Consistency for 0-1 Programming

Danial Davarnia
Iowa State University

John Hooker
Carnegie Mellon University

CPAIOR 2019

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.
  - Even though it is **closely related** to the amount of backtracking…
  - …and even though **valid inequalities** (cutting planes) can reduce backtracking by achieving a **greater degree of consistency**
    - …as well as by tightening a relaxation.

Consistency

- Goal: Adapt consistency to **integer programming**.
  - This can lead to **new methods** to reduce backtracking.
  - Can also help to **explain behavior of cutting planes.**
  - Requires us to **bridge two thought systems**.

Consistency

- Goal: Adapt consistency to **integer programming**.
  - This can lead to **new methods** to reduce backtracking.
  - Can also help to **explain behavior of cutting planes.**
  - Requires us to **bridge two thought systems**.
  - **Caveat:** We don’t claim, at this point, that these ideas will improve IP solvers.
    - Although we have some interesting preliminary results.
    - Reminder: It took 20+ years to learn how to use simple Gomory cuts in an IP solver.

Consistency

- Define a **consistent partial assignment**.
  - A partial assignment $x_J = v_J$ is **consistent with constraint set *S*** if

    $$S \cup \{x_J = v_J\}$$

    is feasible.

    Tuple of v_j s for $j \in J$

  - Constraint set *S* is **consistent** if every partial assignment that **violates no constraint** in *S* is consistent with *S*.

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

Consistency

Example.

$$S = \begin{array}{l} x_1 + x_2 \qquad\quad + x_4 \geq 1 \\ x_1 - x_2 + x_3 \qquad \geq 0 \\ x_1 \qquad\qquad\qquad - x_4 \geq 0 \\ x_j \in \{0,1\} \end{array}$$

Feasible set:

$$\begin{array}{lll} (0,1,1,0) & (1,0,1,0) & (1,1,0,1) \\ (1,0,0,0) & (1,0,1,1) & (1,1,1,0) \\ (1,0,0,1) & (1,1,0,0) & (1,1,1,1) \end{array}$$

S is **not consistent** because $(x_1,x_2) = (0,0)$ violates no constraint in *S* but is inconsistent with *S*; that is,

$$S \cup \{(x_1, x_2) = (0,0)\}$$ is infeasible.

Consistency & 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*

Consistency & 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*

Let $D_J(S)$ be set of assignments $x_J = v_J$ that are consistent with *S*. Then *S* is consistent if and only if

$$D_J(S_J) = D(S)|_J, \text{ all } J \subseteq \{1,\ldots,n\}$$

Set of constraints in *S*
whose variables
belong to x_J

Consistency & Projection

Example.

$$S = \begin{array}{l} x_1 + x_2 \qquad\qquad + x_4 \geq 1 \\ x_1 - x_2 + x_3 \qquad\quad \geq 0 \\ x_1 \qquad\qquad\qquad\;\; - x_4 \geq 0 \\ x_j \in \{0,1\} \end{array}$$

$$D(S) = \begin{array}{ccc} (0,1,1,0) & (1,0,1,0) & (1,1,0,1) \\ (1,0,0,0) & (1,0,1,1) & (1,1,1,0) \\ (1,0,0,1) & (1,1,0,0) & (1,1,1,1) \end{array}$$

S is **not consistent** because

$$D_{\{1,2\}}(S_{\{1,2\}}) \neq D(S)|_{\{1,2\}}$$

(0,0)
(0,1)
(1,0)
(1,1)

$\varnothing$

(0,1)
(1,0)
(1,1)

Domain Consistency

S is **domain consistent** if and only if

Domain of x_j $\longrightarrow$ $$D_j = D(S)|_{\{j\}}, \text{ all } j \in \{1,\ldots,n\}$$

Example.

$$D(S) = \begin{matrix} (0,1,1,0) & (1,0,1,0) & (1,1,0,1) \\ (1,0,0,0) & (1,0,1,1) & (1,1,1,0) \\ (1,0,0,1) & (1,1,0,0) & (1,1,1,1) \end{matrix}$$

S is domain consistent because

$$\underbrace{D_j}_{\{0,1\}} = \underbrace{D(S)}_{\{0,1\}}|_{\{j\}}, \quad j \in \{1,2,3,4\}$$

Domain Consistency

- There is **no backtracking** if domain consistency is achieved at **every node** of the branching tree.

  – At level k, set x_k equal to any value in its domain.

    The corresponding subtree contains a feasible solution, and we can continue branching.

Domain Consistency

- There is **no backtracking** if the original constraint set is **fully consistent**.
  - At level k in the branching tree, where $(x_1,\ldots,x_{k-1}) = (v_1,\ldots,v_{k-1})$:

    if $(x_1,\ldots,x_k) = (v_1,\ldots,v_k)$ violates no constraint, then the subtree formed by setting $x_k = v_k$ contains a feasible solution, and we can continue branching.

k-consistency

- This is a **weaker** type of consistency that can also avoid backtracking.
  - We define *k*-consistency with respect to the **particular variable ordering** $x_1,\ldots,x_n$ (the intended branching order).
  - Constraint set *S* is ***k*-consistent** if

$$D_{J_{k-1}}(S_{J_{k-1}}) = D_{J_k}(S_{J_k})|_{J_{k-1}}$$

where $J_k = \{1,\ldots,k\}$

Or: any assignment to first *k* – 1 variables that violates no constraint can be **extended** to an assignment to first *k* variables that violates no constraint

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

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

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

k-consistency

Example

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

- 1-consistent: trivial
- 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

- Suppose we add a constraint:
  - This is 3-consistent.
    - New constraint rules out the only partial solution that couldn't be extended: $(x_1,x_2) = (0,0)$
  - Now *S* is *k*-consistent for *k* = 1,2,3.
    - No backtracking occurs.
    - For example, $(x_1,x_2,x_3,x_4) = (0,1,1,0)$.

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

k-consistency

- **Two interpretations** of the new constraint
  - Rank 1 Chvátal-Gomory cut
    - Cuts off part of LP relaxation
    - Namely, vertices $x = (\frac{1}{3}, \frac{1}{3}, 0, \frac{1}{3}), (\frac{1}{2}, 0, 0, \frac{1}{2})$
  - Resolvent of (a) and (c)
    - Cuts off an inconsistent partial assignment $(x_1, x_2) = (0, 0)$
    - In this case, achieves 3-consistency.

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

Resolution:

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

LP consistency

- Problem: consistency and *k*-consistency are very hard to achieve.
- Possible solution: Use **LP consistency** and **LP *k*-consistency**
    - LP = linear programming

LP consistency

- Problem: consistency and *k*-consistency are very hard to achieve.
- Possible solution: Use **LP consistency** and **LP *k*-consistency**
  - LP = linear programming
- Applies to integer programming 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 $S_{\text{LP}} = \{Ax \geq b,\ x \in \mathbb{R}^n\}$
  - We assume $Ax \geq b$ contains $0 \leq x_j \leq 1, \text{ all } j$

LP consistency

- Defining **LP consistency**
  - Recall that classical consistency is defined with respect to a **relaxation**:

$$D_J(S_J) = D(S)|_J$$

Relaxation of *S*

LP consistency

- Defining **LP consistency**
  - Recall that classical consistency is defined with respect to a **relaxation**:

$$D_J(S_J) = D(S)|_J$$

Relaxation of *S*

  - Rationale: consistency makes it **easy** to detect inconsistent partial assignments
    - An inconsistent partial assignment $x_J = v_J$ always violates the **relaxation** S_J.
    - $S_J \cup \{x_J = v_J\}$ is obviously infeasible.

