Math 482: Linear Programming[1] Mikhail Lavrov

Lecture 32: Integer Programming

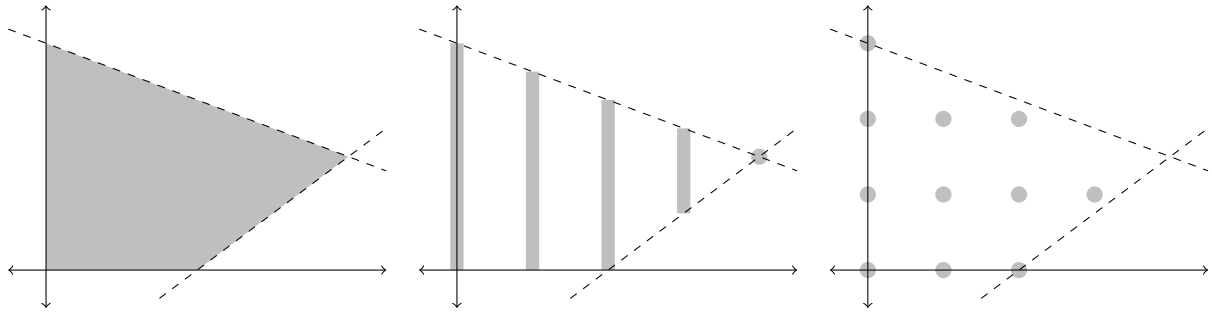
April 24, 2020 *University of Illinois at Urbana-Champaign*

1 Integer linear programming

An integer linear program (often just called an "integer program") is your usual linear program, together with a constraint on some (or all) variables that they must have integer solutions.

We saw these appear earlier in the class when looking at graph theory problems like the bipartite matching problem, but in all of the problems we looked at, we had total unimodularity to save us: we didn't have to think about the integer constraints, because they would hold automatically. This is rare and unusual: typically, the constraints matter.

For example, consider the following three optimization problems:



$$\begin{aligned} \underset{x,y\in\mathbb{R}}{\text{maximize}} \quad & x + y \\ \text{subject to} \quad & 3x + 8y \leq 24 \\ & 3x - 4y \leq 6 \\ & x, y \geq 0 \end{aligned} \qquad \begin{aligned} \underset{x\in\mathbb{Z},y\in\mathbb{R}}{\text{maximize}} \quad & x + y \\ \text{subject to} \quad & 3x + 8y \leq 24 \\ & 3x - 4y \leq 6 \\ & x, y \geq 0 \end{aligned} \qquad \begin{aligned} \underset{x,y\in\mathbb{Z}}{\text{maximize}} \quad & x + y \\ \text{subject to} \quad & 3x + 8y \leq 24 \\ & 3x - 4y \leq 6 \\ & x, y \geq 0 \end{aligned}$$

The first example is an ordinary linear program with optimal solution $(4, \frac{3}{2})$.

The second example is a (mixed) integer program where $(4, \frac{3}{2})$ is still the optimal solution. In fact, here, all vertices of the feasible region have $x \in \mathbb{Z}$; if we know this ahead of time, we can solve the integer program as a linear program.

The last example is an integer program with the same constraints, but the optimal solutions are $(2, 2)$ and $(3, 1)$ instead. Note that we can't even solve the integer program by rounding $(4, \frac{3}{2})$ to the nearest integer; that won't give us a feasible solution.

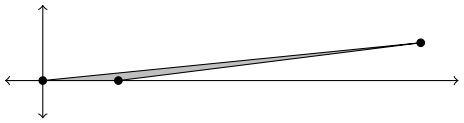
In general, an optimal integer solution can be arbitrarily far from the optimal solution. For example,

[1] This document comes from the Math 482 course webpage: https://faculty.math.illinois.edu/~mlavrov/courses/482-fall-2020.html

consider the region

$$\left\{(x,y) : \frac{x-1}{998} \leq y \leq \frac{x}{1000}, x, y \geq 0\right\}.$$

This region has a vertex at $(x,y) = (500, \frac{1}{2})$, but its only integer points are at $(0,0)$ and $(1,0)$. (A version with $998, 1000$ replaced by $8, 10$ is shown below.)



This is just to illustrate that integer programming is hard, and weird things can happen when we add the integer constraint.

