Consistency for Mixed-integer Programming

Danial Davarnia
Iowa State University

John Hooker
Carnegie Mellon University

INFORMS 2018

Consistency

- **Consistency** is a core concept of constraint programming.
  - Roughly speaking, **consistent** = partial assignments that violate no constraint are consistent with the constraint set.
    - They occur in some feasible solution.
  - Consistency $\Rightarrow$ **less backtracking**
    - Sometimes no backtracking, depending on the type of consistency.

Consistency

- 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

- 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.
  - Goal: adapt consistency concepts to **MIP**
    - This can lead to **new methods** to reduce backtracking.
    - Can also help **explain behavior of cuts**.
    - Requires us to **bridge two thought systems**.

Projection

- Define consistency in terms of **projection**.
  - The **projection** of constraint set *S* onto *J* is

$$D(S)|_J = \{x_J \mid x \in S\}$$

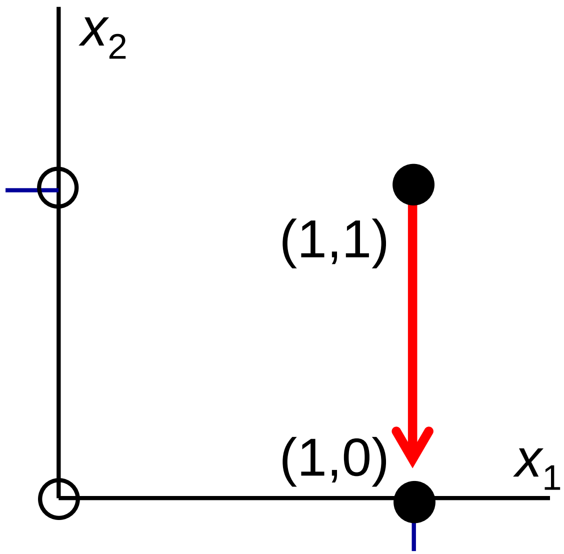
Set of tuples $(x_1,\ldots,x_n)$ satisfying *S*

Subset of $\{x_1,\ldots,x_n\}$

Tuple of x_j s for $x_j \in J$

Projection

Example



Projection of *D*(*S*) onto {x_1} is

$$D(S)|_{\{x_1\}} = \{1\}$$

Constraint set *S*

$$x_1 + x_2 \geq 1$$
$$x_1 - x_2 \geq 0$$
$$x_1, x_2 \in \{0, 1\}$$

Set *D*(*S*)

$$\{(1,0),(1,1)\}$$

Domain Consistency

- This is the workhorse of CP.
  - Constraint set *S* is **domain consistent** if

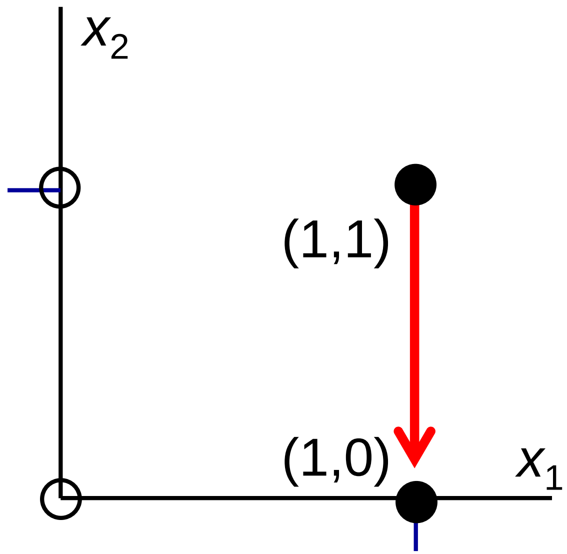
$$D_j = D(S)|_{\{j\}}, \text{ all } j$$

Domain of variable x_j

  - Every value in a variable's domain is consistent with the constraint set.

