# Projection in Logic, CP, and Optimization

John Hooker
Carnegie Mellon University

Workshop on Logic and Search
Melbourne, 2017

# Projection as a Unifying Concept

- **Projection** is a fundamental concept in **logic**, **constraint programming**, and **optimization**.
  - **Logical inference** is **projection** onto a subset of variables.
  - **Consistency maintenance** in CP is a **projection problem**.
  - **Optimization** is **projection** onto a cost variable.

# Projection as a Unifying Concept

- **Projection** is a fundamental concept in **logic**, **constraint programming**, and **optimization**.
  - **Logical inference** is **projection** onto a subset of variables.
  - **Consistency maintenance** in CP is a **projection problem**.
  - **Optimization** is **projection** onto a cost variable.

- Recognizing this unity can lead to **faster search methods**.
  - In both logic and optimization.

# Projection as a Unifying Concept

- Two fundamental **projection methods** occur across multiple fields.

# Projection as a Unifying Concept

- Two fundamental **projection methods** occur across multiple fields.

- **Fourier-Motzkin Elimination** and generalizations.
  - Polyhedral projection.
  - Probability logic
  - Propositional logic (resolution)
  - Integer programming (cutting planes & modular arithmetic)
  - Some forms of consistency maintenance

# Projection as a Unifying Concept

- Two fundamental **projection methods** occur across multiple fields.

- **Benders decomposition** and generalizations.
  - Optimization.
  - Probability logic (column generation)
  - Propositional logic (conflict clauses)
  - First-order logic (partial instantiation)

# Outline

- Projection using **Fourier-Motzkin elimination**
- **Consistency maintenance** as projection
- Projection using **Benders decomposition**

# What Is Projection?

- Projection yields a **constraint set**.
  - We project a **constraint set** onto a **subset of its variables** to obtain **another constraint set**.

# What Is Projection?

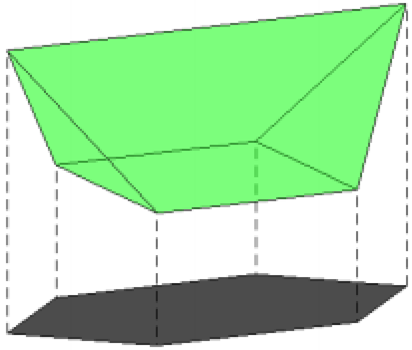
- Projection yields a **constraint set**.
  - We project a **constraint set** onto a **subset of its variables** to obtain **another constraint set**.

- Formal definition
  - Let $x = (x_1, \ldots, x_n)$
  - Let $\bar{x} = (x_1, \ldots, x_k),\ k < n$
  - Let $\mathcal{C}$ be a constraint set.
  - The projection of $\mathcal{C}$ onto $\bar{x}$ is a constraint set, containing only variables in $\bar{x}$, whose satisfaction set is $\{\bar{x} \mid x \text{ satisfies } \mathcal{C}\}$

# Projection Using
# Fourier-Motzkin Elimination
## and Its Generalizations

# Polyhedral Projection

- We wish to project a polyhedron onto a subspace.
  - A method based on an idea of Fourier was proposed by Motzkin.
  - The basic idea of Fourier-Motzkin elimination can be used to compute projections in several contexts.



Fourier (1827)

Motzkin (1936)

# Polyhedral Projection

- Eliminate variables we want to project out.
  - To project $\{x \mid Ax \geq b\}$ onto $x_1, \ldots, x_k$
    project out all variables except $x_1, \ldots, x_k$
  - To project out $x_j$, eliminate it from pairs of inequalities:

$$\begin{array}{r} c_0 x_j + c\bar{x} \geq \gamma \quad (1/c_0) \\ -d_0 x_j + d\bar{x} \geq \delta \quad (1/d_0) \\ \hline \left(\dfrac{c}{c_0} + \dfrac{d}{d_0}\right)\bar{x} \geq \dfrac{\gamma}{c_0} + \dfrac{\delta}{d_0} \end{array}$$

where $c_0, d_0 \geq 0$

  - Then remove all inequalities containing $x_j$

# Polyhedral Projection

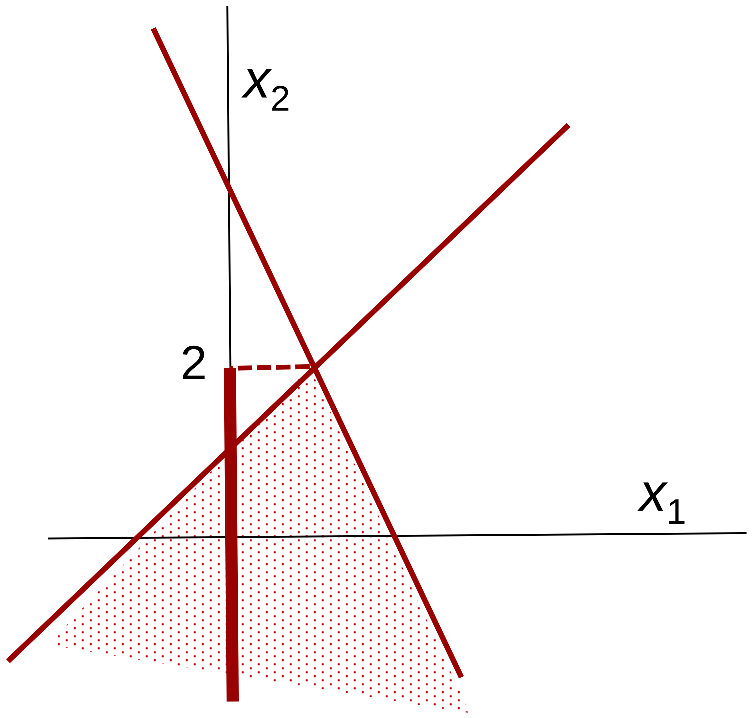
- Example
  - Project $\begin{array}{l} -2x_1 - x_2 \geq -4 \\ x_1 - x_2 \geq -1 \end{array}$ onto $x_2$

    by projecting out $x_1$

$$\begin{array}{rl} -2x_1 - x_2 \geq -4 & (1/2) \\ x_1 - x_2 \geq -1 & (1) \\ \hline -\frac{3}{2}x_2 \geq -3 & \end{array}$$

or

$$x_2 \leq 2$$



# Optimization as Projection

- Optimization is projection onto a single variable.
  - To solve $\min / \max \{f(x) \mid x \in S\}$

    project $\{(x_0, x) \mid x_0 = f(x),\ x \in S\}$

    onto $x_0$ to obtain an interval $x_0^{\min} \leq x_0 \leq x_0^{\max}$

# Optimization as Projection

- Optimization is projection onto a single variable.
  - To solve $\min / \max \{f(x) \mid x \in S\}$
    project $\{(x_0, x) \mid x_0 = f(x),\ x \in S\}$
    onto $x_0$ to obtain an interval $x_0^{\min} \leq x_0 \leq x_0^{\max}$
- Linear programming
  - We can in principle solve $\min / \max \{cx \mid Ax \geq b\}$ with Fourier-Motzkin elimination by projecting $\{(x_0, x) \mid x_0 = cx,\ Ax \geq b\}$ onto $x_0$
  - But this is extremely inefficient.
  - Use simplex or interior point method instead.

# Probability Logic

- Inference in **probability logic** is a polyhedral projection problem
  - Originally stated by George Boole.
  - The **linear programming problem** can be solved, in principle, by Fourier-Motzkin elimination.
- The problem
  - Given a **probability interval** for each of several formulas in propositional logic,
  - Deduce a probability interval for a target formula.

# Probability Logic

## Example

| Formula | Probability |
|---|---|
| $x_1$ | 0.9 |
| if $x_1$ then $x_2$ | 0.8 |
| if $x_2$ then $x_3$ | 0.4 |

Deduce probability range for $x_3$

Boole (1854)

# Probability Logic

## Example

| Formula | Probability |
|---|---|
| $x_1$ | 0.9 |
| if $x_1$ then $x_2$ | 0.8 |
| if $x_2$ then $x_3$ | 0.4 |

Interpret if-then statements as material conditionals

Deduce probability range for $x_3$

Boole (1854)

# Probability Logic

## Example

| Formula | Probability |
|---|---|
| $x_1$ | 0.9 |
| $\bar{x}_1 \vee x_2$ | 0.8 |
| $\bar{x}_2 \vee x_3$ | 0.4 |

Interpret if-then statements
as material conditionals

Deduce probability
range for $x_3$

Boole (1854)

# Probability Logic

## Example

| Formula | Probability |
|---|---|
| $x_1$ | 0.9 |
| $\bar{x}_1 \vee x_2$ | 0.8 |
| $\bar{x}_2 \vee x_3$ | 0.4 |

Deduce probability range for $x_3$

### Linear programming model

$$\min/\max\ \pi_0$$

$$\begin{bmatrix} 01010101 \\ 00001111 \\ 11110011 \\ 11011101 \\ 11111111 \end{bmatrix} \begin{bmatrix} p_{000} \\ p_{001} \\ p_{010} \\ \vdots \\ p_{111} \end{bmatrix} = \begin{bmatrix} \pi_0 \\ 0.9 \\ 0.8 \\ 0.4 \\ 1 \end{bmatrix}$$

$p_{000}$ = probability that $(x_1,x_2,x_3) = (0,0,0)$

Hailperin (1976)

Nilsson (1986)

# Probability Logic

## Example

| Formula | Probability |
|---|---|
| $x_1$ | 0.9 |
| $\bar{x}_1 \vee x_2$ | 0.8 |
| $\bar{x}_2 \vee x_3$ | 0.4 |

Deduce probability range for $x_3$

Linear programming model

$$\min/\max \ \pi_0$$

$$\begin{bmatrix} 01010101 \\ 00001111 \\ 11110011 \\ 11011101 \\ 11111111 \end{bmatrix} \begin{bmatrix} p_{000} \\ p_{001} \\ p_{010} \\ \vdots \\ p_{111} \end{bmatrix} = \begin{bmatrix} \pi_0 \\ 0.9 \\ 0.8 \\ 0.4 \\ 1 \end{bmatrix}$$

$p_{000}$ = probability that $(x_1,x_2,x_3) = (0,0,0)$

Solution: $\pi_0 \in [0.1, 0.4]$

Hailperin (1976)

Nilsson (1986)

# Inference as Projection

- Projection can be viewed as the fundamental inference problem.
  - Deduce information that pertains to a desired subset of propositional variables.
- In propositional logic (SAT), this can be achieved by the **resolution** method.
  - CNF analog of Quine's **consensus** method for DNF.

