Recent Developments in Logic-Based Methods for Optimization

John Hooker
Carnegie Mellon University

Santa Fe Institute
July 2019

Logic and Optimization

- Boole’s probability logic and linear programming
- Decision diagrams and optimization
- Predicate logic and integer programming
- Resolution and cutting planes
- Logic and duality
- Consistency and backtracking

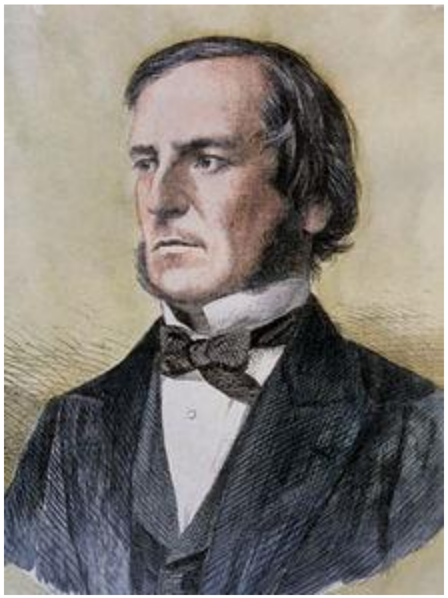
Logic and Optimization

- Boole's probability logic and linear programming
- **Decision diagrams and optimization**
- Predicate logic and integer programming
- Resolution and cutting planes
- Logic and duality
- Consistency and backtracking

Focus on decision diagrams due to possible synergy with quantum computation.

Probability Logic and Linear Programming

Probability Logic and Linear Programming



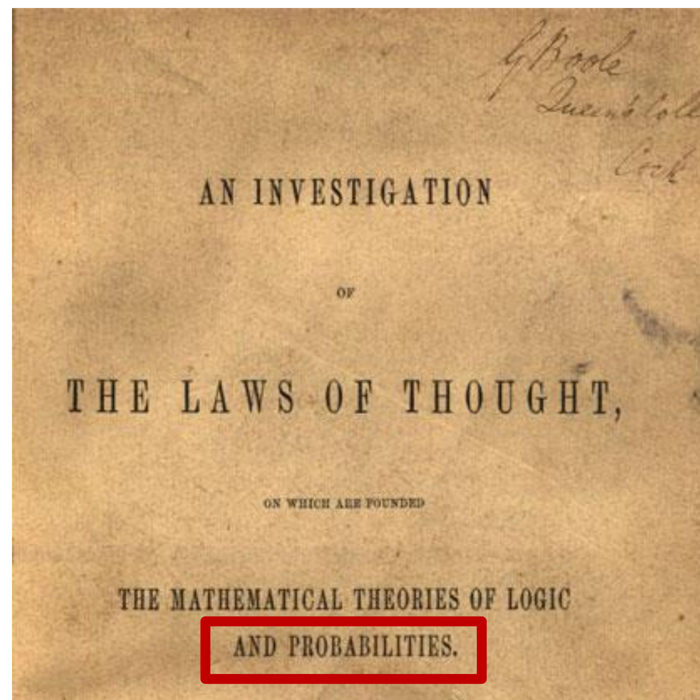
George Boole is best known for Boolean logic.

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

.
…**probability logic.**

It was forgotten or ignored for over 100 years.

Boole 1854



Probability Logic and Linear Programming

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

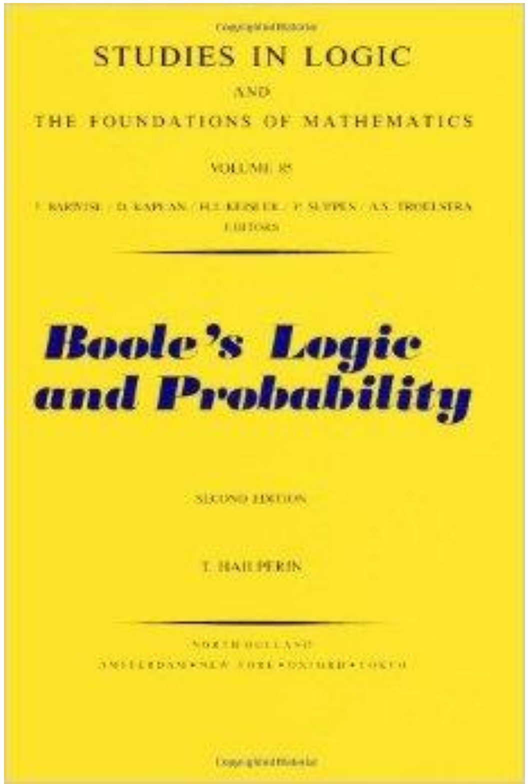
He sees this as implicit in Boole's own work.

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

Nils Nilsson

Hailperin 1976, 1984, 1986

Nilsson 1986



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

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

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

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

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

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

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

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

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.

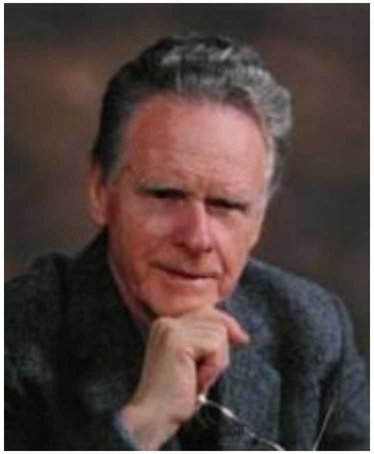
The result is a **range** of probabilities for C: 0.1 to 0.4

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

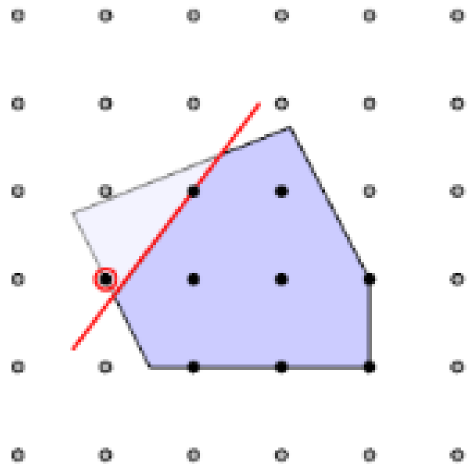
Glenn Shafer

Decision Diagrams and Optimization

Decision Diagrams and Optimization

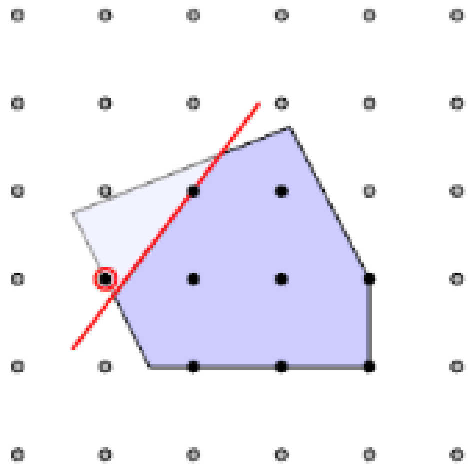
Motivation

- **Mixed integer programming** is mainstream state of the art in combinatorial optimization.
    - Goal: solve NP-hard problems to **proven optimality.**



Decision Diagrams and Optimization

Motivation

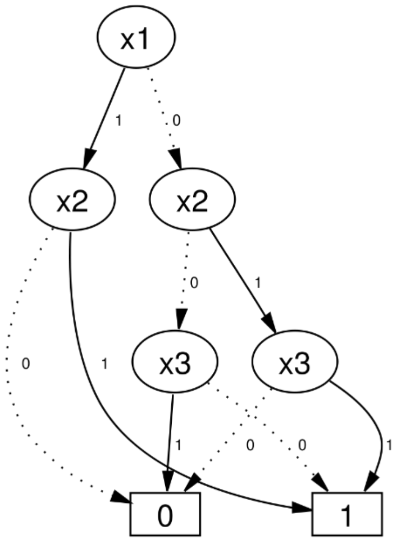


- **Mixed integer programming** is mainstream state of the art in combinatorial optimization.
    - Goal: solve NP-hard problems to **proven optimality**
    - Versatile **modeling** framework (linear integer inequalities)
    - Accommodates complex **constraints**.
    - Solvers developed over decades, extremely **well engineered** (CPLEX, Gurobi, SCIP)
    - Solvers follow (algorithmic) **Moore's Law**, but reaching plateau.
    - Basic operation is solution of a **linear programming** relaxation (complicated)
    - SAT solvers also fast, but less versatile modeling

Decision Diagrams and Optimization

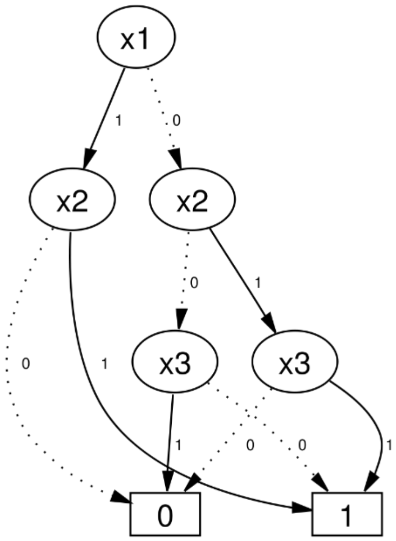
Motivation

- **Mixed integer programming** is mainstream state of the art in combinatorial optimization
  - Basic operation is solution of a **linear programming** relaxation (complicated)
- Recent development: Discrete optimization with **decision diagrams**
  - Goal: solve to **proven optimality**



Decision Diagrams and Optimization

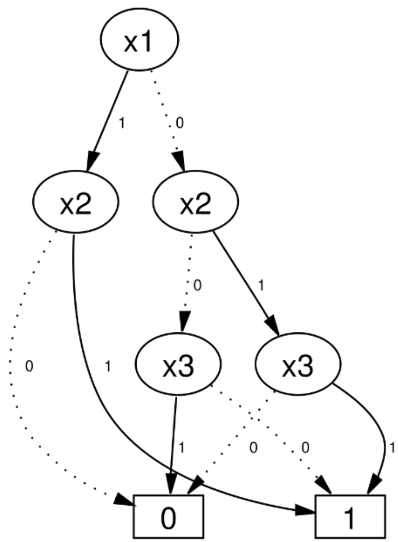
Motivation



- **Mixed integer programming** is mainstream state of the art in combinatorial optimization
  - Basic operation is solution of a **linear programming** relaxation (complicated)
- Recent development: Discrete optimization with **decision diagrams**
  - Goal: solve to **proven optimality**
  - Versatile **modeling** (recursive/dynamic programming)
  - Accommodates complex **constraints** (no need for linearity/convexity)
  - Basic operation is solution of **shortest path problem** (very simple)
  - Highly **parallelizable**.
  - Compute shortest paths with **quantum machine**?
  - Possible **killer app** for quantum computing?

Decision Diagrams and Optimization

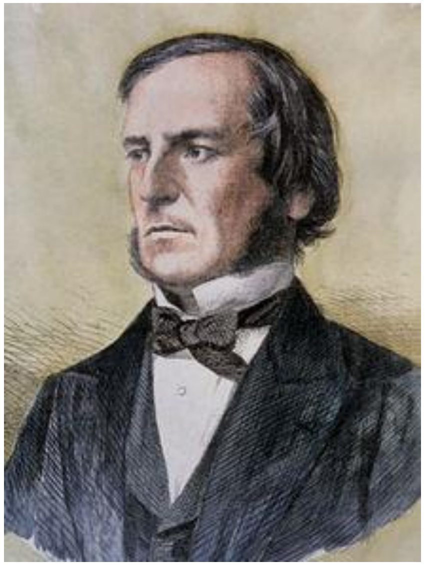
First, some background on **decision diagrams**.



Decision Diagrams

Boolean logic was 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.**



G. Boole

Boole 1847, 1854

Pearce 1886

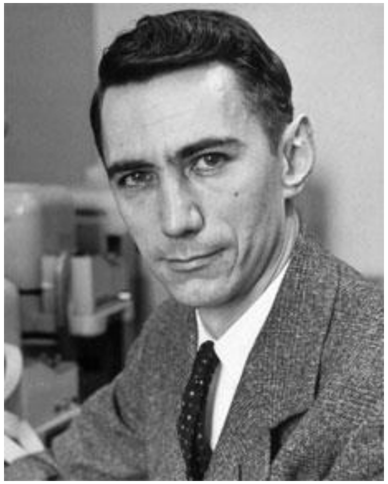
C. S. Pearce

Decision Diagrams

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

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

Shannon 1940



C. Shannon

Decision Diagrams

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

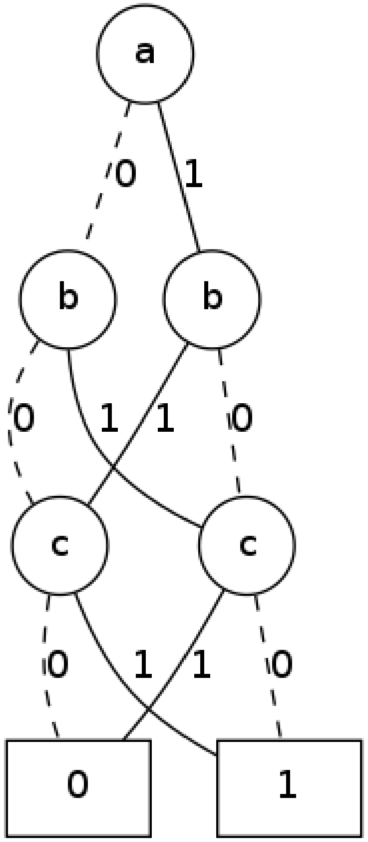
Lee 1959

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

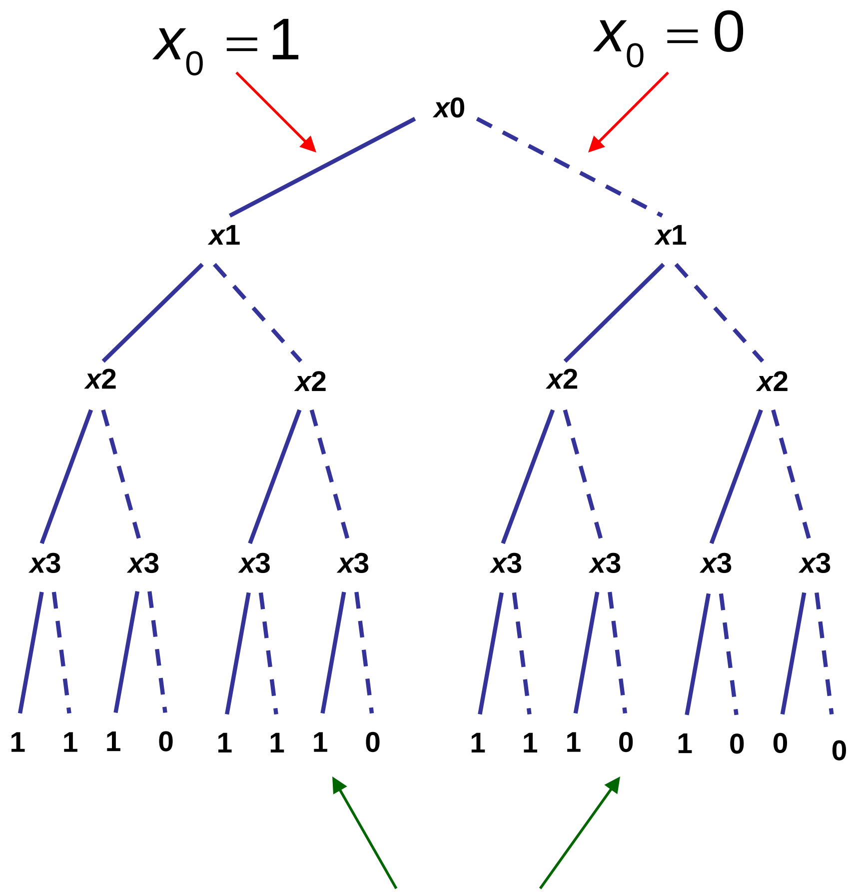
There is a **unique reduced** DD representing any given Boolean function, once the variable ordering is specified.

Bryant (1986)

Randy Bryant

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

Superimpose isomorphic subtrees and remove redundant nodes.

