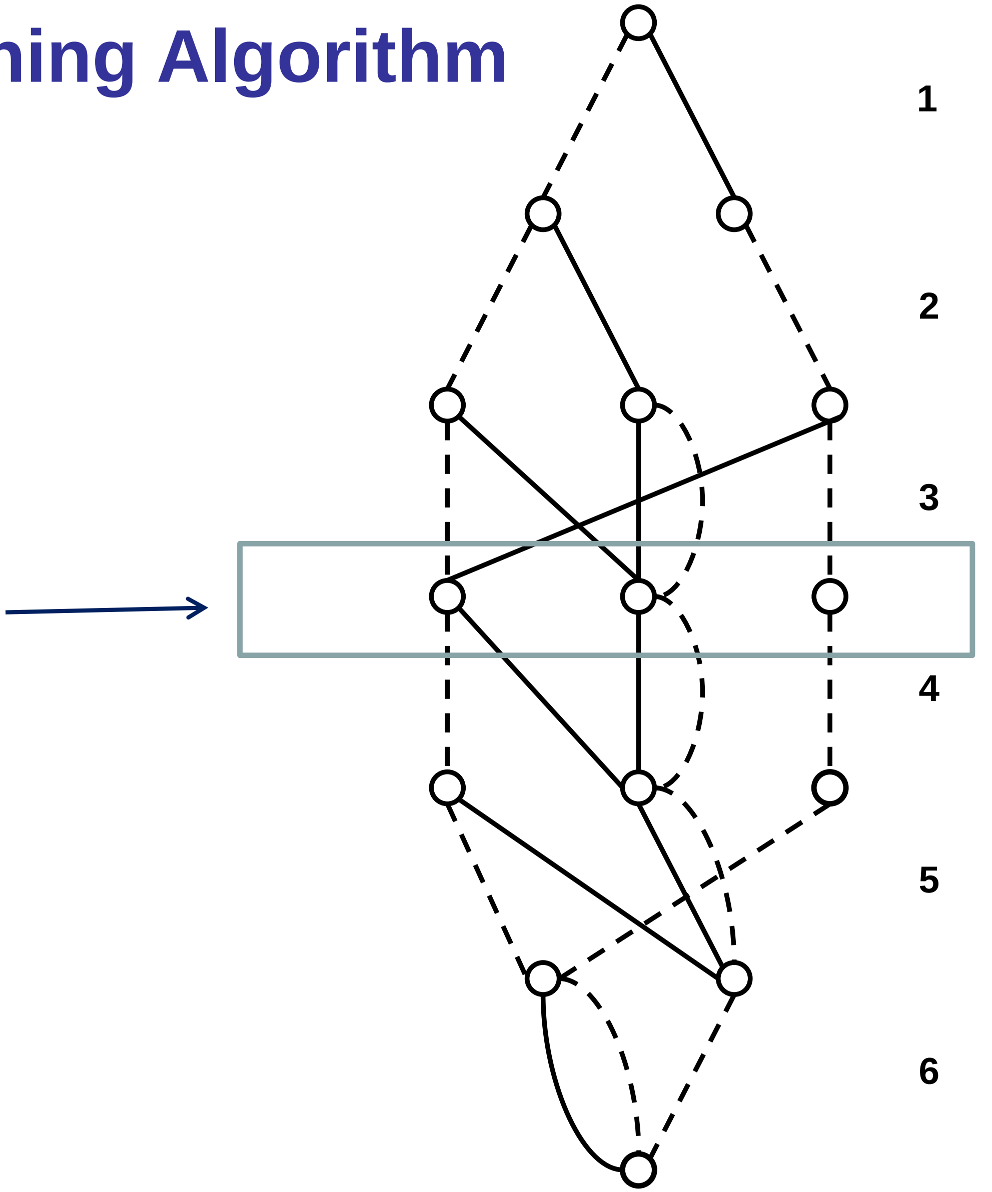
- Solve optimization problem using a novel **branch-and-bound** algorithm.
  - Branch on nodes in **last exact layer** of relaxed decision diagram.
    - …rather than branch on variables.
    - Create a new **relaxed DD rooted** at each branching node.
    - Prune search tree using bounds from relaxed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

Bergman, Ciré, van Hoeve, JH (2016)

Branching Algorithm

Branching in a relaxed decision diagram

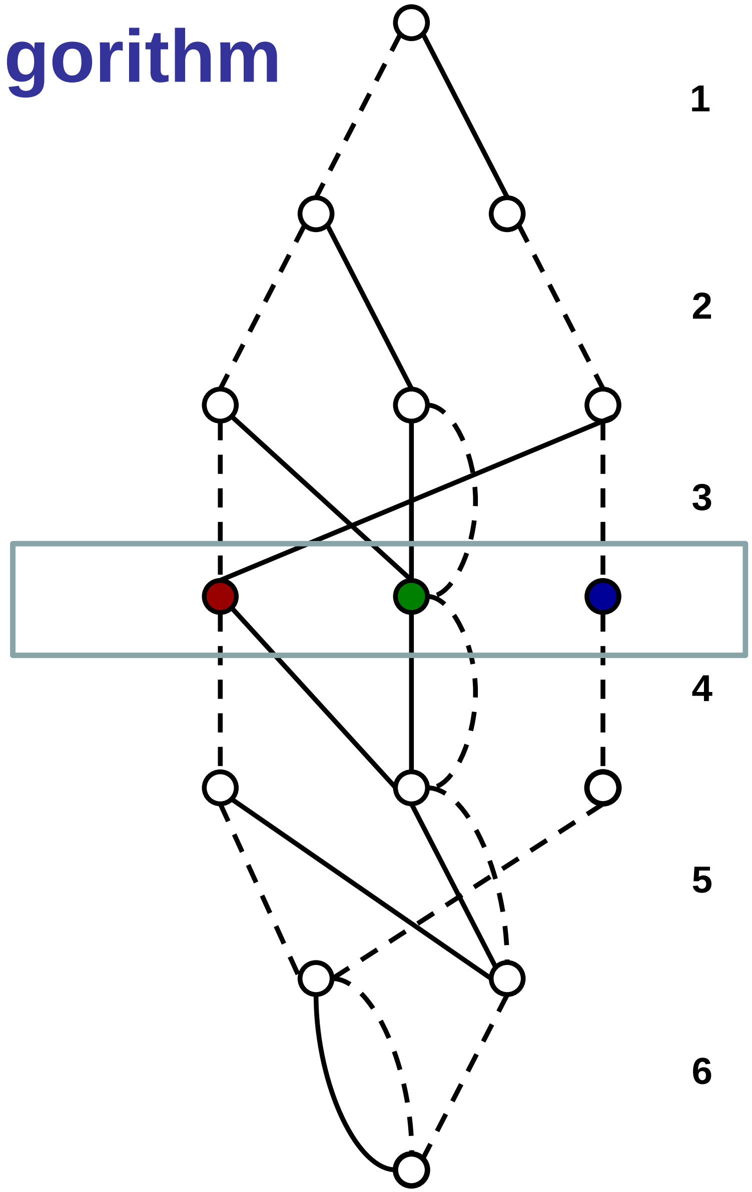
Diagram is exact down to here



Branching Algorithm

Branching in a relaxed
decision diagram

Branch on nodes
in this layer

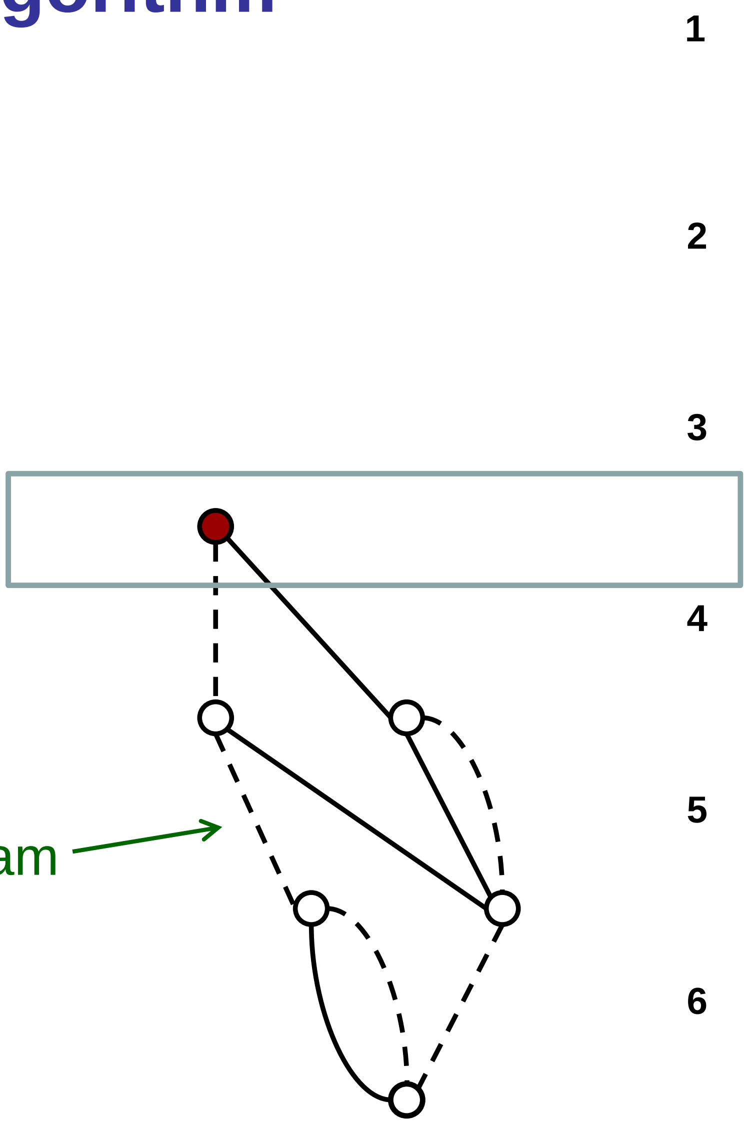


Branching Algorithm

Branching in a relaxed decision diagram

First branch

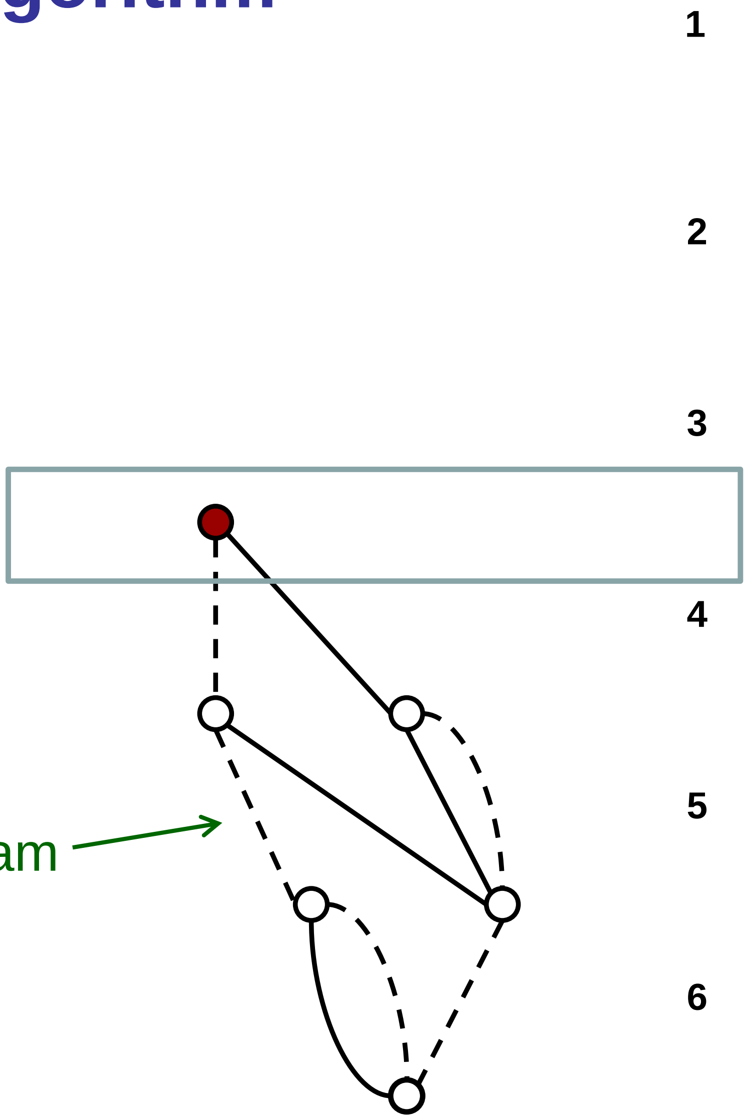
New relaxed decision diagram



Branching Algorithm

Branching in a relaxed
decision diagram

First branch



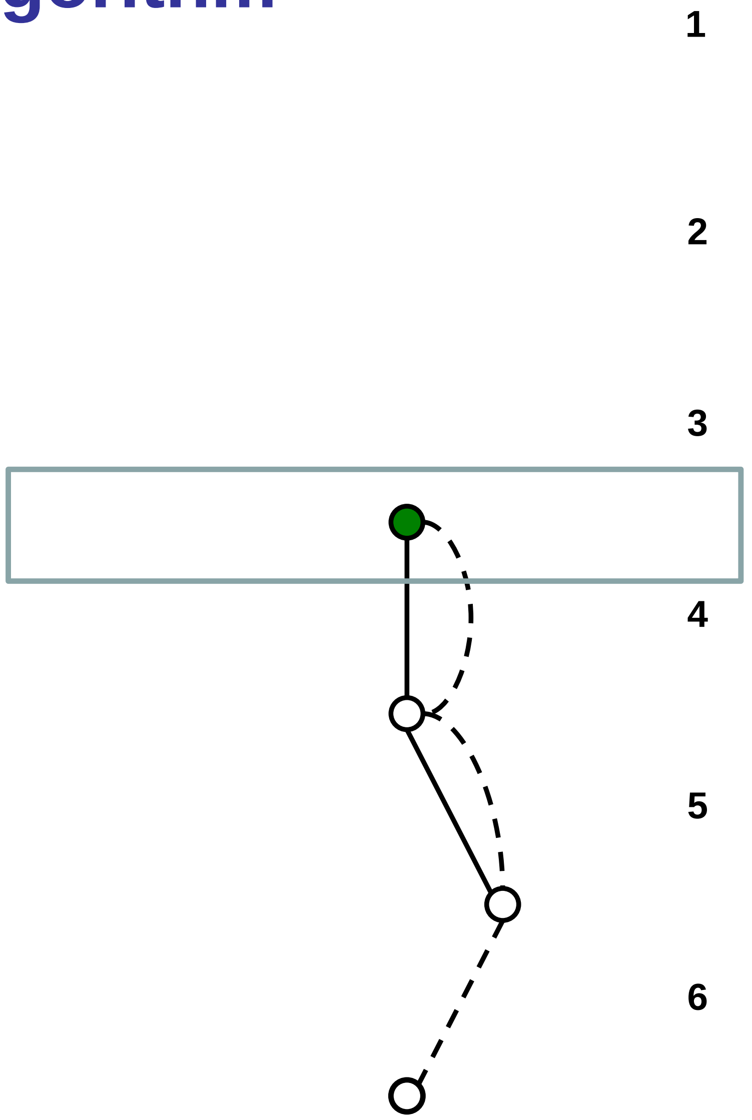
New relaxed decision diagram

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

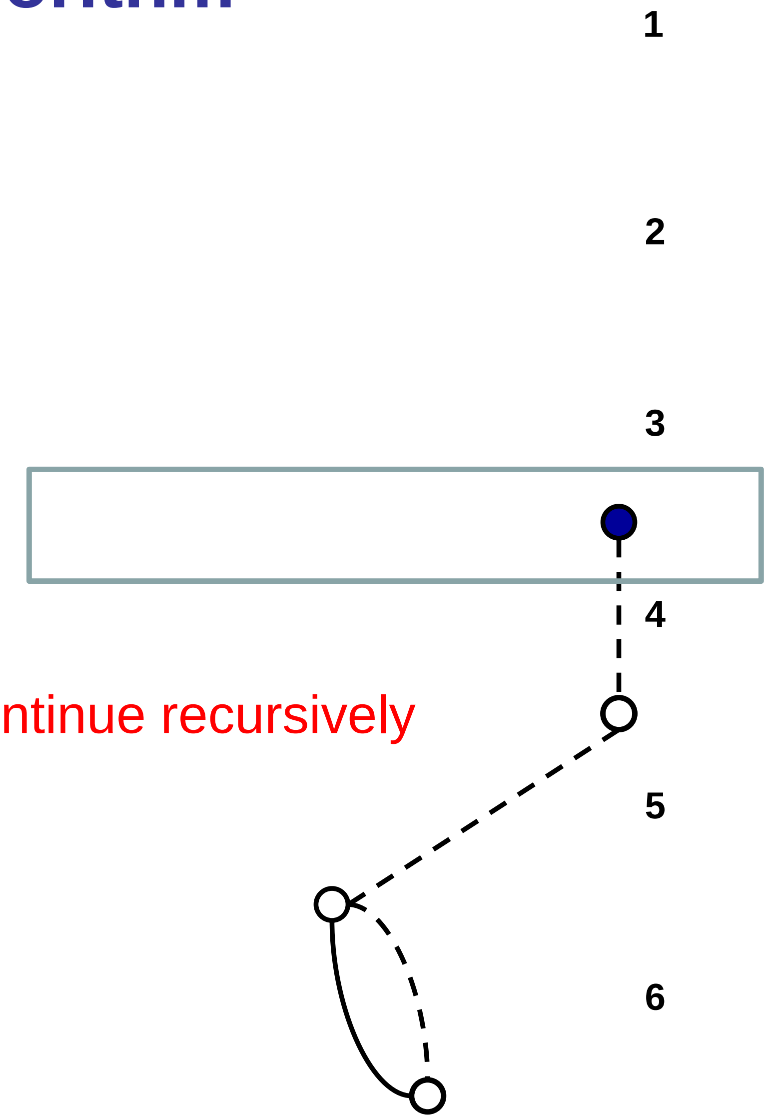
Second branch



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**).

Computational Performance

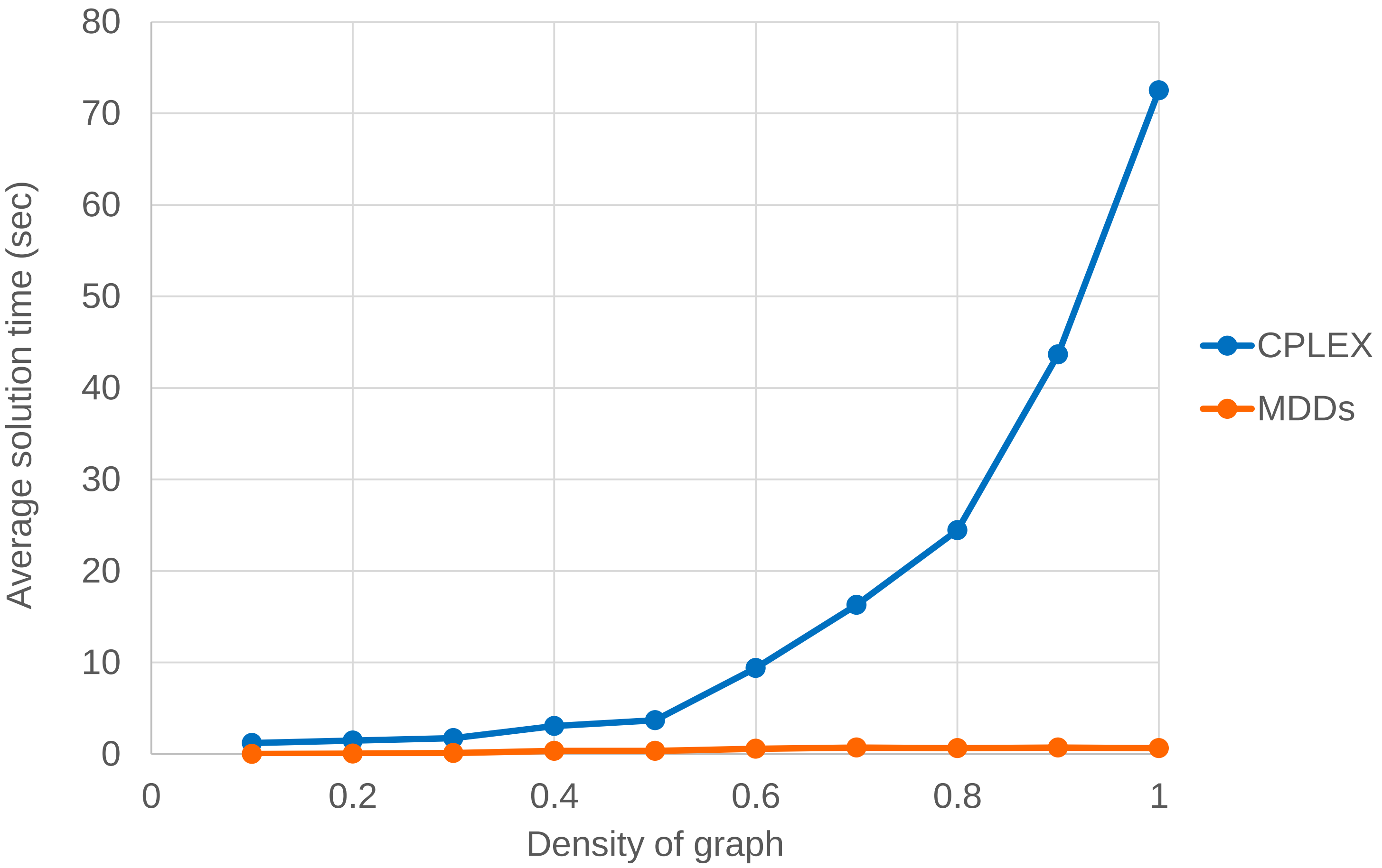
- **Max cut** problem on a graph.
  - Partition nodes into 2 sets so as to maximize total weight of connecting edges.
  - **State** = marginal benefit of placing each remaining vertex on left side of cut.
  - **State merger** =
    - Componentwise min if all components ≥ 0 or all ≤ 0; 0 otherwise
  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

Computational performance

Max cut
on a graph

Avg. solution time
vs
graph density

30 vertices



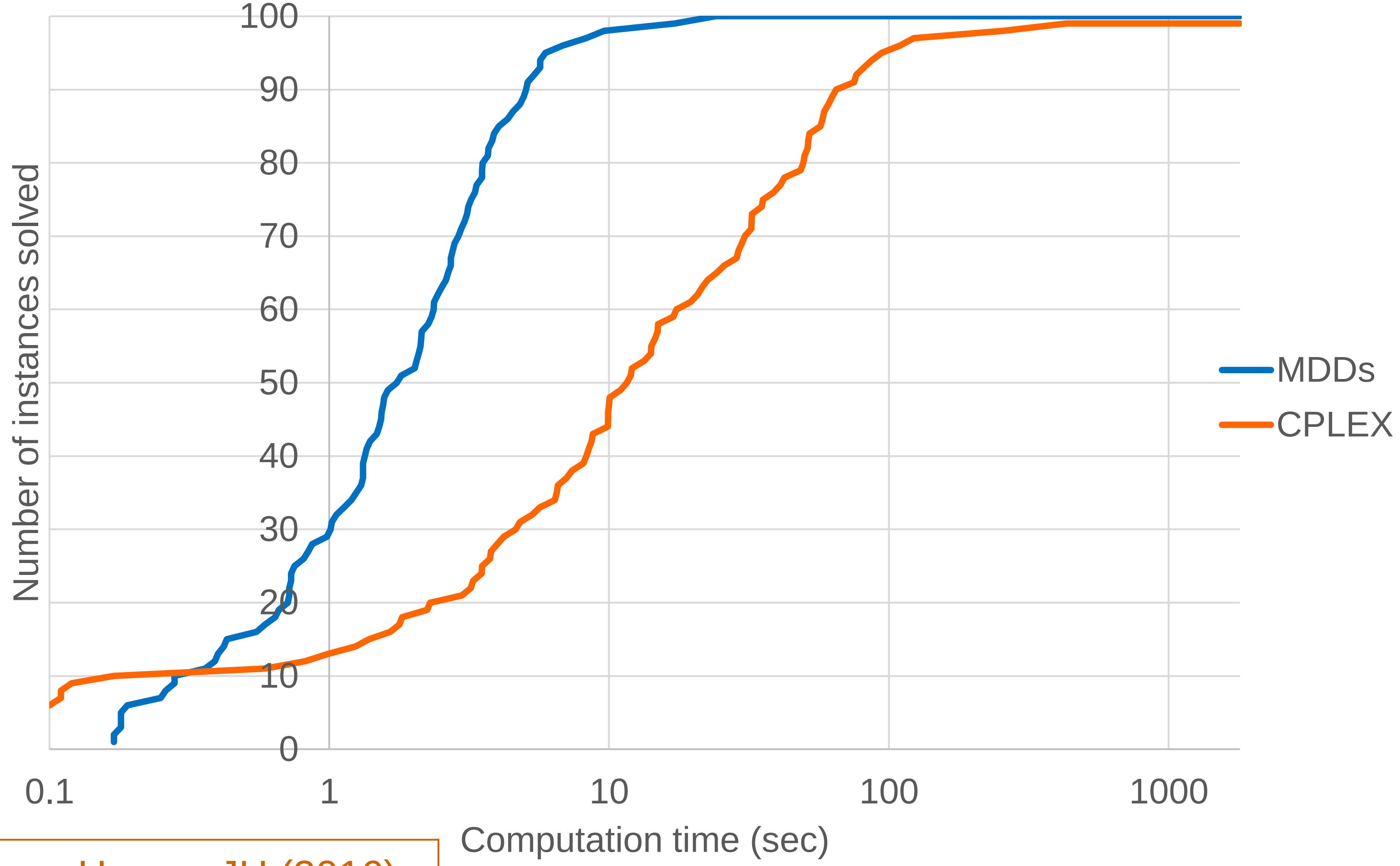
Bergman, Ciré, van Hoeve, JH (2016)