LP consistency

- Defining **LP consistency**
  - Define LP consistency with respect to **LP relaxation**:

$$D_J(S_{\mathrm{LP}}) = D(S)|_J$$

Set of 0-1 assignments to x_J feasible for S_{LP}

LP relaxation of S

LP consistency

- Defining **LP consistency**
  - Define LP consistency with respect to **LP relaxation**:

$$D_J(S_{\mathrm{LP}}) = D(S)|_J$$

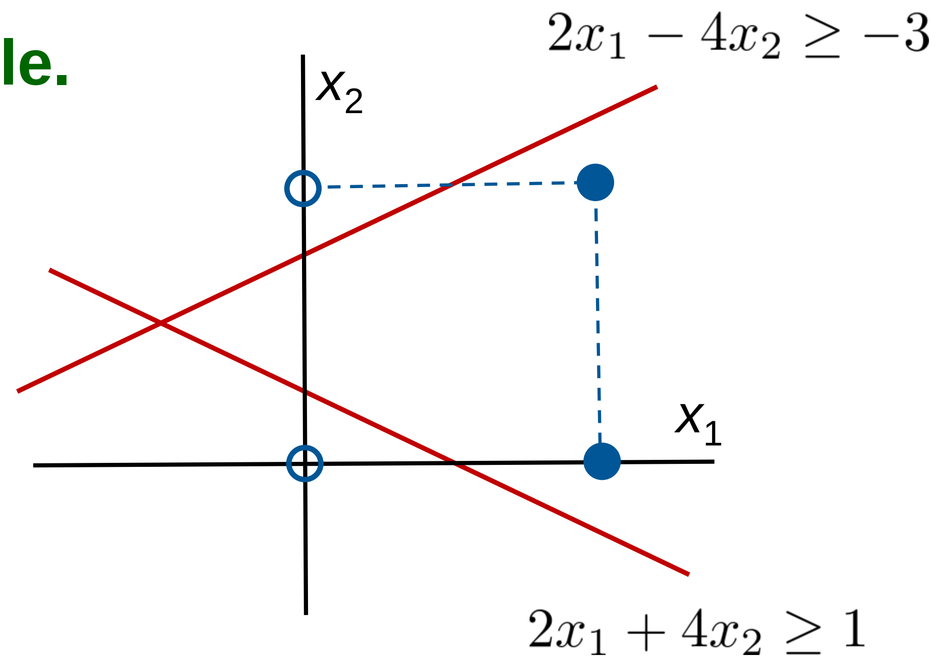
Set of 0-1 assignments to x_J feasible for S_{LP}

LP relaxation of S

  - Rationale: LP consistency makes it **easy** to detect inconsistent 0-1 partial assignments
    - An inconsistent 0-1 partial assignment $x_J = v_J$ always violates the **relaxation** S_{LP}.
    - Infeasibility of $S_{\mathrm{LP}} \cup \{x_J = v_J\}$ is easy to check.
    - It's an LP problem!

LP consistency

Example.



S is **not LP consistent** because the partial assignment $x_1 = 0$ is consistent with S_{LP} but not with *S*.

Both $(x_1, x_2) = (0, 0)$ and $(x_1, x_2) = (0, 1)$ violate *S*.

LP consistency

Theorem. A consistent 0-1 constraint set is LP consistent.

Relationship with integer hull

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

- The converse does not hold. An LP consistent model **need not define the integer hull**.
- LP consistency is not a concept of traditional polyhedral theory.

LP consistency

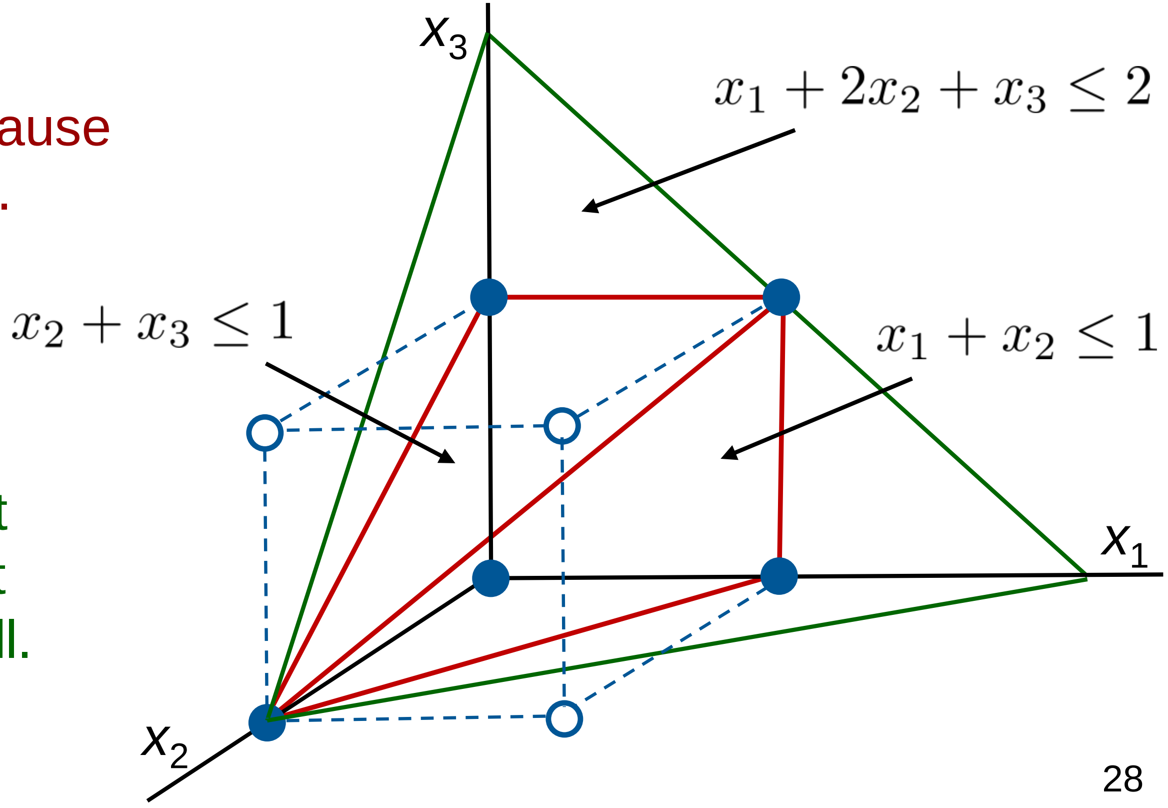
Example

$$S_1 = \left\{ x_1 + x_2 \leq 1,\ x_2 + x_3 \leq 1,\ x_j \in \{0,1\} \right\}$$

$$S_2 = \left\{ x_1 + 2x_2 + x_3 \leq 1,\ x_j \in \{0,1\} \right\}$$

S_1 is LP consistent because it describes integer hull.

S_2 is also LP consistent even though it does not describe the integer hull.



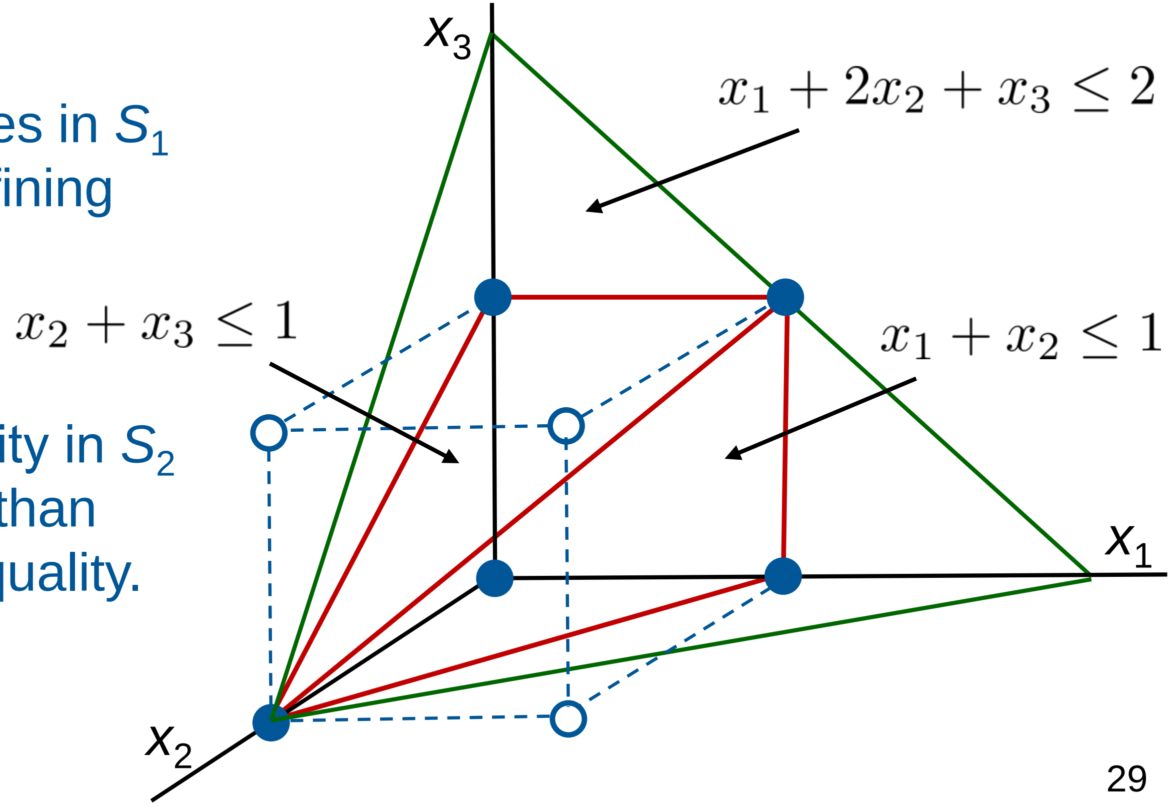
LP consistency

Example

$$S_1 = \left\{ x_1 + x_2 \leq 1,\ x_2 + x_3 \leq 1,\ x_j \in \{0,1\} \right\}$$
$$S_2 = \left\{ x_1 + 2x_2 + x_3 \leq 1,\ x_j \in \{0,1\} \right\}$$

Facet-defining inequalities in S_1 sum to the non-facet-defining inequality in S_2.