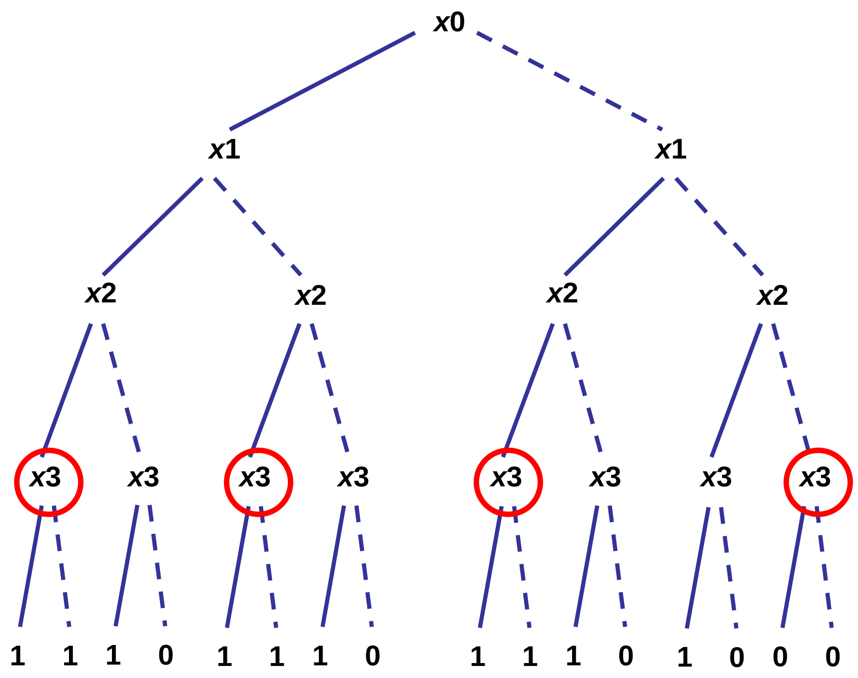


1 indicates feasible solution,
0 infeasible

Branching tree for 0-1 inequality

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

1

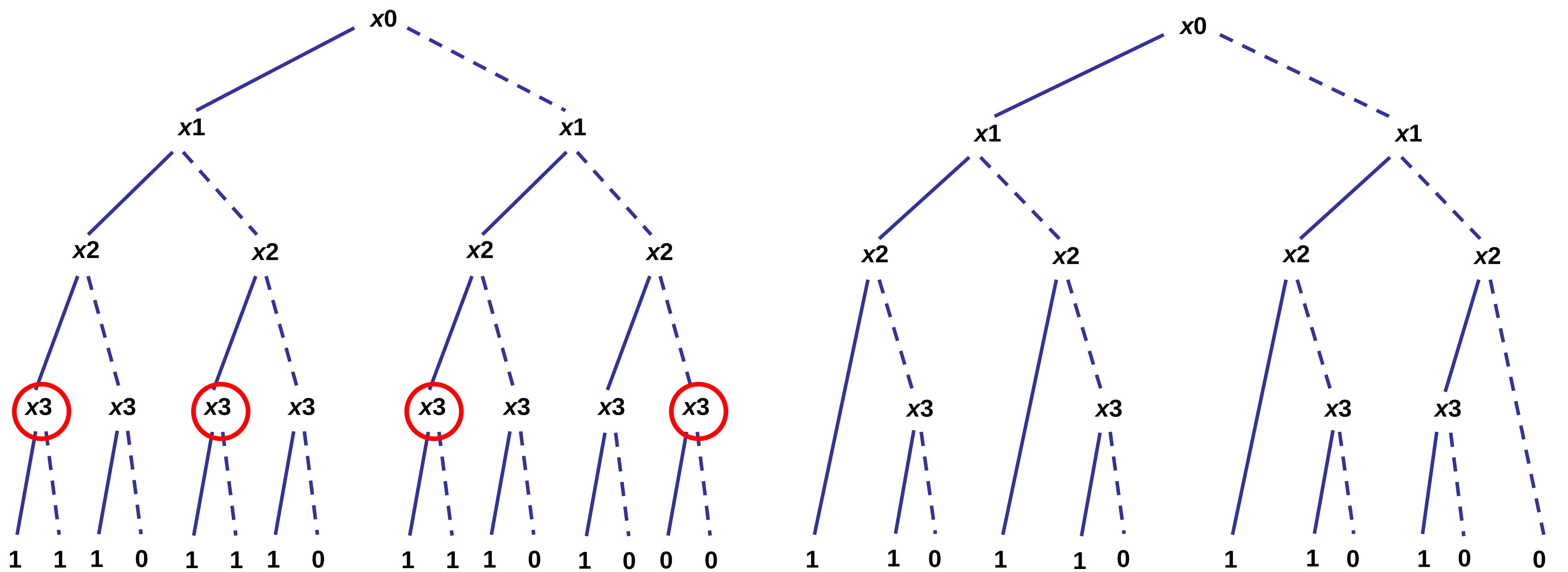


Branching tree for 0-1 inequality

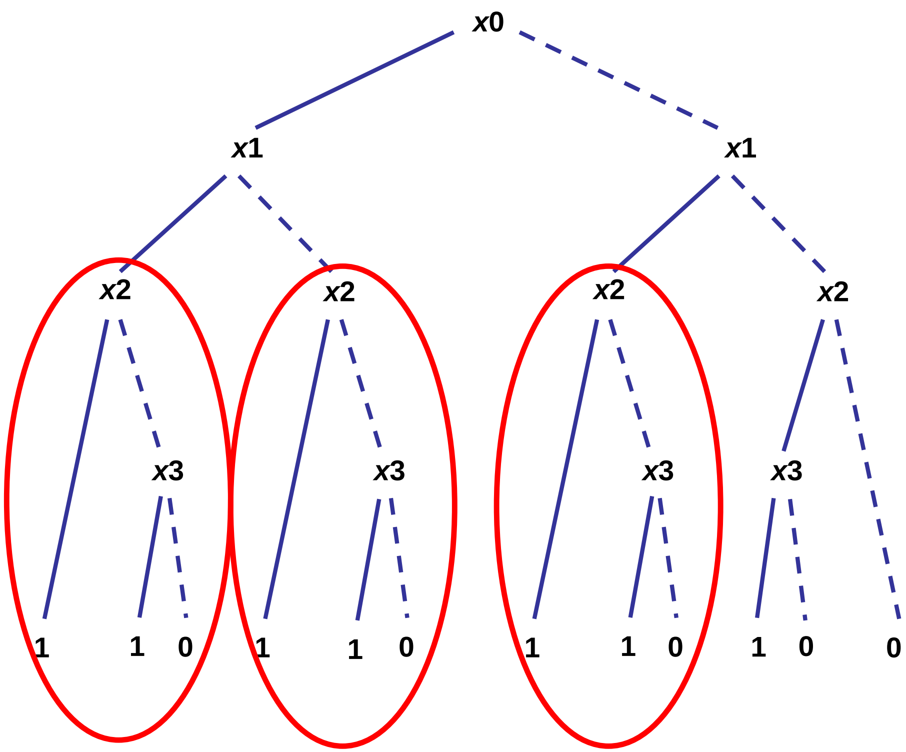
$$2x_0 + 3x_1 + 5x_2 + 5x_3 \geq 7$$

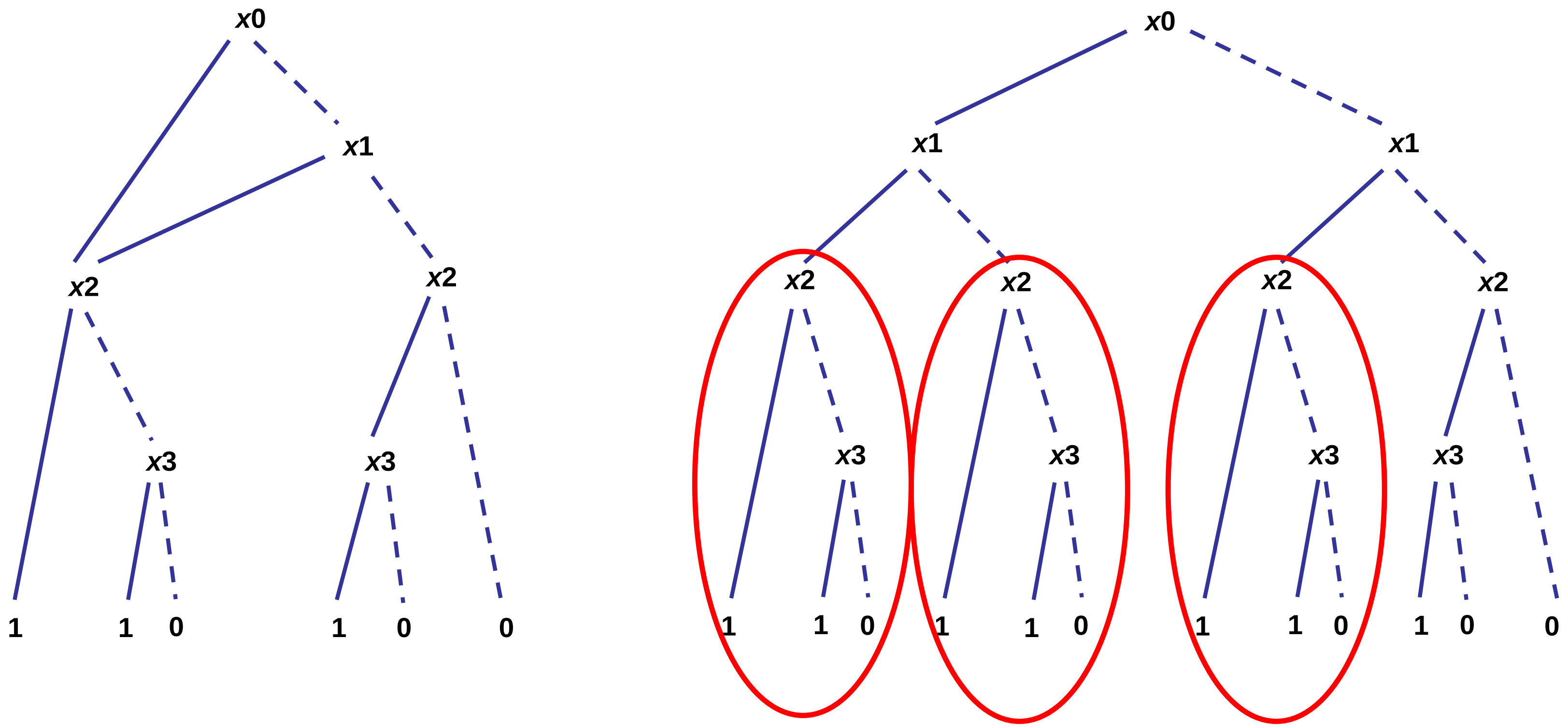
Remove redundant nodes…

1

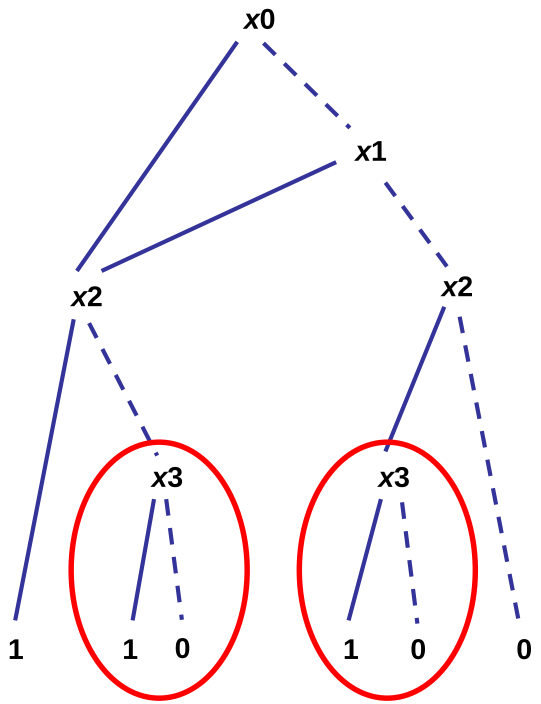

x0
x1
x2
x3
1
0

Superimpose identical subtrees…

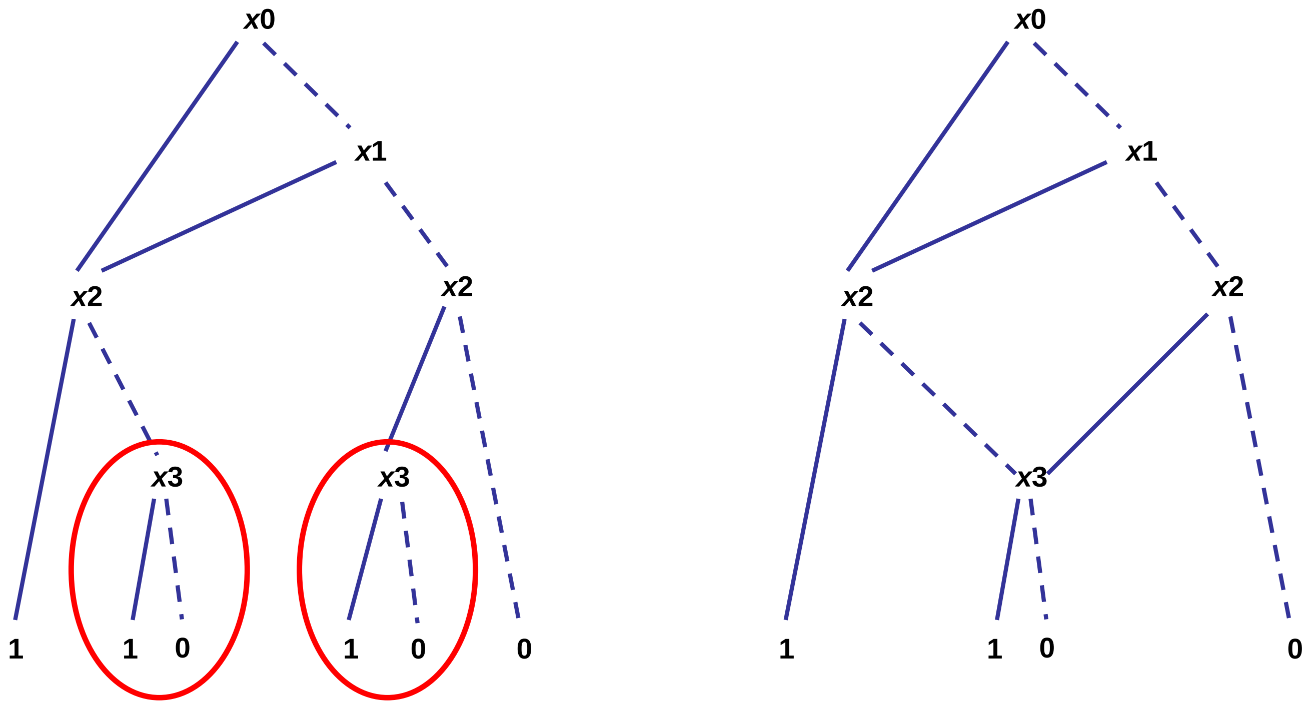




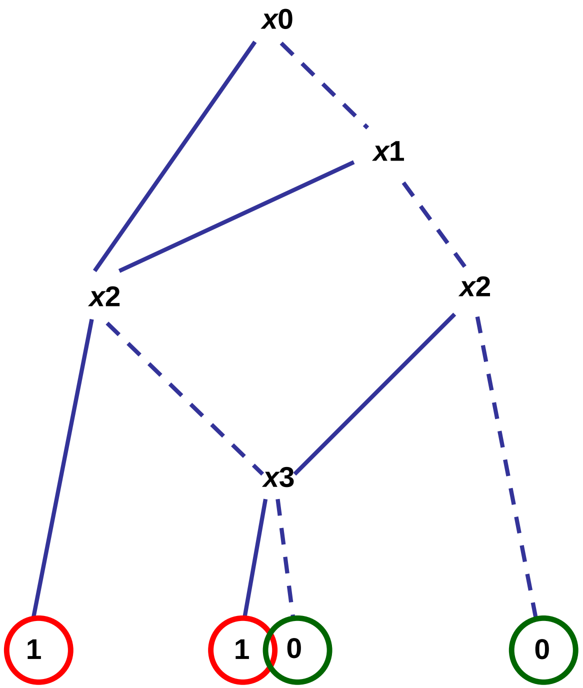
x0
x1
x2
x2
x3
x3
1
1
0
1
0
0
x0
x1
x1
x2
x2
x2
x2
x3
x3
x3
x3
1
1
0
1
1
0
1
1
0
1
0
0

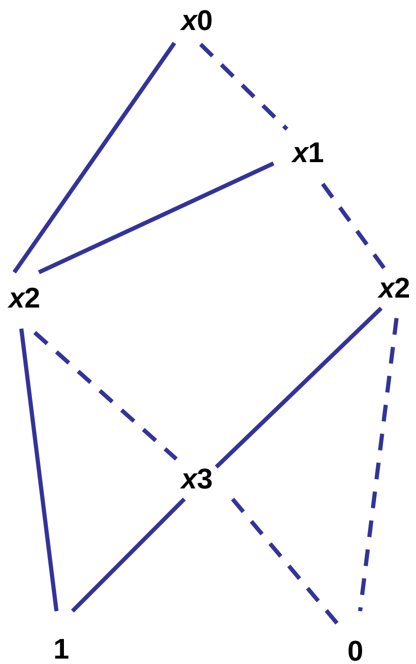


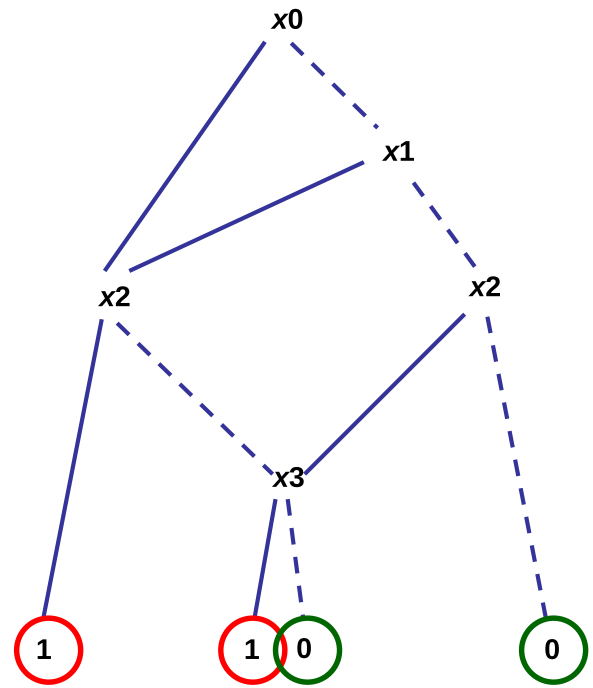
Superimpose identical subtrees…

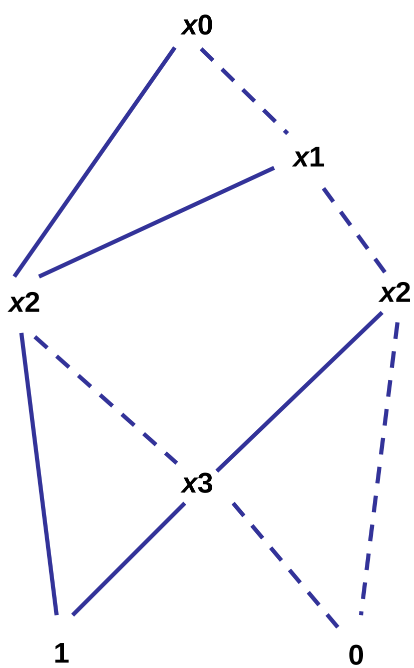

x0
x1
x2
x2
x3
x3
1
1
0
1
0
0
x0
x1
x2
x2
x3
1
1
0
0

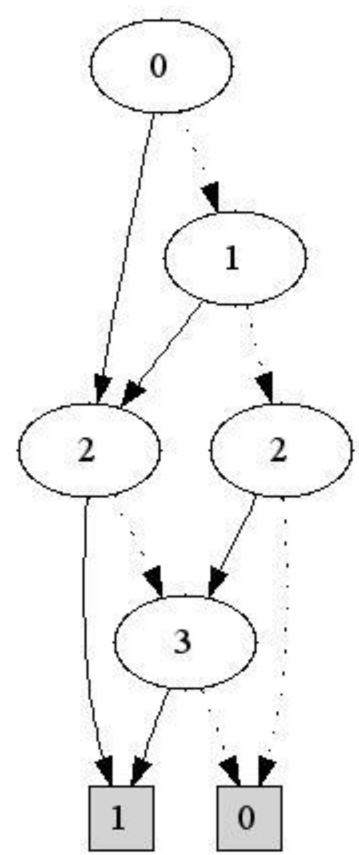
Superimpose identical leaf nodes…




x0
x1
x2
x2
x3
1
0


x0
x1
x2
x2
x3
1
1
0
0





as generated by software

Decision Diagrams and Optimization

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

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

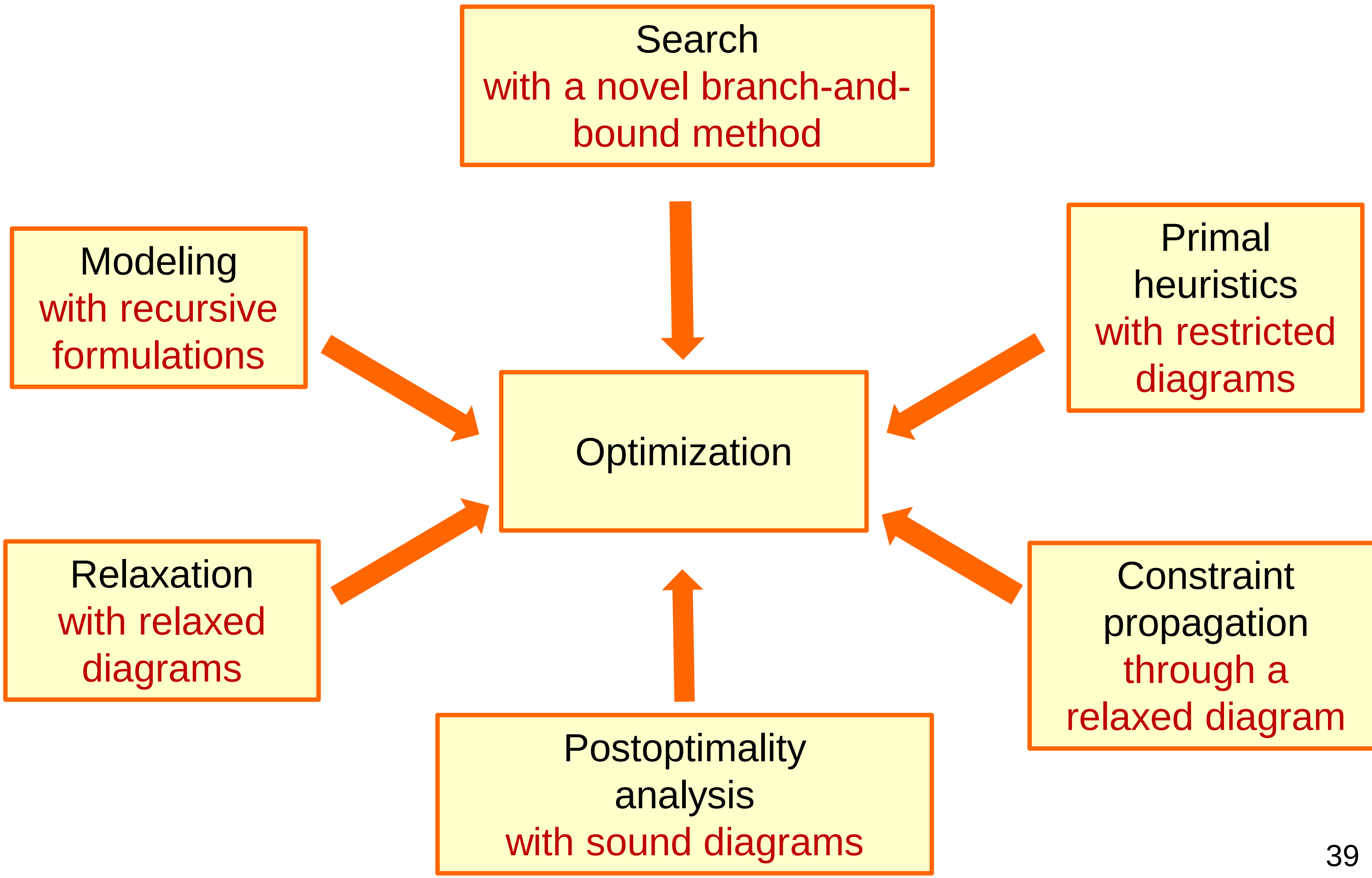
Hadžić and JH (2006, 2007)

Andersen, Hadžić, JH and Tiedemann (2007)

Tarik Hadžić

Henrik Reif Andersen

Decision Diagrams and Optimization



Decision Diagrams and Optimization

Weighted 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 (= weighted)
- Reduces optimization to a shortest (longest) path problem

Given a canonical distribution of arc costs (trivial to compute), Bryant's uniqueness theorem **generalizes** to weighted DDs.

JH (2013)

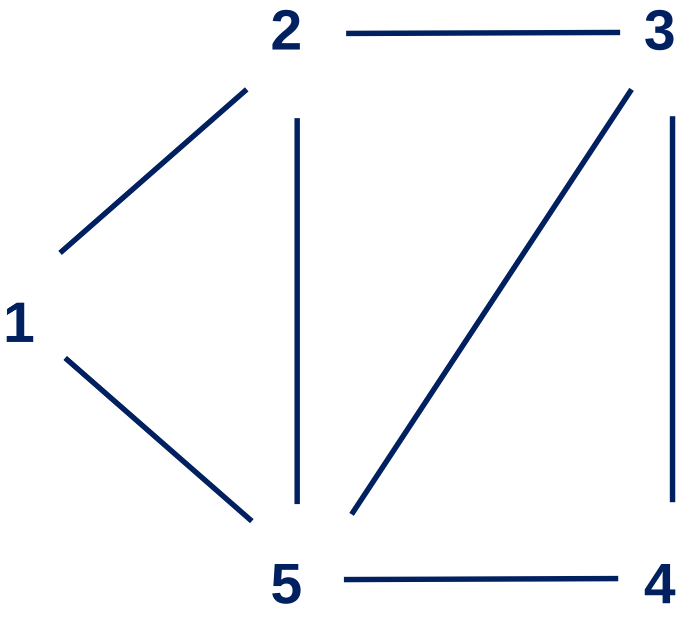
Maximal Stable Set Problem

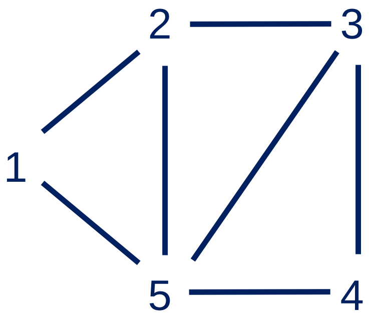
(Maximal independent set problem)

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}

x_1

x_2

x_3

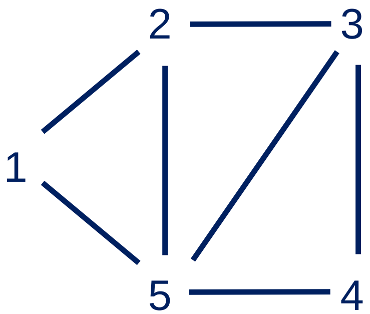
x_4

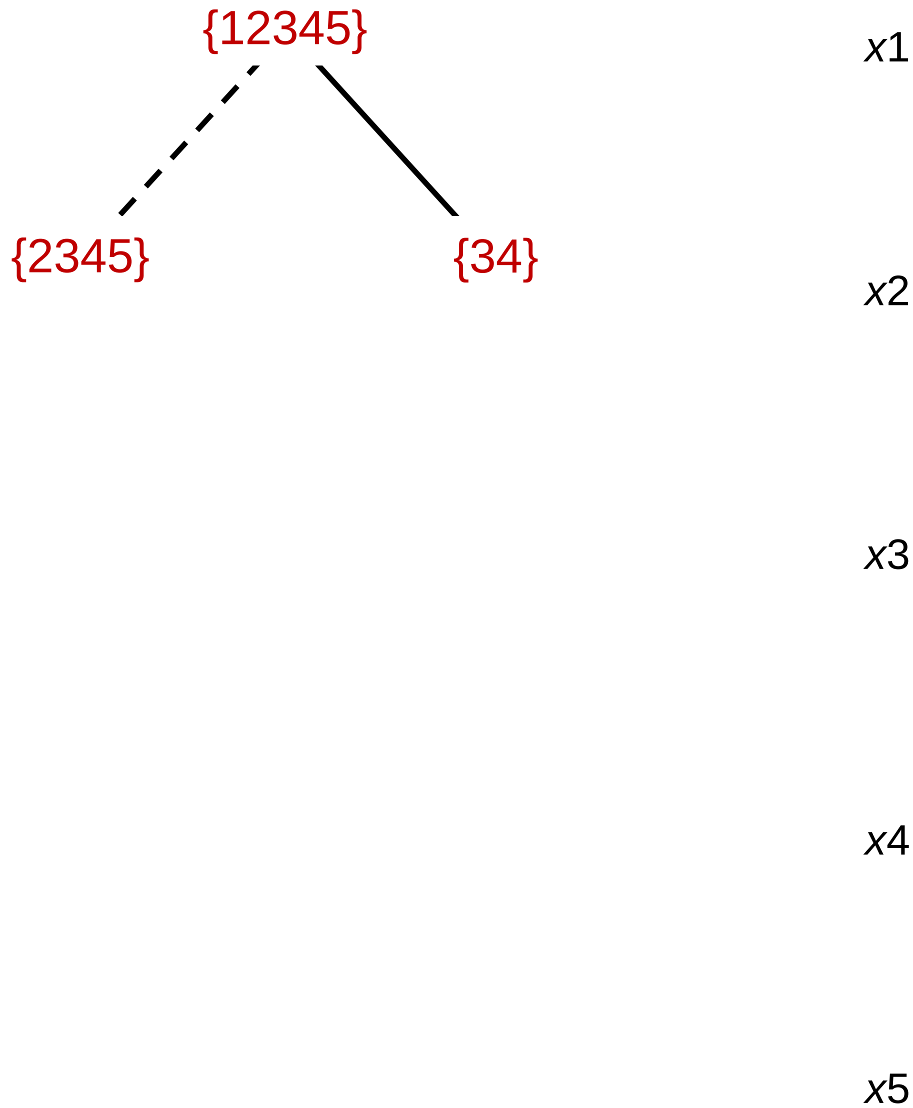
x_5

Exact DD for
stable set problem

Build DD with
top-down
compilation
(unlike CS
literature)

Associate a **state**
with each node

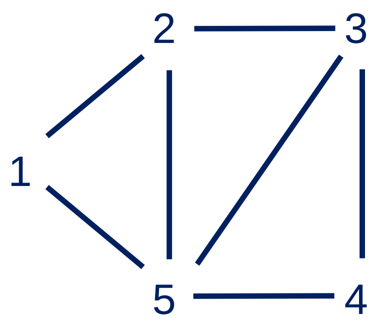


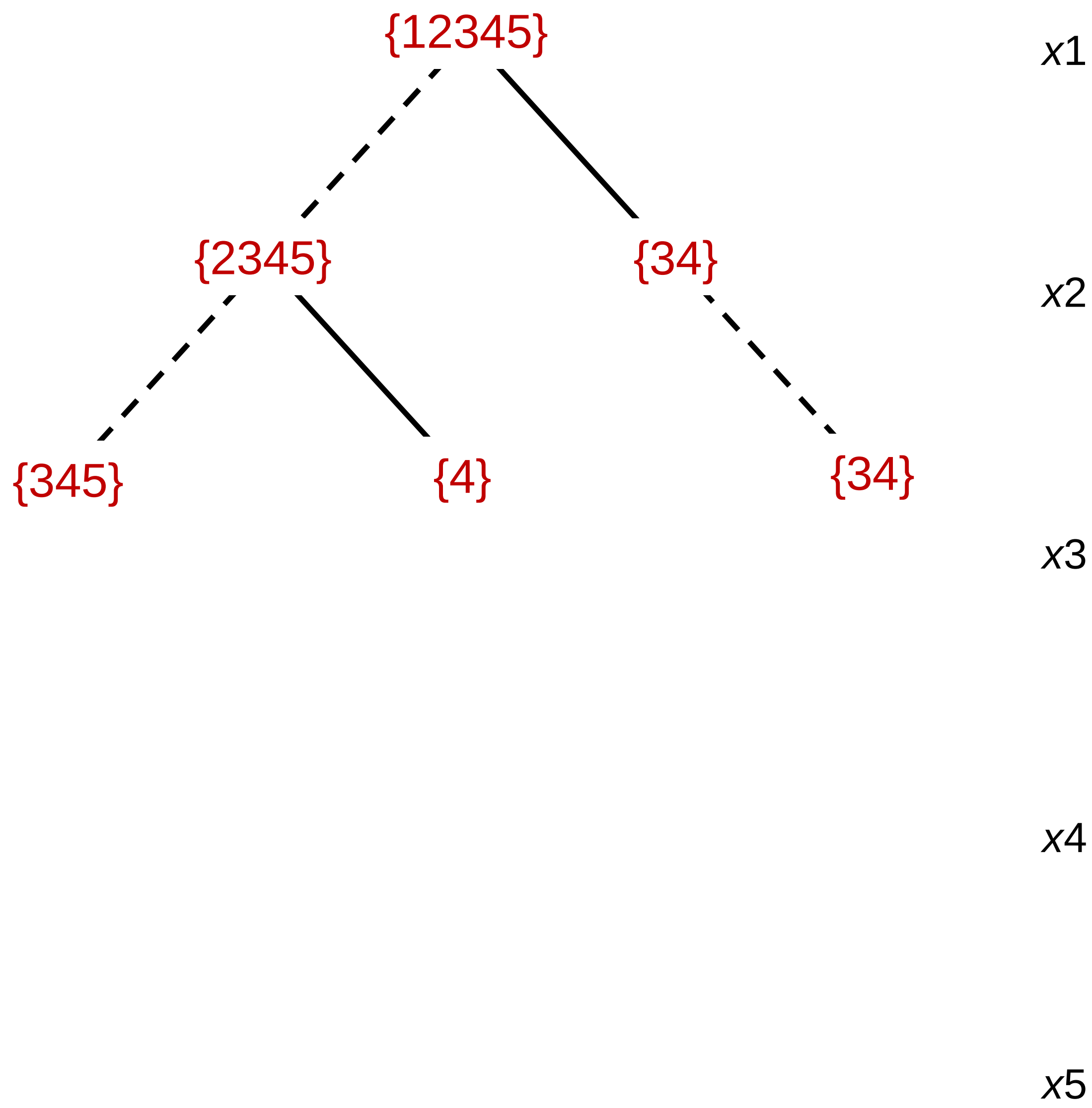


Exact DD for
stable set problem

Build DD with
top-down
compilation
(unlike CS
literature)