Computational performance

Max 2-SAT

Performance profile

30 variables



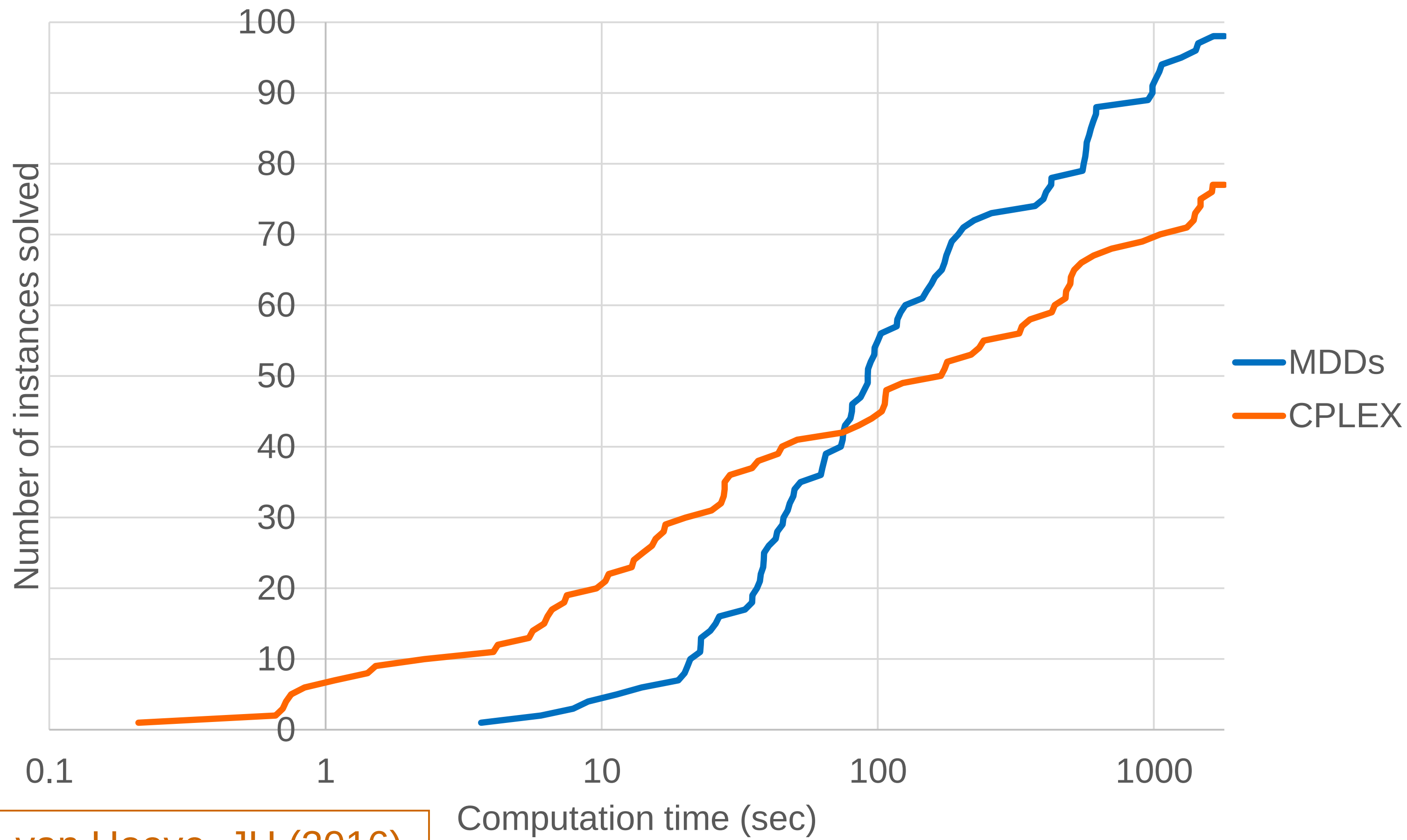
Bergman, Ciré, van Hoeve, JH (2016)

Computational performance

Max 2-SAT

Performance profile

40 variables

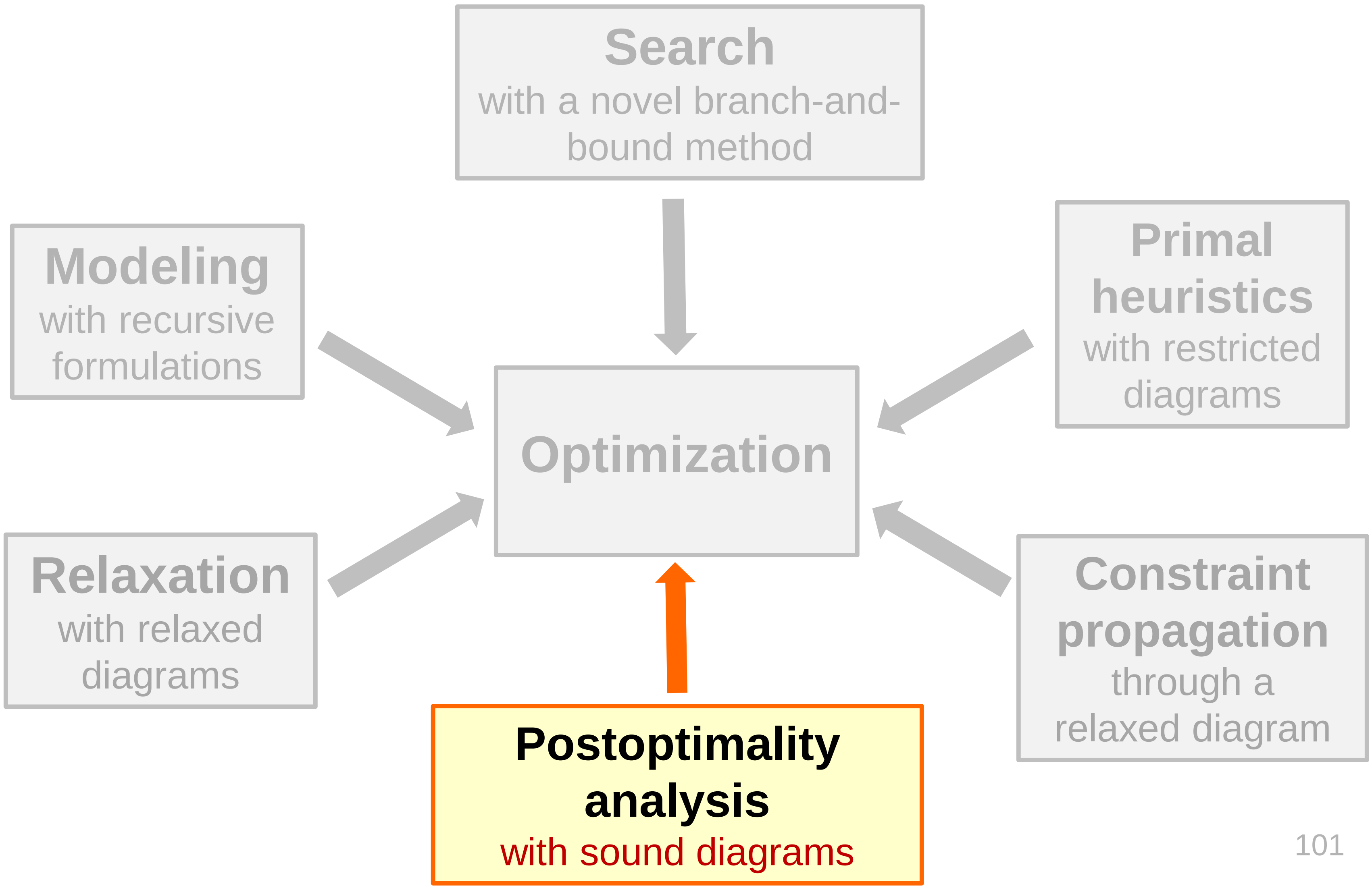


Bergman, Ciré, van Hoeve, JH (2016)

Computational performance

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

Elements of Optimization



Postoptimality Analysis

- Decision diagrams open the door to more **comprehensive** postoptimality analysis.
  - DDs can compactly represent **all near-optimal solutions** (within Δ of optimum).
  - They can be **efficiently queried** with what-if questions.

Serra and JH (2018)

Postoptimality Analysis

- Decision diagrams open the door to more **comprehensive** postoptimality analysis.
  - DDs can compactly represent **all near-optimal solutions** (within Δ of optimum).
  - They can be **efficiently queried** with what-if questions.
- Basic philosophy
  - Solution = conversion from an **opaque** data structure...
    - A constraint set
  - …to a **transparent** data structure.
    - A decision diagram

Serra and JH (2018)

Postoptimality Analysis

- **Sound** DDs can store solutions more compactly.
  - Sound = some **bad** solutions (feasible and infeasible) are included
    - i.e., solutions that are not within Δ of optimum.
    - These solutions are easily screened out.
    - No effect whatever on most queries.
  - Paradoxically, this can result in a **smaller** DD.

Serra and JH (2018)

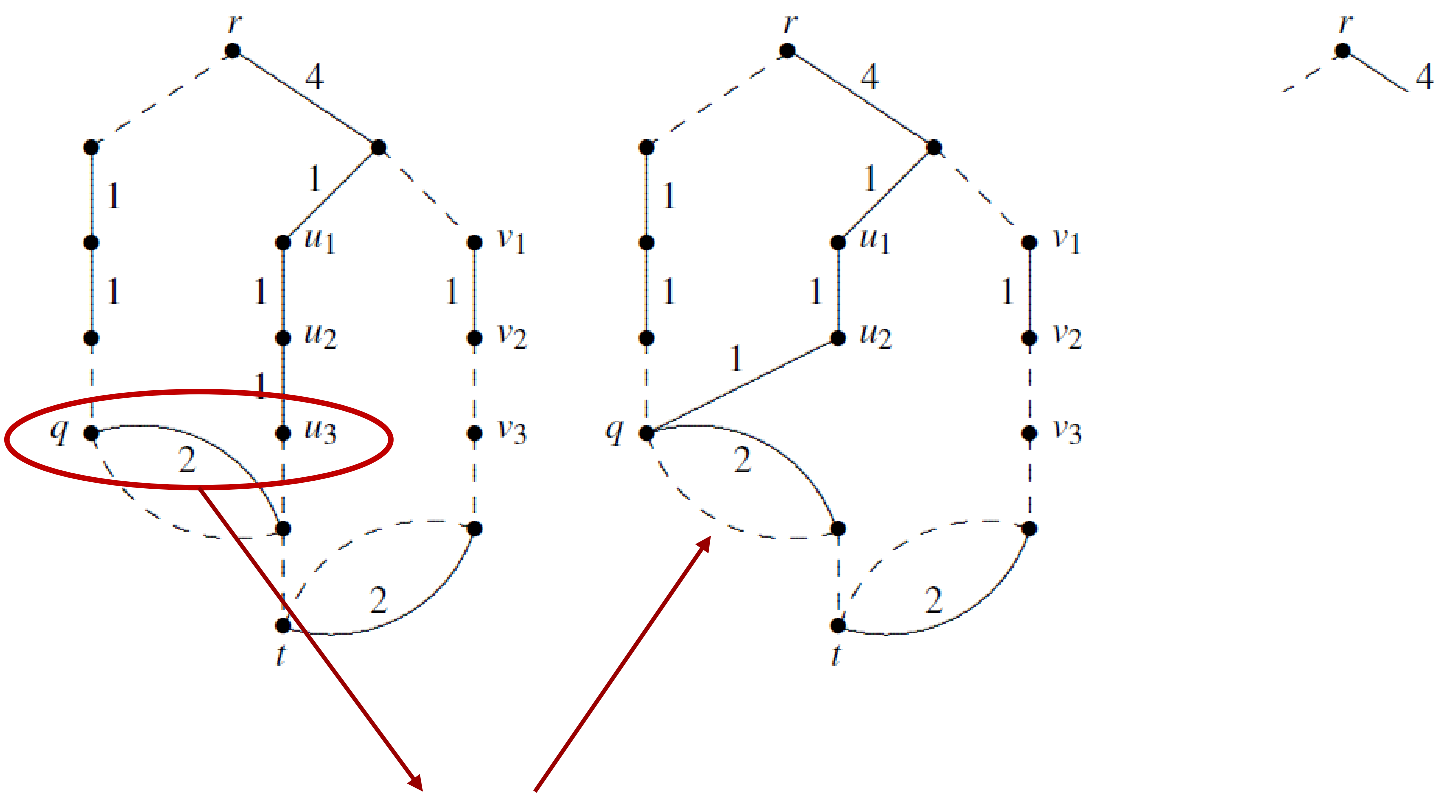
Postoptimality Analysis

- **Sound** DDs can store solutions more compactly.
  - Sound = some **bad** solutions (feasible and infeasible) are included
    - i.e., solutions that are not within Δ of optimum.
    - These solutions are easily screened out.
    - No effect whatever on most queries.
  - Paradoxically, this can result in a **smaller** DD.

Theorem. Repeated application of a certain node merger operation (in any order) yields a **sound reduced** DD – i.e., a DD of **minimum size**.

…even though the sound reduced DD is not unique!

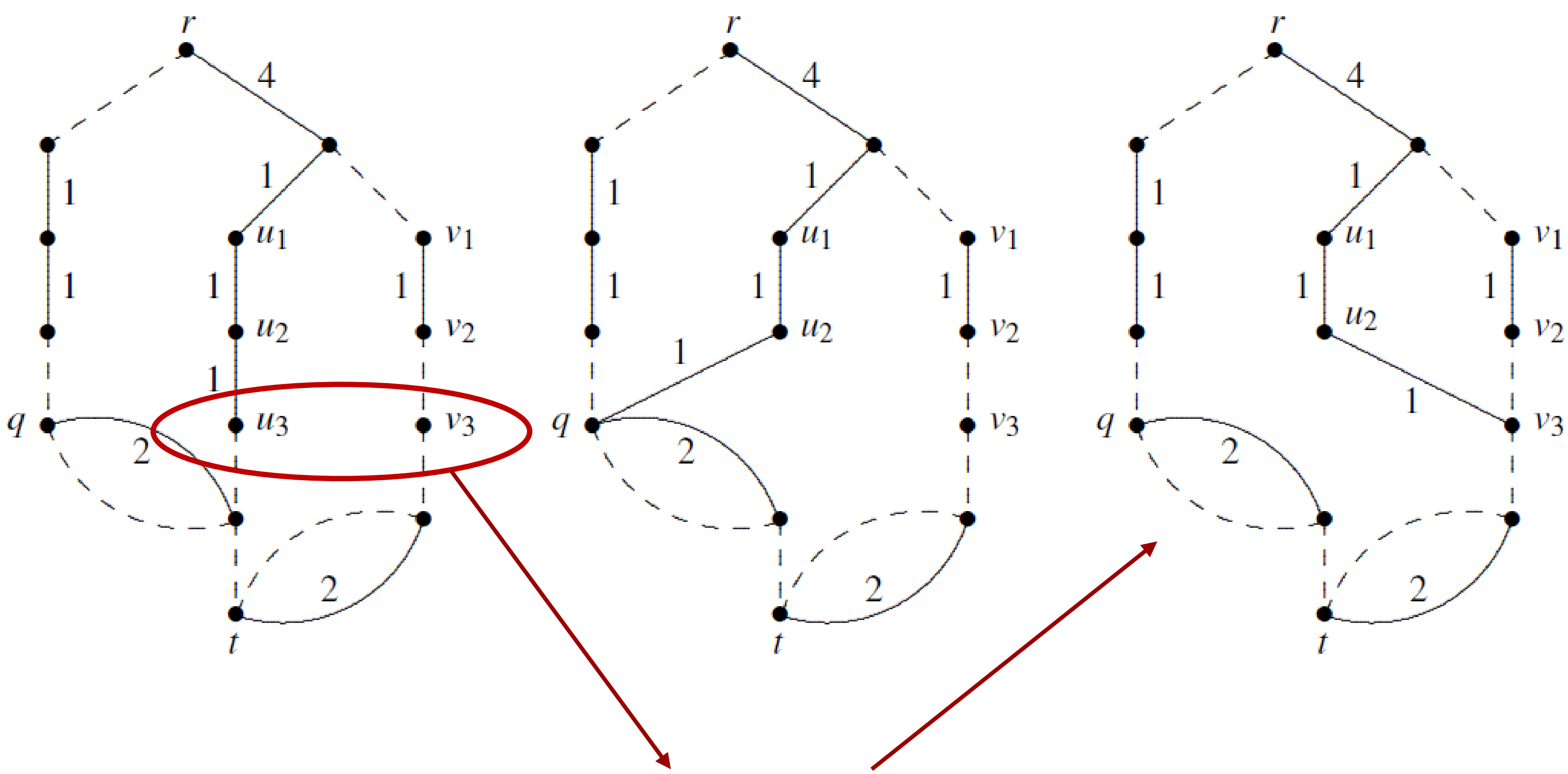
Postoptimality Analysis



Merger yields this
sound-reduced DD

Serra and JH (2018)

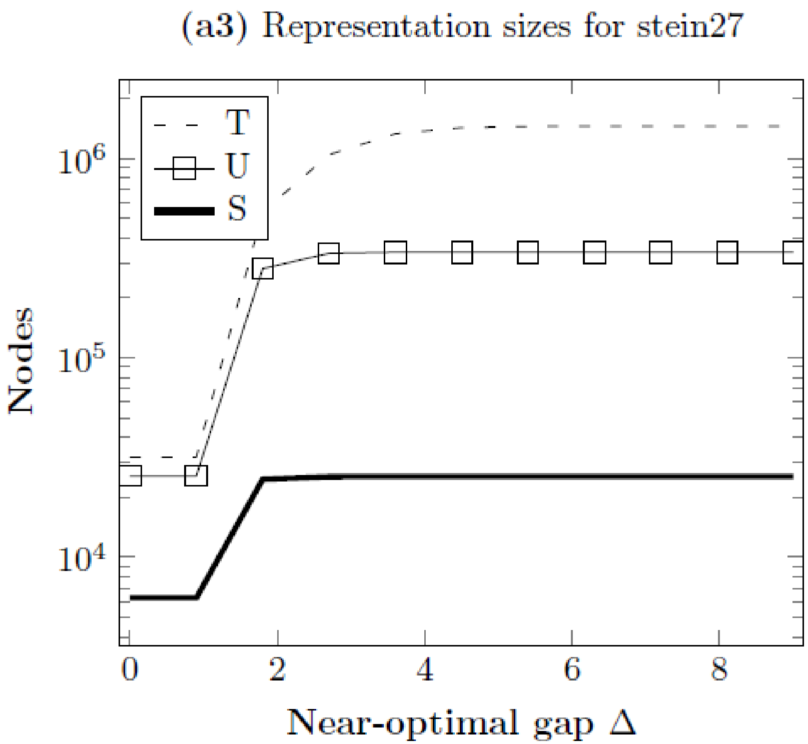
Postoptimality Analysis

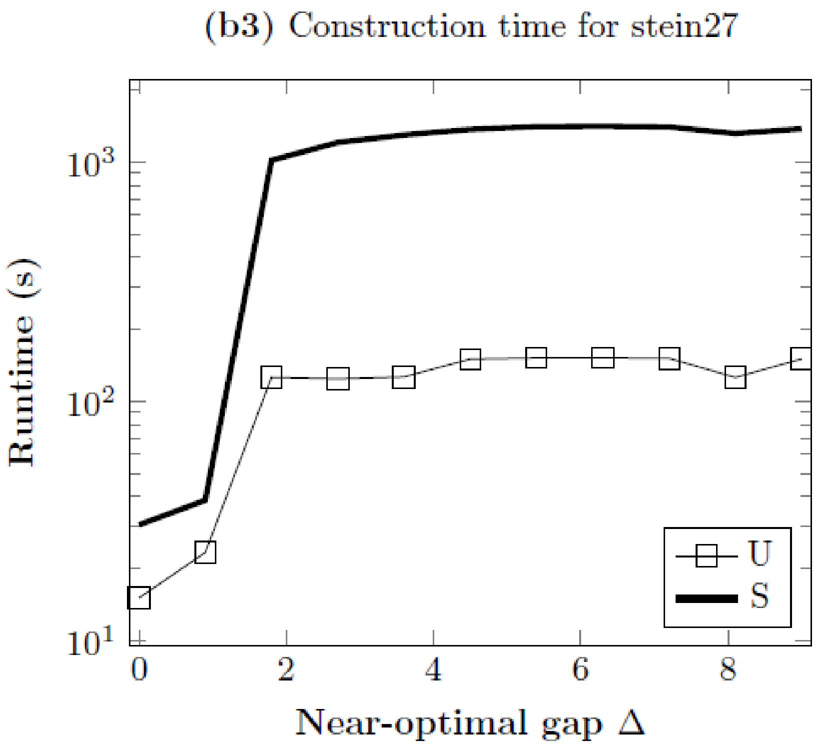


Merger yields this distinct
sound-reduced DD

Serra and JH (2018)

Postoptimality Analysis for IP

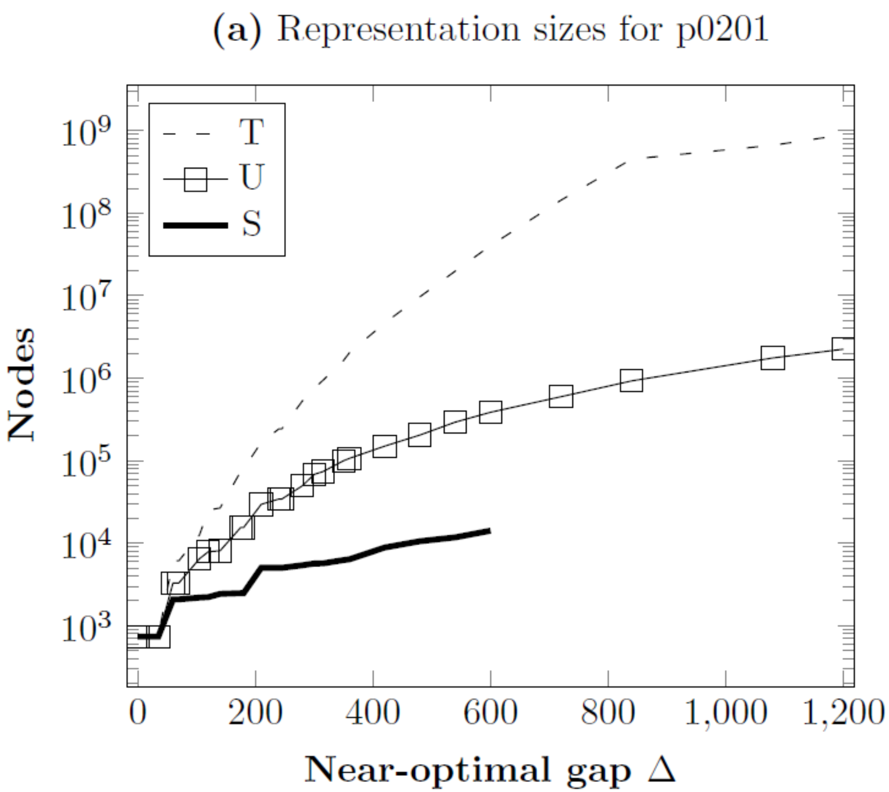


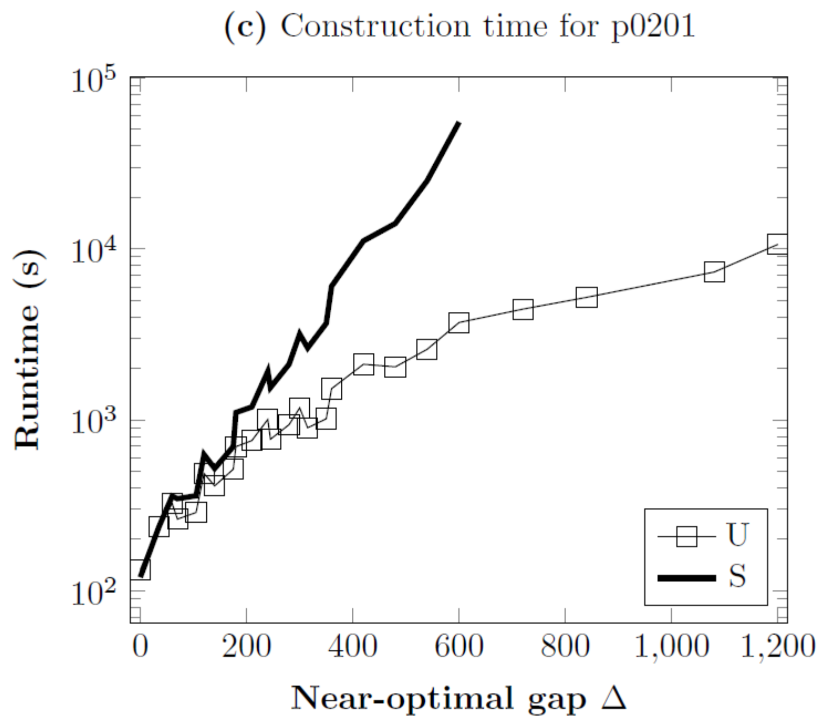


Includes time to find alternate optimal solutions, given optimal value from IP solver

T = tree representation
U = reduced DD
S = sound-reduced DD

Postoptimality Analysis for IP





Includes time to find alternate optimal solutions, given optimal value from IP solver

