Lecture 11 Modeling of Discrete and Continuous Decisions

Can Li

ChE 597: Computational Optimization
Purdue University

Outline of this lecture

- Guidelines for formulating NLP models
- Guidelines for formulating MILP models

NLP modeling

- Motivation: Besides developing more robust algorithms, formulating a good model also could help avoid numerical issues, nonconvexities, etc.

Guideline 1 It is important to try to specify lower, upper bounds on all variables

$$x^L \leq x \leq x^U$$

as this leads to more constrained feasible regions, which in turn helps to obtain better starting points and make the solution of the NLP more robust.
The upper and lower bounds should be made **as tight as possible**.

NLP modeling guidelines

Guideline 2 It is important to try to formulate whenever possible all the constraints as linear constraints. This helps all the methods, especially the global optimization algorithm that we will discuss in the second half of the semester.
As a simple example, it is always better to reformulate the nonlinear inequalities

$$\frac{x_1}{x_2} \leq 2 \quad x_2 \geq 0$$

as the linear equality

$$x_1 - 2x_2 \leq 0 \quad x_2 \geq 0$$

NLP modeling guidelines

It is important to try to the scale variables and constraints. For instance if we have the bounded variables,

$$0 \leq x_1 \leq 1 \quad 0 \leq x_2 \leq 10,000$$

we can define new a new variable $z_2 = \frac{x_2}{10,000}$ so that the bounds become

$$0 \leq x_1 \leq 1 \quad 0 \leq_{z_2} \leq 1$$

Most of the second-order algorithms involve solving linear systems. This helps make the matrix well-conditioned.

Warmstart NLP

Some of the NLP solvers benefit from warmstart. It will take the initial value you provide to start the algorithm. However, not all the solver allow warmstart.

- For interior point solvers, IPOPT allows warmstart; Mosek doesn't.
- LP solvers like CPLEX and Gurobi are amenable to warmstart. If you provide the optimal solution, it will be certified.
- Overall, LP solvers are easier to warmstart than NLP solvers. (Simplex algorithm benefits from primal/dual feasible basis).

```
model.y[0] = 1
model.y[1] = 0
opt = pyo.SolverFactory("cplex")
results = opt.solve(model, warmstart=True)
```

Listing 1: Pyomo warmstart example

Mixed-integer linear programming

$$\begin{aligned} \min Z = \quad & a^T x + b^T y \\ \text{s.t.} \quad & Ax + By \leq d \\ & x \in \mathbb{R}^{n_1} \quad y \in \mathbb{Z}^{n_2} \end{aligned}$$

- y represent the integer variables. In most problems, y are binary variables $y \in \{0,1\}^{n_2}$ (represent logic).
- NP-hard to solve in general.
- Wide applications in engineering, health care, logistics, machine learning, and so on.
- Modeling is nontrivial. Solution time of MILP solvers is very sensitive how well you model the problem.

Examples of modeling with 0-1 variables

Binary variables are usually used to model logic.
Let

$$y_j = \begin{cases} 1 \text{ if } j \text{ true} \\ 0 \text{ if } j \text{ false} \end{cases}$$

