A Brief Tour of Logic and Optimization

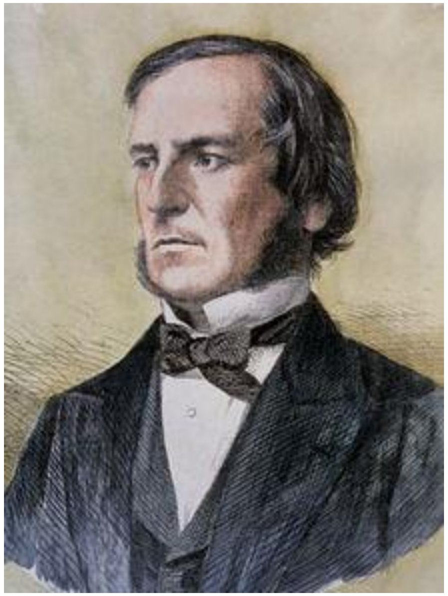
John Hooker
Carnegie Mellon University

INFORMS 2018

Logic and Optimization

There are **deep connections** between logic and optimization, going back at least to George Boole.

Some of these connections can lead to **effective new optimization methods**.

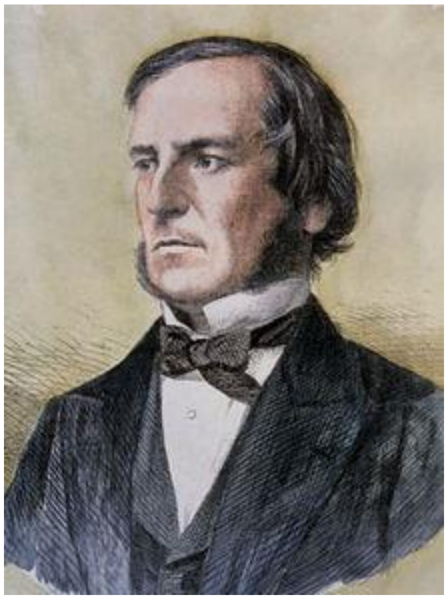


Logic and Optimization

- Probability logic and linear programming
- Decision diagrams and optimization
- Predicate logic and integer programming
- Resolution and cutting planes
- Logic and duality
- Consistency and backtracking

Probability Logic and Linear Programming

Probability Logic and Linear Programming



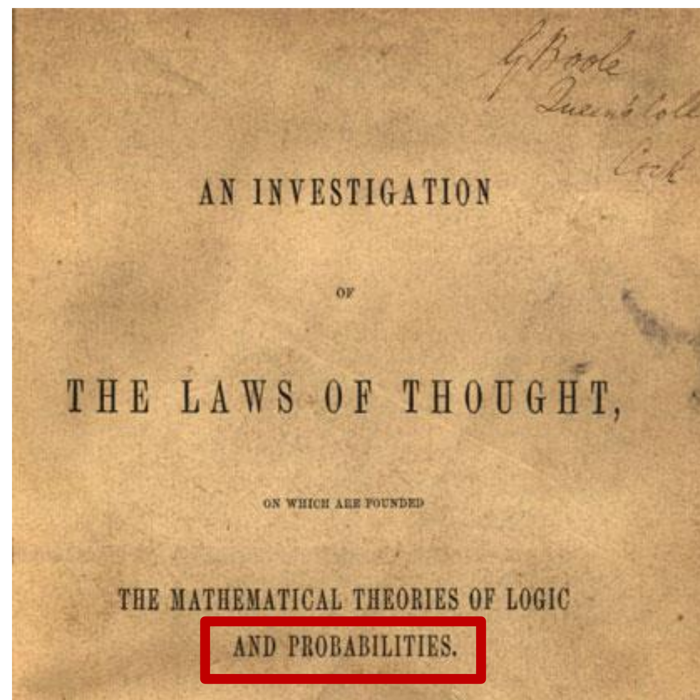
George Boole is best known for Boolean logic.

But he proposed a strikingly original formulation of **reasoning under uncertainty**...

.
…**probability logic.**

It was forgotten or ignored for over 100 years.

Boole 1854



AN INVESTIGATION

OF

THE LAWS OF THOUGHT,

ON WHICH ARE FOUNDED

THE MATHEMATICAL THEORIES OF LOGIC

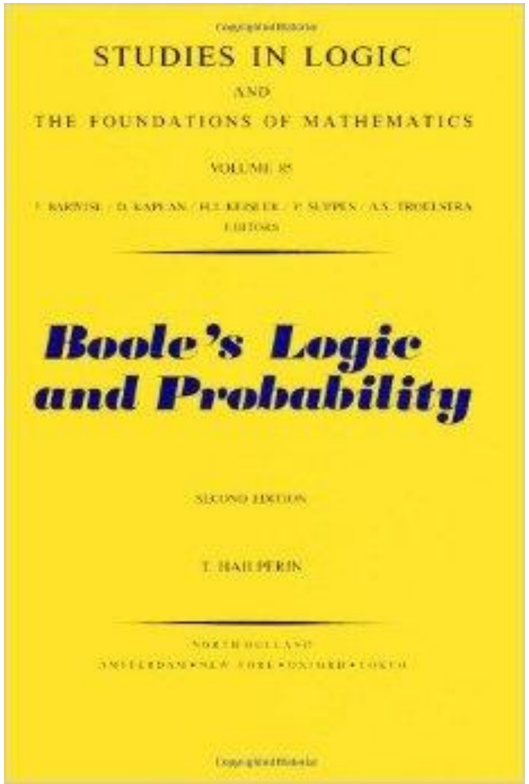
AND PROBABILITIES.

Probability Logic and Linear Programming

In 1970s, Theodore Hailperin showed that probability logic poses a **linear programming** problem.

He sees this as implicit in Boole's own work.

The idea was re-invented by AI community in 1980s.



Hailperin 1976, 1984, 1986

Nilsson 1986

Nils Nilsson

Probability Logic and Linear Programming

Statement	Probability
A	0.9
If A then B	0.8
If B then C	0.4

We can deduce C,
but with what probability?

Boole's insights:

- We can only specify a **range** of probabilities for C.
- The range depends mathematically on the probabilities of **possible states of affairs** (possible worlds).

Probability Logic and Linear Programming

Statement	Probability
A	0.9
If A then B	0.8
If B then C	0.4

Statement	Probability
A	0.9
not-A or B	0.8
not-B or C	0.4

First, interpret the if-then statements as **material conditionals**

Probability Logic and Linear Programming

Statement	Probability
A	0.9
not-A or B	0.8
not-B or C	0.4

Identify the possible outcomes (possible worlds), each having an **unknown** probability.

There are 8 possible worlds:

A	B	C	Prob.
false	false	false	p_{000}
false	false	true	p_{001}
false	true	false	p_{010}
false	true	true	p_{011}
true	false	false	p_{100}
true	false	true	p_{101}
true	true	false	p_{110}
true	true	true	p_{111}

Probability Logic and Linear Programming

Statement	Probability
A	0.9
not-A or B	0.8
not-B or C	0.4

$$\boxed{p_{100} + p_{101} + p_{110} + p_{111} = 0.9}$$

The worlds in which A is true must have probabilities that sum to 0.9.

There are 8 possible worlds:

A	B	C	Prob.
false	false	false	p_{000}
false	false	true	p_{001}
false	true	false	p_{010}
false	true	true	p_{011}
true	false	false	p_{100}
true	false	true	p_{101}
true	true	false	p_{110}
true	true	true	p_{111}

Probability Logic and Linear Programming

Statement	Probability
A	0.9
not-A or B	0.8
not-B or C	0.4

There are 8 possible worlds:

A	B	C	Prob.
false	false	false	p_{000}
false	false	true	p_{001}
false	true	false	p_{010}
false	true	true	p_{011}
true	false	false	p_{100}
true	false	true	p_{101}
true	true	false	p_{110}
true	true	true	p_{111}

$$p_{100} + p_{101} + p_{110} + p_{111} = 0.9$$

$$\boxed{p_{000} + p_{001} + p_{010} + p_{011} + p_{110} + p_{111} = 0.8}$$

Probability Logic and Linear Programming

Statement	Probability
A	0.9
not-A or B	0.8
not-B or C	0.4

$$p_{100} + p_{101} + p_{110} + p_{111} = 0.9$$

$$p_{000} + p_{001} + p_{010} + p_{011} + p_{110} + p_{111} = 0.8$$

$$\boxed{p_{000} + p_{001} + p_{011} + p_{100} + p_{101} + p_{111} = 0.4}$$

There are 8 possible worlds:

A	B	C	Prob.
false	false	false	p_{000}
false	false	true	p_{001}
false	true	false	p_{010}
false	true	true	p_{011}
true	false	false	p_{100}
true	false	true	p_{101}
true	true	false	p_{110}
true	true	true	p_{111}

Probability Logic and Linear Programming

Statement	Probability
A	0.9
not-A or B	0.8
not-B or C	0.4

$$p_{100} + p_{101} + p_{110} + p_{111} = 0.9$$

$$p_{000} + p_{001} + p_{010} + p_{011} + p_{110} + p_{111} = 0.8$$

$$p_{000} + p_{001} + p_{011} + p_{100} + p_{101} + p_{111} = 0.4$$

$$\boxed{p_{000} + \ldots + p_{111} = 1}$$

There are 8 possible worlds:

A	B	C	Prob.
false	false	false	p_{000}
false	false	true	p_{001}
false	true	false	p_{010}
false	true	true	p_{011}
true	false	false	p_{100}
true	false	true	p_{101}
true	true	false	p_{110}
true	true	true	p_{111}

Probability Logic and Linear Programming

Statement	Probability
A	0.9
not-A or B	0.8
not-B or C	0.4

$$p_{100} + p_{101} + p_{110} + p_{111} = 0.9$$

$$p_{000} + p_{001} + p_{010} + p_{011} + p_{110} + p_{111} = 0.8$$

$$p_{000} + p_{001} + p_{011} + p_{100} + p_{101} + p_{111} = 0.4$$

$$p_{000} + \ldots + p_{111} = 1$$

Minimize and maximize probability of C:

$$\boxed{p_{001} + p_{011} + p_{101} + p_{111}}$$

subject to these equations and $p_{ijk} \geq 0$

There are 8 possible worlds:

A	B	C	Prob.
false	false	false	p_{000}
false	false	true	p_{001}
false	true	false	p_{010}
false	true	true	p_{011}
true	false	false	p_{100}
true	false	true	p_{101}
true	true	false	p_{110}
true	true	true	p_{111}

Probability Logic and Linear Programming

Statement	Probability
A	0.9
not-A or B	0.8
not-B or C	0.4

This is a **linear programming** problem.

The result is a **range** of probabilities for C: 0.1 to 0.4

$$p_{100} + p_{101} + p_{110} + p_{111} = 0.9$$

$$p_{000} + p_{001} + p_{010} + p_{011} + p_{110} + p_{111} = 0.8$$

$$p_{000} + p_{001} + p_{011} + p_{100} + p_{101} + p_{111} = 0.4$$

$$p_{000} + \dots + p_{111} = 1$$

Minimize and maximize probability of C:

$$\boxed{p_{001} + p_{011} + p_{101} + p_{111}}$$

subject to these equations and $p_{ijk} \geq 0$

Probability Logic and Linear Programming

Statement	Probability
A	0.9
not-A or B	0.8
not-B or C	0.4

$$p_{100} + p_{101} + p_{110} + p_{111} = 0.9$$

$$p_{000} + p_{001} + p_{010} + p_{011} + p_{110} + p_{111} = 0.8$$

$$p_{000} + p_{001} + p_{011} + p_{100} + p_{101} + p_{111} = 0.4$$

$$p_{000} + \ldots + p_{111} = 1$$

Minimize and maximize probability of C:

$$\boxed{p_{001} + p_{011} + p_{101} + p_{111}}$$

subject to these equations and $p_{ijk} \geq 0$

This is a **linear programming** problem.

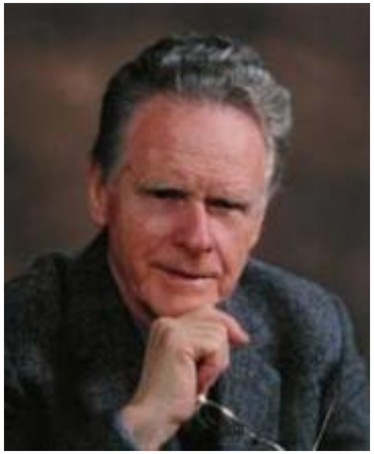
The result is a **range** of probabilities for C: 0.1 to 0.4

Large instances solved by **column generation.**

Probability Logic and Linear Programming

There are linear programming models for **logics of belief and evidence** such as **Dempster-Shafer theory** and related systems.

Dempster 1968, Shafer 1976



A. P. Dempster

Glenn Shafer

Decision Diagrams and Optimization

Decision Diagrams and Optimization

Boolean logic was also forgotten for decades, except in the minds of a few logicians, including philosopher **Charles Sanders Pearce**.

Pearce saw that Boolean logic could be represented by **switching circuits.**

C. S. Pearce

Pearce 1886

Decision Diagrams and Optimization

Boolean logic was also forgotten for decades, except in the minds of a few logicians, including philosopher **Charles Sanders Pearce**.

Pearce saw that Boolean logic could be represented by **switching circuits.**

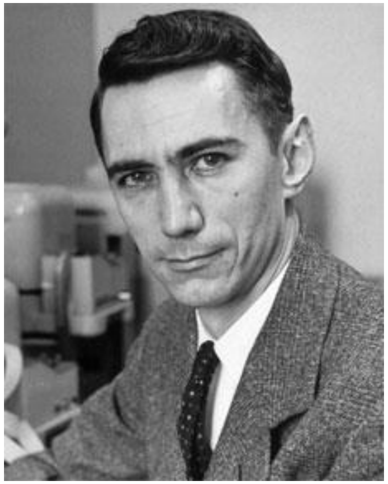
Claude Shannon was required to take a philosophy course while an undergraduate at the University of Michigan, where he was exposed to Pearce's work.

C. S. Pearce

This gave rise to his famous master's thesis , *A Symbolic Analysis of Relay and Switching Circuits*, which provided the basis of modern computing.

Pearce 1886

Shannon 1940



C. Shannon

Decision Diagrams and Optimization

C. Y. Lee proposed **binary-decision programs** as a means of calculating the output of switching circuits.

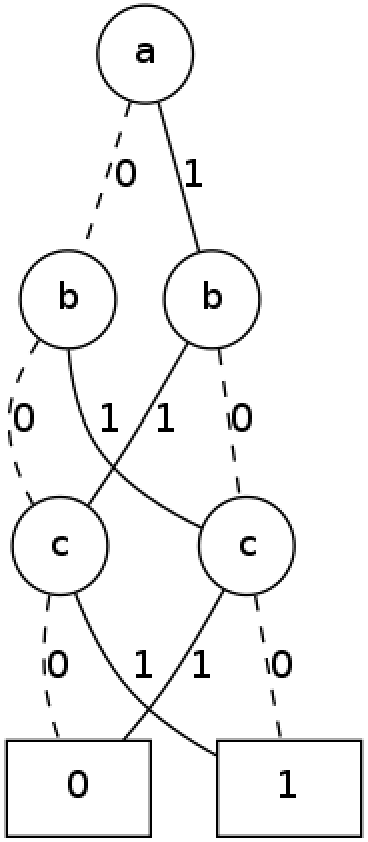
Lee 1959

S. B. Akers represented binary-decision programs with **binary decision diagrams**.

Akers1978

R. E. Bryant showed that **ordered BDDs** provide a unique minimal representation of a Boolean function.

Bryant 1986



Ordered BDD

Decision Diagrams and Optimization

BDDs have long been used for logic circuit design and product configuration.

They were recently adapted to **optimization** and **constraint programming**.

Hadžić and JH (2006, 2007)

Andersen, Hadžić, JH and Tiedemann (2007)

There are at least **12 talks** on DDs and optimization **at this meeting**.

Tarik Hadžić

Henrik Reif Andersen

Decision Diagrams and Optimization

Search
with a novel branch-and-bound method

Modeling
with recursive formulations

Primal heuristics
with restricted diagrams

Optimization

Relaxation
with relaxed diagrams

Constraint propagation
through a relaxed diagram

Postoptimality analysis
with sound diagrams

Decision Diagrams and Optimization

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

Bryant (1986)

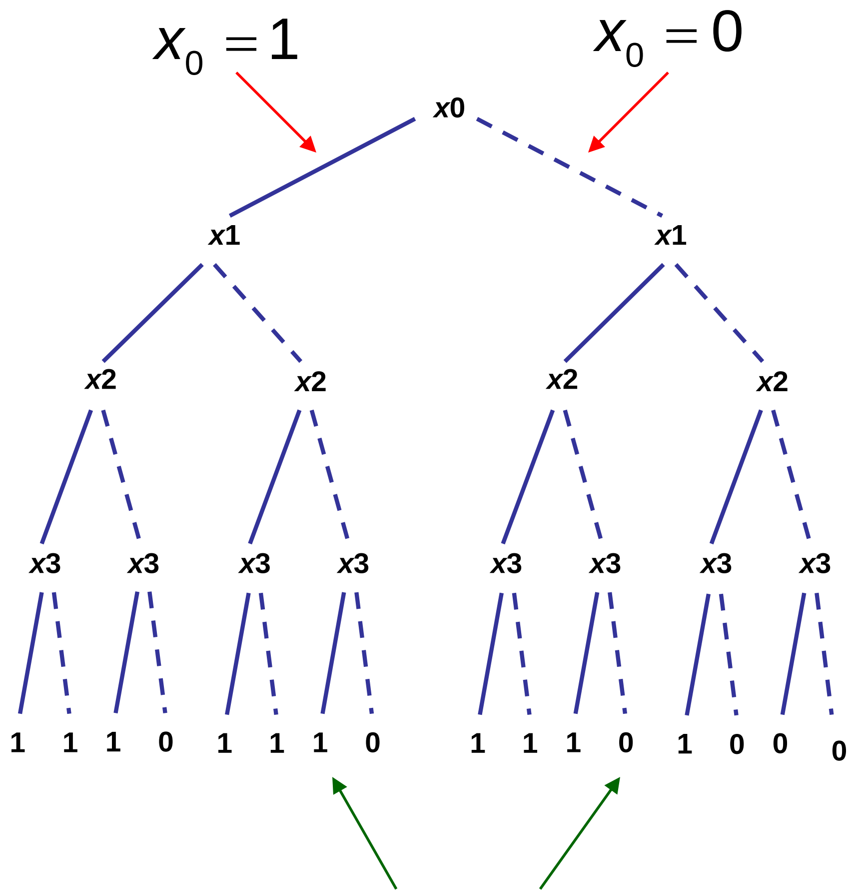
Randy Bryant

The reduced DD can be viewed as a branching tree with **redundancy removed**.

Superimpose isomorphic subtrees and remove redundant nodes.