Domain Consistency

Example



Projection of $D(S)$ onto $\{x_1\}$ is

$$D(S)|_{\{x_1\}} = \{1\}$$

Constraint set S

$$x_1 + x_2 \geq 1$$
$$x_1 - x_2 \geq 0$$
$$x_1, x_2 \in \{0, 1\}$$

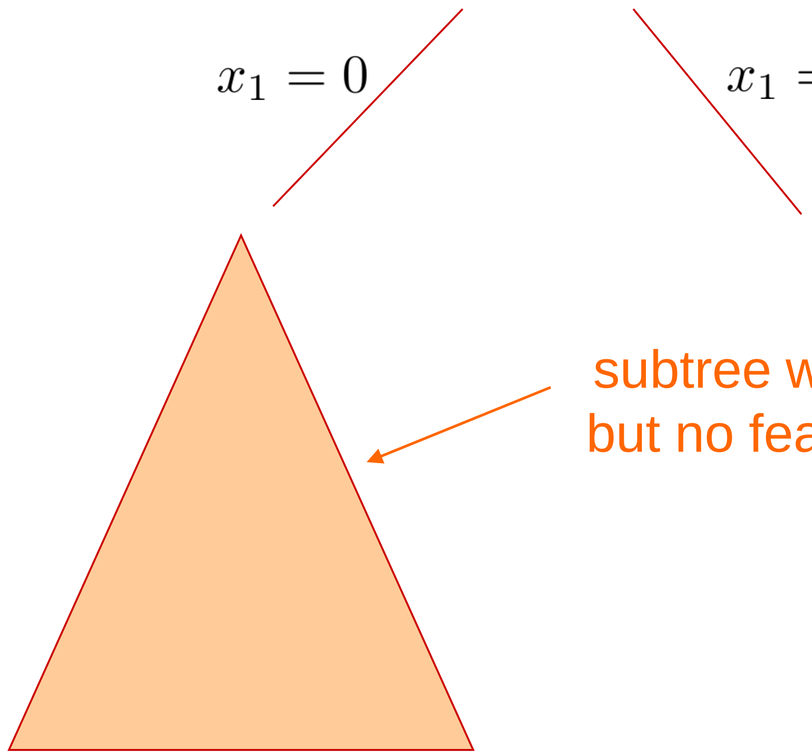
Not domain consistent because

$$D_1 = \{0, 1\} \neq \{1\} = D(S)|_{\{x_1\}}$$

Domain Consistency

Domain consistency can reduce branching.

$$
\begin{aligned}
& x_1 + x_{100} \geq 1 \\
& x_1 - x_{100} \geq 0 \\
& \text{other constraints} \\
& x_j \in \{0,1\}, \ \text{all } j
\end{aligned}
$$



subtree with 2^{99} nodes
but no feasible solution

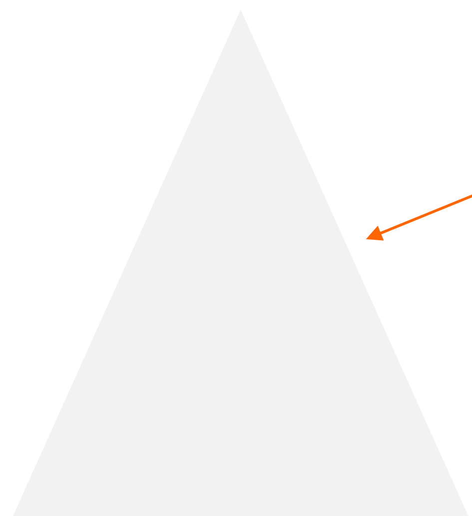
Domain Consistency

Domain consistency can reduce branching.

$$x_1 + x_{100} \geq 1$$
$$x_1 - x_{100} \geq 0$$
$$\text{other constraints}$$
$$\boxed{x_1 \in \{0\}},\ x_j \in \{0,1\},\ j > 1$$

By achieving domain consistency, we avoid searching 2^{99} nodes.

$$x_1 = 1$$



subtree with 2^{99} nodes but no feasible solution

Domain Consistency

- There is **no backtracking** if we achieve domain consistency **at every node** of the search tree.
  - Since this is hard, CP generally achieves domain consistency for **individual constraints**.
    - Or approximates domain consistency.

Full Consistency

- Strongest form of consistency:
  - Constraint set S is **consistent** if

$$D_J(S) = D(S)|_J, \text{ all } J \subseteq N$$

$\{x_1, \ldots, x_n\}$

Set of satisfying assignments to x_J

Satisfying = **violates no constraints** in S

Or: every inconsistent partial assignment is **explicitly ruled out** by some constraint.

A partial assignment can **violate** a constraint only if it **assigns values to all the variables** in the constraint.

We assume S contains all domain constraints $x_j \in D_j$

Full Consistency

Example

Constraint set *S*

$$x_1 + x_2 \geq 1$$
$$x_1 - x_2 \geq 0$$
$$x_1, x_2 \in \{0, 1\}$$

Not consistent because

$$D_{\{x_1\}}(S) = \{0, 1\} \neq \{1\} = D(S)|_{\{x_1\}}$$

The partial assignment $x_1 = 0$ is **inconsistent** but satisfies *S*: no constraint explicitly **rules it out**.