T = tree representation
U = reduced DD
S = sound-reduced DD

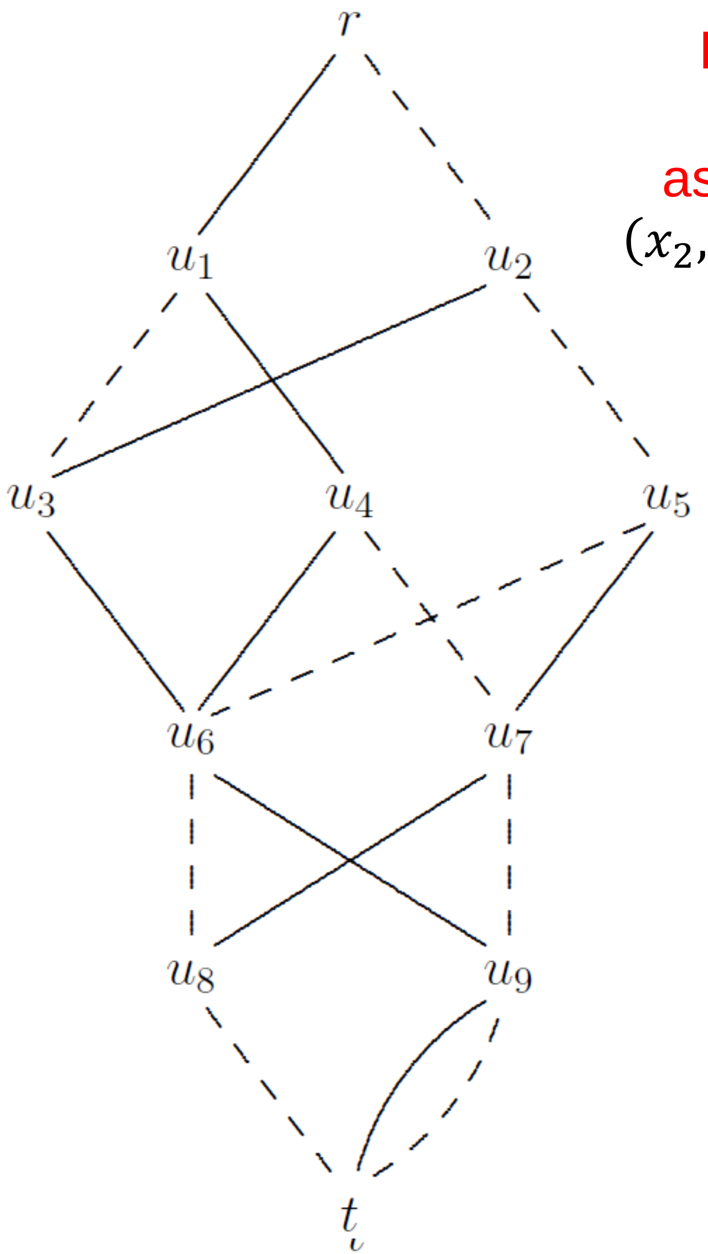
Outline

- Decision diagram **basics**
- Optimization with **exact** decision diagrams
- Providing the basic **elements of optimization**
  - **Modeling, relaxation, primal heuristics, constraint propagation, search, postoptimality**
- Research **frontiers**
  - Separation
  - Radical reduction of state space in DP
  - Nonlinear optimization
  - Nonserial recursion
- References

Separation Problem for BDDs

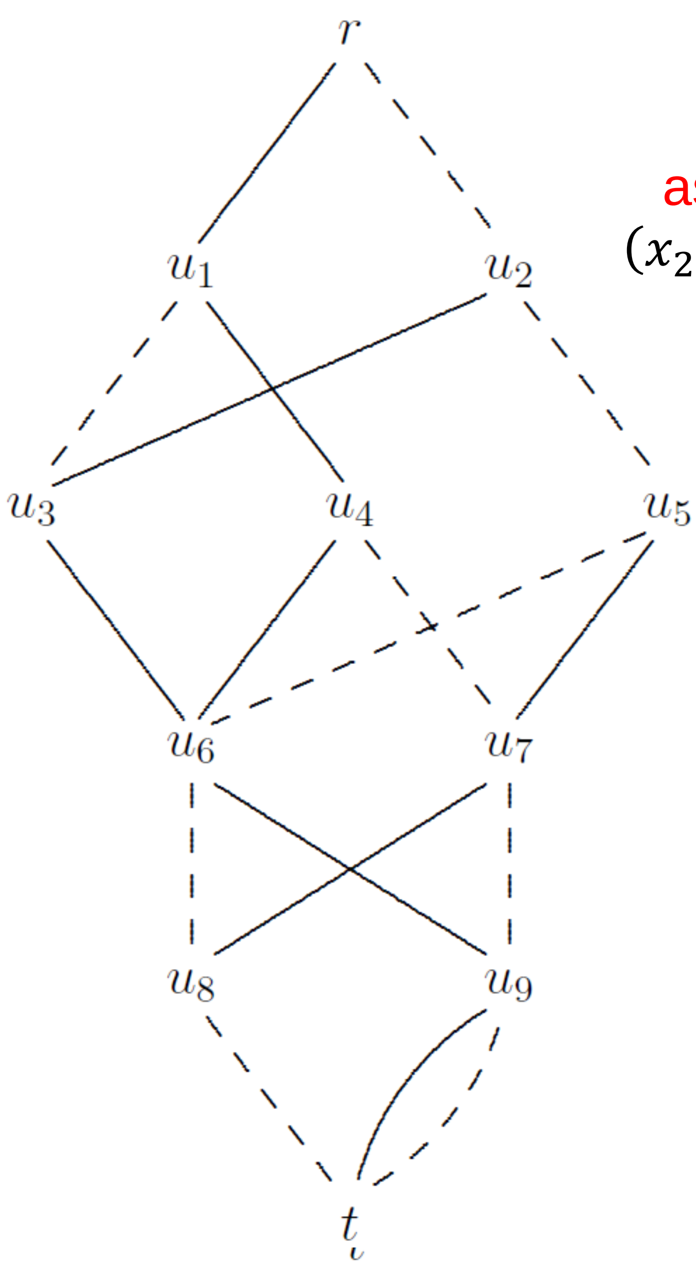
- The separation problem arises when a **new constraint** is added.
  - As in **Benders decomposition**.
  - We wish to separate solutions that **violate** the new constraint.
- Example: exclude a given partial assignment $x_i = \bar{x}_i$ for $i \in I$.
  - That is, remove all paths in which $x_i = \bar{x}_i$ for $i \in I$.
- Example…

Original DD



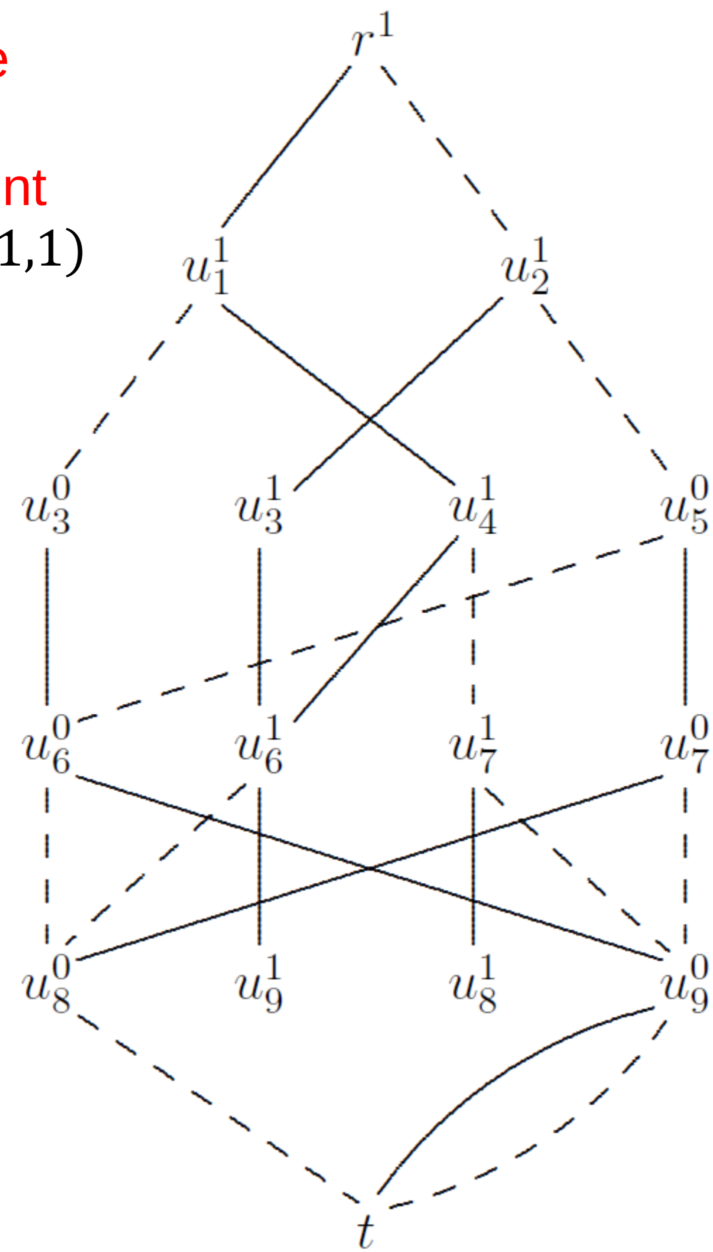
Remove partial assignment $(x_2, x_4) = (1,1)$

Original DD



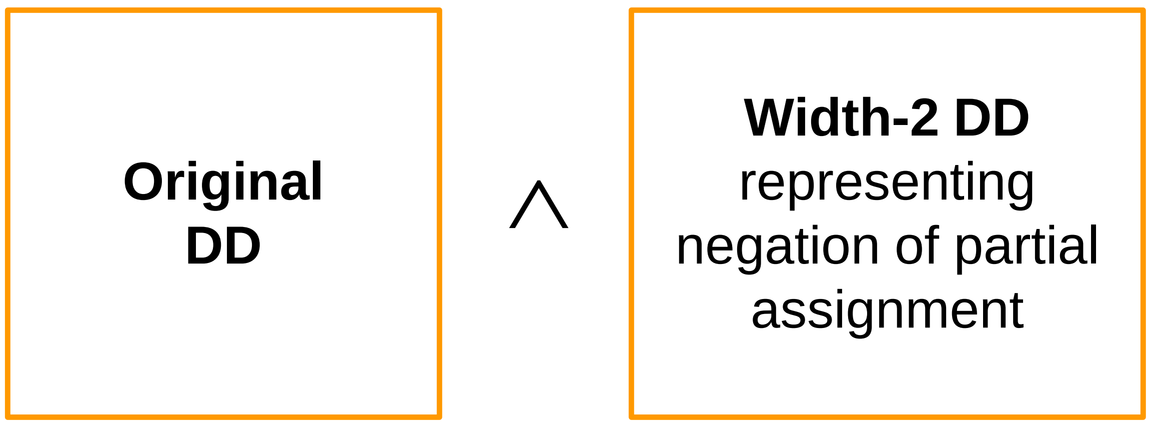
Remove partial assignment $(x_2, x_4) = (1,1)$

Separating DD



Separation Algorithm

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



  - There are efficient algorithms for this.

Size of Separating Diagram

Theorem. The separating DD is at most twice as large as the original DD.

Theorem. In the worst case, the separating DD can grow exponentially with the number of constraints separated.

Ciré and JH (2014)

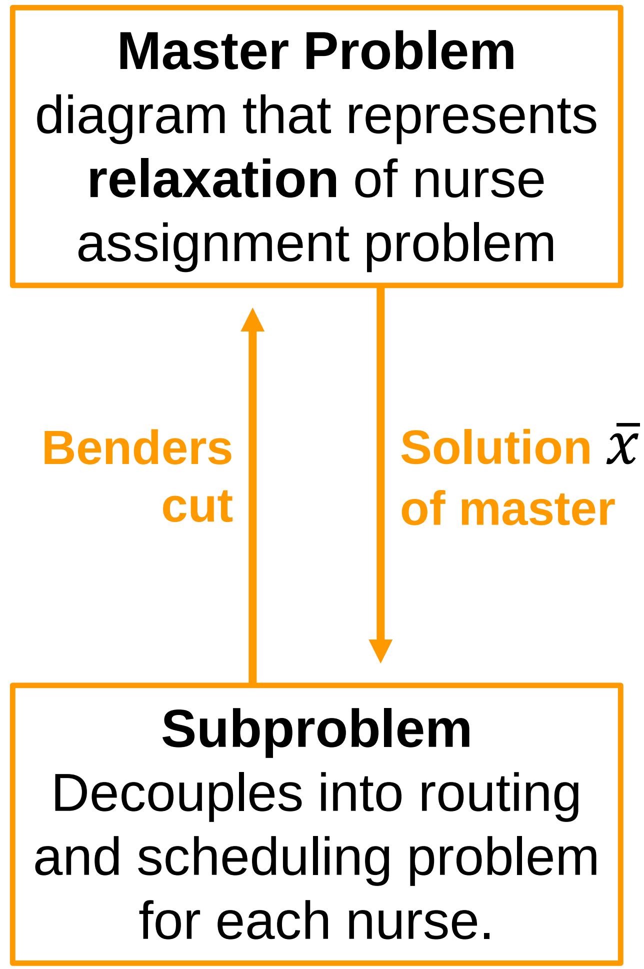
Empirical Growth

- How fast does the separating DD grow in a realistic optimization algorithm?
  - We will look at a **logic-based Benders** algorithm
  - …for the **home health care delivery problem**
    - Assign patients to health care aides.
    - Route aides to assigned patients.

Ciré and JH (2014)

Empirical Growth

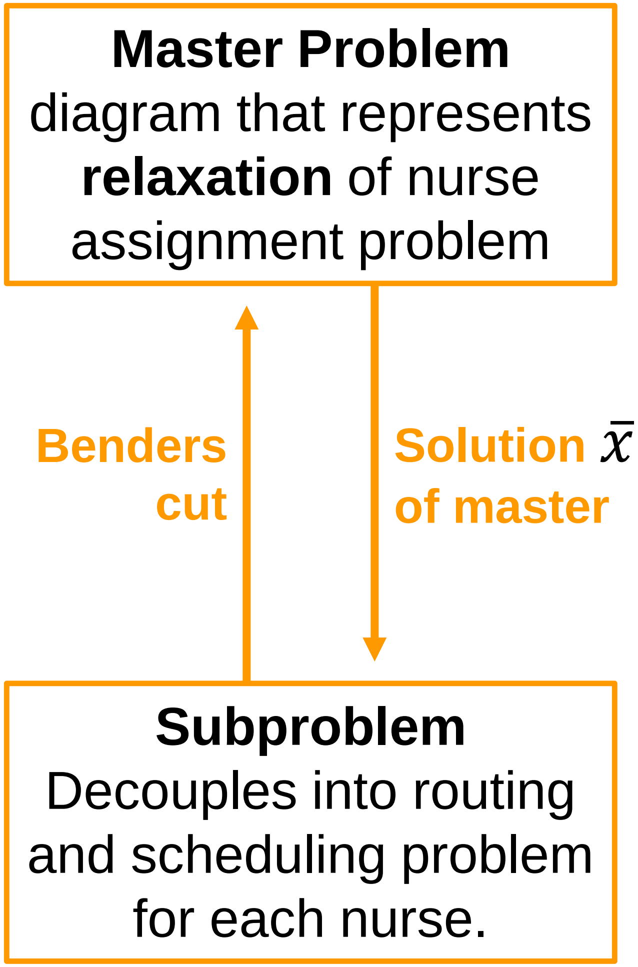
- Solve with **logic-based Benders decomposition**.
  - Assignment problem in master.
  - Subproblem generates Benders cuts when there is no feasible schedule.
  - Each cut excludes a **partial assignment** of aides to patients.
  - Cut is based on **inference dual** of subproblem.



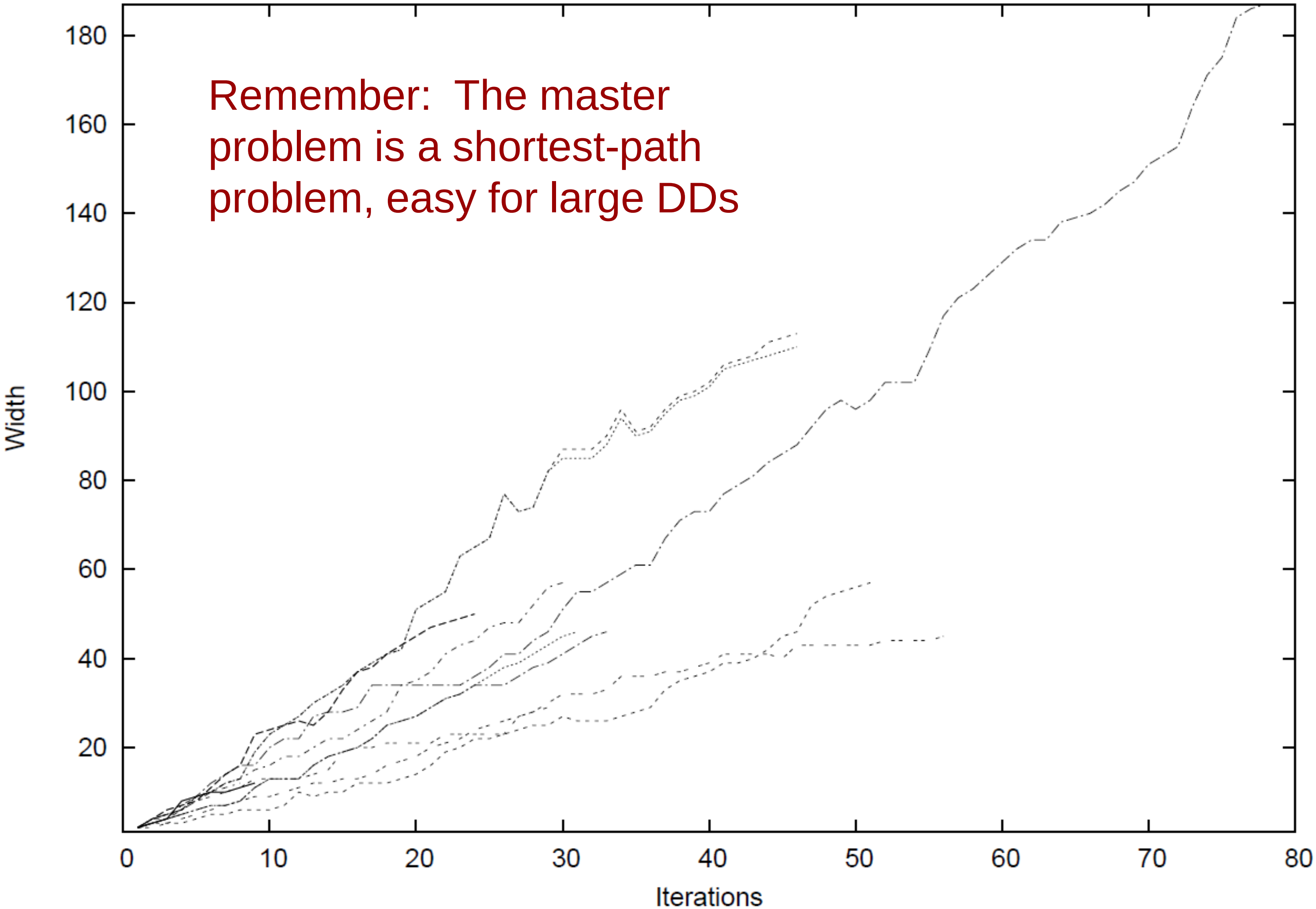
JH (2000),
JH & Ottosson (2003)

Empirical Growth

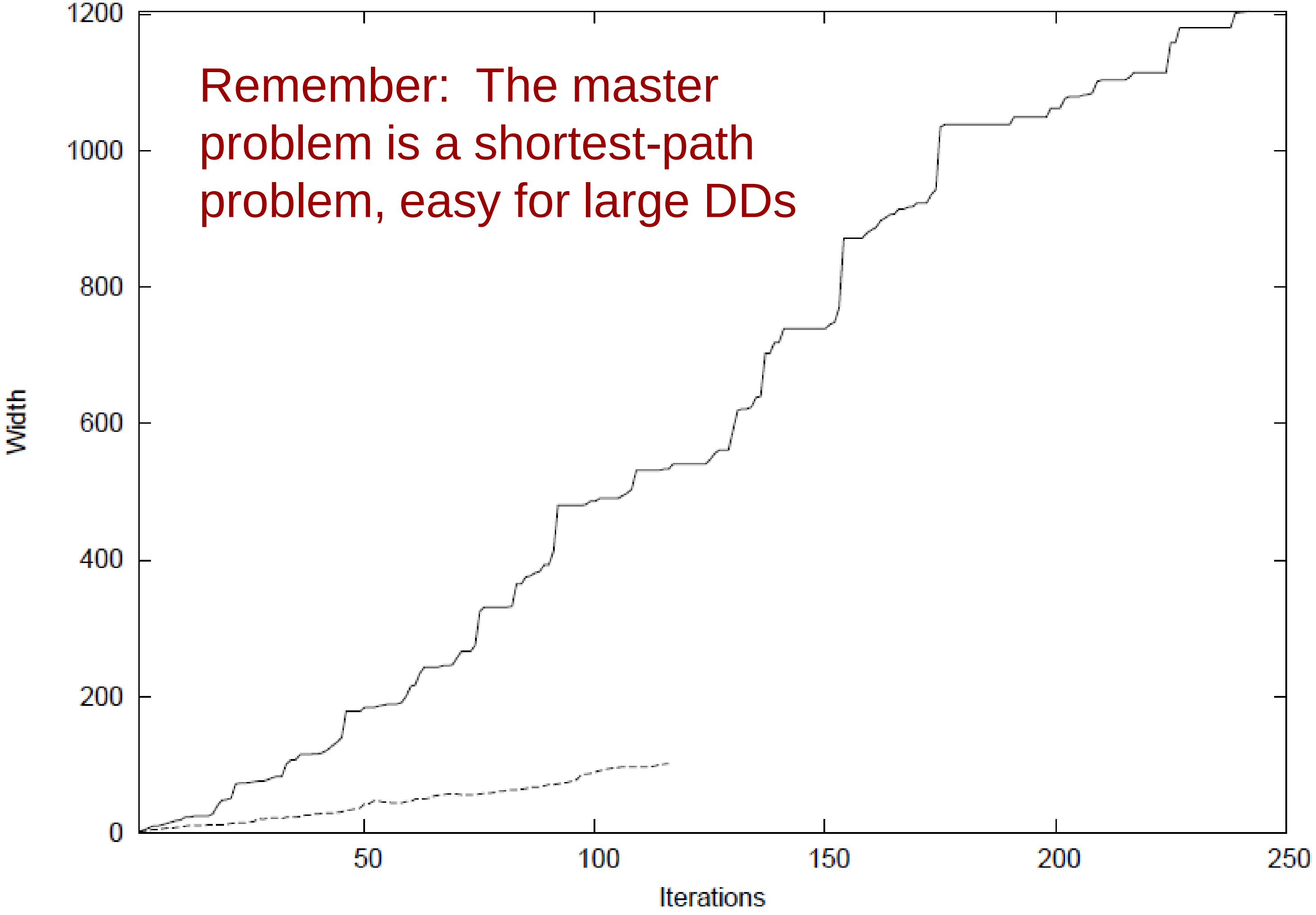
- Solve with **logic-based Benders decomposition**.
  - Assignment problem in master.
  - Subproblem generates Benders cuts when there is no feasible schedule.
  - Each cut excludes a **partial assignment** of aides to patients.
  - Cut is based on **inference dual** of subproblem.
- How fast does the DD grow as cuts are added?