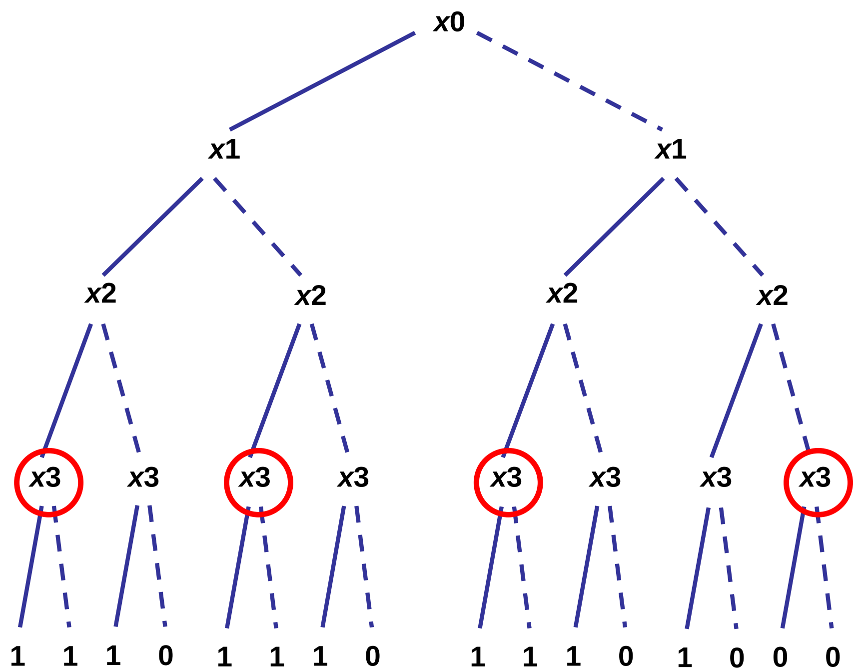
Branching tree for 0-1 inequality

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



1 indicates feasible solution,
0 infeasible

1

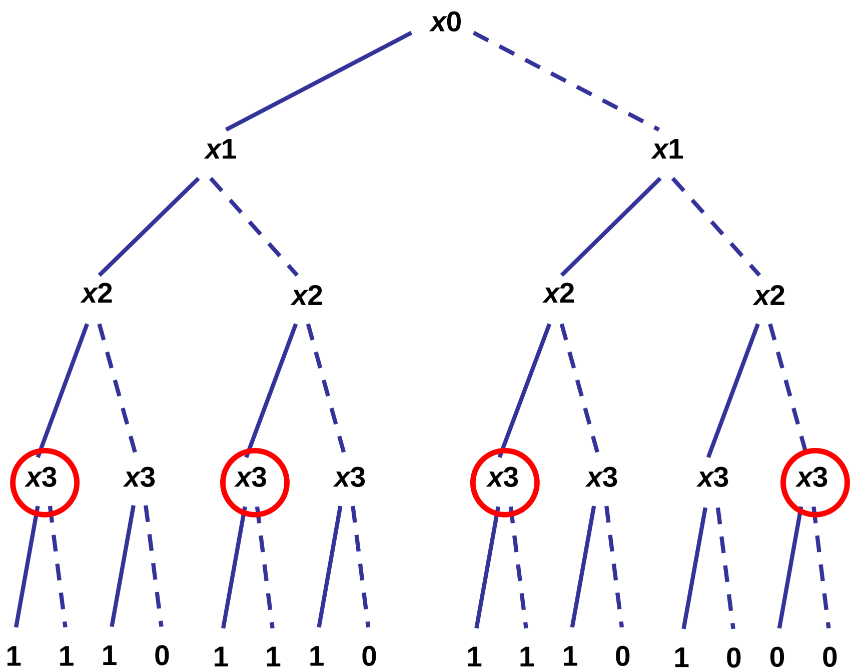


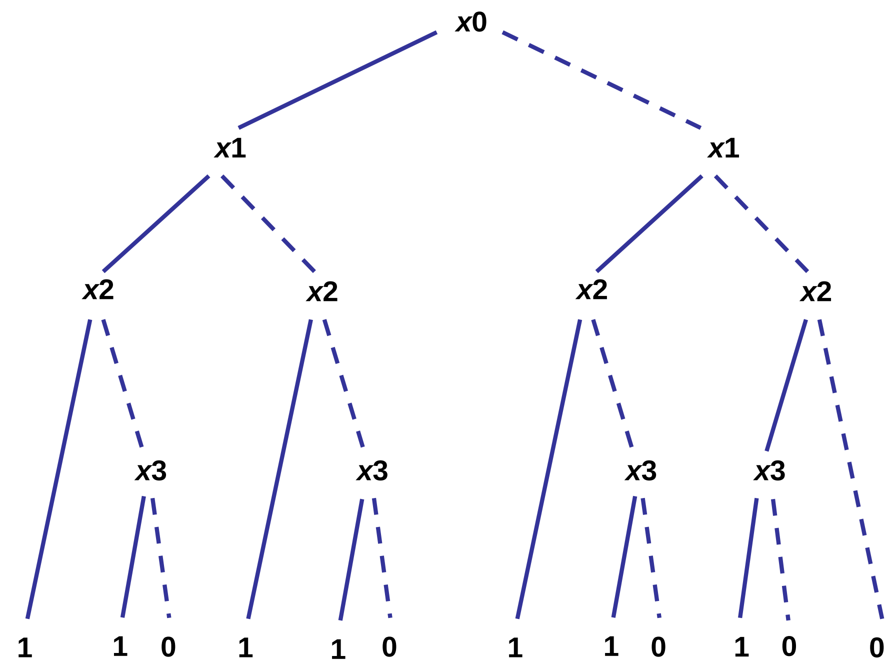
Branching tree for 0-1 inequality

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

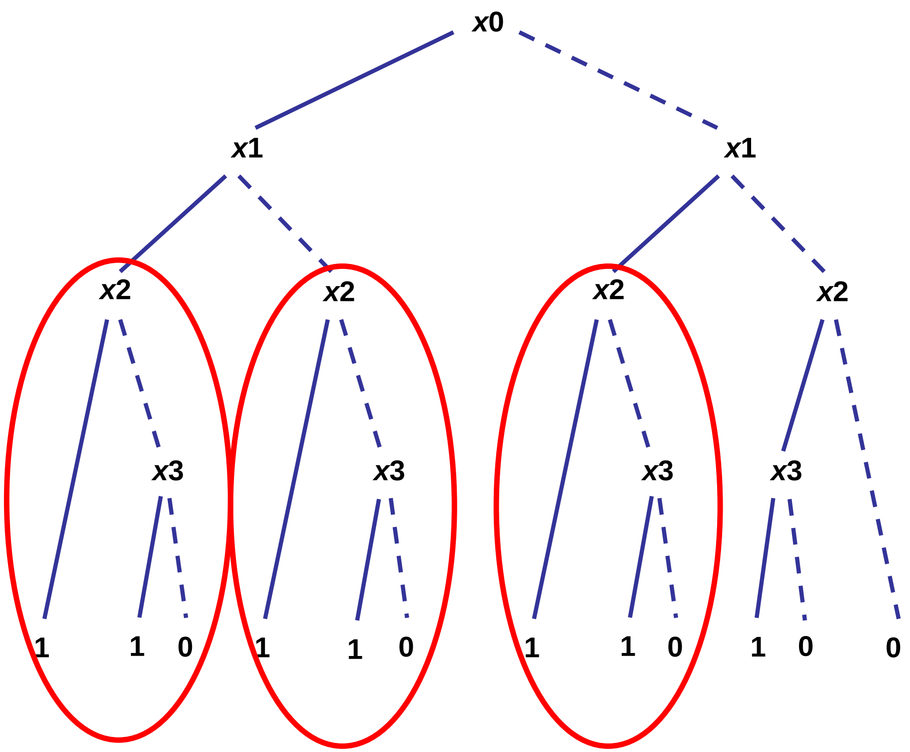
Remove redundant nodes…

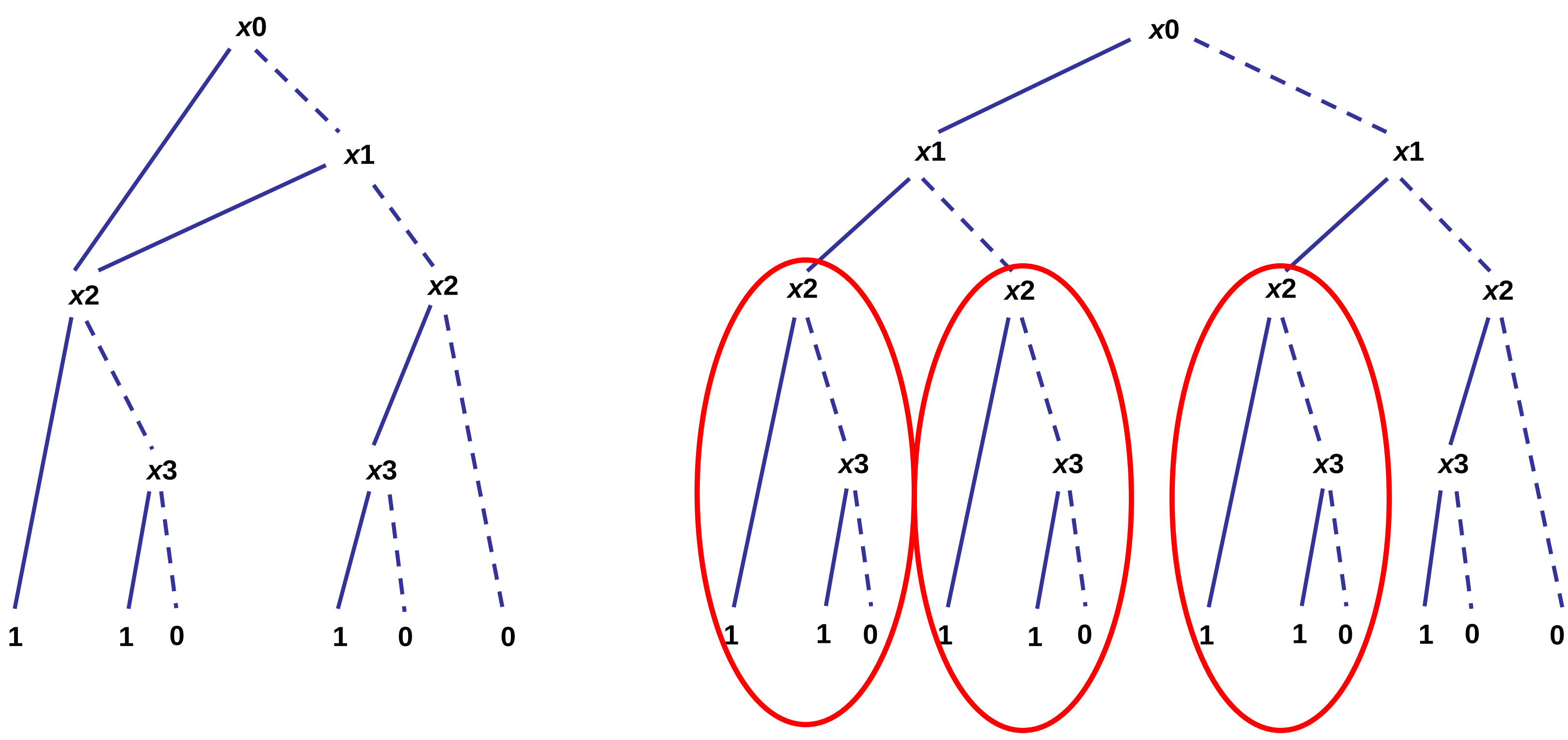
1

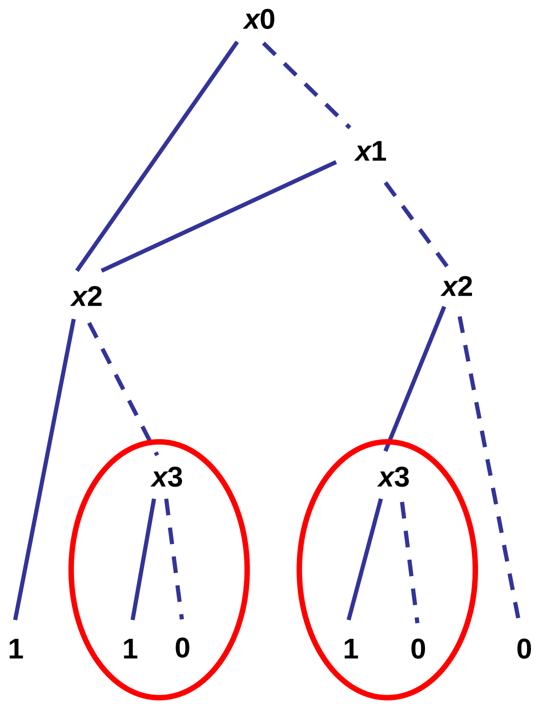

x0
x1
x1
x2
x2
x2
x2
x3
x3
x3
x3
x3
x3
x3
x3
1
1
1
0
1
1
1
0
1
1
1
0
1
0
0
0


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

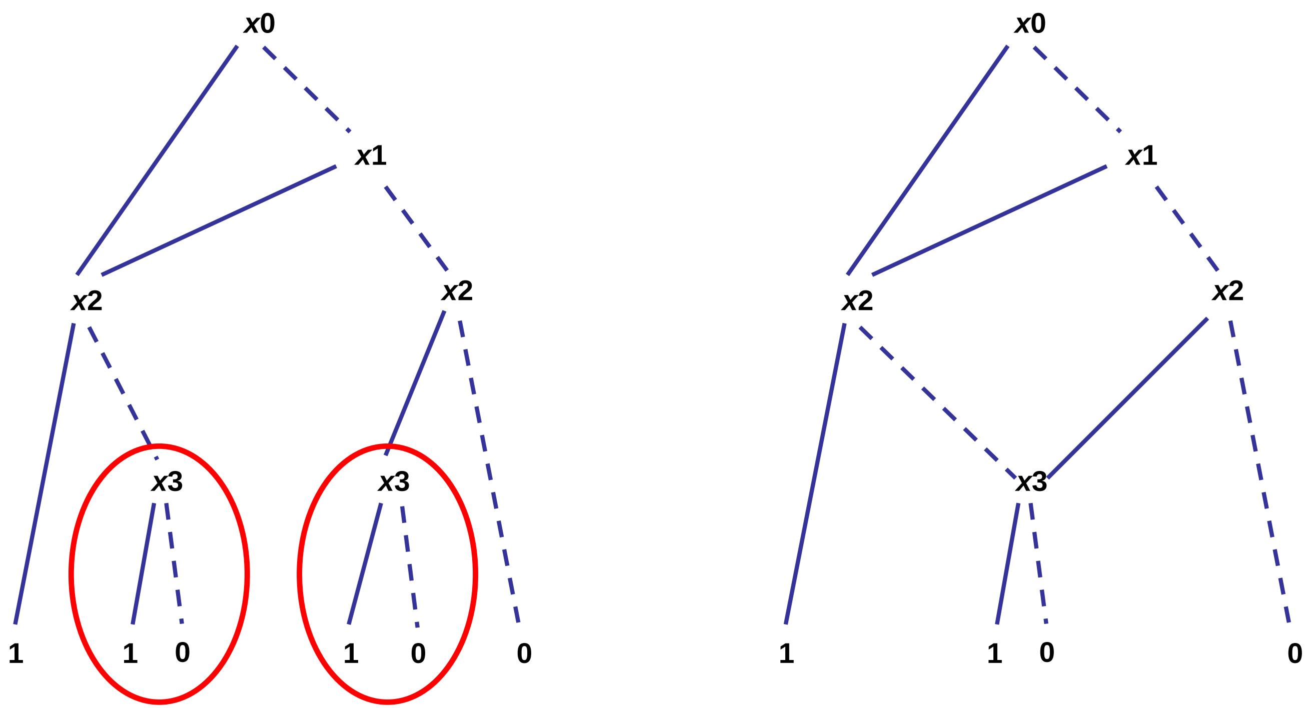
Superimpose identical subtrees…




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

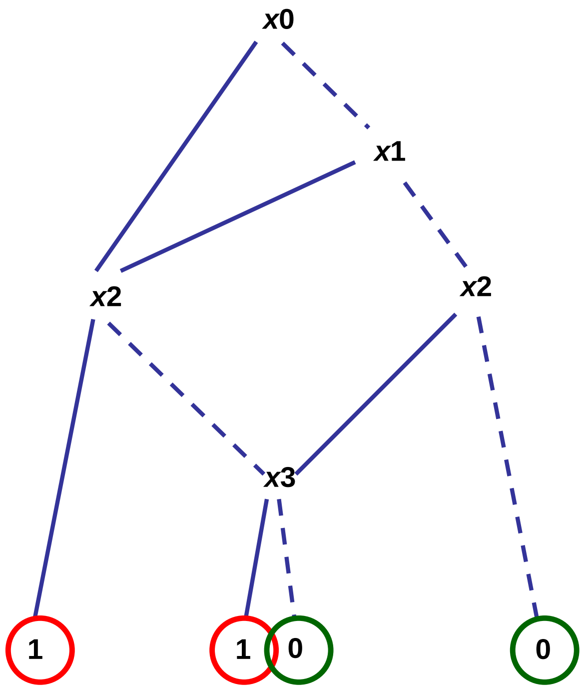


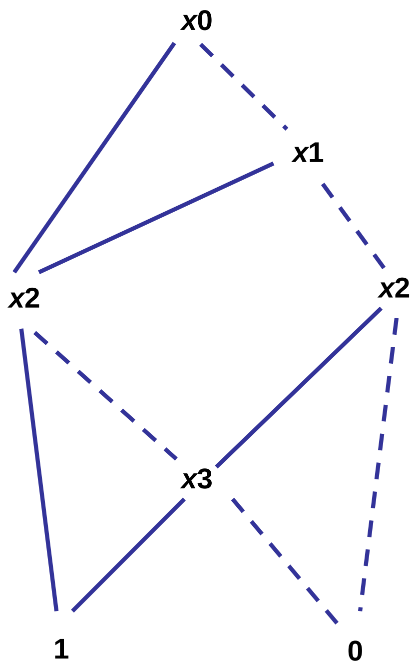
Superimpose identical subtrees…

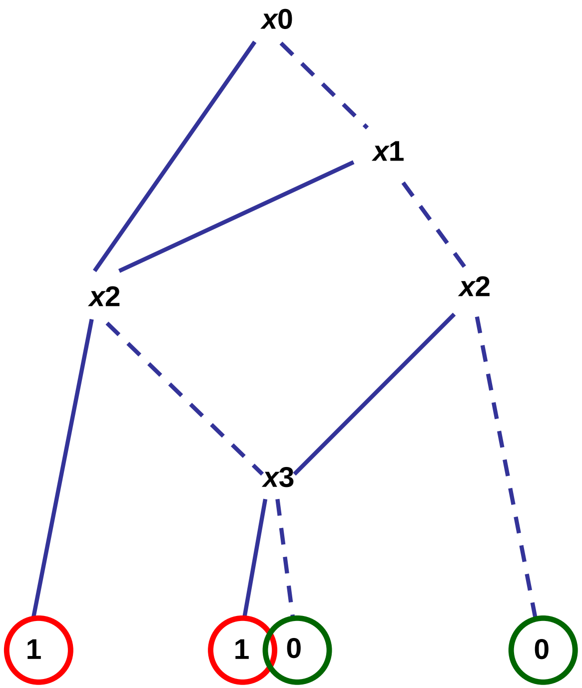


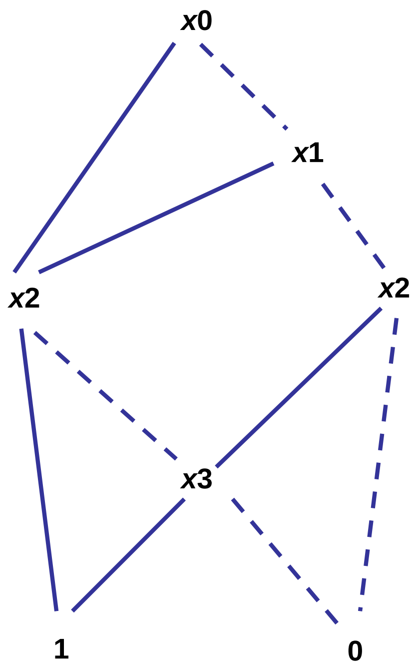
x0
x1
x2
x2
x3
x3
1
1
0
1
0
0
x0
x1
x2
x2
x3
1
1
0
0

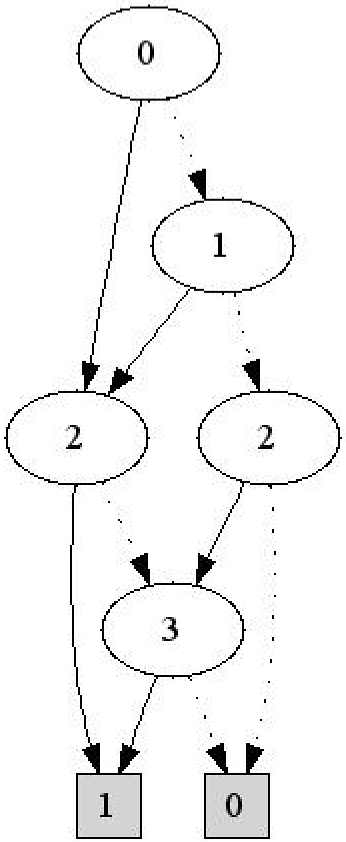
Superimpose identical leaf nodes…




x0
x1
x2
x2
x3
1
0


x0
x1
x2
x2
x3
1
1
0
0





as generated by software

Decision Diagrams and Optimization

Decision diagrams can represent the **feasible set** of an optimization problem.

- Remove paths to 0.
- Paths to 1 are feasible solutions.
- Associate costs with arcs.
- Reduces optimization to a shortest (longest) path problem

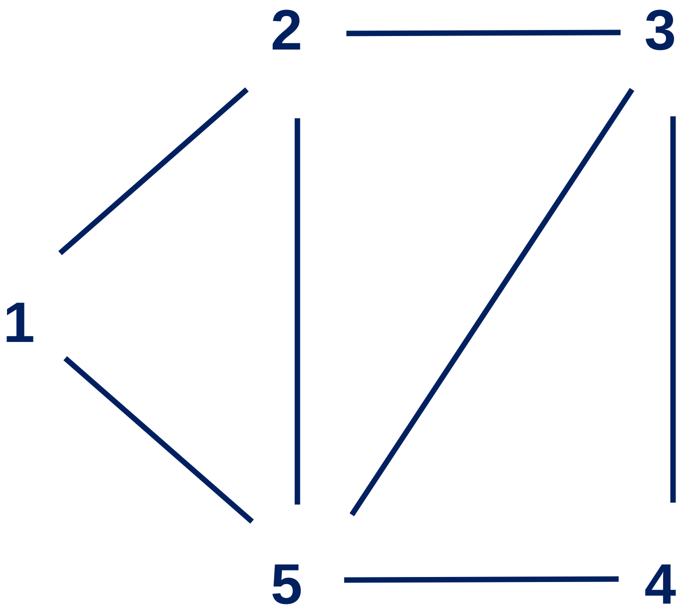
We illustrate with the **stable set problem** (max independent set).

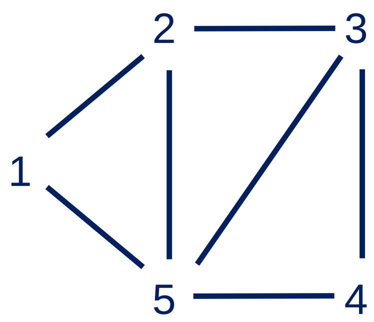
Stable Set Problem

Let each vertex have weight w_i

Let $x_i = 1$ when vertex i is in stable set

Select nonadjacent vertices to maximize $\sum_i w_i x_i$





{12345}

$x1$

$x2$

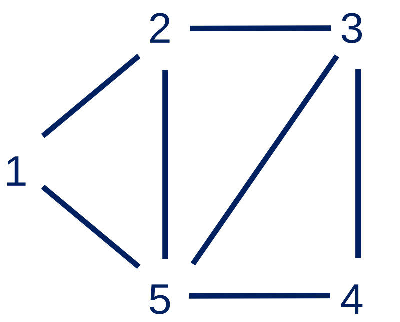
$x3$

$x4$

$x5$

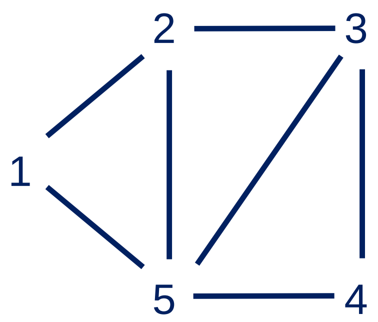
Exact DD for stable set problem

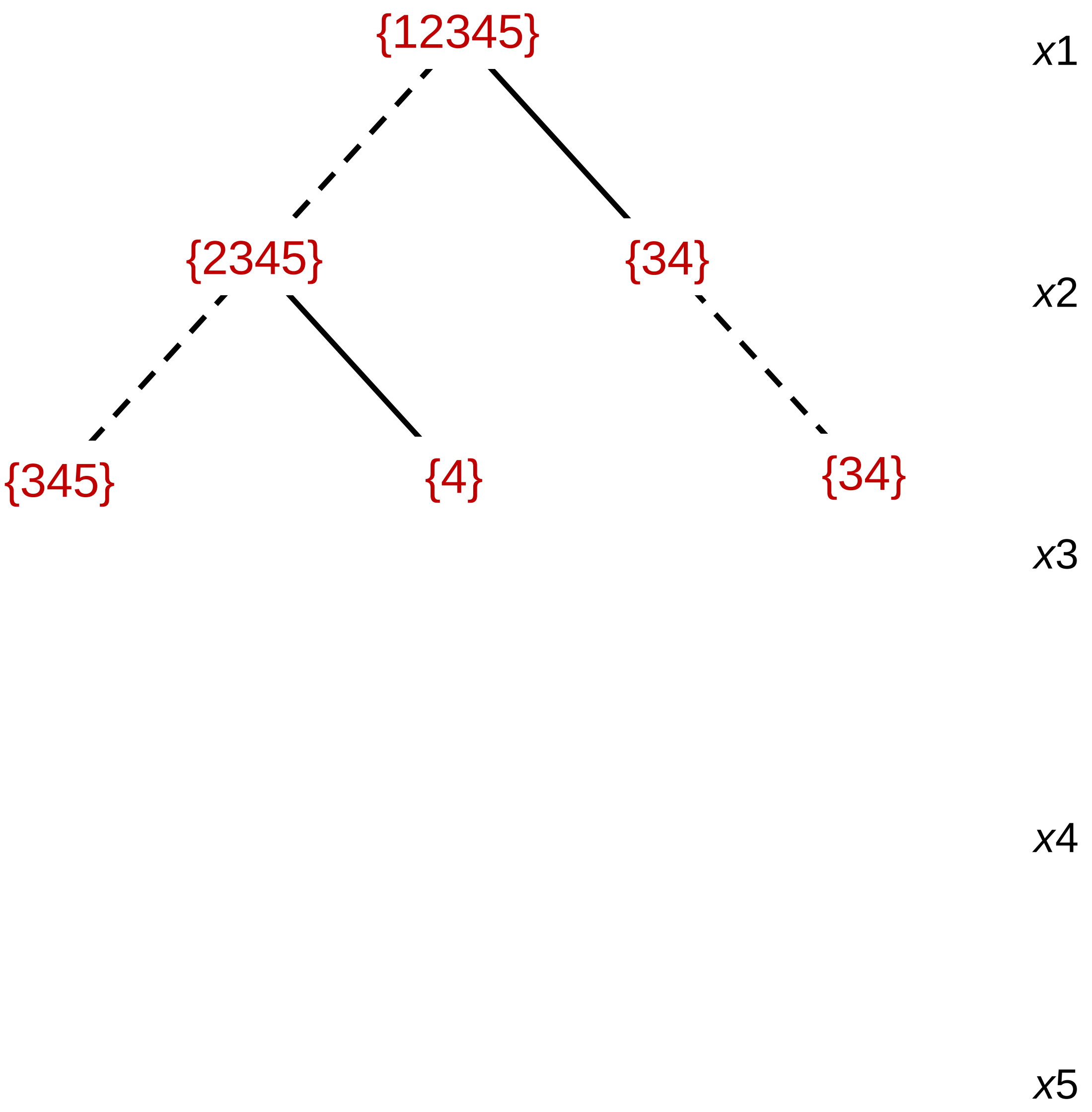
To build DD, associate state with each node



Exact DD for
stable set problem

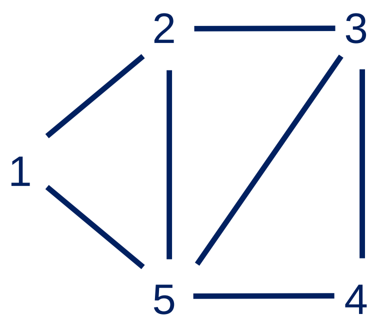
To build DD,
associate state
with each node

