Solving ILPs

Ben Rosenberg

December 24, 2025

Overview

- We need a new method
- Survey of ILP-solving methods
- Branch and bound
- Symmetry (brief)

LP methods as a means of solving ILPs

In general, LP methods are significantly faster than ILP methods.

So if we can solve an ILP using LP methods, that would be ideal.

However, this is only possible if the **LP-relaxation** has the integrality property.

LP relaxation

The **LP-relaxation** of an ILP is the LP that is formed by removing the constraints that decision variables must be integer.

An ILP solution method

We can't always rely on the integrality property, because most LP relaxations of ILPs will not have this property.

(If the LP relaxation of an ILP has the integrality property, there isn't really a reason to model it as an ILP at all.)

Instead, solvers typically use the following method:

- Solve the LP-relaxation of the ILP
- Explore the neighborhood of the LP-relaxation's solution to find integer feasible points
- Determine the optimum from these nearby points

The method of exploring the neighborhood of the LP-relaxation and determining the optimum of these points is typically **branch and bound**.

Branch and bound

1. If the solution to an LP relaxation is fractional (e.g., $x_1 = 2.5$) we branch on that variable, creating new subproblems where we add new constraints, and splitting the search space into two distinct regions:
   - Left branch: add the constraint $x_1 \leq 2$ (take the floor of the current value)
   - Right branch: add the constraint $x_1 \geq 3$ (take the ceiling of the current value)
2. For every new branch, solve the LP relaxation:
   - This yields an bound (upper or lower depending on which direction the objective function is pointing, max or min) on the ILP, because the LP relaxation will never be worse than the ILP solution (as it is less constrained)
   - If the LP relaxation solution is worse than the best integer solution we have so far, we can "prune" (or "fathom") it, because any children of that node will be more constrained, and therefore not better than the LP relaxation

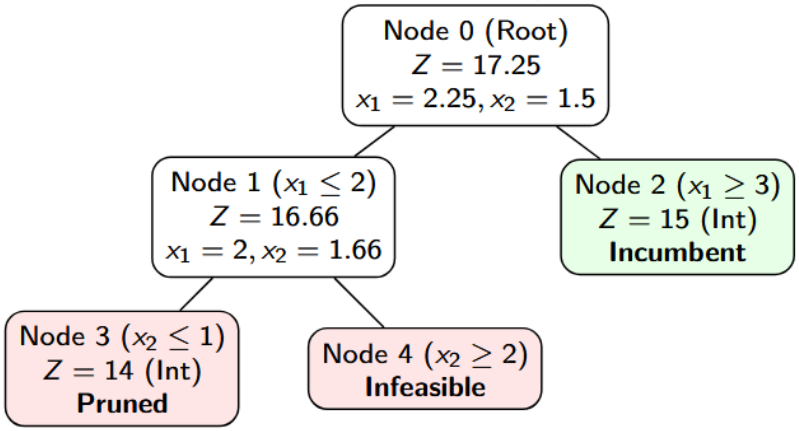
(cont.)