Yet the "weaker" inequality in S_2 cuts off more 0-1 points than either facet-defining inequality.



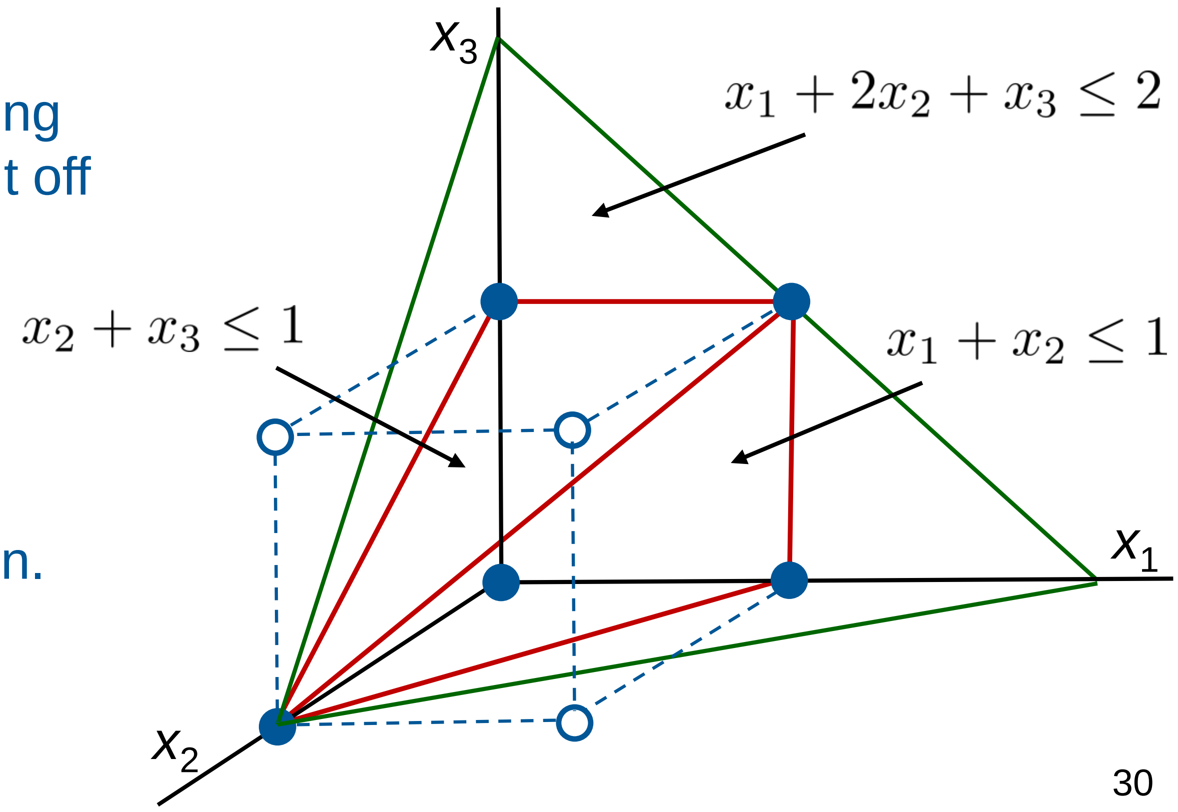
LP consistency

Example

$$S_1 = \left\{ x_1 + x_2 \le 1,\ x_2 + x_3 \le 1,\ x_j \in \{0,1\} \right\}$$
$$S_2 = \left\{ x_1 + 2x_2 + x_3 \le 1,\ x_j \in \{0,1\} \right\}$$

The purpose of achieving LP consistency is to cut off infeasible 0-1 (partial) assignments…

Not to cut off fractional vertices of LP relaxation.



LP consistency

Relationship with **cutting planes**

Definition: the inequality

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

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

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

Theorem. A 0-1 partial assignment is consistent with S_{LP} if and only if it violates no clausal rank 1 Chvátal-Gomory cut for S_{LP}.

LP consistency

Relationship with **cutting planes**

Definition: the inequality

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

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

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

Theorem. A 0-1 partial assignment is consistent with S_{LP} if and only if it violates no clausal rank 1 Chvátal-Gomory cut for S_{LP}.

Theorem. S is LP consistent if and only if all of its implied clausal inequalities are rank 1 C-G cuts for S_{LP}.

- Achieving LP consistency has same power as deriving all **clausal** rank 1 C-G cuts.

LP *k*-consistency

- **LP *k*-consistency** is a weaker form of LP consistency, and easier to achieve.

  – S is **LP *k*-consistent** if

$$D_{J_{k-1}}(S_{\text{LP}}) = D_{J_k}(S_{\text{LP}})\big|_{J_{k-1}}$$

where $J_k = \{1, \ldots, k\}$

Or: any 0-1 assignment to first k – 1 variables that is consistent with S_{LP} can be **extended** to an assignment to first k variables that is consistent with S_{LP}.

LP *k*-consistency

- There is **no backtracking** if the original constraint set is **LP *k*-consistent** for $k = 1,\ldots,n$.
  - …and we solve LPs along the way.

  – At level *k* in the branching tree, where we have fixed $(x_1,\ldots,x_{k-1}) = (v_1,\ldots,v_{k-1})$:

    If $S_{\mathrm{LP}} \cup \{(x_1,\ldots,x_k) = (v_1,\ldots,v_k)\}$ is a feasible LP, then the subtree formed by setting $x_k = v_k$ contains a feasible solution, and we can continue branching.

Achieving LP *k*-consistency

We used a **modified lift-and-project procedure**

Let $S = \left\{ Ax \geq b,\ x_j \in \{0,1\} \right\}$

where $Ax \geq b$ includes $0 \leq x_j \leq 1$

Generate the nonlinear system $\begin{aligned} &(Ax-b)x_k \geq 0 \\ &(Ax-b)(1-x_k) \geq 0 \end{aligned}$

Linearize the system by replacing each x_k^2 with x_k and each $x_i x_k$ with y_{ik}

Theorem. Adding this system to S_{LP} yields an LP *k*-consistent constraint set.

Note that we lift only into 1 higher dimension.

Achieving LP *k*-consistency

We used a **modified lift-and-project procedure**

Optionally:

Project resulting system onto x to obtain constraints
in original variables.

Project system onto $x_{J_{k-1}}$ to obtain sparse cuts.

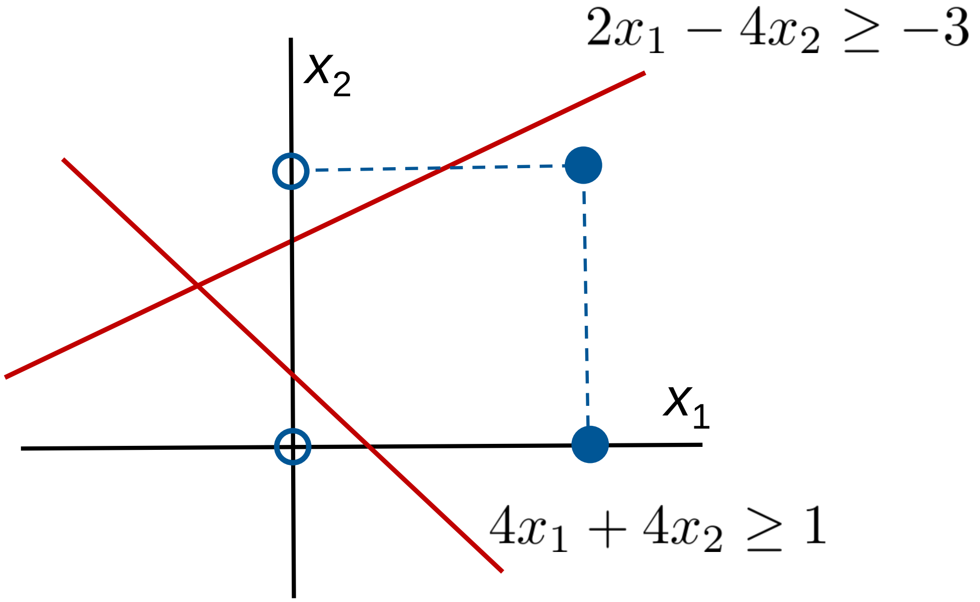
Thus when *k* is small, LP *k*-consistency can be achieved
by adding **very sparse cuts**—which tend to be strong.