Associate a **state**
with each node

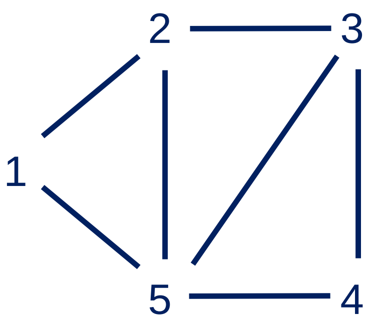


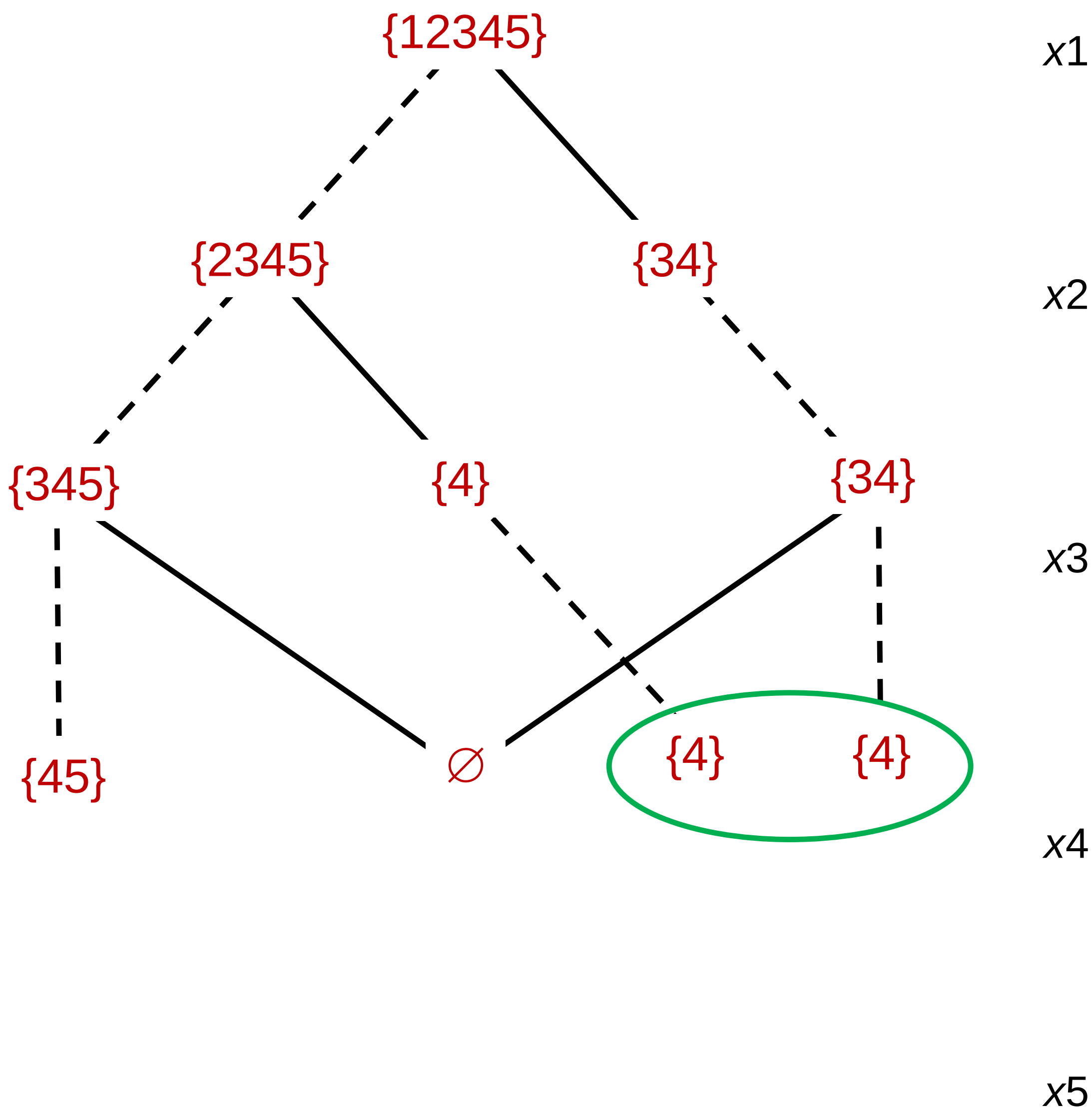


Exact DD for
stable set problem

Build DD with
top-down compilation
(unlike CS literature)

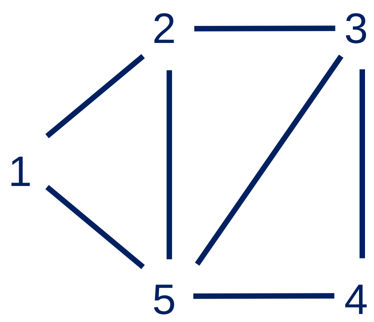
Associate a **state**
with each node

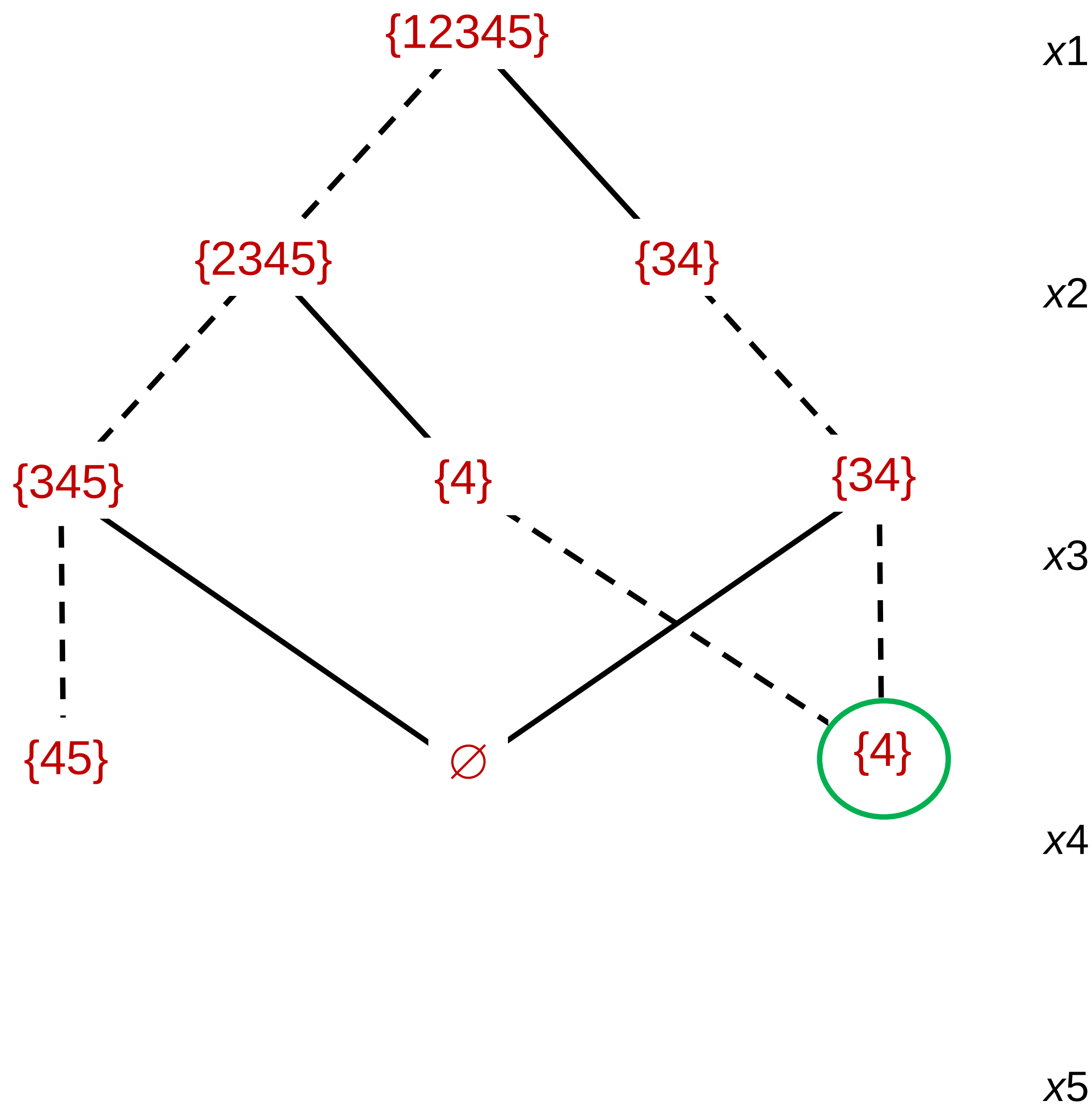




Exact DD for
stable set problem

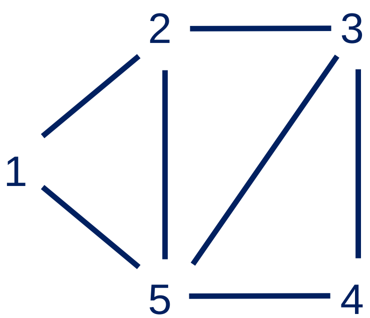
Merge nodes that
correspond to the
same state



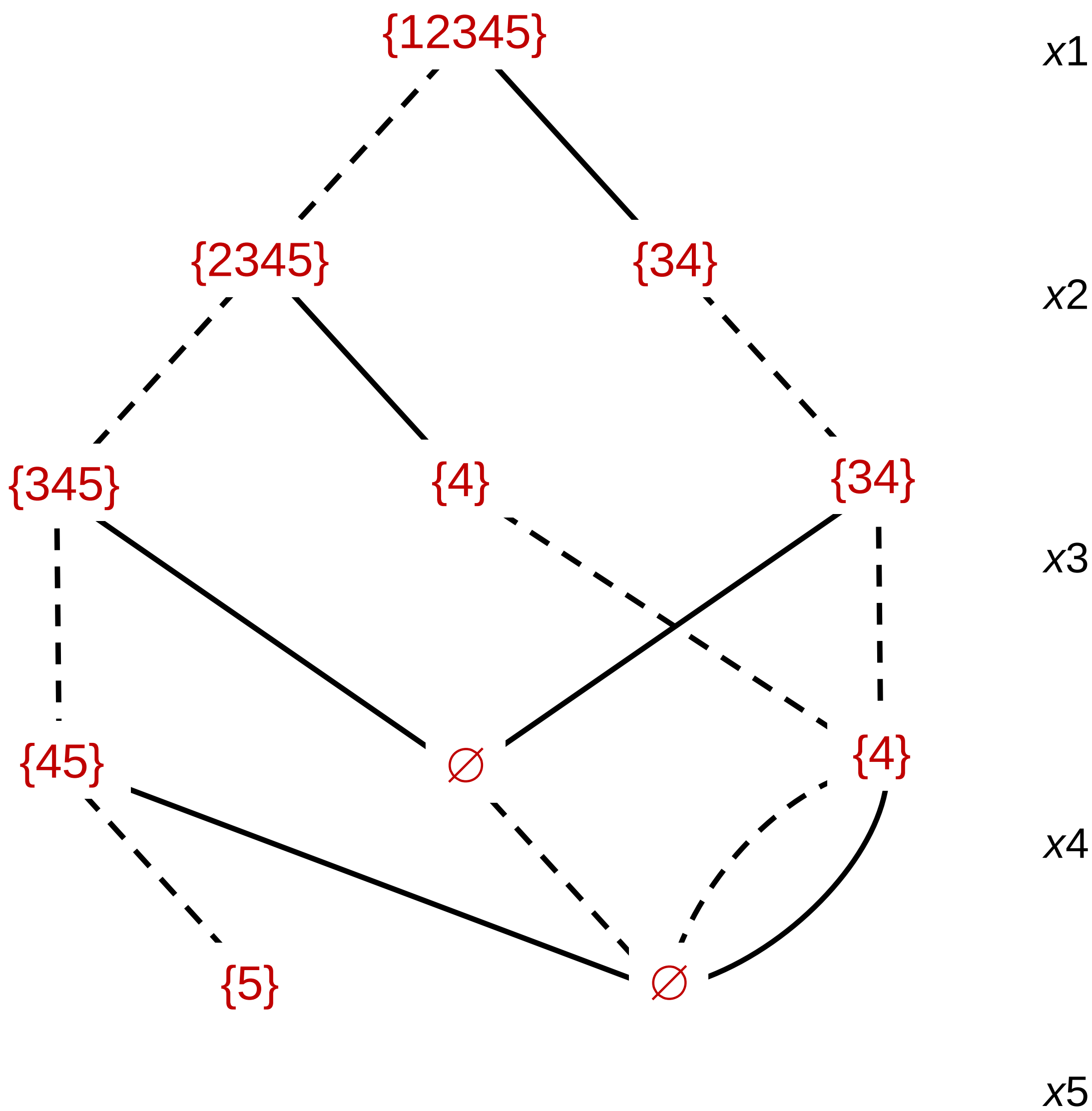


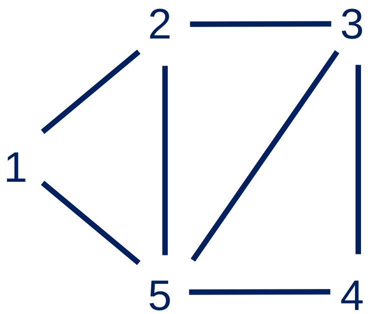
Exact DD for
stable set problem

Merge nodes that
correspond to the
same state



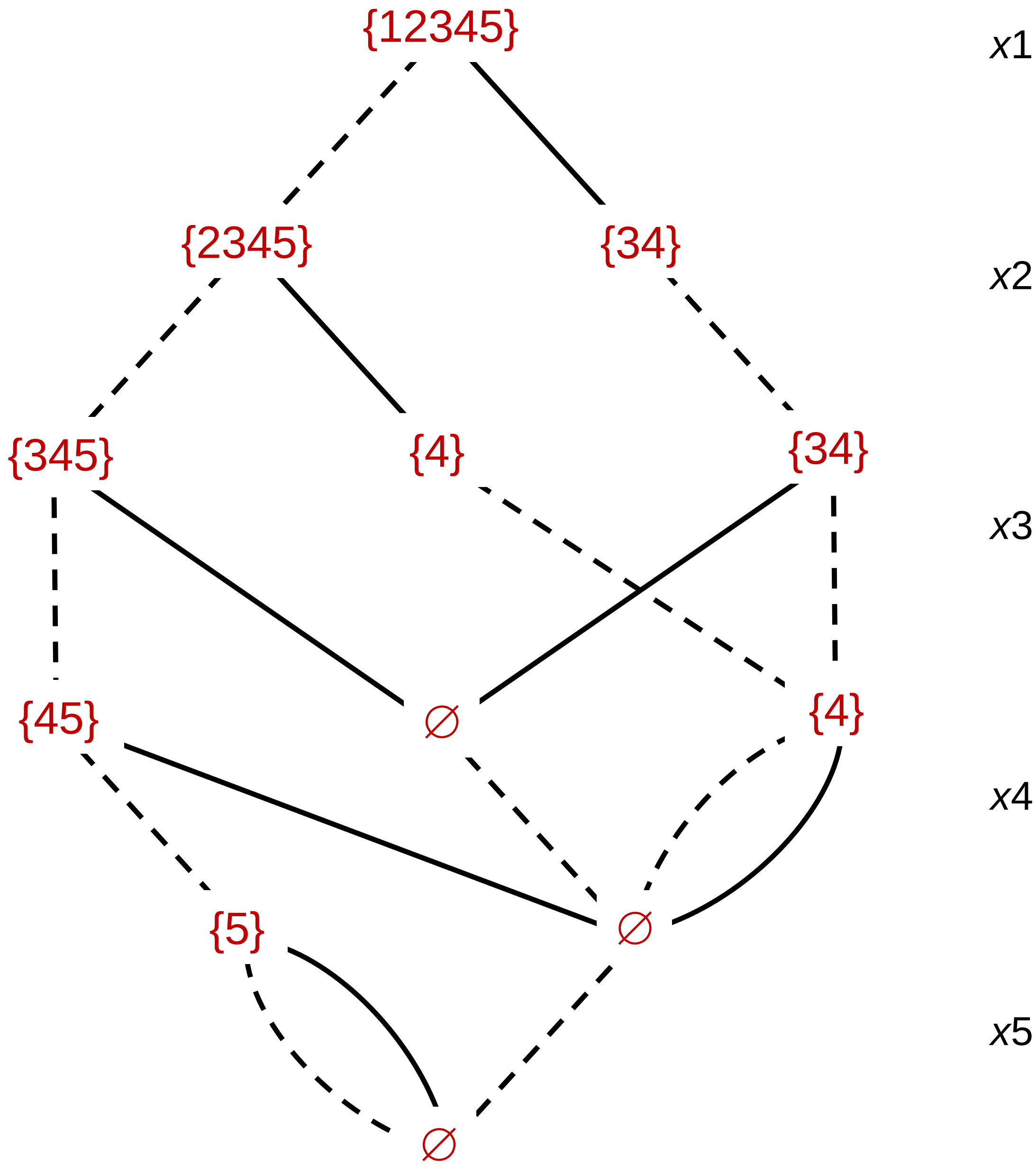
Exact DD for
stable set problem

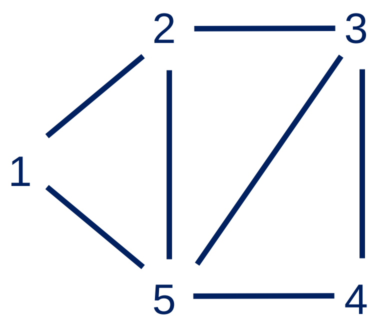


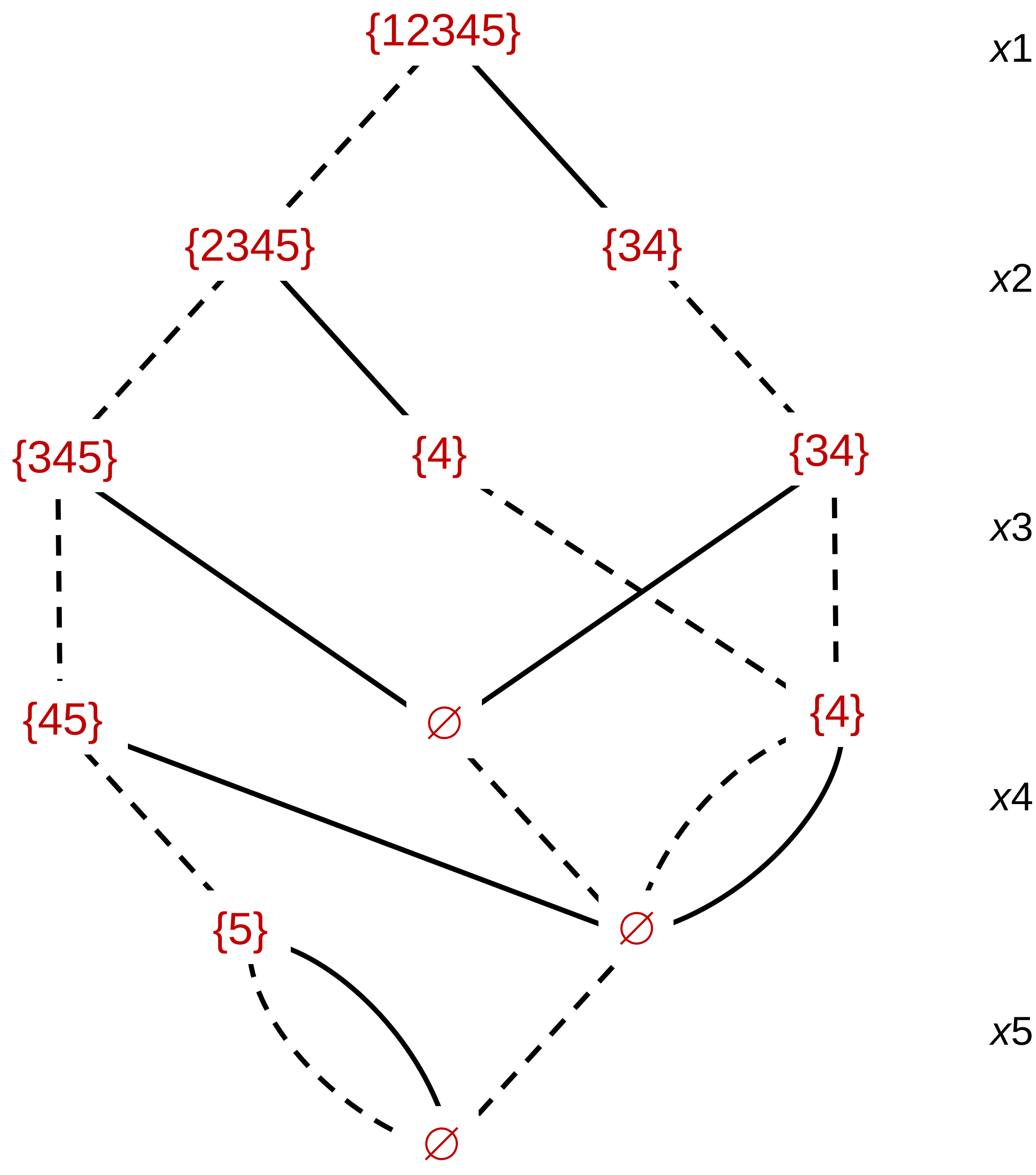


Exact DD for stable set problem

This DD can be viewed as the **state transition graph** for a **dynamic programming** formulation of the problem.







Exact DD for
stable set problem

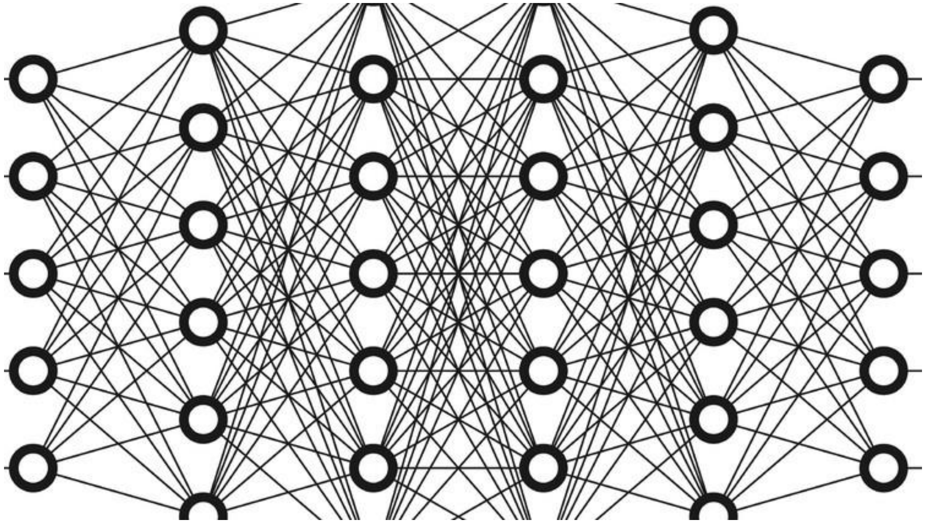
Resulting DD is
not necessarily
reduced
(it is in this case).

DD reduction is a
more powerful
simplification
method than DP

Decision Diagrams and Optimization

Decision diagrams, multilayer neural networks, and **dynamic programming** are based on the **same principle:**

The amount of information that can be represented by the network increases exponentially with the depth.



Decision Diagrams and Optimization

Nonetheless, the **width** of a DD can grow **exponentially** with the size of the problem instance**.**

Decision Diagrams and Optimization

Nonetheless, the **width** of a DD can grow **exponentially** with the size of the problem instance**.**

Solution? Use **relaxed DD** of limited width.

A relaxed DD represents a **superset** of feasible set.

Andersen, Hadžić, JH and Tiedemann (2007)

Decision Diagrams and Optimization

Nonetheless, the **width** of a DD can grow **exponentially** with the size of the problem instance**.**

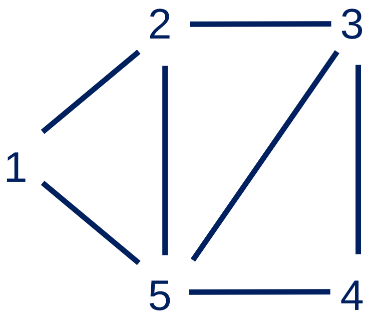
Solution? Use **relaxed DD** of limited width.

