Linear Programming

Alvan Caleb Arulandu

March 2021

1 Introduction

Linear programming is a topic that doesn't appear frequently in competitive programming. The first notable mention was in the ACM ICPC World Finals 2016 Problem I with only one solver. However, the concepts behind linear programming are quite fascinating to learn; hence this lecture.

2 Optimization

2.1 Problems

In general, an optimization problem involves finding the best solution from a finite or infinte set of feasible solutions. A number of optimization problems are famously NP-hard (ex. the traveling salesman problem). Others are NP-complete (ex. halting problem).

2.2 Descent

Gradient descent and stochastic gradient descent have been recently popularised with the growth of machine learning. In fact, these methods of descend are first-order iterative optimization algorithms; these algorithms use the gradient of a differentiable function to iteratively step towards the relative direction of a local minimum. Despite novel improvements and fine tuning with hyperparameters, these algorithms can not guarantee the finding of a global minimum.

3 The Linear Programming Problem

A much simpler problem can be solved in polynomial time. The linear problem involves optimizing a linear objective function with linear equality and inequality constraints.

3.1 A Simple Case

For simplicity, let's start with $N = 2$ dimensions. We can picture the input space as the cartesian space and a linear function

$$f(x, y) = ax + by$$

that we want to optimize. Now suppose, we want to optimize this function over some region defined by a set of inequalities of the form

$$cx + dy \geq q$$

Note the equality here is important. A viable set of equalities (**constraints**) for the problem will define a closed region in the shape of a polygon known as a **polytope** (not this is not a domain). Linear programming algorithms aim to find the maximum value of f on this **feasible region**.

3.2 Expanding to Higher Dimensions

To clarify, suppose we are given a function f that we want to optimize over a region that will be defined later. Since f is a (multivariable) linear function, we can express it as the following:

$$f(x_1, x_2, \cdots x_n) = w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

Note that f is a scalar field mapping an N dimensional space to a scalar value (cost). Using a more compact vector notation using the dot product,

$$f(\vec{x}) = \vec{w} \cdot \vec{x}$$

This function is known as a **linear objective function**. An objective function is a function that defines some scalar quantity that should be optimized.

3.3 Canonical Form

Maximize $\mathbf{c^T x}$ subject to $A\mathbf{x} \leq \mathbf{b}$ and $\mathbf{x} \geq 0$.

4 Naive Solutions

Consider the following simple linear programming problem.

> A factory manufactures doodads and whirligigs. It costs \$2 and takes 3 hours to produce a doodad. It costs \$4 and takes 2 hours to produce a whirligig. The factory has \$220 and 150 hours this week to produce these products. If each doodad sells for \$6 and each whirligig sells for \$7, then how many of each product should be manufactured this week in order to maximize profit?

4.1 A Naive Solution

Picking this problem apart, we can see 4 constraints,

$$2x + 4y \leq 220$$

$$3x + 2y \leq 150$$

$$x \geq 0$$

$$y \geq 0$$

Graphing this we can see our polytope. A naive approach would be to randomly test ordered pairs in the feasible region.

4.2 A Useful Insight

However, we can use a better method. Let P be the maximum profit in the region.

$$P = p(x, y) = 4x + 3y$$

Solving for one variable gives the line,

$$y = -\frac{4}{3}x + \frac{P}{3}$$

Now, we can graph these lines for different values of P and all points on the line and in the region are plausible solutions. Testing, we can see that the optimal solution passes through a vertex. Indeed, this is true *for all* linear programming problems.

4.3 A Better Naive Solution

So, naturally we can come up with a solution:

1. Find the vertices of the polytope defined by the constraints.
2. Iterate through each vertex and evaluate f at this vertex.
3. Maximize the above value for all vertices, computing the optimal vertex.

But, we face some problems here. How do we find the vertices that are in the polytope? Are intersections always on the perimeter of the polytope?

5 The Simplex Algorithm

The simplex algorithm is perhaps the biggest competitive programming take-away from linear programming. It is a nondeterministic method of solving the linear programming problem. It starts at a vertex of the region and moves iteratively to adjacent vertices where the evalutation of the function is non-decreasing until the optimal solution is found.

5.1 Process

We start by redefining the constraints into equalities using **slack variables**.

$$ax + by \leq c \Rightarrow ax + by + s = c$$

Next we turn our objective function into a equality by introducing another variable,

$$z - f(z) = 0$$

Here we can see that the number of variables is usually greater than the number of equations (so no it's not just Cramer's Rule). Instead we can use a linear algebra technique: **matrix augmentation**. The slack variables (have zero coefficients in the row) are known as **basic variables** and the others are known as **non-basic variables**. The **basic solution** is given by setting all non-basic variables to 0. A **pivot** is performed on a nonzero **pivot element** in a nonbasic column. The exact pivoting rules are ommited due to the number of steps to prevent cycles.