# Inference as Projection

- Project onto propositional variables of interest
  - Suppose we wish to infer from these clauses everything we can about propositions $x_1$, $x_2$, $x_3$

$$
\begin{array}{lllllll}
x_1 & & & \vee\, x_4 & \vee\, x_5 & & \\
x_1 & & & \vee\, x_4 & \vee\, \bar{x}_5 & & \\
x_1 & & & & \vee\, x_5 & \vee\, x_6 & \\
x_1 & & & & \vee\, x_5 & \vee\, \bar{x}_6 & \\
 & x_2 & & & \vee\, \bar{x}_5 & \vee\, x_6 & \\
 & x_2 & & & \vee\, \bar{x}_5 & \vee\, \bar{x}_6 & \\
 & & x_3 & \vee\, \bar{x}_4 & \vee\, x_5 & & \\
 & & x_3 & \vee\, \bar{x}_4 & \vee\, \bar{x}_5 & &
\end{array}
$$

# Inference as Projection

- Project onto propositional variables of interest
  - Suppose we wish to infer from these clauses everything we can about propositions $x_1, x_2, x_3$

$$
\begin{array}{l}
x_1 \vee x_4 \vee x_5 \\
x_1 \vee x_4 \vee \bar{x}_5 \\
x_1 \vee x_5 \vee x_6 \\
x_1 \vee x_5 \vee \bar{x}_6 \\
x_2 \vee \bar{x}_5 \vee x_6 \\
x_2 \vee \bar{x}_5 \vee \bar{x}_6 \\
x_3 \vee \bar{x}_4 \vee x_5 \\
x_3 \vee \bar{x}_4 \vee \bar{x}_5
\end{array}
$$

We can deduce

$$x_1 \vee x_2$$

$$x_1 \vee x_3$$

This is a projection onto $x_1, x_2, x_3$

# Inference as Projection

- Resolution as a projection method
  - Similar to Fourier-Motzkin elimination
    - Actually, identical to Fourier-Motzkin elimination + rounding
  - To project out $x_j$, eliminate it from pairs of clauses:

$$C \vee x_j \qquad\qquad D \vee \bar{x}_j$$

$$\searrow \qquad \swarrow$$

$$C \vee D$$

  - Then remove all clauses containing $x_j$

Quine (1952,1955)
JH (1992,2012)

# Inference as Projection

- Interpretation as Fourier-Motzkin + rounding
  - Project out $x_1$ using resolution:

$$\begin{array}{l} x_1 \vee x_2 \vee x_3 \\ \bar{x}_1 \qquad\quad \vee x_3 \vee x_4 \\ \hline \qquad\quad x_2 \vee x_3 \vee x_4 \end{array}$$

# Inference as Projection

- Interpretation as Fourier-Motzkin + rounding
  - Project out $x_1$ using resolution:

$$\frac{\begin{array}{l} x_1 \vee x_2 \vee x_3 \\ \bar{x}_1 \vee x_3 \vee x_4 \end{array}}{x_2 \vee x_3 \vee x_4}$$

  - Project out $x_1$ using Fourier-Motzkin + rounding

$$\begin{array}{rl} x_1 + x_2 + x_3 \geq 1 & (1/2) \\ -x_1 + x_3 + x_4 \geq 0 & (1/2) \\ x_2 \geq 0 & (1/2) \\ x_4 \geq 0 & (1/2) \\ \hline x_2 + x_3 + x_4 \geq \frac{1}{2} & \end{array}$$

$x_j$ = 1,0 corresponds to $x_j$ = T, F

Williams (1987)

rounds to $x_2 + x_3 + x_4 \geq 1$ since $x_j$s are integer

# Projection and Cutting Planes

- A resolvent is a special case of a **rank 1 Chvátal cut**.
  - A general inference method for **integer programming**.
  - All rank 1 cuts can be obtained by taking **nonnegative linear combinations** and **rounding**.
  - We can deduce **all valid inequalities** by recursive generation of rank 1 cuts.
  - …including inequalities describing the **projection** onto a given subset of variables.
  - The minimum number of iterations necessary is the **Chvátal rank** of the constraint set.
  - There is **no upper bound** on the rank as a function of the number of variables.

Chvátal 1973

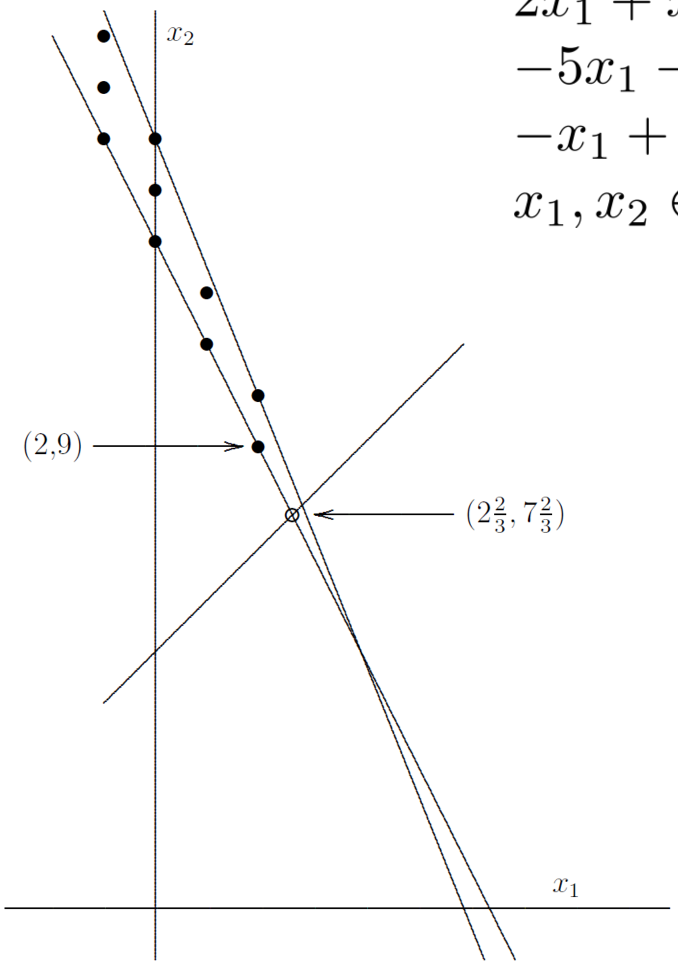
# Projection Methods

- Generalizations of resolution
  - For cardinality clauses JH (1988)
  - For 0-1 linear inequalities JH (1992)
  - For general integer linear inequalities Williams & JH (2015)

# Projection for Integer Programming

Example: solve

$$\begin{array}{ll} \min\ x_2 & \\ 2x_1 + x_2 \geq 13 & \text{C1} \\ -5x_1 - 2x_2 \geq -30 & \text{C2} \\ -x_1 + x_2 \geq 5 & \text{C3} \\ x_1, x_2 \in \mathbb{Z} & \end{array}$$



# Projection for Integer Programming

Example: solve

$$
\begin{array}{ll}
\min\ x_2 & \\
2x_1 + x_2 \geq 13 & \text{C1} \\
-5x_1 - 2x_2 \geq -30 & \text{C2} \\
-x_1 + x_2 \geq 5 & \text{C3} \\
x_1, x_2 \in \mathbb{Z} &
\end{array}
$$

To project out $x_1$, first combine C1 and C2:

$$
\begin{array}{rl}
2x_1 + \ \ x_2 \geq \ \ \ 13 & (5) \\
-5x_1 - 2x_2 \geq -30 & (2) \\
\hline
5(x_2 - 13) + 2(-2x_2 + 30) \geq 0 &
\end{array}
$$

# Projection for Integer Programming

Example: solve

$$\begin{array}{ll} \min\ x_2 & \\ 2x_1 + x_2 \geq 13 & \text{C1} \\ -5x_1 - 2x_2 \geq -30 & \text{C2} \\ -x_1 + x_2 \geq 5 & \text{C3} \\ x_1, x_2 \in \mathbb{Z} & \end{array}$$

To project out $x_1$, first combine C1 and C2:

$$\begin{array}{rl} 2x_1 + \ \ x_2 \geq \ \ \ 13 & (5) \\ -5x_1 - 2x_2 \geq -30 & (2) \\ \hline 5(x_2 - 13) + 2(-2x_2 + 30) \geq 0 & \end{array}$$

Since 2nd term is even, we can write this as

$$5(x_2 - 13 - u) + 2(-2x_2 + 30) \geq 0, \ \ x_2 - 13 - u \equiv 0 \pmod 2$$

where $u \in \{0, 1\}$. This simplifies to

$$x_2 \geq 5 + 5u, \ \ x_2 \equiv u + 1 \pmod 2$$

# Projection for Integer Programming

Example: solve

$$\begin{aligned} &\min\ x_2 \\ &2x_1 + x_2 \geq 13 && \text{C1} \\ &-5x_1 - 2x_2 \geq -30 && \text{C2} \\ &-x_1 + x_2 \geq 5 && \text{C3} \\ &x_1, x_2 \in \mathbb{Z} \end{aligned}$$

After similarly combining C1 and C3, we get the problem with $x_1$ projected out:

$$\begin{aligned} &\min\ x_2 \\ &x_2 \geq 5 + 5u,\ \ 3x_2 \geq 23 + u \\ &x_2 \equiv u + 1 \pmod 2,\ \ u \in \{0, 1\} \end{aligned}$$

# Projection for Integer Programming

Example: solve

$$\begin{array}{ll} \min\ x_2 & \\ 2x_1 + x_2 \geq 13 & \text{C1} \\ -5x_1 - 2x_2 \geq -30 & \text{C2} \\ -x_1 + x_2 \geq 5 & \text{C3} \\ x_1, x_2 \in \mathbb{Z} & \end{array}$$

After similarly combining C1 and C3, we get the problem with $x_1$ projected out:

$$\begin{array}{l} \min\ x_2 \\ x_2 \geq 5 + 5u,\ \ 3x_2 \geq 23 + u \\ x_2 \equiv u + 1 \pmod 2,\ \ u \in \{0, 1\} \end{array}$$

