Intro to integer linear programming (ILP)

Ben Rosenberg

December 24, 2025

Overview

- What is integer linear programming?
- Differences between ILP and LP
- Geometry/intuition of ILP
- Example ILPs
- Why ILPs are more difficult to solve
- SAT reduction to ILP

What is integer linear programming?

Integer linear programming is the same as linear programming, but with the constraint that all variables are integers.

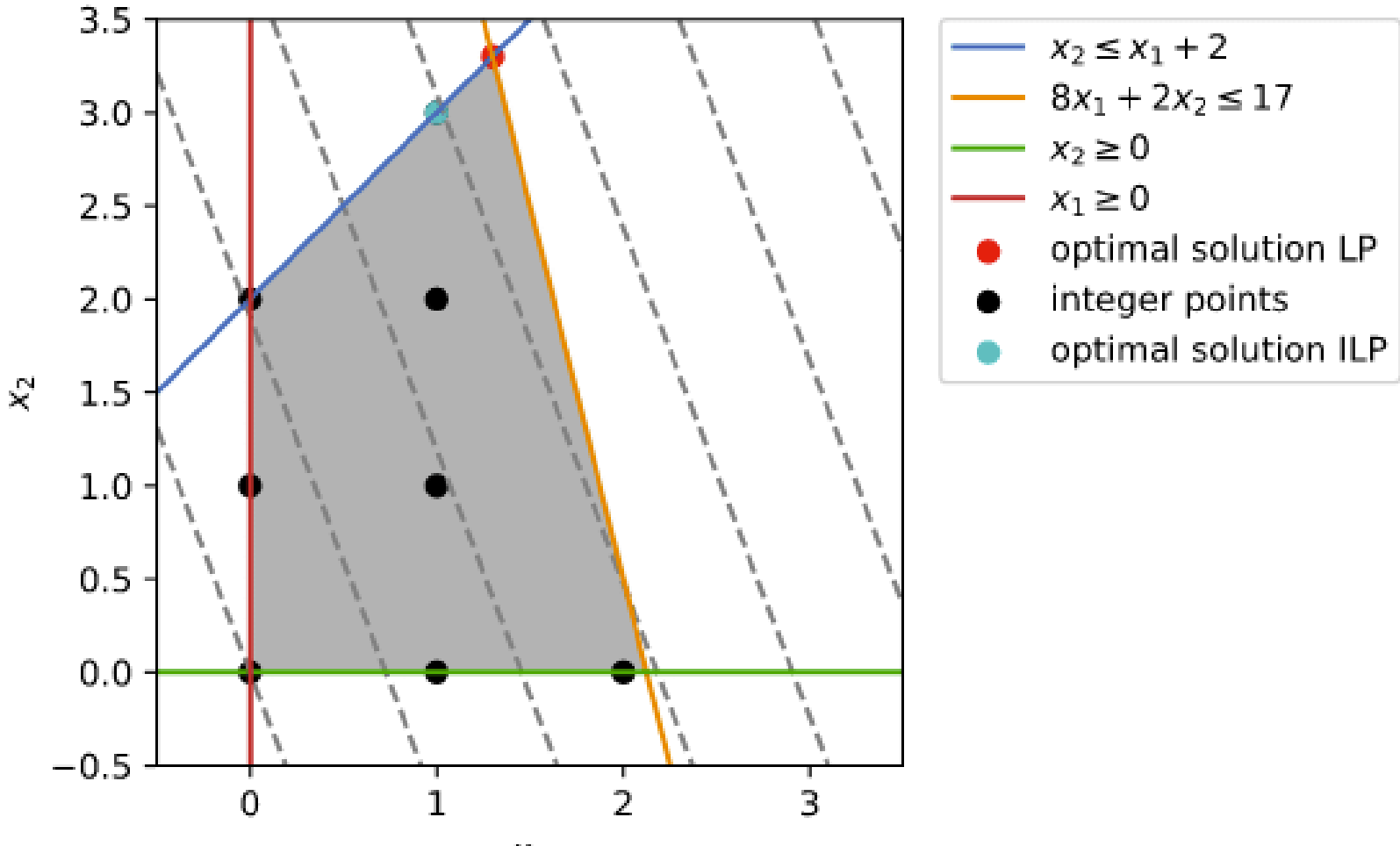
There's also mixed integer programming (MIP) which allows you to have some integer variables and some continuous ones.