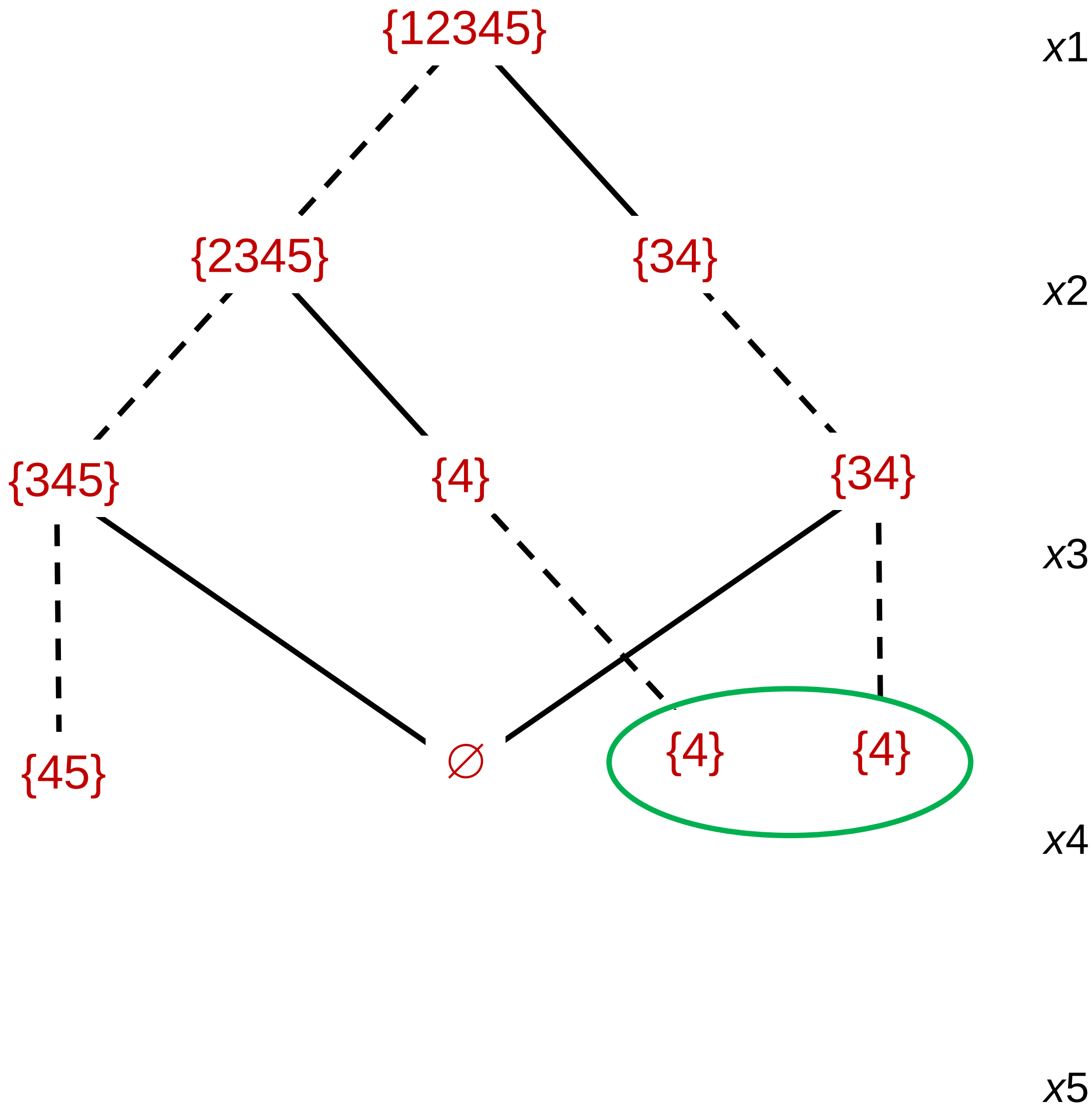




Exact DD for
stable set problem

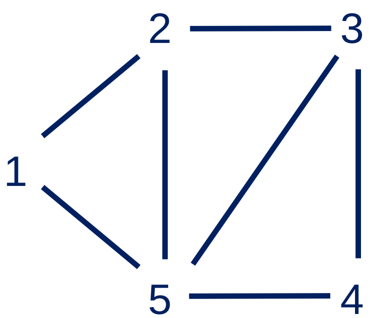
To build DD,
associate state
with each node



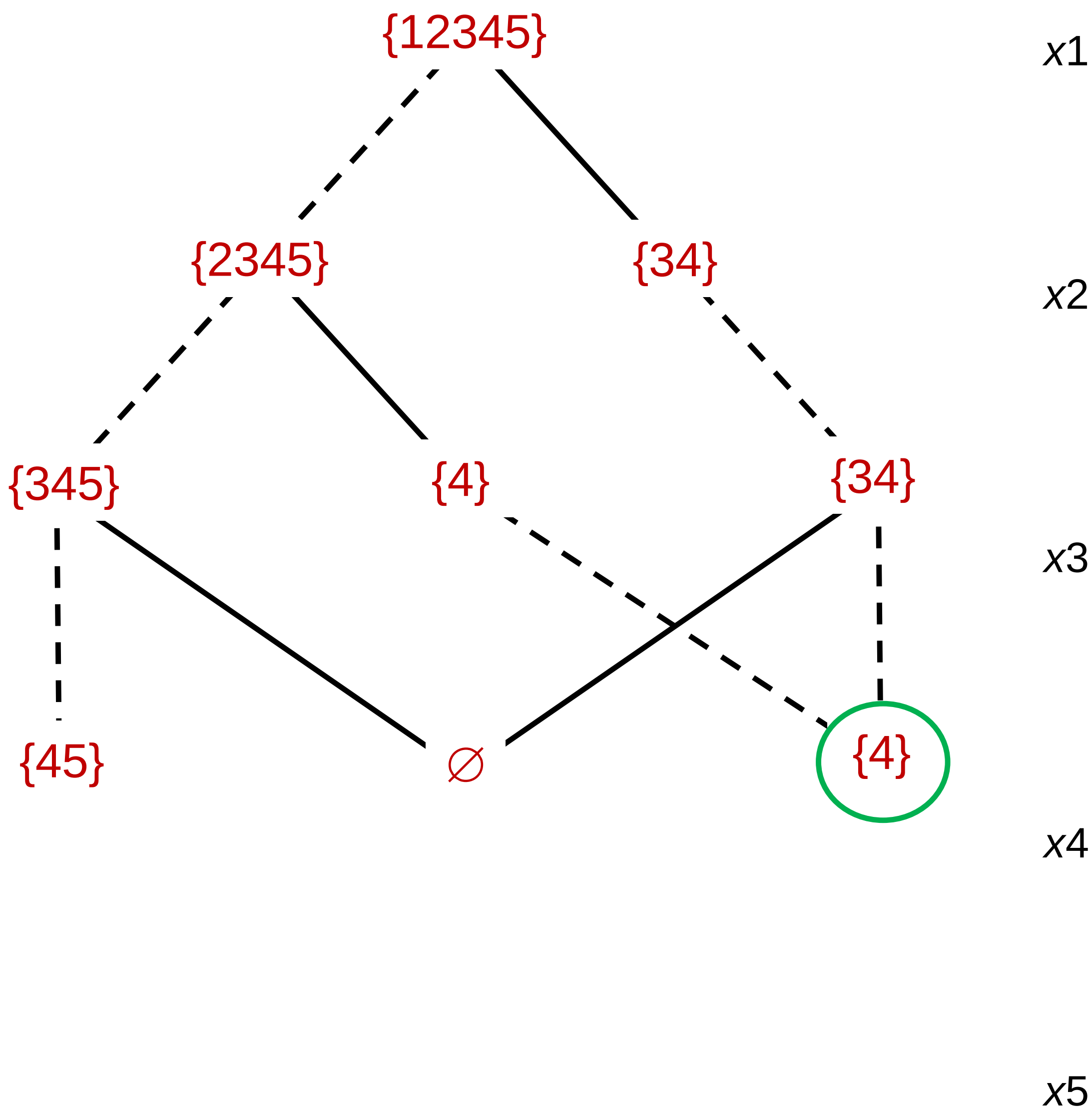


Exact DD for
stable set problem

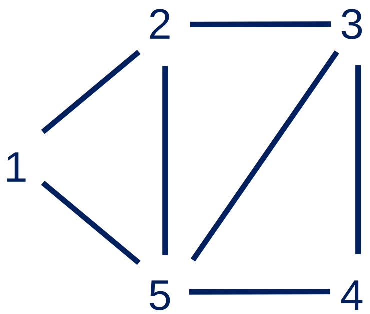
Merge nodes that
correspond to the
same state



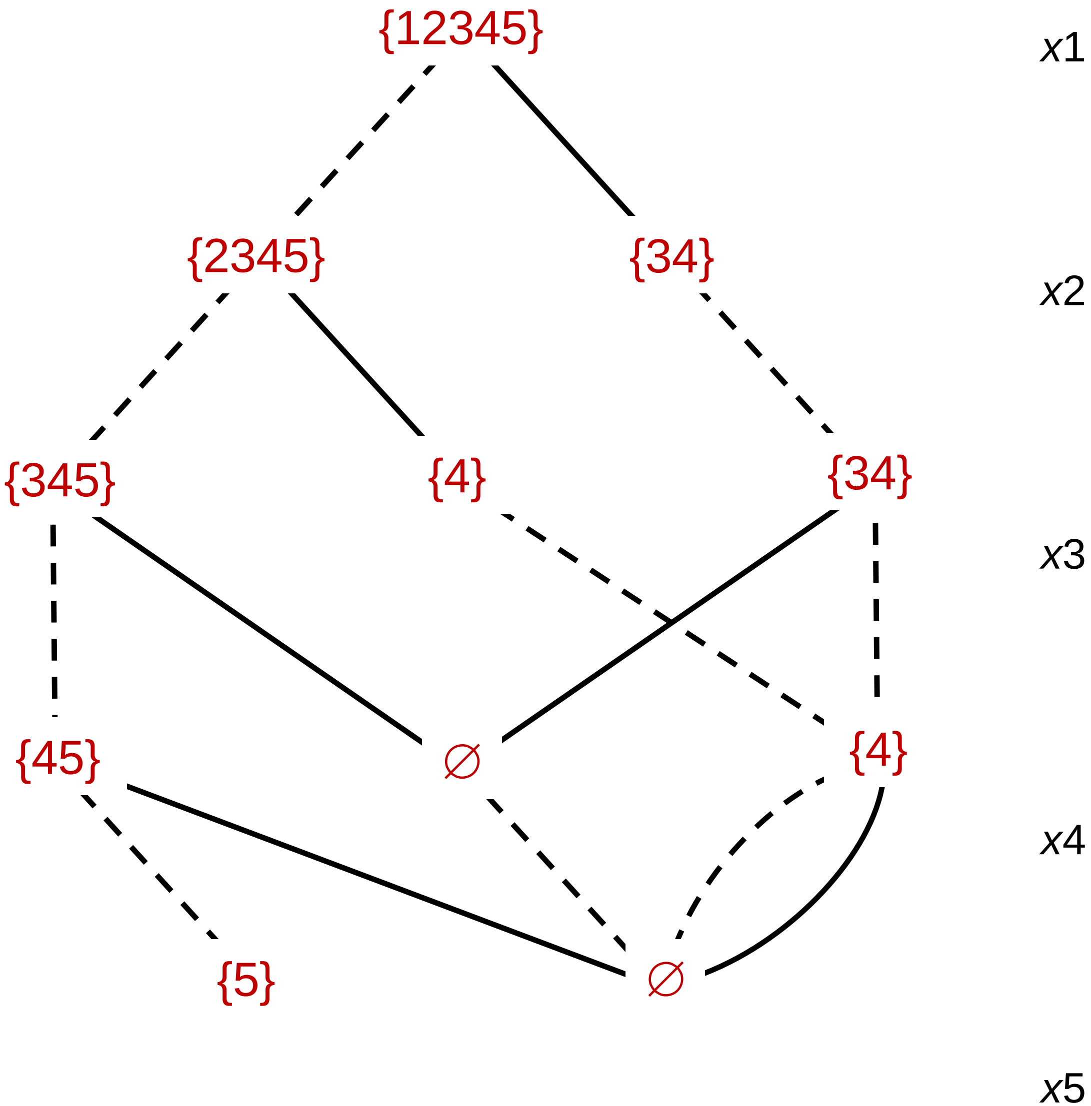
Exact DD for
stable set problem



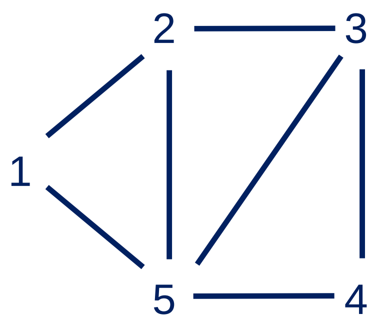
Merge nodes that
correspond to the
same state



Exact DD for
stable set problem



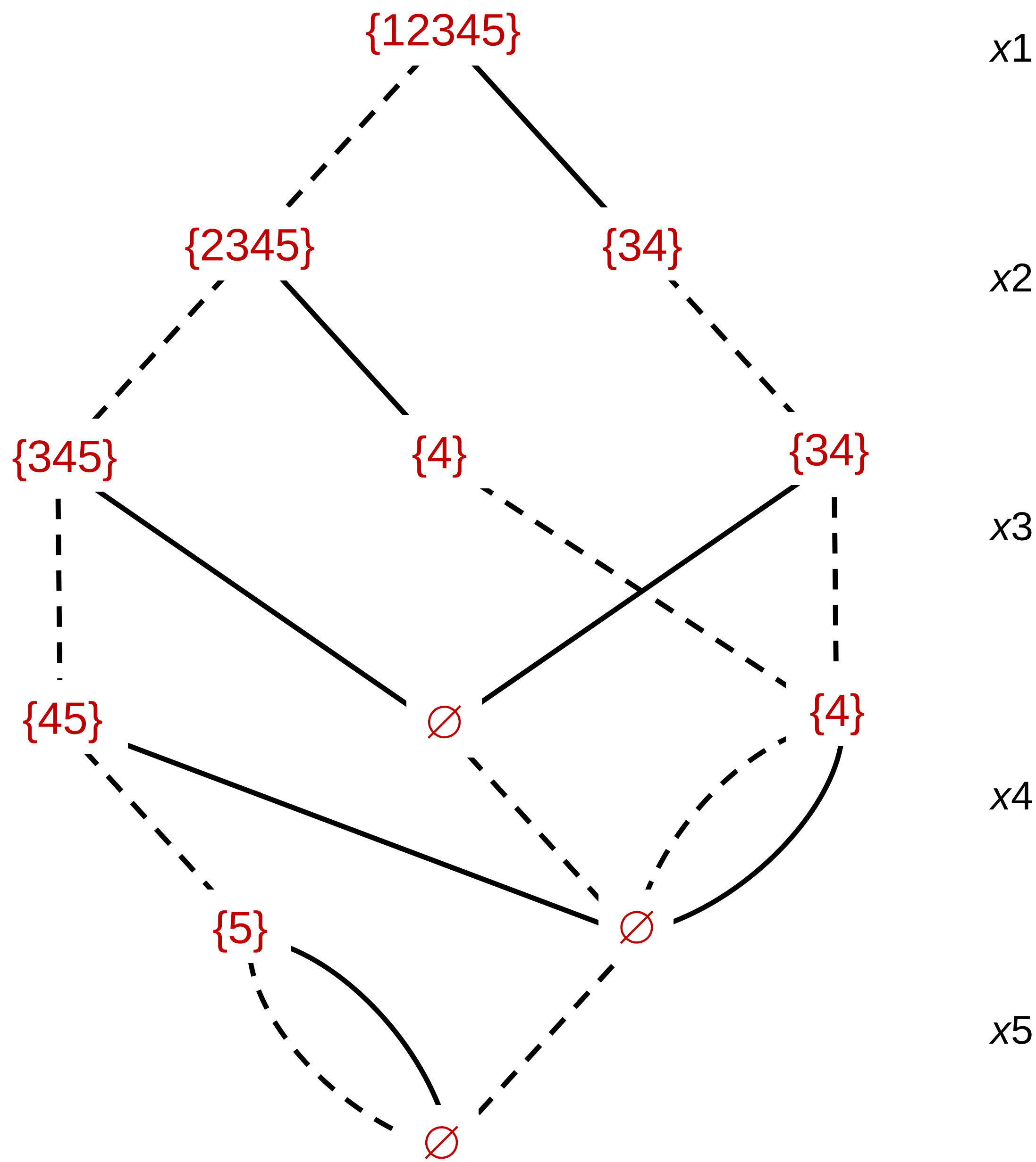
To build DD,
associate state
with each node



Exact DD for
stable set problem

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

DD reduction is a
more powerful
simplification
method than DP



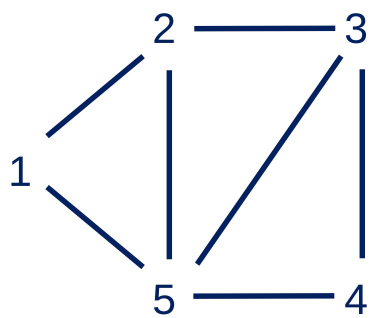
Decision Diagrams and Optimization

Relaxed DDs are essential for obtaining optimization bounds.

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 and Tiedemann (2007)