In fact, the partial assignment fails to fix all the variables in any constraint and so must satisfy *S*.

k-consistency

- Weaker type of consistency that can avoid backtracking if it is achieved **at the root node only**:
  - Constraint set *S* is ***k*-consistent** if

$$D_J(S) = D_{J \cup \{x_j\}}(S)|_J,$$
$$\text{all } J \subseteq N \text{ with } |J| = k-1, \text{ all } x_j \in N \setminus J$$

Or: every satisfying partial assignment to *k* – 1 variables can be extended to any *k*-th variable and still satisfy *S*.

k-consistency

Example

$$
\begin{aligned}
x_1 + x_2 \qquad\quad + x_4 &\geq 1 \\
x_1 - x_2 + x_3 \qquad &\geq 0 \\
x_1 \qquad\qquad\quad - x_4 &\geq 0 \\
x_j \in \{0, 1\} &
\end{aligned}
$$

- 1-consistent: trivial

k-consistency

Example

$$\begin{aligned} x_1 + x_2 \qquad\quad + x_4 &\geq 1 \\ x_1 - x_2 + x_3 \qquad &\geq 0 \\ x_1 \qquad\qquad\qquad - x_4 &\geq 0 \\ x_j \in \{0, 1\} & \end{aligned}$$

- 1-consistent: trivial
- 2-consistent: need only check x_1, x_4

k-consistency

Example

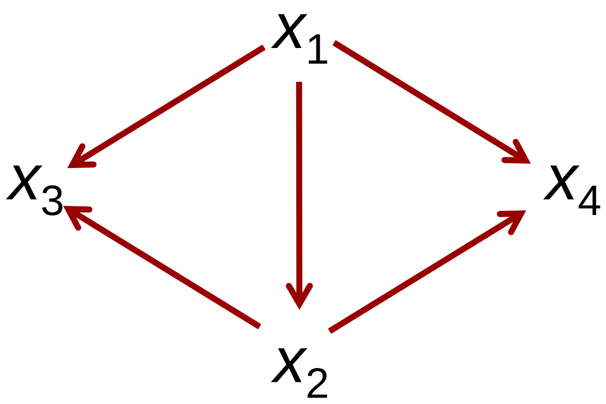
$$\begin{aligned} x_1 + x_2 \quad\quad\;\; + x_4 &\geq 1 \\ x_1 - x_2 + x_3 \quad\quad\;\; &\geq 0 \\ x_1 \quad\quad\quad\quad\;\; - x_4 &\geq 0 \\ x_j \in \{0, 1\} \end{aligned}$$

- 1-consistent: trivial
- 2-consistent: need only check x_1, x_4
- not 3-consistent:
  $(x_1,x_2) = (0,0)$ cannot be extended to $(x_1,x_2,x_4) = (0,0,?)$

k-consistency

- Dependency graph
  - Variables are connected by edges when they occur in a common constraint.
  - Also call primal graph.

$$\begin{aligned} x_1 + x_2 \quad\quad\; + x_4 &\geq 1 \\ x_1 - x_2 + x_3 \quad\quad\; &\geq 0 \\ x_1 \quad\quad\quad\quad\;\; - x_4 &\geq 0 \\ x_j \in \{0,1\} & \end{aligned}$$

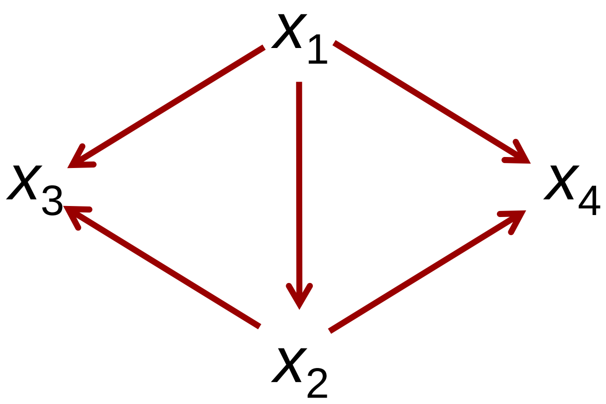


Dependency graph for ordering 1,2,3,4

k-consistency

- Dependency graph
  - Variables are connected by edges when they occur in a common constraint.
  - Also call primal graph.

$$
\begin{aligned}
x_1 + x_2 + x_4 &\geq 1 \\
x_1 - x_2 + x_3 &\geq 0 \\
x_1 - x_4 &\geq 0 \\
x_j \in \{0, 1\} &
\end{aligned}
$$



Dependency graph for ordering 1,2,3,4

Width of the graph is the maximum in-degree (here, 2)

k-consistency

- A constraint set is **strongly *k*-consistent** if it is *i*-consistent for $i = 1,\ldots,k$.