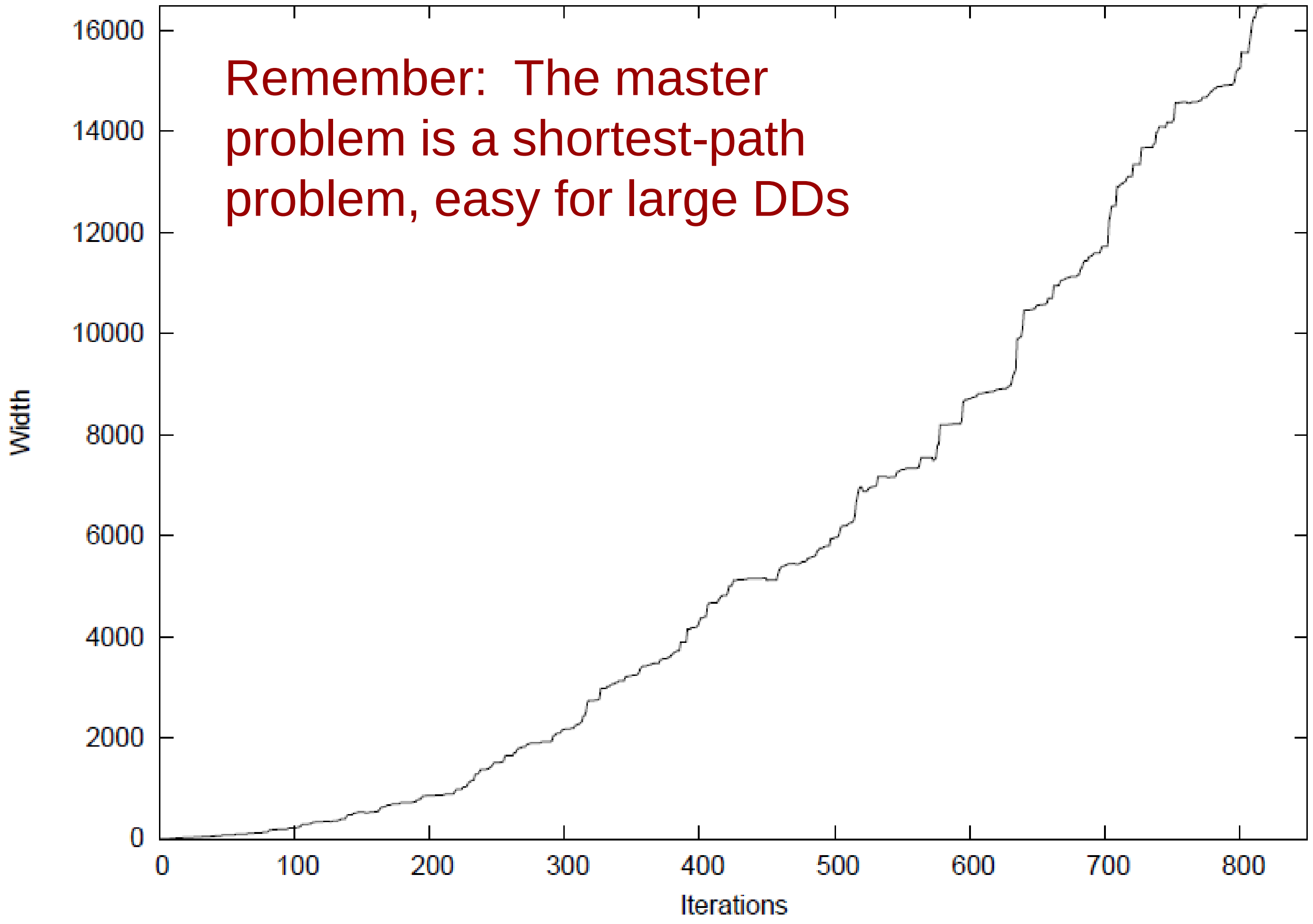
Growth of separating DD for all but 3 instances



Growth of separating DD for 2 harder instances



Growth of separating DD for hardest instance



Outline

- Decision diagram **basics**
- Optimization with **exact** decision diagrams
- Providing the basic **elements of optimization**
  - **Modeling, relaxation, primal heuristics, constraint propagation, search, postoptimality**
- Research **frontiers**
  - Separation
  - Radical reduction of state space in DP
  - Nonlinear optimization
  - Nonserial recursion
- References

Radical Reduction of State Space

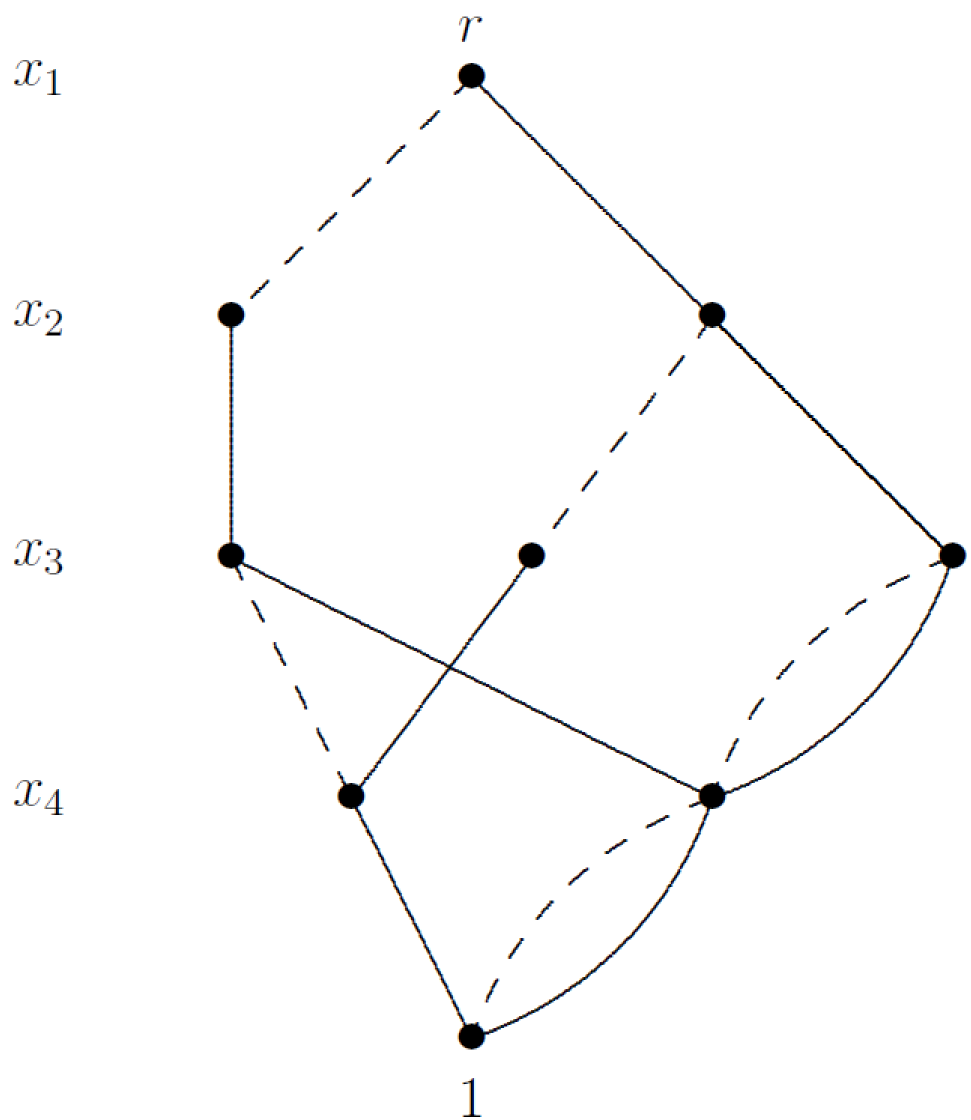
- **DD reduction** can dramatically simplify a DP problem.
  - Arrange arc costs to represent **canonical costs**
  - …while **not changing** the objective function.
  - This may allow **radical reduction** of state space.
  - Illustrate with a textbook **inventory problem**.

Example: Set Covering

DD for a set covering problem

	Set i 1	2	3	4
A	●	●		
B	●		●	●
C		●	●	
D		●		●

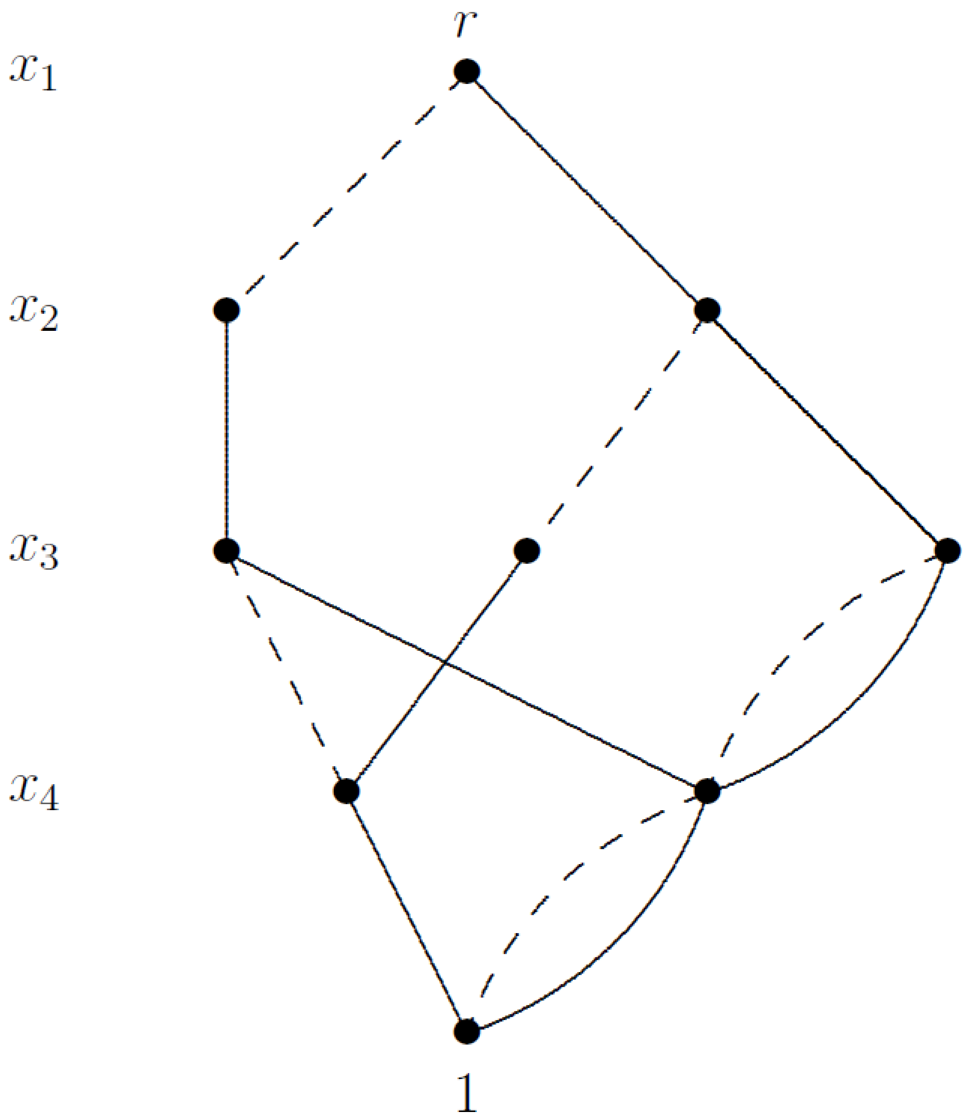
x_i = 1 when we select set i



Example: Set Covering

Suppose we have a **nonseparable** cost function

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

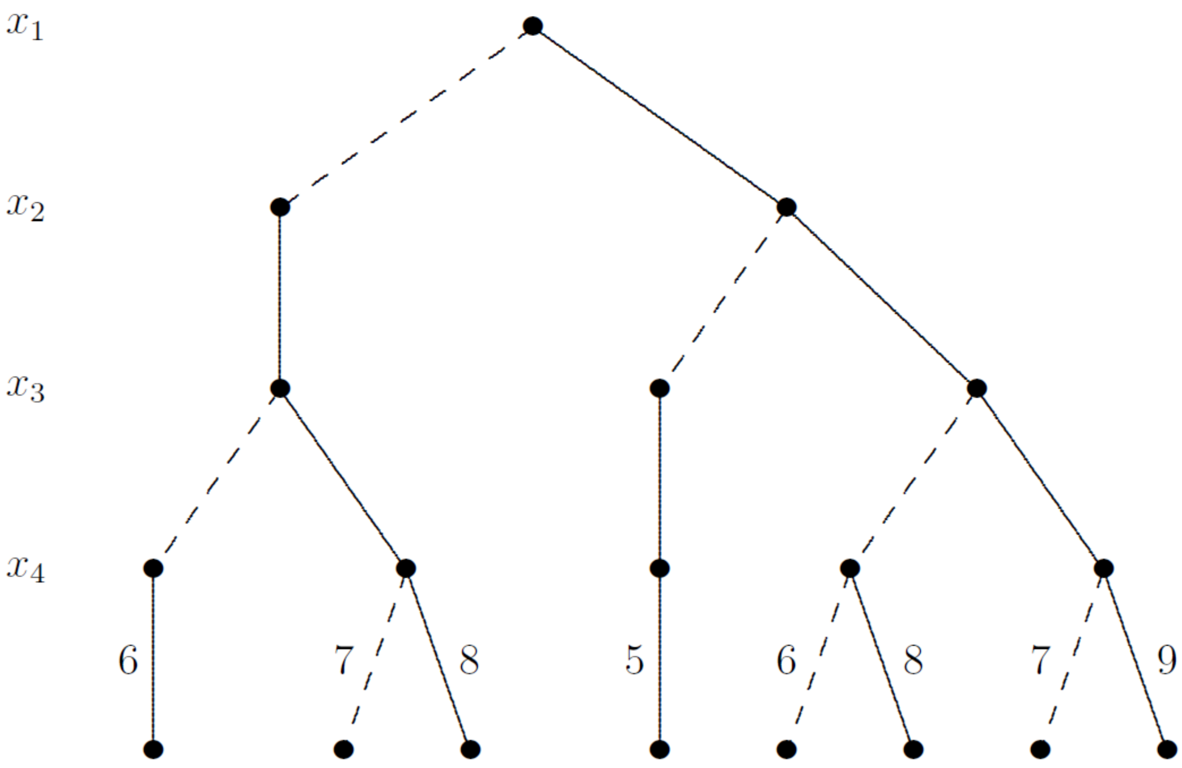


Modeling the Objective Function

Obtaining canonical costs

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

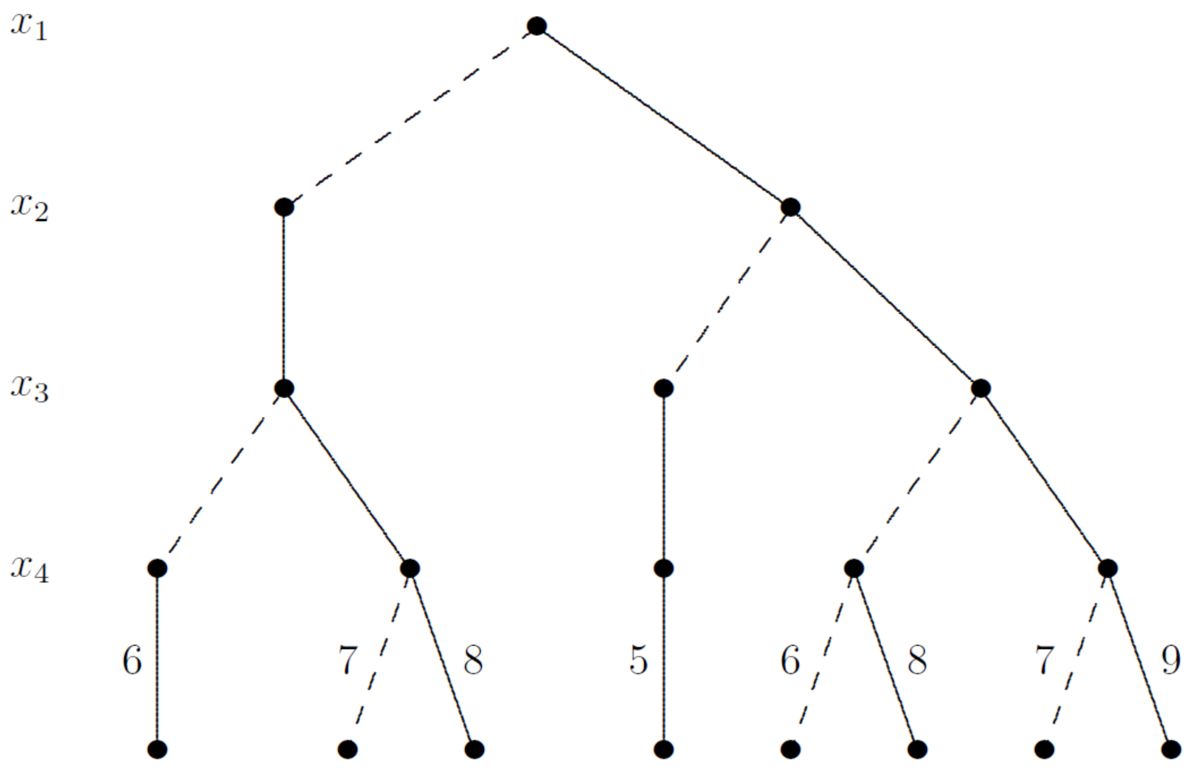


Modeling the Objective Function

Obtaining canonical costs

Put costs on leaves of branching tree.

But now we can't reduce the tree as before.



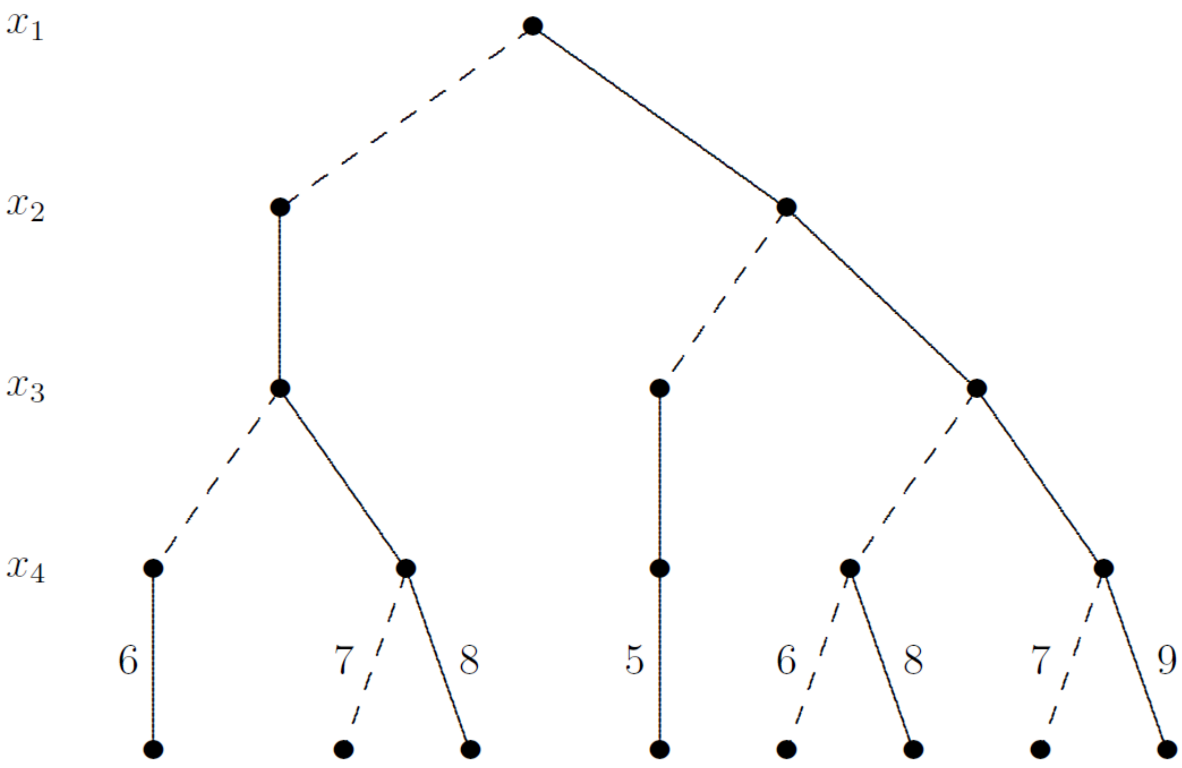
Modeling the Objective Function

Obtaining canonical costs

Put costs on leaves of branching tree.

But now we can't reduce the tree as before.

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



JH (2013)

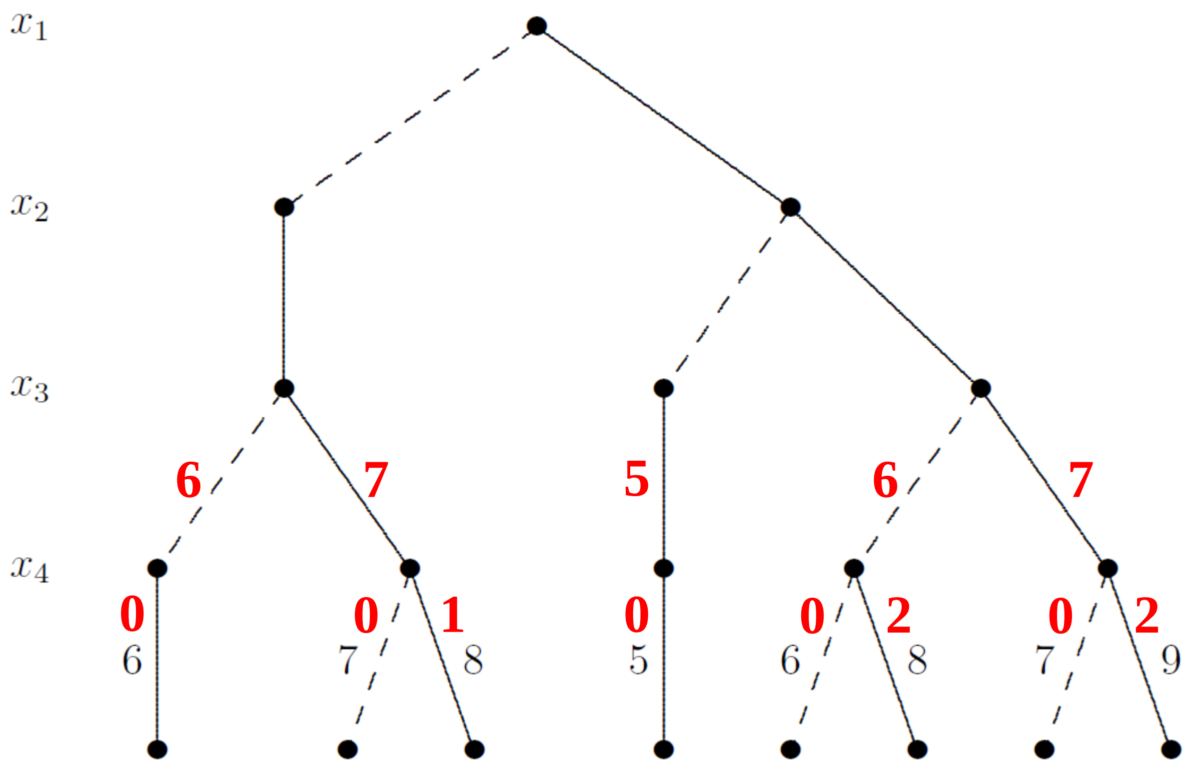
Modeling the Objective Function

Obtaining canonical costs

Put costs on leaves of branching tree.

But now we can't reduce the tree as before.

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



JH (2013)

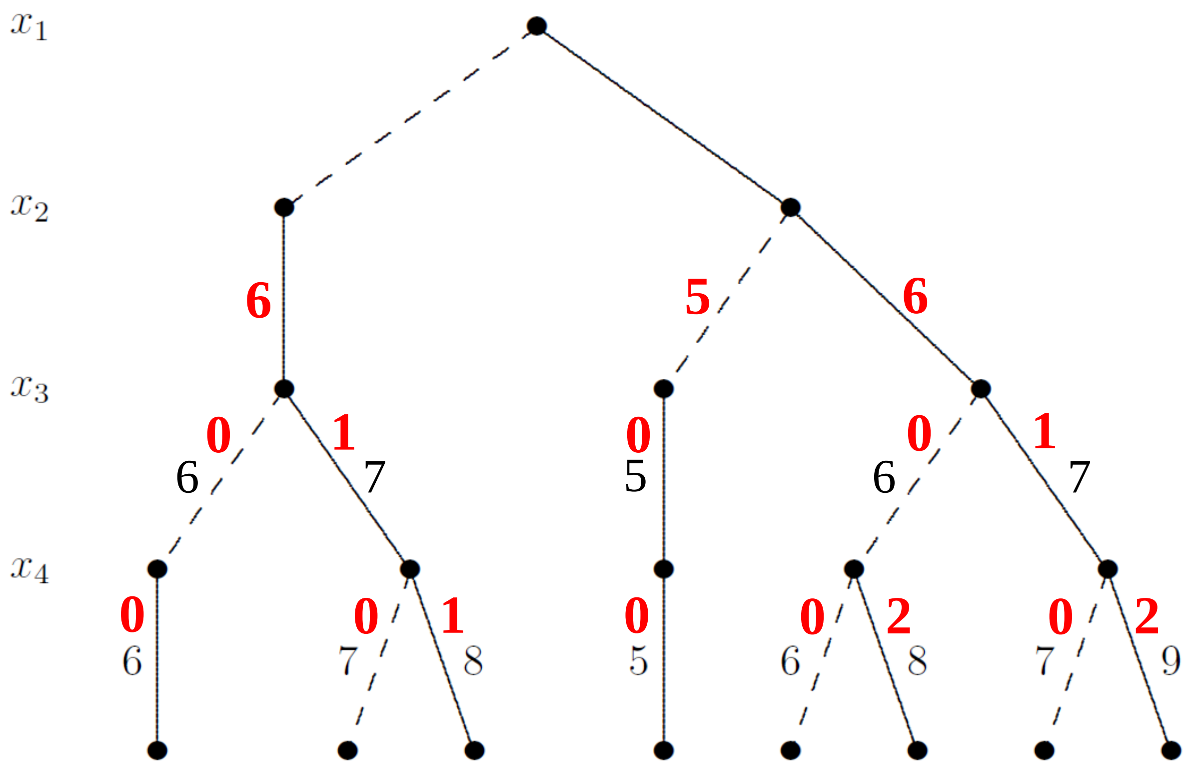
Modeling the Objective Function

Obtaining canonical costs

Put costs on leaves of branching tree.

But now we can't reduce the tree as before.

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



JH (2013)

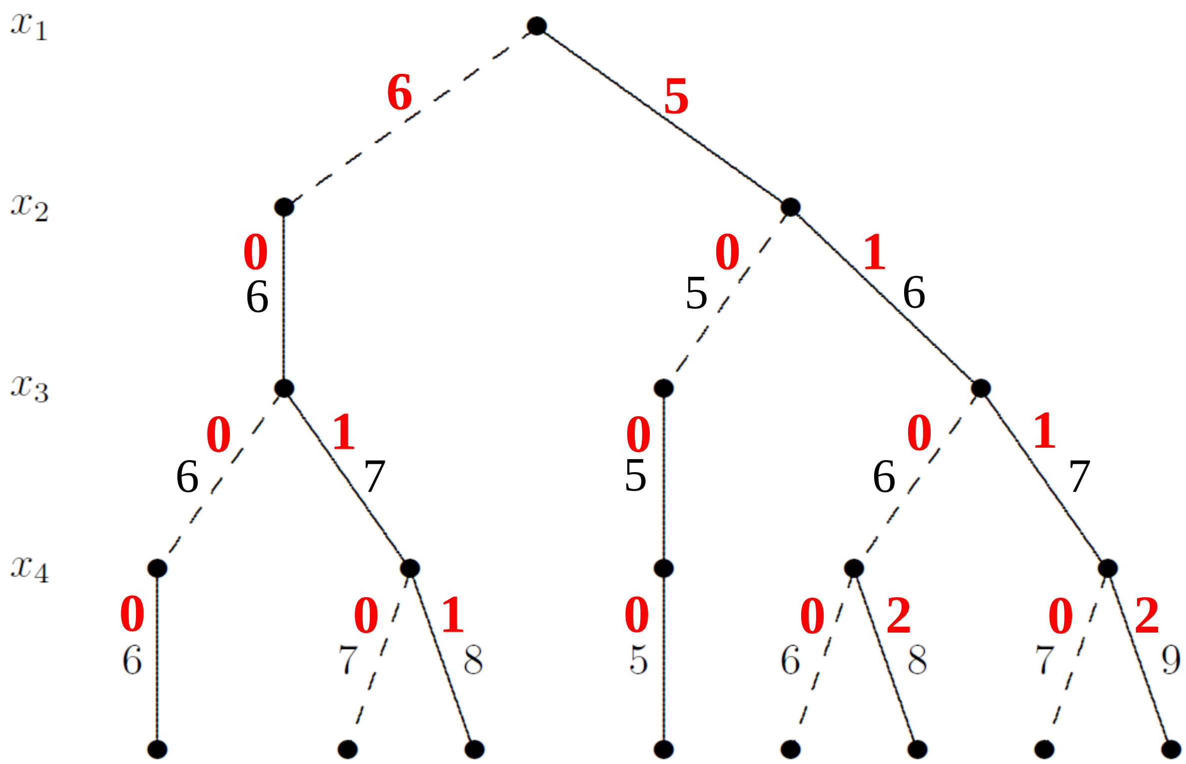
Modeling the Objective Function

Obtaining canonical costs

Put costs on leaves of branching tree.