{12345}

$x1$

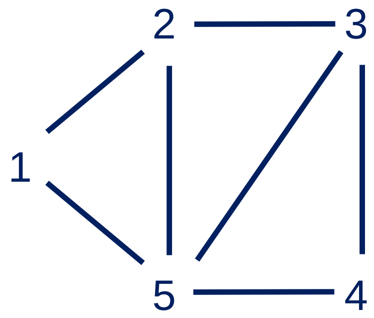
$x2$

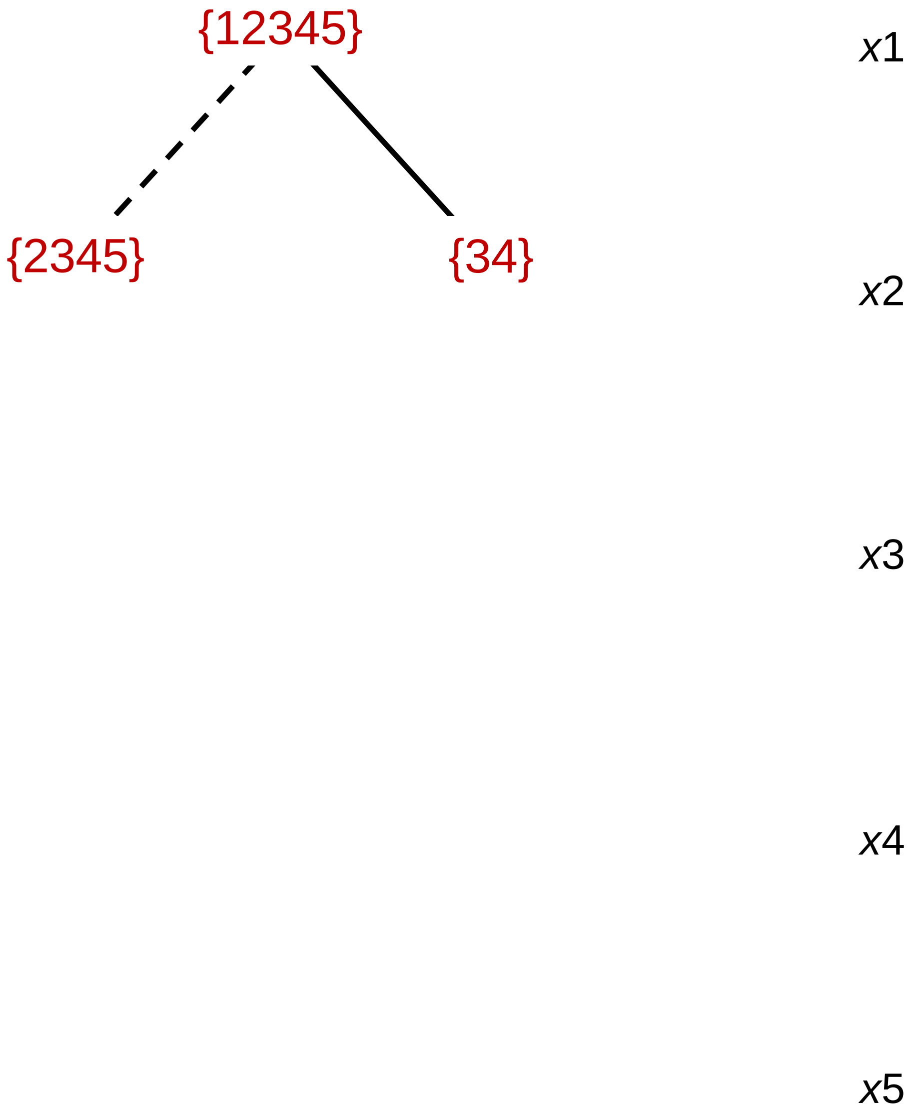
$x3$

$x4$

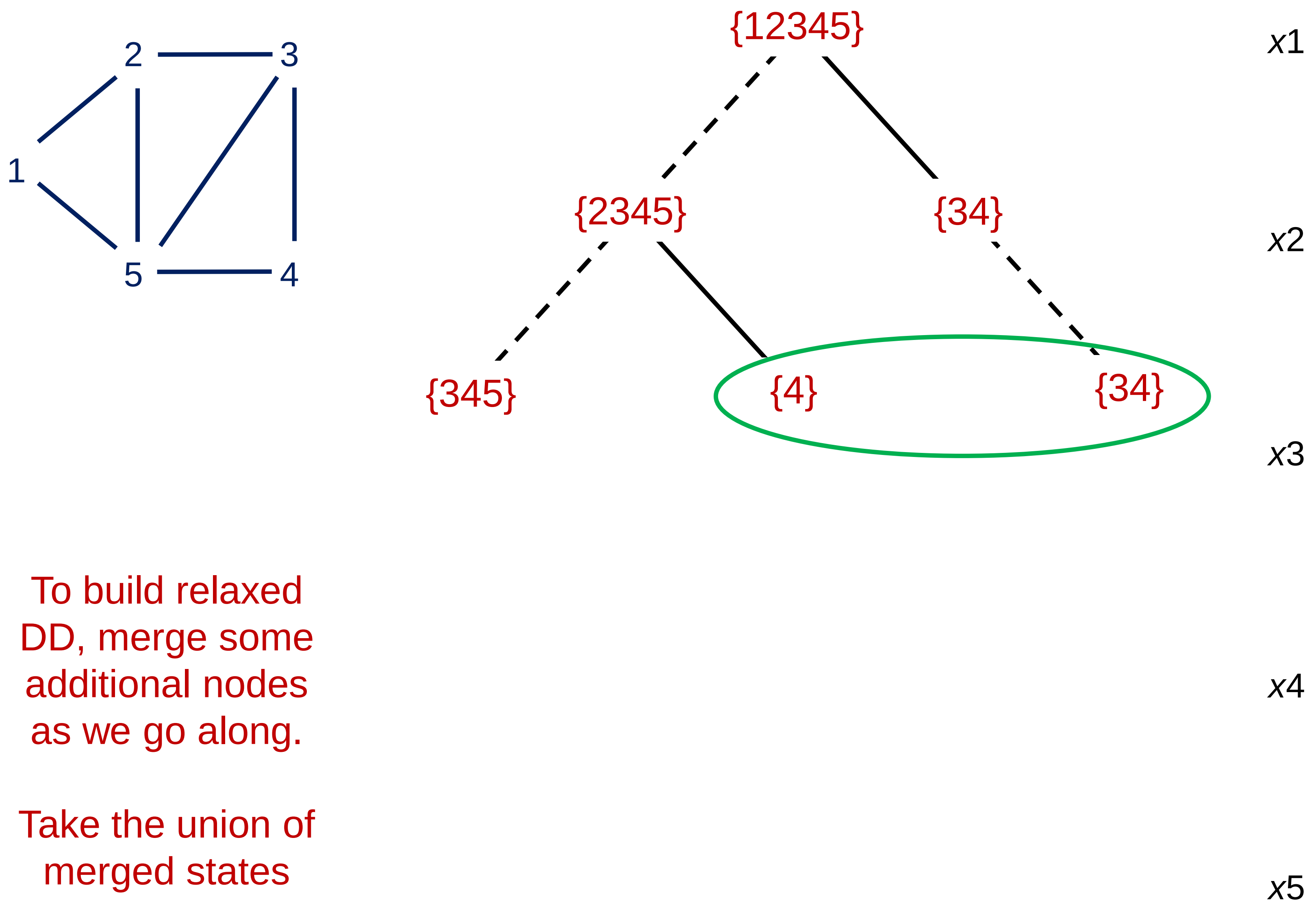
$x5$

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



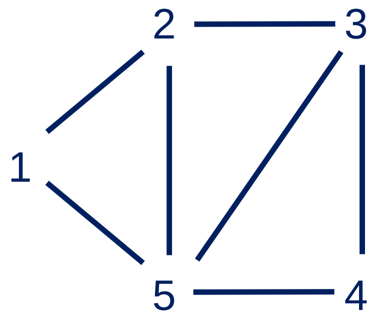


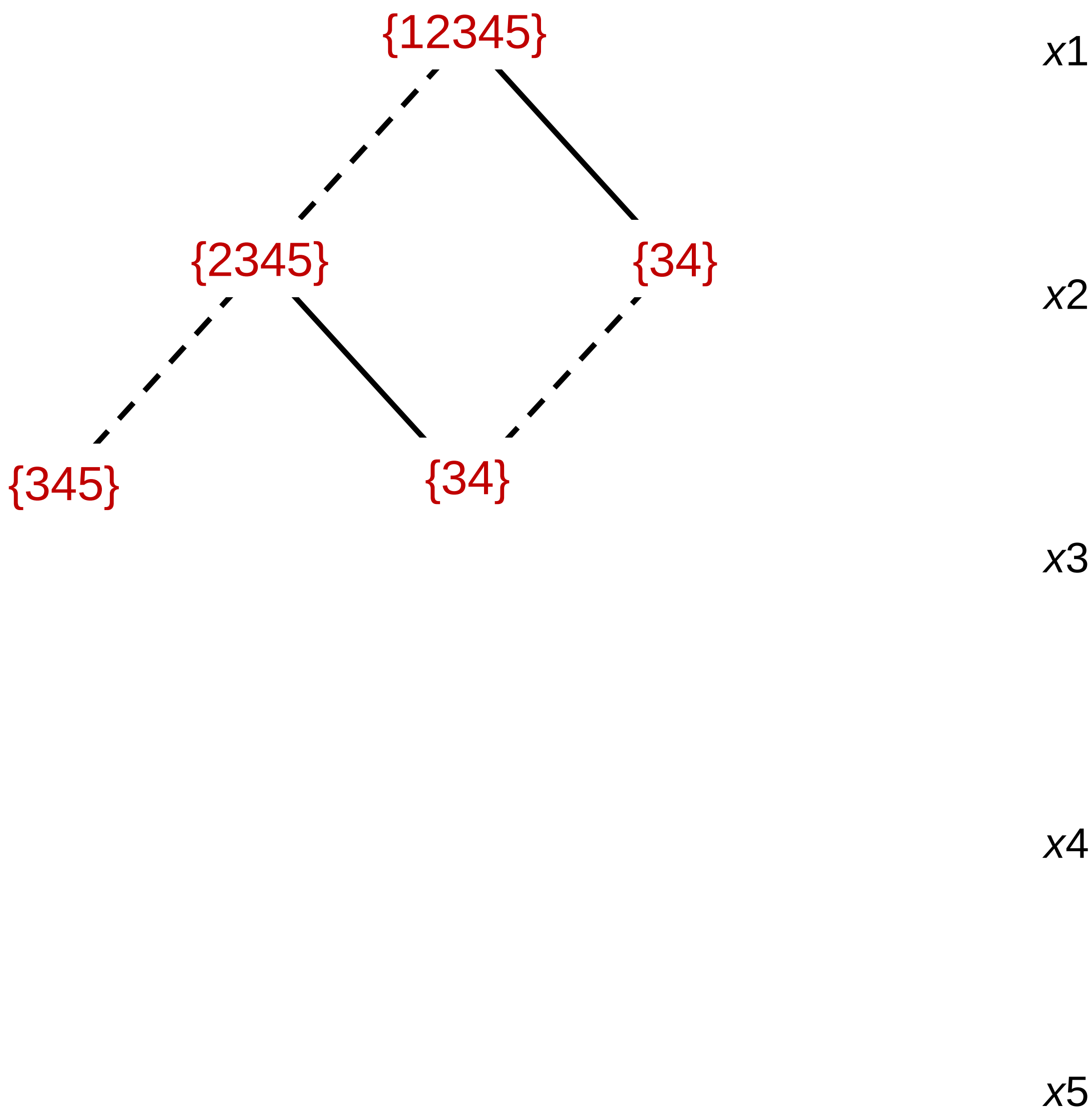
To build relaxed DD, merge some additional nodes as we go along



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

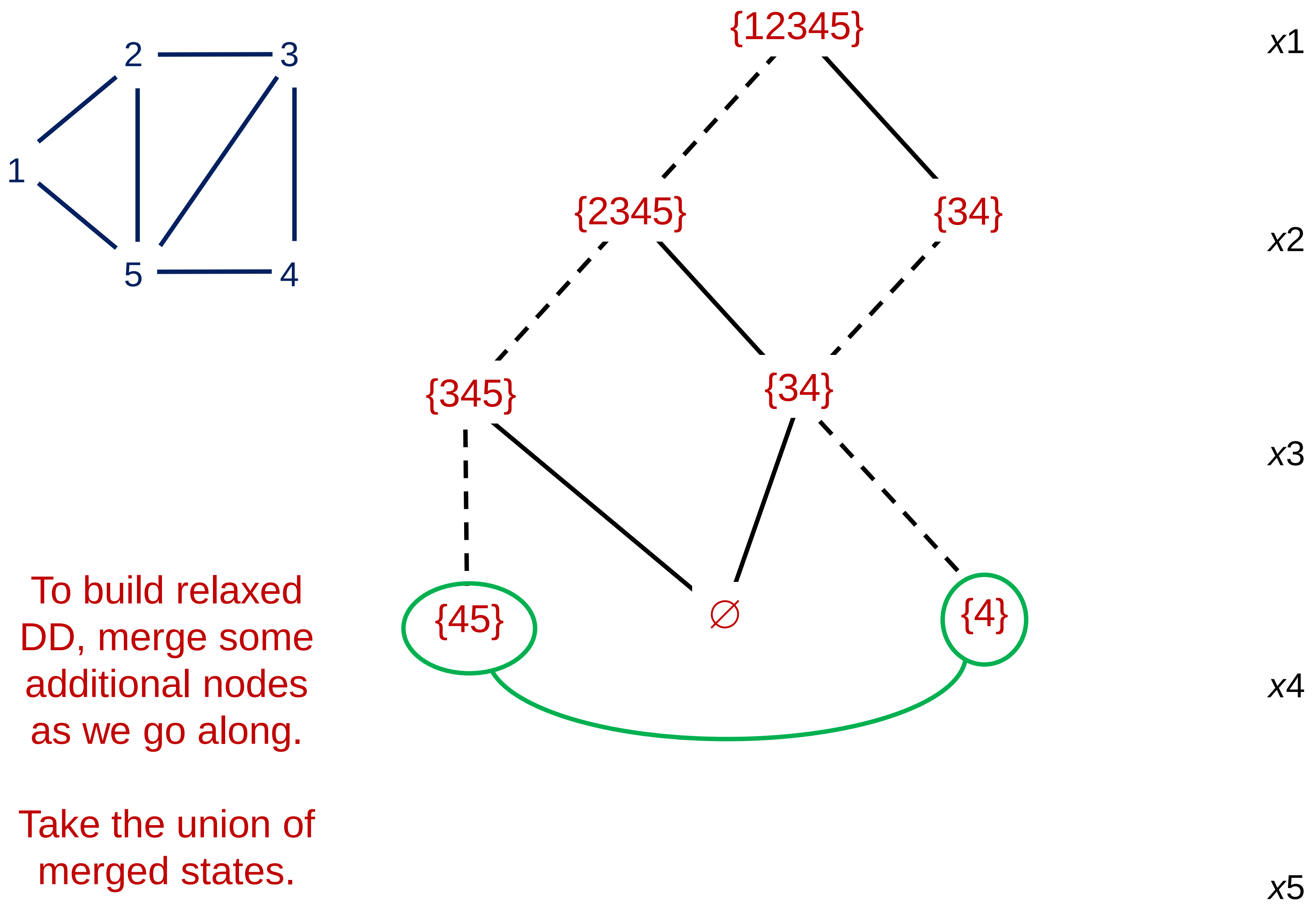
Take the union of merged states





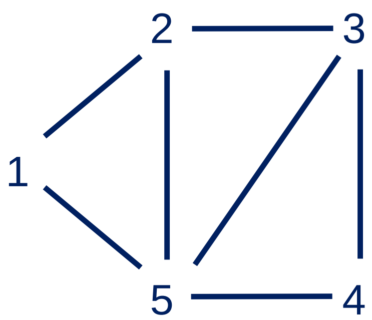
To build relaxed DD, merge some additional nodes as we go along.

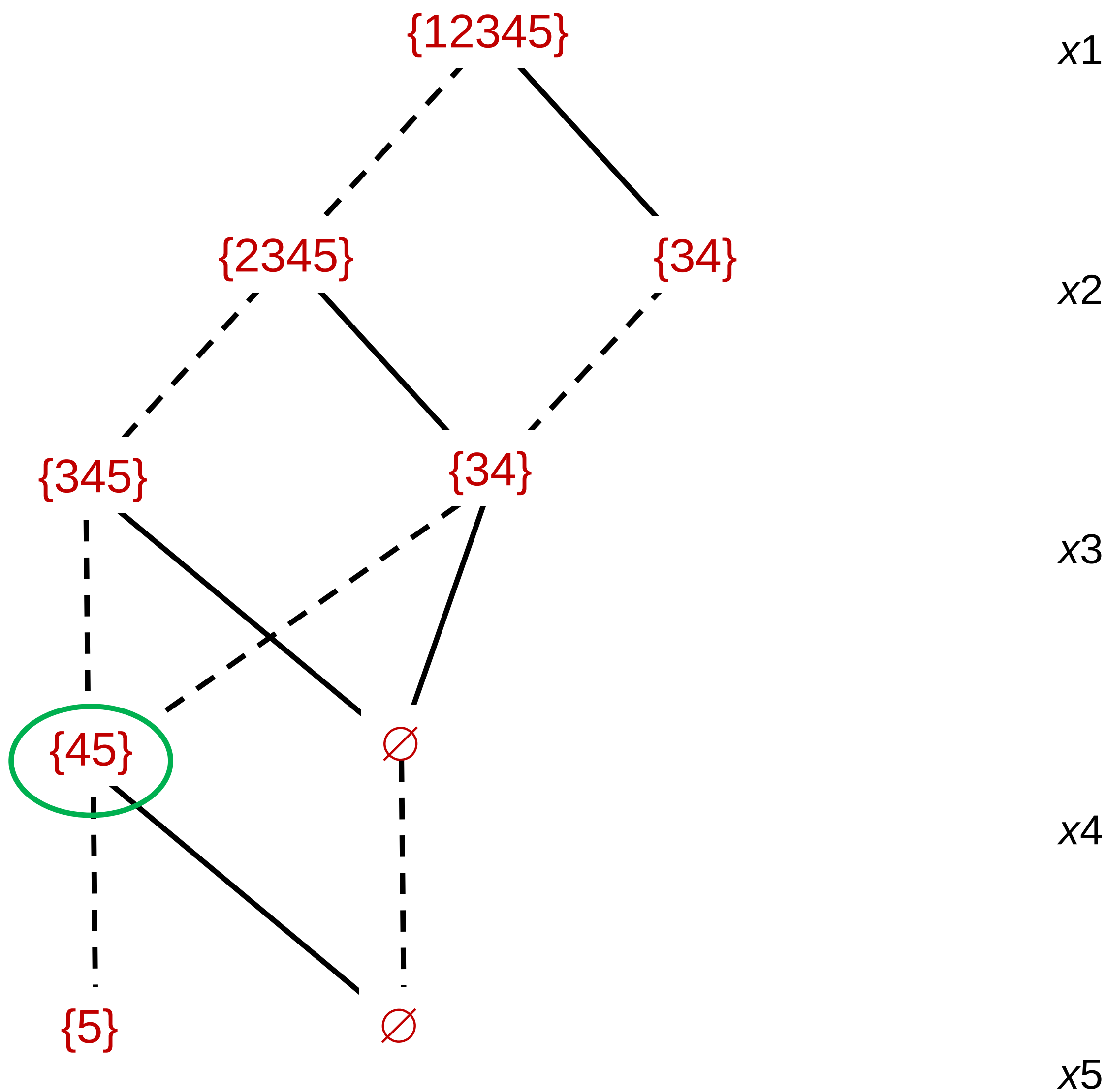
Take the union of merged states.



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

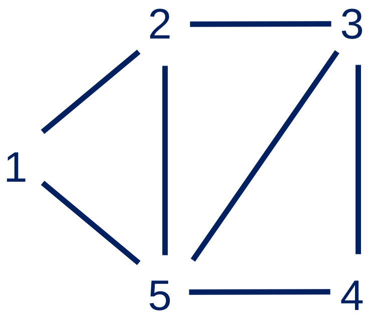
Take the union of merged states.





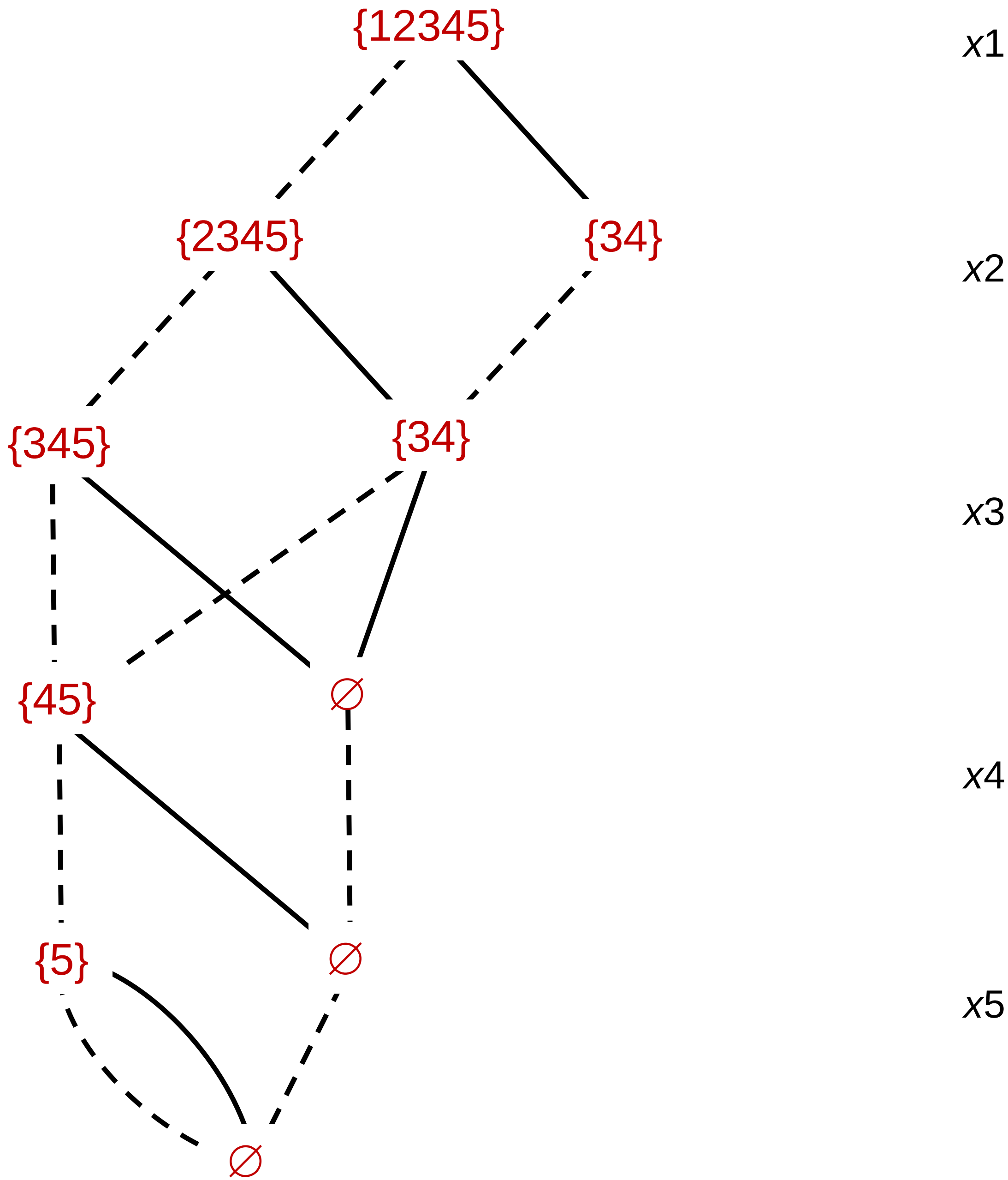
To build relaxed DD, merge some additional nodes as we go along.

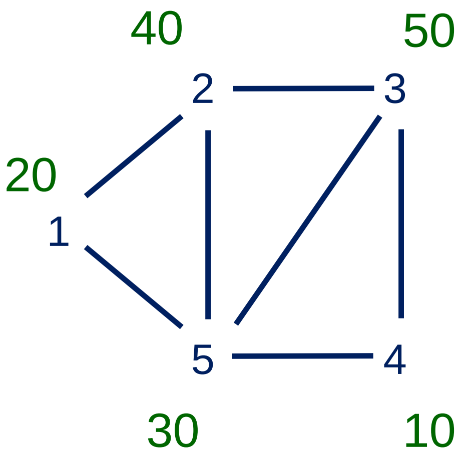
Take the union of merged states.



Width = 2

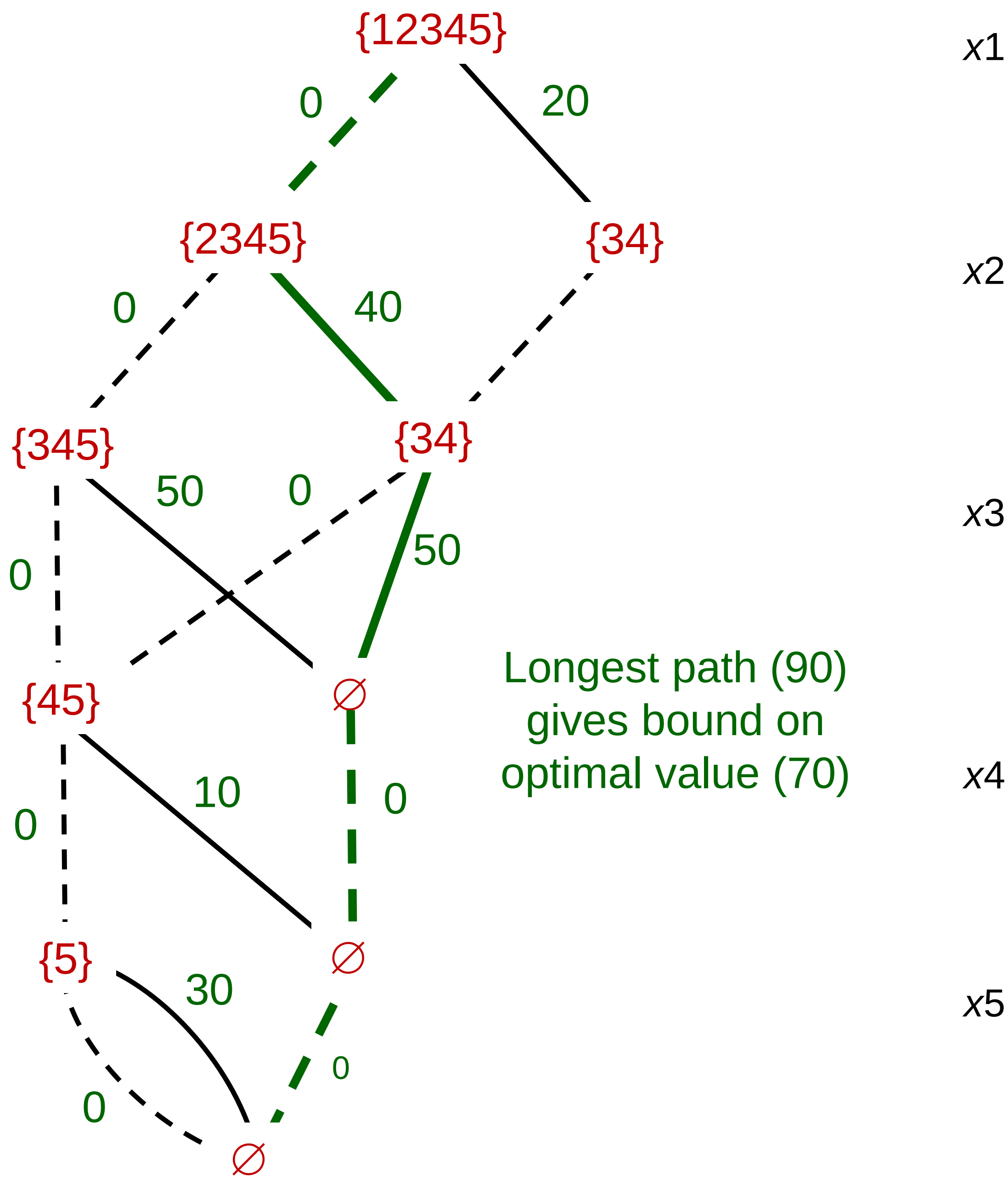
Represents 11
solutions,
including 9
feasible solutions





Width = 2

Represents
11 solutions,
including
9 feasible
solutions

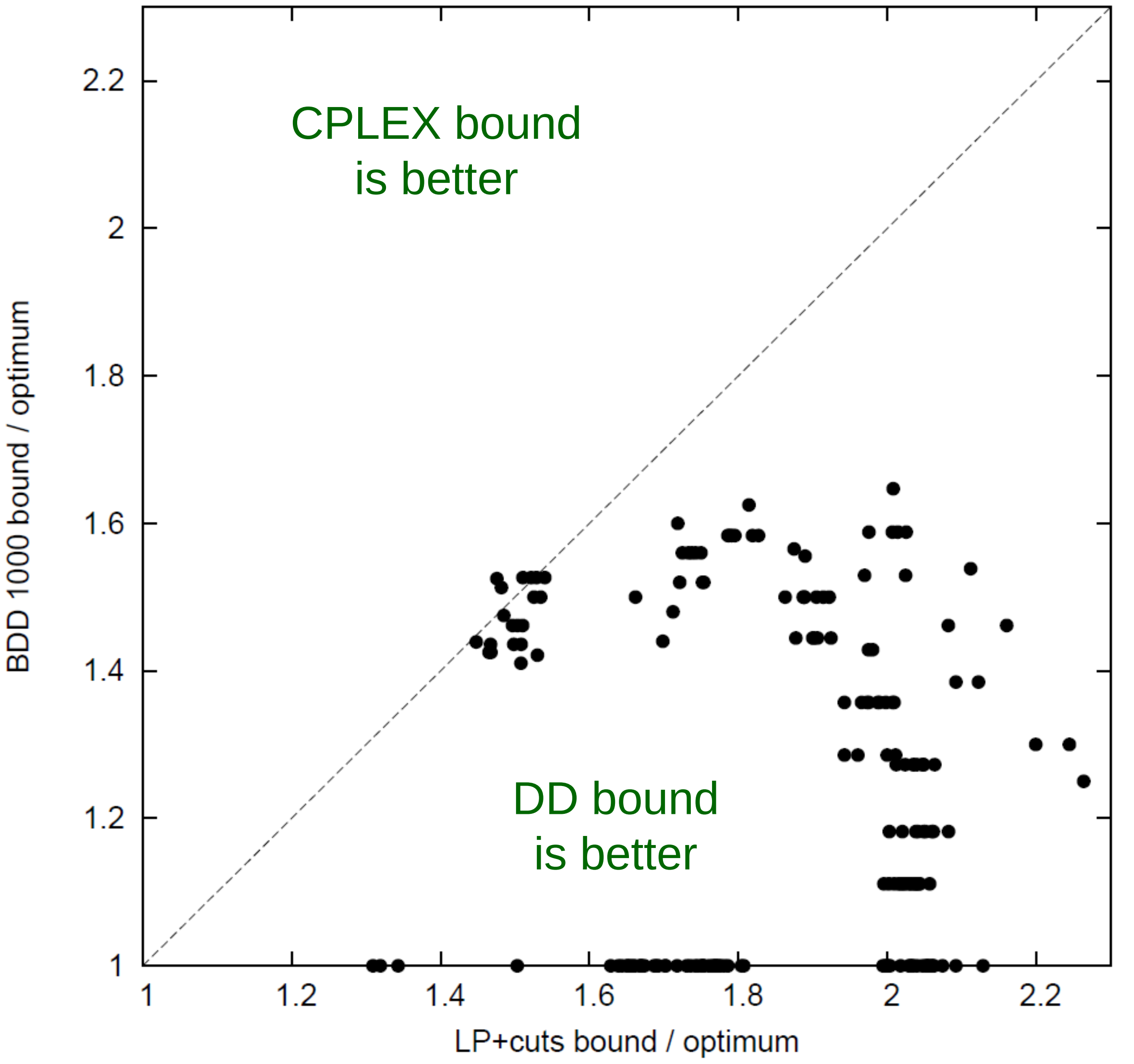


Longest path (90)
gives bound on
optimal value (70)

Bound from Relaxed DD

DDs vs. CPLEX bound at root node for max stable set problem

- Using CPLEX default cut generation
- DD max width of 1000.
- DDs require about 5% the time of CPLEX



Bergman, Ciré, van Hoeve, JH (2013)

Decision Diagrams and Optimization

Propagation through a relaxed DD can substantially improve performance of **constraint programming**.