ILPs are significantly more powerful than LPs. In fact, they can be used to solve any general satisfiability (SAT) problem, which is an entire other space of problems. We will see that we can translate any SAT problem into an ILP (although it's typically easier to use a SAT solver).

ILP feasible region geometry

The only difference between ILPs and LPs is that in ILPs, the variables need to be integer (that is, in $\mathbb{Z}$).

This means that only certain points are in the feasible region now, and importantly, that the region is **no longer convex**.



ILP feasible region geometry (cont.)

In general, the number of points in an ILP's feasible region will scale exponentially with the number of decision variables there are, since each decision variable corresponds to another dimension of the feasible region.

For example, if we have an ILP with 100 binary variables $x_1, \ldots, x_{100}$ (binary means $x_i \in \{0, 1\}$), there are 2^{100} feasible solutions.

Of course with linear programming we generally have infinitely many points in the feasible region. So why is solving ILPs more difficult than solving LPs?

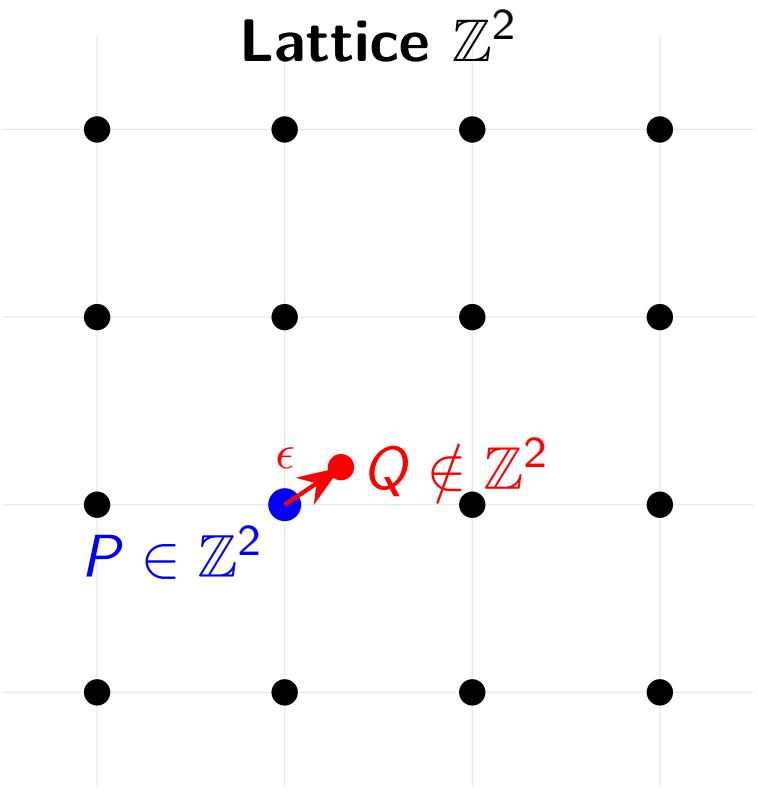
Why ILPs are more difficult to solve than LPs

Convexity is a very useful property: as we saw with LPs, a convex feasible region means that any local optimum is a global optimum.

However, for ILPs, every single feasible point is a "local optimum", because traversing any small distance ϵ outside of that feasible point gives us an infeasible point.

Therefore, we don't have the same property anymore.

Example: $\mathbb{Z}^2$



Assume we have two decision variables and no constraints, giving us the pictured feasible region.

Points P and Q are a small distance ϵ away from each other (not to scale, $\epsilon = 1/N$ where $N >> 0$).

Here, P is a feasible point, while Q is not: the objective value of P will therefore not tell us anything about Q, despite their proximity.

Formalizing difficulty: SAT

There is a well-known "difficult" problem to solve called SAT, in which the satisfiability of a boolean proposition needs to be verified (or the proposition needs to be determined to be unsatisfiable).

This problem has been shown to be **NP-complete**, which means that there is no known method of solving it in polynomial time ("quickly").

To demonstrate that ILP is at least as difficult as SAT, we can demonstrate that it is possible to formulate any SAT problem as an ILP.

Boolean formulas

First, we need to define the SAT problem more thoroughly. We will start by defining boolean formulas, which will take a couple slides.

