Math 482: Linear Programming[1] Mikhail Lavrov

Lecture 11: Worst-Case Running Time

February 17, 2020 *University of Illinois at Urbana-Champaign*

1 The terrible trajectory

What is the worst case for the simplex method? How many pivoting steps do we need?

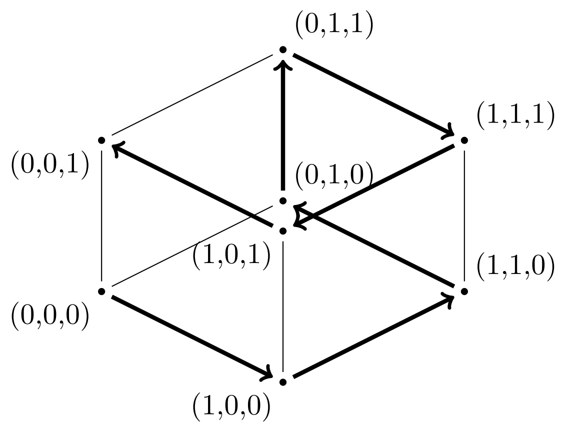
Today, we'll show that for all the pivoting rules we know, the worst case is pretty bad: we can cook up a linear program with only d variables and $2d$ constraints in which we take around 2^d steps. This is approximately as bad as it could possibly get, since there are at most $\binom{2d}{d} < 4^d$ possible basic solutions for such a linear program.

First, consider the following linear program (whose feasible region forms a d-dimensional hypercube):

$$\begin{array}{ll} \underset{\mathbf{x}\in\mathbb{R}^d}{\text{maximize}} & x_d \\ \text{subject to} & 0 \le x_1 \le 1 \\ & 0 \le x_2 \le 1 \\ & \dots \\ & 0 \le x_d \le 1 \end{array}$$

This is not actually the worst case for the simplex method, under any reasonable pivoting rule. The initial basic feasible solution is $\mathbf{x} = \mathbf{0}$, the only variable with positive reduced cost is x_d, and pivoting on x_d gets us to an optimal solution within one step. But tiny modifications will make it much much worse!

First of all, there are some really inefficient trajectories possible in theory: paths we can take going from $(0, 0, \dots, 0, 0)$ to $(0, 0, \dots, 0, 1)$ that visit every other vertex of the hypercube in between. Here is an illustration of such a path in the 3-dimensional case (when the feasible region is a cube):



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

We'll call this path the "terrible trajectory". (Despite the alliteration, this is not established terminology.) The terrible trajectory has a fairly simple recursive definition: to follow it in d-dimensions, first follow the $(d-1)$-dimensional terrible trajectory (keeping $x_d = 0$), then change x_d to 1, then follow the $(d-1)$-dimensional terrible trajectory again, but in reverse.

Second, note that this trajectory is actually kind of close to being reasonable for the linear program we want to solve. Every single step of the terrible trajectory is neutral with respect to the objective value (it does not change x_d), except for one step, which increases it. So it's possible that if we push the corners around a bit, then every single step of the terrible trajectory will increase the objective value. And at that point, we're close to tricking the simplex method into following it.

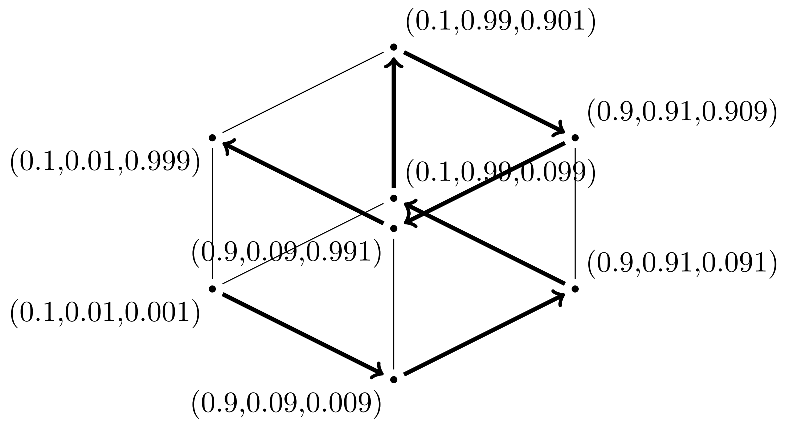
2 Tricking Bland's rule

"Easy mode" is tricking Bland's rule into following the terrible trajectory. To make this happen, we modify the linear program as follows:

$$\begin{array}{ll} \underset{\mathbf{x}\in\mathbb{R}^d}{\text{maximize}} & x_d \\ \text{subject to} & 0.1 \le x_1 \le 1 - 0.1 \\ & 0.1x_1 \le x_2 \le 1 - 0.1x_1 \\ & 0.1x_2 \le x_3 \le 1 - 0.1x_2 \\ & \dots \\ & 0.1x_{d-1} \le x_d \le 1 - 0.1x_{d-1} \end{array}$$

The value 0.1 could be replaced by any reasonably small constant. The smaller we make it, the closer we get to the original cube, and if we set it to 0, we just get back that cube.

Here's what the terrible trajectory looks like for this linear program, in 3 dimensions. (It's a bit of a lie, because with the modification, the feasible region is no longer a perfect cube.)