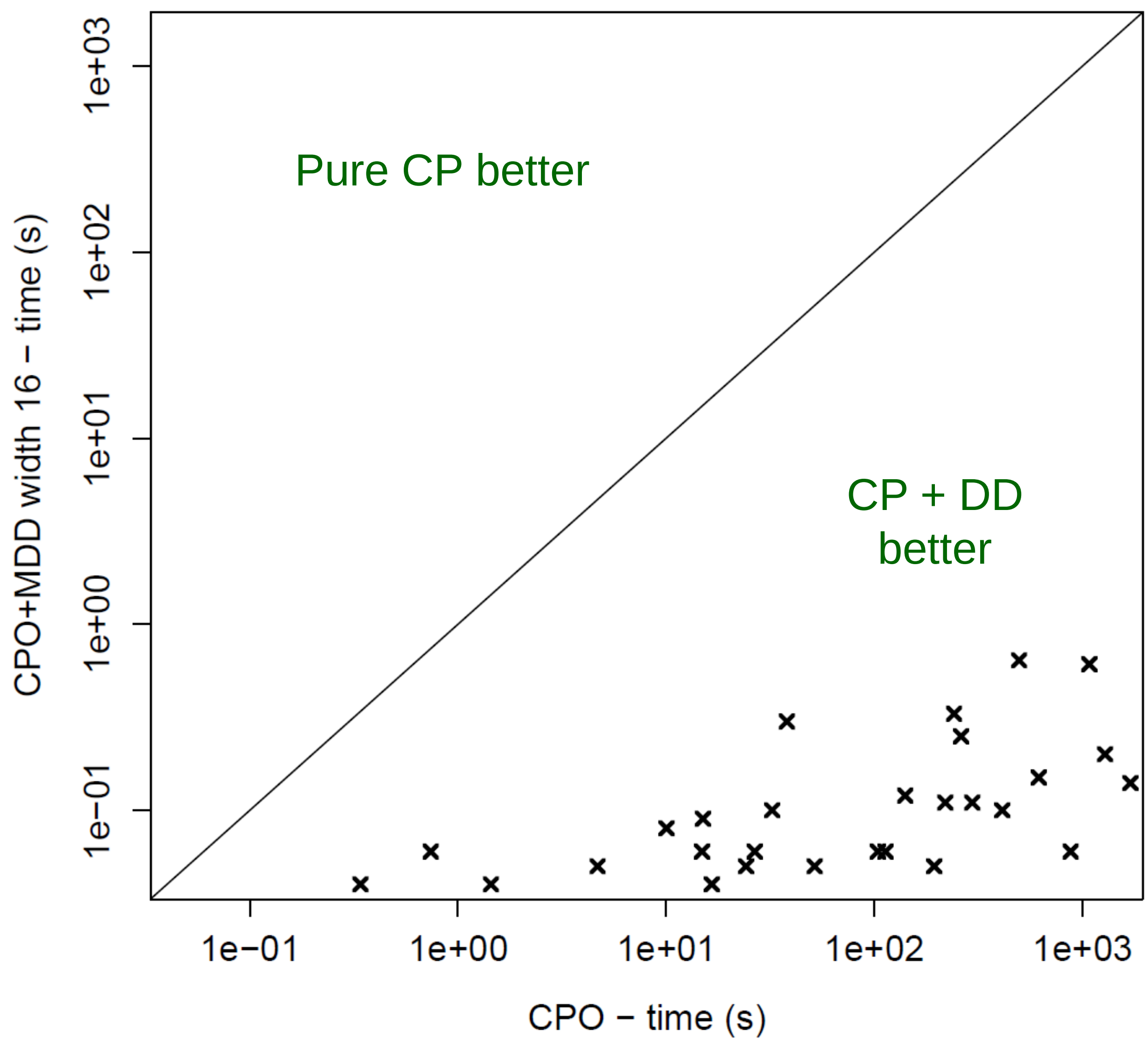
Example: TSP with time windows and other sequencing problems.

DDs allowed closure of several open problem instances.

Ciré, van Hoeve (2013)

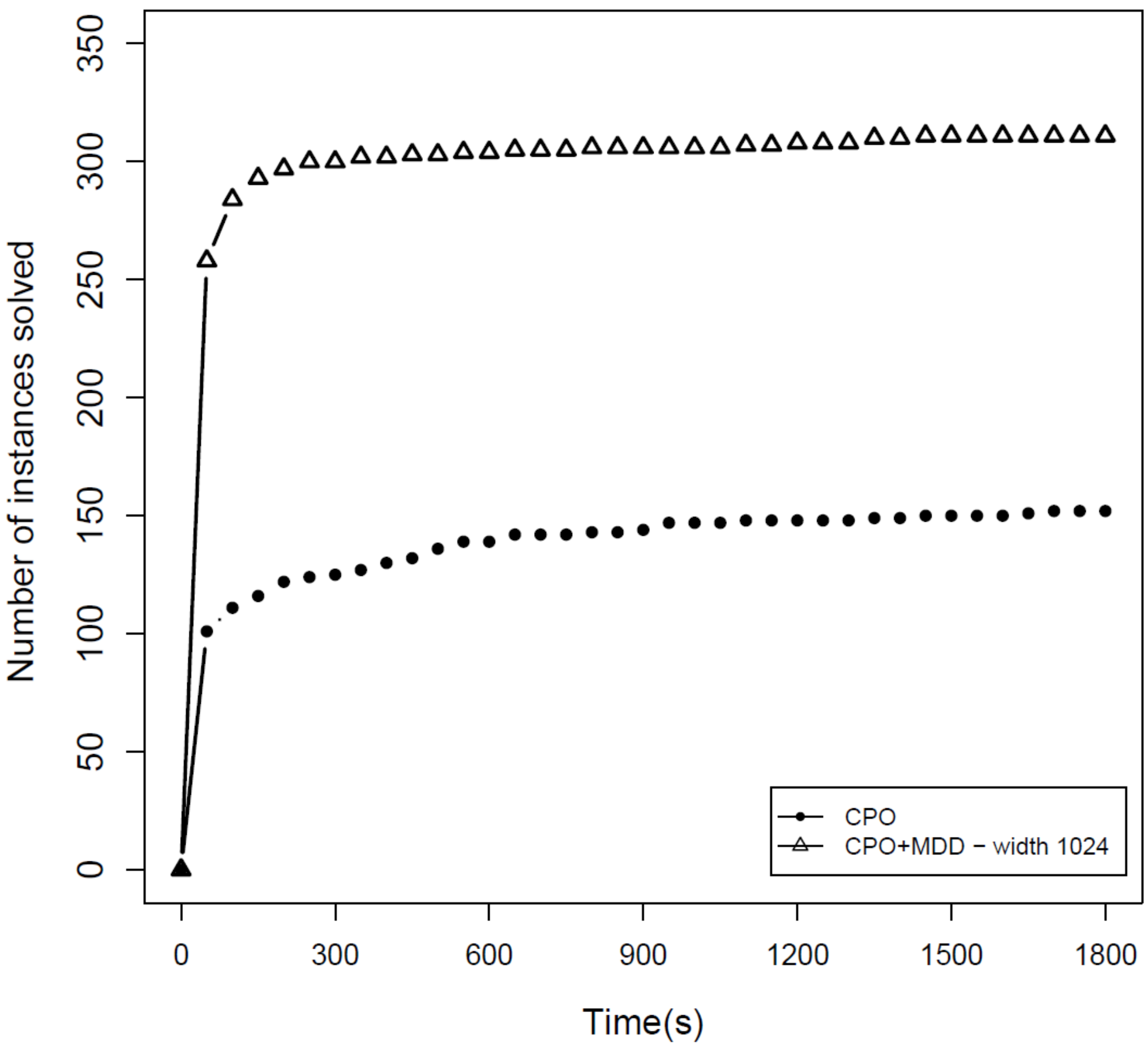
Computation time scatter plot, lex search

CPO =
CP Optimizer



Performance profile, depth-first search

CPO =
CP Optimizer



Decision Diagrams and Optimization

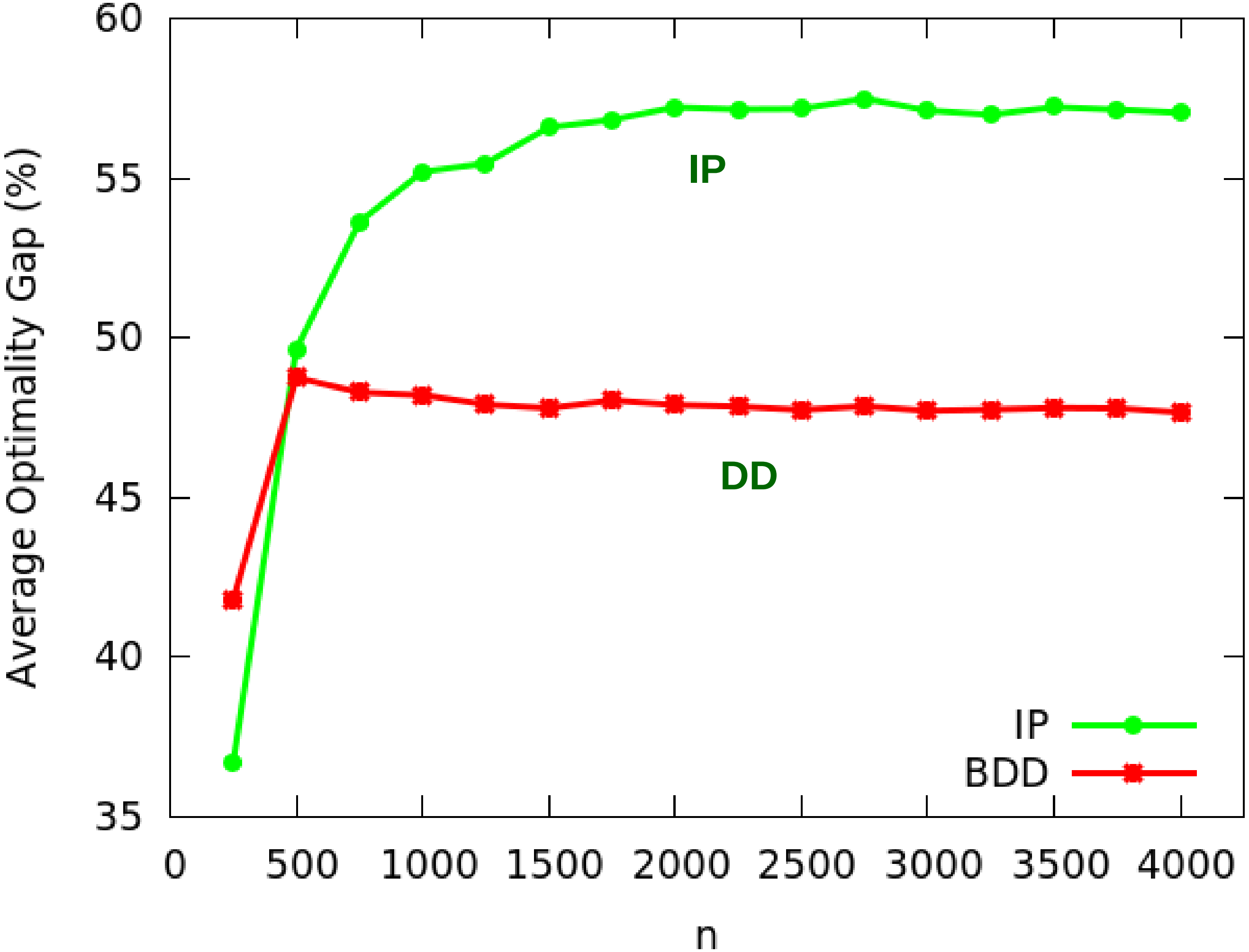
A **restricted** DD represents a **subset** of the feasible set.

Restricted DDs provide a basis for a **primal heuristic**.

Bergman, Ciré, van Hoeve, Yunes (2014)

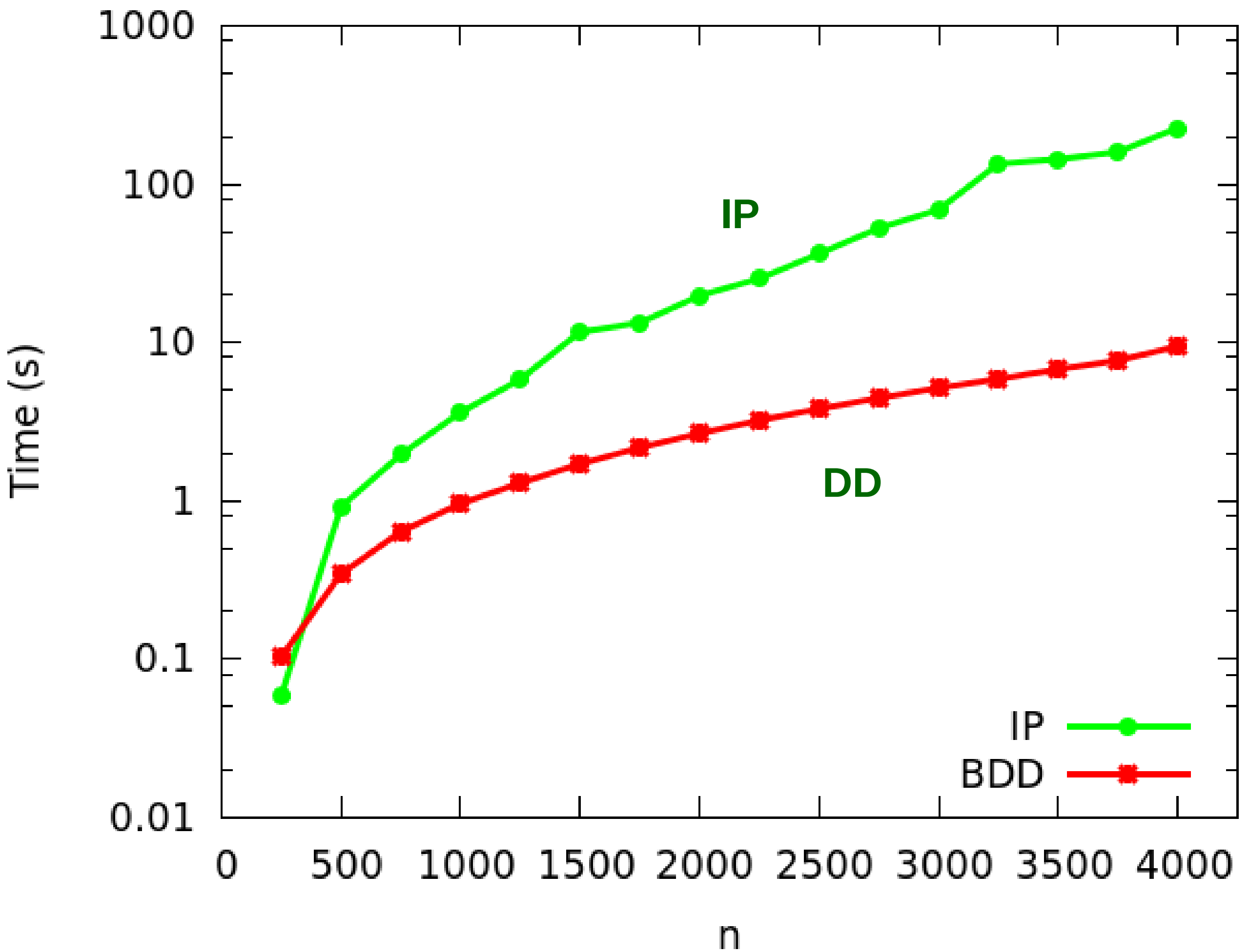
Optimality gap for **set covering**, *n* variables

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)



Decision Diagrams and Optimization

DDs provide a **general purpose solver** for discrete optimization.

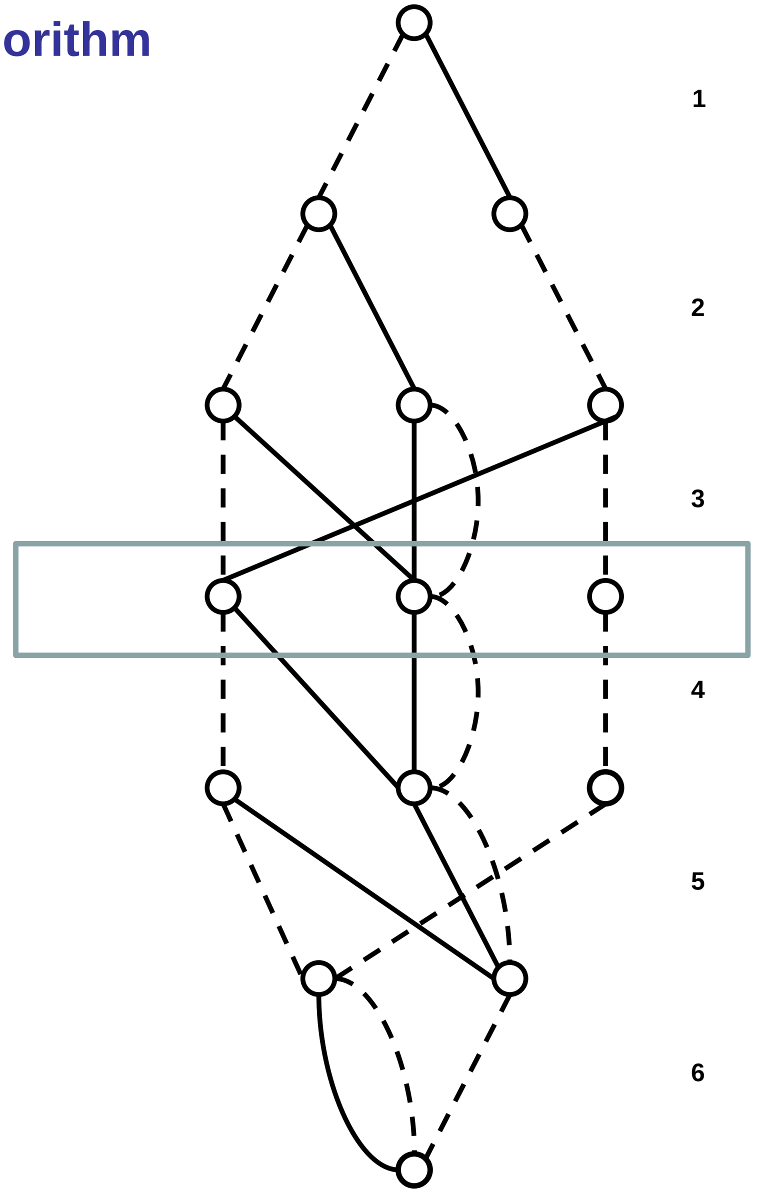
- **Bounds** from relaxed DDs.
- **Primal heuristic** from restricted DDs.
- **Recursive modeling**
- Novel **branching algorithm** – branch inside relaxed DD

Bergman, Ciré, van Hoeve, JH (2016)