This is equivalent to

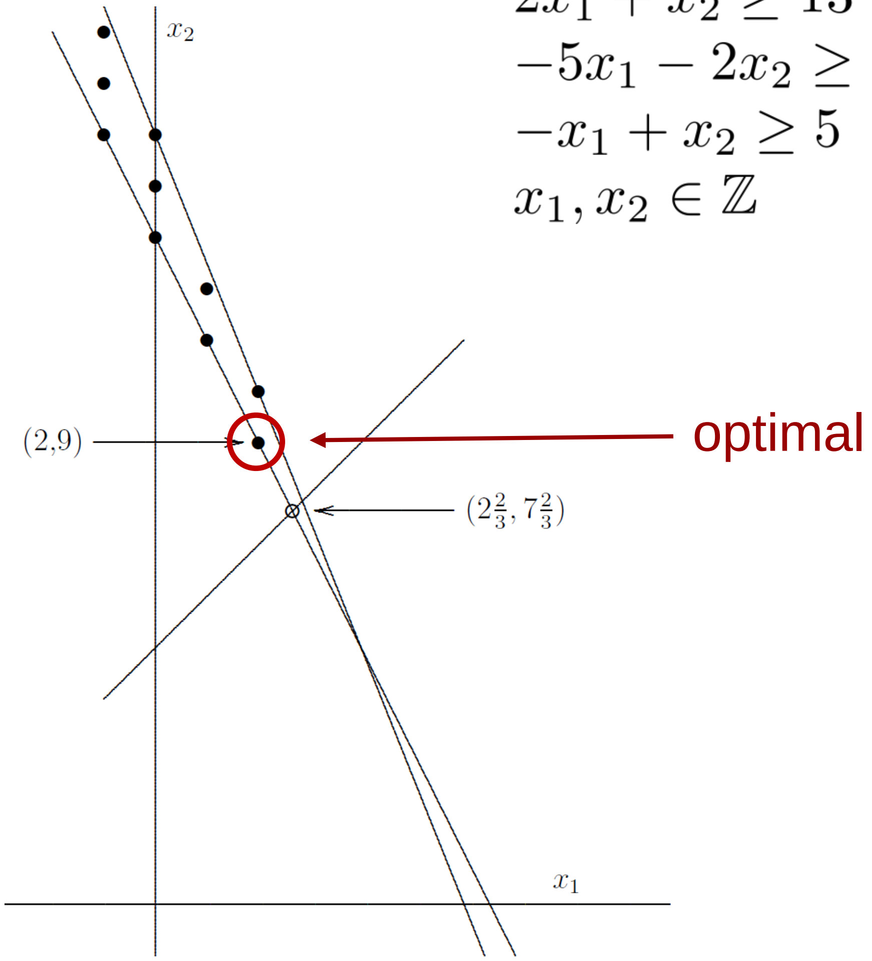
$$\begin{array}{l} \min\ x_2 (= 9) \\ x_2 \geq 5,\ \ 3x_2 \geq 23 \\ x_2 \text{ odd} \end{array} \quad \text{or} \quad \begin{array}{l} \min\ x_2 (= 10) \\ x_2 \geq 10,\ \ 3x_2 \geq 24 \\ x_2 \text{ even} \end{array}$$

So optimal value = 9.

# Projection for Integer Programming

Example: solve

$$\begin{aligned} &\min\ x_2 \\ &2x_1 + x_2 \geq 13 && \text{C1} \\ &-5x_1 - 2x_2 \geq -30 && \text{C2} \\ &-x_1 + x_2 \geq 5 && \text{C3} \\ &x_1, x_2 \in \mathbb{Z} \end{aligned}$$




Number of iterations to compute a projection is bounded by number of variables projected out, unlike Chvátal cuts, for which number of iterations is unbounded.

# Consistency Maintenance as Projection

# Consistency as Projection

- Domain consistency
  - Domain of variable $x_j$ contains only values that $x_j$ assumes in some feasible solution.
  - Equivalently, domain of $x_j$ = **projection** of feasible set onto $x_j$.

# Consistency as Projection

- Domain consistency
  - Domain of variable $x_j$ contains only values that $x_j$ takes in some feasible solution.
  - Equivalently, domain of $x_j$ = **projection** of feasible set onto $x_j$.

**Example:**

Constraint set

$$\text{alldiff}(x_1, x_2, x_3)$$
$$x_1 \in \{a, b\}$$
$$x_2 \in \{a, b\}$$
$$x_3 \in \{b, c\}$$

# Consistency as Projection

- Domain consistency
  - Domain of variable $x_j$ contains only values that $x_j$ takes in some feasible solution.
  - Equivalently, domain of $x_j$ = **projection** of feasible set onto $x_j$.

**Example:**

| Constraint set | Solutions |
|---|---|
| $\text{alldiff}(x_1, x_2, x_3)$ | $(x_1, x_2, x_3)$ |
| $x_1 \in \{a, b\}$ | $(a, b, c)$ |
| $x_2 \in \{a, b\}$ | $(b, a, c)$ |
| $x_3 \in \{b, c\}$ | |

# Consistency as Projection

- Domain consistency
  - Domain of variable $x_j$ contains only values that $x_j$ takes in some feasible solution.
  - Equivalently, domain of $x_j$ = **projection** of feasible set onto $x_j$.

**Example:**

| Constraint set | Solutions |
|---|---|
| $\text{alldiff}(x_1, x_2, x_3)$ | $(x_1, x_2, x_3)$ |
| $x_1 \in \{a, b\}$ | $(a, b, c)$ |
| $x_2 \in \{a, b\}$ | $(b, a, c)$ |
| $x_3 \in \{b, c\}$ | |

Projection onto $x_1$

$$x_1 \in \{a, b\}$$

Projection onto $x_2$

$$x_2 \in \{a, b\}$$

Projection onto $x_3$

$$x_3 \in \{c\}$$

# Consistency as Projection

- Domain consistency
  - Domain of variable $x_j$ contains only values that $x_j$ takes in some feasible solution.
  - Equivalently, domain of $x_j$ = **projection** of feasible set onto $x_j$.

**Example:**

Projection onto $x_1$

$$x_1 \in \{a,b\}$$

| Constraint set | Solutions |
| --- | --- |
| $\text{alldiff}(x_1,x_2,x_3)$ | $(x_1,x_2,x_3)$ |
| $x_1 \in \{a,b\}$ | $(a,b,c)$ |
| $x_2 \in \{a,b\}$ | $(b,a,c)$ |
| $x_3 \in \{b,c\}$ | |

Projection onto $x_2$

$$x_2 \in \{a,b\}$$

Projection onto $x_3$

$$x_3 \in \{c\}$$

This achieves domain consistency.

# Consistency as Projection

- *k*-consistency

$x_J = (x_j \mid j \in J)$

  - Can be defined:
    - A constraint set $S$ is $k$-consistent if:
      - for every $J \subseteq \{1, \ldots, n\}$ with $|J| = k - 1$,
      - every assignment $x_J = v_J \in D_j$ for which $(x_J, x_j)$ does not violate $S$,
      - and every variable $x_j \notin x_J$,

      there is an assignment $x_j = v_j \in D_j$ for which $(x_J, x_j) = (v_J, v_j)$ does not violate $S$.

# Consistency as Projection

$x_J = (x_j \mid j \in J)$

- *k*-consistency
  - Can be defined:
    - A constraint set $S$ is $k$-consistent if:
      - for every $J \subseteq \{1, \ldots, n\}$ with $|J| = k - 1$,
      - every assignment $x_J = v_J \in D_j$ for which $(x_J, x_j)$ does not violate $S$,
      - and every variable $x_j \notin x_J$,

    there is an assignment $x_j = v_j \in D_j$ for which $(x_J, x_j) = (v_J, v_j)$ does not violate $S$.
  - To achieve k-consistency:
    - **Project** the constraints containing each set of $k$ variables onto subsets of $k - 1$ variables.

# Consistency as Projection

- Consistency and backtracking:
  - Strong *k*-consistency for entire constraint set avoids backtracking…
    - if the primal graph has width < *k* with respect to branching order.

Freuder (1982)

  - No point in achieving strong *k*-consistency for **individual constraints** if we propagate through **domain store**.
    - Domain consistency has same effect.

# *J*-Consistency

- A type of consistency more directly related to projection.
  - Constraint set $S$ is ***J*-consistent** if it contains the **projection** of $S$ onto $x_J$.
    - $S$ is domain consistent if it is $\{ j \}$-consistent for each $j$.

$$x_J = (x_j \mid j \in J)$$

# *J*-Consistency

- *J*-consistency and backtracking:
  - If we project a constraint onto $x_1, x_2, \ldots, x_k,$ the constraint will not cause backtracking as we branch on the remaining variables.
    - A natural strategy is to project out $x_n, x_{n-1}, \ldots$ until computational burden is excessive.

# *J*-Consistency

- *J*-consistency and backtracking:
  - If we project a constraint onto $x_1$, $x_2$, …, $x_k$, the constraint will not cause backtracking as we branch on the remaining variables.
    - A natural strategy is to project out $x_n$, $x_{n-1}$, … until computational burden is excessive.
  - No point in achieving *J*-consistency for **individual constraints** if we propagate through a **domain store**.
    - However, *J*-consistency can be useful if we propagate through a richer data structure
    - …such as **decision diagrams**
    - …which can be more effective as a propagation medium.

JH & Hadžić (2006,2007)
Andersen, Hadžić, JH, Tiedemann (2007)
Bergman, Ciré, van Hoeve, JH (2014)

# Propagating *J*-Consistency

**Example:**

$$\text{among}\left((x_1, x_2), \{c, d\}, 1, 2\right)$$
$$(x_1 = c) \Rightarrow (x_2 = d)$$
$$\text{alldiff}\left(x_1, x_2, x_3, x_4\right)$$
$$x_1, x_2 \in \{a, b, c, d\}$$
$$x_3 \in \{a, b\}$$
$$x_4 \in \{c, d\}$$

Already domain consistent for individual constraints.

If we branch on $x_1$ first, must consider all 4 branches $x_1 = a, b, c, d$

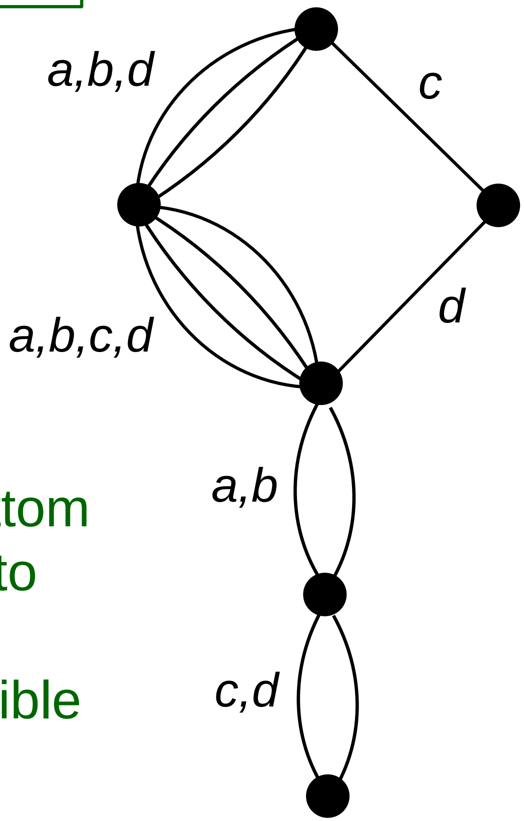
# Propagating *J*-Consistency

**Example:**

$$\text{among}\left((x_1, x_2), \{c, d\}, 1, 2\right)$$
$$(x_1 = c) \Rightarrow (x_2 = d)$$
$$\text{alldiff}\left(x_1, x_2, x_3, x_4\right)$$

$$x_1, x_2 \in \{a, b, c, d\}$$
$$x_3 \in \{a, b\}$$
$$x_4 \in \{c, d\}$$

Suppose we propagate through a relaxed decision diagram of width 2 for these constraints




52 paths from top to bottom represent assignments to $x_1$, $x_2$, $x_3$, $x_4$
36 of these are the feasible assignments.