Boolean formula

A boolean formula ϕ is a formula that has the structure given by the following recursive BNF (Backus-Naur Form) structure, where a is an "atom" (some predefined constant in the associated set of atoms $\mathcal{A}$), and $\top$ and $\bot$ mean "true" and "false" respectively:

$$\phi ::= a \mid \bot \mid \top \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \phi_1 = \phi_2 \mid \phi_1 \rightarrow \phi_2$$

Boolean operators and sufficiency of $\wedge$ and $\neg$

Boolean operator interpretations

The operators in the boolean syntax have the following interpretations, given in terms of their effects on $\top$ and $\bot$:

- $\neg$ (not): $\neg\bot \equiv \top$, and $\neg\top \equiv \bot$
- $\wedge$ (and): $\bot \wedge \top \equiv \top \wedge \bot \equiv \bot \wedge \bot \equiv \bot$; $\top \wedge \top \equiv \top$
- $\vee$ (or): $\bot \vee \top \equiv \top \vee \bot \equiv \top \vee \top \equiv \top$; $\bot \vee \bot \equiv \bot$
- $\rightarrow$ (implies): $\bot \rightarrow \top \equiv \bot \rightarrow \bot \equiv \top \rightarrow \top \equiv \top$; $\top \rightarrow \bot \equiv \bot$
- $=$ (equals): $\bot = \bot \equiv \top = \top \equiv \top$; $\top = \bot \equiv \bot = \top \equiv \bot$

Sufficiency of $\wedge$ and $\neg$

Any boolean formula can be expressed only in terms of operators $\wedge$ and $\neg$, by using the following transformation rules:

- $\phi_1 \vee \phi_2 \equiv \neg(\phi_1 \wedge \phi_2)$
- $\phi_1 \rightarrow \phi_2 \equiv \neg\phi_1 \vee \phi_2$
- $\phi_1 = \phi_2 \equiv (\phi_1 \rightarrow \phi_2) \wedge (\phi_2 \rightarrow \phi_1)$

Satisfiability

A boolean formula ϕ is **satisfiable** if there is some assignment $M : \mathcal{A} \to \{\top, \bot\}$ so that when ϕ is evaluated in terms of the mappings of atoms $a \in \mathcal{A}$, it evaluates to $\top$.

If ϕ is satisfiable we write "ϕ is SAT", otherwise we write "ϕ is UNSAT".

Examples (with $\mathcal{A} = \{p, q\}$):

- p is SAT.
  - We can obtain $\top$ by assigning $p \mapsto \top$.
- $p \wedge \neg p$ is UNSAT.
  - Assigning $p \mapsto \top$ or $p \mapsto \bot$ will always yield a result of $\bot$.
- $(p \vee q) \wedge \neg q$ is SAT.
  - We can assign $q \mapsto \bot$ and $p \mapsto \top$ to satisfy this formula.

Boolean variables as binary ILP variables

So we can rewrite any SAT problem to be in terms of atoms, $\wedge$, and $\neg$.

First, we will need a way to represent atoms.

One of the most common types of variables we will see in ILPs is **binary** variables, which take values in $\{0, 1\}$. There is a direct correspondence between the boolean primitives $\top$ and $\bot$, which should be familiar:

Boolean primitive	Binary value
$\bot$	0
$\top$	1

We can make a variable x (e.g.) binary by requiring that x be an integer in the range $0 \leq x \leq 1$.

ILP constraint correspondence: $\wedge$ and $\neg$

Here is the correspondence for $\neg$:

$$\phi := \neg\psi \equiv x_1 = 1 - x_2, \quad x_1, x_2 \text{ binary}$$

And here is the correspondence for $\wedge$:

$$\phi := \psi_1 \wedge \psi_2 \equiv \begin{cases} x_1 \leq x_2 \\ x_1 \leq x_3 \\ x_1 \geq x_2 + x_3 - 1 \end{cases}, \quad x_1, x_2, x_3 \text{ binary}$$

The method of transforming the $\wedge$ constraint is one case of the McCormick constraints for an n-way product of binary values, which we will discuss in a later lecture along with other ILP modeling tricks.

Summary: SAT reduction

So, over the past few slides we:

- Defined boolean formulas
- Demonstrated that any boolean formula could be rewritten in terms of the operators $\wedge$ and $\neg$
- Defined the SAT problem for boolean formulas
- Demonstrated that any boolean variable can be written as a binary ILP variable
- Demonstrated that any $\neg$ or $\wedge$ condition can be written as an ILP constraint

By writing a SAT problem as an ILP and determining whether there is a feasible solution, we can effectively determine whether or not there is a solution to that SAT problem.

Thus, ILPs are at least as computationally complex as the SAT problem, making them NP-complete.

(This is a simple version of the reduction from SAT, and is not very rigorous, but should be intuitive.)