Branching Algorithm

Branching in a relaxed decision diagram

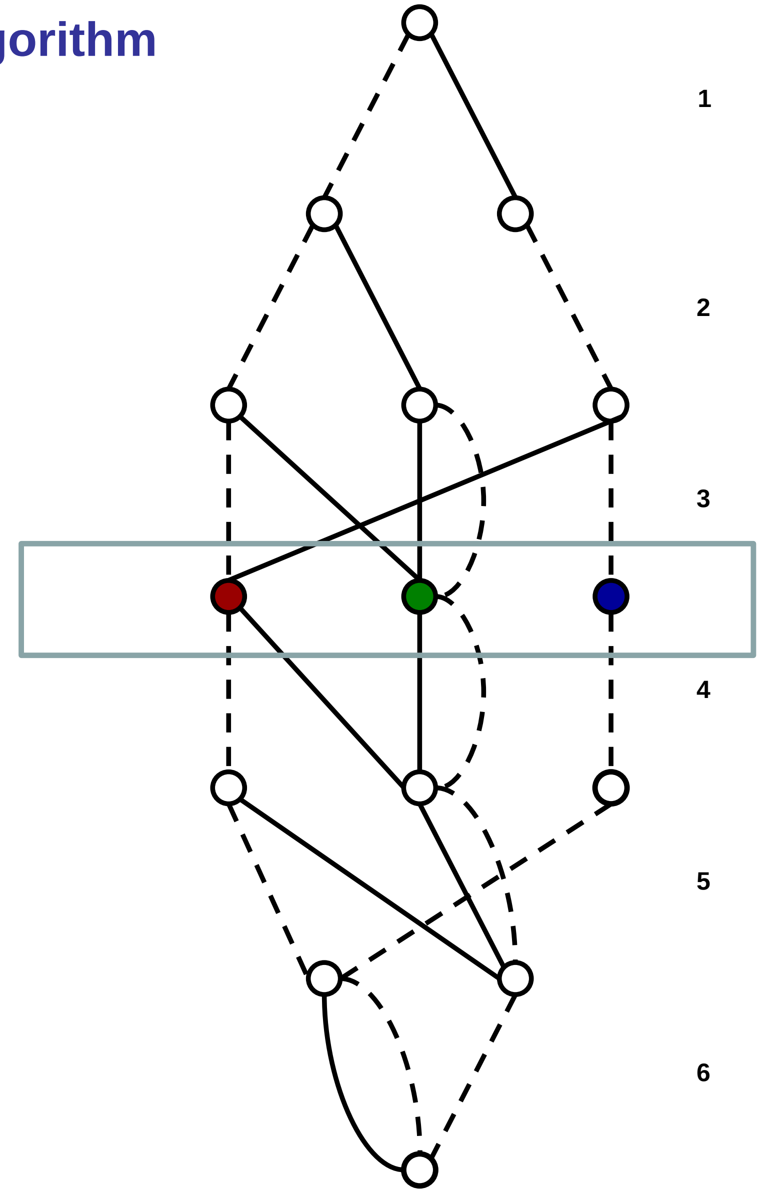
Diagram is exact
down to here



Branching Algorithm

Branching in a relaxed decision diagram

Branch on nodes in this layer

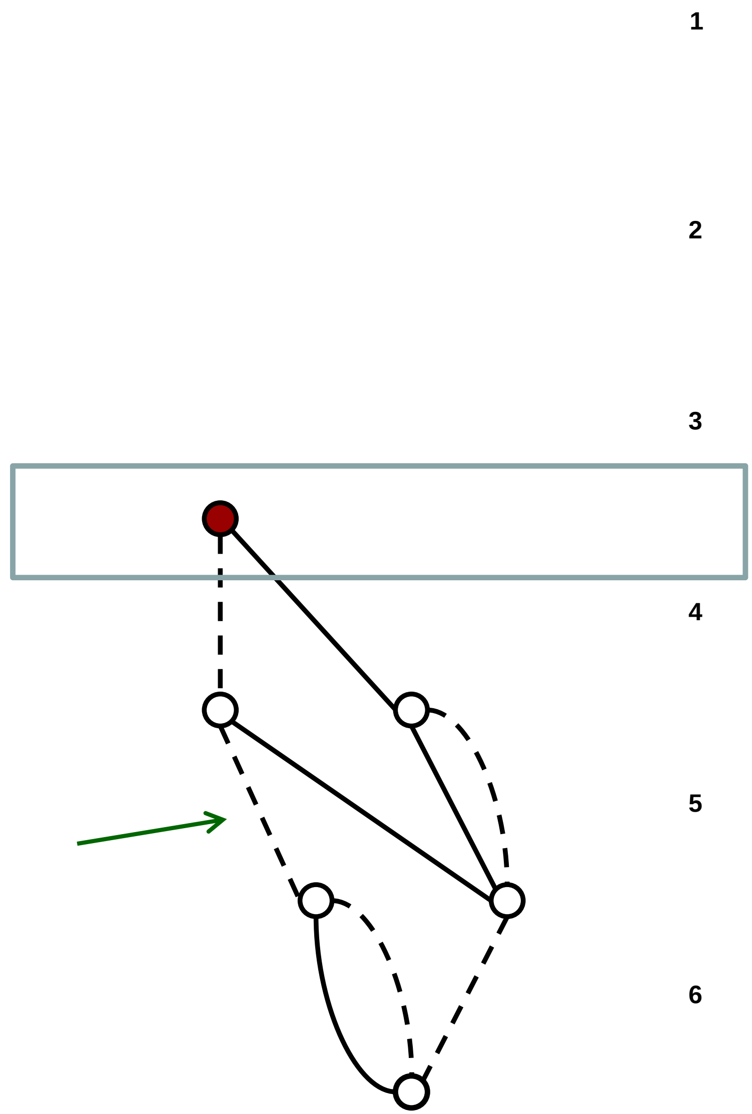


Branching Algorithm

Branching in a relaxed decision diagram

First branch

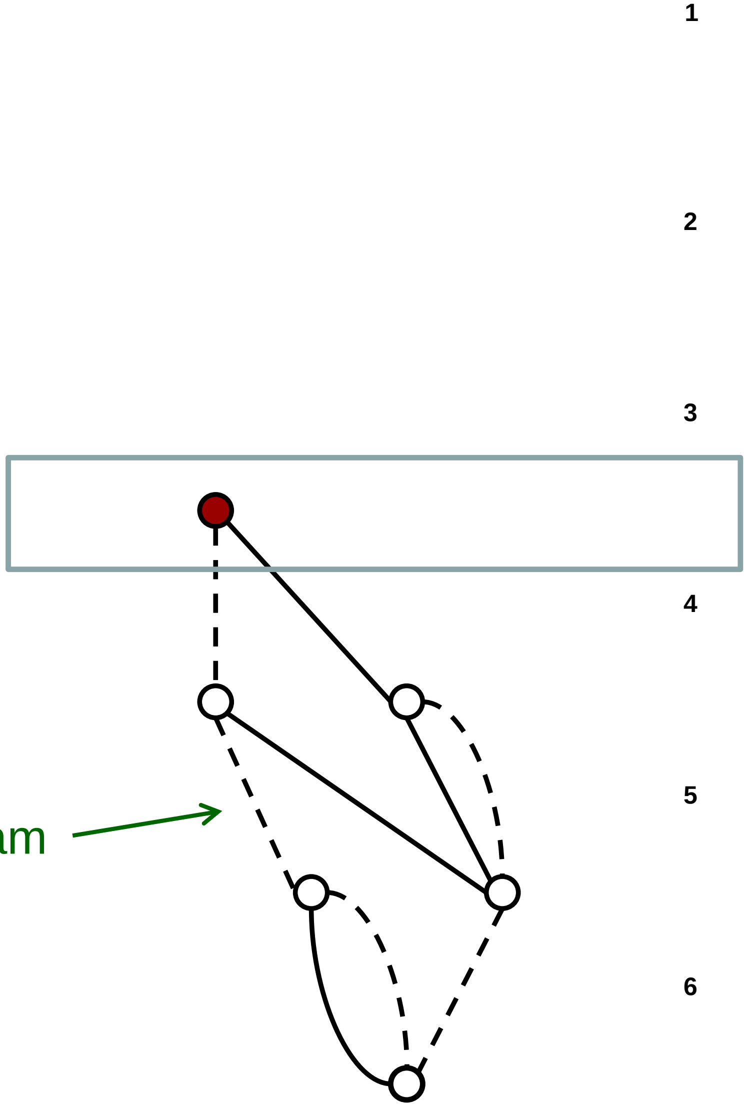
New relaxed decision diagram



Branching Algorithm

Branching in a relaxed
decision diagram

First branch



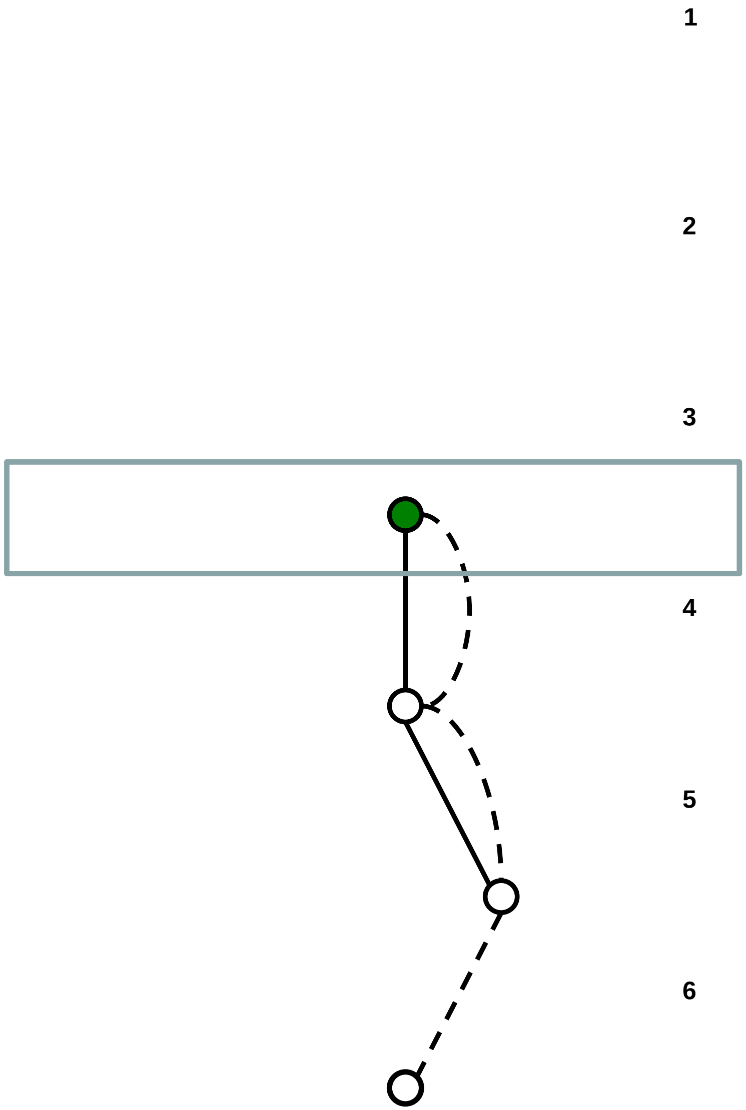
New relaxed decision diagram

Pruning based on cost bounds
from relaxed DDs (branch and bound).

Branching Algorithm

Branching in a relaxed
decision diagram

Second branch



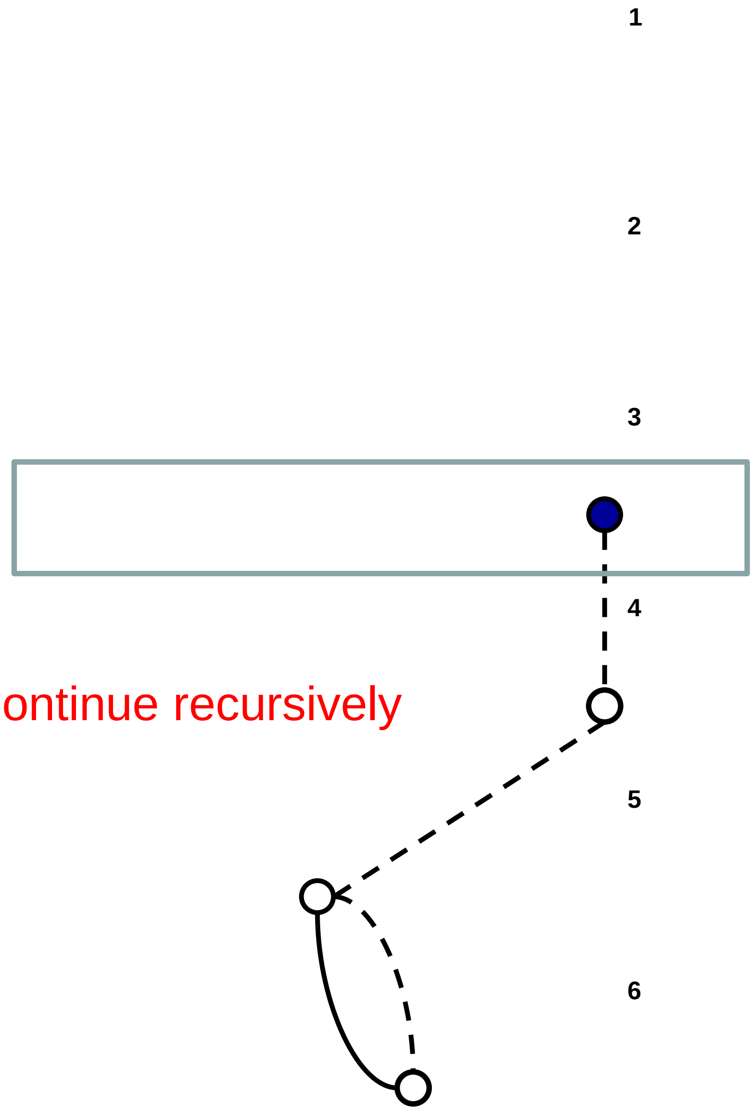
Pruning based on cost bounds
from relaxed DDs (branch and bound).

Branching Algorithm

Branching in a relaxed decision diagram

Third branch

Continue recursively



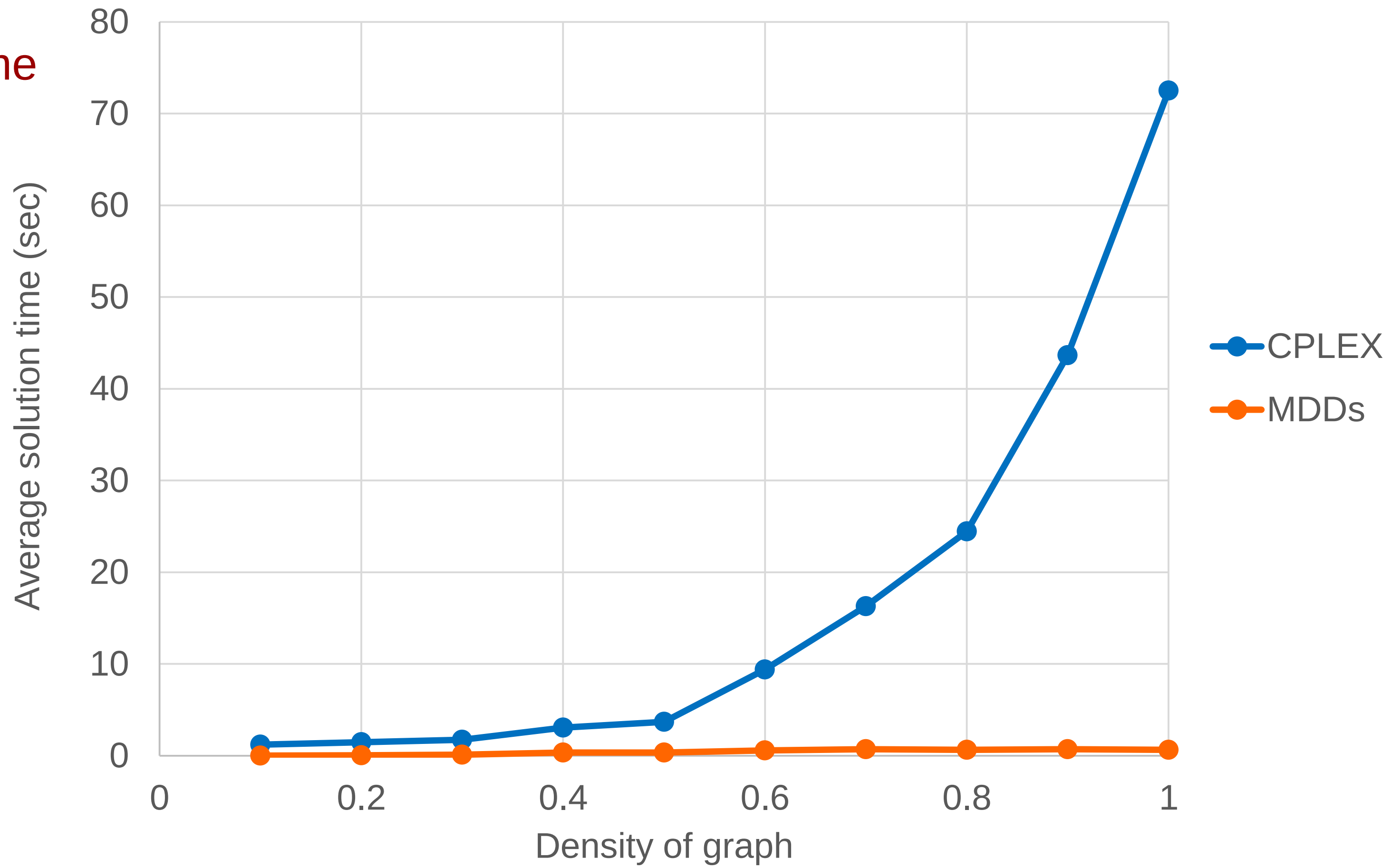
Pruning based on cost bounds from relaxed DDs (branch and bound).

Computational performance

Max cut
on a graph

Avg. solution time
vs
graph density

30 vertices

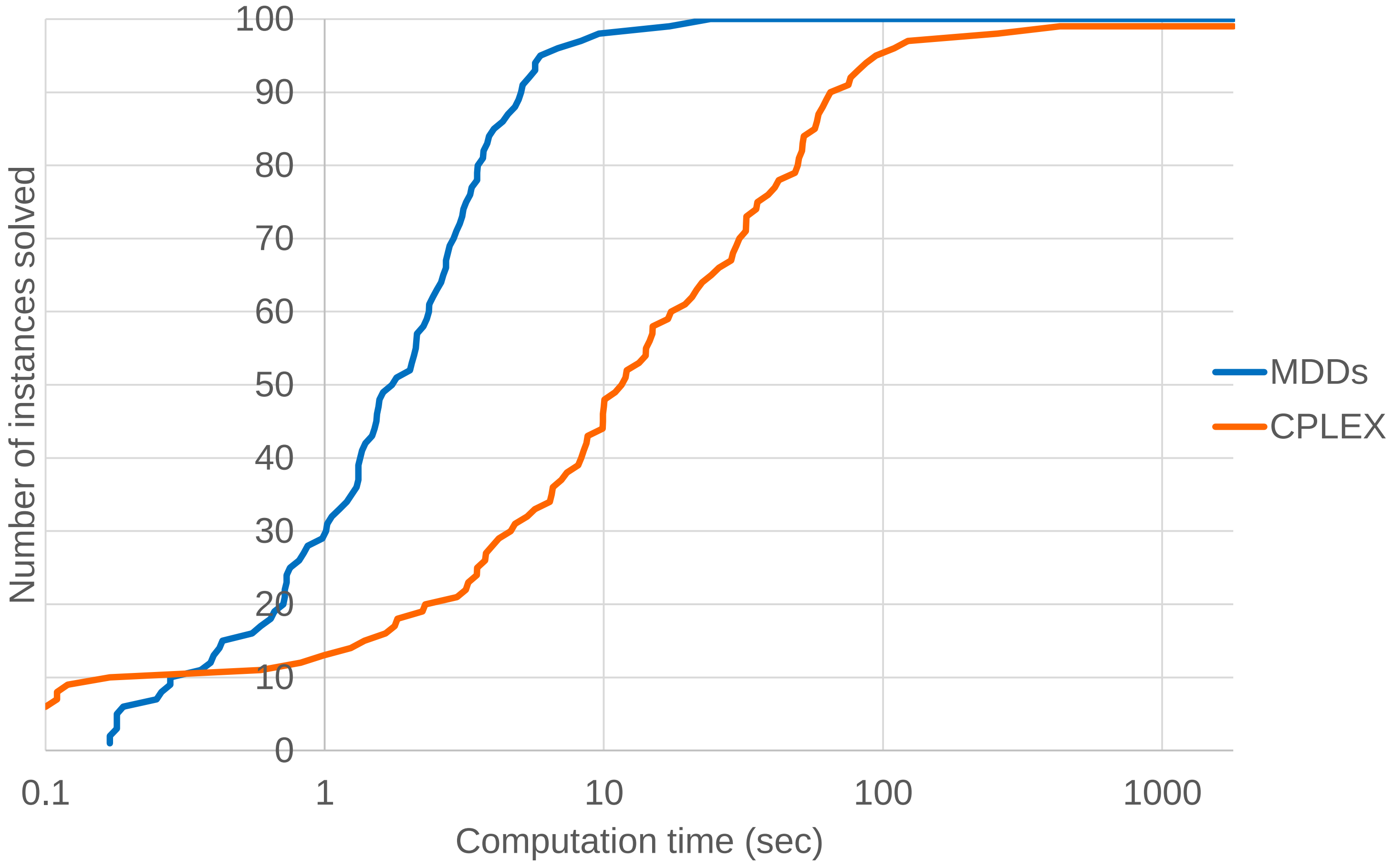


Computational performance

Max 2-SAT

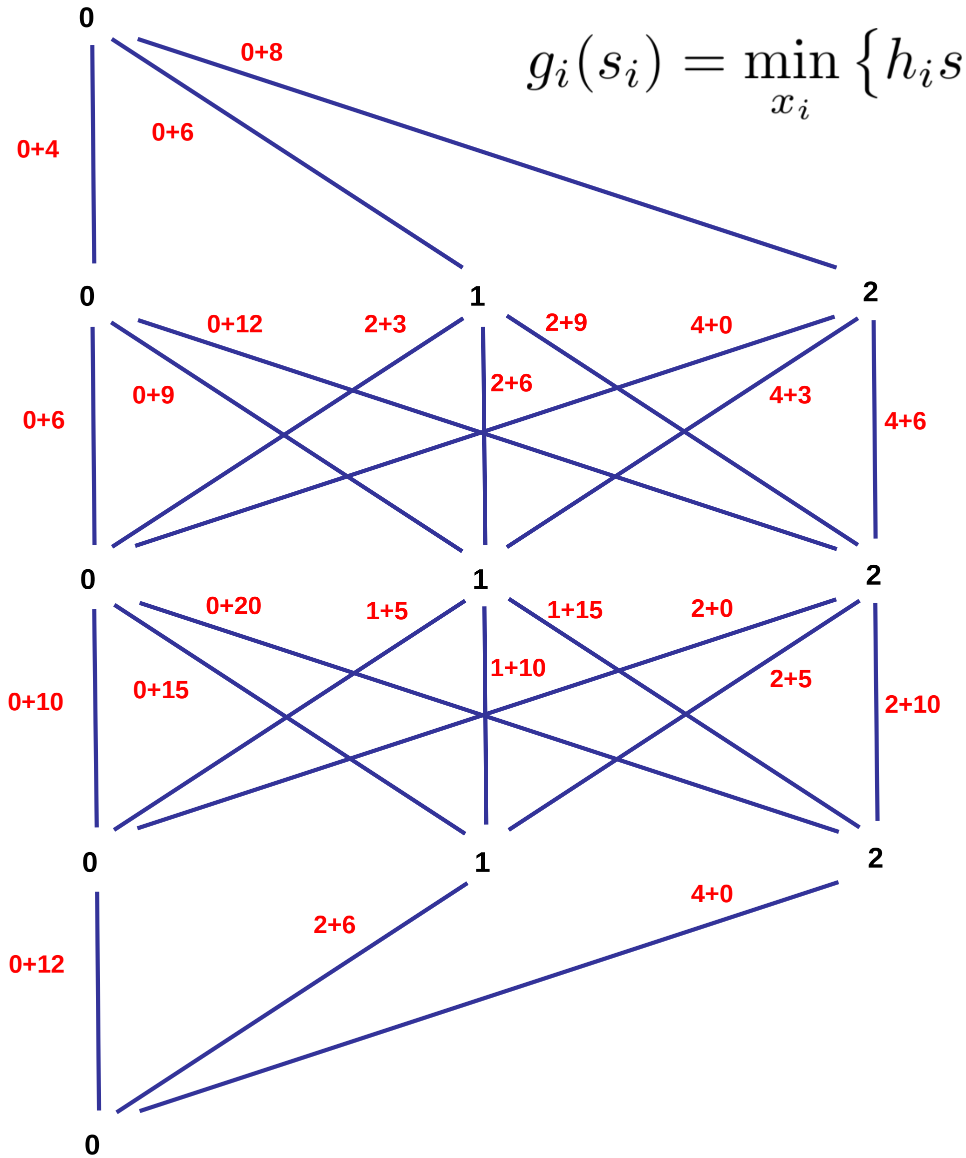
Performance profile

30 variables



Simplification of DP Models

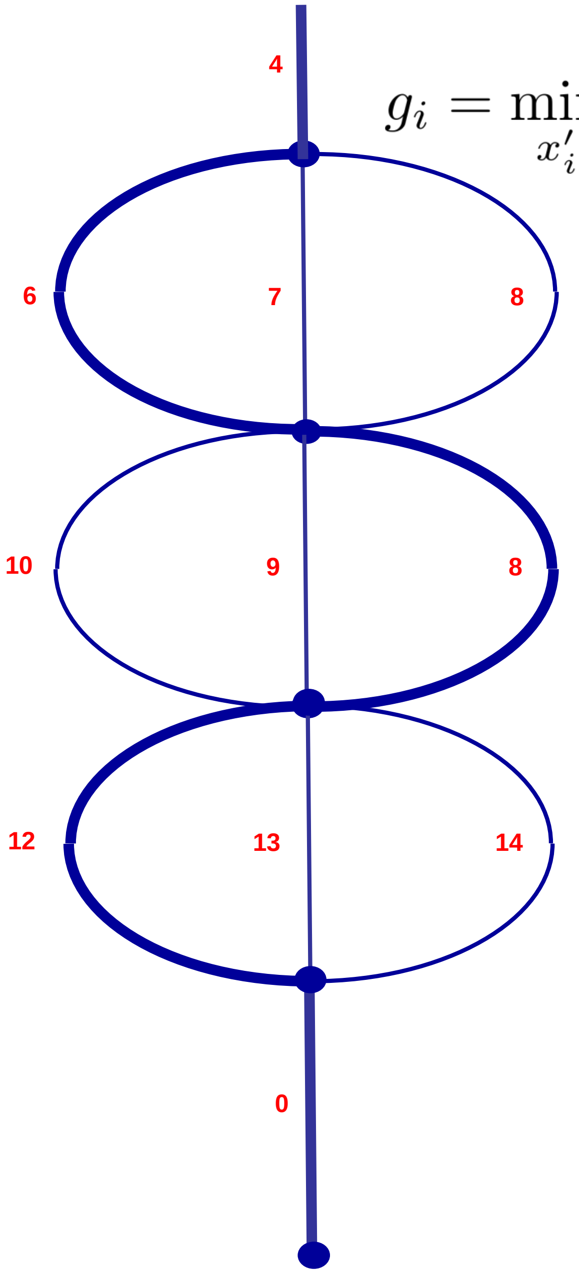
$$g_i(s_i) = \min_{x_i} \left\{ h_i s_i + c_i x_i + g_{i+1}(s_i + x_i - d_i) \right\}$$



State transition graph for
inventory problem

JH (2013)

Simplification of DP Models



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

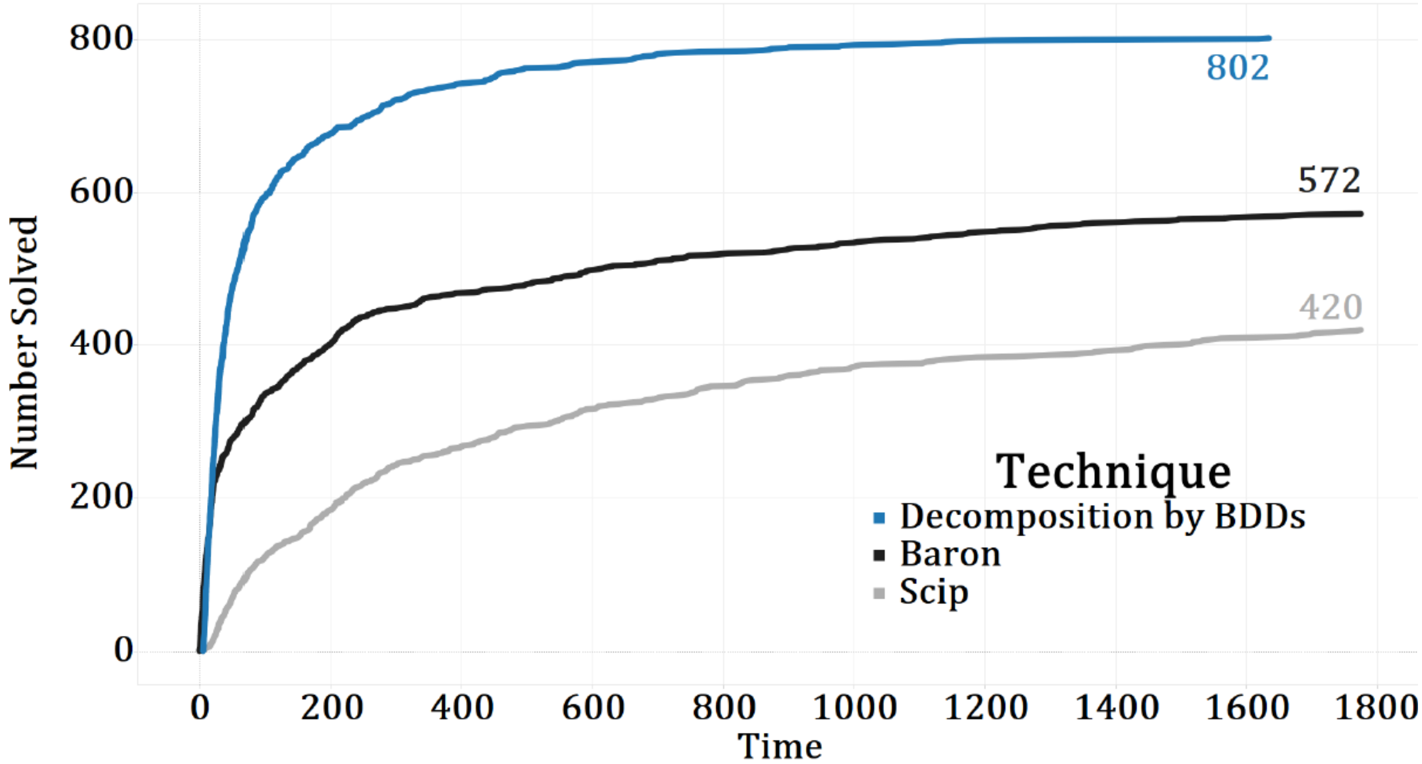
We can reformulate the recursion to yield the **unique reduced weighted DD**.