But now we can't reduce the tree as before.

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

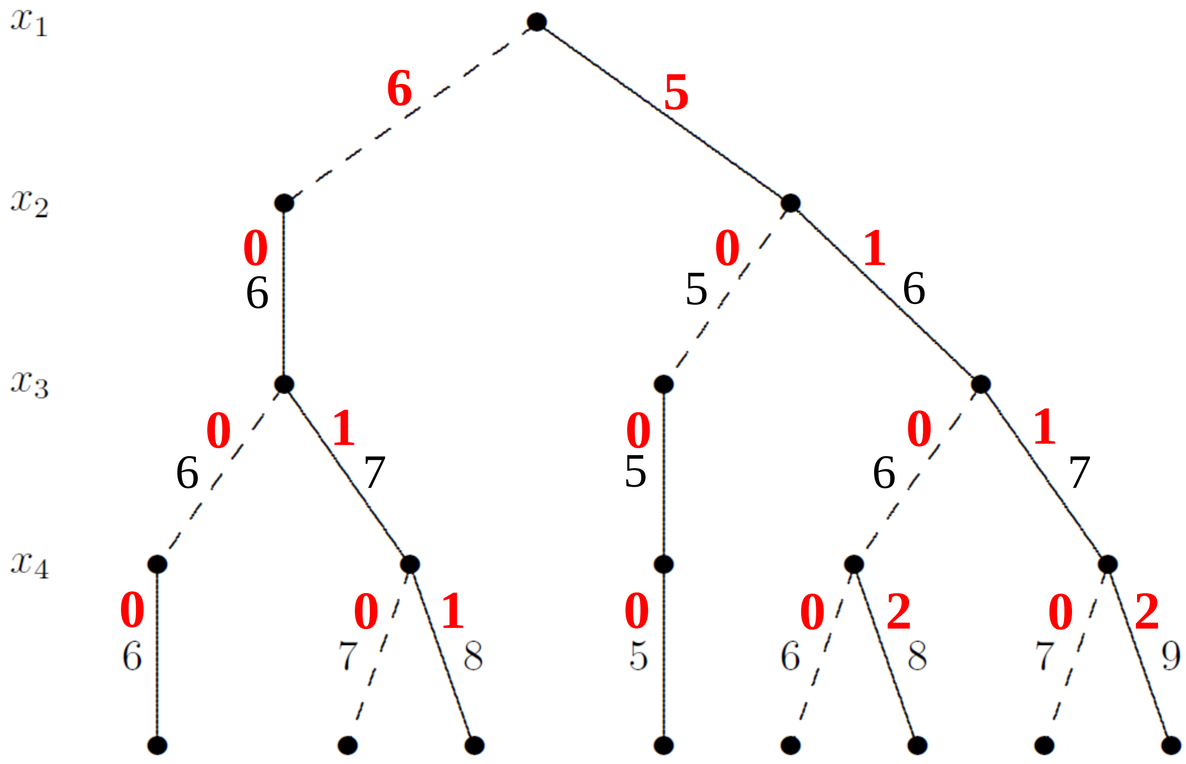


JH (2013)

Modeling the Objective Function

Obtaining a reduced DD

Now the tree can be reduced.

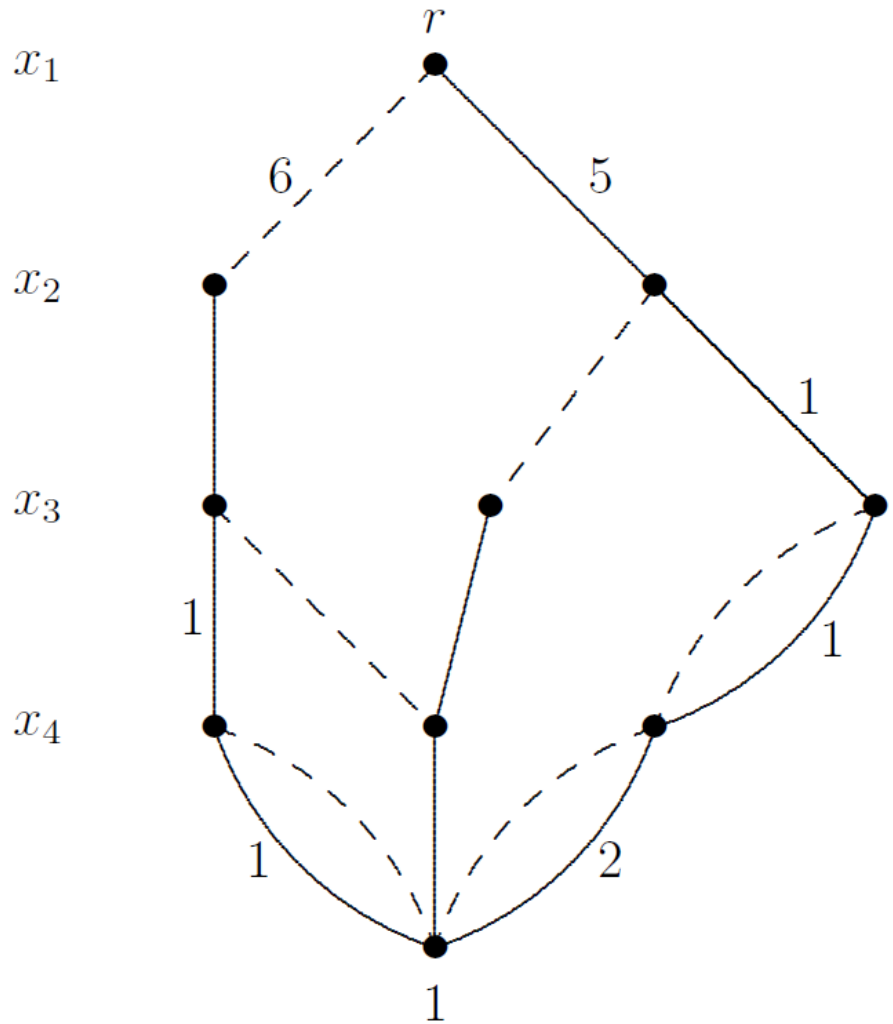


JH (2013)

Modeling the Objective Function

Obtaining a reduced DD

Now the tree can be reduced.

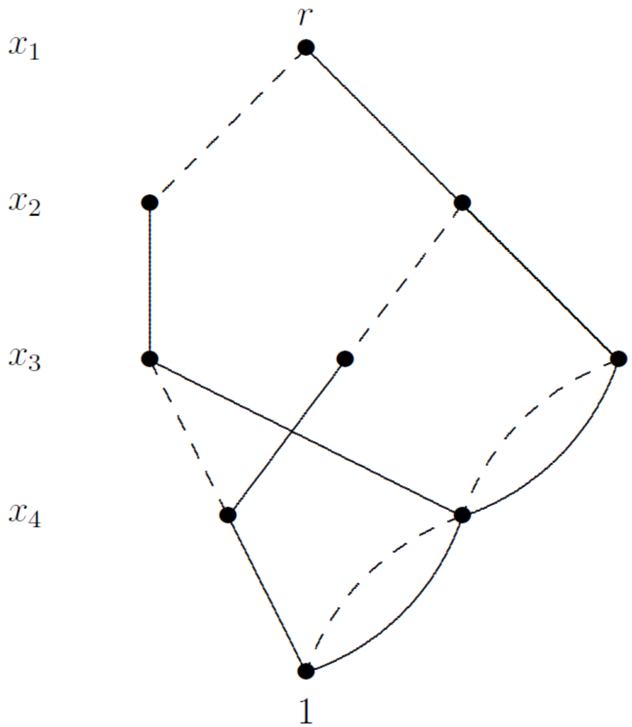


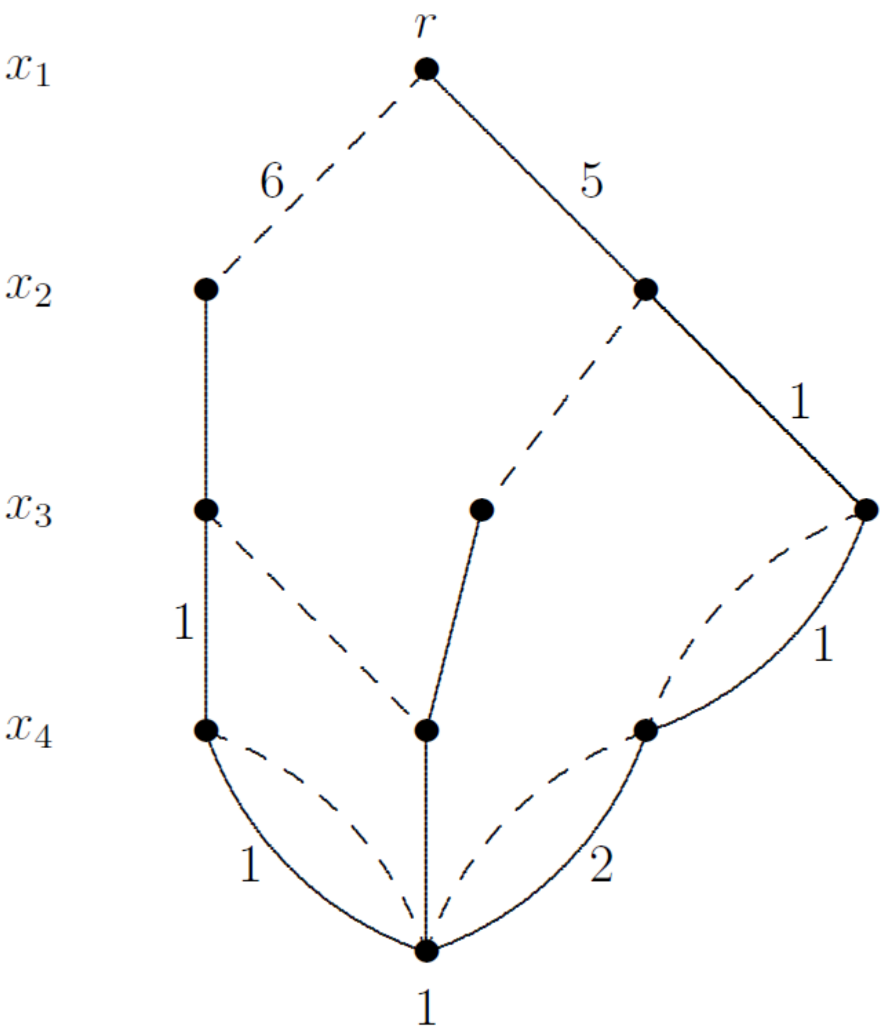
JH (2013)

Modeling the Objective Function

Obtaining a reduced DD

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





Modeling the Objective Function

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

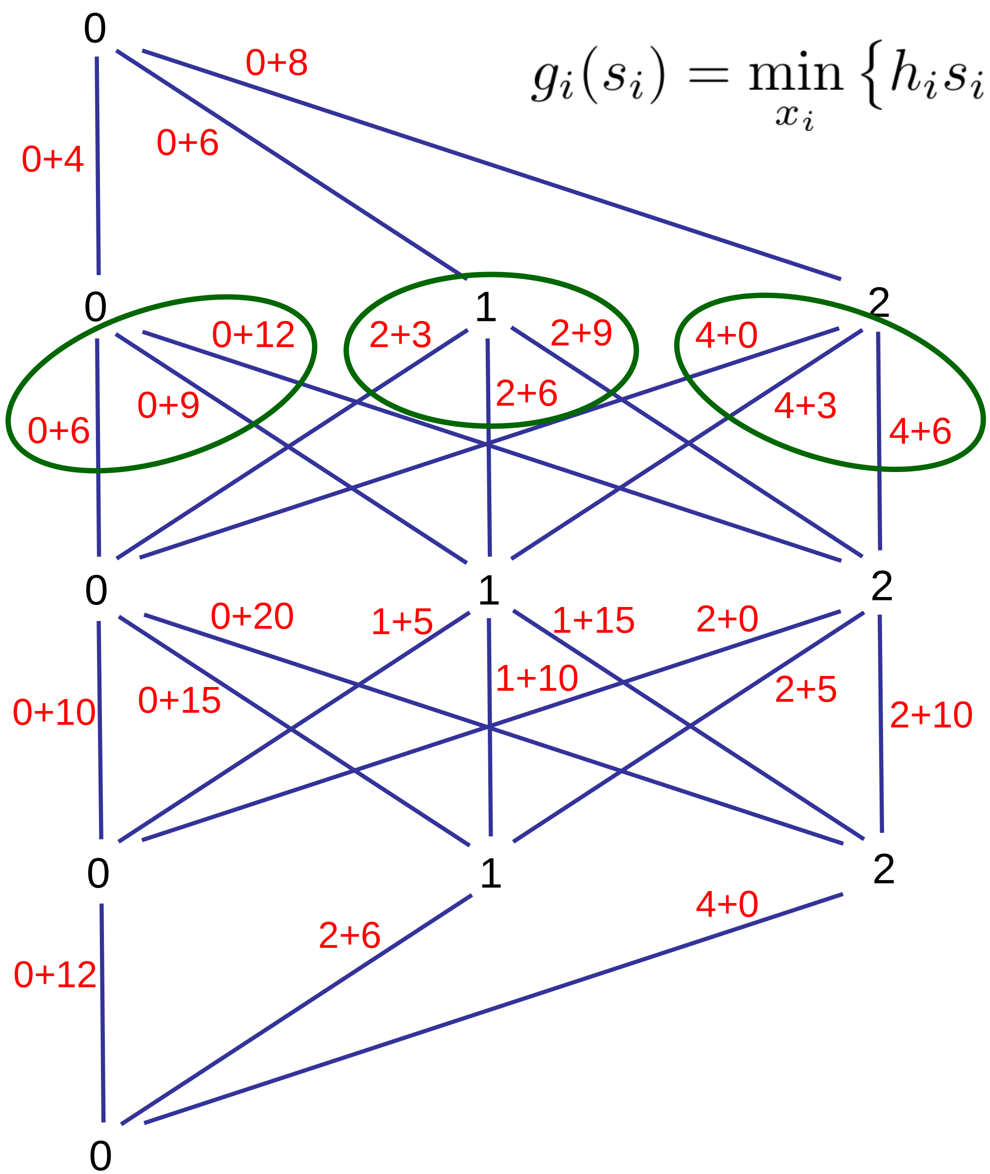
JH (2013),
Similar result for AADDs:
Sanner & McAllester (2005)

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

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

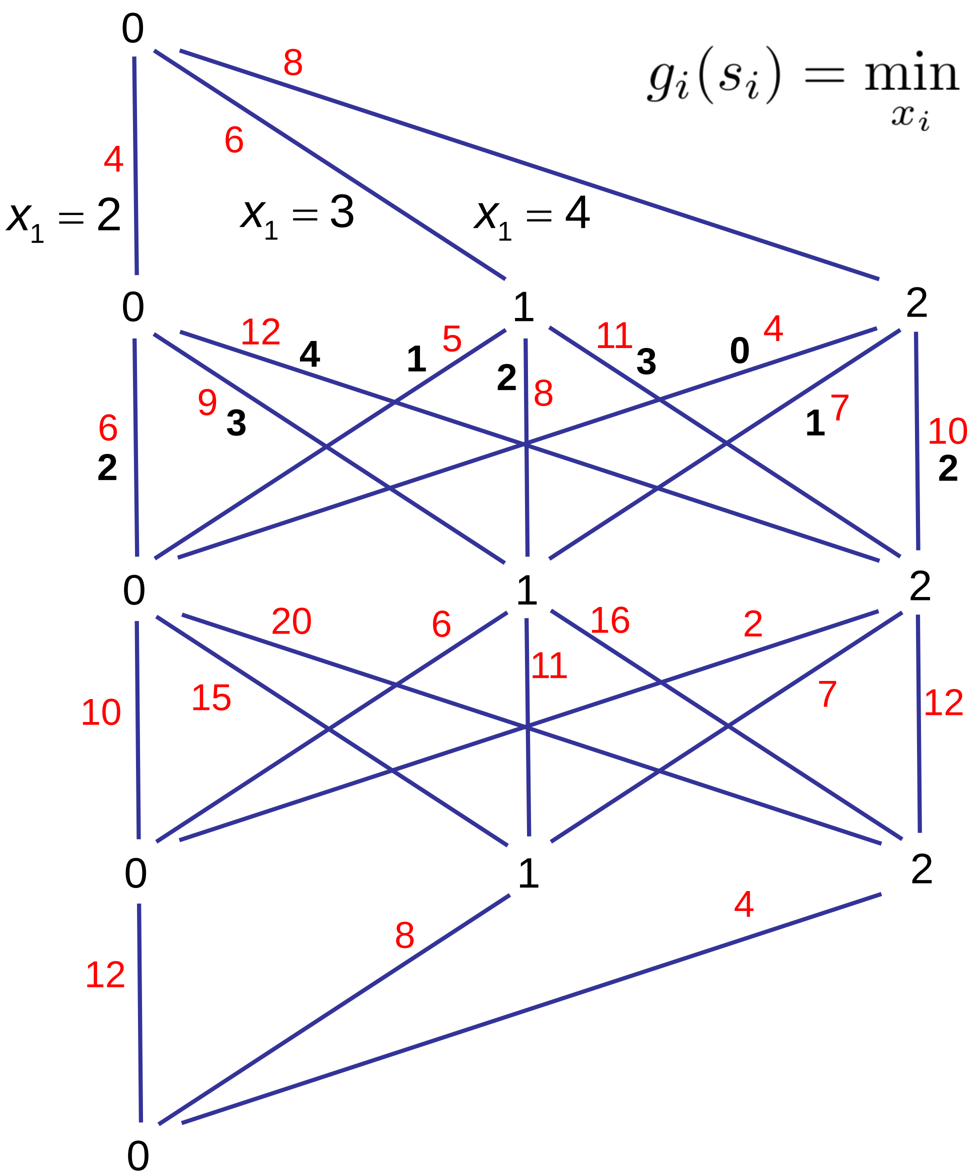


Arcs leaving each node are very similar.

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

JH (2013)

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

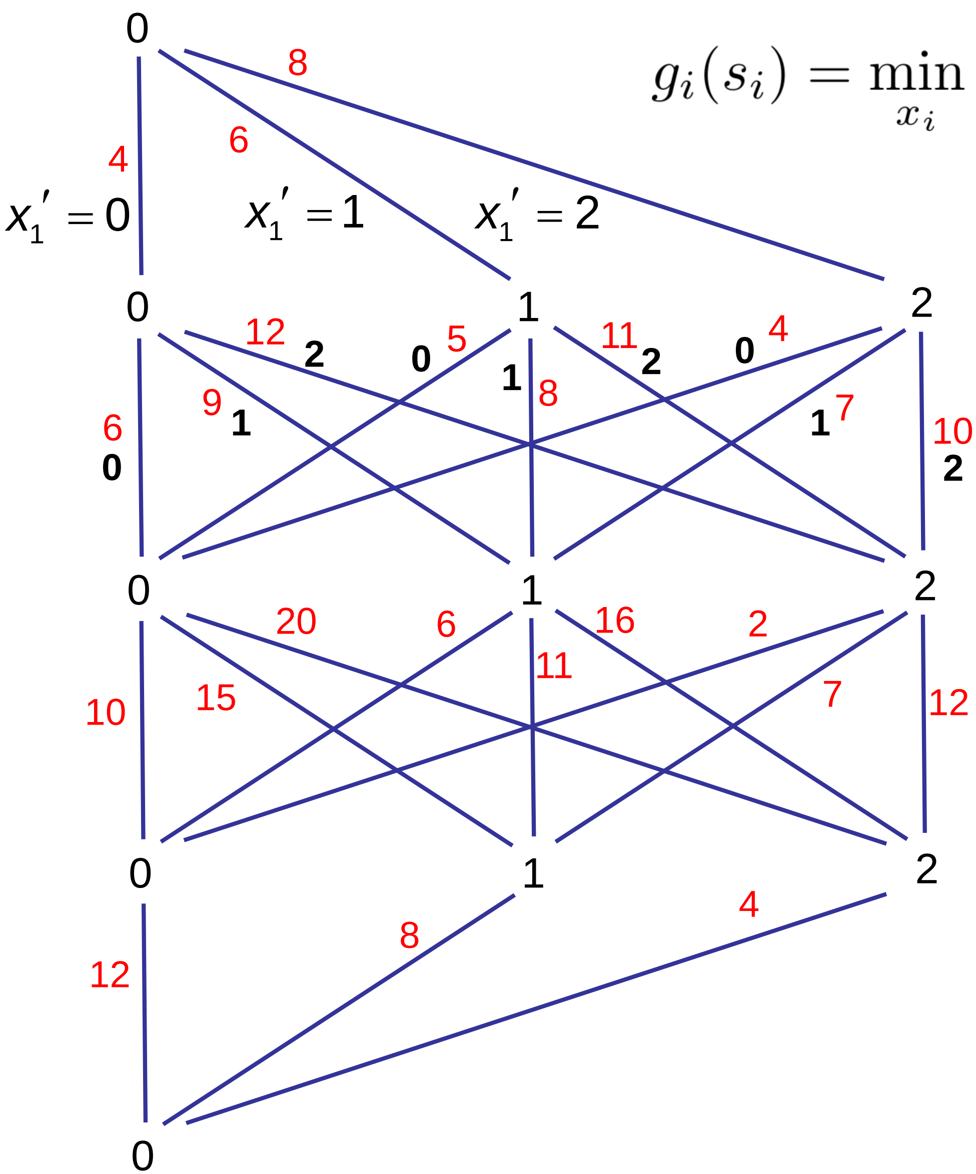
$$x'_i = s_i + x_i - d_i$$

be the stock level in next period.

JH (2013)

Inventory Problem

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



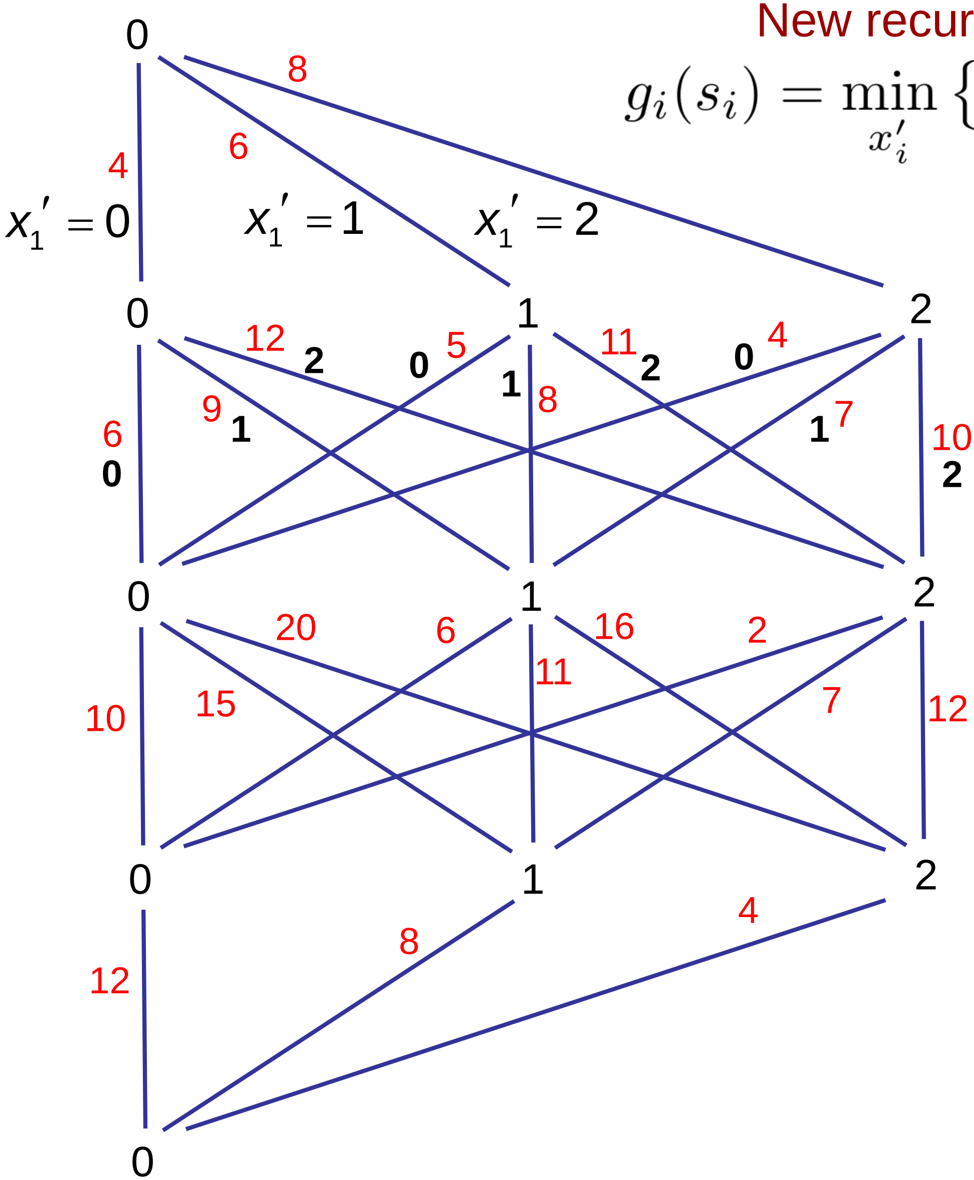
To equalize controls, let

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

be the stock level in next period.