# Propagating *J*-Consistency

**Example:**

$\text{among}\left((x_1,x_2),\{c,d\},1,2\right)$

$(x_1 = c) \Rightarrow (x_2 = d)$

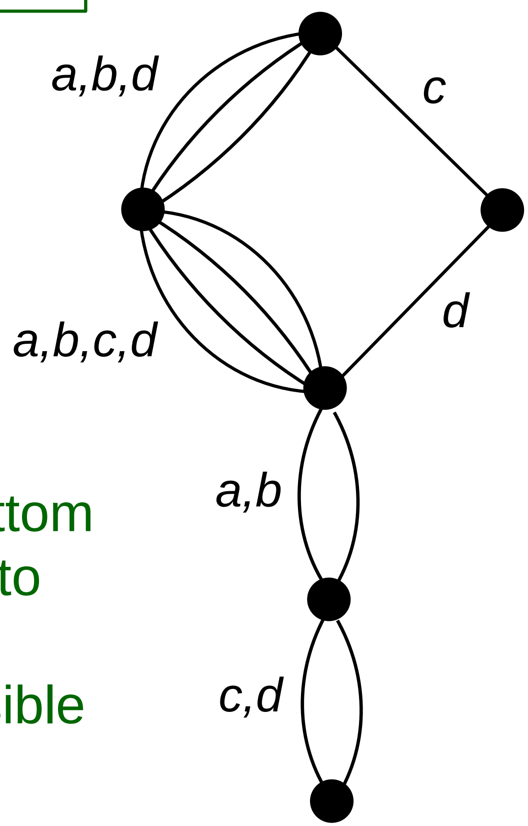
$\text{alldiff}\left(x_1,x_2,x_3,x_4\right)$

$x_1,x_2 \in \{a,b,c,d\}$

$x_3 \in \{a,b\}$

$x_4 \in \{c,d\}$

Suppose we propagate through a relaxed decision diagram of width 2 for these constraints




Projection of alldiff onto $x_1$, $x_2$ is

$\text{alldiff}\left(x_1,x_2\right)$

$\text{atmost}\left((x_1,x_2),\{a,b\},1\right)$

$\text{atmost}\left((x_1,x_2),\{c,d\},1\right)$

52 paths from top to bottom represent assignments to $x_1$, $x_2$, $x_3$, $x_4$

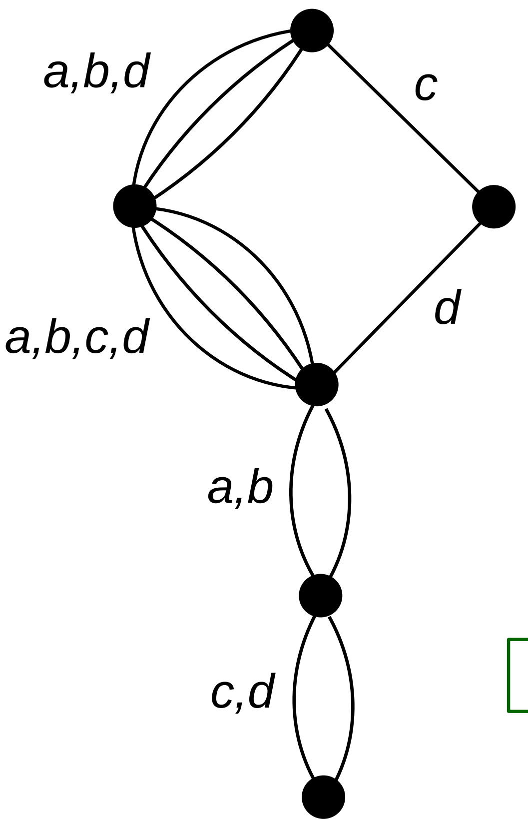
36 of these are the feasible assignments.

# Propagating *J*-Consistency

Let's propagate the 2nd atmost constraint in the projected alldiff through the relaxed decision diagram.

Let the length of a path be number of arcs with labels in {*c*,*d*}.

For each arc, indicate length of shortest path from top to that arc.




Projection of alldiff onto $x_1$, $x_2$ is

$\text{alldiff}(x_1, x_2)$

$\text{atmost}((x_1, x_2), \{a, b\}, 1)$

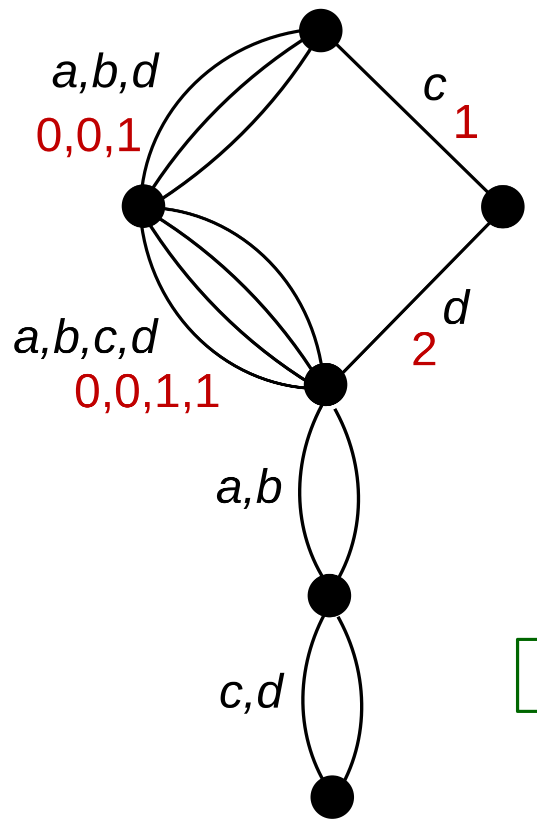
$\text{atmost}((x_1, x_2), \{\text{c}, \text{d}\}, 1)$

# Propagating *J*-Consistency

Let's propagate the 2nd atmost constraint in the projected alldiff through the relaxed decision diagram.

Let the length of a path be number of arcs with labels in {*c*,*d*}.

For each arc, indicate length of shortest path from top to that arc.




Projection of alldiff onto $x_1$, $x_2$ is

$$\text{alldiff}(x_1, x_2)$$

$$\text{atmost}((x_1, x_2), \{a, b\}, 1)$$

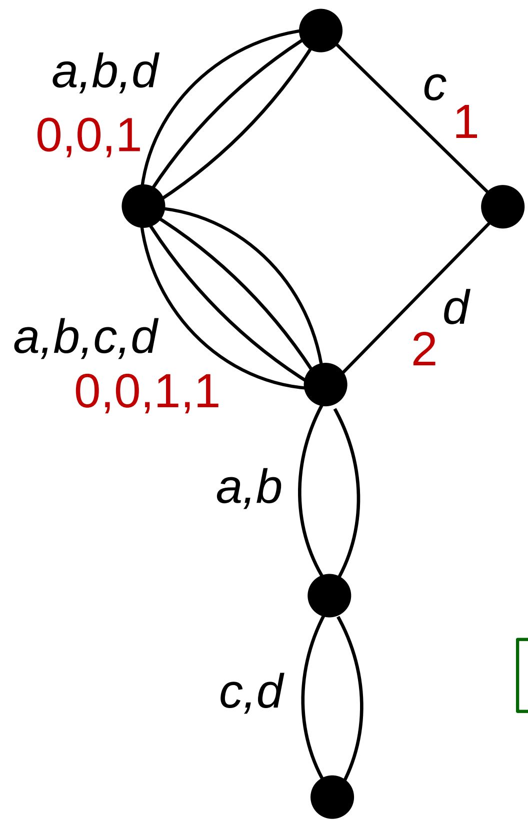
$$\text{atmost}((x_1, x_2), \{\text{c}, \text{d}\}, 1)$$

# Propagating *J*-Consistency

Let's propagate the 2nd atmost constraint in the projected alldiff through the relaxed decision diagram.

Let the length of a path be number of arcs with labels in {*c*,*d*}.

For each arc, indicate length of shortest path from top to that arc.




Remove arcs with label > 1

Projection of alldiff onto $x_1$, $x_2$ is

$$\text{alldiff}(x_1, x_2)$$

$$\text{atmost}\big((x_1, x_2), \{a, b\}, 1\big)$$

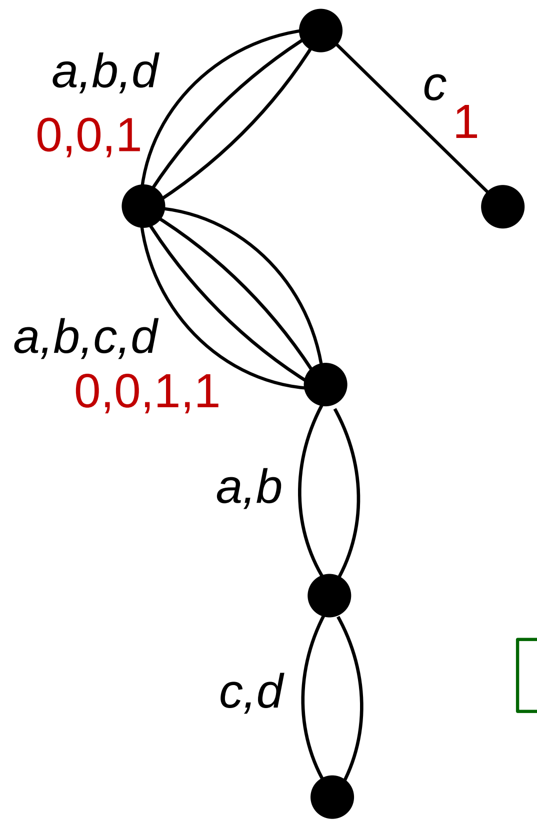
$$\text{atmost}\big((x_1, x_2), \{\text{c}, \text{d}\}, 1\big)$$

# Propagating *J*-Consistency

Let's propagate the 2nd atmost constraint in the projected alldiff through the relaxed decision diagram.

Let the length of a path be number of arcs with labels in {*c*,*d*}.

For each arc, indicate length of shortest path from top to that arc.




