Math 3272: Linear Programming[1] Mikhail Lavrov

Lecture 25: Integer programming

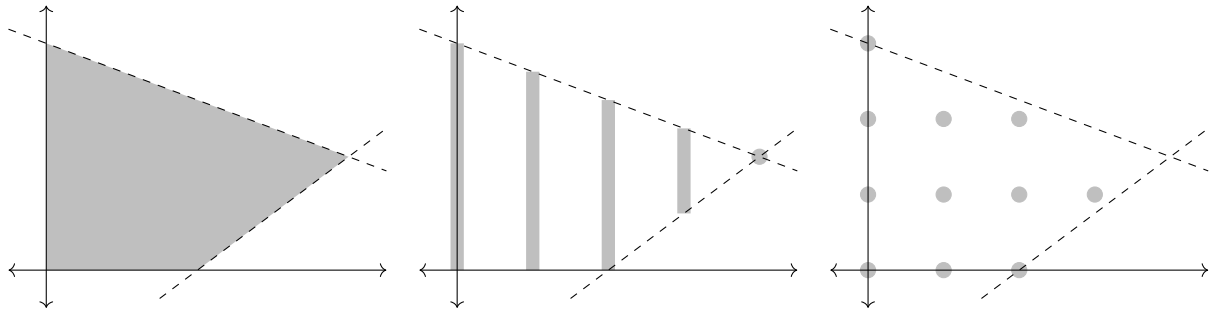
November 15, 2022 *Kennesaw State University*

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 encountered this requirement in some of the applications of network flow problems, but these all had the miraculous property of 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, requiring our variables to be integers *does* change the optimal solution.

For example, consider the following three optimization problems:



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

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.

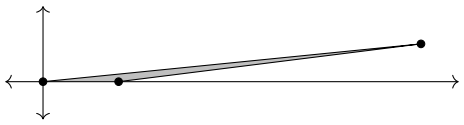
In general, an optimal integer solution can be arbitrarily far from the optimal solution. For example,

[1] This document comes from an archive of the Math 3272 course webpage: http://misha.fish/archive/3272-fall-2022

consider the region

$$\left\{(x,y) \in \mathbb{R}^2 : \frac{x-1}{8} \leq y \leq \frac{x}{10}, x, y \geq 0\right\}.$$

This region (shown below) has a vertex at $(x,y) = (5, \frac{1}{2})$, but its only integer points are at $(0,0)$ and $(1,0)$.



We can replace 8 and 10 by large numbers like 998 and 1000 to get a vertex arbitrarily far away from an integer point. This is just one of the reasons that integer programming is hard, and weird things can happen when we add the integer constraint.

2 Logical constraints

2.1 Sudoku

In a Sudoku puzzle, you have a 9×9 grid (divided into nine 3×3 subgrids called "boxes" that will become important later); your goal is to fill in the cells with numbers from 1 to 9. Some of the cells are pre-filled with numbers to begin with; the other constraint on the solution is that every row, column, and box must contain each of the numbers 1 through 9 exactly once. Below is an example of a Sudoku puzzle.[2] Its solution is given on the last page of these notes, in case you want to solve it yourself without being spoiled.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Suppose that instead of solving this Sudoku ourselves, we want to write an integer program so that an automatic solver can do it for us. How can we do that?

If you ask yourself, "What should my variables be?" there is a tempting but incorrect answer: for each row i and column j, have an integer variable x_{ij} with the constraint $1 \leq x_{ij} \leq 9$ that tells you the number in cell (i,j).

[2] The example is taken from Wikipedia, and its author is listed as Tim Stellmach.

The reason that this is not the right choice of variables is that the constraint "these nine cells contain each value exactly once" cannot be written as a linear inequality in terms of these variables. For example, it is possible for a row of a Sudoku grid to contain the numbers $(1, 2, 3, 4, 5, 6, 7, 8, 9)$, in that order; it is also possible for that row to contain the numbers $(9, 8, 7, 6, 5, 4, 3, 2, 1)$, in that order. However, any set of linear constraints that allows these two points as solutions also allows their midpoint, which is $(5, 5, 5, 5, 5, 5, 5, 5, 5)$: not a valid way to fill in a row of a Sudoku grid!