LP *k*-consistency

Example

Lift & project generates LP 2-consistent constraint set

$$
\begin{array}{ll}
-x_2 + 2y \geq 0 & y \geq 0 \\
2x_1 - 3x_2 - 2y + 3 \geq 0 & x_1 - y \geq 0 \\
3x_2 + 4y \geq 0 & x_2 - y \geq 0 \\
4x_1 + x_2 - 4y - 1 \geq 0 & -x_1 - x_2 + y + 1 \geq 0
\end{array}
$$

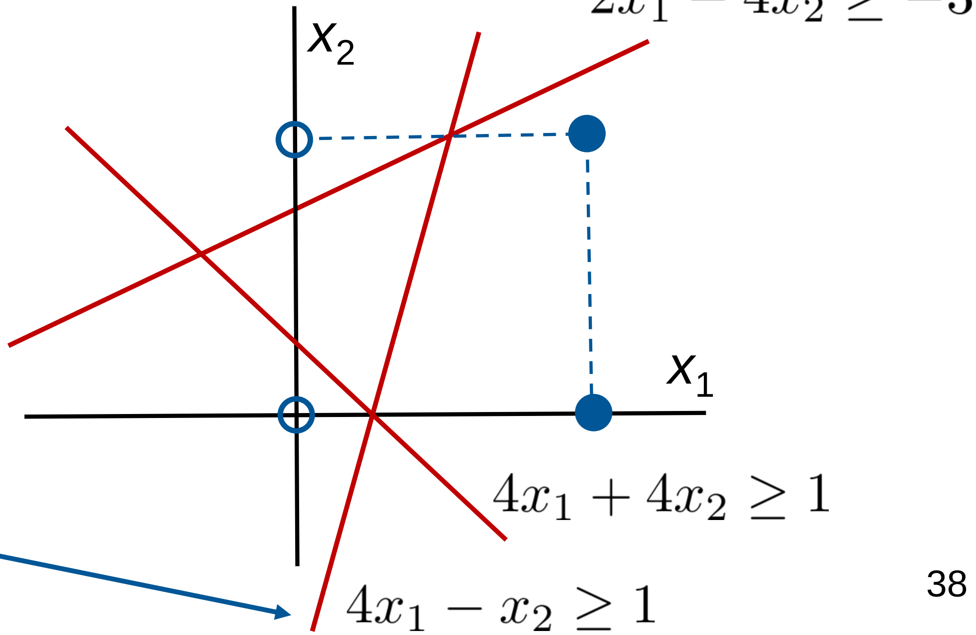


LP *k*-consistency

Example

Lift & project generates LP 2-consistent constraint set

$-x_2 + 2y \geq 0$	$y \geq 0$
$2x_1 - 3x_2 - 2y + 3 \geq 0$	$x_1 - y \geq 0$
$3x_2 + 4y \geq 0$	$x_2 - y \geq 0$
$4x_1 + x_2 - 4y - 1 \geq 0$	$-x_1 - x_2 + y + 1 \geq 0$



Projection onto (x_1,x_2) yields

LP *k*-consistency and Backtracking

Achieving LP *k*-consistency can reduce backtracking when traditional separating cuts do not.

This is shown in the following example.

A lift-and-project cut that achieves LP 2-consistency results in a smaller search tree than separating lift-and-project cuts.

LP *k*-consistency and Backtracking

Achieving LP *k*-consistency can reduce backtracking when traditional separating cuts do not.

This is shown in the following example.

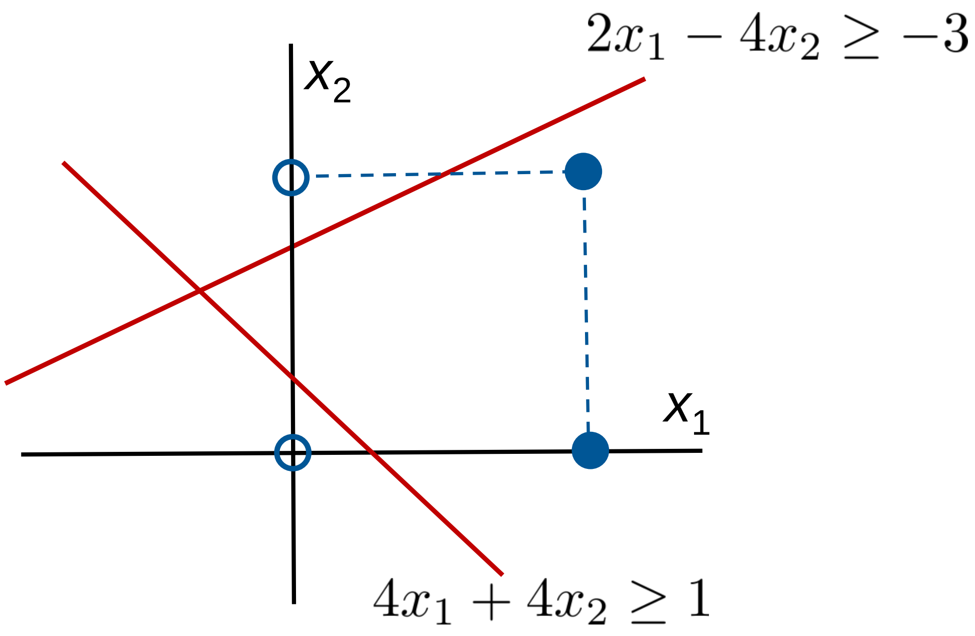
A lift-and-project cut that achieves LP 2-consistency results in a smaller search tree than separating lift-and-project cuts.

The example does not show that achieving LP *k*-consistency is practical in an IP solver.

It only shows that, even in a very small example, achieving LP *k*-consistency can cut off partial assignments and reduce backtracking when separating cuts do not.