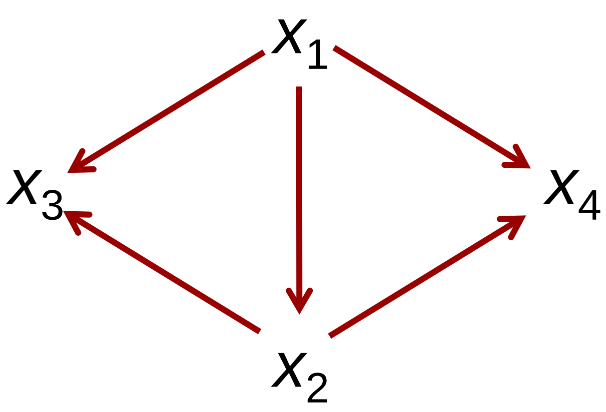
Theorem (Freuder). If a feasible problem is **strongly *k*-consistent**, and the **width** of its dependency graph is **less than *k*** with respect to some ordering of the variables, then branching in that order **avoids backtracking**.

k-consistency

- The example doesn't meet the conditions of the theorem.
  - Width = 2, not strongly 3-consistent.
  - Backtracking is possible, and it occurs when we set

    $(x_1,x_2,x_3,x_4) = (0,0,0,?)$
    - A feasible solution is $(x_1,x_2,x_3,x_4) = (1,0,0,0)$.

$$
\begin{aligned}
x_1 + x_2 \qquad\quad + x_4 &\geq 1 \\
x_1 - x_2 + x_3 \qquad\quad &\geq 0 \\
x_1 \qquad\qquad\quad - x_4 &\geq 0 \\
x_j \in \{0,1\} &
\end{aligned}
$$



Width = 2

k-consistency

- Suppose we add a constraint:
  - This is strongly 3-consistent.
    - New constraint rules out the only partial solution that couldn't be extended: $(x_1,x_2) = (0,0)$
  - Now it meets the conditions of the theorem.
    - No backtracking occurs.
    - For example, $(x_1,x_2,x_3,x_4) = (0,1,1,0)$.

$$
\begin{array}{llll}
x_1 + x_2 & + x_4 & \geq 1 & (a) \\
x_1 - x_2 + x_3 & & \geq 0 & (b) \\
x_1 & - x_4 & \geq 0 & (c) \\
\hline
\multicolumn{1}{|l}{x_1 + x_2} & & \geq 1 & \multicolumn{1}{l|}{(d)} \\
\hline
x_j \in \{0,1\} & & &
\end{array}
$$

k-consistency

- Two interpretations of the new constraint
  - Rank 1 Chvátal cut
    - Cuts off part of LP relaxation
  - Resolvent of (a) and (c)
    - Cuts off an inconsistent partial assignment.
    - In this case, achieves strong 3-consistency.

$$
\begin{array}{ll}
x_1 + x_2 \phantom{{}- x_3} + x_4 \geq 1 & (a) \\
x_1 - x_2 + x_3 \phantom{{}+ x_4} \geq 0 & (b) \\
x_1 \phantom{{}+ x_2 + x_3} - x_4 \geq 0 & (c) \\
\boxed{x_1 + x_2 \phantom{{}+ x_3 + x_4} \geq 1} & \boxed{(d)} \\
x_j \in \{0, 1\} &
\end{array}
$$

Resolution:

$$
\begin{array}{ll}
x_1 \vee x_2 \vee x_4 & (a) \\
x_1 \vee \neg x_4 & (c) \\
\hline
x_1 \vee x_2 & (d)
\end{array}
$$

k-consistency

- Problem: *k*-consistency is very hard to achieve.
- Possible solution: Use **LP-consistency**
  - A new form of consistency that takes advantage of the LP relaxation.
    - Intermediate concept between a **satisfying** partial assignment and a **consistent** partial assignment.
  - Even a weak form of LP-consistency **avoids backtracking**
    - It is much easier to achieve than *k*-consistency.
    - Yields a different kind of cut.

LP-consistency

- LP consistency applies to IP constraint sets.
  - For simplicity, assume variables are 0-1
- Definitions
  - Let $S = \{Ax \geq b,\ x \in \mathbb{Z}^n\}$
  - Let the LP relaxation be $S_{\mathrm{LP}} = \{Ax \geq b,\ x \in \mathbb{R}^n\}$
  - We assume $Ax \geq b$ contains $0 \leq x_j \leq 1,\ \text{all } j$

LP-consistency

- LP-consistent partial assignment
  - 0-1 partial assignment $x_J = v_J$ is **LP-consistent** with *S* if $S_{\mathrm{LP}} \cup \{x_J = v_J\}$ is feasible.
    - Unlike the traditional concept of a consistent assignment, this is **easily checked** by solving an LP.
    - A consistent partial assignment is necessarily LP-consistent.