Instead, we will use *binary* variables. These are a very common choice in integer programs; they are possibly the most common integer variable. What we'll do here is have an integer variable x_{ijk} where i, j, and k are all between 1 and 9. These 729 variables will be bound by constraints $0 \leq x_{ijk} \leq 1$; since they're all integers, then they can only be 0 or 1. The meaning we'll attach to these variables is that x_{ijk} will be 1 if cell (i, j) contains the number k, and 0 otherwise.

There are many constraints to be added to enforce the rules of Sudoku, but they all come in four types:

- In every row, the numbers must all be different; in other words, no value can be repeated. This is a set of 81 constraints: for every row i, and for every value k, we add the constraint

  $$x_{i1k} + x_{i2k} + x_{i3k} + x_{i4k} + x_{i5k} + x_{i6k} + x_{i7k} + x_{i8k} + x_{i9k} = 1$$

  to ensure that exactly one of the cells $(i, 1)$ through $(i, 9)$ contains the value k.

- In every column, the numbers must all be different. This is a very similar-looking set of 81 constraints: for every column j, and for every value k, we add the constraint

  $$x_{1jk} + x_{2jk} + x_{3jk} + x_{4jk} + x_{5jk} + x_{6jk} + x_{7jk} + x_{8jk} + x_{9jk} = 1$$

  to ensure that exactly one of the cells $(1, j)$ through $(9, j)$ contains the value k.

- There are also 81 constraints for the boxes. These also look very similar, though they're slightly harder to describe. For example, for the top left 3×3 box, we will have a constraint

  $$x_{11k} + x_{12k} + x_{13k} + x_{21k} + x_{22k} + x_{23k} + x_{31k} + x_{32k} + x_{33k} = 1$$

  for each k between 1 and 9, which ensures that the value k appears exactly once in that box. We add similar sets of constraints for the other 3×3 boxes.

- The last set of constraints is easier to forget about. It does not enforce the rules of Sudoku as described above; rather, it enforces a rule of common sense that comes from the meaning we attach to those variables. We want to ensure that every cell of the 9×9 grid contains *exactly one number*. Thus, for every row i, and for every column j, we add the constraint

  $$x_{ij1} + x_{ij2} + x_{ij3} + x_{ij4} + x_{ij5} + x_{ij6} + x_{ij7} + x_{ij8} + x_{ij9} = 1.$$

The resulting integer program has 729 variables and 324 constraints (not counting the 729 constraints of the form $x_{ijk} \leq 1$, and the 729 nonnegativity constraints), so it is well out of reach of what we'll be able to solve by hand. However, computers are very good at problems like this: a general-purpose algorithm for solving problems with $\{0, 1\}$-valued variables can solve the Sudoku above basically instantly. Though in general, all known integer programming algorithms take exponential time in the worst case, this takes a while to kick in, especially in special cases.

2.2 Boolean satisfiability problems

The Sudoku integer program above falls under the umbrella of **Boolean satisfiability problems**. The word "Boolean" refers to variables that have two values (in our case, 1 and 0) which mean that some statement is true or false, respectively (in our case, $x_{ijk} = 1$ corresponds to the statement "Cell (i, j) contains value k" being true). The word "satisfiability" means the same thing as "feasibility" in other problems we've solved this semester: we have no objective function, we just want to know if a feasible solution exists.

Boolean satisfiability problems are typically expressed using logical operations: "and", "or", "not", and others. However, all of these can be expressed using linear expressions and linear constraints:

- If x and y are $\{0, 1\}$-valued variables and 1 is interpreted as "true", then $x + y \geq 1$ encodes the constraint "x or y is true".
- If x and y are $\{0, 1\}$-valued variables and 1 is interpreted as "true", then $x + y = 2$ encodes the constraint "x and y are both true".
- If x is a $\{0, 1\}$-valued variable and 1 is interpreted as "true", then $1 - x$ represents the same truth value as "not x".