Separating cuts

Maximize $3x_2 - x_1$

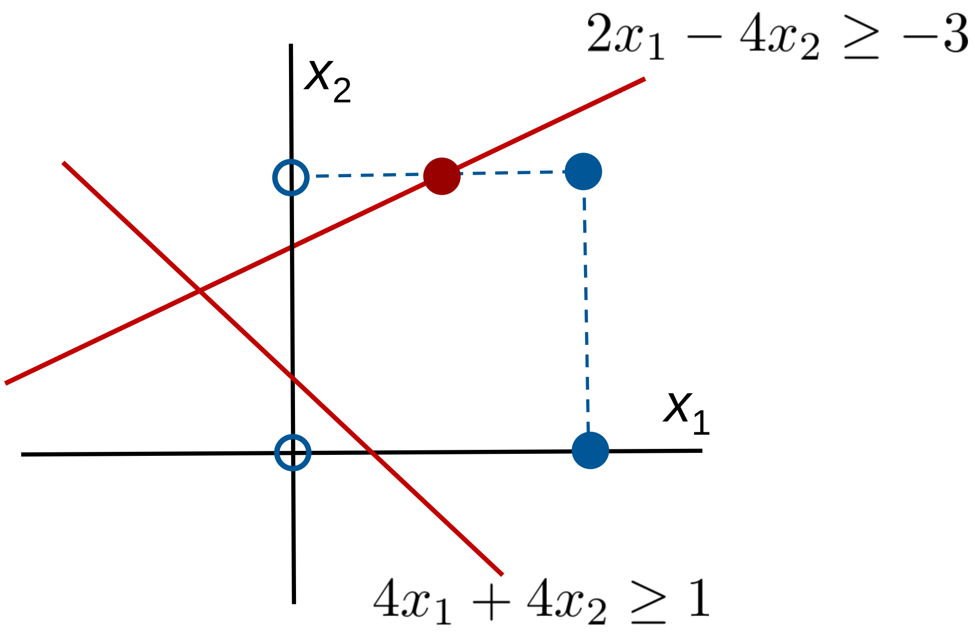


Separating cuts

$$x = (\tfrac{1}{2}, 1)$$

Maximize $3x_2 - x_1$

LP solution is $x = (\tfrac{1}{2}, 1)$



Separating cuts

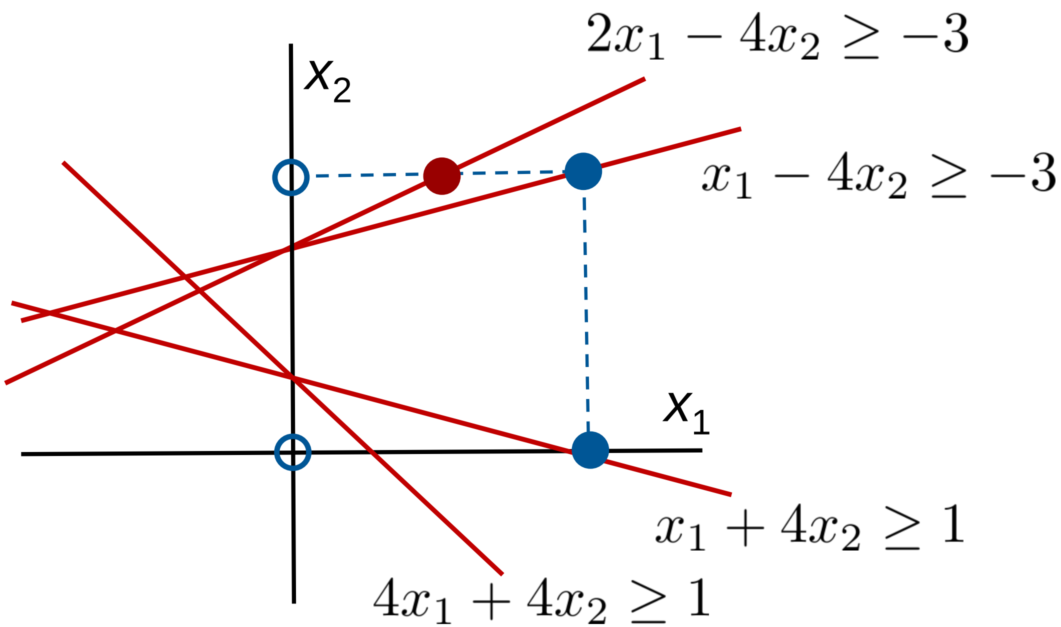
$$x = (\tfrac{1}{2}, 1)$$

Maximize $3x_2 - x_1$

LP solution is $x = (\tfrac{1}{2}, 1)$

Generate lift & project cuts on x_1

Only one cut is separating



Separating cuts

$$x = (0, \tfrac{3}{4})$$

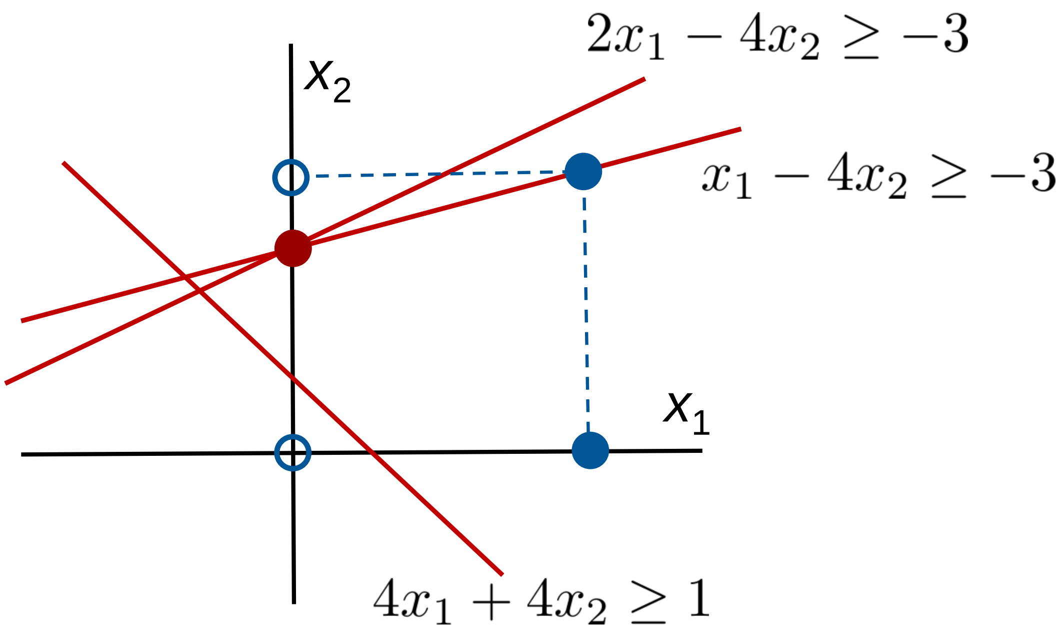
Maximize $3x_2 - x_1$