LP-consistency

- LP-consistent partial assignment
  - 0-1 partial assignment $x_J = v_J$ is **LP-consistent** with *S* if $S_{\mathrm{LP}} \cup \{x_J = v_J\}$ is feasible.
    - Unlike the traditional concept of a consistent assignment, this is **easily checked** by solving an LP.
    - A consistent partial assignment is necessarily LP-consistent.
- LP-consistency
  - A 0-1 constraint set *S* is **LP-consistent** if every LP-consistent partial assignment is consistent:

$$L_J(S) = D(S)|_J$$

Set of 0-1 assignments to x_J
that are LP-consistent with *S*

LP-consistency

- Relationship with convex hull description

Theorem. A feasible 0-1 constraint set S is LP-consistent if S_{LP} describes the convex hull of S.

- The converse does not hold, but we will see that even a weak version of LP-consistency allows one to avoid backtracking.

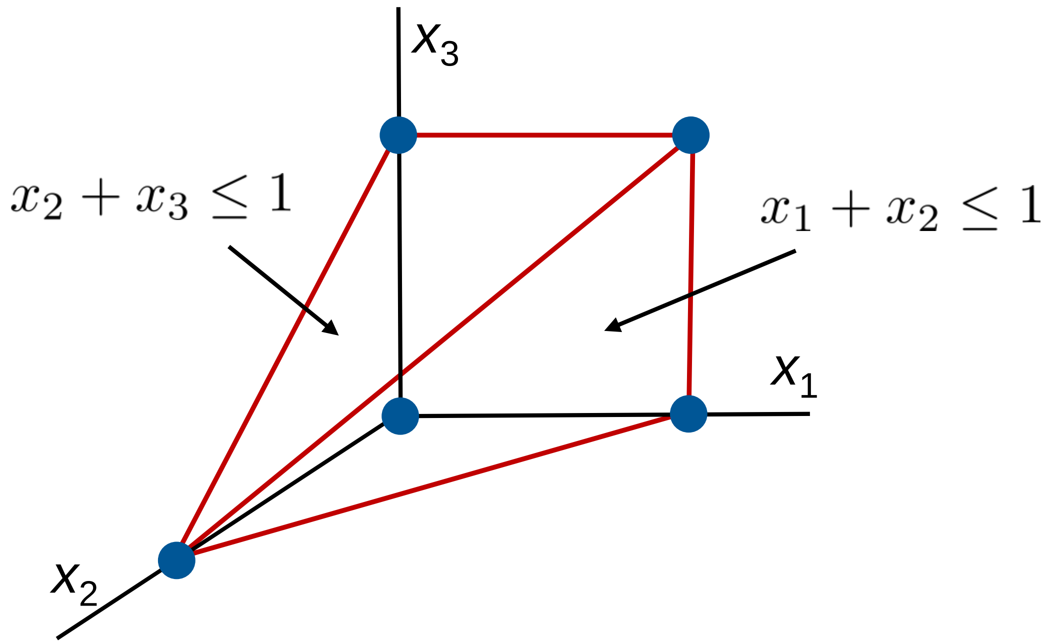
LP-consistency

Example

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

S_{LP} describes convex hull of S.

So S is LP-consistent.