A relaxed DD represents a **superset** of feasible set.

Relaxed DDs yield **optimization bounds**.

- Shortest (longest) path length is a **bound** on optimal value.
- Paradoxically, a relaxed DD that represents **more solutions** can be **smaller**.
  - Analogous to LP relaxation in IP, but **discrete**.
  - Does **not** require **linearity**, **convexity**, or **inequality** constraints.

Andersen, Hadžić, JH and Tiedemann (2007)



{12345}

$x1$

$x2$

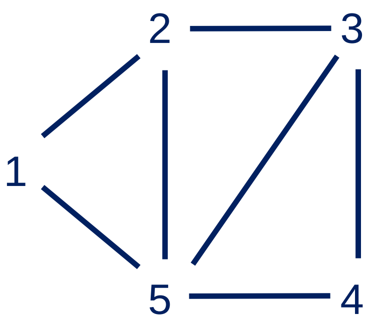
$x3$

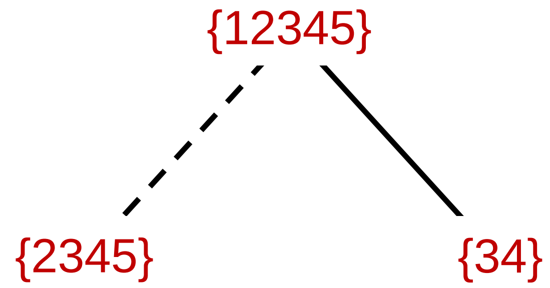
$x4$

$x5$

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

Use generic merging heuristics.





$x1$

$x2$

$x3$

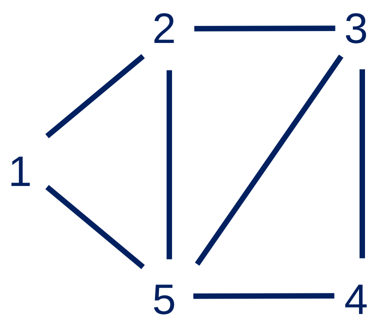
$x4$

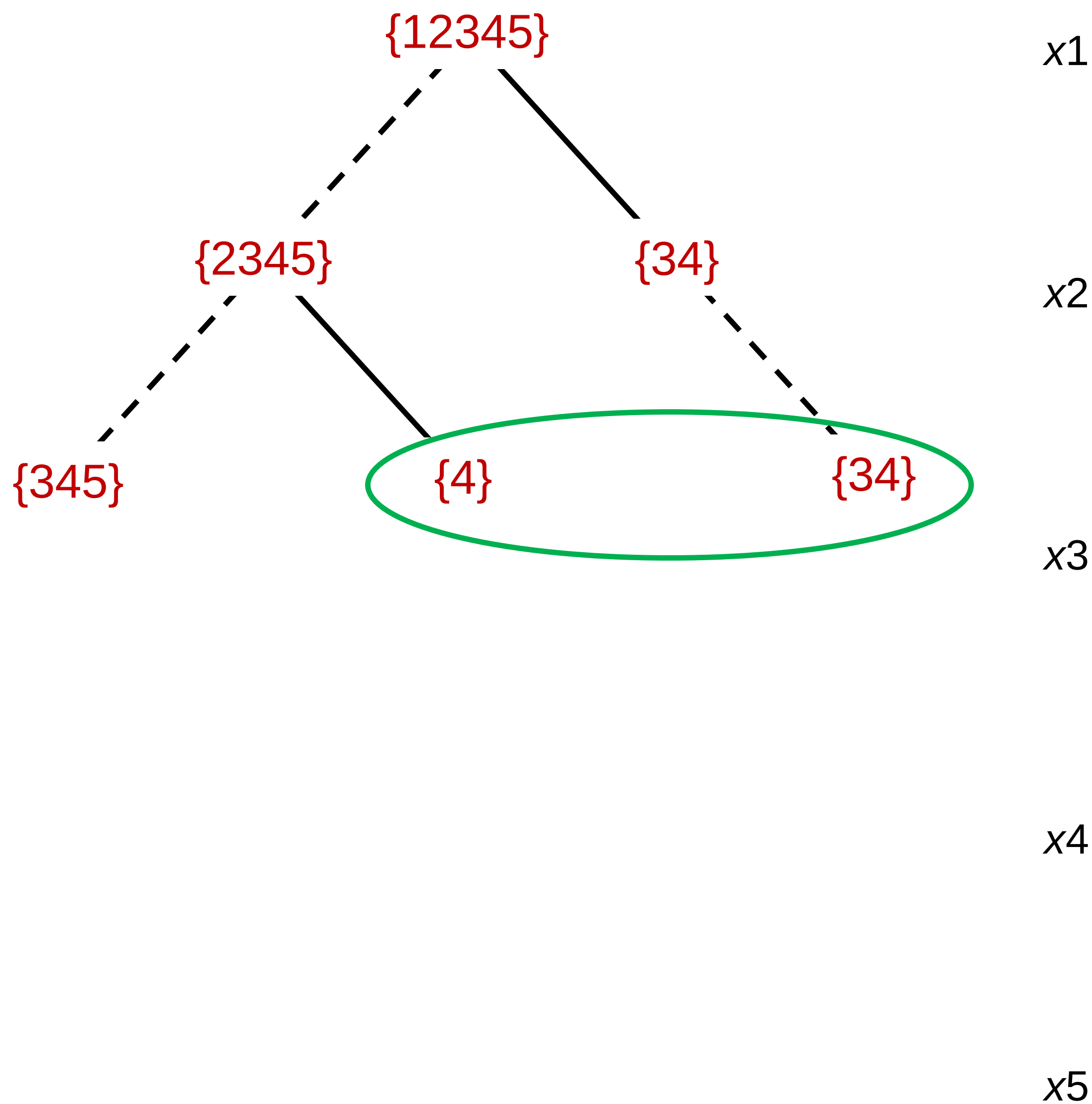
$x5$

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

Use generic merging heuristics.

In this case, take union of merged states

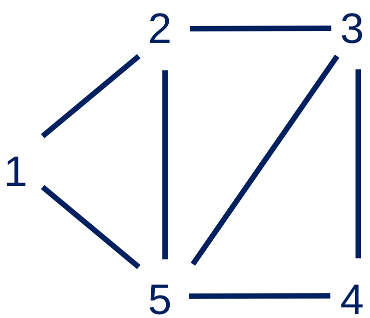


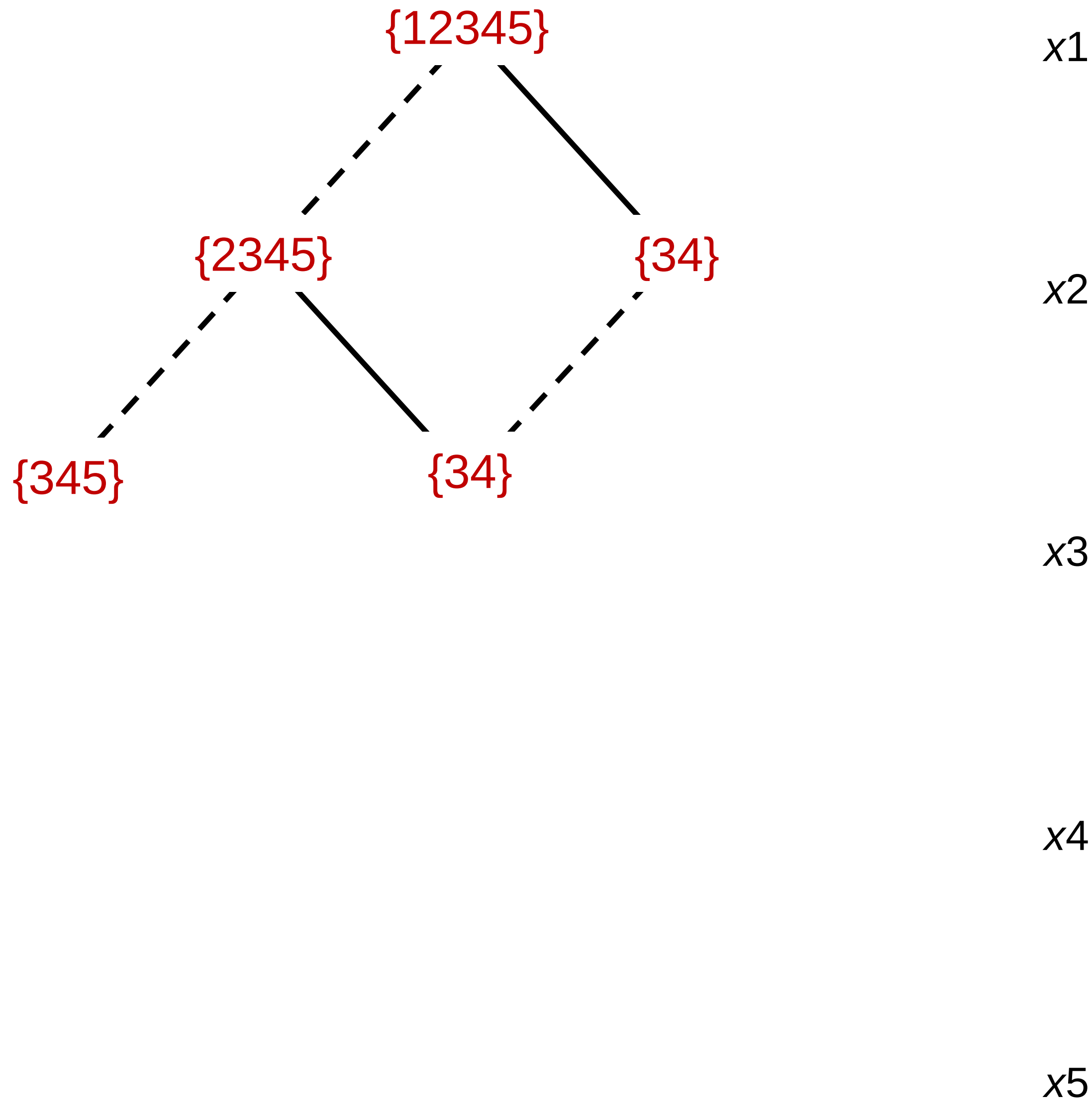


To build **relaxed** DD, merge some additional nodes as we go along.

Use generic merging heuristics.

In this case, take union of merged states

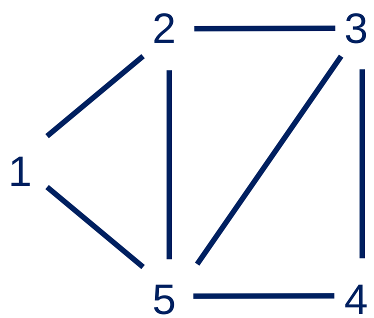


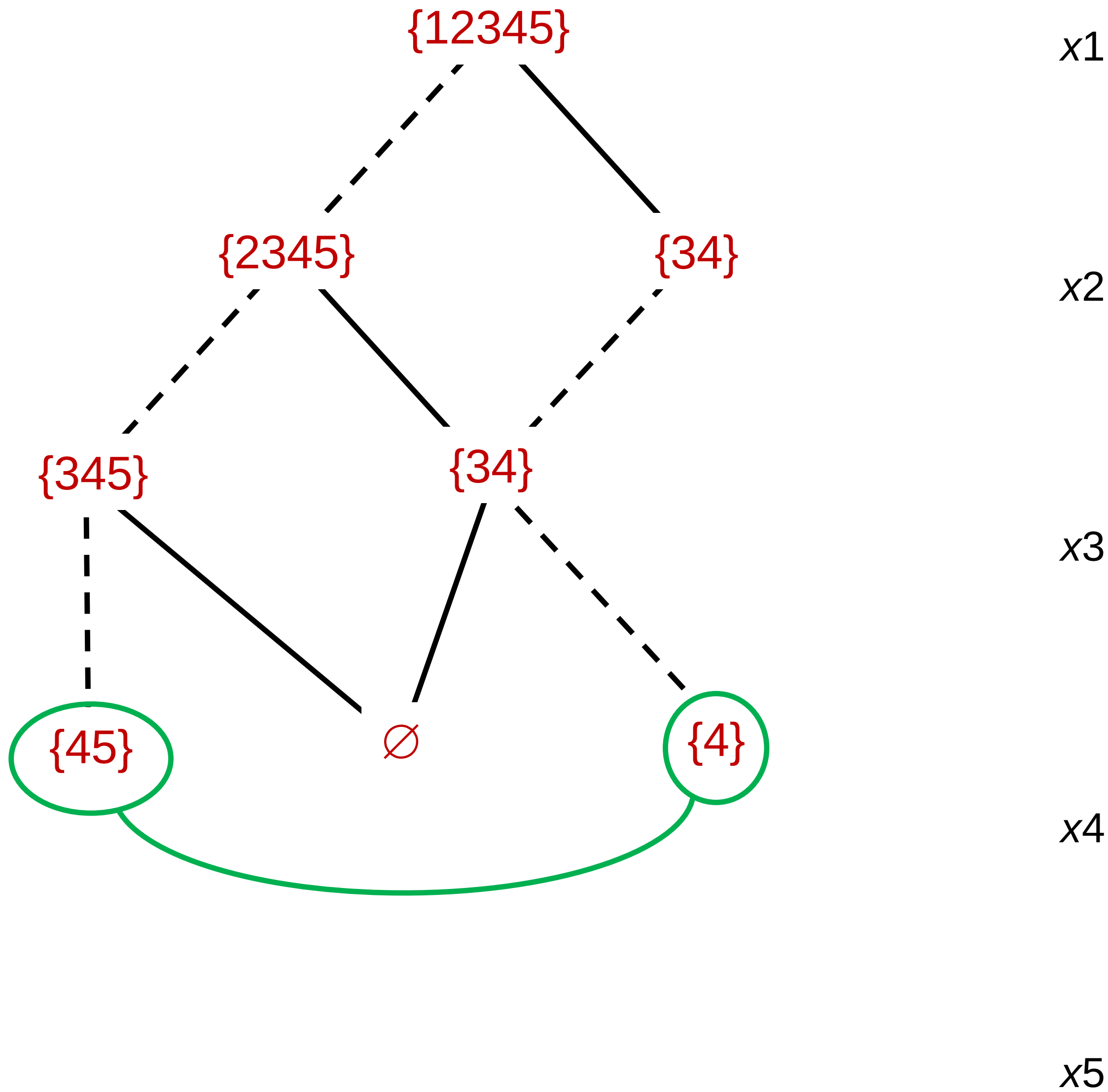


To build **relaxed** DD, merge some additional nodes as we go along.

Use generic merging heuristics.

In this case, take union of merged states

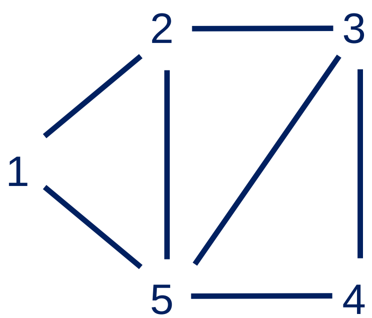


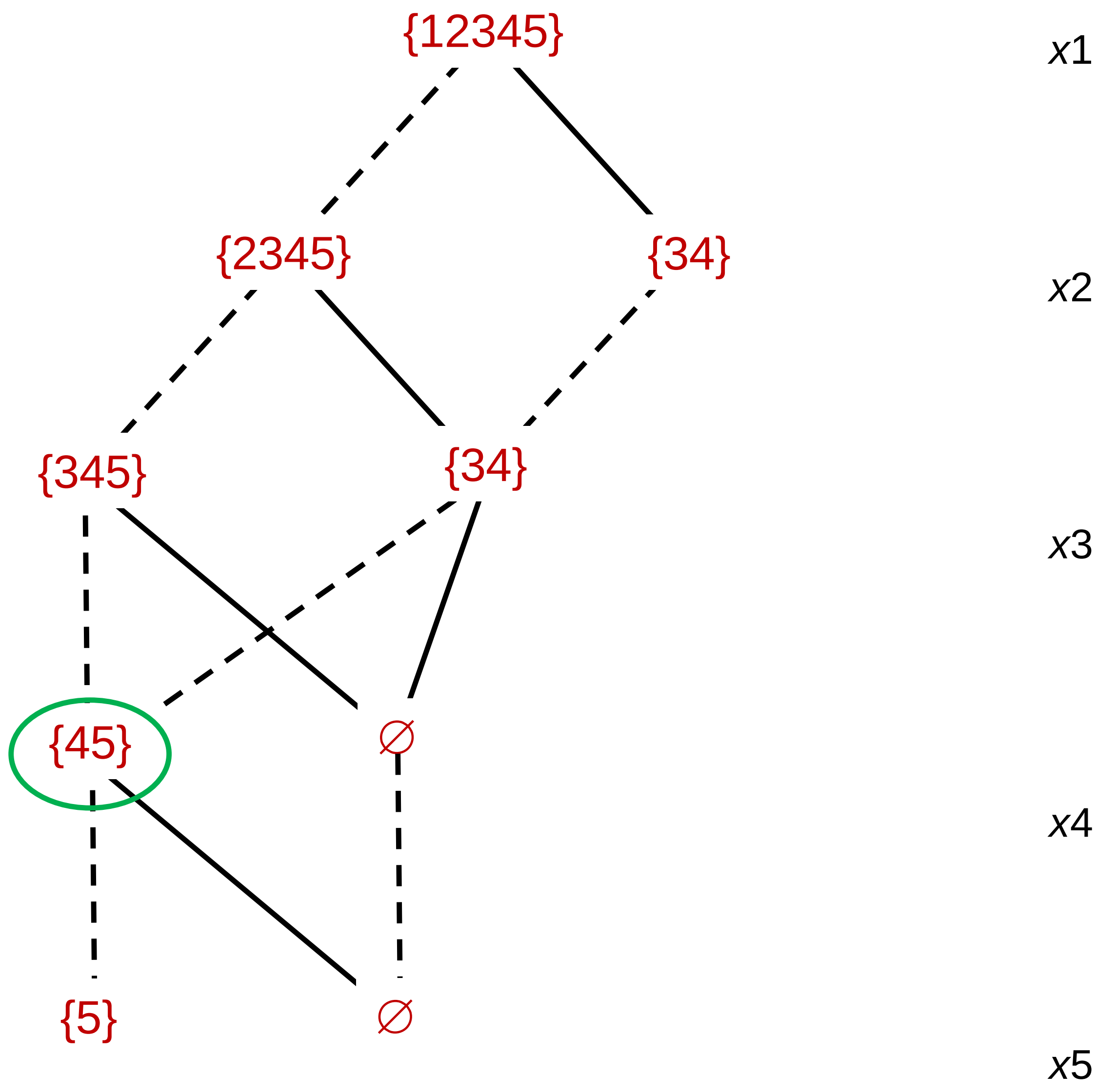


To build **relaxed** DD, merge some additional nodes as we go along.

Use generic merging heuristics.

In this case, take union of merged states

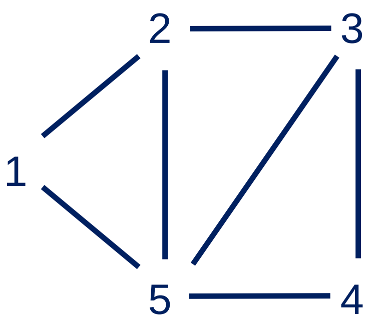




To build **relaxed** DD, merge some additional nodes as we go along.

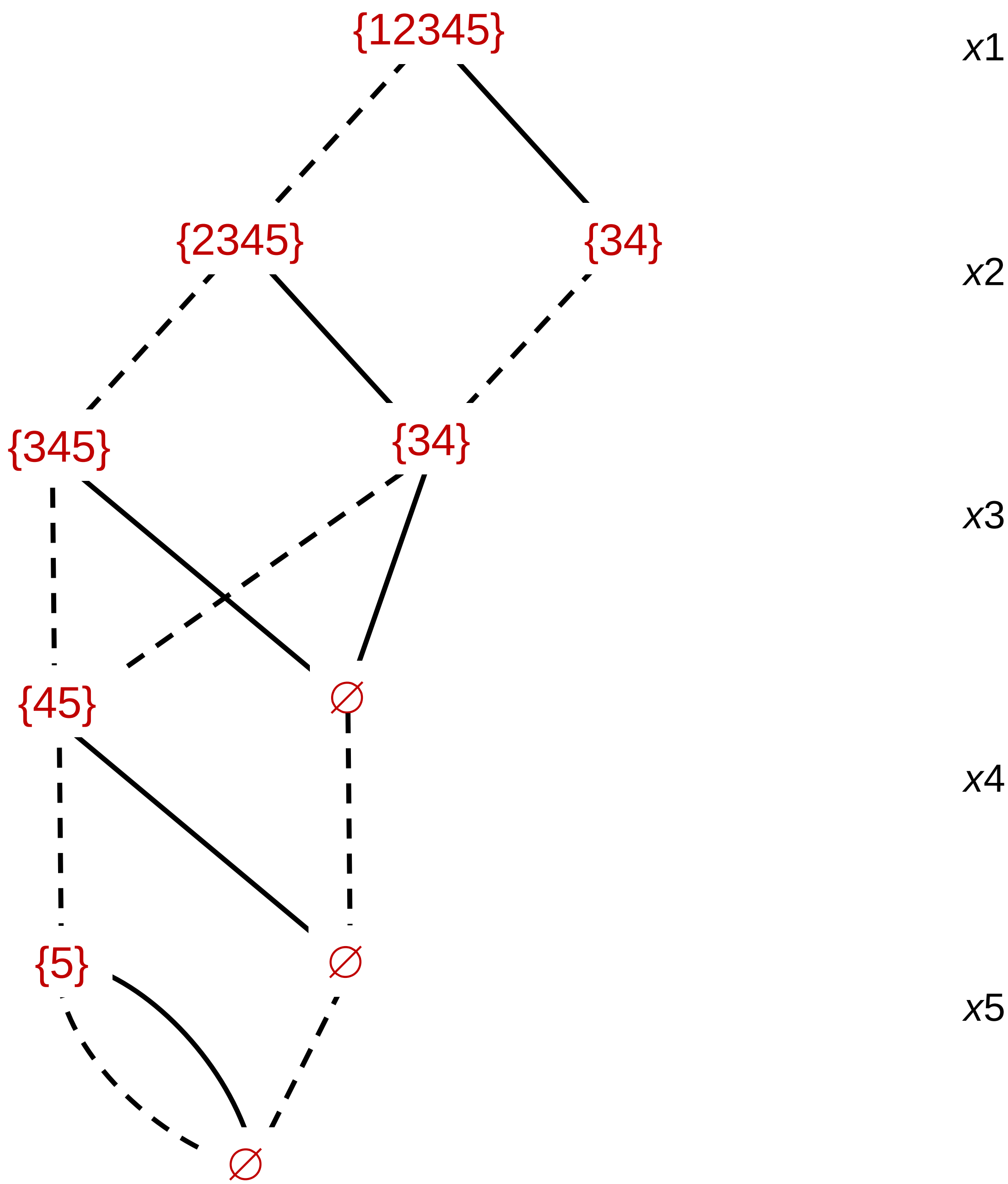
Use generic merging heuristics.