LP solution is $x = (\frac{1}{2}, 1)$

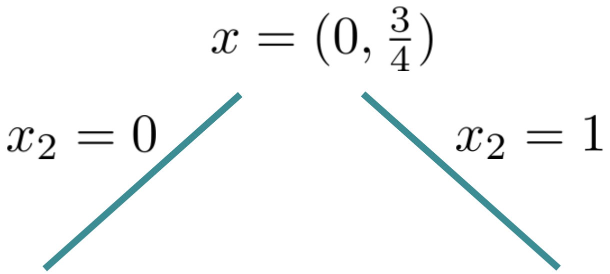
Generate lift & project cuts on x_1

Only one cut is separating

New LP solution is $x = (0, \frac{3}{4})$

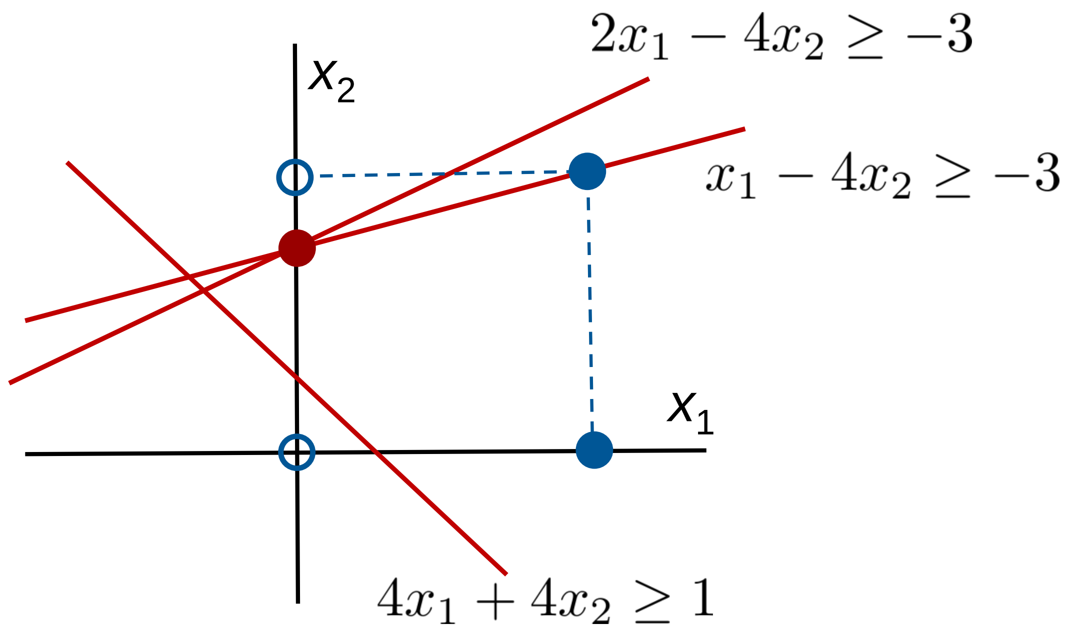


Separating cuts

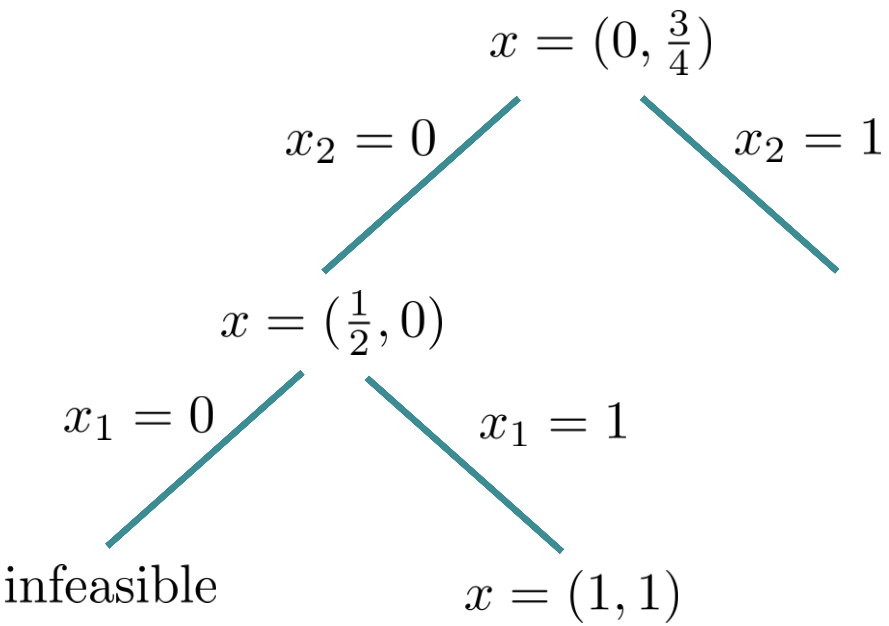


New LP solution is $x = (0, \frac{3}{4})$

Branch on x_2

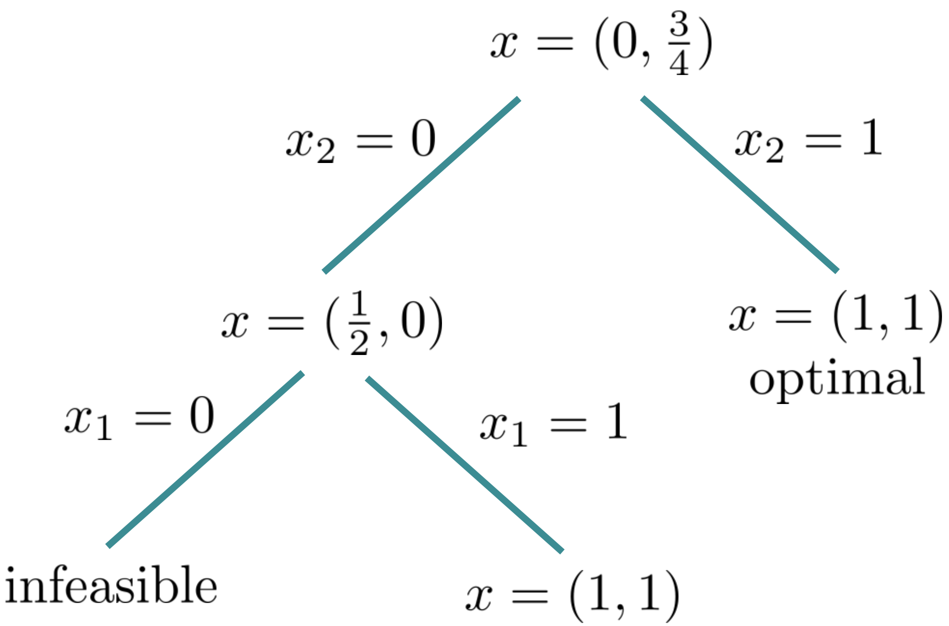


Separating cuts



Branch on x_1

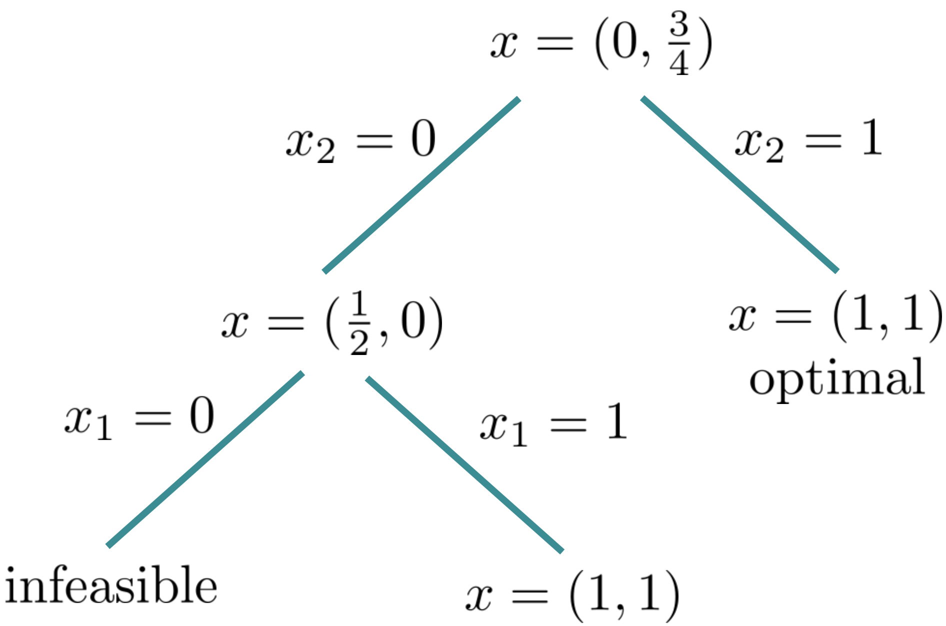
Separating cuts



Branch on x_1

Backtrack.

