ILP modeling tricks

Ben Rosenberg

April 11, 2026

Overview

- Min, max, and negation (review)
- McCormick constraints
- Or and implies
- Big-M method
- Not equal

Min and max, and negation

For $z = \min(x_1, \ldots, x_n)$ or $z = \max(x_1, \ldots, x_n)$, we can use the same logic as for continuous variables:

$$z = \min(x_1, \ldots, x_n) \equiv z \leq x_1; z \leq x_2; \ldots; z \leq x_n$$

$$z = \max(x_1, \ldots, x_n) \equiv z \geq x_1; z \geq x_2; \ldots; z \geq x_n$$

For negation ($\neg$), we already saw this as part of our demonstration that SAT could be reduced to ILP:

$$x = \neg y \equiv x = 1 - y, \quad x, y \text{ binary}$$

McCormick constraints

We already saw an example of the McCormick constraints as part of the SAT reduction for $\wedge$:

$$z = x_1 \wedge x_2 \equiv \begin{cases} z \leq x_1 \\ z \leq x_2 \\ z \geq x_1 + x_2 - 1 \end{cases}, \quad z, x_1, x_2 \text{ binary}$$

More generally, for $\bigwedge_{i=1}^{n} x_i$, we have the following formulation:

$$z = x_1 \wedge \cdots \wedge x_n \equiv \begin{cases} z \leq x_i & \forall i \in \{1, \ldots, n\} \\ x_1 \geq \sum_{i=1}^{n} x_i - (n-1) \\ z, x_1, \ldots, x_n \text{ binary} \end{cases},$$

Note that this works for multiplication (e.g., $x_1 \cdot x_2 \cdot x_3$) as well, because it has the same interpretation as $\wedge$ for binary variables.

McCormick constraints example

Consider $n = 3$, with $z = x_1 \wedge x_2 \wedge x_3$.

The McCormick constraints to model this are:

$$z \leq x_1$$
$$z \leq x_2$$
$$z \leq x_3$$
$$z \geq x_1 + x_2 + x_3 - 2$$

If all the inputs are 1, then we have $z = 1 + 1 + 1 - 2 = 1$.

If any of the inputs are 0, then we have $z \leq 0$, so $z = 0$.

Or ($\vee$) constraints

We could try to model an OR constraint $z = \bigvee_{i=1}^{n} x_i$ like this:

$$z \geq \sum_{i=1}^{n} x_i, \quad z, x_1, \ldots, x_n \text{ binary}$$

But there's a problem with this formulation...

Or ($\vee$) constraints (cont.)

As-is, this formulation doesn't prevent z from being 1 even when $x_1 = \cdots x_n = 0$.

To fix this, we need to ensure that if $x_1 = \cdots x_n = 0$, we set z to 0 (without overconstraining z).

The best way to do that is:

$$z \leq \sum_{i=1}^{n} x_i$$

This is essentially the reverse of the McCormick constraints.

Or ($\vee$) constraints - the other type

Another type of $\vee$ constraint, perhaps simpler, is the one in which you want at least one of $x_1, x_2, \ldots, x_n$ to be 1.

This is easy to model as well:

$$x_1 + x_2 + \cdots + x_n \geq 1$$

Implies ($\rightarrow$), with a truth table

We can come up with a way to model $x \rightarrow y$ by thinking about the values x and y can take, but let's take the chance to do something more explicit with a truth table:

x	y	$x \rightarrow y$
0	0	1
0	1	1
1	0	0
1	1	1

Here, we can note that $x \rightarrow y$ has the same value as $y \geq x$. It follows that $x \leq y$ suffices to model $x \rightarrow y$.

Big-M constraints

Sometimes rather than a simple relationship between two (or more) variables, we may want to enable or disable an entire constraint.

Some things we may want to model (LHS and RHS are linear functions of binary decision variables):

- "If $x = 1$, then $LHS \geq RHS$"
- "If $x = 0$, then $LHS \geq RHS$"
- "If $x = 1$, then $LHS = RHS$"
- "$LHS \neq RHS$ always"

Big-M constraints (cont.)

To express these more general relationships we can use a constant that is sufficiently large that certain constraints are "deactivated" by its presence.

For example: "If $x = 0$, then $y \leq 100$ is enabled; otherwise, it is disabled" can be represented as follows:

$$y \leq 100 + M \cdot x$$

Here, if $x = 1$, the constraint becomes $y \leq 100 + M$, and M is so large that this will always be true, and therefore have no impact on the value of y.

Big-M constraints: not equal

Another possible use of Big-M is the $LHS \neq RHS$ constraint type.

We can model "$LHS \neq RHS$ always" as follows, where x is a binary helper decision variable:

$$LHS - RHS \leq -\epsilon + M \cdot x$$
$$LHS - RHS \leq -\epsilon + M \cdot (1 - x)$$

Here, the helper variable x allows the model to choose whether $LHS > RHS$ or $LHS < RHS$.

The ϵ is a small constant that suffices as the difference between LHS and RHS to ensure the two are not equal. For integers this can be 1; for continuous variables this would be smaller.

Big-M constraints: downsides

There are several downsides with the Big-M method of formulating an LP:

- Inefficiency: Adding a Big-M constraint is explicitly making the feasible region for the ILP "less convex". By introducing a split based on a decision variable, we make the LP relaxation less useful for determining a solution close to the ILP optimum
- Numerical instability: the M in the Big-M method may cause the solver to end up with small values for $1/M$ that fall below some tolerance required for the solver to determine solutions correctly

Big-M constraints: choosing a small M

Both of these problems are issues that can be mitigated by choosing M as small as possible (while still being sufficiently "big").

- Inefficiency is mitigated by a choice of smaller M, because the feasible region is smaller and therefore the solution to the LP relaxation is less likely to be very different from the ILP optimum
- Numerical instability is mitigated because the values that a solver may end up with for $1/M$ are less likely to fall under the numerical tolerance

How small to make M will vary based on the problem you are modeling.

Typically when we give the mathematical formulation of a model, we do not include a specific value for M, but if you use Big-M constraints in any formulations you implement in OR-Tools, you will need to determine what M should be.