Remove arcs with label > 1

Projection of alldiff onto $x_1$, $x_2$ is

$$\text{alldiff}(x_1, x_2)$$

$$\text{atmost}((x_1, x_2), \{a, b\}, 1)$$

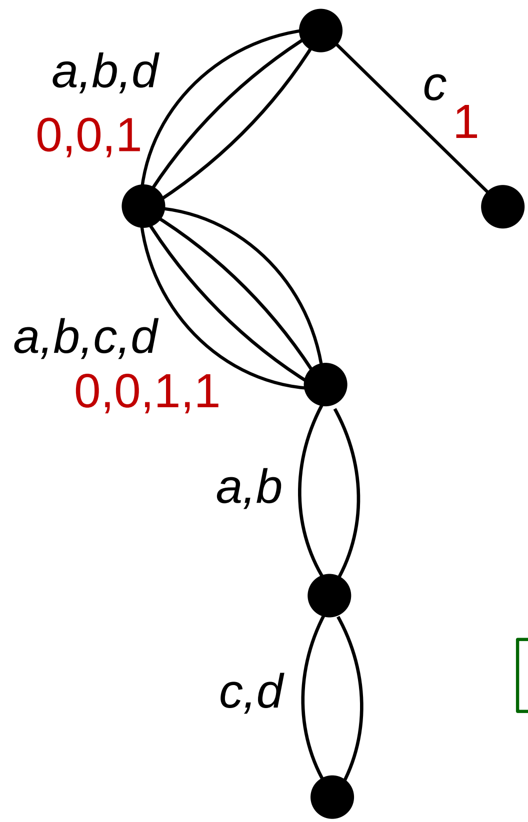
$$\text{atmost}((x_1, x_2), \{\text{c}, \text{d}\}, 1)$$

# Propagating *J*-Consistency

Let's propagate the 2nd atmost constraint in the projected alldiff through the relaxed decision diagram.

Let the length of a path be number of arcs with labels in {*c*,*d*}.

For each arc, indicate length of shortest path from top to that arc.




Remove arcs with label > 1

Clean up.

Projection of alldiff onto $x_1$, $x_2$ is

$\text{alldiff}\left(x_1, x_2\right)$

$\text{atmost}\left((x_1, x_2), \{a, b\}, 1\right)$

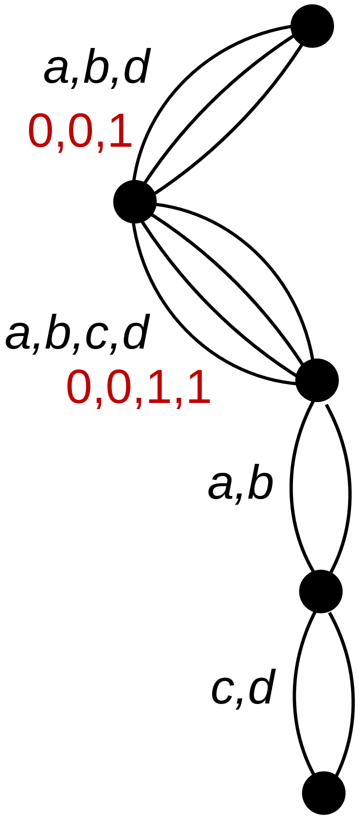
$\text{atmost}\left((x_1, x_2), \{\text{c}, \text{d}\}, 1\right)$

# Propagating *J*-Consistency

Let's propagate the 2nd atmost constraint in the projected alldiff through the relaxed decision diagram.

Let the length of a path be number of arcs with labels in {*c*,*d*}.

For each arc, indicate length of shortest path from top to that arc.




Remove arcs with label > 1

Clean up.

Projection of alldiff onto $x_1$, $x_2$ is

$\text{alldiff}(x_1, x_2)$

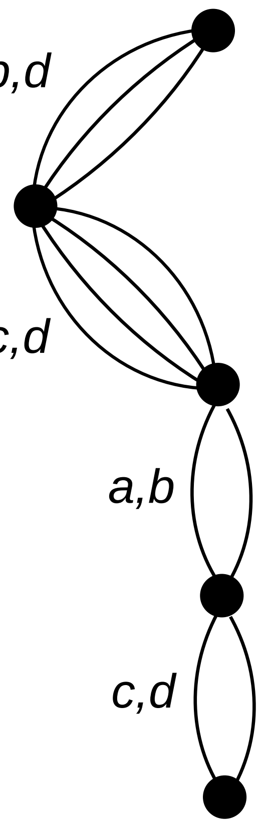
$\text{atmost}((x_1, x_2), \{a, b\}, 1)$

$\text{atmost}((x_1, x_2), \{\text{c}, \text{d}\}, 1)$

# Propagating *J*-Consistency

Let's propagate the 2nd atmost constraint in the projected alldiff through the relaxed decision diagram.

We need only branch on *a,b,d* rather than *a,b,c,d*




Remove arcs with label > 1

Clean up.

Projection of alldiff onto $x_1$, $x_2$ is

$$\text{alldiff}(x_1, x_2)$$

$$\text{atmost}((x_1, x_2), \{a, b\}, 1)$$

$$\text{atmost}((x_1, x_2), \{\text{c}, \text{d}\}, 1)$$

# Achieving *J*-consistency

| **Constraint** | **How hard to project?** |
|---|---|
| among | Easy and fast. |
| sequence | More complicated but fast. Since polyhedron is integral, can write a formula based on **Fourier-Motzkin** |
| regular | Easy and basically same labor as domain consistency. |
| alldiff | Quite complicated but practical for small domains. |

# Projection Using
# Benders Decomposition
## and Its Generalizations

# Logic-Based Benders

- **Logic-based Benders decomposition** is a generalization of classical Benders decomposition.
  - Solves a problem of the form

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

JH (2000), JH & Ottosson (2003)

# Logic-Based Benders

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

**Master problem**

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

Minimize cost $z$ subject to bounds given by Benders cuts, obtained from values of $x$ attempted in 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.
  - This yields optimal solution.

**Master problem**

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

Minimize cost $z$ subject to bounds given by Benders cuts, obtained from values of $x$ attempted in 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

- The Benders cuts define the **projection** of the feasible set onto $(z,x)$.
  - If all possible cuts are generated.

**Master problem**

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

Minimize cost $z$ subject to bounds given by Benders cuts, obtained from values of $x$ attempted in 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-Based Benders

- Fundamental concept: **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 *v** by searching over **proofs *P***.

# Logic-Based Benders

- Popular optimization duals are **special cases** of the inference dual.
  - Result from different choices of **inference method**.
  - For example....
    - Linear programming dual (gives **classical Benders cuts**)
    - Lagrangean dual
    - Surrogate dual
    - Subadditive dual

# Classical Benders

- **Linear programming dual** results in **classical Benders method**.
  - The problem is $\min \; cx + dy$
    $Ax + By \geq b$

Master problem

$\min z$

(Benders cuts)

Minimize cost $z$ subject to bounds given by Benders cuts, obtained from values of $x$ attempted in previous iterations $k$.

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

Benders cut $z \geq cx + u(b - Ax)$

Subproblem

$\min c\bar{x} + dy$

$By \geq b - A\bar{x}$

Obtain proof of optimality by solving **LP dual**:

$\max u(b - A\bar{x})$

$uB \leq d, u \geq 0$

Benders (1962)

# Application to Planning & Scheduling

- Assign tasks in master, schedule in subproblem.
  - Combine **mixed integer programming** and **constraint programming**

**Master problem**

Assign tasks to resources to minimize cost.

Solve by **mixed integer programming**.

→ Trial assignment $\bar{x}$ →

← Benders cut $z \geq g_k(x)$ ←

**Subproblem**

Schedule jobs on each machine, subject to time windows.

**Constraint programming** obtains proof of optimality (dual solution).

Use **same proof** to deduce cost for some other assignments, yielding Benders cut.

# Application to Planning & Scheduling

- Objective function
  - Cost is based on **task assignment only**.

$$\text{cost} = \sum_{ij} c_{ij} x_{ij}, \quad x_{ij} = 1 \text{ if task } j \text{ assigned to resource } i$$

  - So cost appears only in the **master problem**.
  - Scheduling subproblem is a **feasibility problem**.

# Application to Planning & Scheduling

- Objective function
  - Cost is based on **task assignment only**.

    $$\text{cost} = \sum_{ij} c_{ij} x_{ij}, \quad x_{ij} = 1 \text{ if task } j \text{ assigned to resource } i$$

    - So cost appears only in the **master problem**.
    - Scheduling subproblem is a **feasibility problem**.

- Benders cuts
  - They have the form $\sum_{j \in J_i} (1 - x_{ij}) \geq 1, \text{ all } i$
    - where $J_i$ is a set of tasks that create infeasibility when assigned to resource $i$.

# Application to Planning & Scheduling

- Resulting Benders decomposition:

Master problem

$$\min\ z$$
$$z = \sum_{ij} c_{ij} x_{ij}$$
Benders cuts

Trial assignment $\bar{x}$

Benders cuts

$$\sum_{j \in J_i} (1 - x_{ij}) \geq 1,$$

for infeasible resources *i*

Subproblem

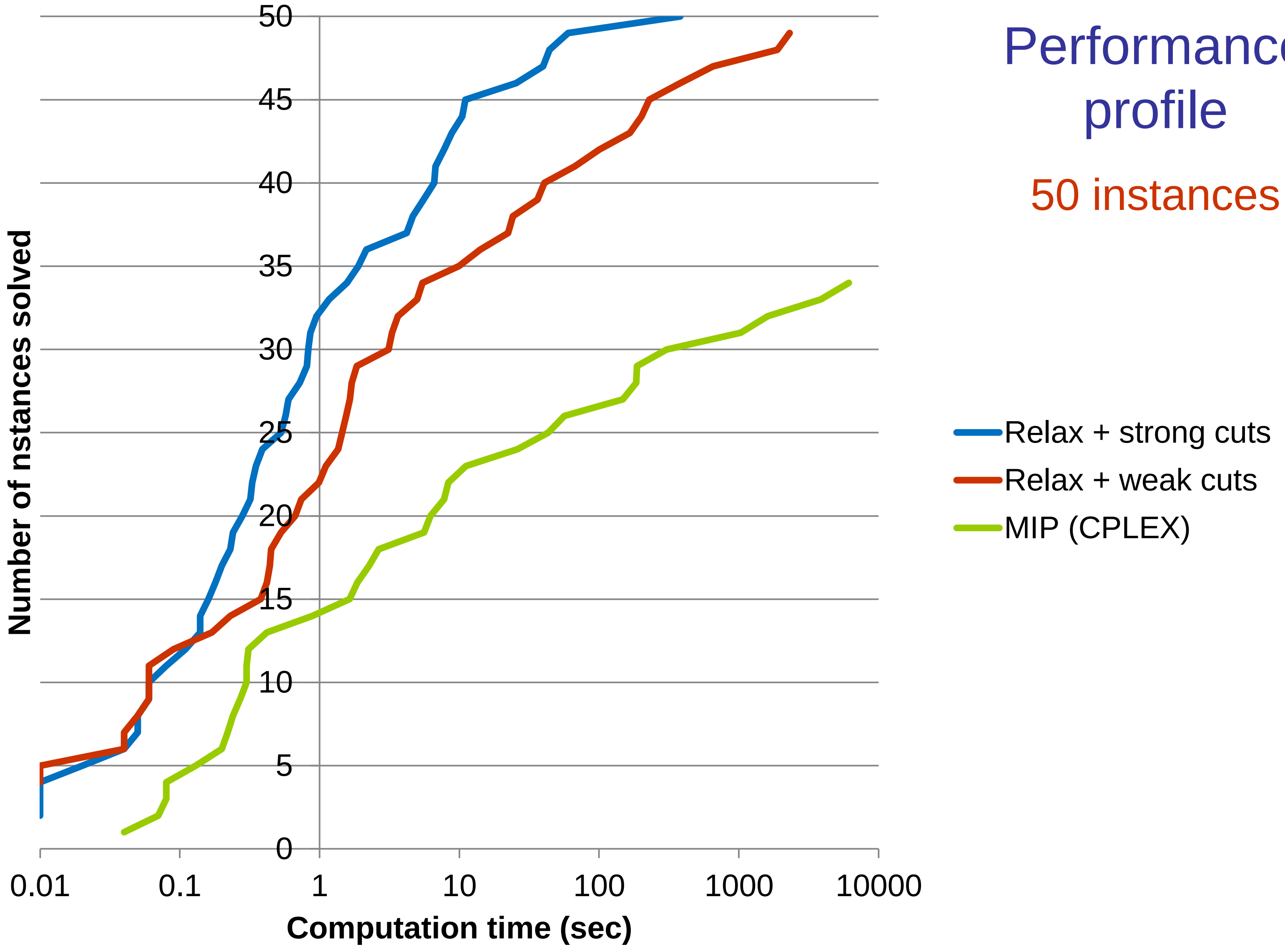
Schedule jobs on each resource.