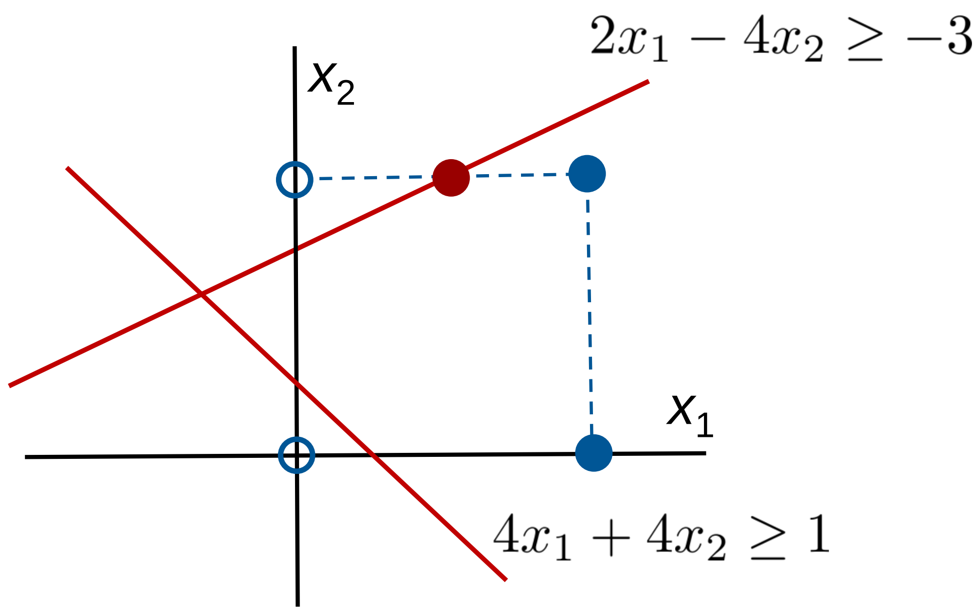
Separating cuts



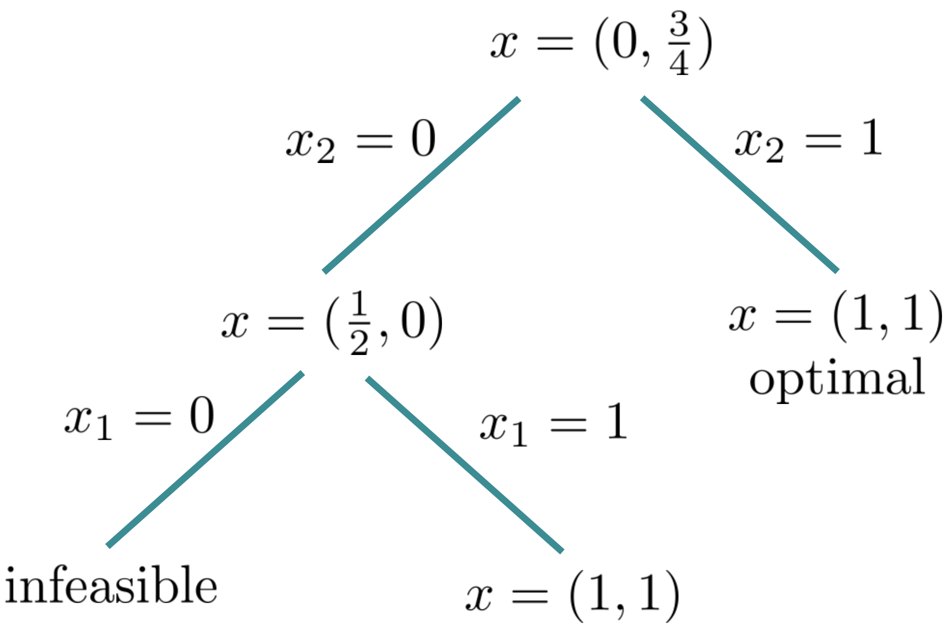
LP 2-consistency

$x = (\frac{1}{2}, 1)$

Branching order x_1, x_2



Separating cuts

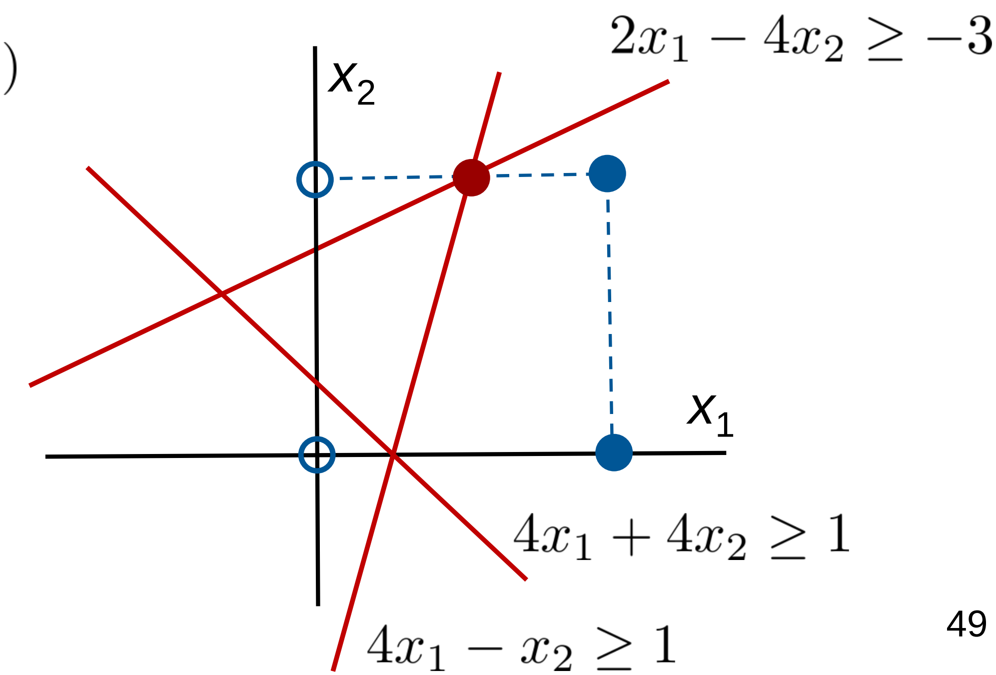


LP 2-consistency

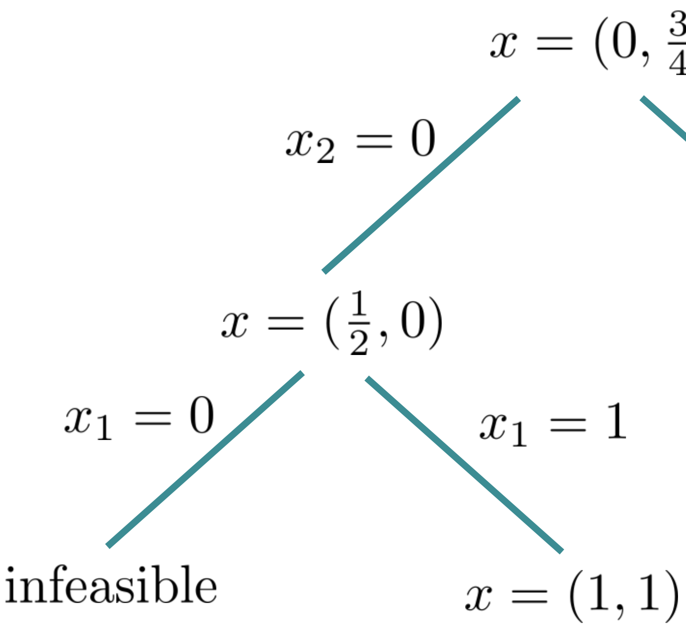
$x = (\frac{1}{2}, 1)$

Branching order x_1, x_2

Achieve 2-consistency by generating lift & project cut on x_2



Separating cuts



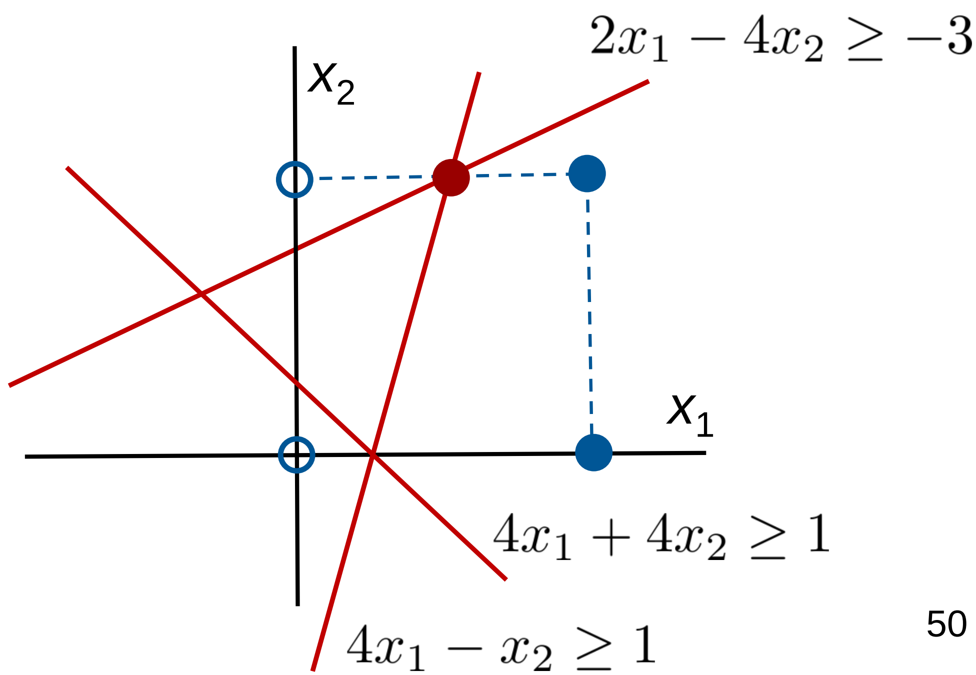
Keep this cut even though it is **not** separating

LP 2-consistency

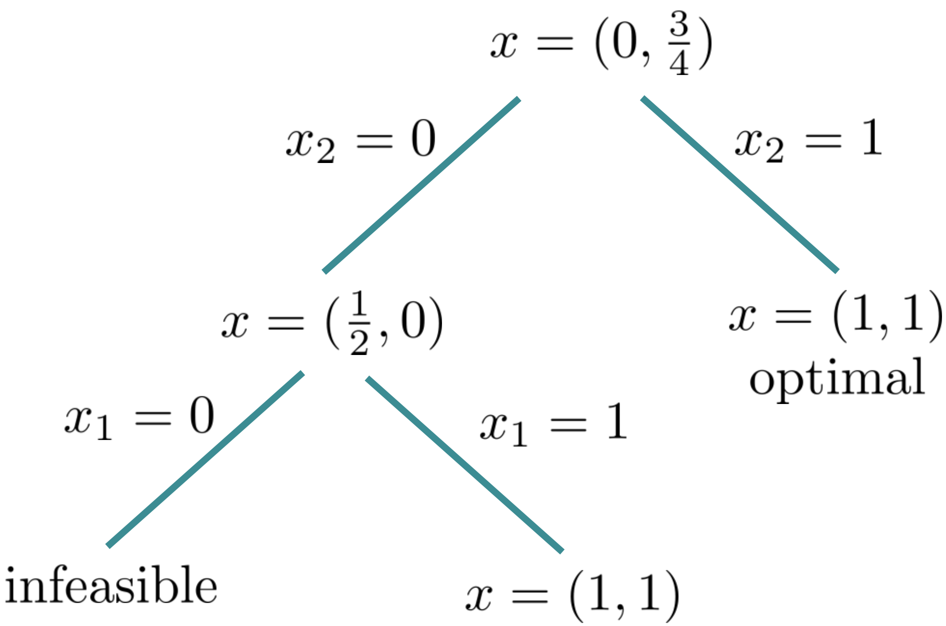
$$x = (\tfrac{1}{2}, 1)$$

Branching order x_1, x_2

Achieve 2-consistency by generating lift & project cut on x_2



Separating cuts



Keep this cut even though it is **not** separating.