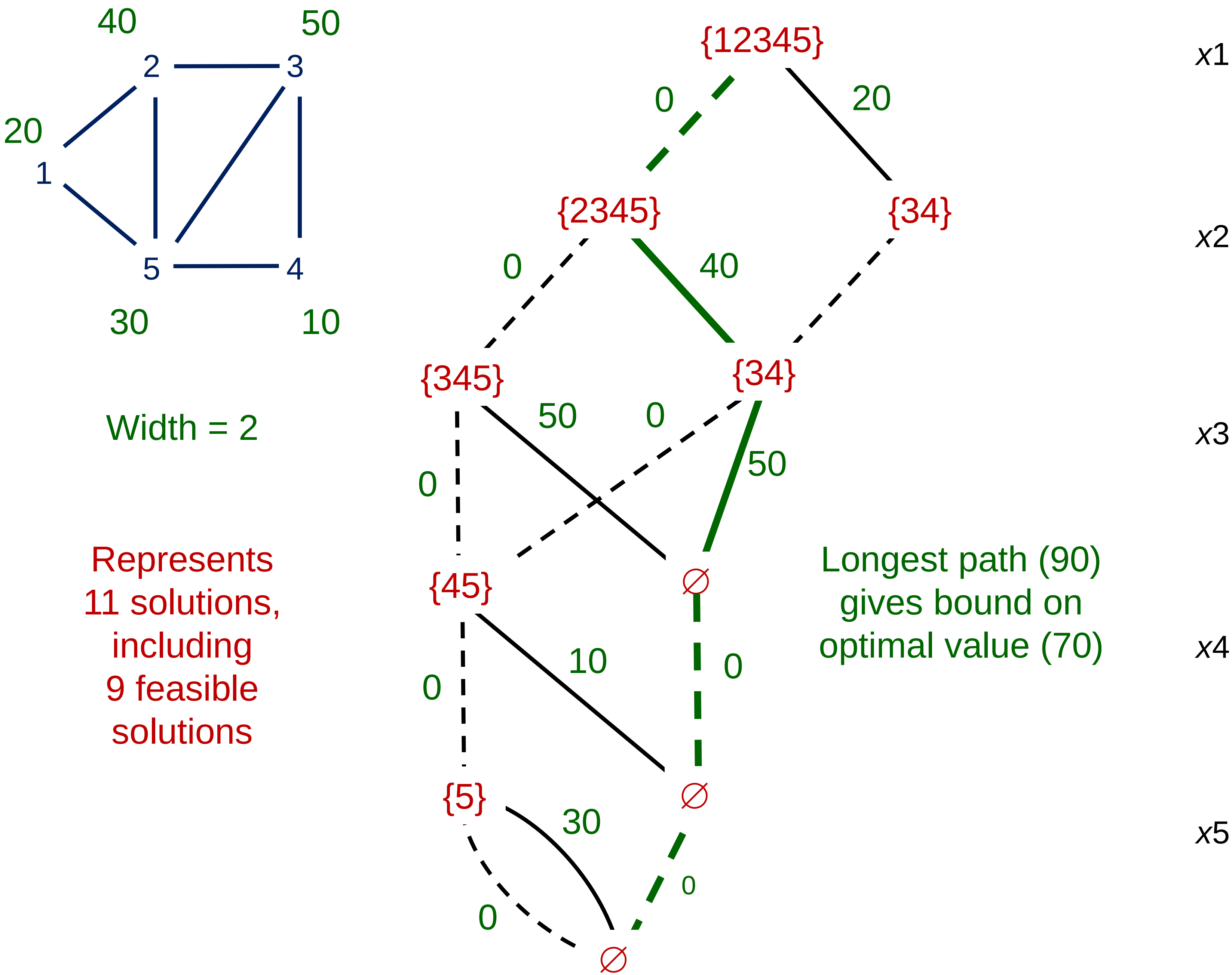
In this case, take union of merged states



Width = 2

Represents 11 solutions, including 9 feasible solutions



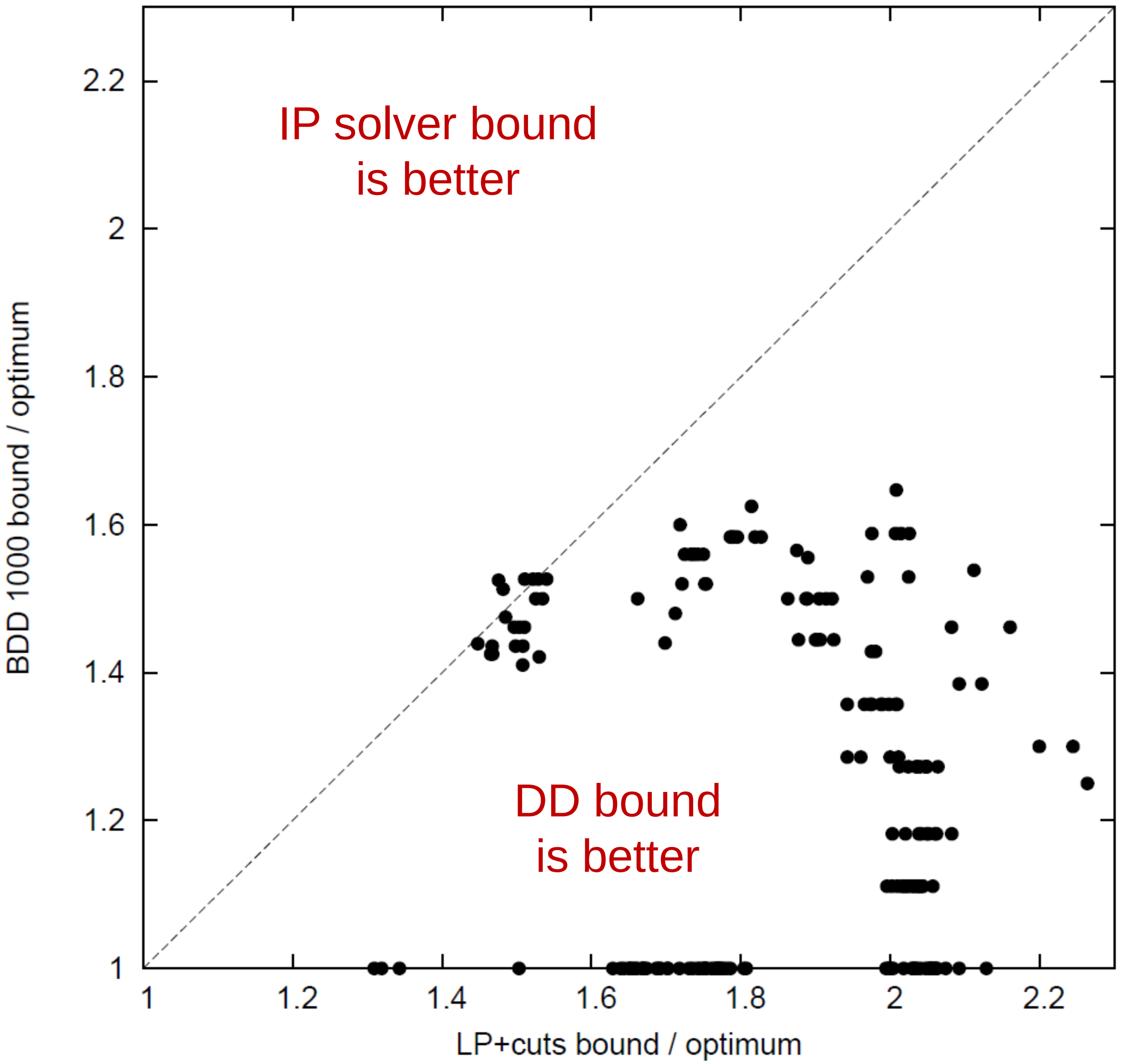


40
50
2
3
20
1
5
4
30
10
Width = 2
Represents
11 solutions,
including
9 feasible
solutions
{12345}
0
20
{2345}
{34}
0
40
{345}
{34}
50
0
0
50
{45}
∅
10
0
0
{5}
∅
30
0
0
∅
Longest path (90)
gives bound on
optimal value (70)
x1
x2
x3
x4
x5

Bound from Relaxed DD

Bounds from **DDs** vs. state-of-the-art **integer programming solver** for max stable set problem

- IP solver bound relies on 50 years of experience with cutting planes
- DD max width of 1000.
- DDs require about 5% the time of IP solver



Bergman, Ciré, van Hoeve, JH (2013)

Bound from Relaxed DD

DDs normally provide bounds within a solver, but they can also provide bounds for problems solved **heuristically**.

DDs plus **Lagrangian duality** can provide **very sharp** bounds.

For example, bounds mostly within **0.1%** of best known solutions of Biskup-Feldman machine scheduling instances with time windows (never solved to optimality). Sometimes, the bounds prove optimality.

JH (2019)

Decision Diagrams and Optimization

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

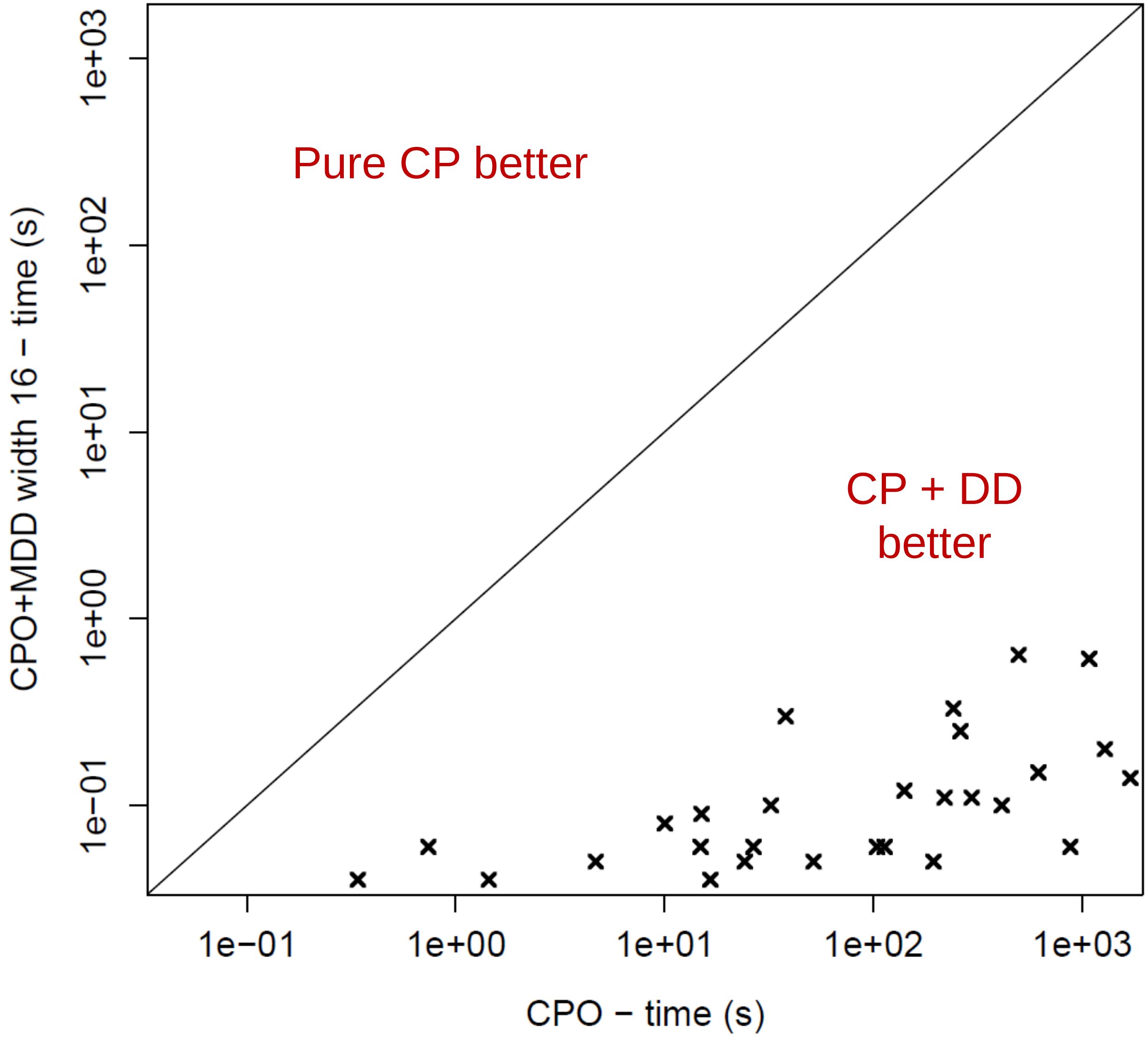
Example: Traveling salesman problem with time windows and other sequencing problems.

DDs allowed closure of several open problem instances.

Ciré, van Hoeve (2013)

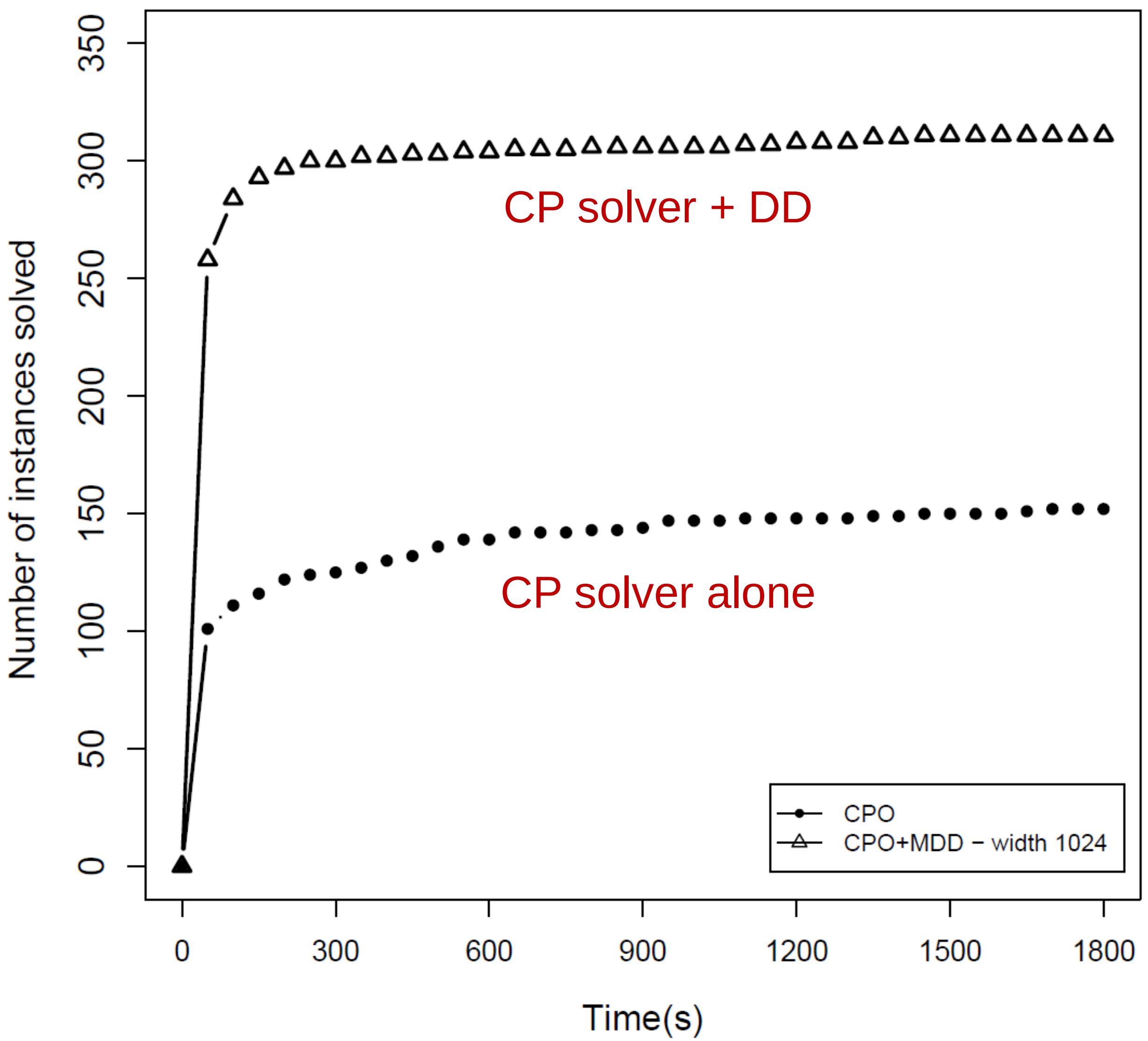
Computation time scatter plot, lex search

TSP with
time windows



Performance profile, depth-first search

TSP with
time windows



Decision Diagrams and Optimization

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

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

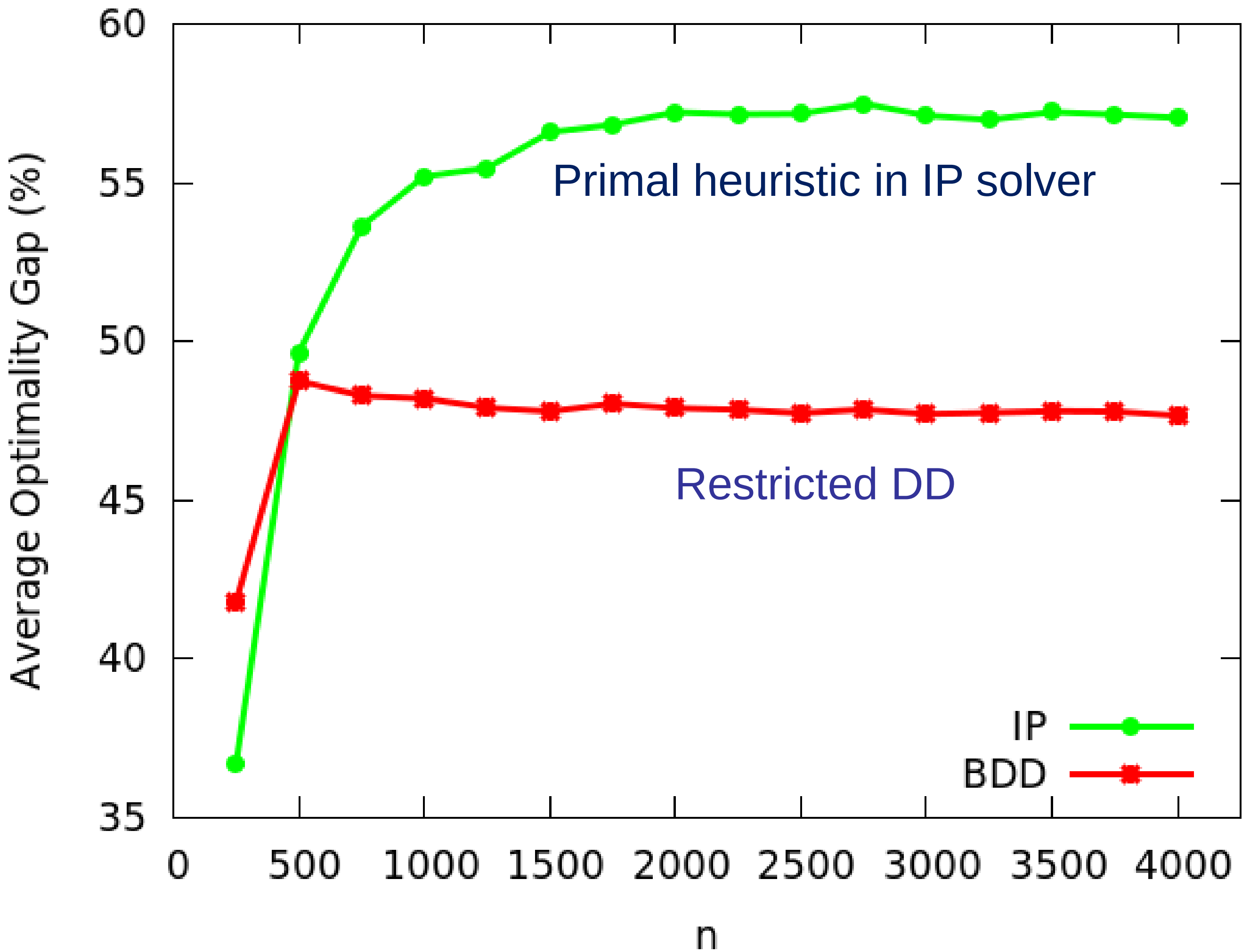
Primal heuristic = algorithm that finds solutions that are not optimal in general

Primal heuristics are an important factor in recent improvements of IP solvers.

Bergman, Ciré, van Hoeve, Yunes (2014)

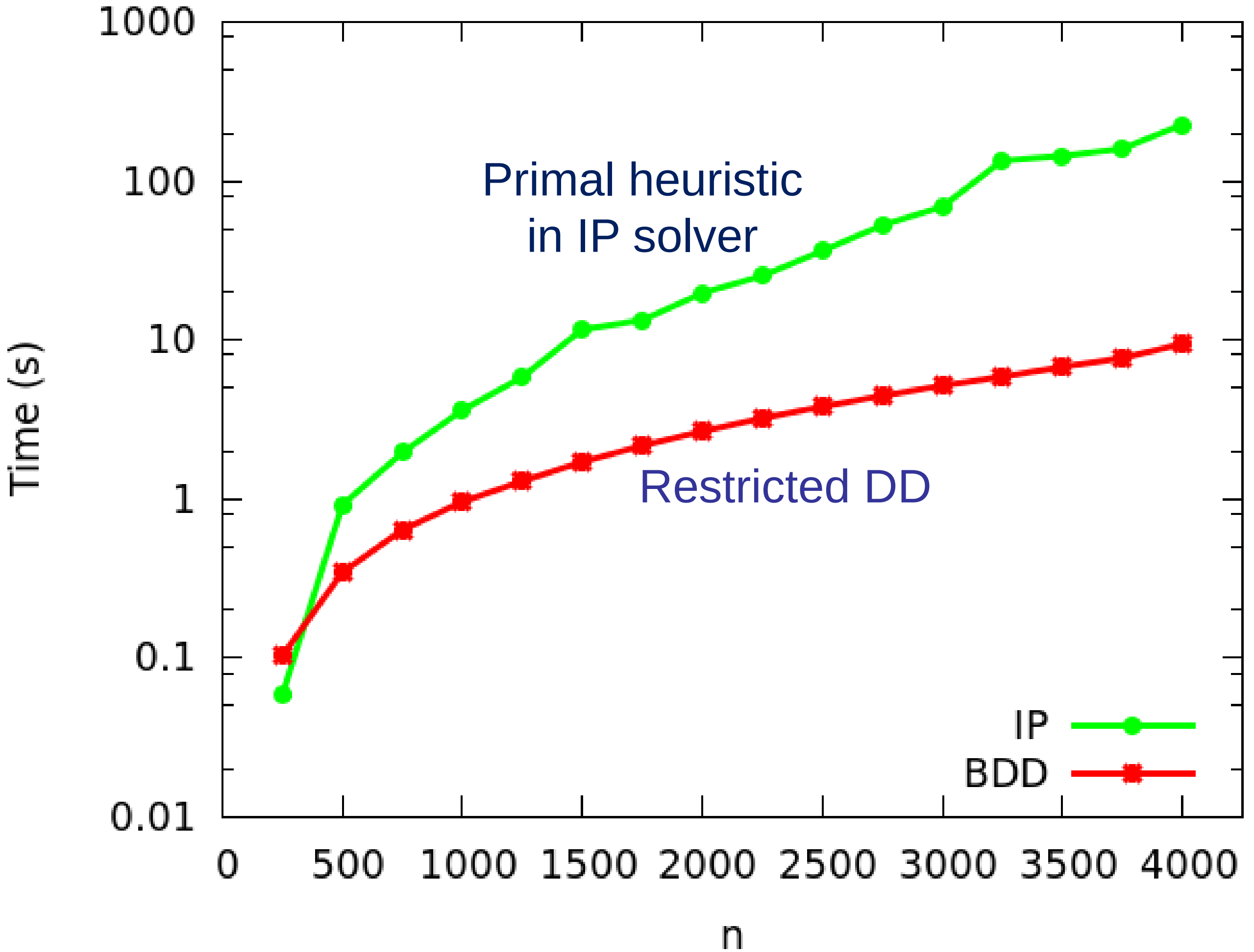
Optimality gap for **set covering**, *n* variables

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.