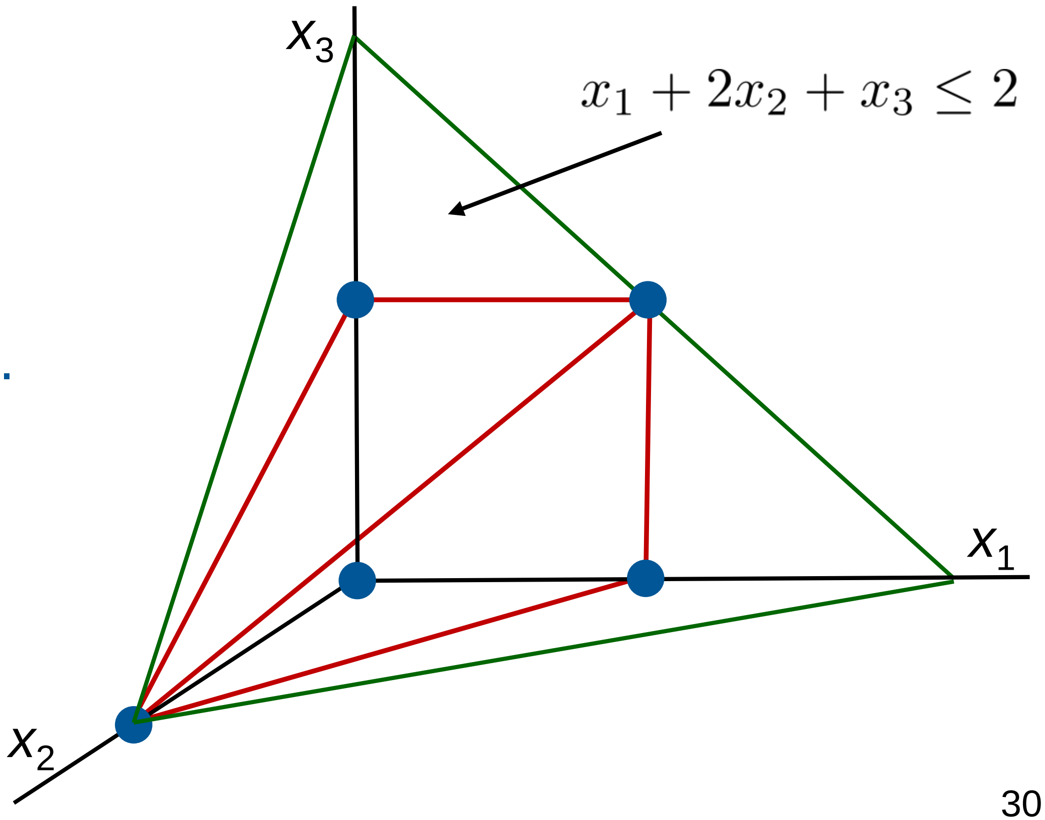
LP-consistency

Example

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

S'_{LP} does not describe convex hull of S.

But S' is LP-consistent.



LP-consistency

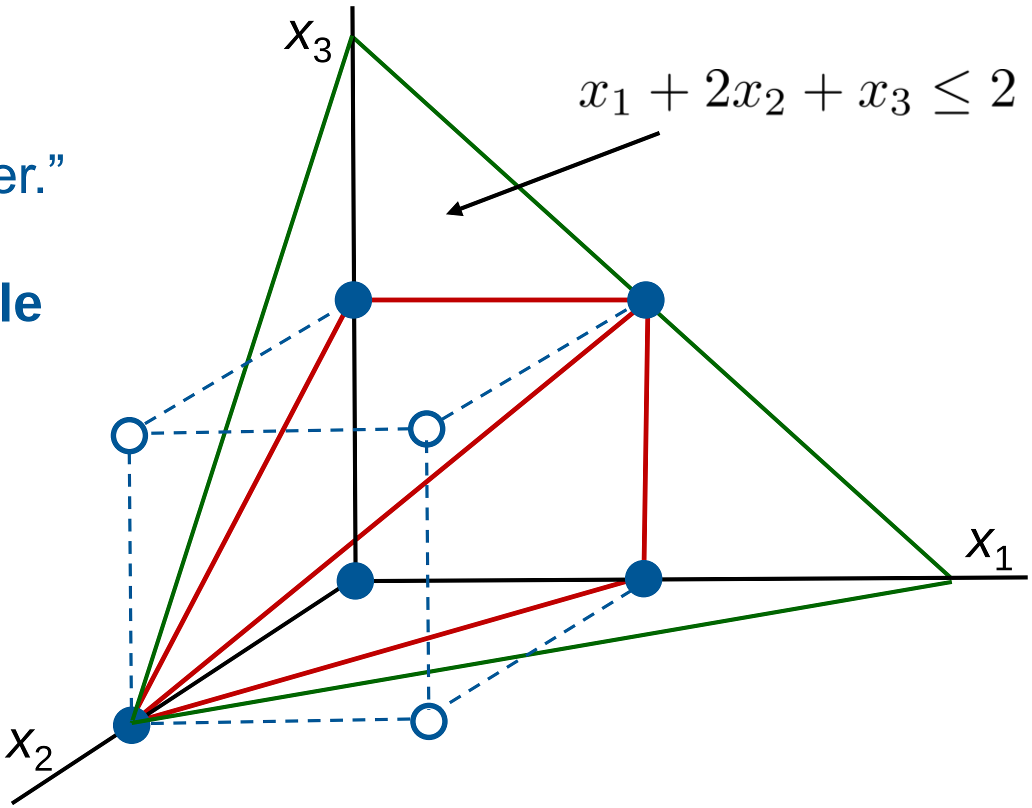
Example

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

This inequality is the **sum** of the 2 facet-defining inequalities and so is "weaker."

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

LP-consistency leads to inequalities that cut off more infeasible 0-1 points & so reduce backtracking.



LP-consistency

- Relationship with **Chvátal closure**
  - Let S_C = set of **clausal inequalities** in Chvátal closure of S.

Theorem. If S is LP-consistent, a 0-1 partial assignment is consistent with S if and only if it satisfies S_C.