- At most one of y_1 and y_2 is true. One might be tempted to model it as

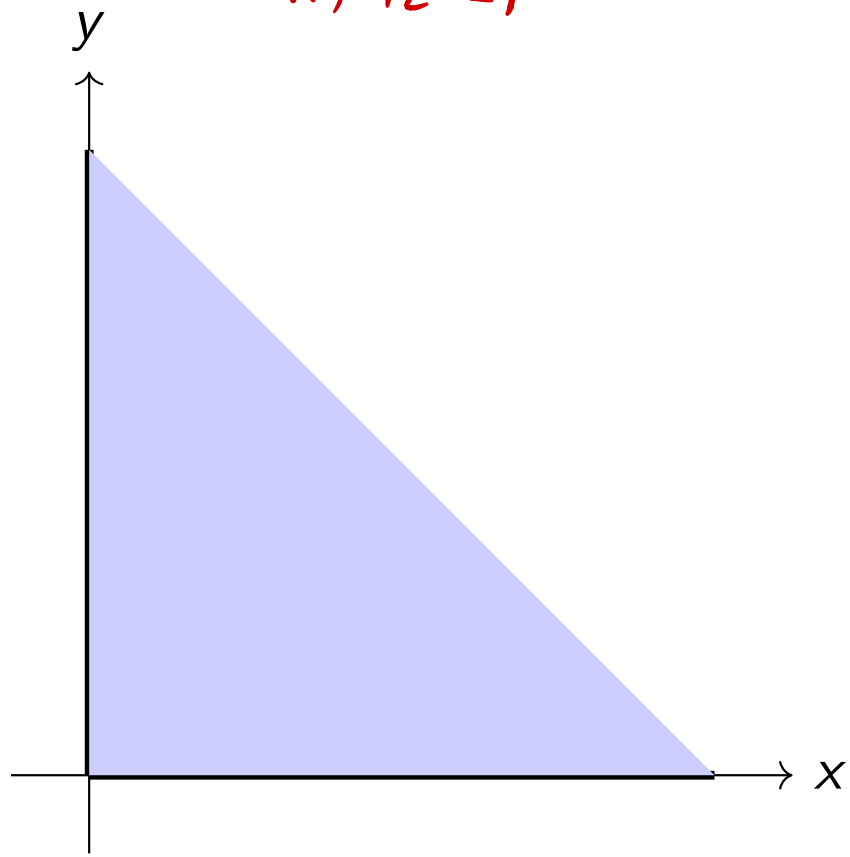
  $$y_1 \cdot y_2 = 0$$

  However, it is nonlinear thus not MILP-representable. We can also express this constraint with a linear inequality

  $$y_1 + y_2 \leq 1$$

$y_1 \, y_2 = 0$

$0 \le y_1, y_2 \le 1$



$y_1 + y_2 \le 1$

$0 \le y_1, y_2 \le 1$

Figure: Continuous relaxation of $y_1 \cdot y_2 = 0$ is a nonconvex region. Continuous relaxation of $y_1 + y_2 \le 1$ is convex.

Big-M constraints

We want to represent the condition if $y = 1 \Rightarrow h(x) = 0$. A simple nonlinear constraint representing this condition, which is nonconvex, is given by,

$$yh(x) = 0$$

$h(x) = a^T x - b = 0$

$y\,h(x) = y(a^T x - b) = 0$

It can clearly be seen that $h(x) = 0$ if $y = 1$, while for $y = 0$, $h(x)$ can take any value. If $h(x)$ is linear, this destroys the linearity.

Big-M inequalities

$$-M(1 - y) \leq h(x) \leq M(1 - y)$$

- If $y = 1$, it reduces to $h(x) = 0$, since $0 \leq h(x) \leq 0$,
- For $y = 0$, it reduces to the inequality $-M \leq h(x) \leq M$, which is redundant for sufficiently large M.
- In practice, however, one should make the M **as small as possible** while not eliminating any potential optimal solution. The reason is that it affects continuous relaxation of the mixed-integer programming problem.

Multiple choice constraints

a) Select at least one

$$\sum_{j \in J} y_j \geq 1$$

b) Select exactly one

$$\sum_{j \in J} y_j = 1$$

c) Select not more than one

$$\sum_{j \in J} y_j \leq 1$$

If then condition

Assume we state the condition "If select k then select j " ($y_k => y_j$), this condition can written as the linear inequality,

$$y_k - y_j \leq 0$$

This can be easily verified as follows:

- if $y_k = 1 \rightarrow y_j = 1$
- if $y_k = 0 \rightarrow y_j = 0$ or 1

We should note that the condition iff (if and only if) can be expressed as the equality,

$$y_k = y_j$$

Disjunction

Consider the disjunctive constraint: $f_1(x) \leq 0$ or $f_2(x) \leq 0$, meaning either $f_1(x) \leq 0$ or $f_2(x) \leq 0$ must be satisfied. This disjunction can be expressed with the two following constraints that are linear in y :

$$
\begin{aligned}
& f_1(x) \leq M\,(1 - y_1) \\
& f_2(x) \leq M\,(1 - y_2) \\
& y_1 + y_2 = 1 \\
& y_1, y_2 \in \{0, 1\}
\end{aligned}
$$

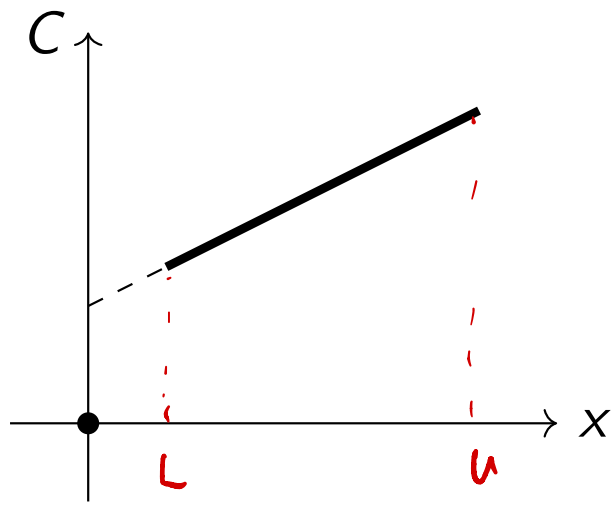
We can easily verify that two cases can occur

- if $y_1 = 1$ then $y_2 = 0$ and we have $f_1(x) \leq 0, f_2(x) \leq M$.
- if we have $y_1 = 0$ then $y_2 = 1$ and we have $f_2(x) \leq M, f_2(x) \leq 0$.

Discontinuous domains and functions

Fixed charge model describing the cost of a equipment:

$$C = \begin{cases} \alpha + \beta x & L \leq x \leq U \\ 0 & x = 0 \end{cases}$$



The function has a discontinuous domain given by $x = 0$ and $L \leq x \leq U$. α represents the fixed cost, β represents the variable cost coefficient.

Discontinuous domains and functions

Fixed charge model:

$$C = \begin{cases} \alpha + \beta x & L \leq x \leq U \\ 0 & x = 0 \end{cases}$$

MILP formulation

$$C = \alpha y + \beta x$$
$$Ly \leq x \leq Uy$$
$$x \geq 0, \quad y \in \{0, 1\}$$

- if $y = 0$ then $x = 0$ and $C = 0$
- if $y = 1$ then $L \leq x \leq U$ and $C = \alpha + \beta x$
- if $L = 0$, y takes $y = 0$ when $x = 0$ in a minimization problem.

Propositional logic

Motivation: we aim to develop a systematic approach of modeling logic using linear constraints that involve binary variables.
We deals with logic operators

1. "$\vee$" OR (inclusive OR)
2. "$\wedge$" AND
3. "$\neg$" NOT
4. "$\Rightarrow$" Implication
5. "$\underline{\vee}$" XOR (exclusive OR)
6. "$\Leftrightarrow$" IFF (equivalence)

Truth Tables for Logical Operators

A truth table is used to determine the output of a logical expression based on all possible combinations of its inputs. At the end, we need to make the linear constraints consistent with the truth table.