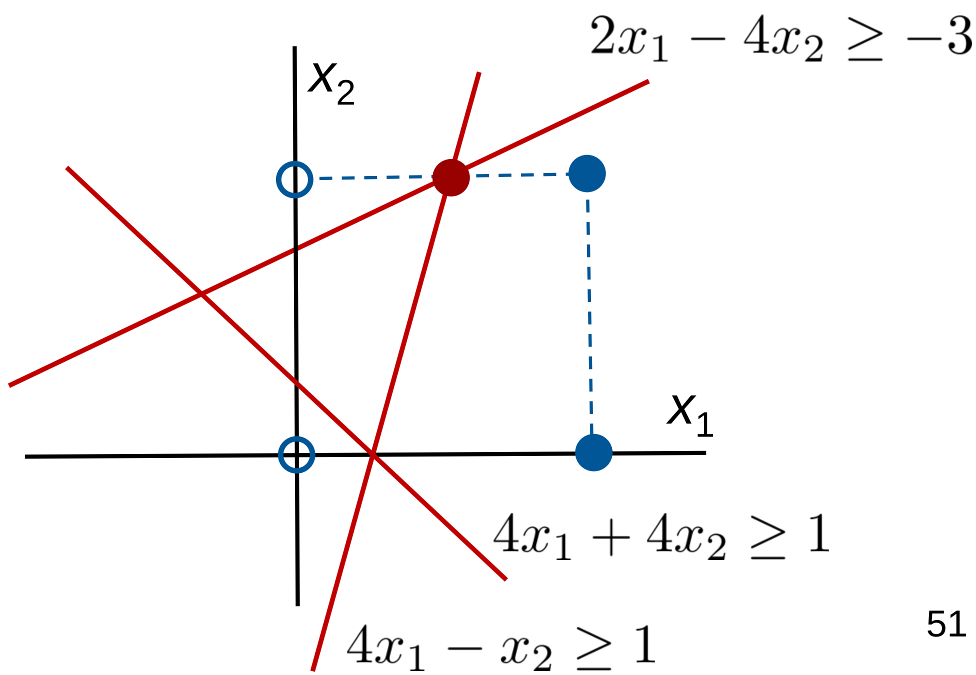
$x_1 = 0$ is **inconsistent** with LP relaxation

LP 2-consistency

$x = (\frac{1}{2}, 1)$

Branching order x_1, x_2

Achieve 2-consistency by generating lift & project cut on x_2



Separating cuts

$x = (0, \frac{3}{4})$

$x_2 = 0$ → $x = (\frac{1}{2}, 0)$

- $x_1 = 0$ → infeasible
- $x_1 = 1$ → $x = (1, 1)$

$x_2 = 1$ → $x = (1, 1)$ optimal

LP 2-consistency

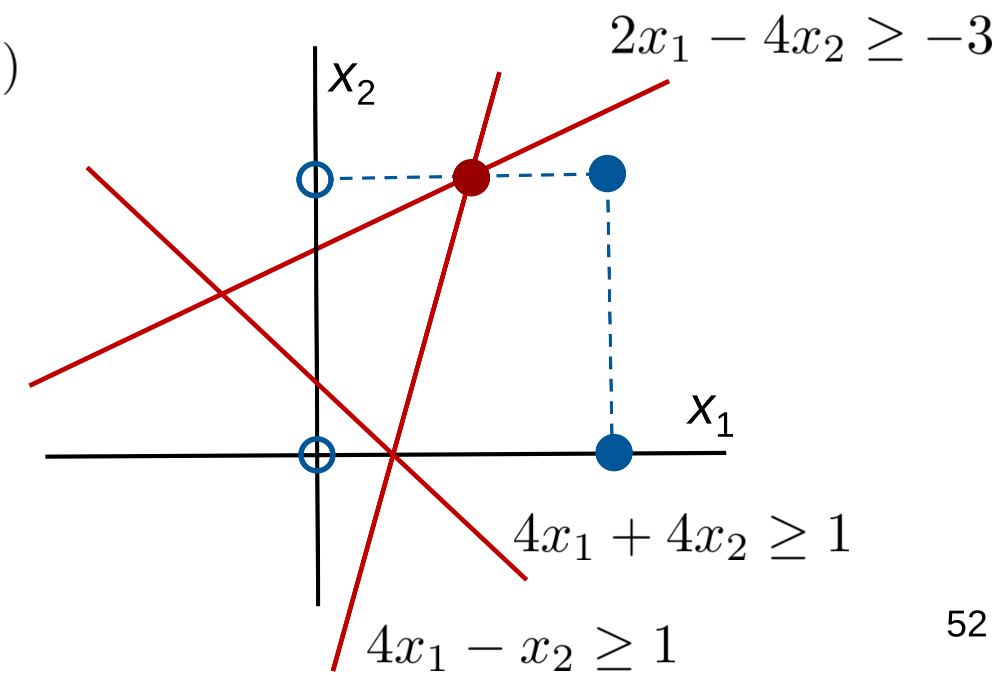
$x = (\frac{1}{2}, 1)$

$x_1 = 1$

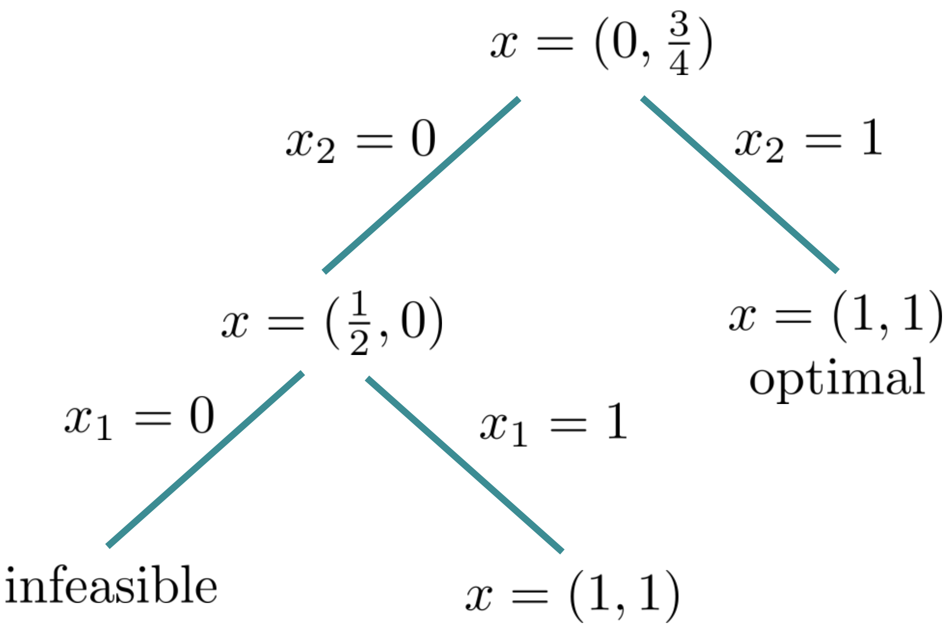
Keep this cut even though it is **not** separating.

$x_1 = 0$ is **inconsistent** with LP relaxation

So branch $x_1 = 1$



Separating cuts

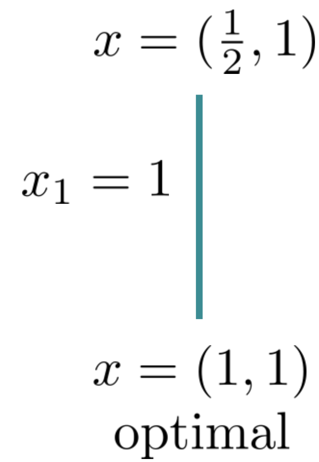


Keep this cut even though it is **not** separating.

$x_1 = 0$ is **inconsistent** with LP relaxation

So branch $x_1 = 1$

LP 2-consistency



This solves the problem with smaller search tree.

Application

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

Application

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

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.

Very preliminary computational tests

Random instances.

CPLEX with default cuts No presolve	vs.	CPLEX with default cuts (no presolve) plus RLT cuts to achieve LP *k*-consistency at all nodes. Bound on objective function included in constraint set.

Very preliminary computational tests

Random instances.

Vars.	Con-strs.	# CPLEX cuts	CPLEX tree size	CPLEX time (s)	# our cuts	Our tree size	Our time (s)
25	25	58	**263**	3.6	34	**82**	87
30	30	55	**194**	2.6	48	**158**	194
35	35	105	**1412**	19	175	**394**	905

Contributions

- New concept of consistency
  - LP consistency, based on defining consistency with respect to a relaxation
- Novel approach to IP.
  - Identify cuts that exclude infeasible partial solutions rather than fractions solutions.
  - May be computationally useful at some point.
- Rethinking IP.
  - How an inequality can be stronger than facet-defining.
  - How cuts can reduce backtracking without an LP relaxation, by achieving some form of consistency.

Research Issues

- Extend to MILP
  - Probably straightforward
- Computational issues
  - Heuristics to generate sparse cuts (by achieving LP *k*-consistency for small *k*)
  - At which nodes to achieve (partial) *k*-consistency?
- Reinterpret traditional cuts
  - To what extent do they achieve consistency?
  - Traditional cuts that are useful even when non-separating