Radical simplification of the problem – only 1 state per stage.

JH (2013)

Decision Diagrams and Optimization

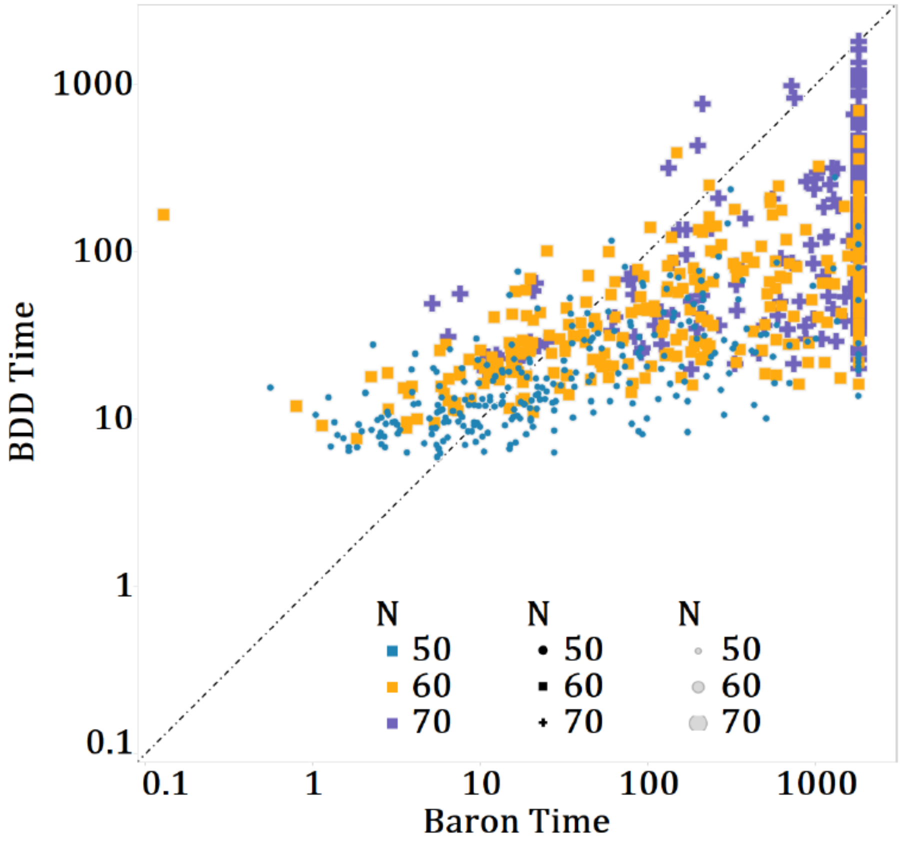
Nonlinear optimization: Portfolio design



Bergman and Ciré (2018)

Decision Diagrams and Optimization

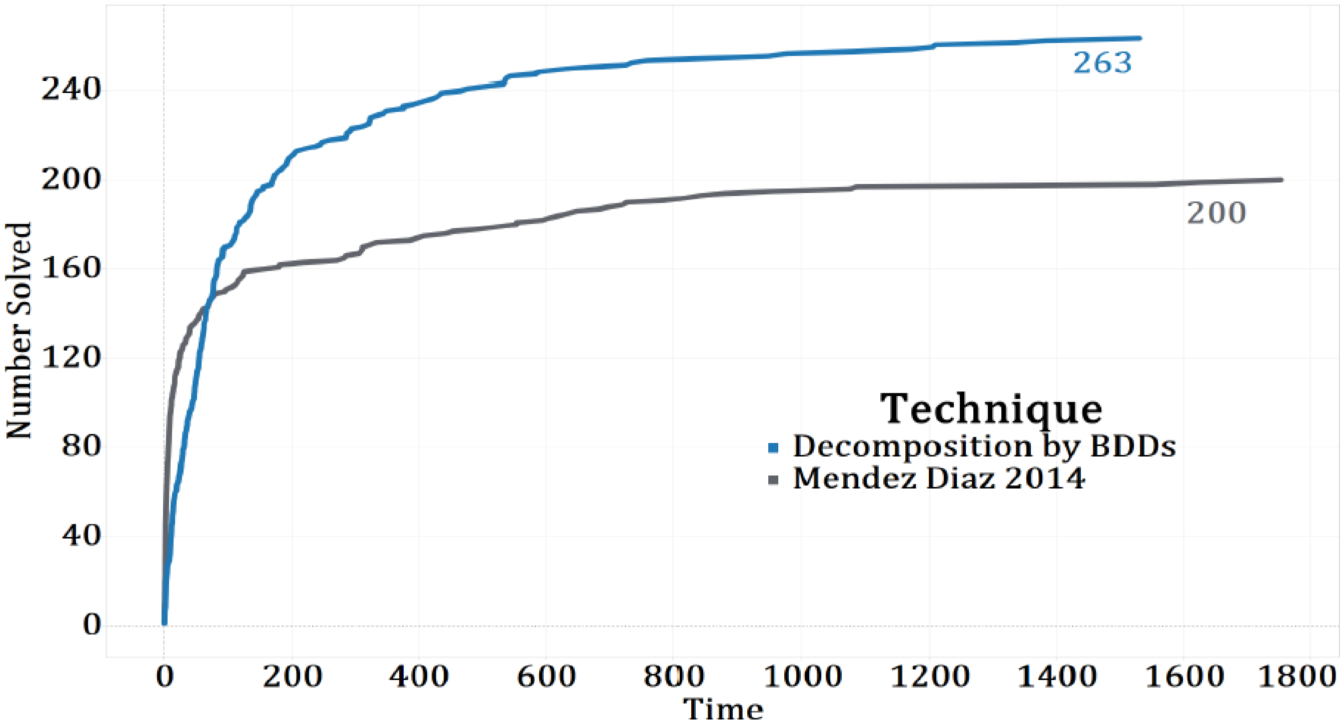
Nonlinear optimization: Portfolio design



Bergman and Ciré (2018)

Decision Diagrams and Optimization

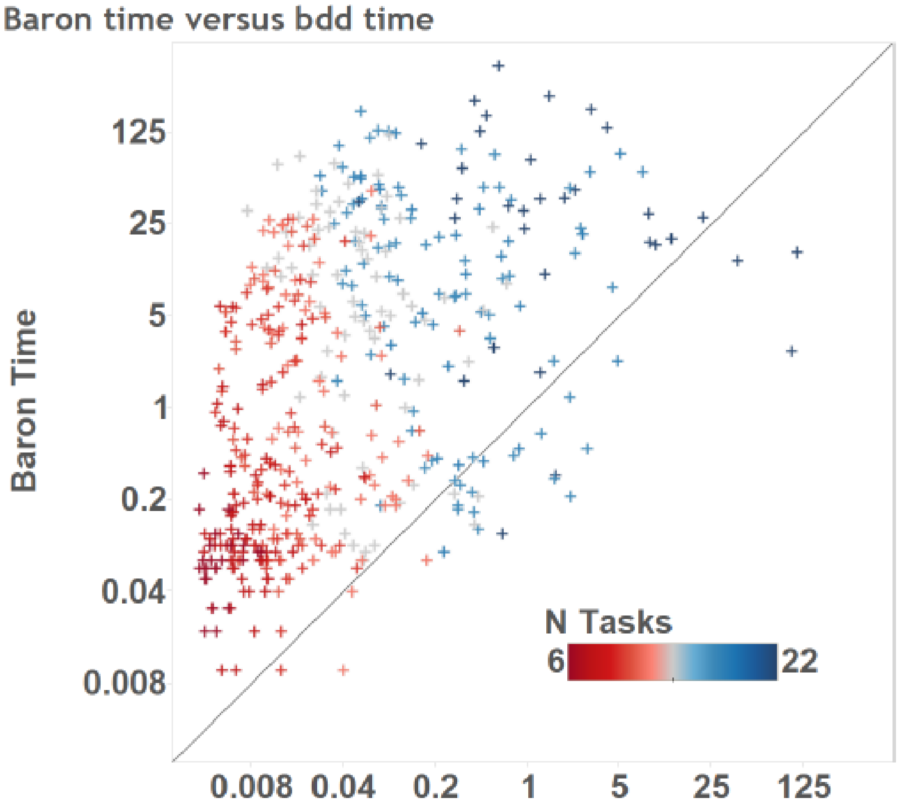
Nonlinear optimization: Product assortment



Bergman and Ciré (2018)

Decision Diagrams and Optimization

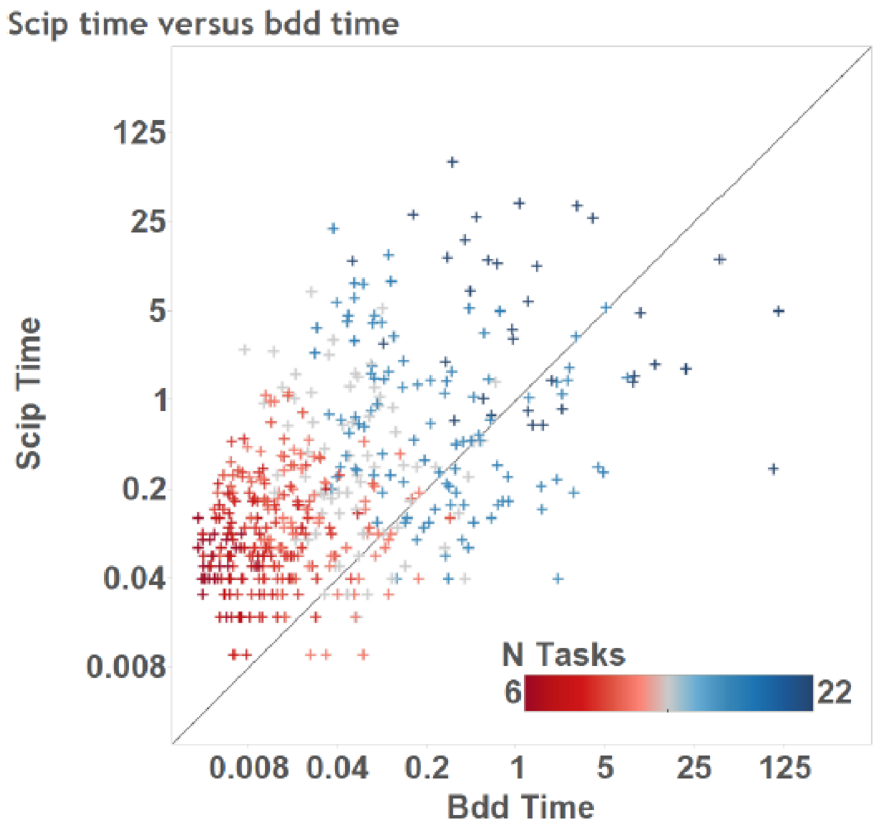
Nonlinear optimization: Workflow employee assignment



Bergman and Ciré (2018)

Decision Diagrams and Optimization

Nonlinear optimization: Workflow employee assignment



Bergman and Ciré (2018)

Predicate Logic and Integer Programming

Predicate Logic and Integer Programming

Fundamental compactness result in **1st-order predicate logic:**

Theorem (Herbrand). A formula in Skolem normal form is unsatisfiable if and only if some **finite** combination of Herbrand ground instances of its clauses is unsatisfiable.

Herbrand 1930

Jacques Herbrand

Predicate Logic and Integer Programming

Fundamental compactness result in **1^{st}-order predicate logic:**

Theorem (Herbrand). A formula in Skolem normal form is unsatisfiable if and only if some **finite** combination of Herbrand ground instances of its clauses is unsatisfiable.

Herbrand 1930

Jacques Herbrand

Fundamental compactness result in **infinite integer programming:**

Theorem. An IP with infinitely many constraints is infeasible if and only if some **finite** subfamily of the constraints is infeasible.

Predicate Logic and Integer Programming

Fundamental compactness result in **1st-order predicate logic:**

Theorem (Herbrand). A formula is Skolem normal form is unsatisfiable if and only if some **finite** combination of Herbrand ground instances of its clauses is unsatisfiable.

Herbrand 1930

Jacques Herbrand

Fundamental compactness result in **infinite integer programming:**

Theorem. An IP with infinitely many constraints is infeasible if and only if some **finite** subfamily of the constraints is infeasible.

These are the **same theorem**!

Resolution and Cutting Planes

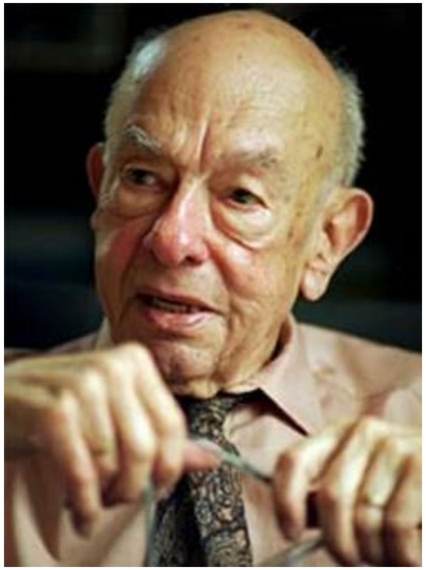
Resolution and Cutting Planes

Resolution is a complete inference method for propositional logic.

Resolution:

$$\frac{\begin{array}{l} x_1 \vee x_2 \vee x_4 \\ x_1 \vee \neg x_4 \end{array}}{x_1 \vee x_2}$$

Quine1952,1955



W. V. Quine

An **input proof** is a resolution proof in which one parent of every resolvent is among the original premises.

Resolution and Cutting Planes

A resolvent is a **rank 1 Chvátal cut.**

$$
\begin{array}{llr}
x_1 + x_2 + x_4 \geq 1 & & (1/2) \\
x_1 \qquad\quad - x_4 \geq 0 & & (1/2) \\
\qquad x_2 \qquad\quad \geq 0 & & (1/2) \\
\hline
x_1 + x_2 \qquad\quad \geq \lceil \frac{1}{2} \rceil & &
\end{array}
$$

V. Chvátal

Chvátal's **cutting plane proof** implicitly relies on resolution!

Chvátal 1973

Resolution and Cutting Planes

A resolvent is a **rank 1 Chvátal cut.**

$$
\begin{array}{llllll}
x_1 + x_2 + x_4 \geq 1 & (1/2) \\
x_1 \quad\quad\; - x_4 \geq 0 & (1/2) \\
\quad\quad x_2 \quad\quad\;\; \geq 0 & (1/2) \\
\hline
x_1 + x_2 \quad\quad\;\; \geq \lceil \tfrac{1}{2} \rceil &
\end{array}
$$

V. Chvátal

Chvátal's **cutting plane proof** implicitly relies on resolution!

Chvátal 1973

Theorem. The logical clauses one can infer using input proofs are precisely those that are rank 1 cuts.

JH 1989

Theorem. Resolution can be generalized to a complete inference method for 0-1 inequalities (a logical analog of Chvátal's theorem).

JH 1992

Logic and Duality

Logic and Duality

Optimization **duals** are **logical inference problems**.

This implies a **tight connection** between logic and optimization.

It leads to an **extension of Benders decomposition**
that has seen many applications.

Numerische Mathematik 4, 238—252 (1962)

Partitioning procedures for solving mixed-variables programming problems*

By

J. F. BENDERS**

Logic and Duality

All optimization duals are special cases of **inference duality**

Primal problem:
Optimization

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

Find **best** feasible solution by searching over **values of *x***.

Dual problem:
Inference

$$\max v$$
$$x \in S \overset{P}{\Rightarrow} f(x) \geq v$$
$$P \in \mathcal{P}$$

Find a proof of optimal value by searching over **proofs *P***.

In classical LP, the proof is a tuple of dual multipliers

Logic and Duality

Type of Dual	Inference Method	Strong?
Linear programming	Nonnegative linear combination + material implication	Yes*
Lagrangian	Nonnegative linear combination + domination	No
Surrogate	Nonnegative linear combination + material implication	No
Subadditive	Cutting planes	Yes**

*Due to Farkas Lemma
**Due to Chvátal’s theorem

Logic and Duality

LP Duality

$$\begin{aligned} \min\ & cx \\ & Ax \geq b \\ & x \geq 0 \end{aligned} \quad = \quad \boxed{\begin{aligned} \max\ & v \\ & Ax \geq b \overset{x \geq 0}{\Longrightarrow} cx \geq v \end{aligned}}$$

implies

Dual problem: Find the tightest lower bound on the objective function that is implied by the constraints.

Logic and Duality

LP Duality

$$\begin{array}{lcl} \min\ cx & = & \max\ v \\ Ax \geq b & & Ax \geq b \overset{x\geq 0}{\Rightarrow} cx \geq v \\ x \geq 0 & & \end{array}$$

From Farkas Lemma: If $Ax \geq b$, $x \geq 0$ is feasible,

$$Ax \geq b \overset{x\geq 0}{\Rightarrow} cx \geq v \quad \text{iff} \quad \lambda Ax \geq \lambda b \ \boxed{\text{implies}} \ cx \geq v$$

$$\text{for some } \lambda \geq 0$$

$\lambda A \leq c$ and $\lambda b \geq v$

Logic and Duality

LP Duality

$$\min\ cx \quad = \quad \max\ v \quad = \quad \max\ \lambda b$$

$$Ax \ge b \qquad Ax \ge b \overset{x\ge 0}{\Rightarrow} cx \ge v \qquad \lambda A \le c$$

$$x \ge 0 \qquad \lambda \ge 0$$

This is the **classical LP dual**

From Farkas Lemma: If $\boxed{Ax \ge b,\ x \ge 0 \text{ is feasible,}}$

$Ax \ge b \overset{x\ge 0}{\Rightarrow} cx \ge v$ iff $\lambda Ax \ge \lambda b$ $\boxed{\text{implies}}$ $cx \ge v$
for some $\lambda \ge 0$

$\lambda A \le c$ and $\lambda b \ge v$

Lagrangian Duality

Primal

$\min\ f(x)$

$g(x) \ge 0$

$x \in S$

Dual

$\max\ v$

$g(x) \ge b \overset{s \in S}{\Rightarrow} f(x) \ge v$

Surrogate

Let us say that

$g(x) \ge 0 \overset{x \in S}{\Rightarrow} f(x) \ge v$ iff $\boxed{\lambda g(x) \ge 0}$ $\boxed{\text{dominates}}$ $f(x) - v \ge 0$

for some $\lambda \ge 0$

$\lambda g(x) \le f(x) - v$ for all $x \in S$

That is, $v \le f(x) - \lambda g(x)$ for all $x \in S$

Lagrangian Duality

Primal

$$\min \; f(x)$$
$$g(x) \geq 0$$
$$x \in S$$

Dual

$$\max \; v$$
$$g(x) \geq b \overset{s \in S}{\Rightarrow} f(x) \geq v$$

Surrogate

Let us say that

$$g(x) \geq 0 \overset{x \in S}{\Rightarrow} f(x) \geq v \quad \text{iff} \quad \boxed{\lambda g(x) \geq 0} \; \boxed{\text{dominates}} \; f(x) - v \geq 0$$

for some $\lambda \geq 0$

$\lambda g(x) \leq f(x) - v$ for all $x \in S$

That is, $v \leq f(x) - \lambda g(x)$ for all $x \in S$

If we replace domination with material implication,
we get the **surrogate dual**, which gives better bounds
but lacks the nice properties of the Lagrangean dual.

Lagrangian Duality

Primal

$$\min\ f(x)$$
$$g(x) \geq 0$$
$$x \in S$$

Dual

$$\max\ v$$
$$g(x) \geq b \overset{s \in S}{\Rightarrow} f(x) \geq v$$

Surrogate

Let us say that

$g(x) \geq 0 \overset{x \in S}{\Rightarrow} f(x) \geq v$ iff $\boxed{\lambda g(x) \geq 0}$ $\boxed{\text{dominates}}$ $f(x) - v \geq 0$

for some $\lambda \geq 0$

$\lambda g(x) \leq f(x) - v$ for all $x \in S$

That is, $v \leq f(x) - \lambda g(x)$ for all $x \in S$

Or $v \leq \min\limits_{x \in S} \{ f(x) - \lambda g(x) \}$

If we replace domination with material implication,
we get the **surrogate dual**, which gives better bounds
but lacks the nice properties of the Lagrangean dual.

Lagrangian Duality

Primal

$$\min \; f(x)$$
$$g(x) \geq 0$$
$$x \in S$$

Dual

$$\max \; v$$
$$g(x) \geq b \overset{s \in S}{\Rightarrow} f(x) \geq v$$

Surrogate

Let us say that

$g(x) \geq 0 \overset{x \in S}{\Rightarrow} f(x) \geq v$ iff $\boxed{\lambda g(x) \geq 0}$ $\boxed{\text{dominates}}$ $f(x) - v \geq 0$

for some $\lambda \geq 0$

$\lambda g(x) \leq f(x) - v$ for all $x \in S$

That is, $v \leq f(x) - \lambda g(x)$ for all $x \in S$

Or $v \leq \min_{x \in S} \{ f(x) - \lambda g(x) \}$

So the dual becomes

$$\max \; v$$
$$v \leq \min_{x \in S} \{ f(x) - \lambda g(x) \} \text{ for some } \lambda \geq 0$$

Logic and Duality

Classical Benders decomposition requires an LP subproblem.
The Benders cuts are obtained from the **LP dual** of the subproblem.