JH (2013)

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

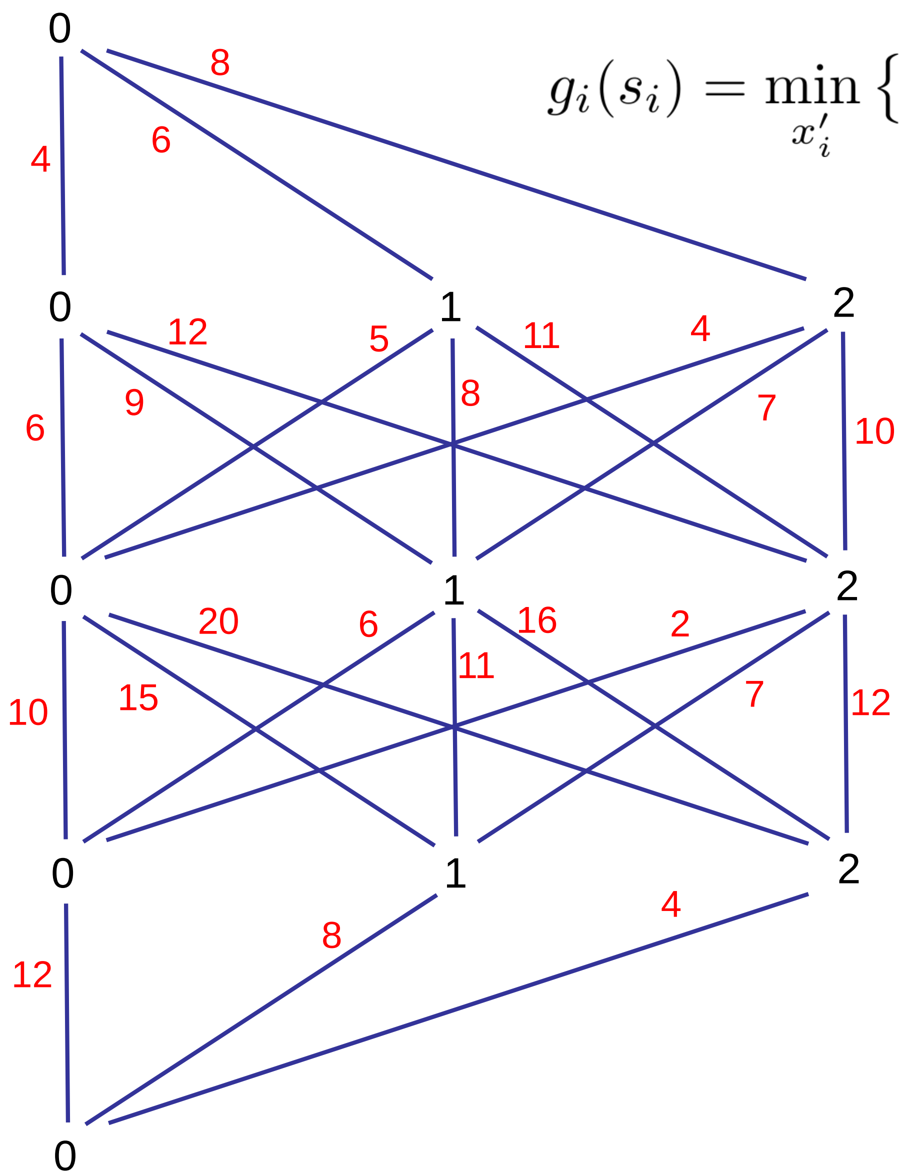
$$x_i' = s_i + x_i - d_i$$

be the stock level in next period.

JH (2013)

Inventory Problem

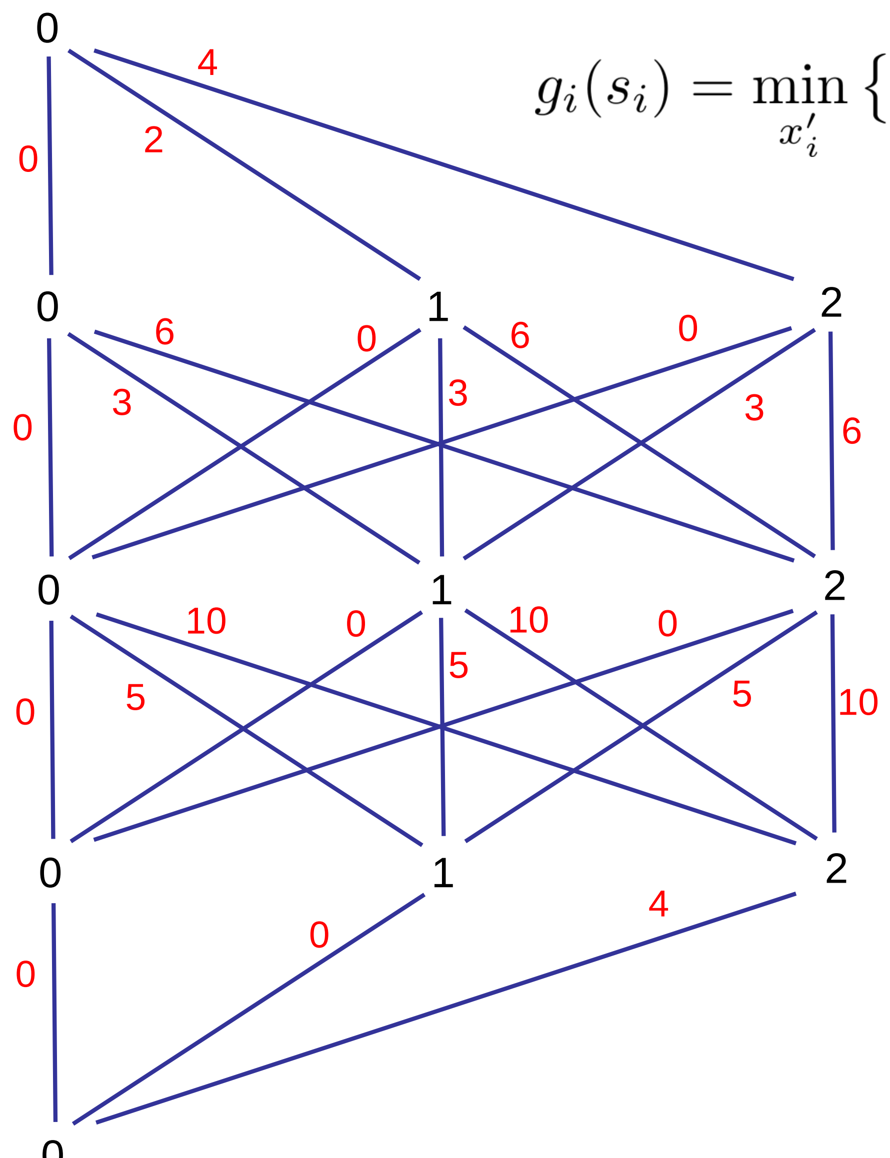
$$g_i(s_i) = \min_{x_i'} \left\{ h_i s_i + c_i(x_i' - s_i + d_i) + g_{i+1}(x_i') \right\}$$



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

JH (2013)

Inventory Problem



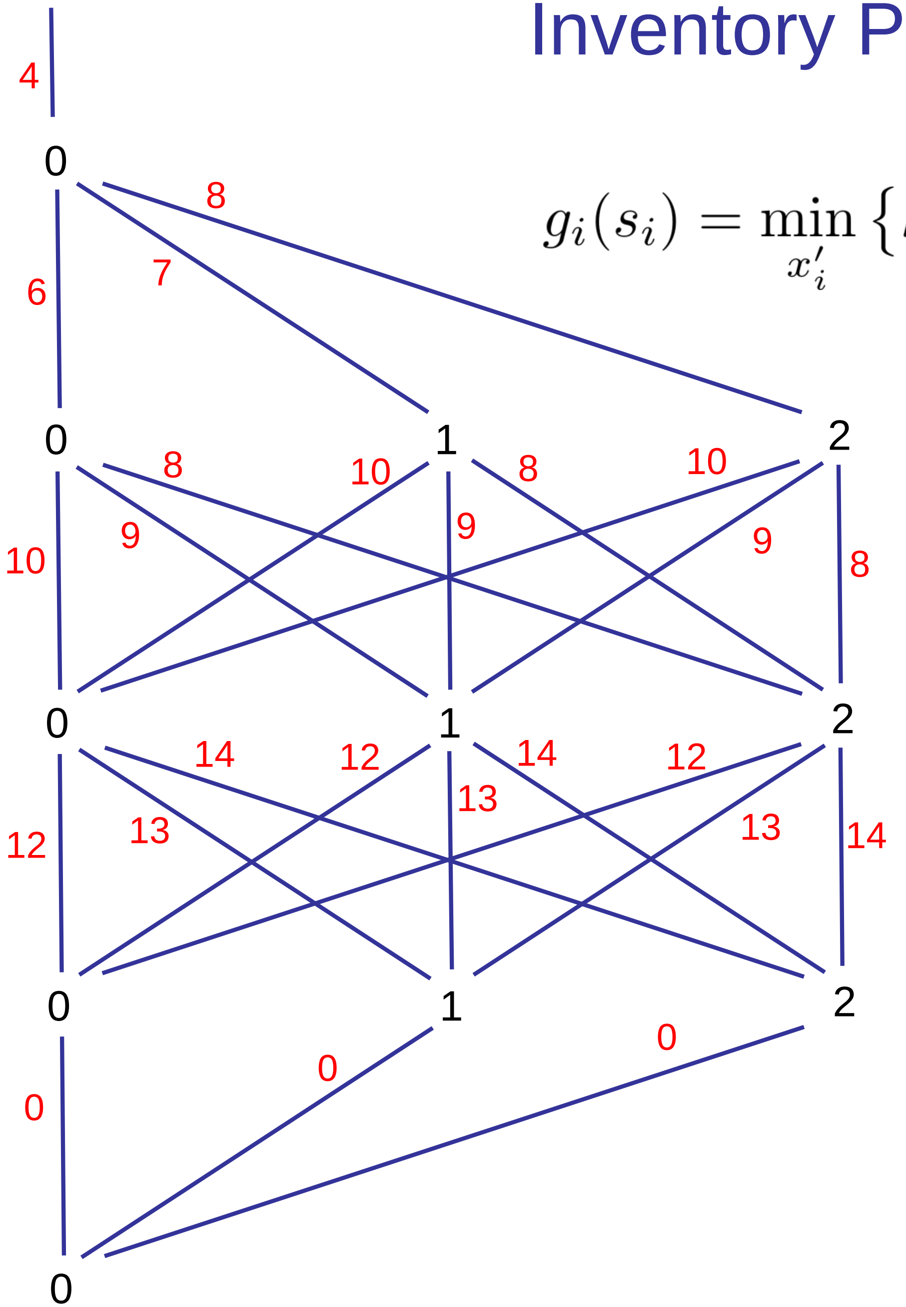
$$g_i(s_i) = \min_{x_i'} \left\{ h_i s_i + c_i(x_i' - s_i + d_i) + g_{i+1}(x_i') \right\}$$

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.

JH (2013)

Inventory Problem



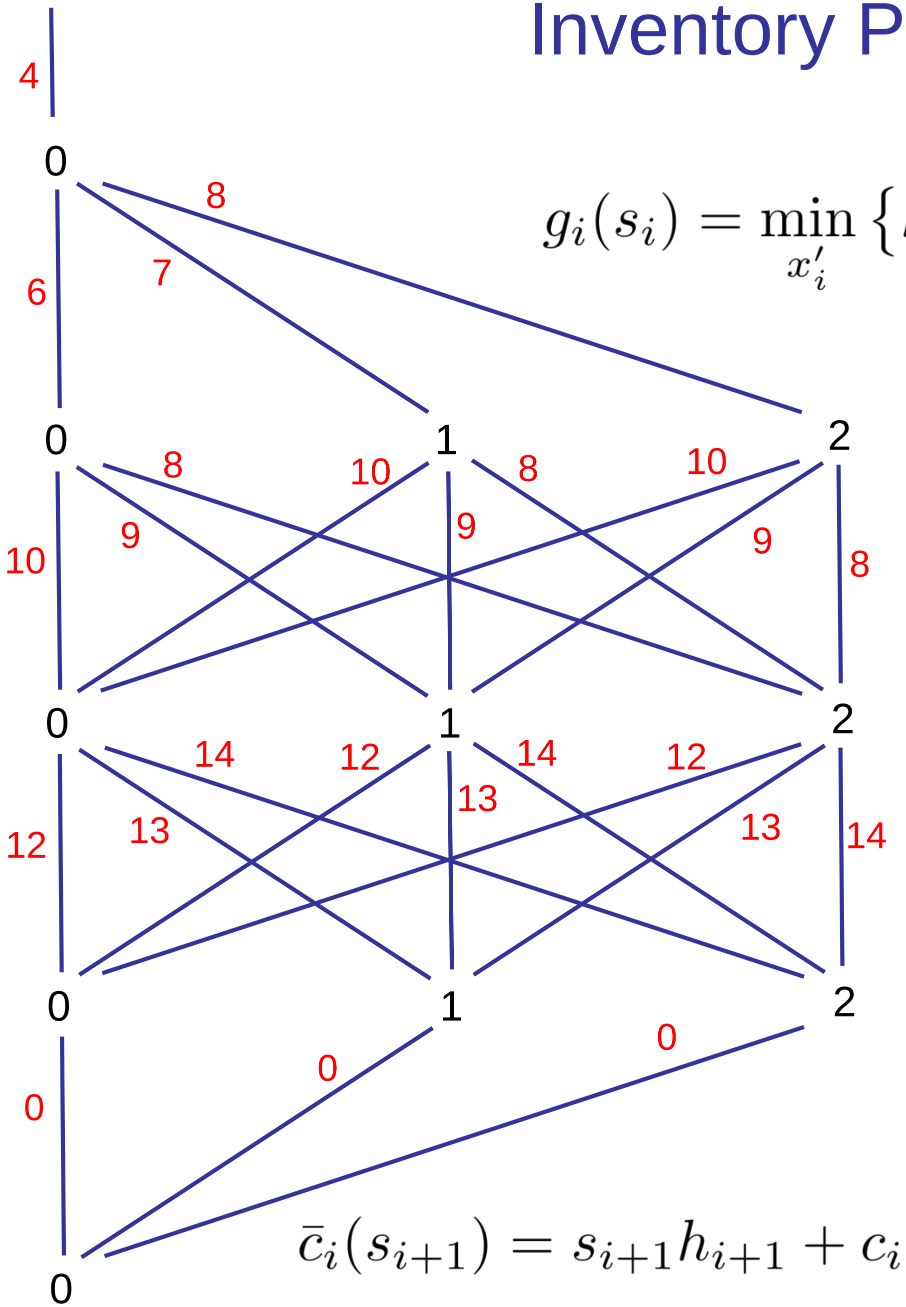
$$g_i(s_i) = \min_{x_i'} \left\{ h_i s_i + c_i(x_i' - s_i + d_i) + g_{i+1}(x_i') \right\}$$

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.

JH (2013)

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}).

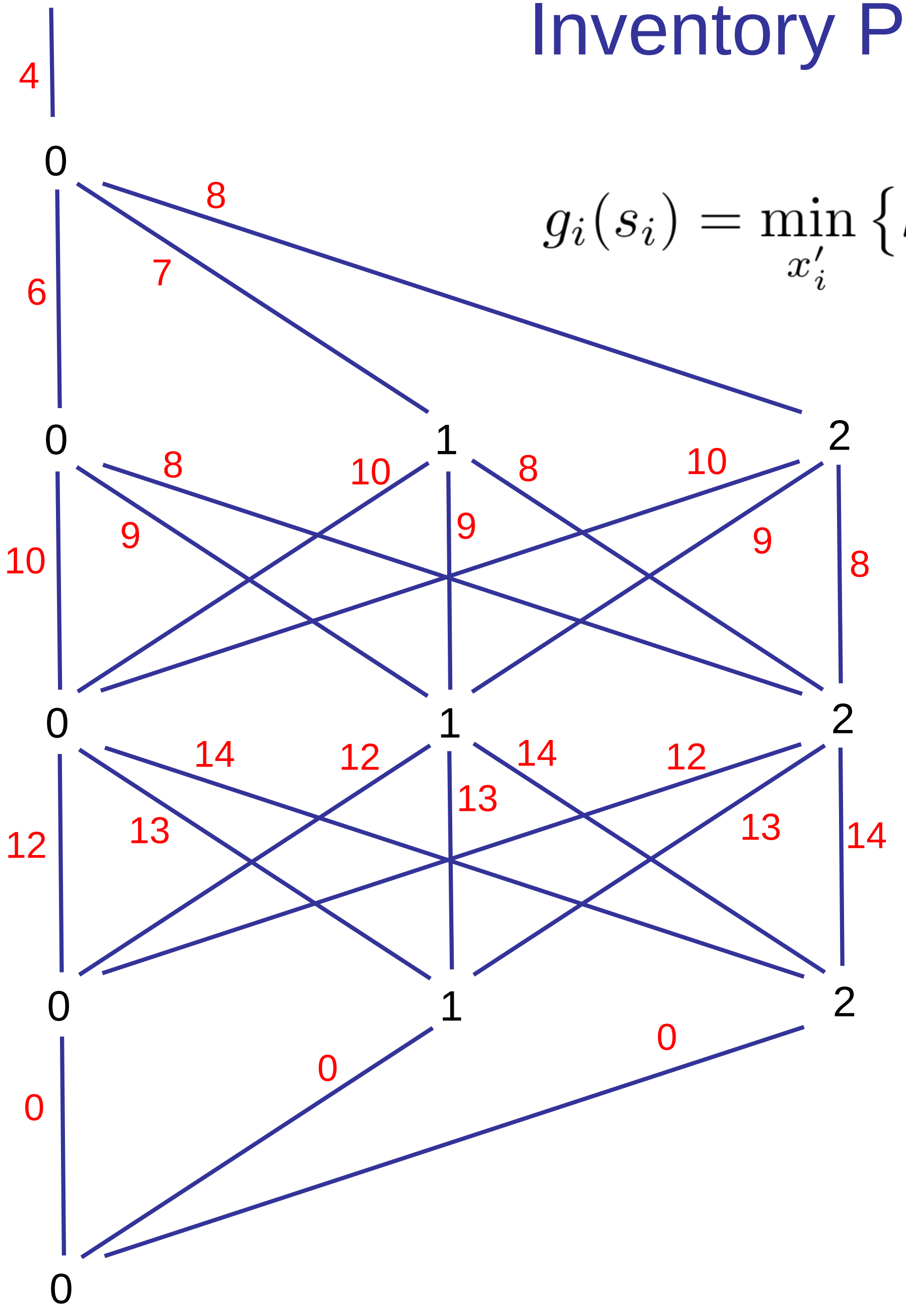
Add these offsets to incoming arcs.

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



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

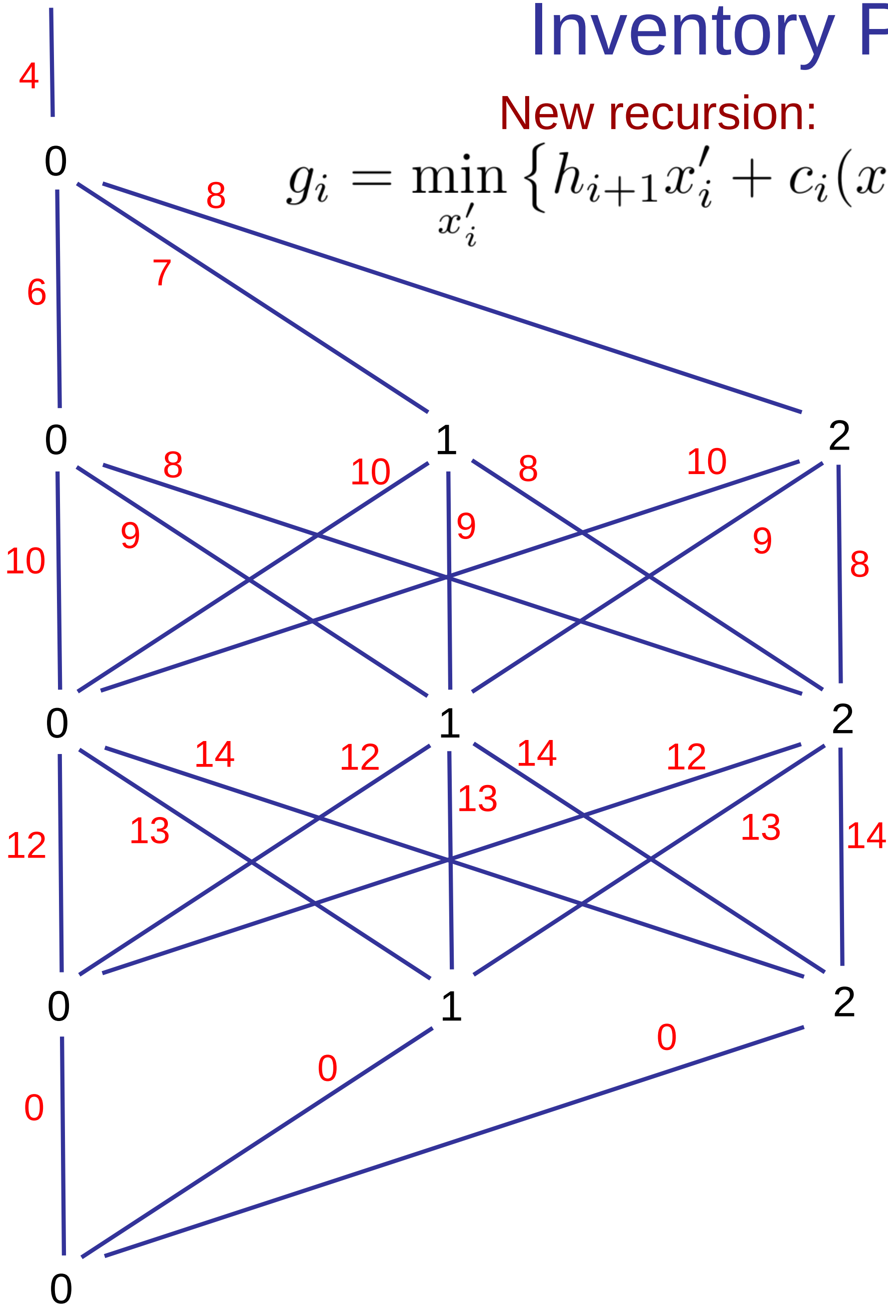
These are canonical costs with offset $\min_{s_{i+1}} \left\{ \bar{c}_i(s_{i+1}) \right\}$

JH (2013)

Inventory Problem

New recursion:

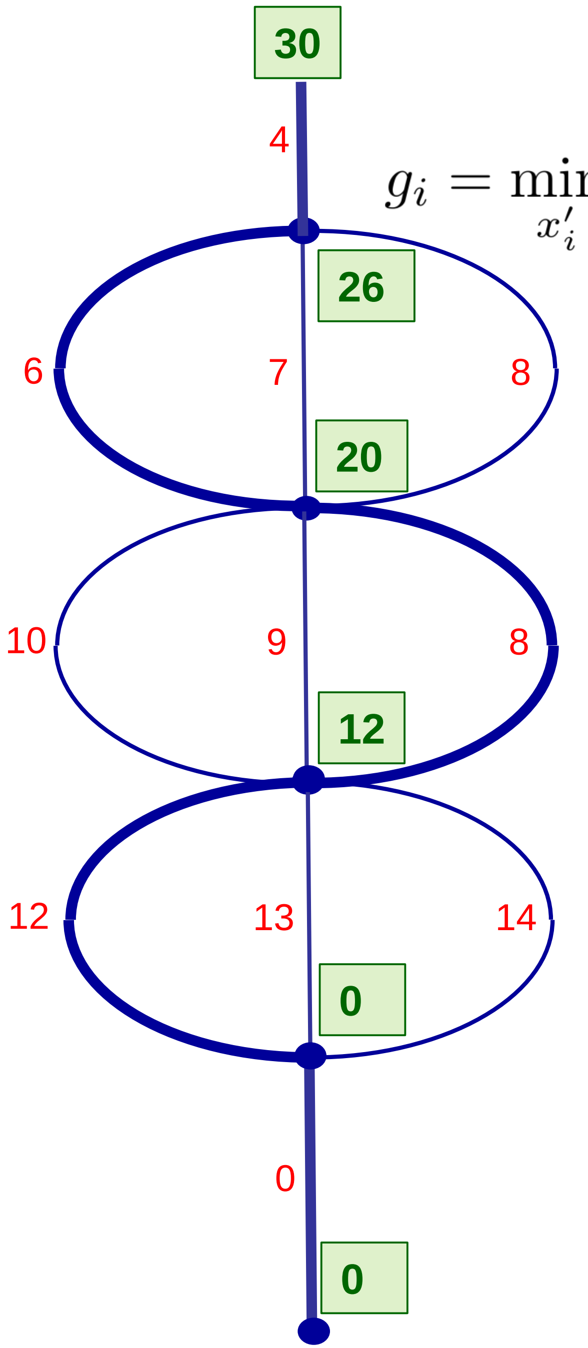
$$g_i = \min_{x_i'} \left\{ h_{i+1}x_i' + c_i(x_i' - m + d_i) + c_{i+1}(m - x_i') + g_{i+1} \right\}$$



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

JH (2013)

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.

Do this analysis **before** constructing the diagram.

JH (2013)

Outline

- Decision diagram **basics**
- Optimization with **exact** decision diagrams
- Providing the basic **elements of optimization**
  - **Modeling, relaxation, primal heuristics, constraint propagation, search, postoptimality**
- Research **frontiers**
  - Separation
  - Radical reduction of state space in DP
  - Nonlinear optimization
  - Nonserial recursion
- References

Nonlinear Optimization

- Several research projects now underway.
- If variables are discrete…
  - We don’t care whether the “constraints” are nonlinear.
  - We need only the state transition function.
  - Only the **cost function** is an issue.
- One study recently published:
  - **Nonlinear, nonseparable cost function**.