OR ($\vee$)

A	B	$A \vee B$
T	T	T
T	F	T
F	T	T
F	F	F

NOT ($\neg$)

A	$\neg A$
T	F
F	T

Exclusive OR (XOR)

A	B	$A \veebar B$
T	T	F
T	F	T
F	T	T
F	F	F

AND ($\wedge$)

A	B	$A \wedge B$
T	T	T
T	F	F
F	T	F
F	F	F

Implication ($\Rightarrow$)

A	B	$A \Rightarrow B$
T	T	T
T	F	F
F	T	T
F	F	T

Equivalence (IFF)

A	B	$A \Leftrightarrow B$
T	T	T
T	F	F
F	T	F
F	F	T

Calculus of Logic Operators

a) De Morgan's Theorem

$$\neg(A \vee B) \equiv \neg A \wedge \neg B$$
$$\neg(A \wedge B) \equiv \neg A \vee \neg B$$

A	B	$A \vee B$	$\neg(A \vee B)$	$\neg A$	$\neg B$	$\neg A \wedge \neg B$
T	T	T	F	F	F	F
T	F	T	F	F	T	F
F	T	T	F	T	F	F
F	F	F	T	T	T	T

Calculus of Logic Operators

a) De Morgan's Theorem

$$\neg(A \vee B) \equiv \neg A \wedge \neg B$$
$$\neg(A \wedge B) \equiv \neg A \vee \neg B$$

b) Distribution of OR

$(a+b) \times c = ac + bc$

$$(A \wedge B) \vee C \equiv (A \vee C) \wedge (B \vee C)$$

c) Distribution of AND

$$(A \vee B) \wedge C \equiv (A \wedge C) \vee (B \wedge C)$$

d) Implication

$$A \Rightarrow B \equiv \neg A \vee B$$

Represent logic propositions as constraints

To express a logic proposition let p_j be literal representing a selection or action. We can associate to that literal a binary variable y_j such that $y_j = 1$ means that the literal is true, while $y_j = 0$ means that it is false. Applying the basic operators to the literals p_j can be represented in terms of the binary variables as follows:

- NOT: $\neg p_j$ by $1 - y_j$
- OR: $p_i \vee p_j$ by $y_i + y_j \geq 1$
- AND: $p_i \wedge p_j$ by $y_i = 1, y_j = 1$
- Implication: $p_i \Rightarrow p_j$ is equivalent to $\neg p_i \vee p_j$, which can be represented by $1 - y_i + y_j \geq 1$ or ~~$y_i \geq y_j$~~ $y_i \leq y_j$
- XOR: $p_i \veebar p_j$ by $y_i + y_j = 1$
- EQUIVALENCE: $p_i \Leftrightarrow p_j$ by $y_i = y_j$

How do we represent a general logic proposition using constraints?

For example, what if we have

$$(p_1 \wedge p_2) \Rightarrow (p_3 \wedge p_4)$$

$$p_F \Rightarrow (p_D \wedge \neg p_M)$$

We are going to show a systematic method for converting any logic functional to linear constraints.

Functional Completeness of AND and NOT

Definition: A set of logical operators is *functionally complete* if any logical propositions (any truth tables) can be constructed using only the operators from this set.
Theorem: The set consisting of AND ($\wedge$) and NOT ($\neg$) operators is functionally complete.
Objective: Prove that logical operations OR, XOR, IMPLICATION, and EQUIVALENCE can be expressed using only AND and NOT operators.
Implication: It suffices to show how to represent logic propositions that involve AND, NOT using constraints. But for convenience, we assume one may write logical propositions using AND, NOT, OR, IMPLICATION (usually works in practice).

Proof of Functional Completeness

OR Operation:

$$A \vee B \equiv \neg(\neg A \wedge \neg B)$$

XOR Operation:

$$A \underline{\vee} B \equiv (A \wedge \neg B) \vee (\neg A \wedge B) \equiv \neg(\neg(A \wedge \neg B) \wedge \neg(\neg A \wedge B))$$

IMPLICATION Operation:

$$A \Rightarrow B \equiv \neg A \vee B \equiv \neg(A \wedge \neg B)$$

EQUIVALENCE Operation:

$$A \Leftrightarrow B \equiv (A \wedge B) \vee (\neg A \wedge \neg B) \equiv \neg(\neg(A \wedge B) \wedge \neg(\neg A \wedge \neg B))$$

Each operation is constructed using only AND ($\wedge$) and NOT ($\neg$), demonstrating the functional completeness of these operators.