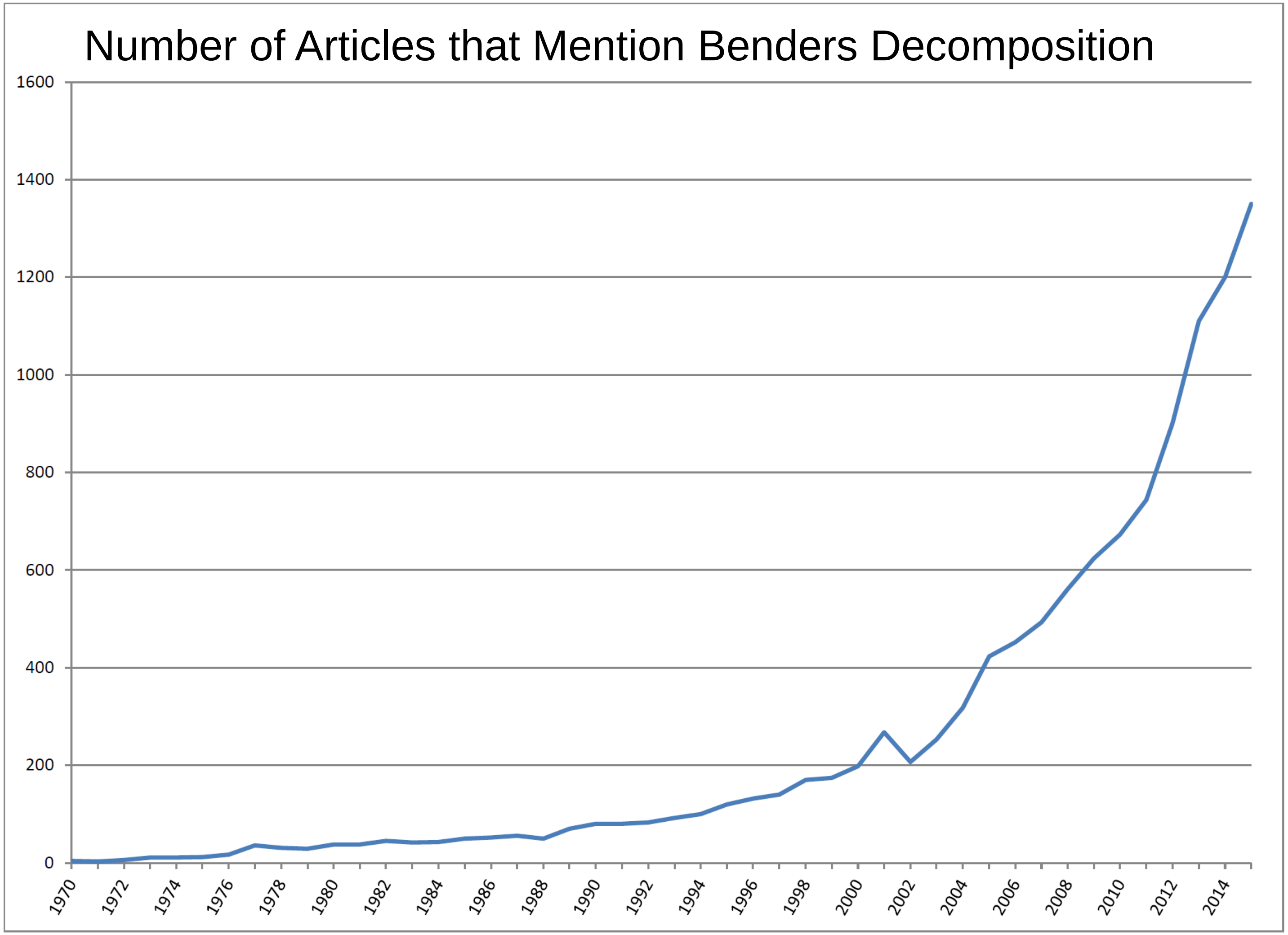
Logic-based Benders decomposition accepts **any** optimization or feasibility problem as the subproblem.

- Benders cuts are obtained from an **inference dual** of the subproblem.
- Speedup over state of the art can be several orders of magnitude.
- Benders cuts must be designed specifically for every class of problems.

JH 2000

JH, Ottosson 2003

Logic and Duality



Logic and Duality

Logic-based Benders decomposition solves a problem of the form

$$\begin{aligned}&\min f(x,y)\\&(x,y)\in S\\&x\in D_x, y\in D_y\end{aligned}$$

…where the problem **simplifies** when x is fixed to a specific value.

Logic and Duality

Decompose problem into **master** and **subproblem**.
Subproblem is obtained by fixing *x* to solution value in master problem.

Master problem

$$\min z$$
$$z \ge g_k(x) \quad \text{(Benders cuts)}$$
$$x \in D_x$$

Minimize cost *z* subject to bounds given by Benders cuts, obtained from previous iterations *k*.

Trial value $\bar{x}$ that solves master →

← Benders cut $z \ge g_k(x)$

Subproblem

$$\min f(\bar{x}, y)$$
$$(\bar{x}, y) \in S$$

Obtain proof of optimality (solution of inference dual). Use **same proof** to deduce cost bounds for other assignments, yielding Benders cut.

Logic and Duality

Iterate until master problem value equals best subproblem value so far.

Classical Benders uses LP dual of subproblem to obtain a proof.

Master problem

$$\min z$$
$$z \geq g_k(x) \quad \text{(Benders cuts)}$$
$$x \in D_x$$

Minimize cost z subject to bounds given by Benders cuts, obtained from previous iterations k.

Trial value $\bar{x}$ that solves master →

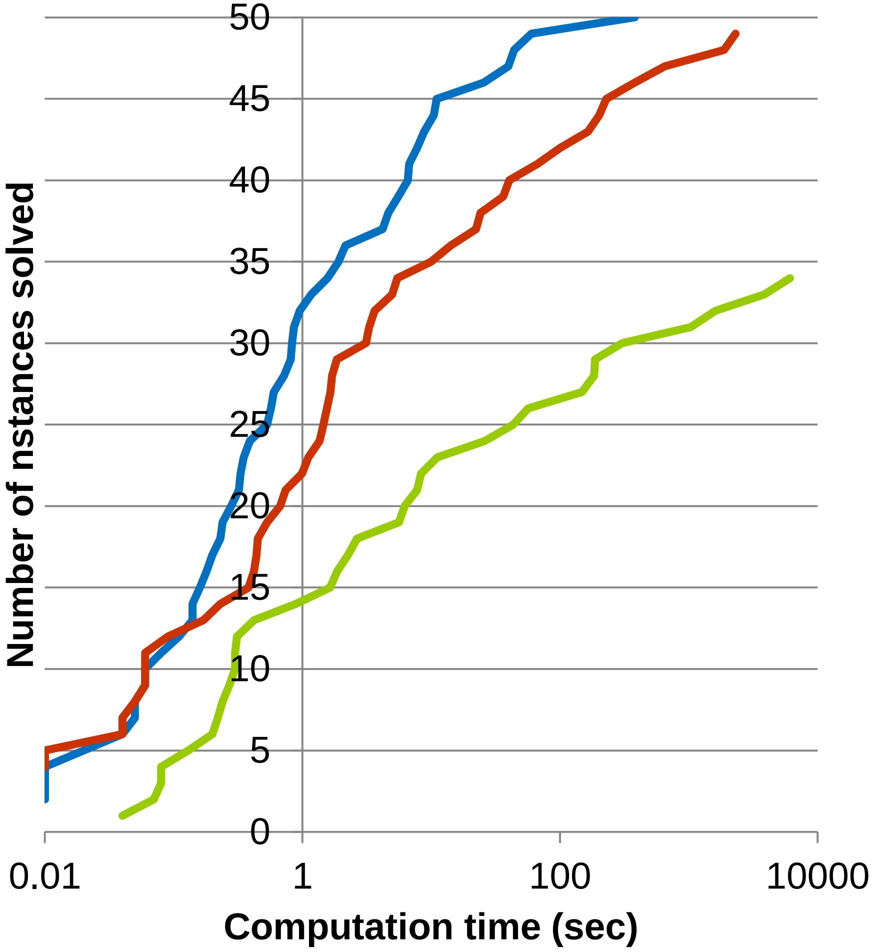
← Benders cut $z \geq g_k(x)$

Subproblem

$$\min f(\bar{x}, y)$$
$$(\bar{x}, y) \in S$$

Obtain proof of optimality (solution of inference dual). Use **same proof** to deduce cost bounds for other assignments, yielding Benders cut.

Machine Assignment and Scheduling



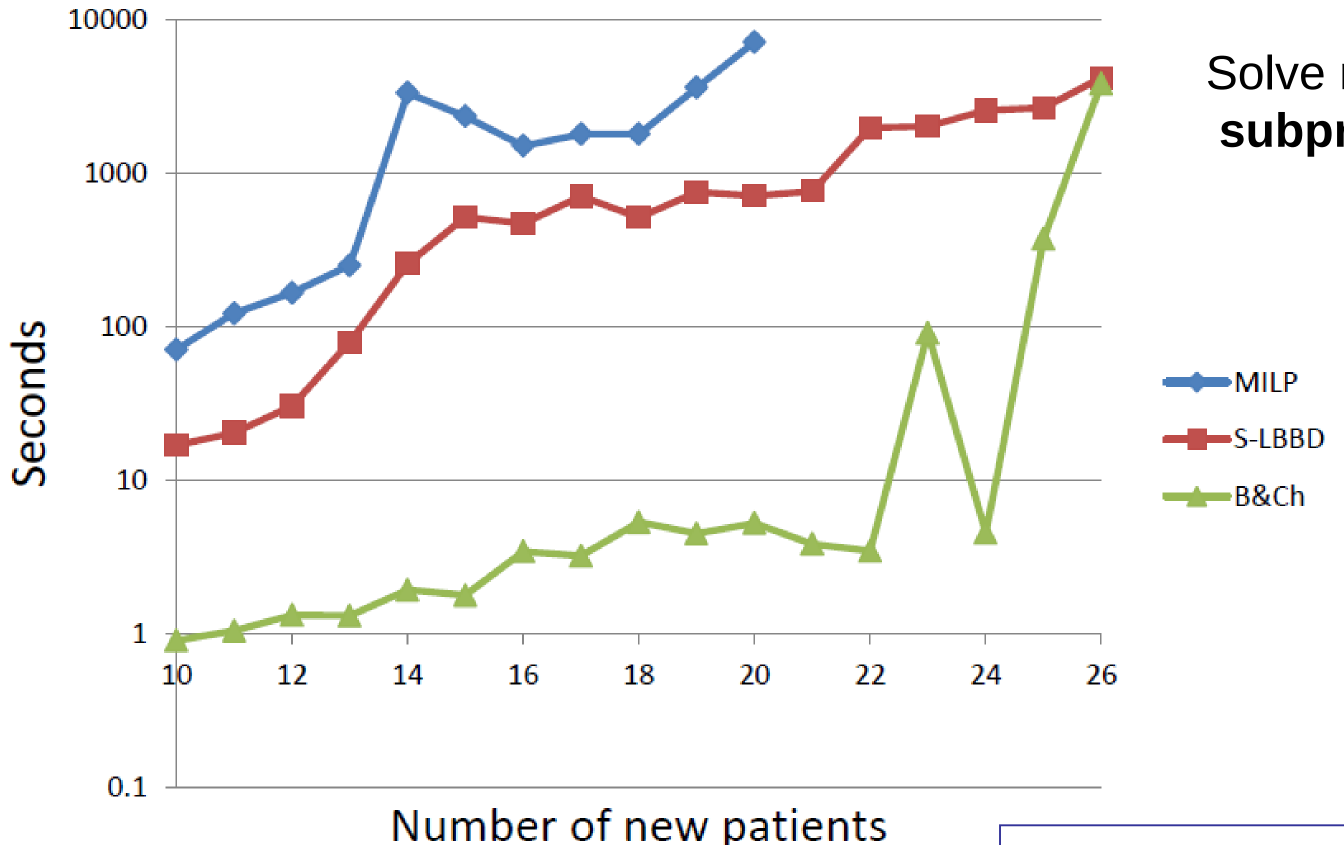
Performance profile

50 problem instances

Solve master by MIP,
subproblem by CP

Ciré, Çoban, JH 2016

Home Healthcare Routing and Scheduling



Solve master by MIP, **subproblem by CP**

Heching, JH, Kimura 2018

S-LBBD = standard LBBD
B&Ch = branch and check, variant of LBBD in which Benders cuts are generated during a single branch-and-bound solution of master problem

Logic and Duality

LBBD in planning and scheduling:

- Chemical batch processing (BASF, etc.)
- Auto assembly line management (Peugeot-Citroën)
- Allocation and scheduling of multicore processors (IBM, Toshiba, Sony)
- Steel production scheduling
- Worker assignment in a queuing environment

Logic and Duality

Other scheduling applications:

- Lock scheduling
- Shift scheduling
- Permutation flow shop scheduling
- Resource-constrained scheduling
- Hospital scheduling
- Optimal control of dynamical systems
- Sports scheduling

Logic and Duality

LBBD in routing and scheduling:

- Vehicle routing
- Home health care
- Food distribution
- Automated guided vehicles in flexible manufacturing
- Traffic diversion
- Concrete delivery

Logic and Duality

LBBD in location and design:

- Allocation of frequency spectrum (U.S. FCC)
- Wireless local area network design
- Facility location-allocation
- Stochastic facility location and fleet management
- Capacity and distance-constrained plant location
- Queuing design and control

105

Logic and Duality

Other LBBD applications:

- Logical inference (SAT solvers essentially use Benders!)
- Logic circuit verification
- Bicycle sharing
- Service restoration in a network
- Inventory management
- Supply chain management
- Space packing

Consistency and Backtracking

Consistency and Backtracking

Consistency is a core concept of **constraint programming**.

A **consistent partial assignment** is one that occurs in some feasible solution.

A **constraint set is consistent** if all partial assignments that violate no constraint are consistent with the constraint set.

Consistency and Backtracking

Consistency is a core concept of **constraint programming**.

A **consistent partial assignment** is one that occurs in some feasible solution.

A **constraint set is consistent** if all partial assignments that violate no constraint are consistent with the constraint set.

Various **forms** of consistency: full consistency, *k*-consistency, domain consistency.

Consistency implies **less backtracking**

Consistency and Backtracking

The concept of consistency **never developed in the optimization literature**.

Yet **valid inequalities** (cutting planes) reduce backtracking by achieving a greater degree of consistency, as well as by tightening a relaxation.

Consistency and Backtracking

The concept of consistency **never developed in the optimization literature**.

Yet **valid inequalities** (cutting planes) reduce backtracking by achieving a greater degree of consistency, as well as by tightening a relaxation.

Consistency can be **adapted to MILP**.

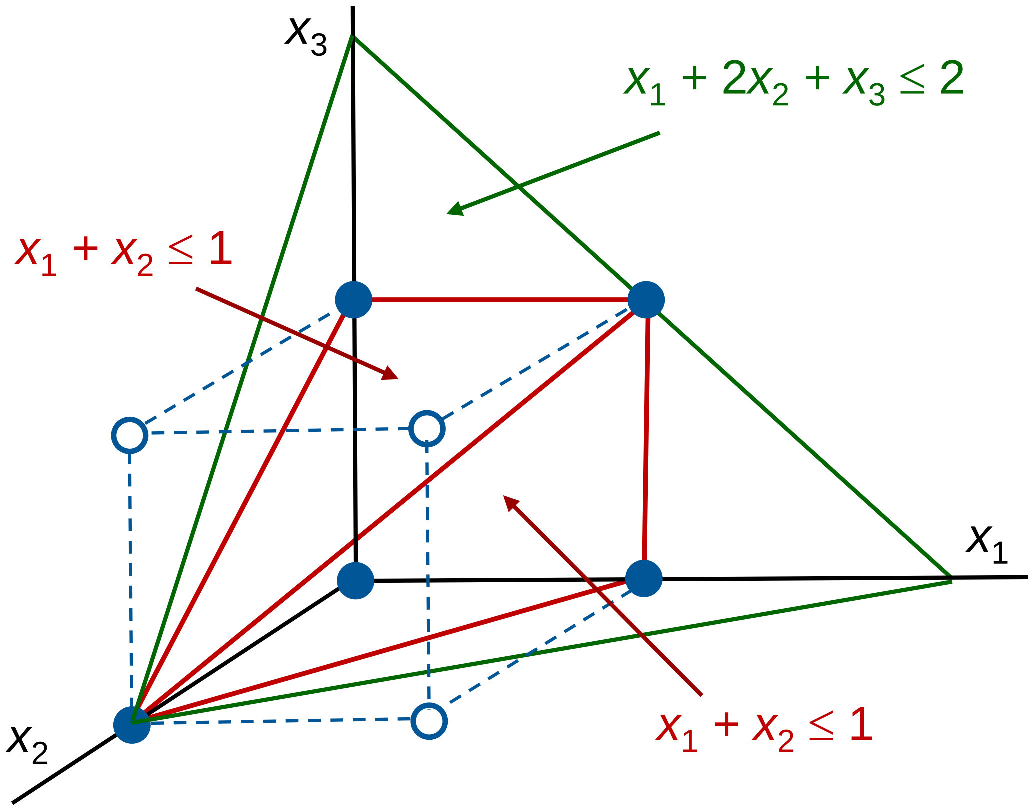
Cuts that achieve consistency **cut off inconsistent 0-1 partial assignments** and so reduce backtracking.

Consistency and Backtracking

$$S = \Big\{ x_1 + 2x_2 + x_3 \le 2,\ x_j \in \{0,1\} \Big\}$$

This inequality is the **sum** of the 2 nontrivial facet-defining inequalities for *S* and so is **“weaker.”**

Yet it cuts off **more infeasible 0-1 points** than either facet-defining inequality.



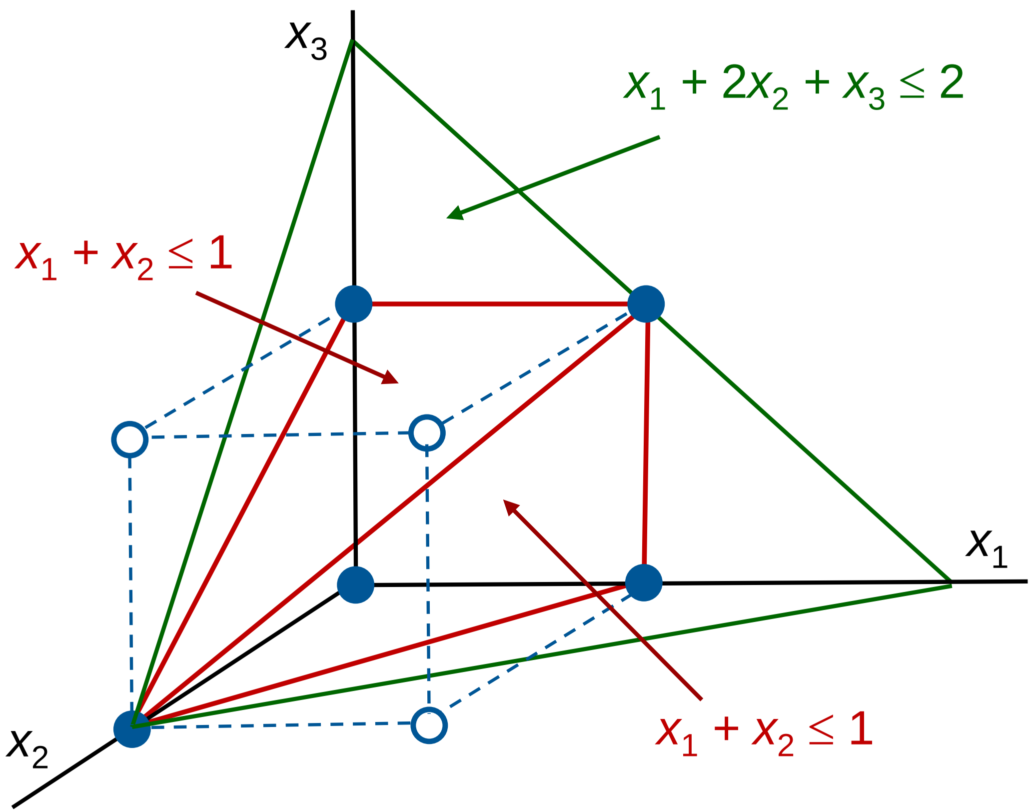
Consistency and Backtracking

$$S = \left\{ x_1 + 2x_2 + x_3 \le 2,\ x_j \in \{0,1\} \right\}$$

The constraint set S is **LP-consistent**.

It explicitly excludes infeasible 0-1 partial assignments.

A weak form of LP-consistency can **reduce backtracking** by excluding inconsistent partial assignments that facet-defining inequalities may not exclude.

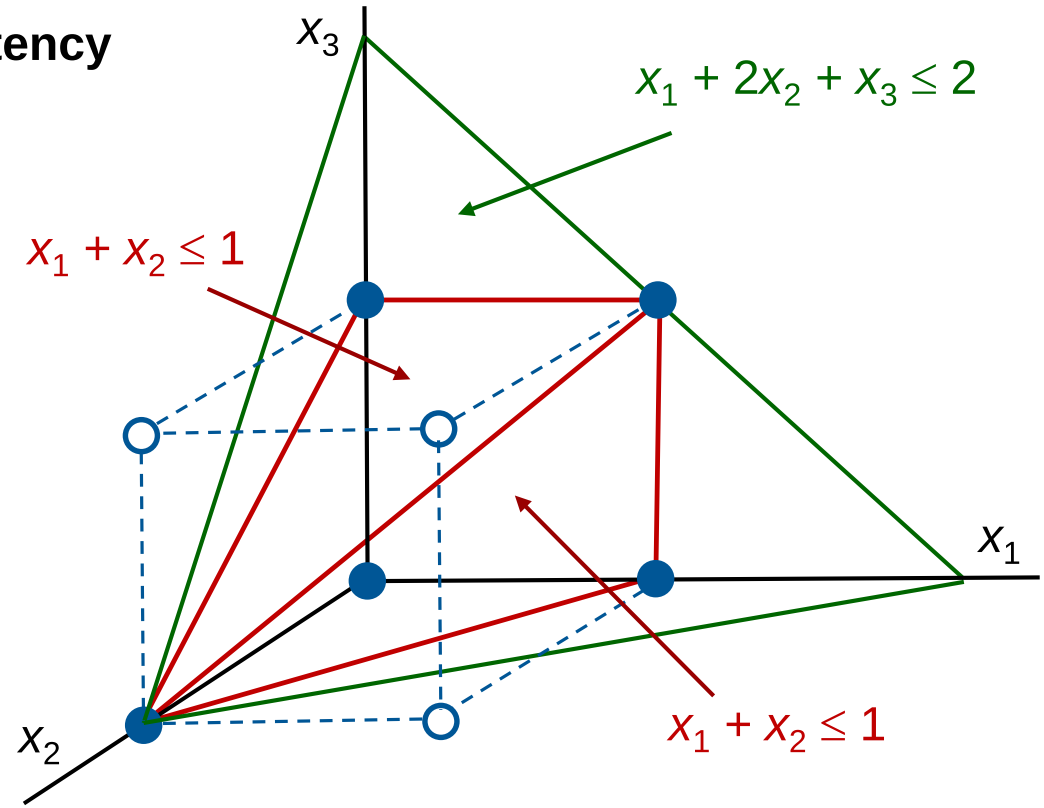


Consistency and Backtracking

$$S = \left\{ x_1 + 2x_2 + x_3 \leq 2,\ x_j \in \{0, 1\} \right\}$$

We obtain a **theory of consistency** parallel to the one in CP.

Details in my INFORMS talk.
Next session, TC04.



Questions? Comments?