Bergman and Ciré (2018)

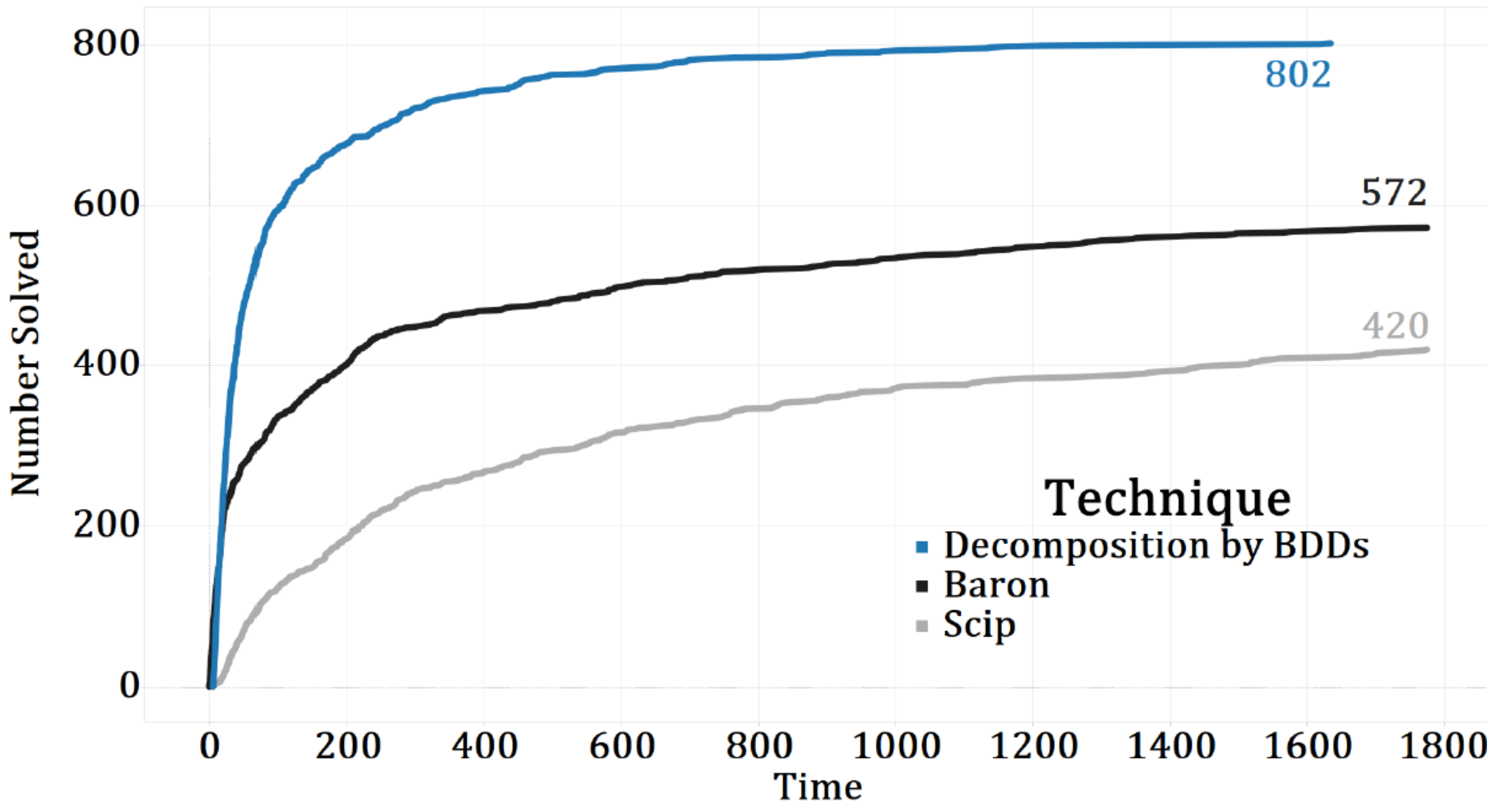
Nonlinear Optimization

- Approach
  - Construct **exact** diagram **for each term** of objective function.
    - Terms may be nonseparable.
    - Practical if limited number of variables **in each term**.
  - Immediate cost (arc cost) in DP model is effect of corresponding control on objective function value.
  - View each diagram as a 0-1 network flow problem with unit flow from root.
    - **Equate flow variables** in different diagrams that represent **same value** of **same control variable**.
    - Solve the resulting MIP.

Bergman and Ciré (2018)

Nonlinear Optimization

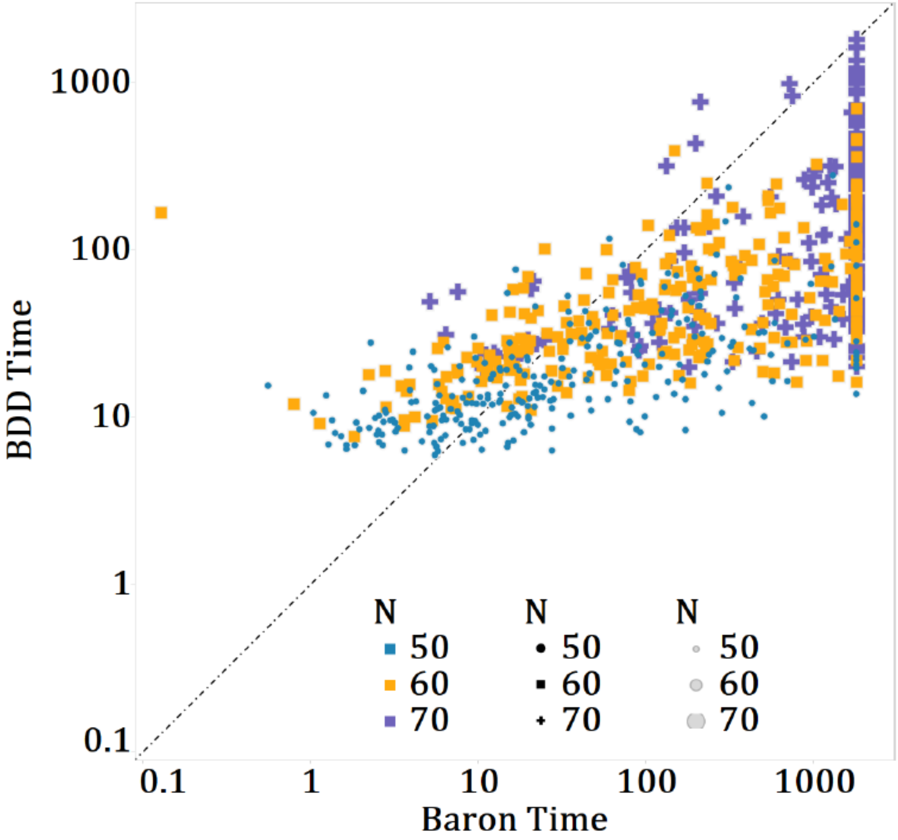
Portfolio Optimization



Bergman and Ciré (2018)

Nonlinear Optimization

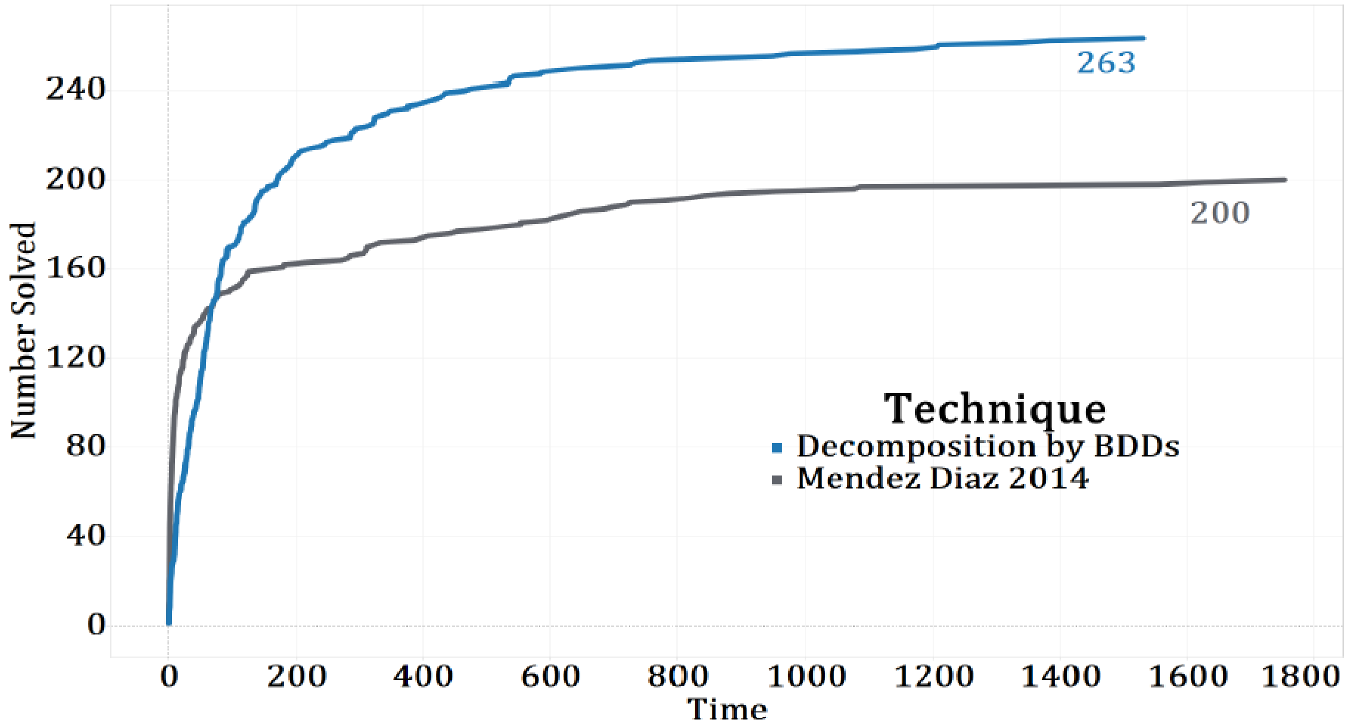
Portfolio Optimization



Bergman and Ciré (2018)

Nonlinear Optimization

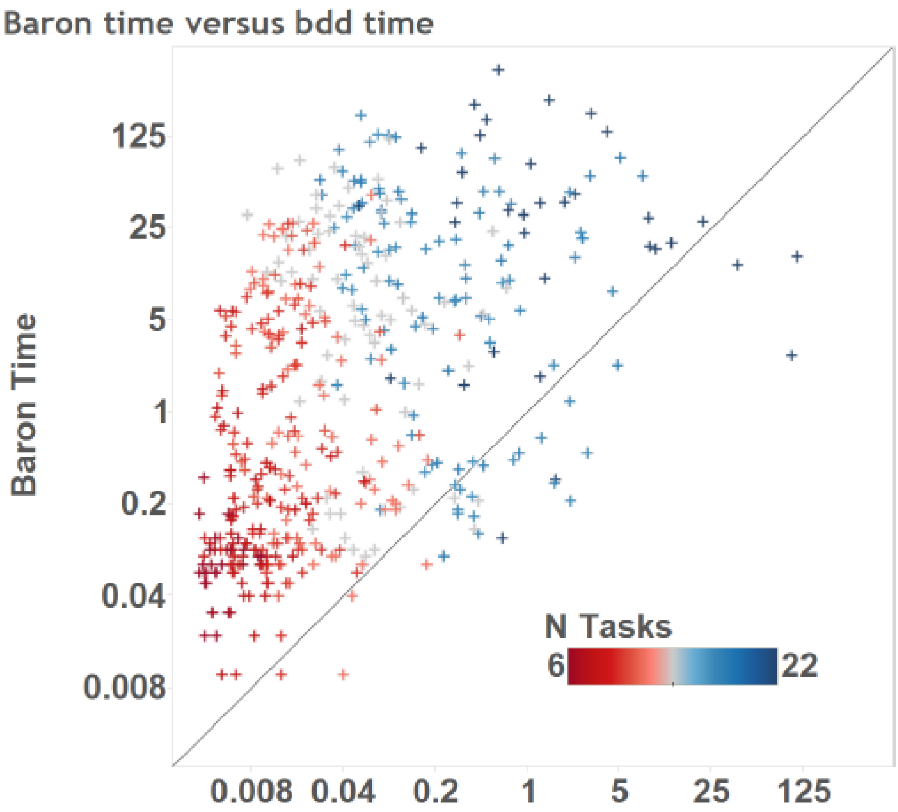
Product Assortment
(Latent Class Logit Assortment)



Bergman and Ciré (2018)

Nonlinear Optimization

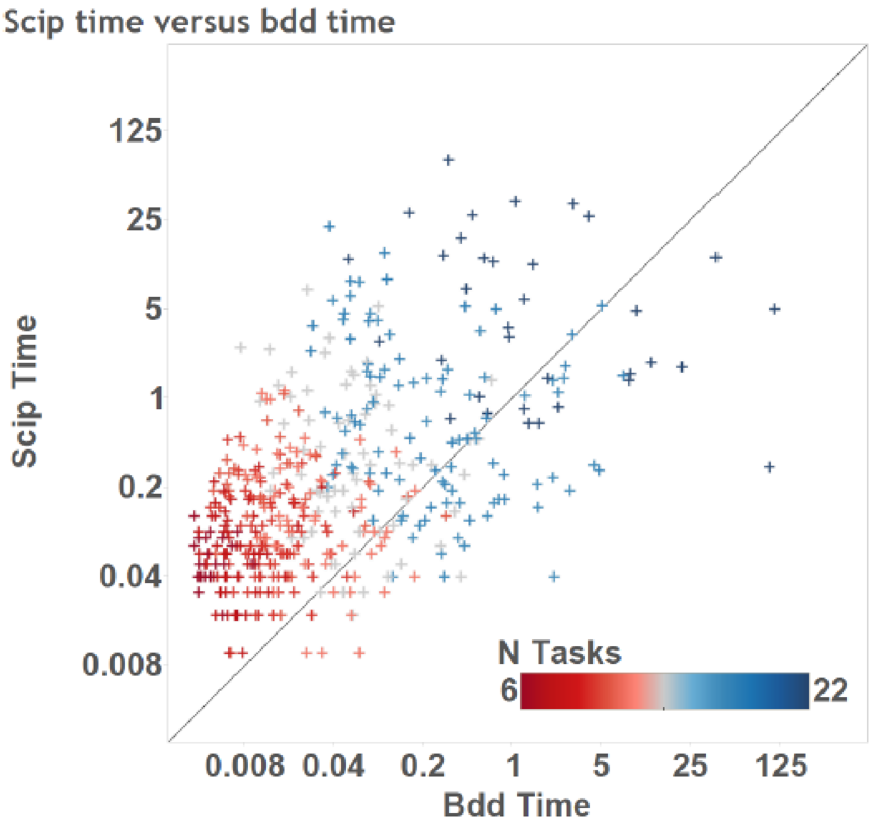
Workflow Employee Assignment



Bergman and Ciré (2018)

Nonlinear Optimization

Workflow Employee Assignment



Bergman and Ciré (2018)

Outline

- Decision diagram **basics**
- Optimization with **exact** decision diagrams
- Providing the basic **elements of optimization**
  - **Modeling, relaxation, primal heuristics, constraint propagation, search, postoptimality**
- Research **frontiers**
  - Separation
  - Radical reduction of state space in DP
  - Nonlinear optimization
  - Nonserial recursion
- References