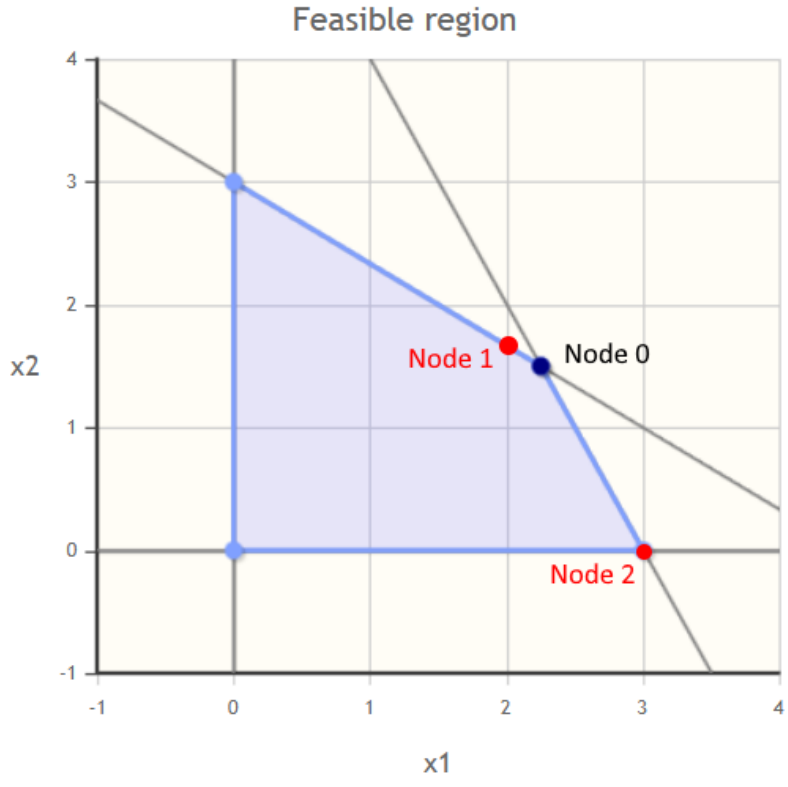
Branch and bound (cont.)

3. We stop expanding a branch (fathom it) in the following cases:
   - Fathom by *integrality*: The LP solution naturally results in integers. We don't need to expand this any more because all the variables are already integers
   - Fathom by *bound*: The LP objective is worse than our current best solution
   - Fathom by *infeasiblity*: The constraints in this branch are impossible to satisfy

Branch and bound example

$$
\begin{aligned}
\max \quad & 5x_1 + 4x_2 \\
s.t. \quad & x_1 + 1.5x_2 \leq 4.5 \\
& 2x_1 + x_2 \leq 6 \\
& x_1, x_2 \geq 0, \text{ integer}
\end{aligned}
$$





Branch and bound example (cont.)

- The first node, Node 0, is what we get from solving the LP relaxation
- We then branch on x_1 to get nodes 1 and 2
- The LP relaxation of Node 1 as a non-integer solution, but the LP relaxation of Node 2 is integer, so we mark Node 2 as the incumbent solution
- We then continue expanding Node 1, this time branching on x_2
- Node 3's objective is worse than the incumbent solution, so we prune it by bound
- Node 4's LP relaxation is infeasible, so we prune it by infeasibility, leaving Node 2 as the optimal ILP solution

Branch and bound: adversarial case

Consider the following ILP:

$$\begin{aligned} \max \quad & x_1 \\ s.t. \quad & 2x_1 + 2x_2 + \cdots + 2x_{101} = 101 \\ & x_1, \ldots, x_{101} \in \{0, 1\} \end{aligned}$$