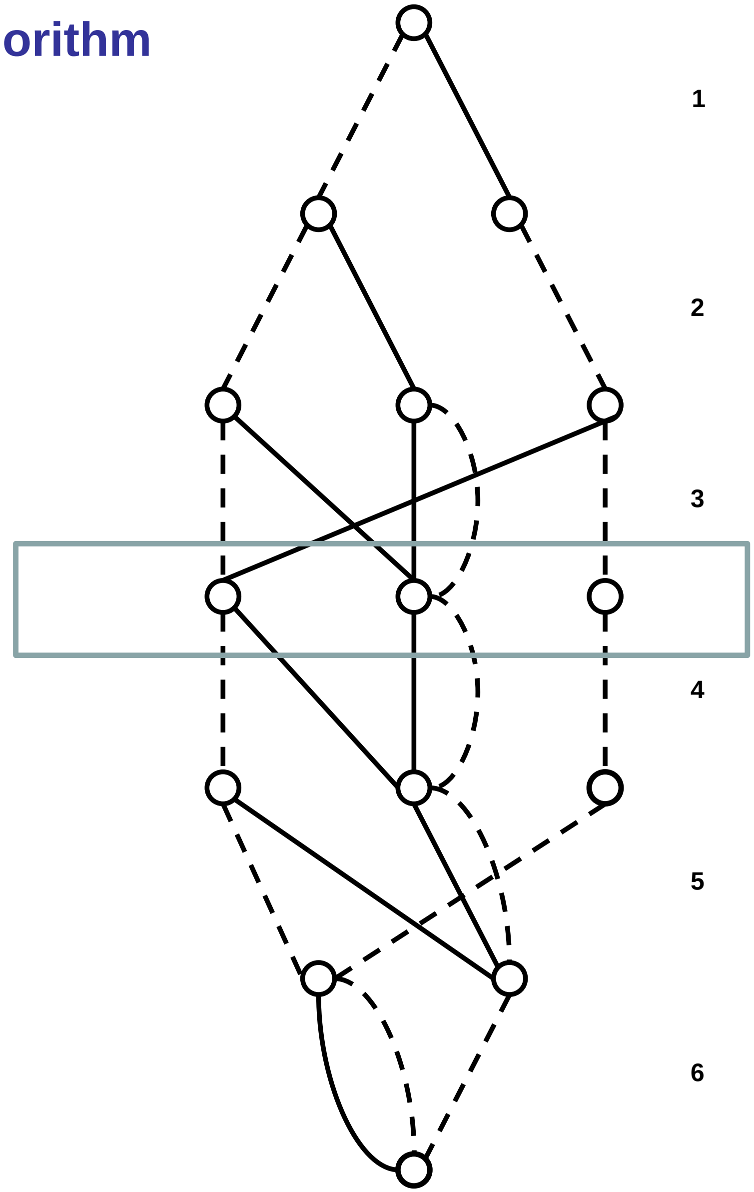
- **Bounds** from relaxed DDs.
- **Primal heuristic** from restricted DDs.
- **Recursive modeling**
- Novel **branching algorithm** – branch inside relaxed DD

Bergman, Ciré, van Hoeve, JH (2016)

Branching Algorithm

Branching in a relaxed decision diagram

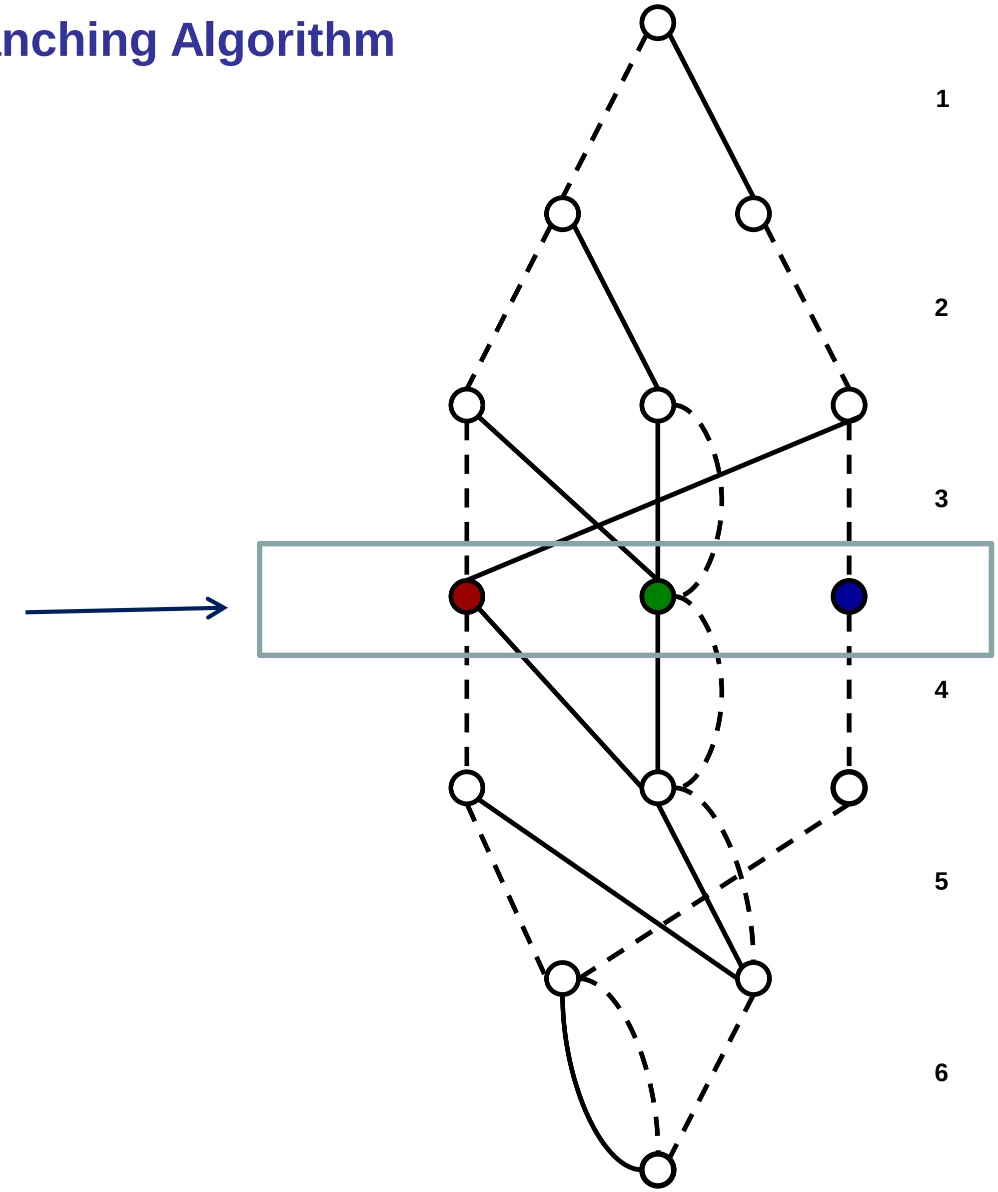
Diagram is exact
down to here



Branching Algorithm

Branching in a relaxed
decision diagram

Branch on nodes in
this layer



Branching Algorithm

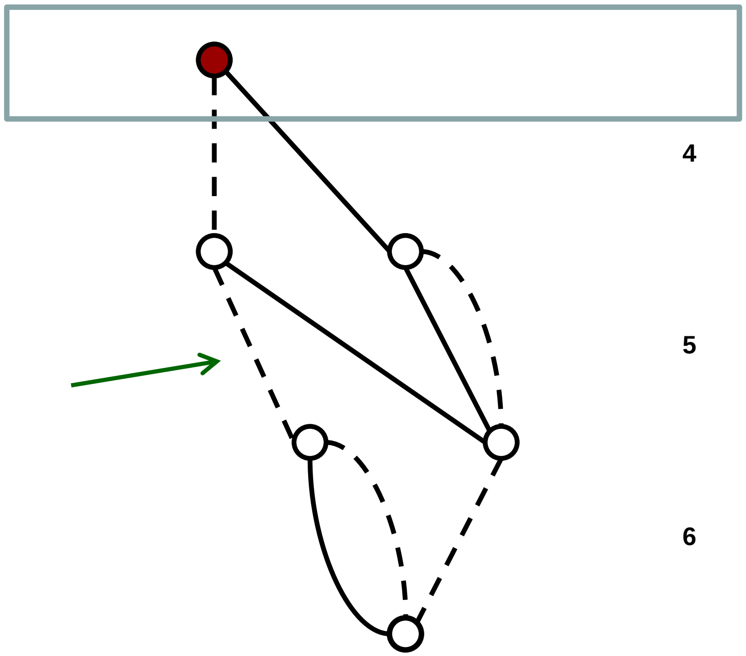
1

Branching in a relaxed
decision diagram

2

3

First branch



4

5

New relaxed decision diagram

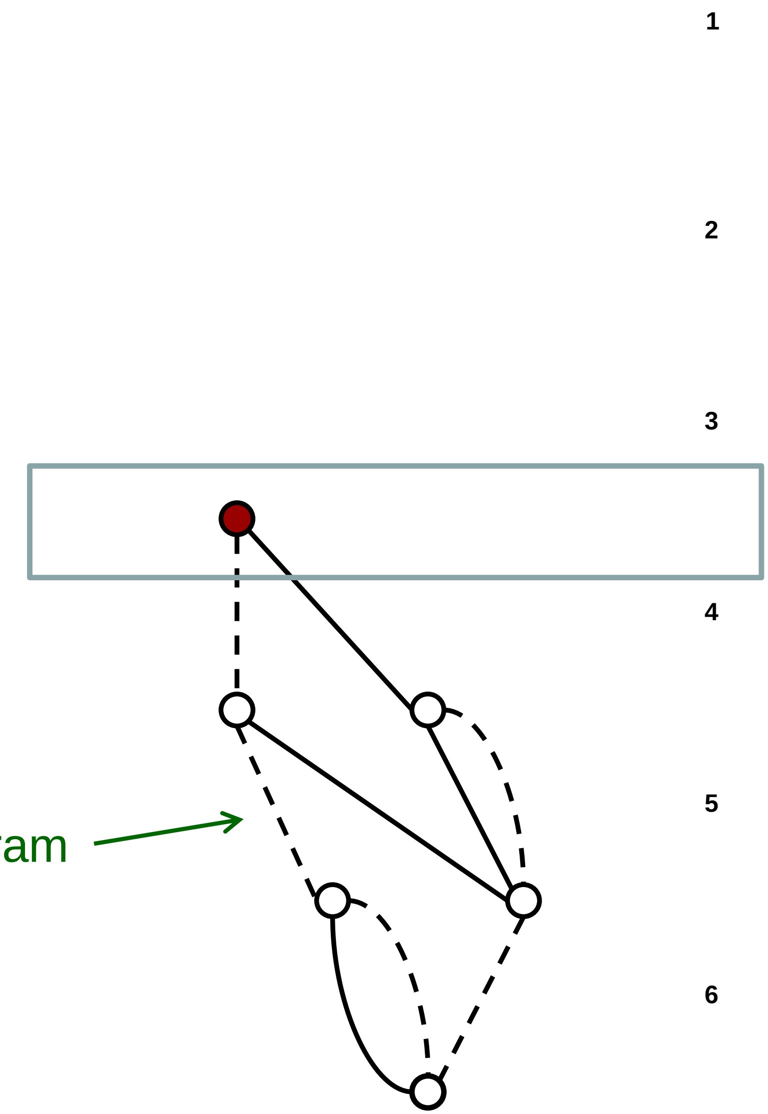
6

Branching Algorithm

Branching in a relaxed decision diagram

First branch

New relaxed decision diagram

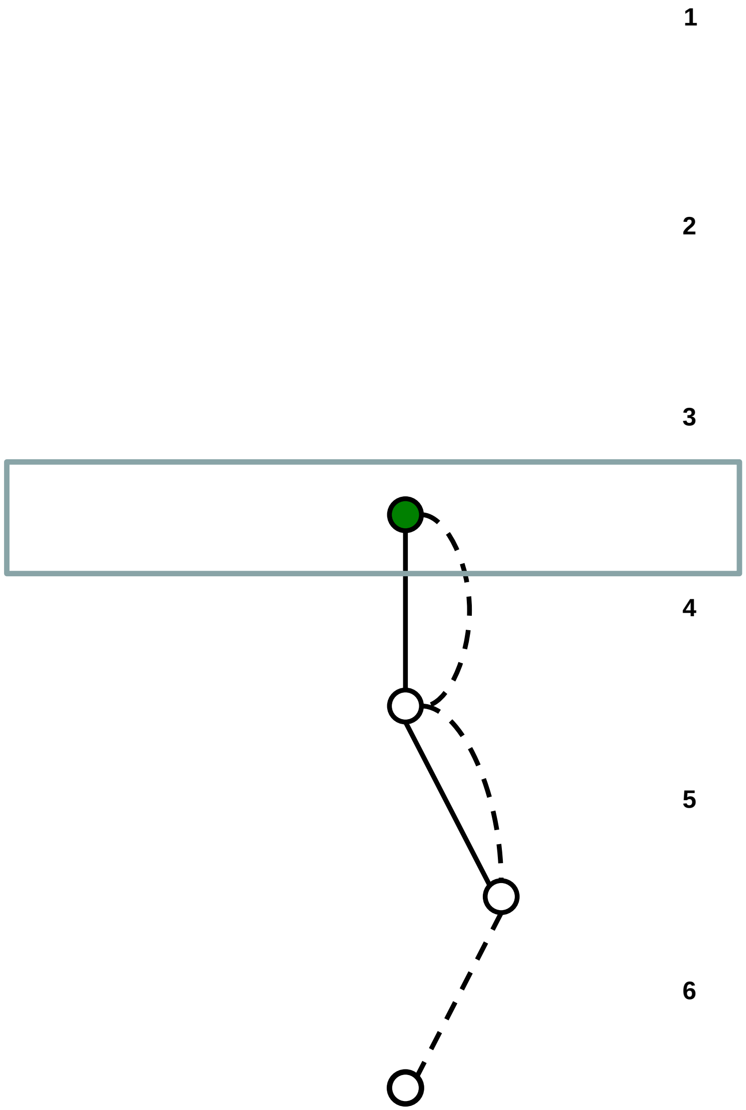


Pruning based on **cost bounds** obtained from solving **shortest path problem** in relaxed DDs

Branching Algorithm

Branching in a relaxed
decision diagram

Second branch



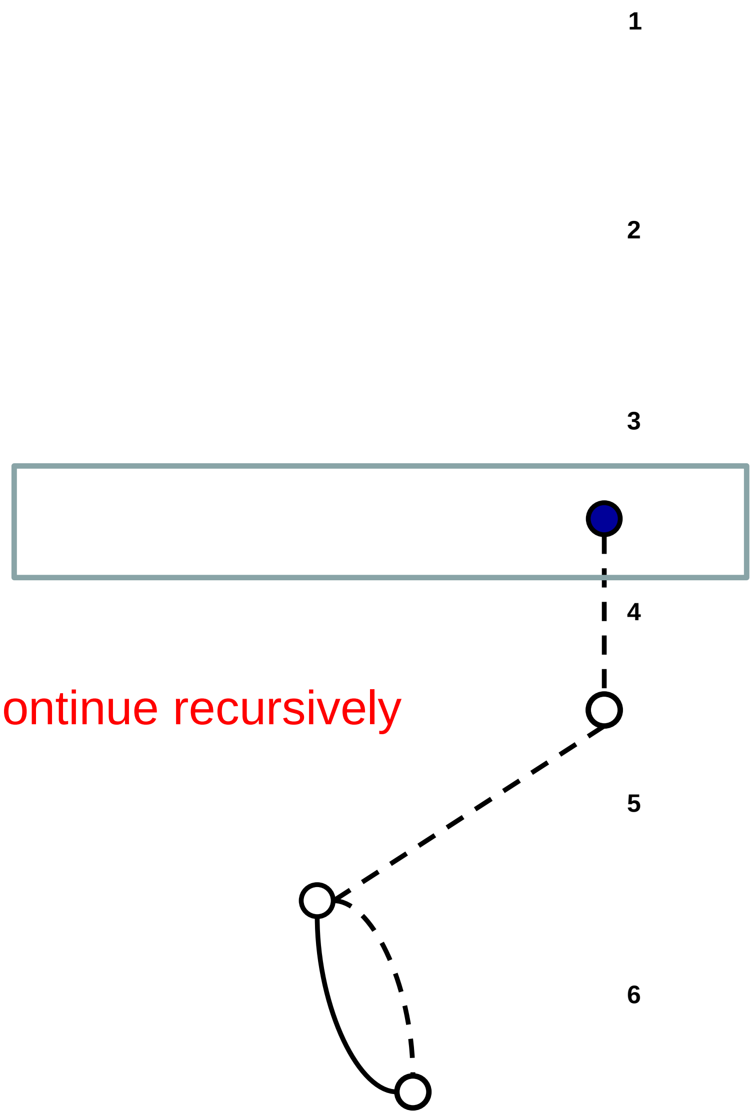
Pruning based on **cost bounds** obtained
from solving **shortest path problem**
in relaxed DDs

Branching Algorithm

Branching in a relaxed decision diagram

Third branch

Continue recursively



Pruning based on **cost bounds** obtained from solving **shortest path problem** in relaxed DDs

Branching Algorithm

Finding **shortest (longest) path** in a relaxed DD is the **fundamental calculation**,

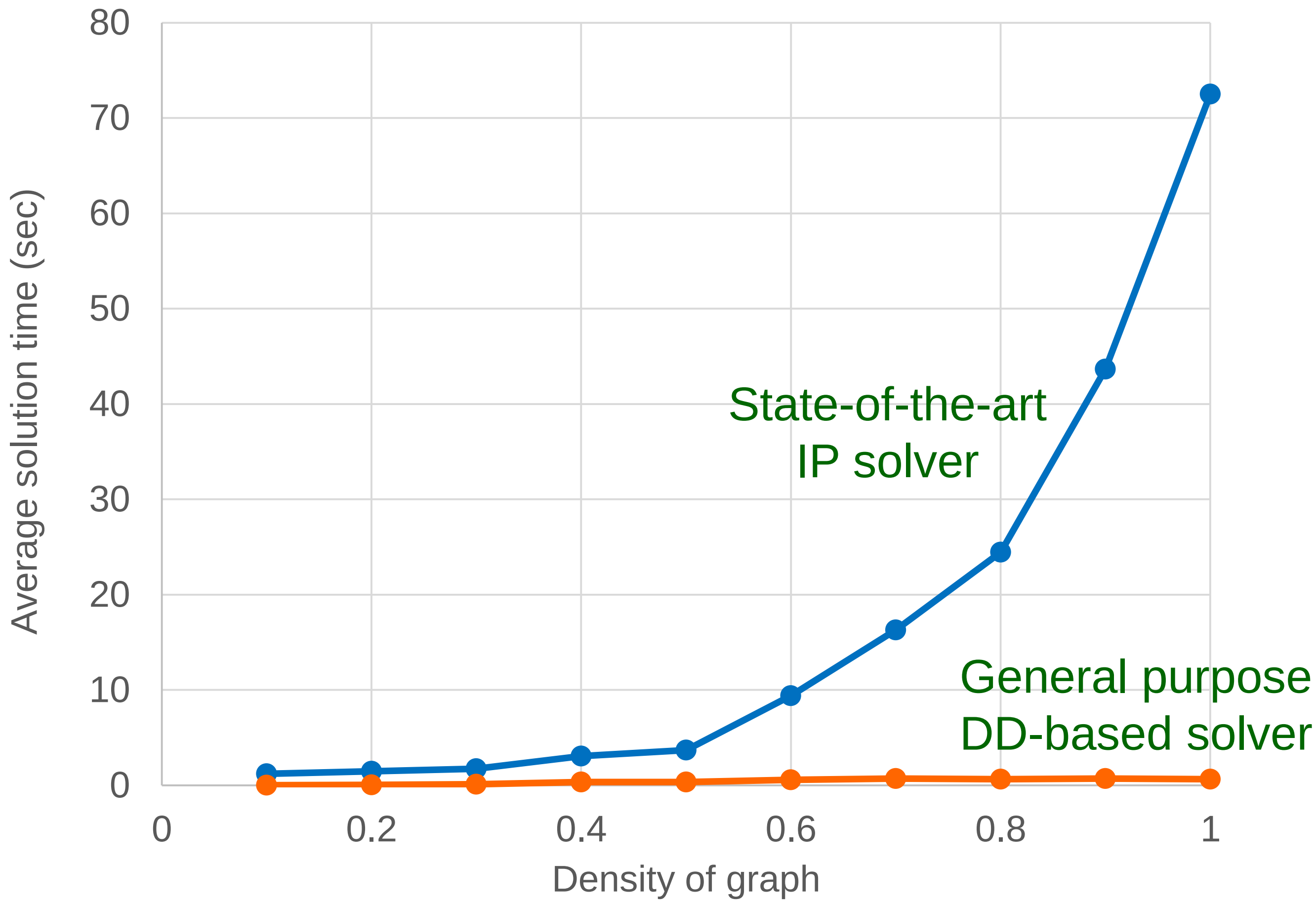
…much as solving an LP is the fundamental calculation in traditional MIP solvers.

Computational performance

Max cut
on a graph

Avg. solution time vs graph density

30 vertices

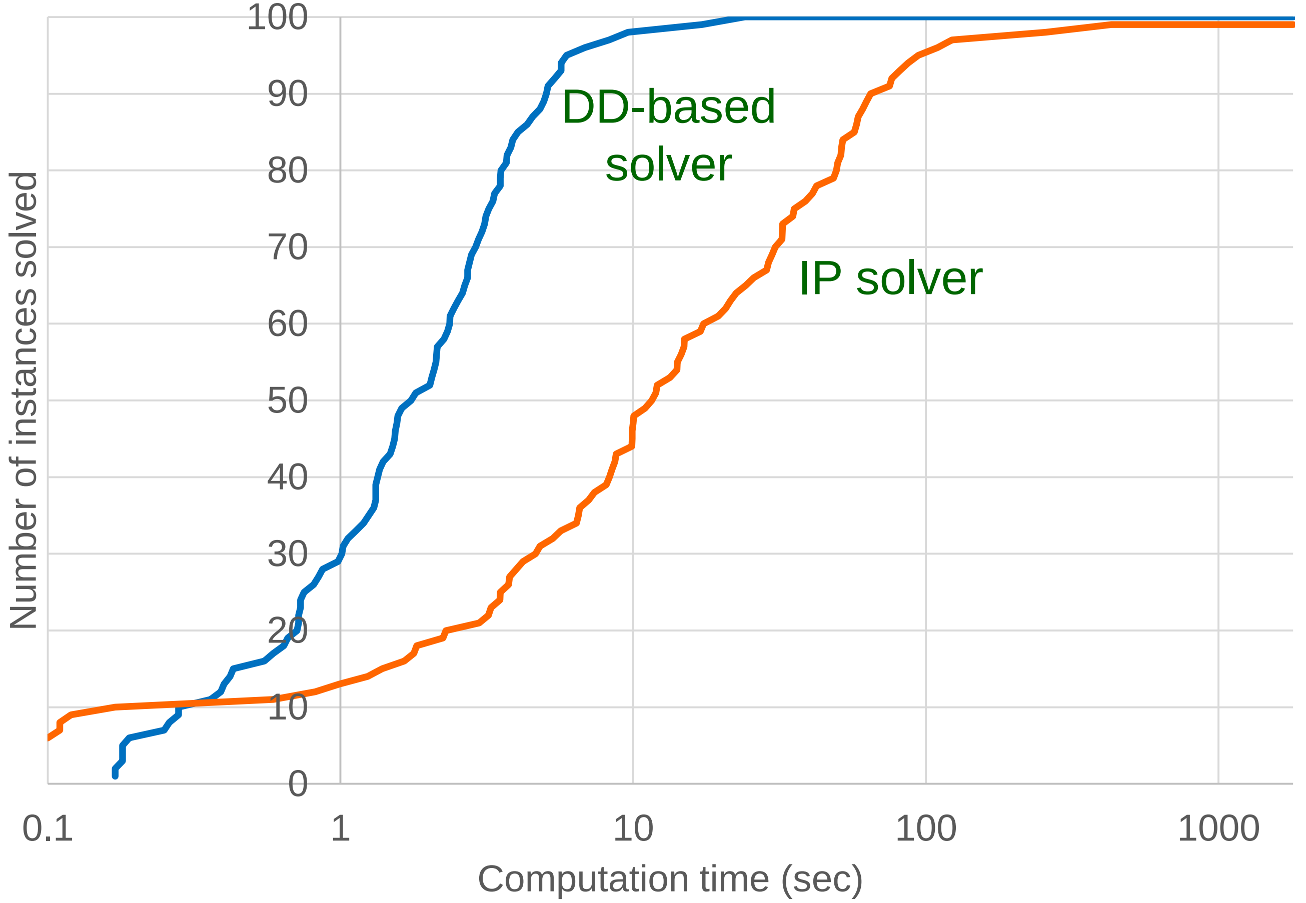


Computational performance

Max 2-SAT

Performance profile

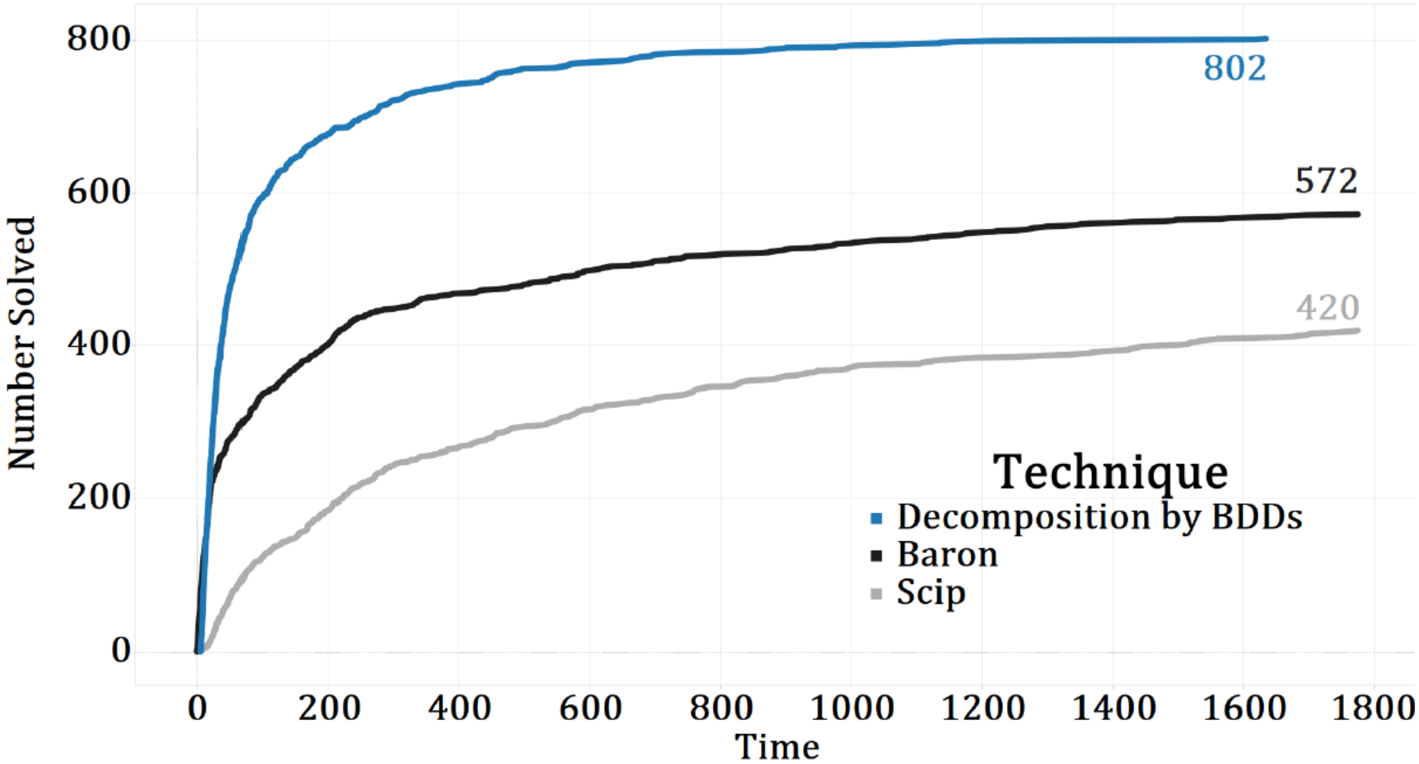
30 variables



Nonlinear Optimization

Decompose problem into multiple DDs

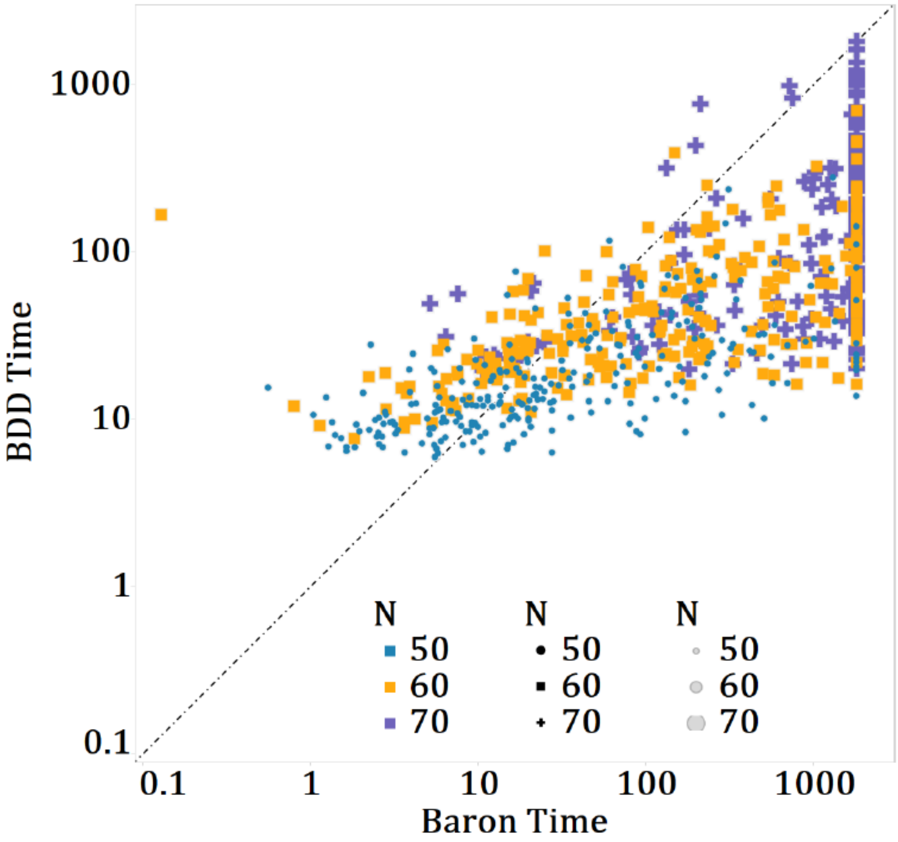
Portfolio design



Bergman and Ciré (2018)