**Constraint programming** may obtain proof of infeasibility on some resources (dual solution).

Use **same proof** to deduce infeasibility for some other assignments, yielding Benders cut.

# Performance profile

50 instances



# Application to Probability Logic

**Exponentially many** variables in LP model. What to do?

| Formula | Probability |
|---|---|
| $x_1$ | 0.9 |
| $\bar{x}_1 \vee x_2$ | 0.8 |
| $\bar{x}_2 \vee x_3$ | 0.4 |

Deduce probability range for $x_3$

Linear programming model

$$\min/\max \ \pi_0$$

$$\begin{bmatrix} 01010101 \\ 00001111 \\ 11110011 \\ 11011101 \\ 11111111 \end{bmatrix} \begin{bmatrix} p_{000} \\ p_{001} \\ p_{010} \\ \vdots \\ p_{111} \end{bmatrix} = \begin{bmatrix} \pi_0 \\ 0.9 \\ 0.8 \\ 0.4 \\ 1 \end{bmatrix}$$

$p_{000}$ = probability that $(x_1,x_2,x_3) = (0,0,0)$

# Application to Probability Logic

**Exponentially many** variables in LP model. What to do?
Apply classical Benders to **linear programming dual**!

| Formula | Probability |
|---|---|
| $x_1$ | 0.9 |
| $\bar{x}_1 \vee x_2$ | 0.8 |
| $\bar{x}_2 \vee x_3$ | 0.4 |

Deduce probability range for $x_3$

Linear programming model

$$\min/\max\ \pi_0$$

$$\begin{bmatrix} 01010101 \\ 00001111 \\ 11110011 \\ 11011101 \\ 11111111 \end{bmatrix} \begin{bmatrix} p_{000} \\ p_{001} \\ p_{010} \\ \vdots \\ p_{111} \end{bmatrix} = \begin{bmatrix} \pi_0 \\ 0.9 \\ 0.8 \\ 0.4 \\ 1 \end{bmatrix}$$

$p_{000}$ = probability that $(x_1,x_2,x_3) = (0,0,0)$

# Application to Probability Logic

**Exponentially many** variables in LP model. What to do?

Apply classical Benders to **linear programming dual**!

This results in a **column generation** method that introduces variables into LP only as needed to find optimum.

| Formula | Probability |
|---|---|
| $x_1$ | 0.9 |
| $\bar{x}_1 \vee x_2$ | 0.8 |
| $\bar{x}_2 \vee x_3$ | 0.4 |

Deduce probability range for $x_3$

Linear programming model

$$\min/\max \ \pi_0$$

$$\begin{bmatrix} 01010101 \\ 00001111 \\ 11110011 \\ 11011101 \\ 11111111 \end{bmatrix} \begin{bmatrix} p_{000} \\ p_{001} \\ p_{010} \\ \vdots \\ p_{111} \end{bmatrix} = \begin{bmatrix} \pi_0 \\ 0.9 \\ 0.8 \\ 0.4 \\ 1 \end{bmatrix}$$

$p_{000}$ = probability that $(x_1,x_2,x_3) = (0,0,0)$

# Inference as Projection

- Recall that logical inference is a projection problem.
  - We wish to infer from these clauses everything we can about propositions $x_1, x_2, x_3$

$$
\begin{array}{llllll}
x_1 & & & \vee\, x_4 & \vee\, x_5 & \\
x_1 & & & \vee\, x_4 & \vee\, \bar{x}_5 & \\
x_1 & & & & \vee\, x_5 & \vee\, x_6 \\
x_1 & & & & \vee\, x_5 & \vee\, \bar{x}_6 \\
 & x_2 & & & \vee\, \bar{x}_5 & \vee\, x_6 \\
 & x_2 & & & \vee\, \bar{x}_5 & \vee\, \bar{x}_6 \\
 & & x_3 & \vee\, \bar{x}_4 & \vee\, x_5 & \\
 & & x_3 & \vee\, \bar{x}_4 & \vee\, \bar{x}_5 &
\end{array}
$$

We can deduce

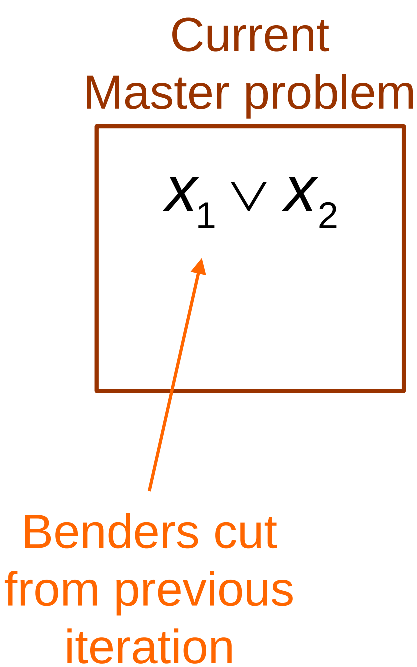
$$x_1 \vee x_2$$

$$x_1 \vee x_3$$

This is a **projection** onto $x_1, x_2, x_3$

# Inference as Projection

- Benders decomposition computes the projection!
  - Benders cuts describe projection onto $x_1$, $x_2$, $x_3$



# Inference as Projection

- Benders decomposition computes the projection!
  - Benders cuts describe projection onto $x_1$, $x_2$, $x_3$

Current Master problem

$$x_1 \vee x_2$$

→ solution of master $(x_1,x_2,x_3) = (0,1,0)$ →

Resulting subproblem

$$x_4 \vee x_5$$
$$x_4 \vee \bar{x}_5$$
$$x_5 \vee x_6$$
$$x_5 \vee \bar{x}_6$$
$$\bar{x}_4 \vee x_5$$
$$\bar{x}_4 \vee \bar{x}_5$$

# Inference as Projection

- Benders decomposition computes the projection!
  - Benders cuts describe projection onto $x_1, x_2, x_3$

Current Master problem

$$x_1 \vee x_2$$

→ solution of master $(x_1,x_2,x_3) = (0,1,0)$ →

Resulting subproblem

$$x_4 \vee x_5$$
$$x_4 \vee \bar{x}_5$$
$$x_5 \vee x_6$$
$$x_5 \vee \bar{x}_6$$
$$\bar{x}_4 \vee x_5$$
$$\bar{x}_4 \vee \bar{x}_5$$

Subproblem is **infeasible**. $(x_1,x_3)=(0,0)$ creates infeasibility

# Inference as Projection

- Benders decomposition computes the projection!
  - Benders cuts describe projection onto $x_1$, $x_2$, $x_3$

Current Master problem

$$x_1 \vee x_2$$
$$x_1 \vee x_3$$

solution of master
$(x_1,x_2,x_3) = (0,1,0)$

Benders cut
(nogood)

Resulting subproblem

$$x_4 \vee x_5$$
$$x_4 \vee \bar{x}_5$$
$$x_5 \vee x_6$$
$$x_5 \vee \bar{x}_6$$
$$\bar{x}_4 \vee x_5$$
$$\bar{x}_4 \vee \bar{x}_5$$

Subproblem is **infeasible.**
$(x_1,x_3)=(0,0)$
creates infeasibility

# Inference as Projection

- Benders decomposition computes the projection!
  - Benders cuts describe projection onto $x_1$, $x_2$, $x_3$

Current Master problem

$$x_1 \vee x_2$$
$$x_1 \vee x_3$$

solution of master
$(x_1, x_2, x_3) = (0,1,1)$

Resulting subproblem

$$x_4 \vee x_5$$
$$x_4 \vee \bar{x}_5$$
$$x_5 \vee x_6$$
$$x_5 \vee \bar{x}_6$$

# Inference as Projection

- Benders decomposition computes the projection!
  - Benders cuts describe projection onto $x_1, x_2, x_3$

Current Master problem

$x_1 \vee x_2$

$x_1 \vee x_3$

solution of master
$(x_1,x_2,x_3) = (0,1,1)$

Resulting subproblem

$x_4 \vee x_5$

$x_4 \vee \bar{x}_5$

$x_5 \vee x_6$

$x_5 \vee \bar{x}_6$

Subproblem is **feasible**

# Inference as Projection

- Benders decomposition computes the projection!
  - Benders cuts describe projection onto $x_1, x_2, x_3$

Current Master problem

$x_1 \vee x_2$

$x_1 \vee x_3$

$x_1 \vee \bar{x}_2 \vee \bar{x}_3$

solution of master $(x_1,x_2,x_3) = (0,1,1)$ →

← Enumerative Benders cut

Resulting subproblem

$x_4 \vee x_5$

$x_4 \vee \bar{x}_5$

$x_5 \vee x_6$

$x_5 \vee \bar{x}_6$

Subproblem is **feasible**

# Inference as Projection

- Benders decomposition computes the projection!
  - Benders cuts describe projection onto $x_1$, $x_2$, $x_3$