  - Achieving LP-consistency has same power as deriving all rank 1 clausal Chvátal cuts.

$$x_1 + (1 - x_2) + x_3 \geq 1$$

is **clausal** because it represents the logical clause

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

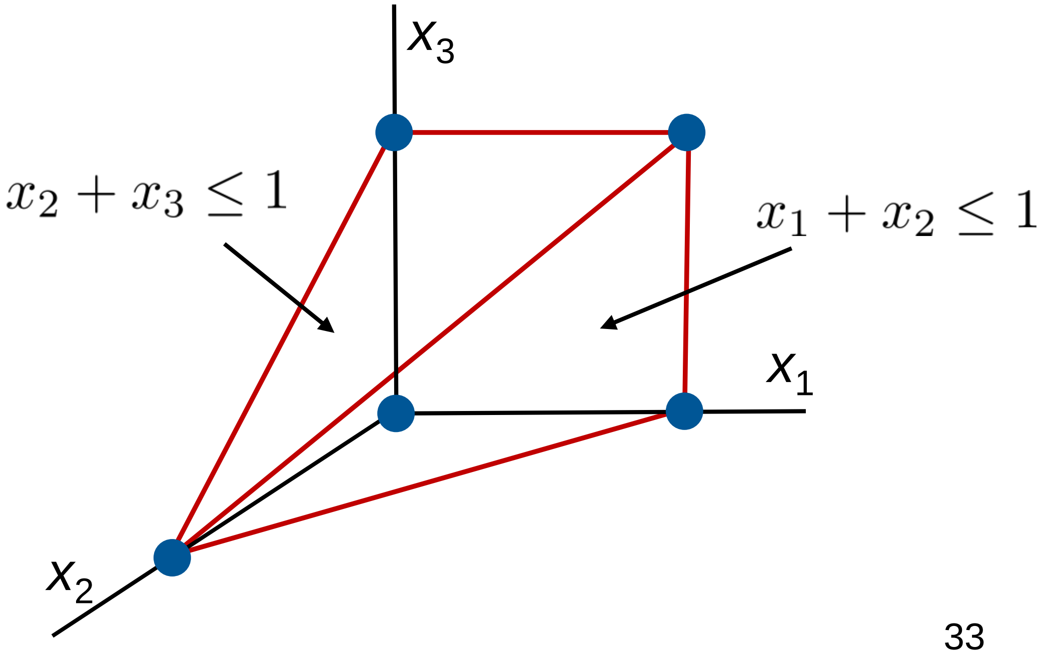
LP-consistency

Example

$$S' = \Big\{ x_1 + 2x_2 + x_3 \leq 2,\ x_j \in \{0,1\} \Big\}$$
$$S'_{\mathrm{C}} = \Big\{ (1-x_1) + (1-x_2) \geq 1,\ (1-x_2) + (1-x_3) \geq 1 \Big\}$$

In this case,
S'_C consists of
the 2 facet-defining
inequalities.

They identify precisely
$(x_1,x_2) = (1,1)$
$(x_2,x_3) = (1,1)$
as the LP-inconsistent
partial assignments.



LP k-consistency

- LP k-consistency is enough to avoid backtracking.
  - Fix the variable ordering, and let $J_k = \{x_1,...,x_k\}$.
  - S is **LP *k*-consistent** if $L_{J_{k-1}}(S) = L_{J_k}(S)|_{J_{k-1}}$
    - Every 0-1 assignment to $(x_1,\ldots,x_{k-1})$ that is LP-consistent with S can be extended to an assignment to $(x_1,\ldots,x_k)$ that is LP-consistent with S.

LP k-consistency

- LP k-consistency is enough to avoid backtracking.
  - Fix the variable ordering, and let $J_k = \{x_1,\ldots,x_k\}$.
  - S is **LP *k*-consistent** if $L_{J_{k-1}}(S) = L_{J_k}(S)|_{J_{k-1}}$
    - Every 0-1 assignment to $(x_1,\ldots,x_{k-1})$ that is LP-consistent with S can be extended to an assignment to $(x_1,\ldots,x_k)$ that is LP-consistent with S.

Theorem. If S is LP k-consistent for $k = 1,\ldots,n$ and we branch in the order $x_1,\ldots,x_n$, we can avoid backtracking by solving at most 2 LPs before each variable assignment.

If we have fixed $(x_1,\ldots,x_{k-1}) = (v_1,\ldots,v_{k-1})$, solve the LP

$$S_{\mathrm{LP}} \cup \big\{(x_1,\ldots,x_{k-1},x_k) = (v_1,\ldots,v_{k-1},v_k)\big\}$$

for $v_k = 0,1$. If feasible for v_k, set $x_k = v_k$.