Nonlinear Optimization

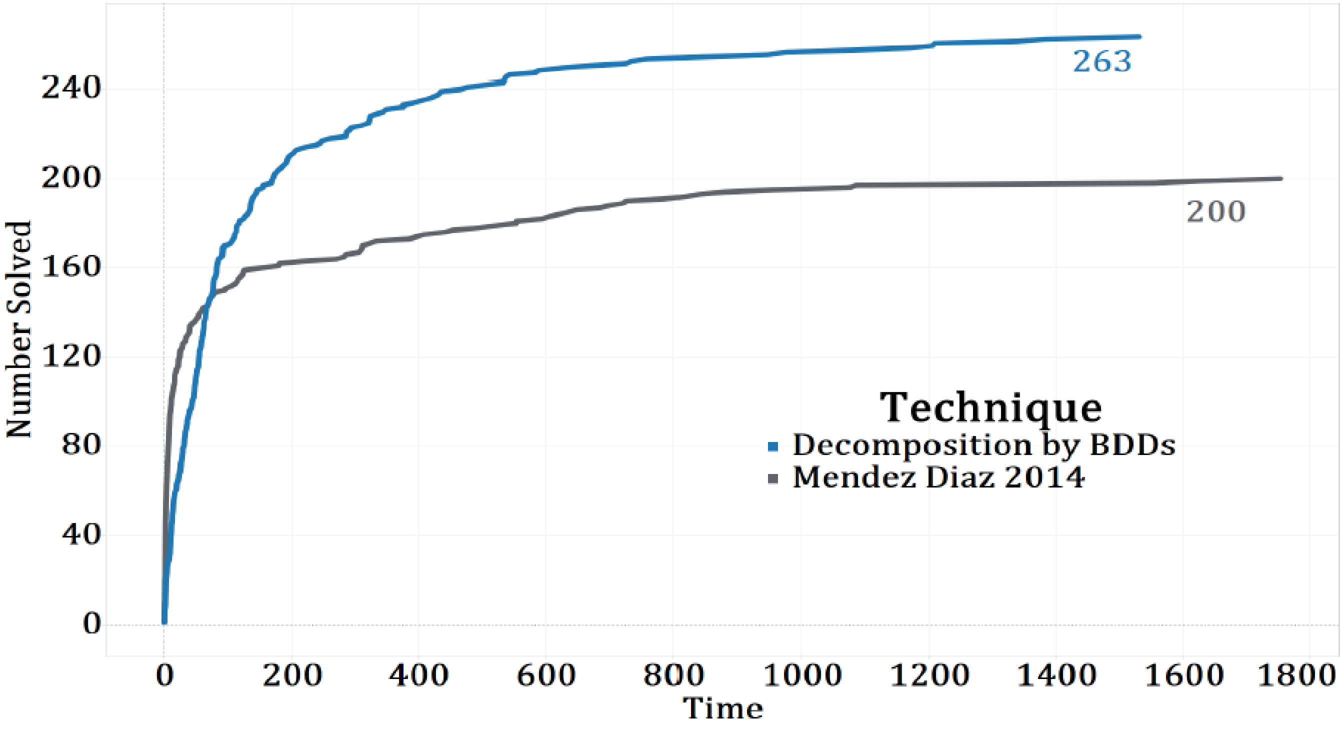
Portfolio design



Bergman and Ciré (2018)

Nonlinear Optimization

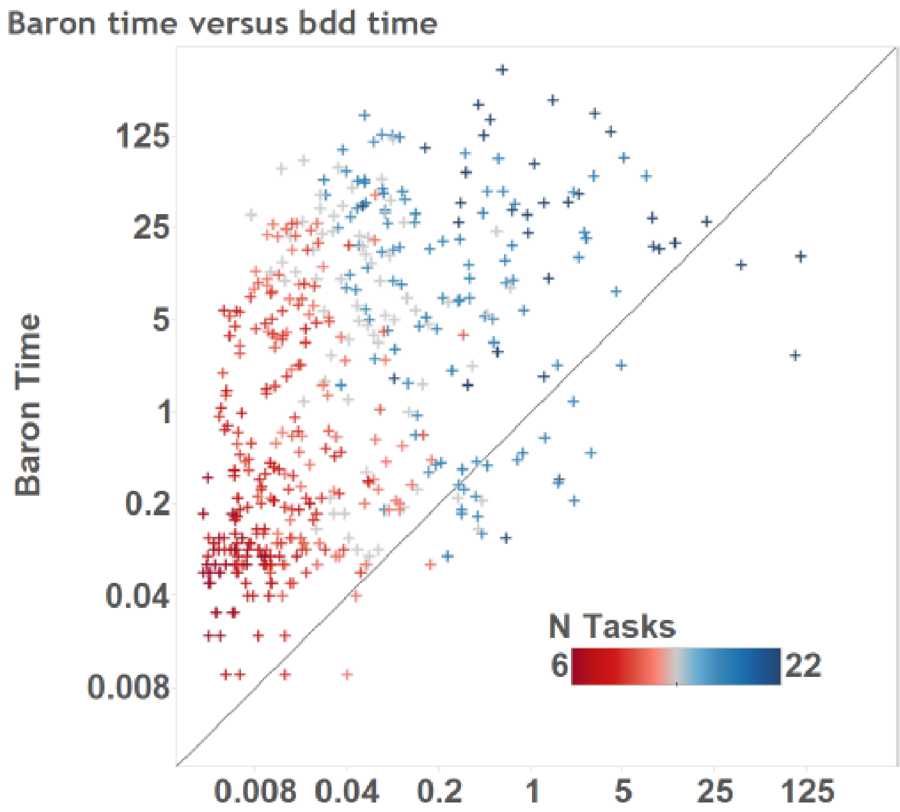
Product assortment



Bergman and Ciré (2018)

Nonlinear Optimization

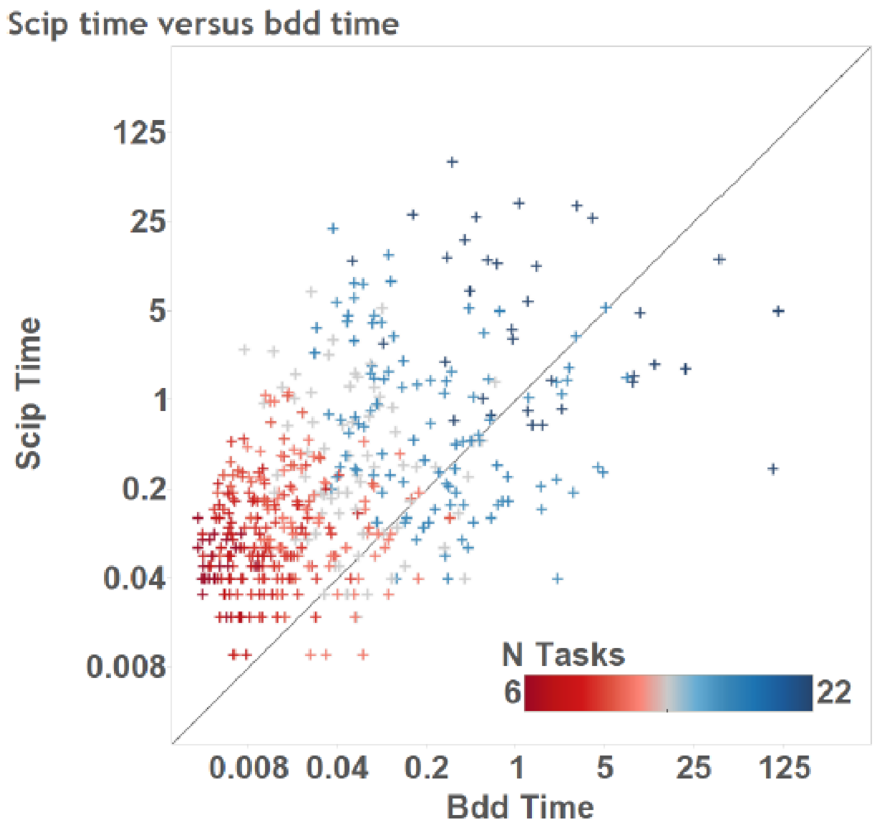
Workflow employee assignment



Bergman and Ciré (2018)

Nonlinear Optimization

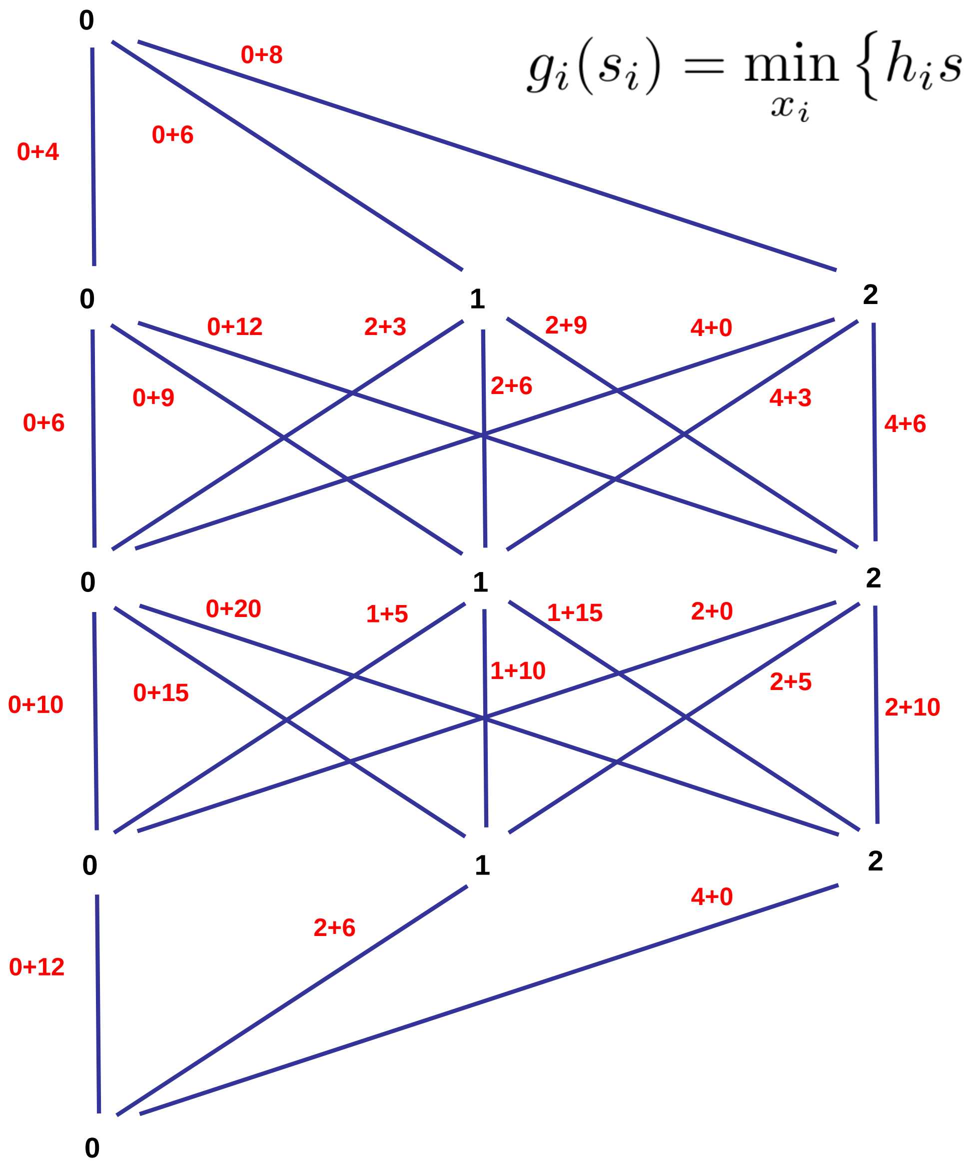
Workflow employee assignment



Bergman and Ciré (2018)

Simplification of Dynamic Programming Models

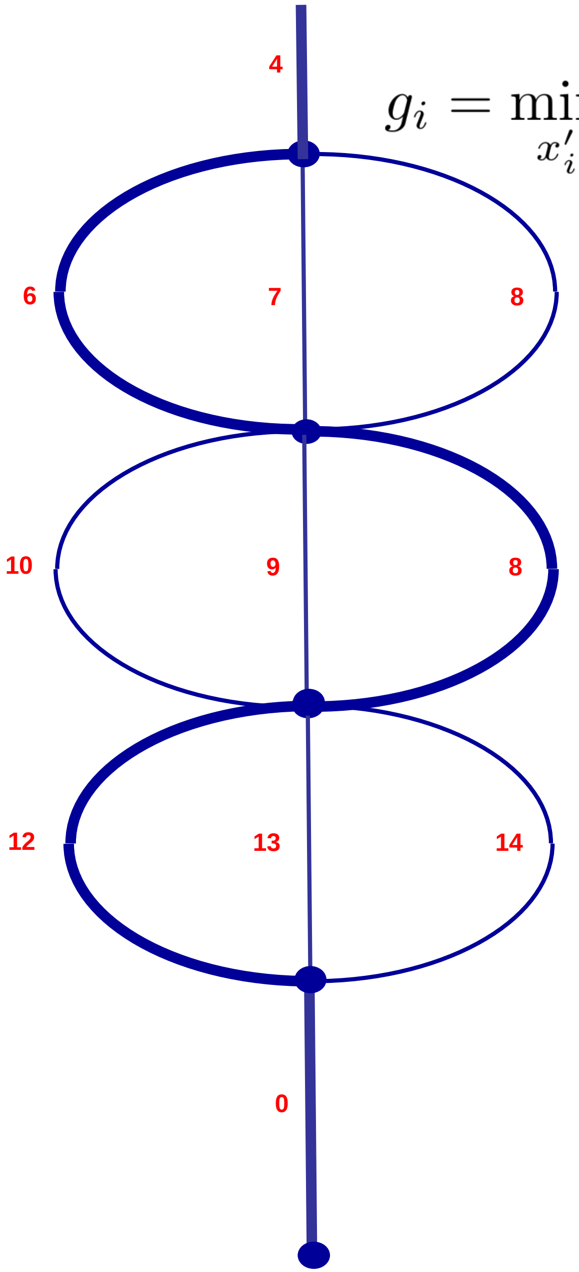
$$g_i(s_i) = \min_{x_i} \left\{ h_i s_i + c_i x_i + g_{i+1}(s_i + x_i - d_i) \right\}$$



State transition graph for
inventory problem

JH (2013)

Simplification of Dynamic Programming 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**.

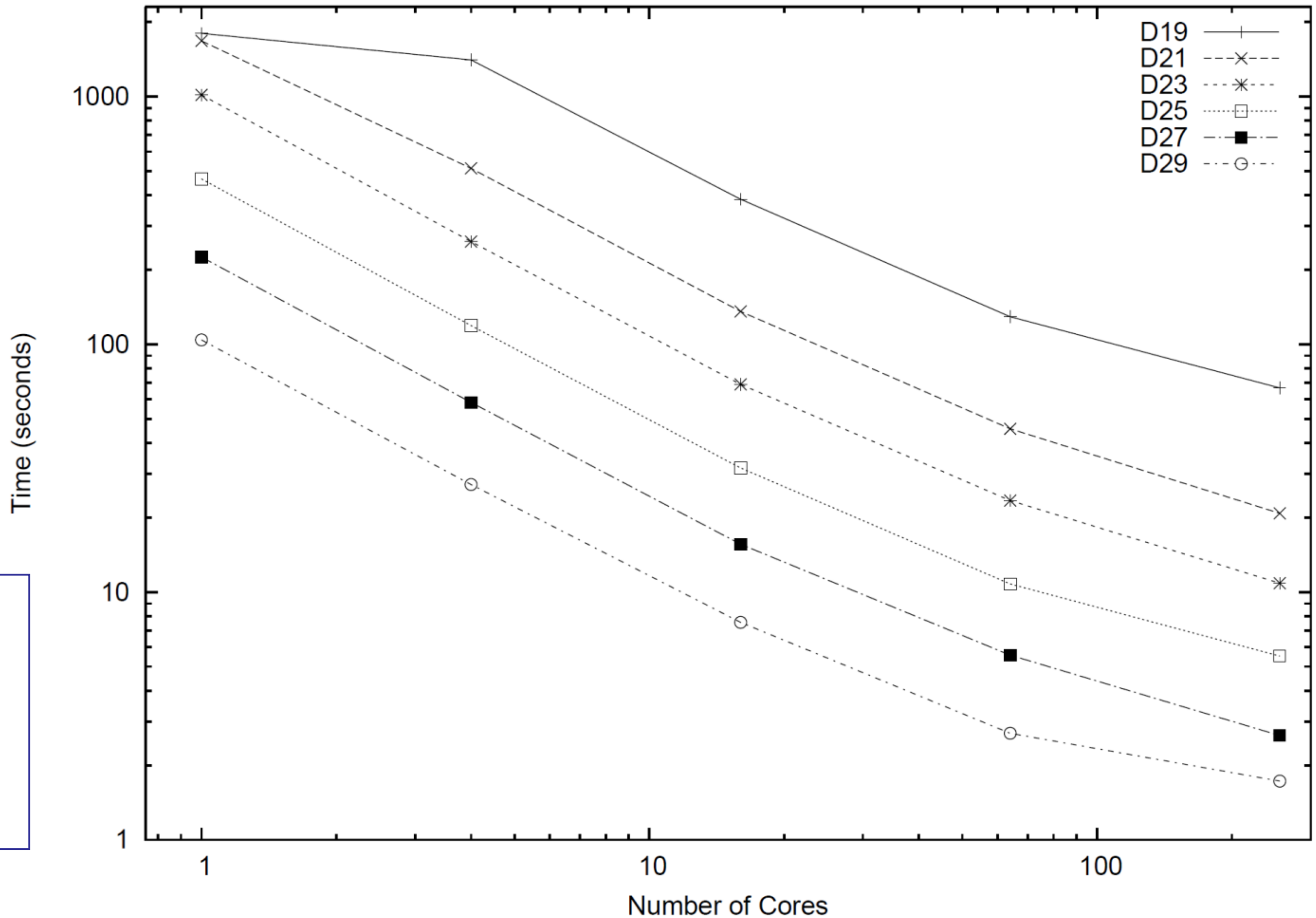
Radical simplification of the problem – only 1 state per stage.

JH (2013)

Parallelization

Parallel computation with DDs achieves almost **100% linear speedup**.
(Mixed integer programming achieves about 30% speedup)

Stable set problem



Bergman, Cire, Sabharwal, Samulowitz, Saraswat, van Hoeve (2014)

Killer App?

Can we turn over the shortest path problem to a **quantum computer**?

- Possibly radical **speedup**
- Possible application of quantum computing to a **wide variety** of constrained combinatorial optimization problems.
- Possible massive **parallization** (solve many shortest path problmems at once).

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 **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.

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

Cutting planes are an essential component of IP solvers.

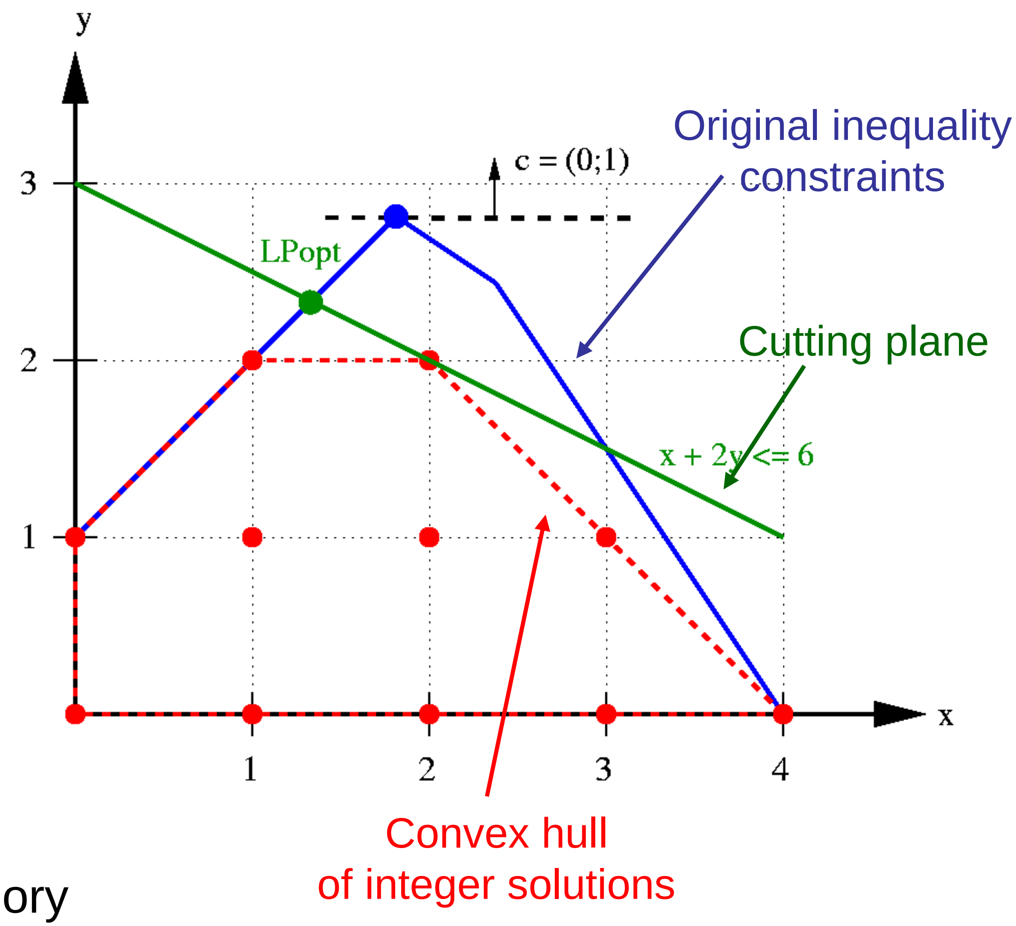
Studied for over 50 years.