2 Logical constraints

Now let's switch gears away from optimization problems to logic puzzles. Here, a logic puzzle can be a traditional kind, such as Sudoku. However, it also includes combinatorial problems we've already looked at in this class, such as bipartite matching. There are many other important problems out there that also fall in this class, such as graph coloring.

The framework here is that we consider Boolean variables, which can take on only two values: TRUE and FALSE. We combine these with several kinds of logical operations:

- AND: we define X_1 AND X_2 to be TRUE when both X_1 and X_2 are TRUE, and FALSE otherwise.
- OR: we define X_1 OR X_2 to be TRUE if at least one (possibly both) of X_1 or X_2 is TRUE, and FALSE otherwise.
- NOT(): we define NOT(X) to be TRUE if X is FALSE, and FALSE if X is TRUE.

We could define more operations along these lines, but these are already enough to work with.

A Boolean satisfiability problem is the problem of determining if, given a logical expression in variables $X_1, X_2, \ldots, X_n$, such as

$$(X_1 \text{ OR } \text{NOT}(X_2)) \text{ AND } (X_2 \text{ OR } \text{NOT}(X_3)) \text{ AND } (X_3 \text{ OR } X_1),$$

we can assign values to $X_1, X_2, \ldots, X_n$ to make it TRUE.

There is a standard form for logical expressions: conjunctive normal form (CNF). This standard form consists of:

- *literals*, which are either X_i or NOT(X_i),
- combined into *clauses*, which join many literals with the OR operation,
- and finally the logical expression is an AND of multiple clauses.

We will skip the details, but it will be convenient for us to know that any logical expression can be put into conjunctive normal form.

2.1 Logical constraints and integer programming

The reason I bring up logical expression and the satisfiability problem is that integer programming can be used to encode logical constraints. In fact, this is one of the primary ways that we encounter integer programs.

What we do here is represent Boolean variables by integer variables between 0 and 1: as a convention, we represent TRUE by 1 and FALSE by 0. Then, a logical expression in conjunctive normal form can be represented as a system of linear inequalities in these variables:

- If a boolean variable X_i is represented by an integer variable x_i, then $\mathsf{NOT}(X_i)$ can be represented by $(1 - x_i)$.
- Each clause gives us an inequality: we can write $X_1 \mathsf{OR} X_2 \mathsf{OR} \dots \mathsf{OR} X_k$ as
$$x_1 + x_2 + \cdots + x_k \geq 1.$$
Of course, if the clause contains $\mathsf{NOT}(X_i)$ literals, we replace those by $1 - x_i$.
- We always ask for all inequalities in a system to hold, so they're already combined with AND without us having to do anything.
- To ensure that each Boolean variable is represented correctly, we put the constraint $0 \leq x_i \leq 1$ on each integer variable x_i.

If we can find a feasible solution to the resulting integer program, we can determine if the logical expression is satisfiable, which is extremely powerful. This is both good and bad. It's good because it means studying integer programming is extremely worthwhile, and has applications to lots of very hard problems.

It's bad because we can't expect efficient algorithms to solve integer programs—otherwise, we'd have much better ways to solve these very hard problems. But we can do our best.

2.2 Tying together logical and linear constraints

We can even combine these logical constraints with the linear constraints we've already been using.

One way this comes up is in an objective function with fixed costs that don't scale per unit. For example, a banana factory might earn \$1 per banana produced; however, it might have to pay \$1000 to rent a warehouse, no matter how many bananas are stored there.

We can imagine using a binary variable w (an integer variable with the constraint $0 \leq w \leq 1$) to represent whether we rent the warehouse. Then, we can include the cost of rental as a $1000w$ term in our objective function.

We can further use such a binary variable in constraints.

- For example, if x_1, x_2, x_3 represent the number of various products stored in a warehouse, we might have $x_1 + x_2 + x_3 \leq 100$ if the warehouse is rented, but $x_1 = x_2 = x_3 = 0$ otherwise. We can model this (assuming $x_1, x_2, x_3 \geq 0$) by the single inequality
$$x_1 + x_2 + x_3 \leq 100w.$$

- There is also a standard trick to have a constraint that's enforced if $w = 0$ but not if $w = 1$. For example, suppose that we have some bound $x + y \leq 50$ when no warehouse is rented, but no bound on $x + y$ otherwise. We can often model this as

$$x + y \leq 50 + 1000000w$$

where 1000000 is some sufficiently large number. (This works assuming $x + y$ can't possibly get that large, even with no constraints on it.)