LP *k*-consistency

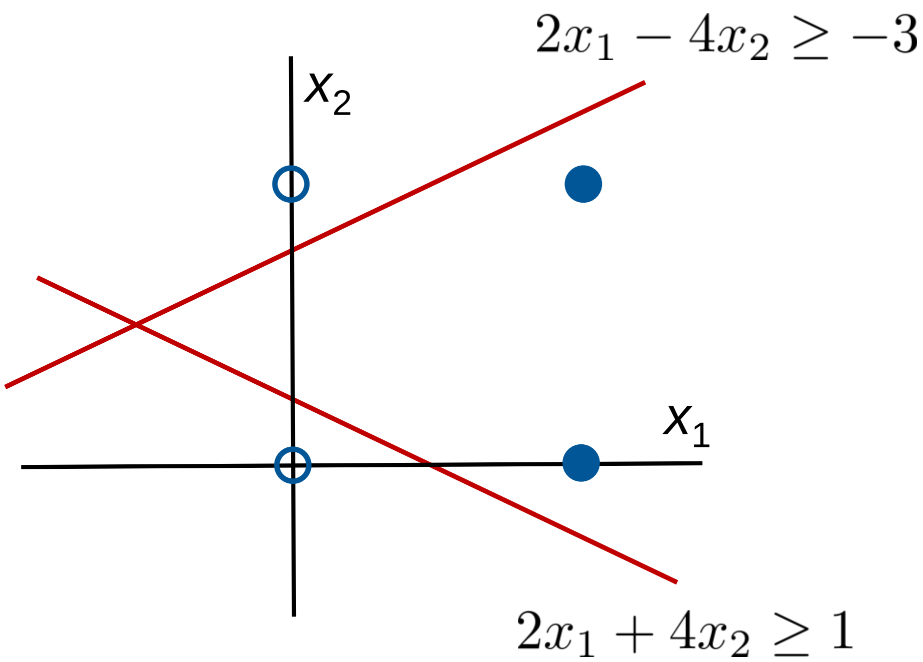
Example

$$S = \left\{ 2x_1 + 4x_2 \geq -1,\ 2x_1 - 4x_2 \geq -3,\ x_j \in \{0,1\} \right\}$$

$x_1 = 0$ is LP-consistent with S, but neither $(x_1,x_2) = (0,0)$ nor $(x_1,x_2) = (0,1)$ is LP-consistent with S.

So S is **not** LP 2-consistent.

Setting $x_1 = 0$ will require backtracking.



LP *k*-consistency

Example

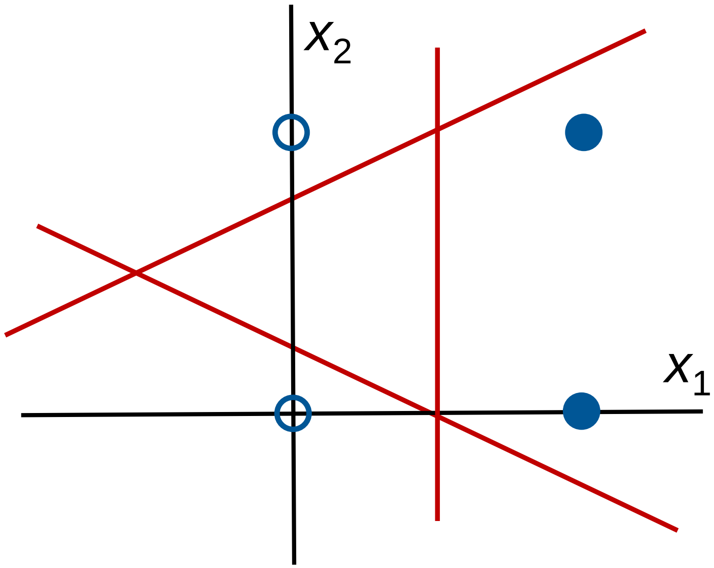
$$S = \left\{ 2x_1 + 4x_2 \geq -1,\ 2x_1 - 4x_2 \geq -3,\ x_j \in \{0,1\} \right\}$$

One step of RLT
(or lift-and-project)
yields new constraint

$$x_1 \geq \frac{1}{2}$$

Constraint set is now
LP 2-consistent.

No backtracking.



LP *k*-consistency

- We can achieve LP *k*-consistency at any level *k* of the branching tree with 1 step of RLT or lift-and-project.
  - That is, lift into 1 higher dimension and project.
  - This allows us to avoid backtracking.

LP *k*-consistency

- We can achieve LP *k*-consistency at any level *k* of the branching tree with 1 step of RLT or lift-and-project.
  - That is, lift into 1 higher dimension and project.
  - This allows us to avoid backtracking.
- This gets computationally hard as *k* increases.
  - So achieve LP *k*-consistency at **top few levels** of the tree.
    - This yields **sparse** cuts.
  - Lift into several higher dimensions if desired, rather than 1.
    - To reduce future backtracking.

LP *k*-consistency

- Resulting cuts are **different** than in standard branch and cut
  - They contain variables that are **already fixed**
    - …rather than variables not yet fixed.
  - They have a different purpose.
    - They are intended to cut off **inconsistent 0-1 partial assignments** rather than tighten LP relaxation.
    - Although they can do both, just as traditional cuts can do both.