Approximate convex hull of integer solutions…

…so that LP relaxation gives tighter dual bound.

Gomory 1960, etc.

Ralph Gomory



Resolution and Cutting Planes

A resolvent is a **cutting plane** (rank 1 Chvátal cut)

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

V. Chvátal

Proof of Chvátal's **cutting plane theorem** (fundamental result in cutting plane theory) implicitly relies on resolution!

Chvátal 1973

Resolution and Cutting Planes

A resolvent is a **cutting plane** (rank 1 Chvátal cut)

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

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

An optimization problem can be viewed from two perspectives:

Primal problem:
Search

Search for **minimum cost** solution that satisfies constraints.

Dual problem:
Logical inference

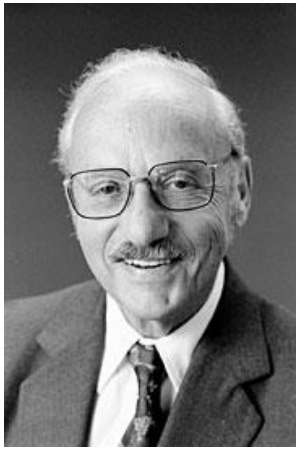
Deduce from constraints the tightest possible lower **bound** on cost.

When the dual inference method is complete,
the primal and dual have the same optimal value
(**strong duality**)

Logic and Duality

Duality first described for **linear programming** & **game theory**

George Dantzig



Primal problem	Dual problem
LP minimization	Also an LP, solution provides multipliers for optimality proof.

John von Neumann

Primal problem	Dual problem
Player 1 in 2-person noncooperative game	Player 2 in game

Logic and Duality

Optimization problems are typically solved by **primal-dual** algorithms.
Search for primal and dual solution simultaneously.

All optimization **duals** are **logical inference problems**.

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

Logic and Duality

Primal problem:
Optimization

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

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

Inference dual

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

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

$$\min \; cx \qquad = \qquad \max \; v$$

$$Ax \geq b \qquad\qquad Ax \geq b \overset{x\geq 0}{\Rightarrow} cx \geq v$$

$$x \geq 0$$

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,

$$Ax \geq b \overset{x \geq 0}{\Rightarrow} cx \geq v \quad \text{iff} \quad \begin{array}{c} \lambda Ax \geq \lambda b \ \boxed{\text{implies}}\ cx \geq v \\ \text{for some } \lambda \geq 0 \end{array}$$

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

Logic and Duality

LP Duality

$$\begin{array}{lll} \min\ cx & = \quad \max\ v & = \quad \max\ \lambda b \\ Ax \ge b & \quad Ax \ge b \overset{x\ge 0}{\Rightarrow} cx \ge v & \quad \lambda A \le c \\ x \ge 0 & & \quad \lambda \ge 0 \end{array}$$

This is the **classical LP dual**

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

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

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

$\max \; v$

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

Surrogate

Let us say that

$g(x) \ge 0 \overset{x \in S}{\Longrightarrow} f(x) \ge v$ 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

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

Let us say that

Surrogate

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

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

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

Dual

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

Surrogate

Let us say that

$g(x) \ge 0 \overset{x \in S}{\Longrightarrow} f(x) \ge v$ 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 $x \in S$

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

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

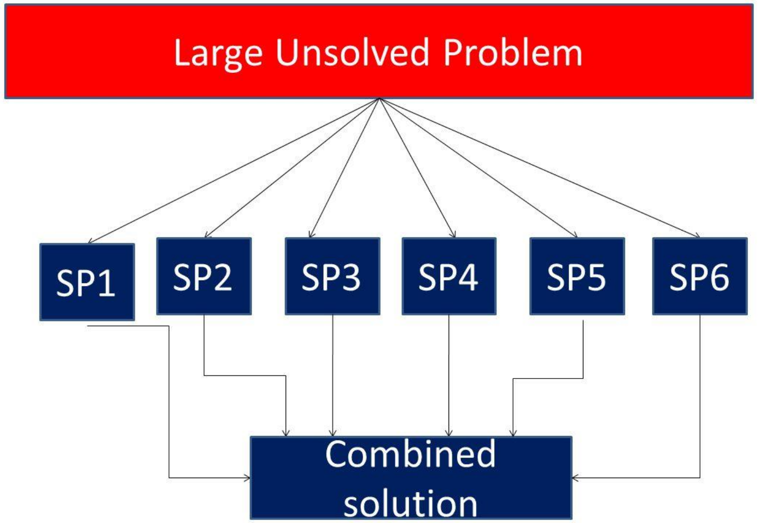
So the dual becomes

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

Logic, Duality and Decomposition

Often, **decomposition** is the key to solving large optimization problems.

- Break the problem into **smaller components**.
- Solution time increases **superlinearly** with component size.
- Faster to solve **many small problems** than one large one.



Components must **communicate** somehow to reach globally optimal solution.

- Decomposition can nonetheless make problem tractable,
- Given the **right problem structure**.

Benders Decomposition

Benders decomposition is a well-known and often successful decomposition method.

Benders 1962

Classical Benders decomposition requires an LP subproblem.

- The Benders cuts are obtained from the **LP dual** of the subproblem.

Benders Decomposition

Benders decomposition is a well-known and often successful decomposition method.

Benders 1962

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 problem as the subproblem.

- Benders cuts are obtained from an **inference dual** of the subproblem.

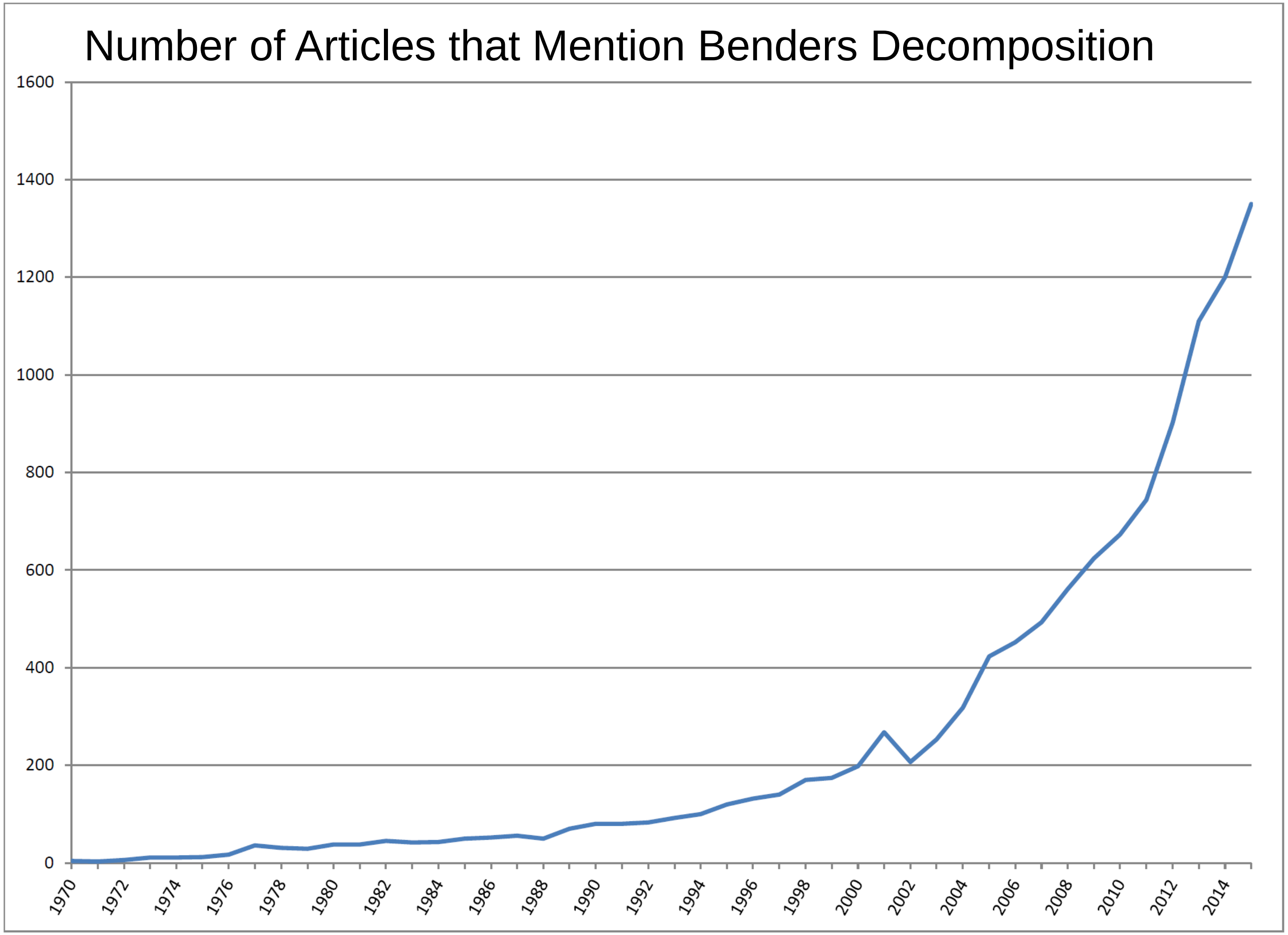
Speedup over state of the art can be several orders of magnitude.

- Benders cuts must be designed specifically for every class of problems.

JH 2000

JH, Ottosson 2003

Logic and Duality



Logic-based Benders

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,

…usually by decoupling into small components

Logic-based Benders

Decompose problem into **master** and **subproblem**.

Subproblem is obtained by fixing x to solution value in master problem.

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.

Logic-based Benders

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 $z \geq g_k(x)$

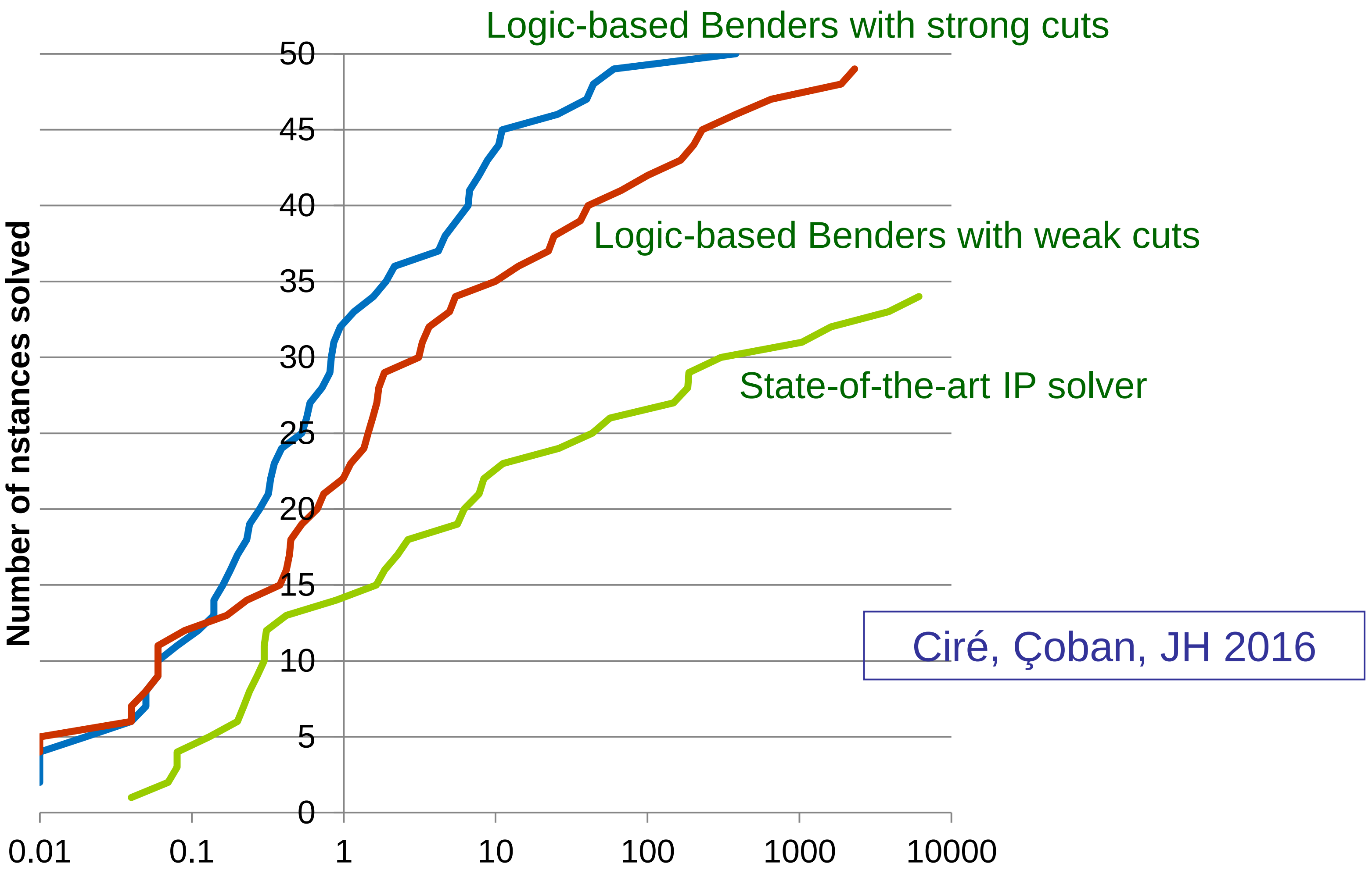
Subproblem

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

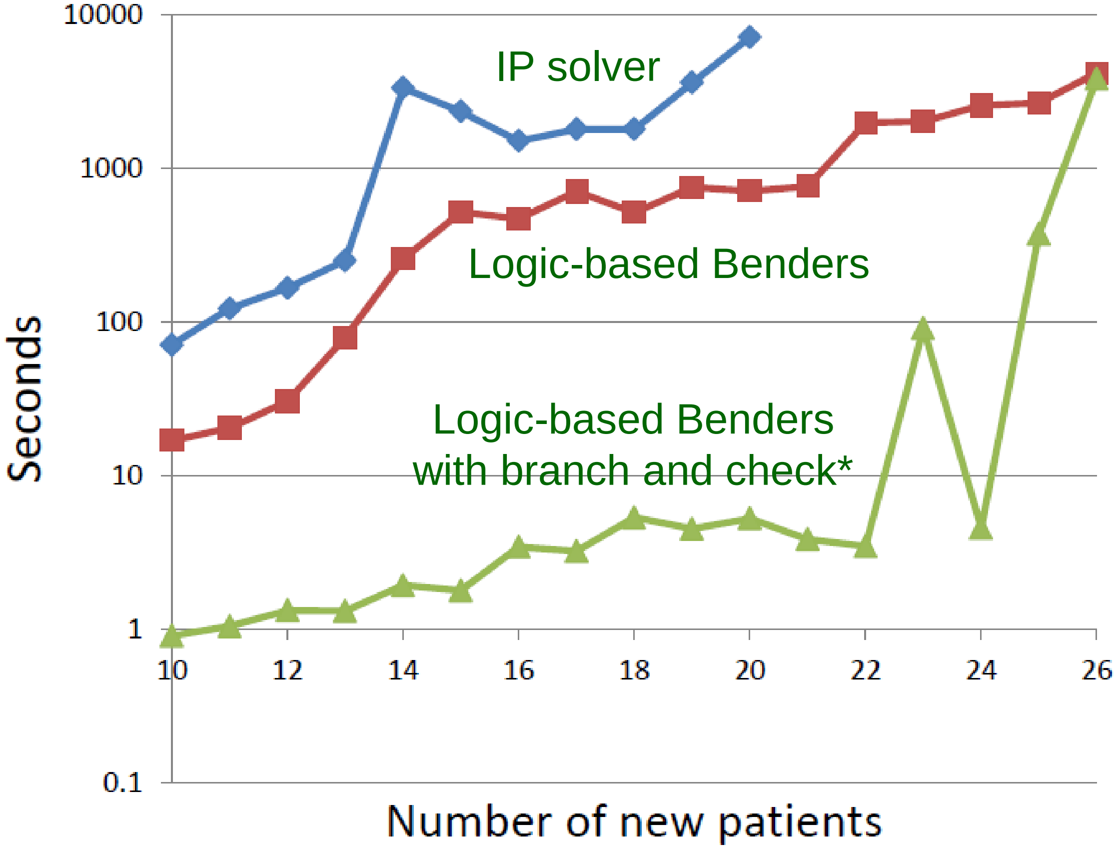
Obtain proof of optimality (solution of inference dual). Use **same proof** to deduce cost bounds for other assignments, yielding Benders cut.

Machine Assignment and Scheduling

Performance profile



Home Healthcare Routing and Scheduling



Heching, JH, Kimura 2018

*Cuts are generated during a single branch-and-bound solution of master problem

Logic-based Benders Applications

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-based Benders Applications

Other scheduling applications:

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

Logic-based Benders Applications

LBBD in routing and scheduling:

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

Logic-based Benders Applications

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

123

Logic-based Benders Applications

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**.

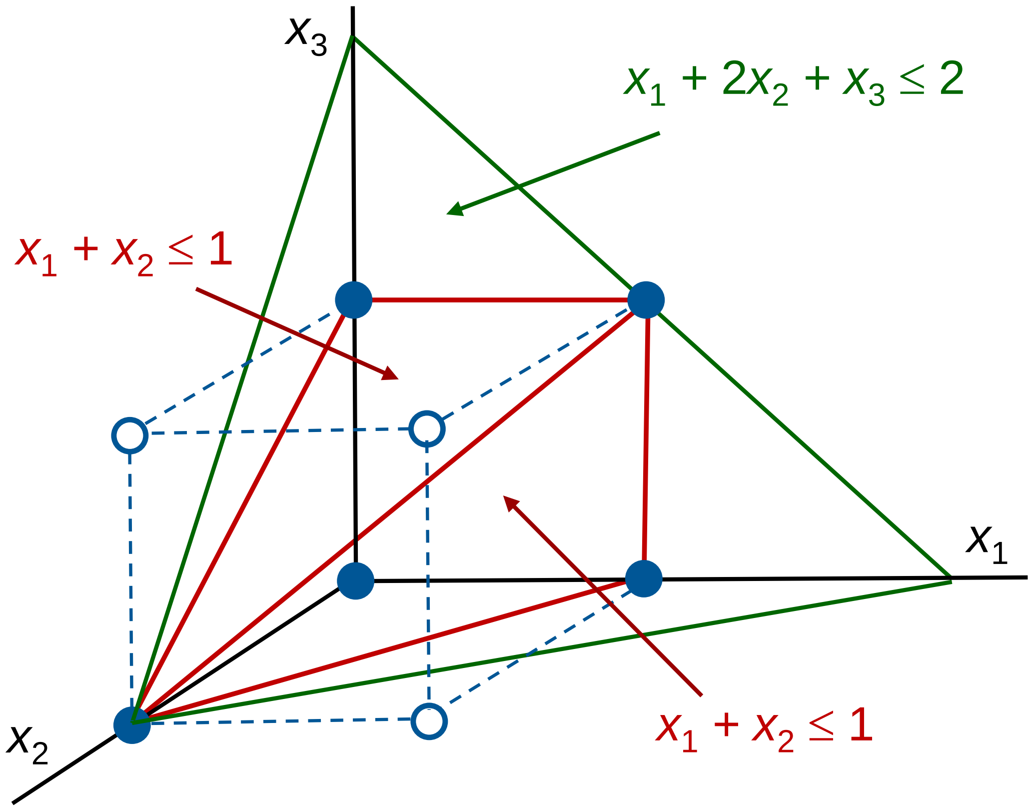
Cuts that achieve consistency **cut off inconsistent 0-1 partial assignments** and so reduce backtracking.

Consistency and Backtracking

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

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.



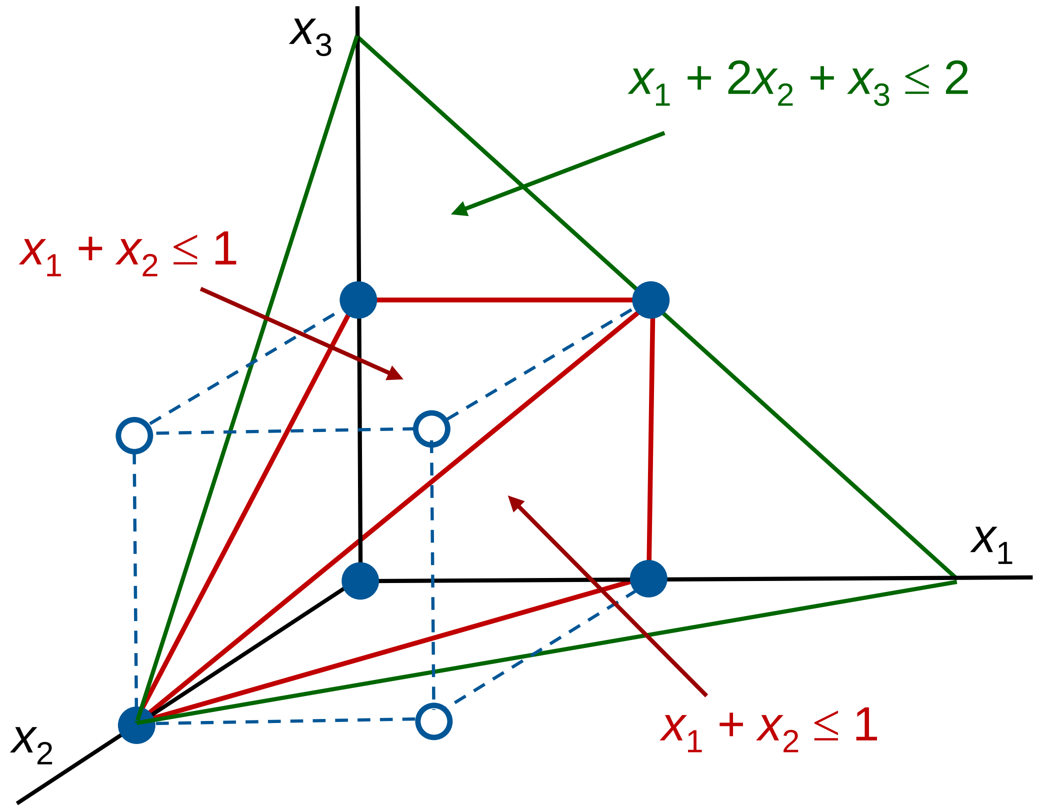
Consistency and Backtracking

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

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

It explicitly excludes infeasible 0-1 partial assignments.

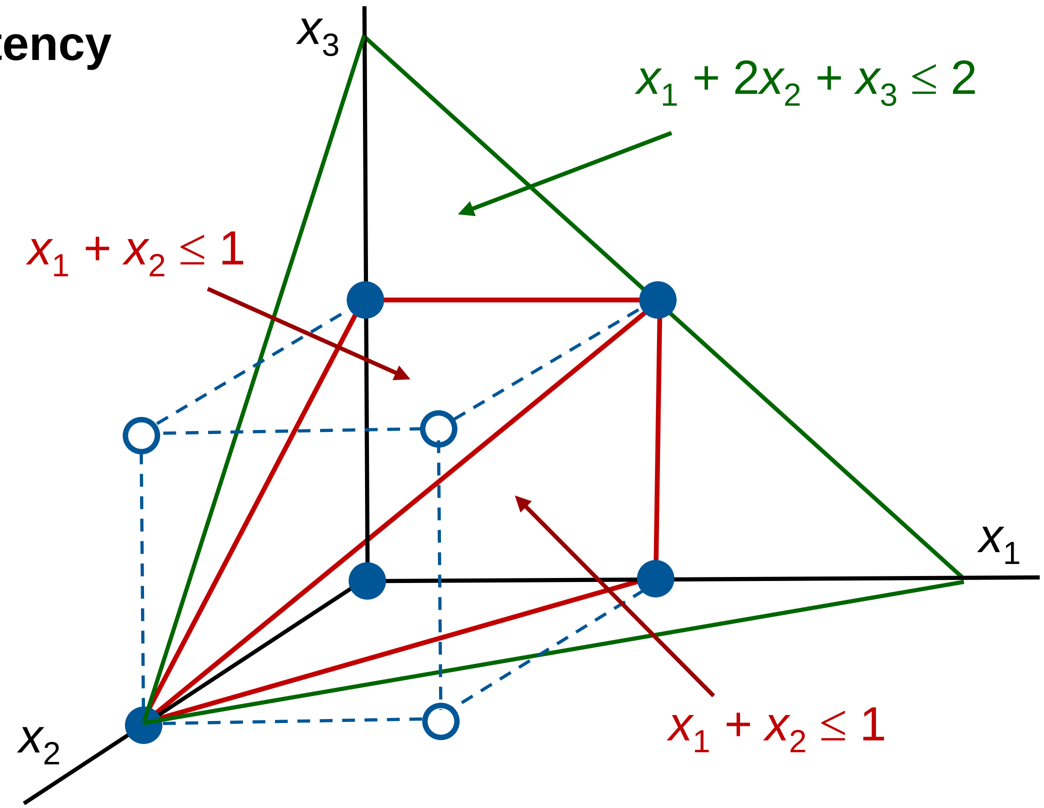
A weak form of LP-consistency can **reduce backtracking** by excluding inconsistent partial assignments that facet-defining inequalities may not exclude.



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.



Questions? Comments?