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				●		●

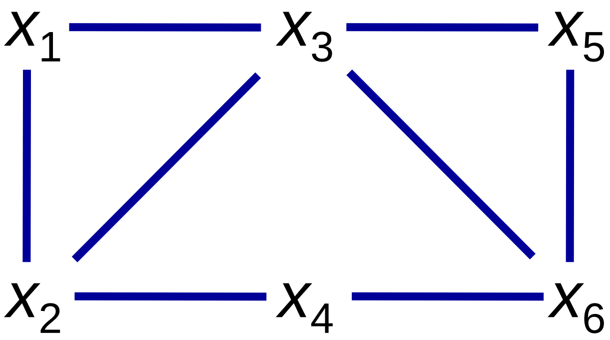
0-1 formulation

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

$x_j = 1 \Rightarrow$ set j selected

Set Partitioning example

Dependency graph



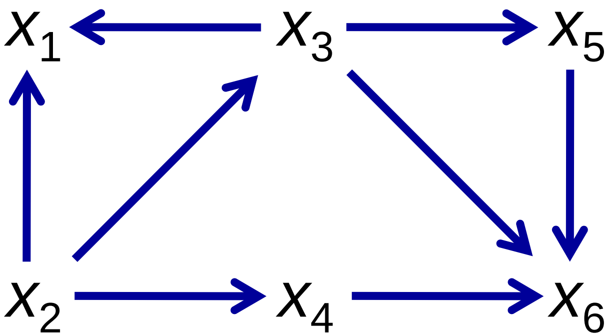
0-1 formulation

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

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

Set Partitioning example

Dependency graph



Enumeration order

x_2

x_3

x_4

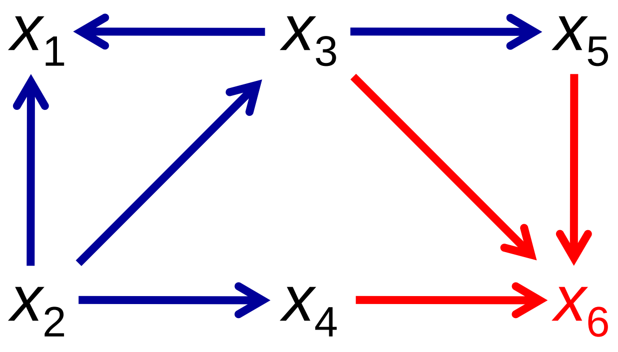
x_1

x_5

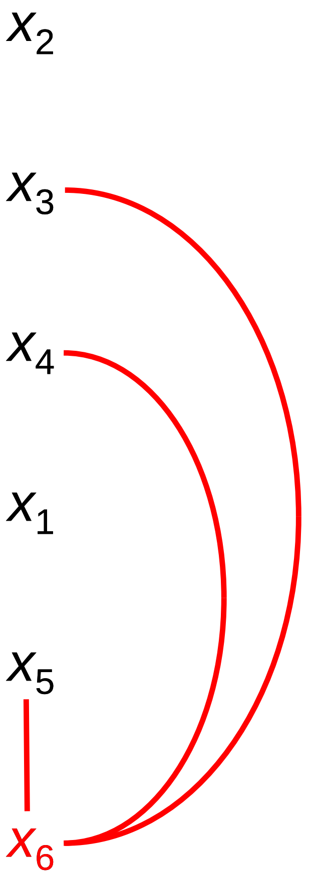
x_6

Set Partitioning example

Dependency graph

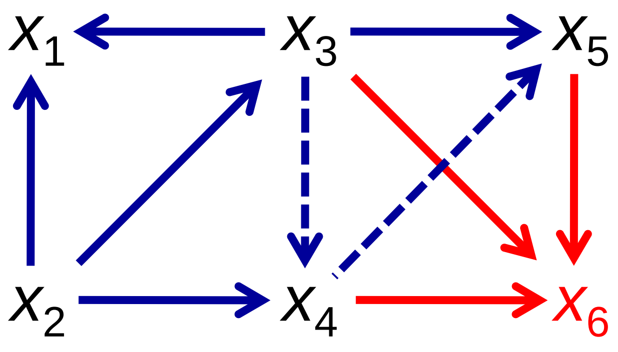


Enumeration order

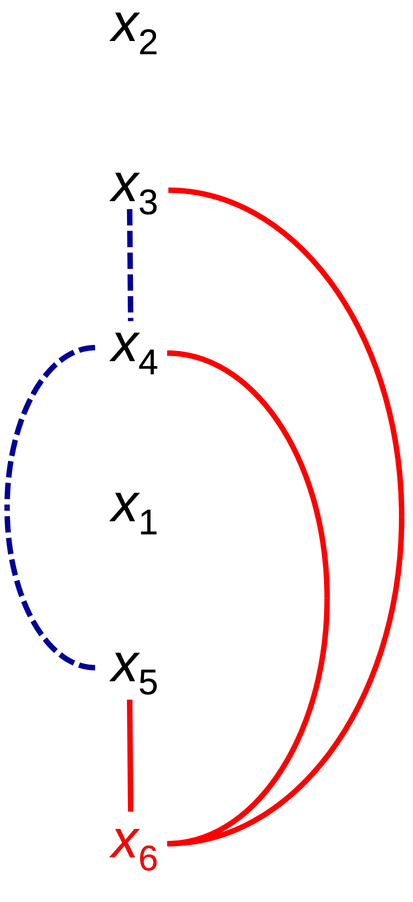


Set Partitioning example

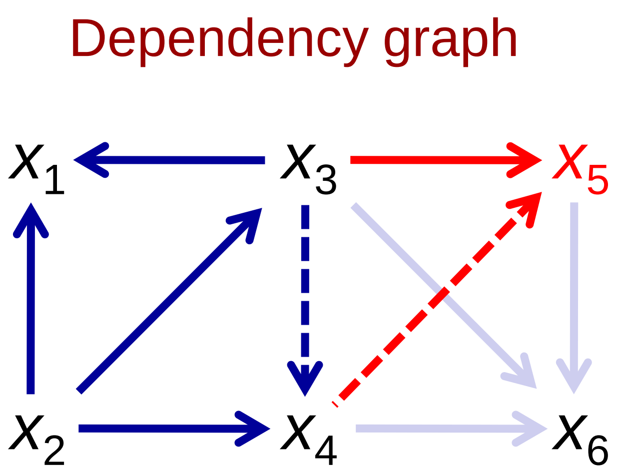
Dependency graph

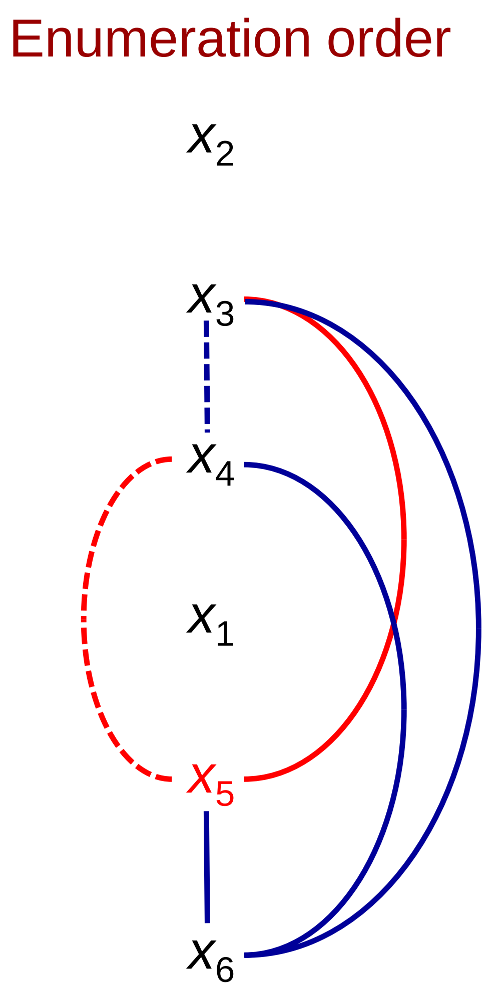


Enumeration order

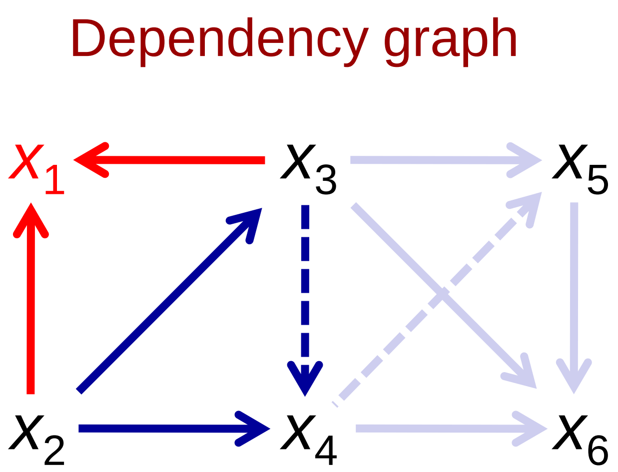


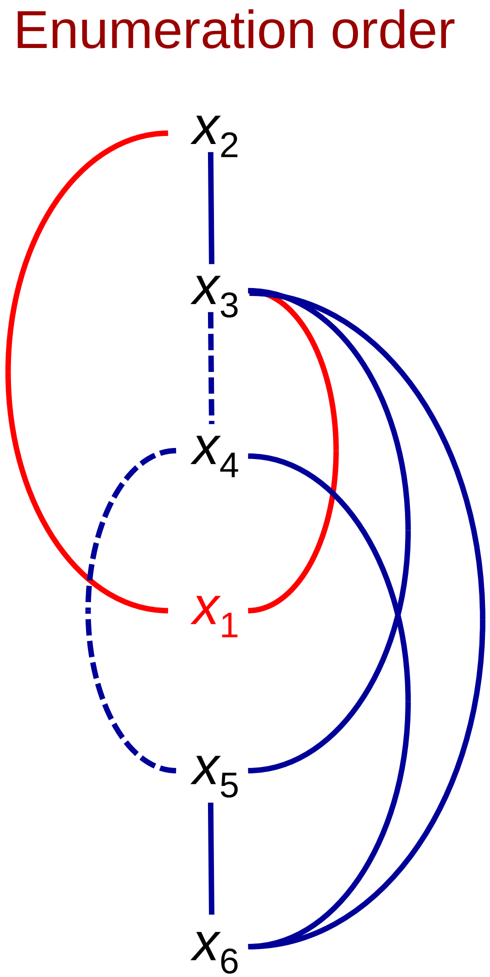
Set Partitioning example





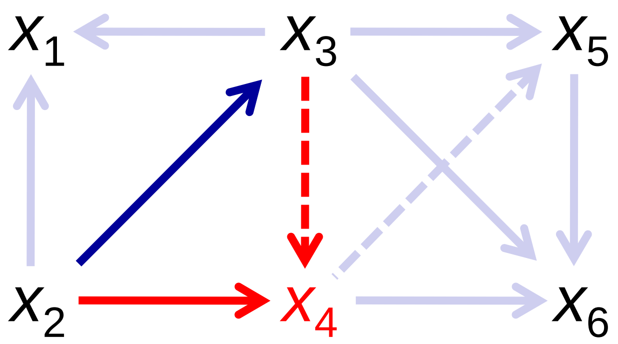
Set Partitioning example



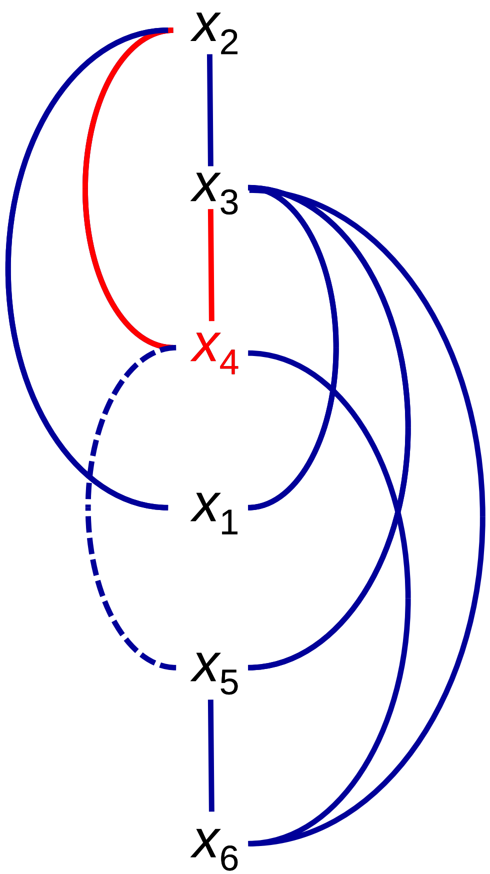


Set Partitioning example

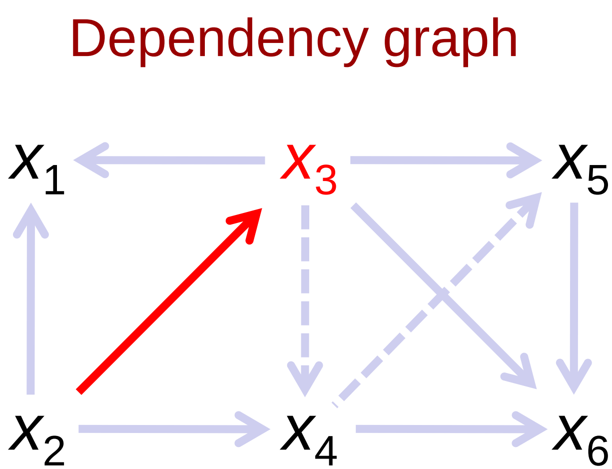
Dependency graph

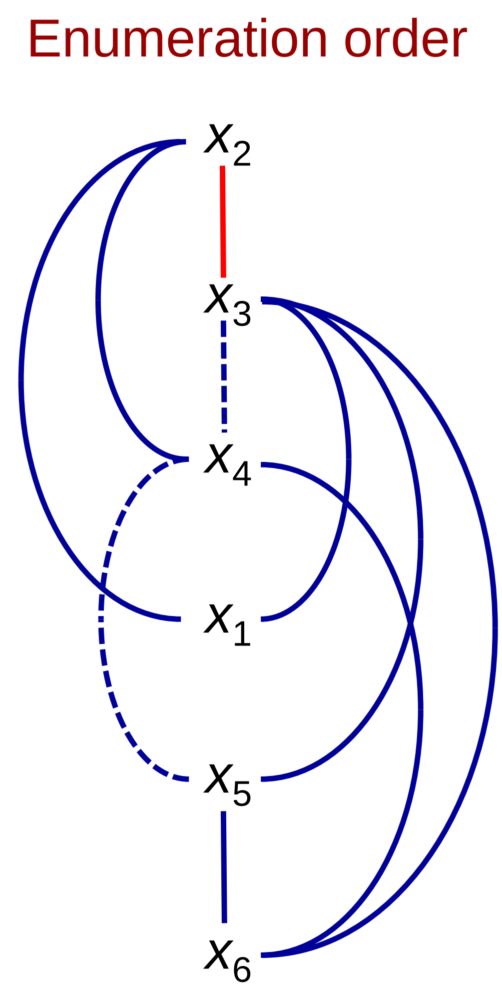


Enumeration order



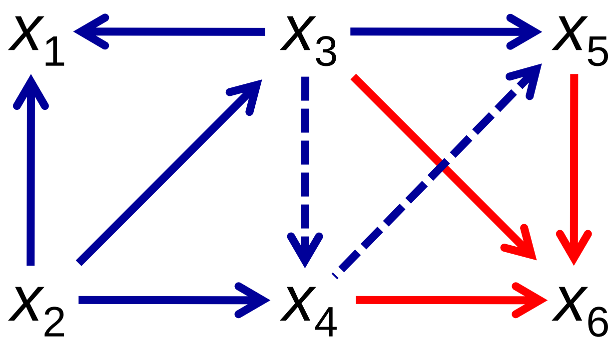
Set Partitioning example





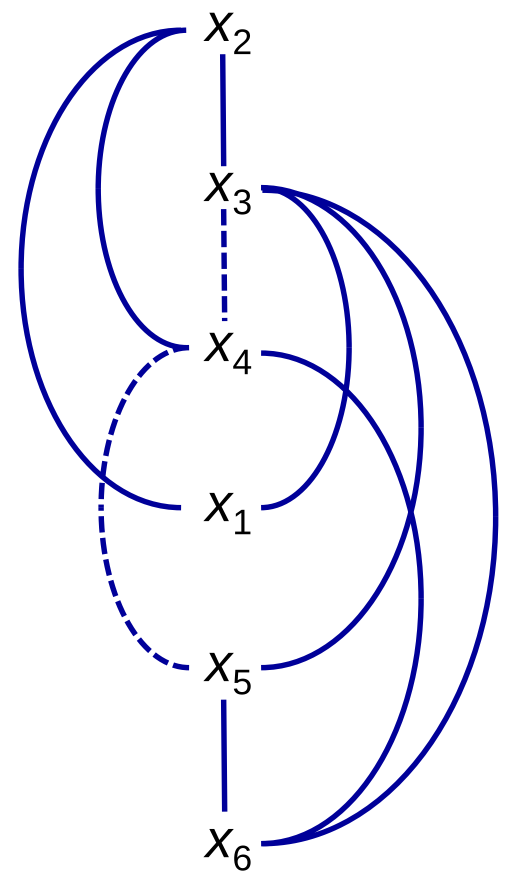
Set Partitioning example

Dependency graph



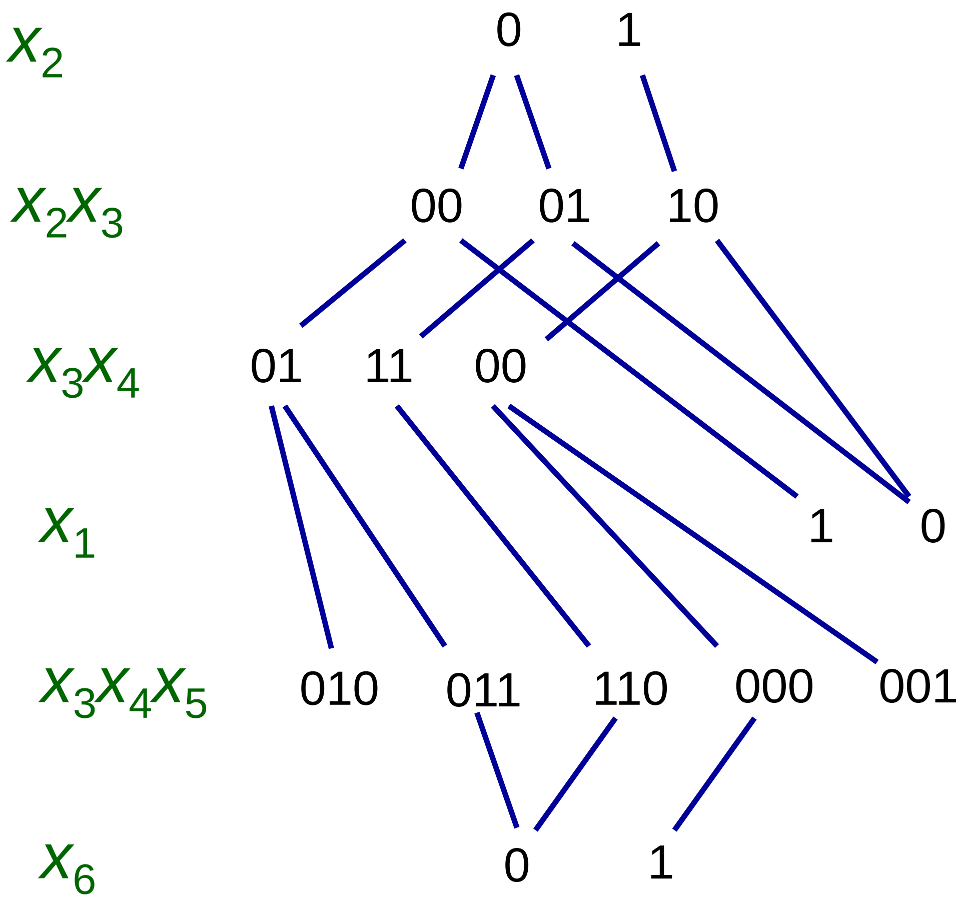
Induced width = 3
(max in-degree)

Enumeration order

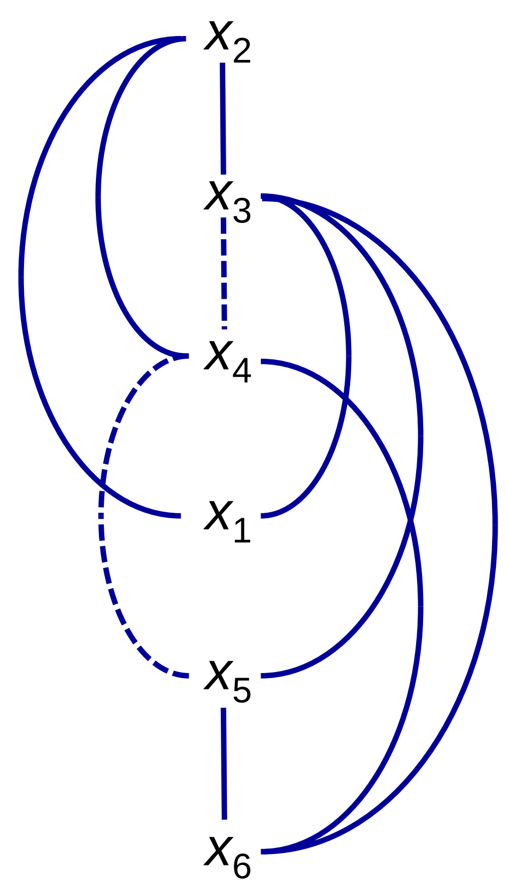


Set Partitioning example

Solution by nonserial DP

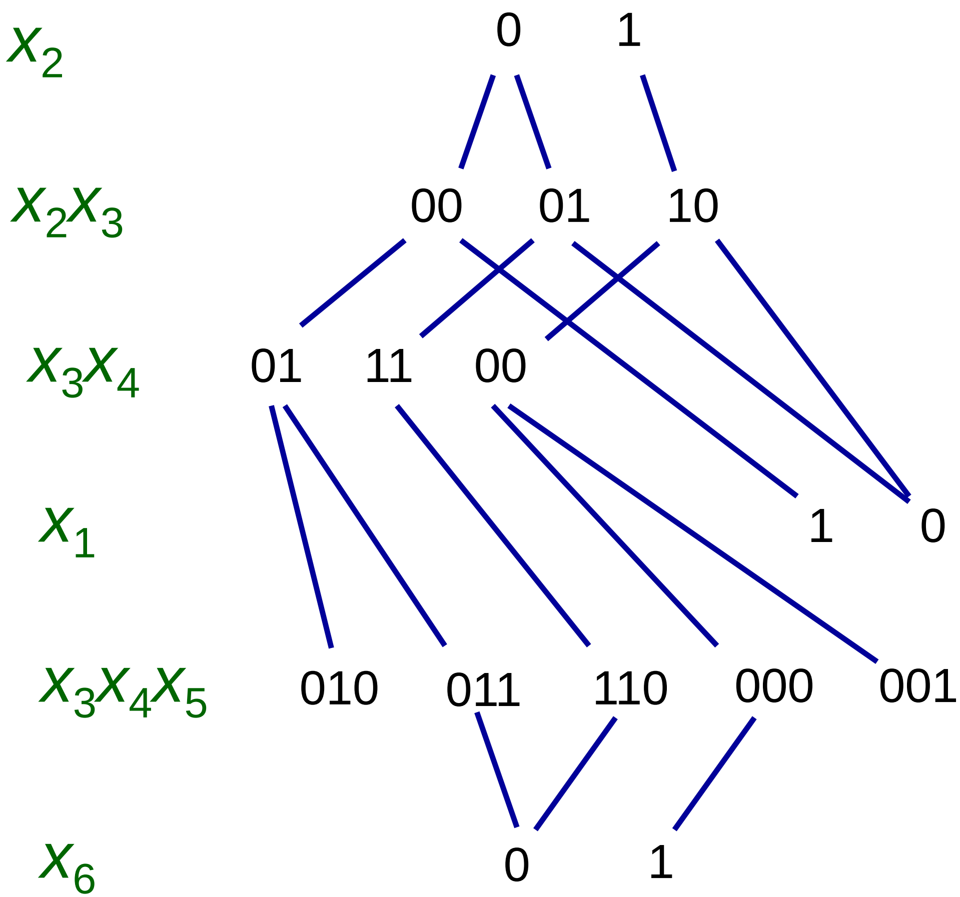


Enumeration order



Set Partitioning example

Solution by nonserial DP

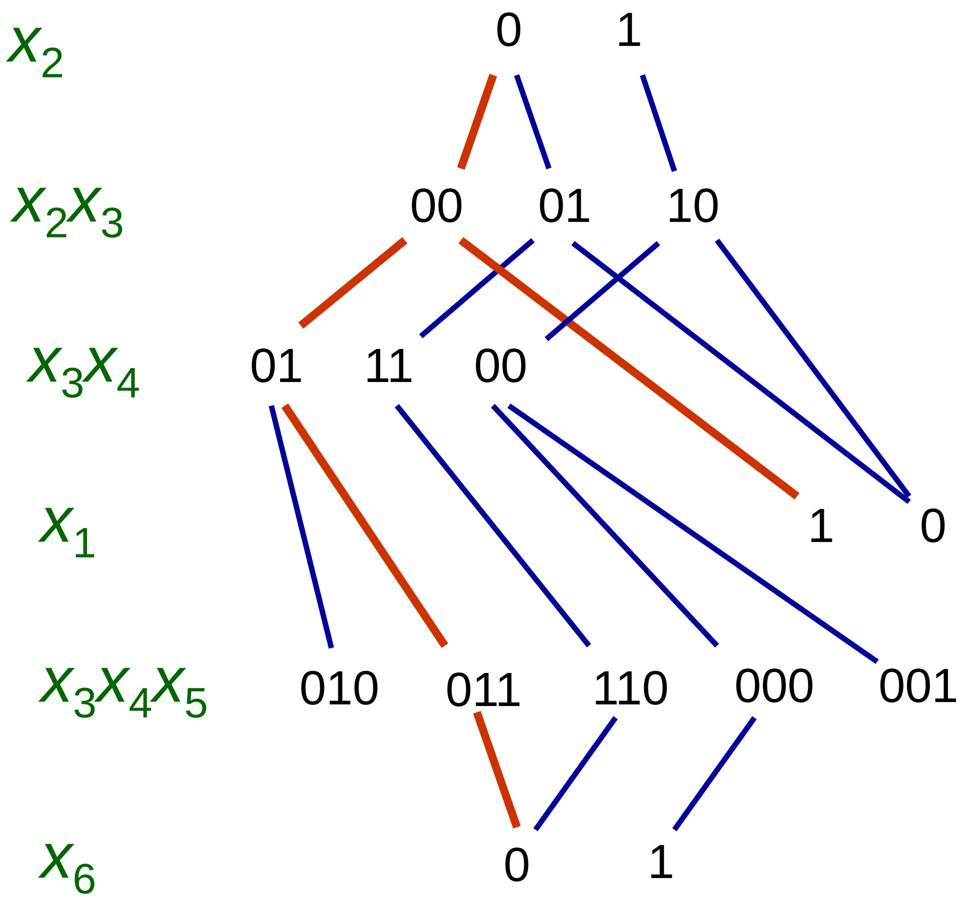


Sets

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

Set Partitioning example

Feasible solution

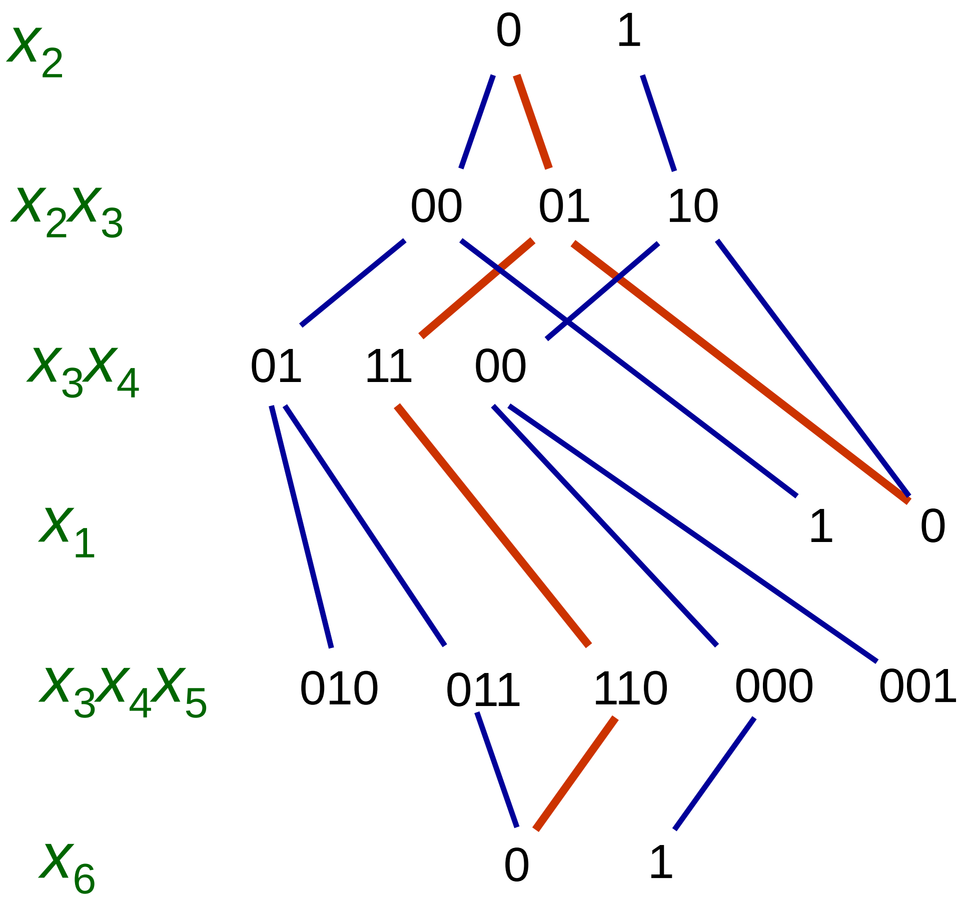


Sets

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

Set Partitioning example

Feasible solution

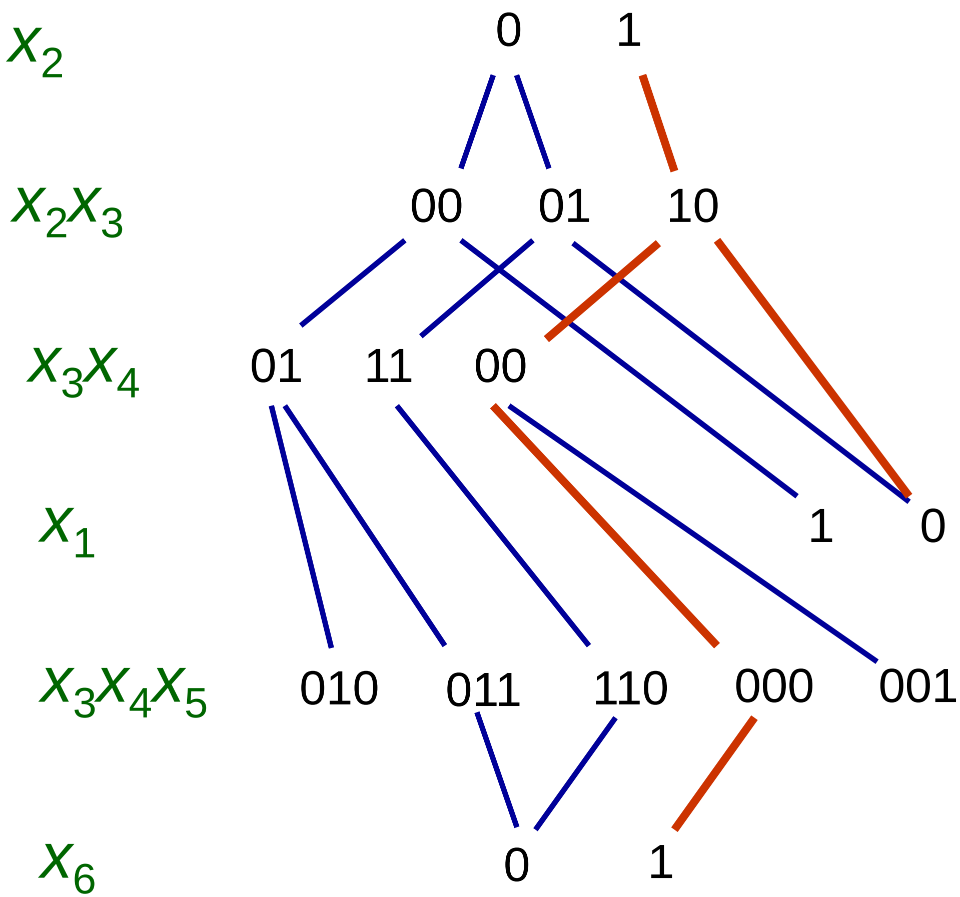


Sets

	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

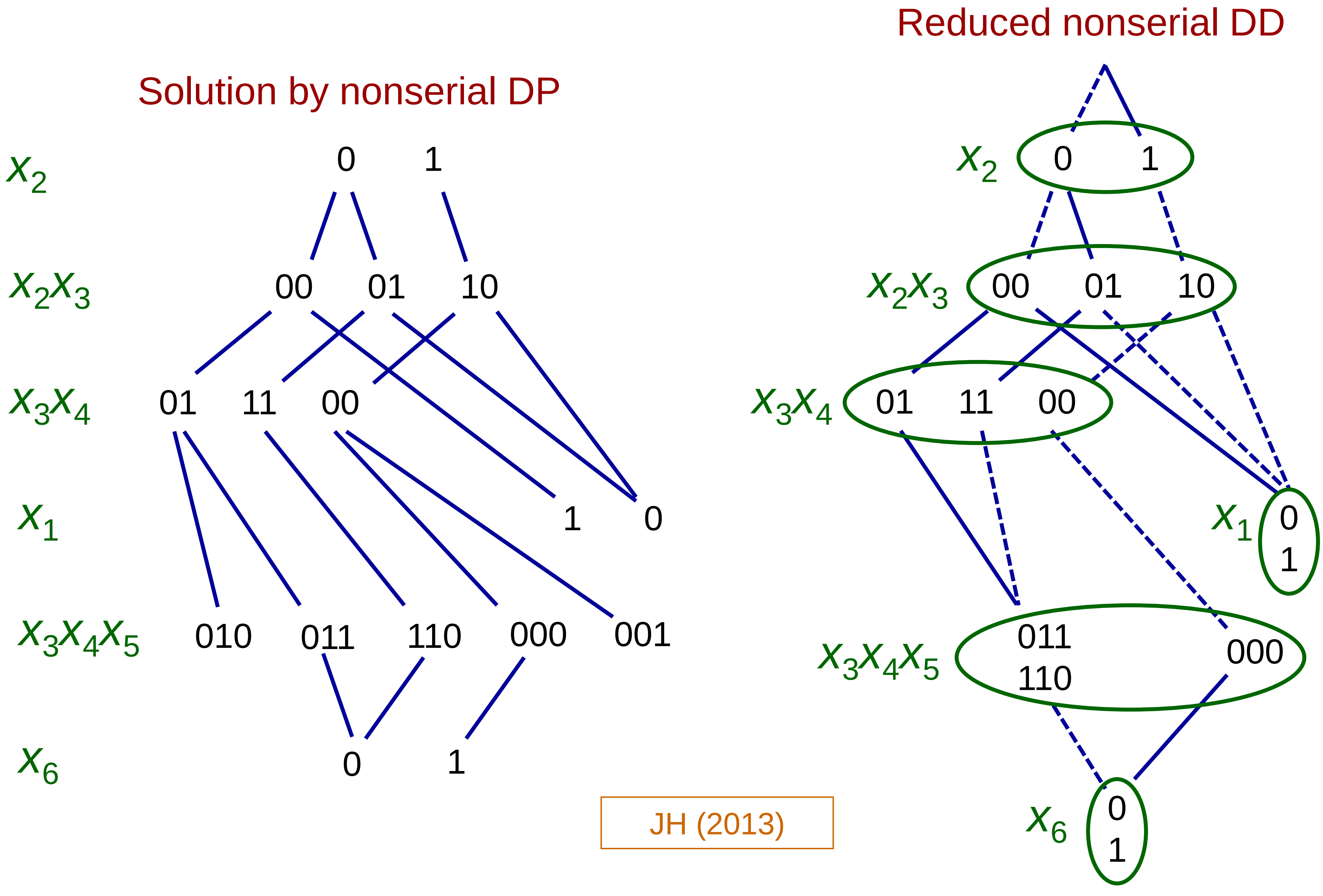


Sets

	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

Construct using **join tree**

Set Partitioning example



JH (2013)

Nonserial Decision Diagrams

- Every technique described here for DDs can be generalized to nonserial DDs.

Other Ongoing Research

- Solving **stochastic DPs** with DDs.
- **Continuous global optimization** with DDs.
- **Cutting planes** from DDs.
- Etc.

Congratulations!

You survived 176 slides!

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 (tech report).
- 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.
- D. Bergman, A. A. Cire, W.-J. van Hoeve, T. Yunes, BDD-based heuristics for binary optimization, *Journal of Heuristics* 20, 211-234, 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, 2016.
- 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.

References

2017

- J. N. Hooker, Job sequencing bounds from decision diagrams, *Proceedings of CP*, LNCS 10416, 565-578, 2017

2018

- T. Serra and J. N. Hooker, Compact representation of near-optimal integer programming solutions, submitted, 2018.
- D. Bergman and A. Cire, Discrete nonlinear decompositions by state-space decompositions, *Management Science*, published online March 2018.
- L. Lozano, D. Bergman, J. C. Smith, On the consistent path problem, submitted 2018.