Conjunctive Normal Form (CNF)

$(p_1 \vee p_2) \wedge (\neg p_3 \vee p_4)$

Definition: CNF is a form of logical expression where a conjunction (AND, $\wedge$) of one or more clauses forms the whole formula, and each clause is a disjunction (OR, $\vee$) of literals. A literal is either a variable or its negation.

Formal Representation:

$$\Omega = \bigwedge_{j \in J} \left(\bigvee_{i \in I_j} l_i \right)$$

where l_i represents a literal, which may be a proposition p_i or its negation $\neg p_i$.

Example of CNF to constraints

Example: Consider the CNF expression $(p_1 \vee \neg p_2) \wedge (\neg p_1 \vee p_3)$. This expression comprises two clauses, where each clause contains literals that can be either negated or non-negated.
Inequalities Representation: Translating to 0-1 variables, the clauses can be expressed as inequalities:

$$y_1 + (1 - y_2) \geq 1$$
$$(1 - y_1) + y_3 \geq 1$$

Simplifying, we get:

$$y_1 - y_2 \geq 0$$
$$-y_1 + y_3 \geq 0$$

Translating CNF to Constraints

Given a CNF expression:

$$\Omega = \bigwedge_{j \in J} \left(\bigvee_{i \in I_j} l_i \right)$$

where l_i is a literal, either a proposition p_i or its negation $\neg p_i$.

1. Each proposition p_i is represented by a binary variable y_i, where $y_i = 1$ if p_i is true, and $y_i = 0$ if p_i is false.
2. The negation $\neg p_i$ is represented as $1 - y_i$.
3. A disjunction (OR) within a clause $\bigvee_{i \in I_j} l_i$ translates to the constraint that at least one of the literals in the clause must be true. This is modeled as:

$$\sum_{i \in I_j} y'_i \geq 1 \quad \forall j \in J$$

$(P_1 \vee P_2 \vee \neg P_3)$

$y_1 + y_2 + (1 - y_3) \geq 1$

where y'_i corresponds to y_i for non-negated p_i and to $1 - y_i$ for negated $\neg p_i$.

How to translate any logic proposition to CNF

It is clear that if the logic is given in CNF form, one can readily transform it into linear 0-1 inequalities. The problem is that arbitrary logic is usually not expressed in CNF form. Apply the following steps recursively we convert any logic propositions in AND, OR, NOT, IMPLICATION to CNF.

1. Remove implications using the fact

$$A \Rightarrow B \equiv \neg A \vee B$$

2. Move negation inwards by applying De Morgan's Theorem.

$$\neg(A \vee B) \equiv \neg A \wedge \neg B$$
$$\neg(A \wedge B) \equiv \neg A \vee \neg B$$

3. Recursively distribute OR over AND

$$(A \wedge B) \vee C \equiv (A \vee C) \wedge (B \vee C)$$

Context: If select flash (P_F) then select distillation (P_D) and not membrane (P_M).

$$P_F \Rightarrow (P_D \wedge \neg P_M)$$

$$\neg P_F \vee (P_D \wedge \neg P_M)$$

$$(\neg P_F \vee P_D) \wedge (\neg P_F \vee \neg P_M)$$

$$1 - Y_F + Y_D \geq 1, \quad 1 - Y_F + 1 - Y_M \geq 1$$

Example

Context: Consider a process synthesis problem where the logic statement is "If select flash then select distillation and not membrane".