5.2 Algorithm for Maximization

```
def setup():
    make_equalities()
    generate_matrix()

def simplex_algo():
    while True:
        select_variable()
        # Select a pivot row by minimizing the ratio of
        # the augmented elemnt to the target element.
        # This ratio must be positive.
        select_row()
        pivot()
        if all_coeffs_positive():
            break
        else:
            continue

def run_algo():
    setup()
    simplex_algo()
```

5.3 Note

The simplex method only works for minimum or maximum constraints, not a mix as the basic solution is not obtainable. Note that a number of additional steps need to be added to prevent cycles.

5.4 Big-M method

The Big-M method is a modification to the simplex method that allows the simplex method to work with a mix of constraints. It is used to move an infeasible basic solution to a feasible one using **aritificial variables**. The idea is to take the infeasible constraint and add a aritificial variable a_1 with a large coefficient M representing an arbitrarily large constant amount. This allows us to pivot to get a valid basic position.

```
def big_M_simplex_algo():
    add_artificial_vars()
    minimize_to_maximize()
    move_solution()
    simplex_algo()

def run_algo_2():
    setup()
    big_M_simplex_algo()
```

5.5 Complexity

The algorithms has been showed to solve "random" and practical problems in cubic steps. There does exist a family, **Klee-Minty cubes**, that cause exponential steps.

6 Interior Point Algorithms

Unlike the simplex method which traverses edges between vertices, interior-point methods move through the interior of the region.

6.1 Ellipsoid Algorithm

The Ellipsoid algorithm was the first to have a defined worst-case polynomial time which runs in $O(n^6L)$ where n is the number of variables and L is the number of input bits.

6.2 Modern Day Algorithm

Recently, (2019) a algorithm using matrix multiplication has been shown to have a $O(n^{2+\frac{1}{18}}L)$ complexity.

7 Duality

The **dual** of a linear program (LP) is another LP derived from a **primal** LP such that each variable in the primal LP becomes a constraint, every constraint becomes a variable, and objective direction is inversed.

7.1 Weak Duality

The objective value of a dual LP at any feasible solution is the bound on the objective of the primal LP.

7.2 Strong Duality

If the primal LP has an optimal solution, the dual does as well for the same value.

7.3 Uses

The dual problem is sometimes easier to solve (only works for "non-mixed" constraints in LP). Additionally, in machine learning, the dual problem is often solved instead to reduce complexity and computation time.

8 Applications

Linear programming is widely used as many problems can be simplified to linear programming problems. Network flow problems are specialized linear programming problems. Additionally, study into linear programming algorithms has pushed the limitations of optimization theory: duality, decomposition, and copmlexity.

9 Cool Stuff

9.1 Quadratic Programming

If you are interested in gradient descent and machine learning algorithms, check out a related topic known as **quadratic programming**.

9.2 Klee-Minty Cube

The cube for which the simplex method has a exponential complexity is given as follows:

$$x_1 \leq 5$$

$$4x_1 + x_2 \leq 25$$

$$\vdots$$

$$2^D x_1 + 2^{D-1} x_2 + \cdots + x_D \leq 5^D$$

$$x_1 \geq, \cdots x_D \geq 0$$

The simplex algorithm visits all vertices of the deformed cube 2^D in the worst case.

10 Implementation

An linear programming solver implementation using the simplex method can be found in Stanford's ICPC notebook: https://cs.stanford.edu/group/acm/SLPC/notebook.pdf?fbclid=IwAR1mfhgnqJQZoNAApIPR7RWPiB2xABGR4EmaBCd4jK_0vtPgeAk70AHf78g.

11 Problems

- 2-Player Zero-Sum Games
- Max-Flow / Network flow
- Non-simplex Problem: https://codeforces.com/contest/1355/problem/C
- Simplex Method Problem: https://community.topcoder.com/stat?c=problem_statement&pm=3942&rd=6520
- ACM ICPC World Finals 2016 Problem 1: Road Times. http://www.csc.kth.se/~austrin/icpc/finals2016solutions.pdf

12 Sources

- Advanced Algorithms (Codeforces): https://codeforces.com/blog/entry/47511
- Linear Programming (Wikipedia): https://en.wikipedia.org/wiki/Linear_programming
- Linear Programming (MIT OCW): https://www.youtube.com/watch?v=feb9j65Iz4w
- Linear Programming (Brilliant): https://brilliant.org/wiki/linear-programming/
- Dual Linear Programs (Wikipedia): https://en.wikipedia.org/wiki/Dual_linear_program
- Klee-Minty Cube (Wikipedia): https://en.wikipedia.org/wiki/Klee%E2%80%93Minty_cube
- Simplex Method (Stanford): http://stanford.edu/class/cme338/notes/notes06-simplex.pdf