Typically, Boolean satisfiability problems are written in a standard form called "conjunctive normal form". This standard form consists of:

- **literals**, which are either "x_i" or "not x_i" for some variable x_i;
- combined into **clauses**: sets of literals among which at least one must be true;
- finally, the overall problem is a collection of clauses among which *all* must be true.

We will not get into the weeds of encoding problems in this form, but it is a form that lends itself particularly well to an integer programming representation. Each clause can be represented by a single linear inequality: for example, "x or y or not z" could be written as

$$x + y + (1 - z) \geq 1$$

or $x + y - z \geq 0$.

2.3 Tying together logical and linear constraints

Integer programming, however, is even more expressive than just Boolean satisfiability problems. In the next lecture, we will see an important example where some larger integers (not just 0 and 1) come into play. We can also get a lot of mileage out of using a few $\{0, 1\}$-valued variables inside a larger linear program, which is what we'll see today.

Here are just a few ways these can come into play:

Fixed costs. Not all costs (and profits) in an optimization problem scale continuously per unit. For example, imagine that a factory produces calculators and stores them before shipping. The factory might earn \$50 per calculator[3] produced; however, it might pay \$1000 to rent a warehouse, no matter how many calculators are stored there.

[3] Calculator prices are ridiculous, considering they are often based on technology that was available in the 1980s.

We can use a binary variable y (as before, this will be an integer variable with the constraint $0 \leq y \leq 1$) to represent whether we rent the warehouse. Then, our profits from producing x calculators might be expressed as $50x - 1000y$. It is probably reasonable to treat x as a real-valued variable; though it is impossible to produce $\frac{1}{2}$ of a calculator, the number of calculators will probably be large enough that this won't make a big difference.

Choice of constraints. Of course, if the warehouse rental does not affect anything, we will always set y to 0. We can also switch between different constraints depending on the value of w. To give a simple example, suppose we can produce up to 100 calculators if we don't have a warehouse, but up to 2000 if we do have one. This can be written an inequality

$$x \leq 100 + 1900y.$$

Here, we have an upper bound on x in either case, one bound is just larger than the other. If, on the other hand, the warehouse can store an effectively unlimited number of calculators, then we might have to "make up" an upper bound. That is, we'd write an inequality

$$x \leq 100 + My$$

where M is a number chosen large enough that this constraint won't limit the number of calculators we can produce. For example, based on looking at the rest of the linear program, we might conclude that we'll never have the materials to produce more than 10000 calculators over the time period we're considering, and then we could set $M = 10000$.

Note that in practice, we'll see better behavior when we solve our integer programs if the value of M is not chosen to be gratuitously large. (But we must be careful: if we pick too small an M, we might cut off legitimate solutions.)

Multiple binary variables. There are many ways to make this setup more complicated.

Maybe we can rent multiple storage locations with different costs and different capacities: we could end up a constraint like

$$x \leq 100 + 200y_1 + 500y_2 + 1600y_3$$

where y_1, y_2, y_3 correspond to different storage options. (Our objective value might become something like $50x + 200y_1 + 400y_2 + 1000y_3$ to account for their prices.)

It's possible that the ideas from the previous section might get involved. Maybe options y_1 and y_2 are mutually exclusive, but option y_3 requires option y_2 to be chosen first. We could represent such logical constraints by the inequalities $y_1 + y_2 \leq 1$ and $y_3 \leq y_2$, respectively.

3 Sudoku solution

Here is the solution to the Sudoku puzzle that appeared earlier in these notes:

5	3	*4*	*6*	7	*8*	*9*	*1*	*2*
6	*7*	*2*	1	9	5	*3*	*4*	*8*
1	9	8	*3*	*4*	*2*	*5*	6	*7*
8	*5*	*9*	*7*	6	*1*	*4*	*2*	3
4	*2*	*6*	8	*5*	3	*7*	*9*	1
7	*1*	*3*	*9*	2	*4*	*8*	*5*	6
9	6	*1*	*5*	*3*	*7*	2	8	*4*
2	*8*	*7*	4	1	9	*6*	*3*	5
3	*4*	*5*	*2*	8	*6*	*1*	7	9