$$p_F \Rightarrow (p_D \wedge \neg p_M)$$

1. Remove the implication:

$$\neg p_F \vee (p_D \wedge \neg p_M)$$

2. Since there's no need to apply De Morgan's laws, distribute the OR over the AND to get CNF:

$$(\neg p_F \vee p_D) \wedge (\neg p_F \vee \neg p_M)$$

3. Translate into inequalities:

$$1 - y_F + y_D \geq 1, \quad 1 - y_F + 1 - y_M \geq 1$$

Simplifying, we obtain:

$$y_D \geq y_F, \quad y_F + y_M \leq 1$$

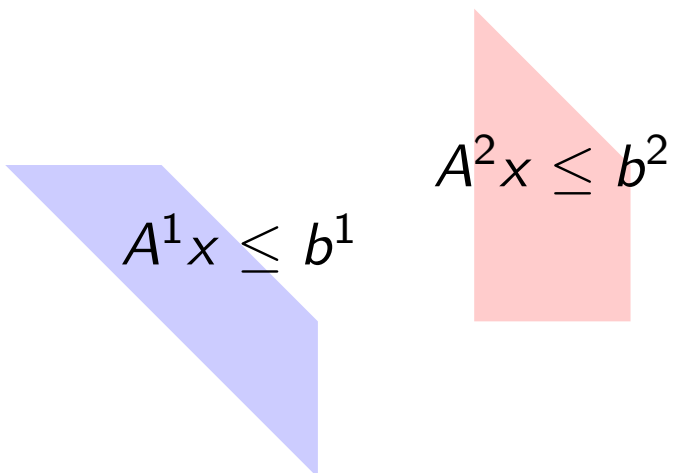
Linear Disjunctions

Overview: Linear disjunctions take the form:

$$\bigvee_{j \in D} [A^j x \leq b^j]$$

Here, $j \in D$ indexes disjuncts, x is a vector of continuous variables, A^j matrices, and b^j vectors of constraints. In most cases, the polyhedral sets $\{x \mid A^j x \leq b^j\}$ are disjoint.

History: Introduced by Egon Balas (1960s).



Linear disjunctions represent the union of two disjoint polyhedral sets.

Example: Batch Scheduling Problem

Scenario: In batch scheduling, a common condition is either "A before B" or "B before A".
Formulation: Given start times t_A, t_B and processing times p_A, p_B:

$$(t_A + p_A \leq t_B) \vee (t_B + p_B \leq t_A)$$

Interpretation: The left disjunct means A is processed before B. The right disjunct means B is processed before A. The $\vee$ indicates at least one condition must be true, functioning like an exclusive OR.

Big-M Reformulation

Big-M Reformulation: Assign each disjunct a binary variable y_j, where $y_j = 1$ means the disjunct's inequality is true, and $y_j = 0$ otherwise.

$$A^j x \leq b^j + M^j(1 - y_j), \quad j \in D$$

$$\sum_{j \in D} y_j = 1$$

$$y_j \in \{0, 1\}$$

M^j is a **vector** (same dimension as b^j) ensuring $A^j x \leq b^j$ for $y_j = 1$. For $y_j = 0$, $A^j x \leq b^j + M^j$ makes the inequality redundant.
Key of the big-M formulation Make M^j as small as possible. For different constraints, the values can be different.

Formulating Disjunctions with Big-M

Given the disjunction:

$$(t_A + p_A \leq t_B) \vee (t_B + p_B \leq t_A)$$

We introduce binary variables y_1 and y_2 for each disjunct. The Big-M formulation is:

- For $t_A + p_A \leq t_B$:

$$t_A + p_A - t_B \leq M_1(1 - y_1)$$

- For $t_B + p_B \leq t_A$:

$$t_B + p_B - t_A \leq M_2(1 - y_2)$$

- Ensuring exclusivity:

$$y_1 + y_2 = 1, \quad y_1, y_2 \in \{0, 1\}$$

M_1 and M_2 are large constants. This formulation ensures that only one of the conditions is active, adhering to the logical OR operation.