You can see that in this trajectory, the objective values steadily increase:

$$0.001 < 0.009 < 0.091 < 0.099 < 0.901 < 0.909 < 0.991 < 0.999.$$

It turns out that, with a natural choice of variable ordering, Bland's rule will end up picking this trajectory. Let's first add slack variables to the problem, rewriting $0.1x_{i-1} \leq x_i \leq 1 - 0.1x_{i-1}$ as

$$\begin{cases} 0.1x_{i-1} - x_i + s_i = 0 \\ 0.1x_{i-1} + x_i + s_i' = 1 \end{cases}$$

We have to use the two-phase simplex method for this problem, since $\mathbf{0}$ is not feasible, but let's skip ahead and suppose we arrive at the correct initial basic feasible solution. Here, the variables $x_1, \dots, x_d$ are all basic—and they'll stay basic forever, because none of them can be 0. The variables $s_1', s_2', \dots, s_d'$ also start out being basic, and the variables $s_1, s_2, \dots, s_d$ are nonbasic.

Moreover, in each of the basic feasible solutions we can encounter, exactly one of s_i and s_i' is basic for each i. Moving from one corner to an adjacent one will involve pivoting so that s_i enters the basis and s_i' leaves, or vice versa.

If we put the slack variables in the order

$$s_1, s_1', s_2, s_2', \dots, s_d, s_d'$$

then Bland's rule will pivot on s_1 or s_1' whenever this improves the objective value, which is every other step. In between those, it will pivot on s_2 or s_2' as often as possible, and so on. The entering variables will be

$$s_1, s_2, s_1', s_3, s_1, s_2', s_1'$$

(in 3 dimensions; the pattern continues in higher dimensions) and this traces out exactly the terrible trajectory.

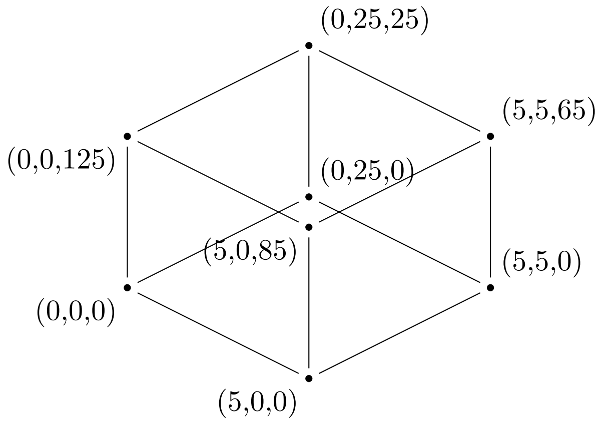
3 The Klee–Minty cube

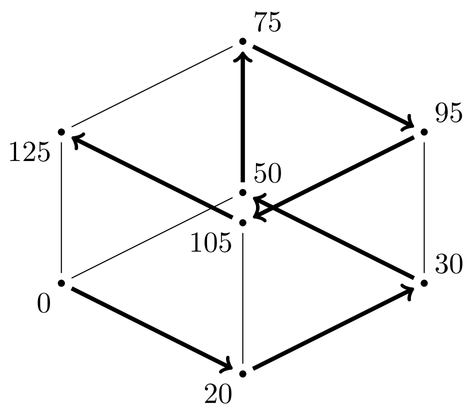
You may think that this example simply exploits a flaw in Bland's rule (which, after all, does not try very hard to pick a good pivot). But a famous linear program called the Klee–Minty cube shows that the highest-reduced-cost pivoting rule is also vulnerable to the "terrible trajectory".

The Klee–Minty linear program in d dimensions is given below (I've written the inequalities backwards to make the pattern easier to see):

$$\begin{array}{ll} \underset{\mathbf{x}\in\mathbb{R}^d}{\text{maximize}} & 2^{d-1}x_1 + 2^{d-2}x_2 + \cdots + x_d \\ \text{subject to} & 5 \geq x_1 \\ & 25 \geq 4x_1 + x_2 \\ & 125 \geq 8x_1 + 4x_2 + x_3 \\ & 625 \geq 16x_1 + 8x_2 + 4x_3 + x_4 \\ & \dots \\ & 5^d \geq 2^d x_1 + 2^{d-1}x_2 + 2^{d-2}x_3 + \cdots + 8x_{d-2} + 4x_{d-1} + x_d \\ & x_1, x_2, \dots, x_d \geq 0. \end{array}$$

This is also shaped kind of like a hypercube in d dimensions (and a cube in 3 dimensions), but it is very distorted. Here is a picture of the "terrible trajectory" for the Klee–Minty cube, with coordinates given in the figure on the left, and their objective values on the right. (For this linear program, the shape of the cube is an incredible lie, but the adjacencies between the corners are the same.)





The best way to understand why this cube tricks the highest-reduced cost rule is to try doing it, and see how the reduced costs change. But essentially, this construction exploits a weakness of the pivoting rule: in this linear program, earlier variables always have much higher reduced costs than later ones, but are actually much worse choices for pivots, because they can't change by as much.

4 Other pivoting rules

Other pivoting rules which we have not discussed exist. Some of them are slightly less vulnerable to this strategy: for example, there is the "best neighbor" pivoting rule, which considers all possible entering variables, and determines which one will lead to the greatest improvement in the objective function.

The best neighbor pivoting rule cannot be fooled by any of the examples we saw today. After all, in both examples, the optimal solution is within one pivot step of the starting solution. The best neighbor pivoting rule will notice this and solve the problem in one step.

However, this does not mean that the best neighbor pivoting rule is always guaranteed to be efficient. It's possible to cook up examples in which it, too, takes an exponentially long time. If the best neighbor pivoting rule is within one step of the optimal solution, it will work well; but if it's within two steps, and the first step does not look promising, the pivoting rule will remain happily oblivious.

There are complicated pivoting rules out there which have a better worst case: on a linear program with n inequalities and d variables, the number of pivot steps they take is bounded by functions such as $C^{\sqrt{d}\log n}$ for some constant C, which is better than exponential. But the upper bound we dream of is "a polynomial function in n and d", and it's an open problem whether any pivoting rule can achieve this.