Current Master problem

$x_1 \vee x_2$

$x_1 \vee x_3$

$x_1 \vee \bar{x}_2 \vee \bar{x}_3$

solution of master
$(x_1,x_2,x_3) = (0,1,1)$

Enumerative Benders cut

Resulting subproblem

$x_4 \vee x_5$

$x_4 \vee \bar{x}_5$

$x_5 \vee x_6$

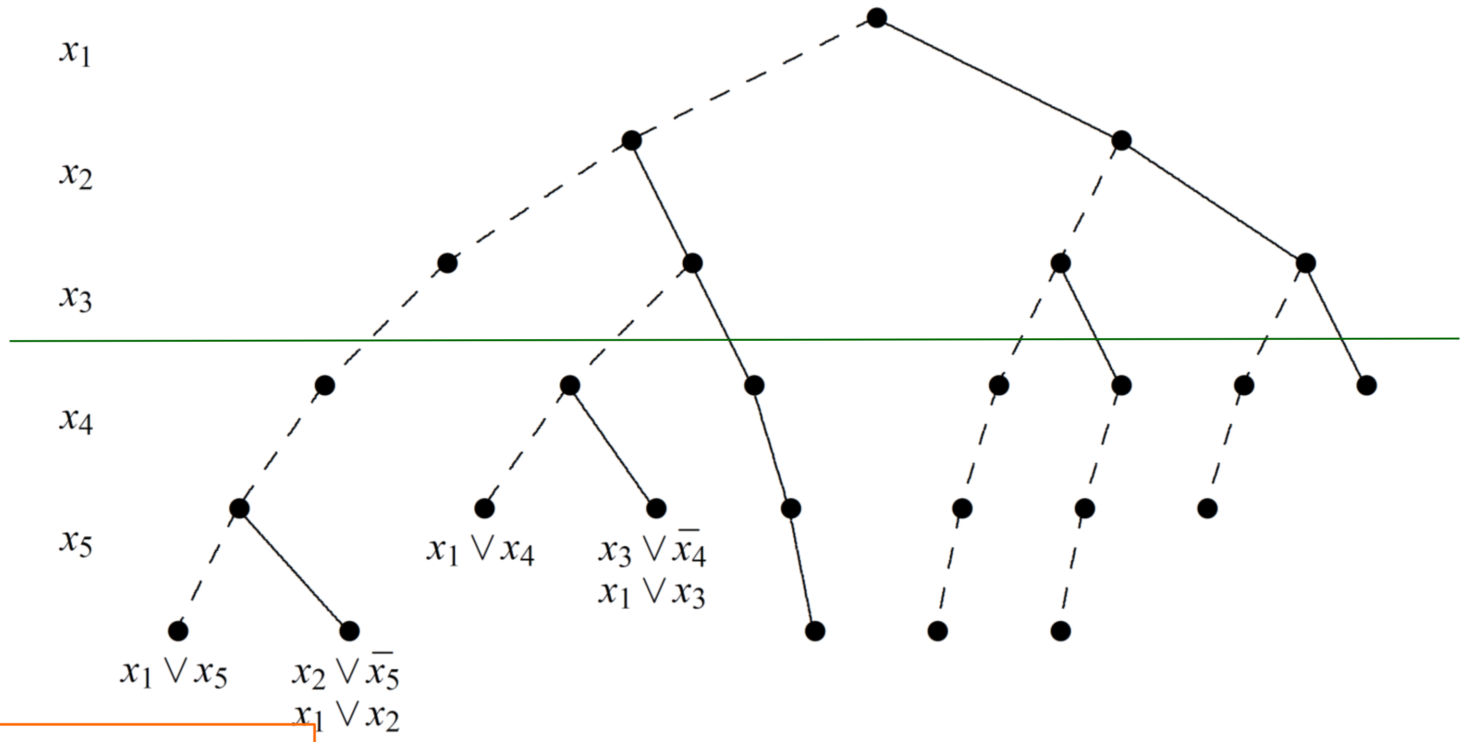
$x_5 \vee \bar{x}_6$

Continue until **master** is infeasible.
Black Benders cuts describe projection.

JH (2000, 2012)

# Inference as Projection

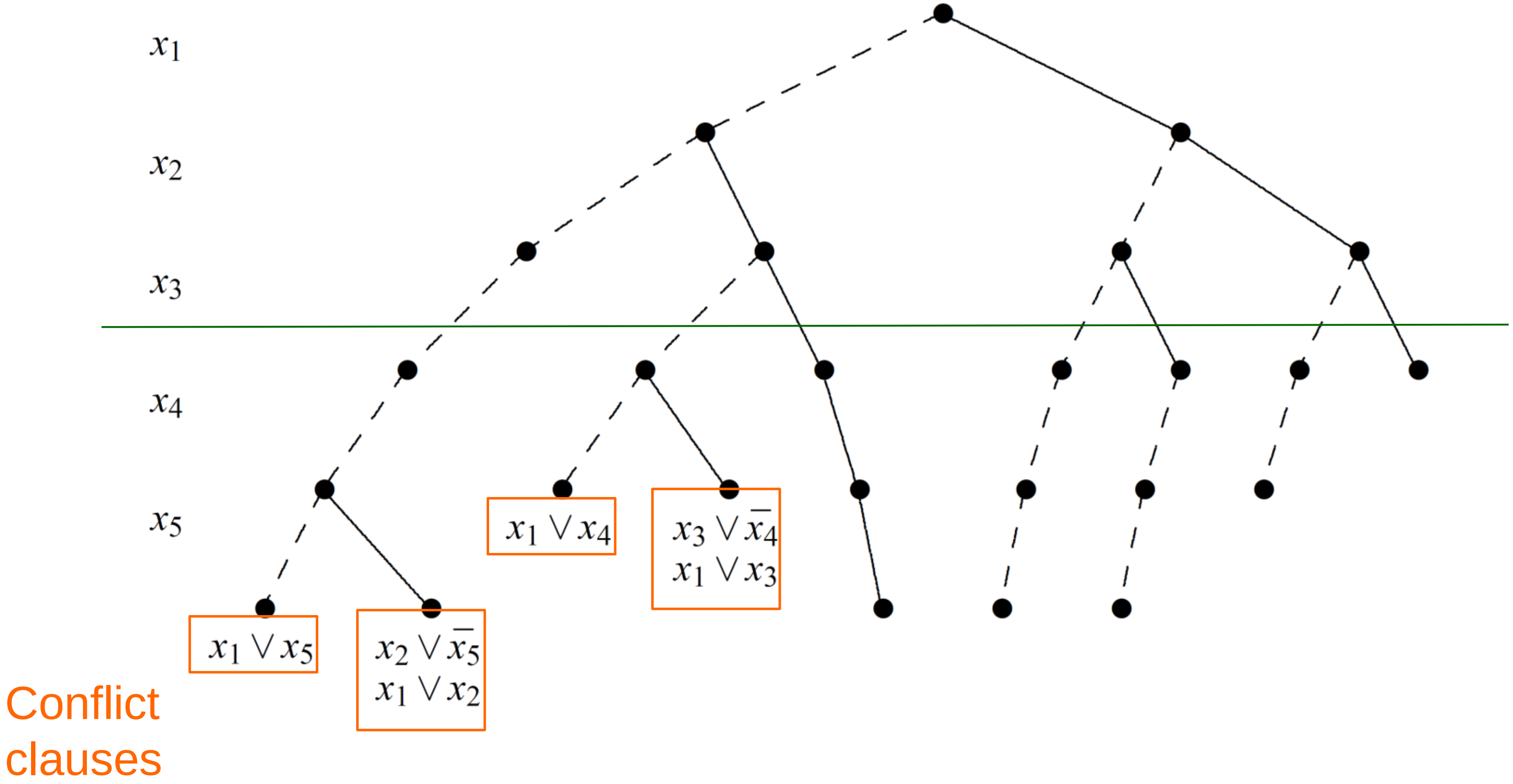
- Benders cuts = **conflict clauses** in a SAT algorithm!
  - Branch on $x_1$, $x_2$, $x_3$ first.




JH (2012, 2016)

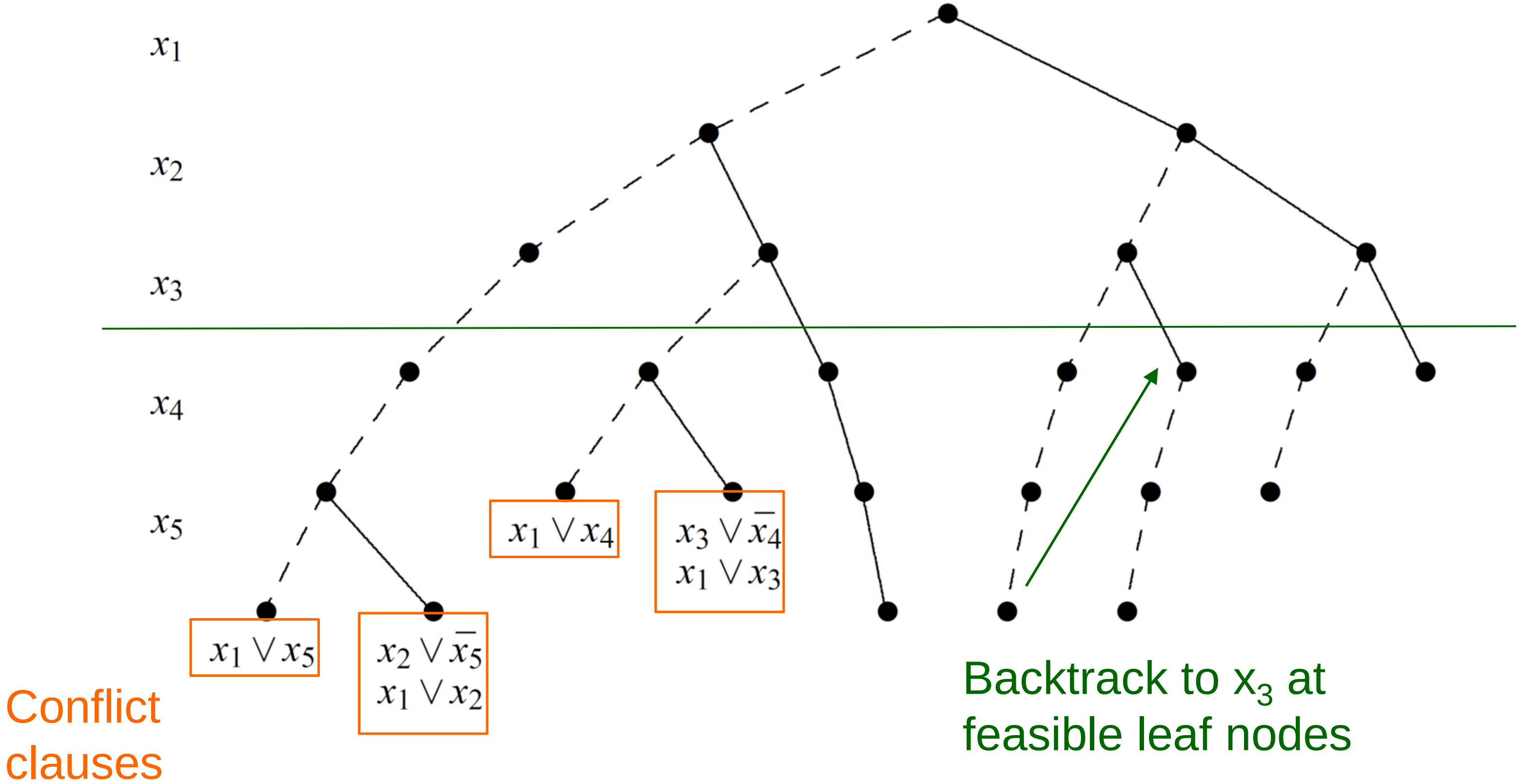
# Inference as Projection

- Benders cuts = **conflict clauses** in a SAT algorithm!
  - Branch on $x_1$, $x_2$, $x_3$ first.