Convex hull reformulation

$$x = \sum_{j \in J} z^j$$

$$A^j z^j \leq b^j y_j \quad j \in D$$

$$\sum_{j \in J} y_j = 1$$

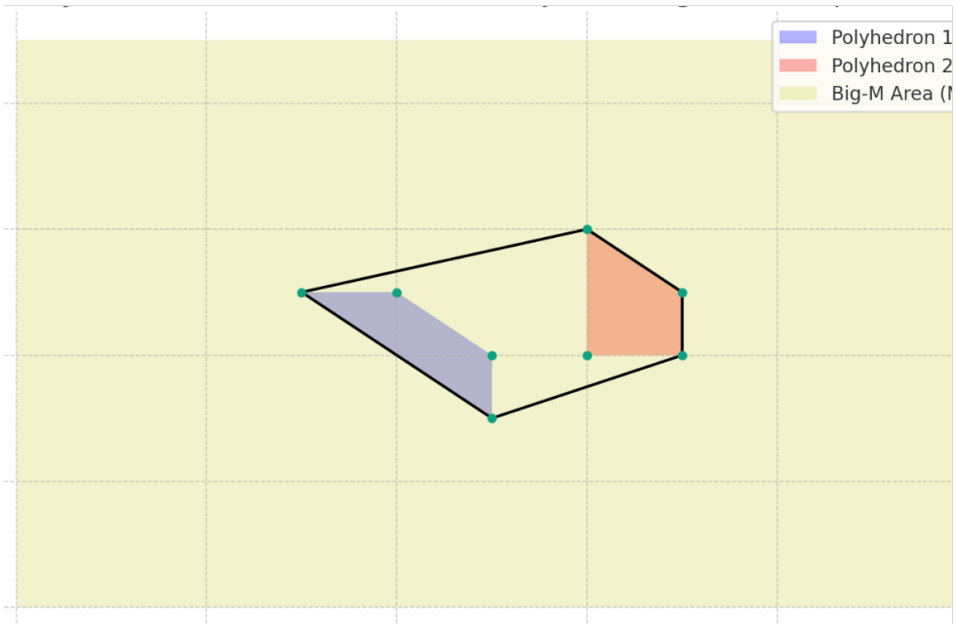
$$0 \leq z^j \leq U^j y_j \quad j \in D$$

$$y_j = 0, 1 \quad \forall j \in D$$

$$z^j \in \mathbb{R}^n \quad \forall j \in D$$

It can be shown that the convex hull reformulation describes the convex hull of the union of the polyhedra. It is a tighter than the big-M reformulation but uses more continuous variables. In practice, usually formulating disjunctions as big-M is better for MILP solvers.

Illustrative figure



Example

$$[x_1 - x_2 \leq -1] \vee [-x_1 + x_2 \leq -1]$$
$$0 \leq x_1, x_2, \leq 4$$

Big-M

$$x_1 - x_2 \leq -1 + M(1 - y_1)$$
$$-x_1 + x_2 \leq 1 + M(1 - y_2)$$
$$y_1 + y_2 = 1$$
$$y_1, y_2 = 0, 1$$
$$0 \leq x_1, x_2, \leq 4$$

Example-continued

Convex Hull

$$x_1 = z_1 + z_2$$

$$x_2 = w_1 + w_2$$

$$z_1 - w_1 \leq -y_1$$

$$-z_2 + w_2 \leq -y_2$$

$$y_1 + y_2 = 1$$

$$0 \leq z_1 \leq 4y_1$$

$$0 \leq z_2 \leq 4y_1$$

$$0 \leq w_1 \leq 4y_2$$

$$0 \leq w_2 \leq 4y_2$$

$$y_1 + y_2 = 1$$

$$0 \leq x_1, x_2, \leq 4$$

References

- Grossmann, I. E. (2021). Advanced optimization for process systems engineering. Cambridge University Press.