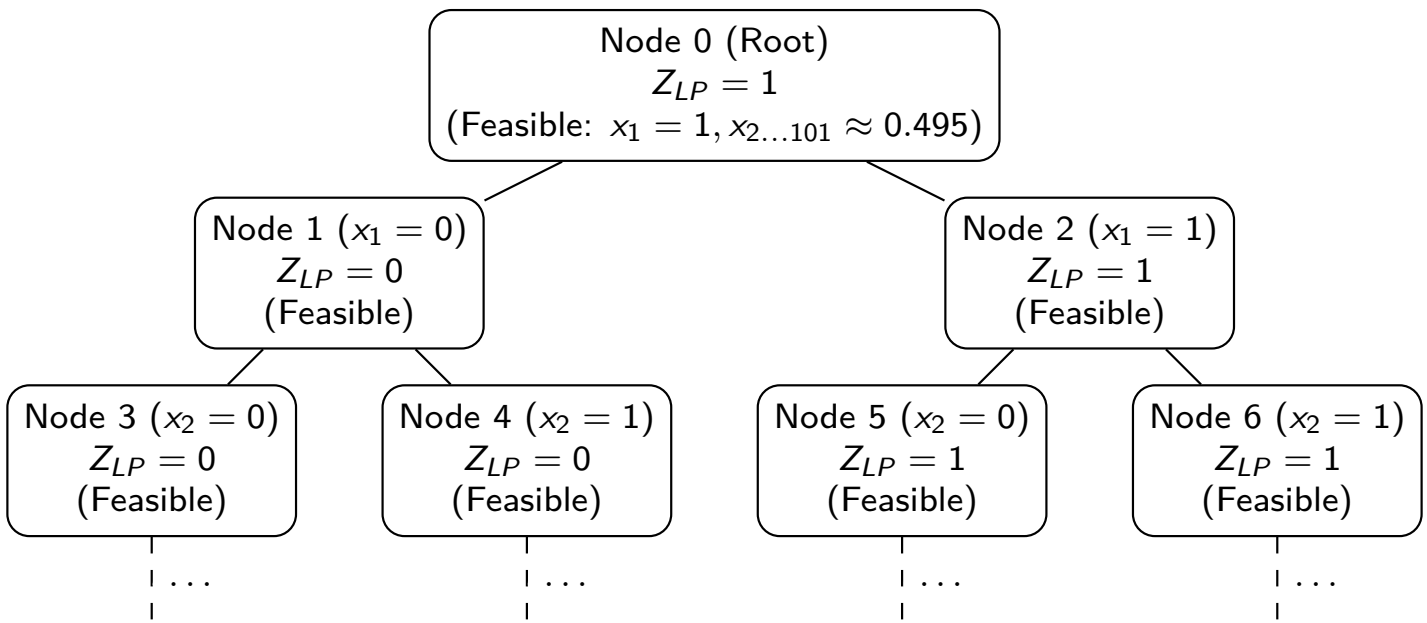
This ILP is clearly infeasible, because we can rewrite the constraint as follows:

$$\begin{aligned} 2(x_1 + x_2 + \cdots + x_{101}) &= 101 \\ x_1 + x_2 + \cdots + x_{101} &= 50.5 \end{aligned}$$

Since $x_1, \ldots, x_{101}$ are integers, we can never get a non-integer result.

Branch and bound: adversarial case (cont.)

The B&B tree for that ILP continues to depth 101 without pruning, before determining that the problem is infeasible across all possible permutations of x and all LP-feasible solutions. The total node count will be around 2^{101}, which is around 10^{30} (very big).



Symmetry (and presolving)

There are a couple ways that solvers might see this problem and avoid getting stuck on it (this is only a few):

1. Presolving - some higher-level method of determining that the problem is infeasible even before trying to solve the initial LP relaxation of it, e.g. by checking the parity of the LHS and RHS
2. **Symmetry breaking** - this problem has symmetry, because all of the variables can be reordered without changing anything. To deal with this, the solver might add some constraint like $x_1 \leq x_2 \leq \cdots \leq x_{101}$, collapsing the tree into a straight path (e.g., around 101 nodes instead of 2^{101})
   - The key thing here is differentiating the variables. This is a common issue in not just ILP but optimization in general, where interchangeable variables double the solution count or feasible region space. This doubling is done for each such pair of interchangeable variables, leading to the exponential explosion that we see on this problem

Other techniques

More advanced solvers could use something like SMT (satisfiability modulo theories) techniques and/or more general constraint programming approaches in parallel, as this ILP contains only binary variables and is very easy to port to another solver input.

SMT solvers, or "satisfiability modulo theories" solvers, are a type of SAT solver that can handle more complicated relationships and primitive datatypes than the boolean algebra expressions we saw previously.

These groups of additional "domain" elements, along with their associated predicates and functions, are each known as a "theory". For example, FOL + arithmetic (FOL stands for first order logic, i.e. boolean algebra with $\forall$ and $\exists$) is an example of a specific theory.

ILP and SMT

ILP with only binary variables is pretty similar to SMT, although SMT solvers are much more efficient for these types of problems. ILP is best suited to large problem instances with integer variables that are not (only) binary.

OR-Tools actually has some of this more general constraint programming/SMT functionality, but we will be sticking to LP, ILP and (maybe) QP in this course.