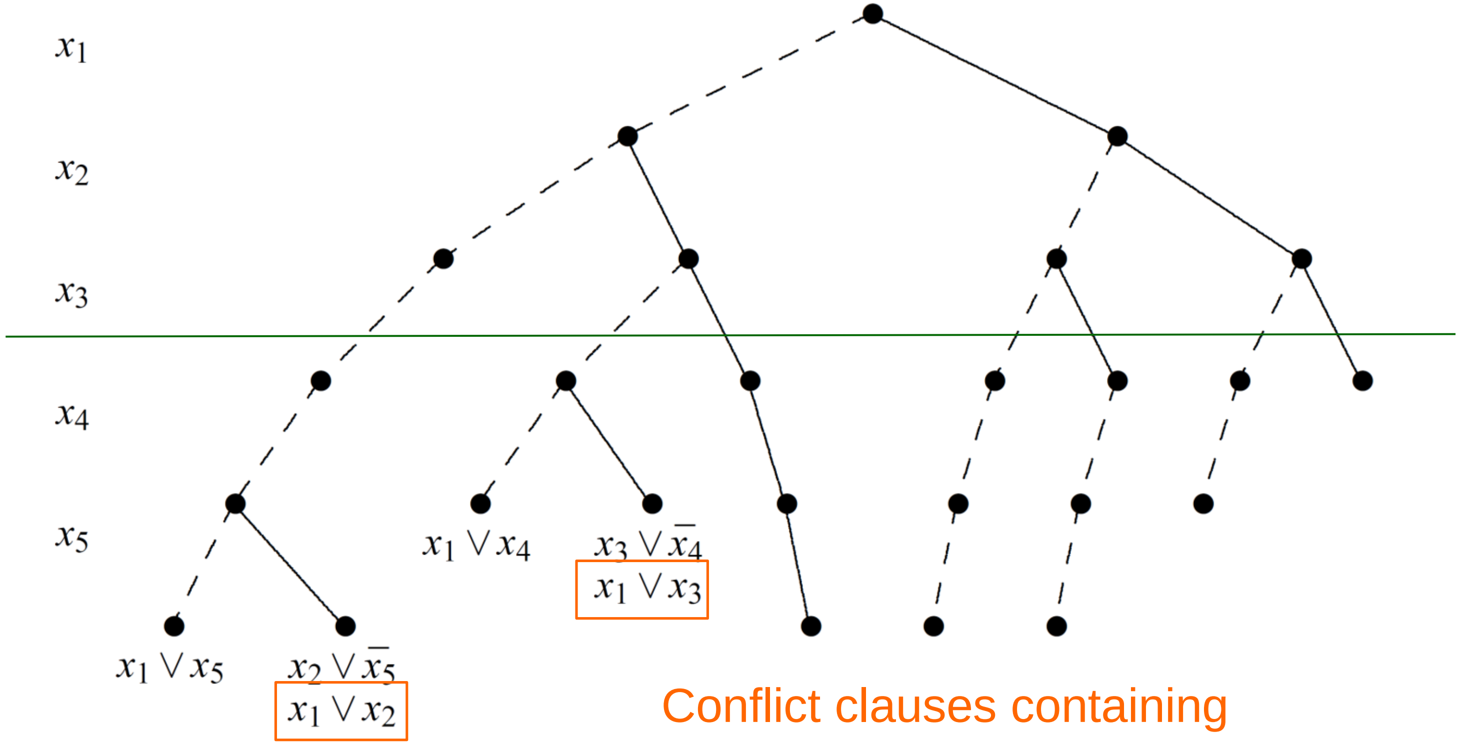
# Inference as Projection

- Benders cuts = **conflict clauses** in a SAT algorithm!
  - Branch on $x_1$, $x_2$, $x_3$ first.



# Inference as Projection

- Benders cuts = **conflict clauses** in a SAT algorithm!
  - Branch on $x_1$, $x_2$, $x_3$ first.




Conflict clauses containing
$x_1$, $x_2$, $x_3$ describe projection

# Accelerating Search

- Logic-based Benders can speed up search in several domains.
  - Several orders of magnitude relative to state of the art.
- Some applications:
  - Circuit verification
  - Chemical batch processing (BASF, etc.)
  - Steel production scheduling
  - Auto assembly line management (Peugeot-Citroën)
  - Automated guided vehicles in flexible manufacturing
  - Allocation and scheduling of multicore processors (IBM, Toshiba, Sony)
  - Facility location-allocation
  - Stochastic facility location and fleet management
  - Capacity and distance-constrained plant location

# Logic-Based Benders

- Some applications…
  - Transportation network design
  - Traffic diversion around blocked routes
  - Worker assignment in a queuing environment
  - Single- and multiple-machine allocation and scheduling
  - Permutation flow shop scheduling with time lags
  - Resource-constrained scheduling
  - Wireless local area network design
  - Service restoration in a network
  - Optimal control of dynamical systems
  - Sports scheduling

# First-Order Logic

- **Partial instantiation methods** for first-order logic can be viewed as Benders methods
  - The **master problem** is a SAT problem for the current formula *F*,
    - The solution of the master finds a **satisfier mapping** that makes one literal of each clause of *F* (the satisfier of the clause) true.

JH, Rago, Chandru, Shrivastava (2002)

# First-Order Logic

- **Partial instantiation methods** for first-order logic can be viewed as Benders methods
  - The **master problem** is a SAT problem for the current formula *F*,
    - The solution of the master finds a **satisfier mapping** that makes one literal of each clause of *F* (the satisfier of the clause) true.
  - The subproblem checks whether a satisfier mapping is **blocked**.
    - This means atoms assigned true and false can be **unified**.

JH, Rago, Chandru, Shrivastava (2002)

# First-Order Logic

- **Partial instantiation methods** for first-order logic can be viewed as Benders methods
  - The **master problem** is a SAT problem for the current formula *F*,
    - The solution of the master finds a **satisfier mapping** that makes one literal of each clause of *F* (the satisfier of the clause) true.
  - The subproblem checks whether a satisfier mapping is **blocked**.
    - This means atoms assigned true and false can be **unified**.
  - In case of blockage, more complete instantiations of the blocked clauses are added to *F* as **Benders cuts**.

JH, Rago, Chandru, Shrivastava (2002)

# First-Order Logic

- Resulting Benders decomposition:

**Master problem**

Current partially instantiated formula *F*.

Solve SAT problem for a satisfier mapping.

→ Satisfier mapping →

**Subproblem**

Check if the satisfier mapping is **blocked** by unifying atoms that receive different truth values.

The **dual** solution is the most general unifier.

Use **same unifier** to create **Benders cuts**: fuller instantiations of the relevant clauses.

← More fully instantiated clauses to append to *F* ←

# First-Order Logic

Consider the formula $F = \forall x C_1 \wedge \forall y C_2$

where $C_1 = P(a, x) \vee Q(a) \vee \neg R(x)$ $C_2 = \neg Q(y) \vee \neg P(y, b)$

# First-Order Logic

Consider the formula $F = \forall x C_1 \wedge \forall y C_2$

**True** **False**

where $C_1 = P(a,x) \vee Q(a) \vee \neg R(x)$ $C_2 = \neg Q(y) \vee \neg P(y,b)$

Solution of master problem yields satisfiers shown.

# First-Order Logic

Consider the formula $F = \forall x C_1 \wedge \forall y C_2$

**True** **False**

where $C_1 = P(a,x) \vee Q(a) \vee \neg R(x)$ $C_2 = \neg Q(y) \vee \neg P(y,b)$

Solution of master problem yields satisfiers shown.

The satisfier mapping is **blocked** because the atoms $P(a,x)$ and $P(y,b)$ can be unified.

# First-Order Logic

Consider the formula $F = \forall x C_1 \wedge \forall y C_2$

**True** **False**

where $C_1 = P(a,x) \vee Q(a) \vee \neg R(x)$ $C_2 = \neg Q(y) \vee \neg P(y,b)$

Solution of master problem yields satisfiers shown.

The satisfier mapping is **blocked** because the atoms $P(a,x)$ and $P(y,b)$ can be unified.

Generate **Benders cuts** by applying the most general unifier of the atoms to the clauses containing them, and adding the result to *F*. Now,

$$F = \forall x C_1 \wedge \forall y C_2 \wedge C_3 \wedge C_4$$

where $C_3 = P(a,b) \vee Q(a) \vee \neg R(b)$ $C_4 = \neg Q(y) \vee \neg P(y,b)$

# First-Order Logic

Consider the formula $F = \forall x C_1 \wedge \forall y C_2$

**True** **False**

where $C_1 = P(a,x) \vee Q(a) \vee \neg R(x)$ $C_2 = \neg Q(y) \vee \neg P(y,b)$

Solution of master problem yields satisfiers shown.

The satisfier mapping is **blocked** because the atoms $P(a,x)$ and $P(y,b)$ can be unified.

Generate **Benders cuts** by applying the most general unifier of the atoms to the clauses containing them, and adding the result to *F*. Now,

$$F = \forall x C_1 \wedge \forall y C_2 \wedge C_3 \wedge C_4$$

where $C_3 = P(a,b) \vee Q(a) \vee \neg R(b)$ $C_4 = \neg Q(y) \vee \neg P(y,b)$

Solution of the new master problem yields a satisfier mapping that is **not blocked** in the subproblem, and the procedure terminates with satisfiability.

# First-Order Logic

- We can accommodate full first-order logic with functions
  - If we replace **blocked** with ***M*-blocked**
    - Meaning that the satisfier mapping is blocked within a **nesting depth** of *M.*
  - The procedure always **terminates** if *F* is unsatisfiable.
    - It may not terminate if *F* is satisfiable, since first-order logic is **semidecidable**.
    - The master problem has infinitely many variables, because the **Herbrand base is infinite**.


THE END
IS NEAR
HERE