An investigation of the use of linear programming to solve real-world problems

Michael Khalfin
Plainview Old-Bethpage JFK High School
Grade 10
February 24, 2020

ABSTRACT

We introduce linear programming as a relevant discipline and show how to define a linear program using decision variables and constraints. We show how to solve basic linear programming problems by graphing. Then we get into more complicated solutions, and implement methods such as the Simplex Algorithm, the Ellipsoid Algorithm, and various Interior-Points Algorithms. We show variations on the aforementioned algorithms, and offer methods for someone to arrive at the best possible solution. We discuss the importance of technology for solving complex linear programs that would take days or months to do by hand. We parse through functions for one specific program, the IBM ILOG CPLEX Optimization Studio, to illustrate this point. Functions include the Simplex Optimizer, Dual-Simplex Optimizer, and Barrier Optimizer.

PROBLEM STATEMENT

Linear programming is an optimization technique to maximize or minimize a linear function over a set of given **constraints**.

Linear programs have various applications within the real world. Manufacturing industries often use linear programs to analyze their supply chain operations. They look for models with maximum efficiency and minimum operations costs.[1]

Businesses such as Amazon and FedEx use linear programs to optimize delivery routes. This is an extension of the Travelling Salesman Problem (TSP) in graph theory. In the TSP, the traveling salesman needs to go through *n* cities that are plotted on a graph. The objective is to calculate the most efficient route with the least total distance.[2] Transit companies attempt to find routes that minimize both cost and time. In this case, cost and time are the **decision variables.**

The decision variables are the outcomes which will decide the output. Decision variables have a **non-negativity restriction,** meaning that they only take values greater than or equal to 0. This restriction is reasonable since linear programs deal with activities or resources, which only exist over a positive domain.

In order to define a linear programming model, start by identifying the decision variables. Then write the objective function. Let's assume that our decision variables are *X* and *Y*, and *Z* is our maximum or minimum value. Then, $Z = aX \pm bY$. Next, figure out the constraints, and express them as inequalities. These should be in the form $cX \pm dY \leq e$. Finally, account for the non-negativity restriction. Set $X \geq 0$ and $Y \geq 0$.

One of the most common approaches to solving a linear programming problem is by graphing the constraints, and then plugging back into the objective function. Take the equation

$Z = 50X + 120Y$ with constraints $100X + 200Y \leq 10,000$, $10X + 30Y \leq 1200$, and $X + Y \leq 110$ over the domain $X \geq 0$ and the range $Y \geq 0$. (Note: $100X + 200Y \leq 10,000$ simplifies to $X + 2Y \leq 100$, and $10X + 30Y \leq 1200$ simplifies to $X + 3Y \leq 120$.)

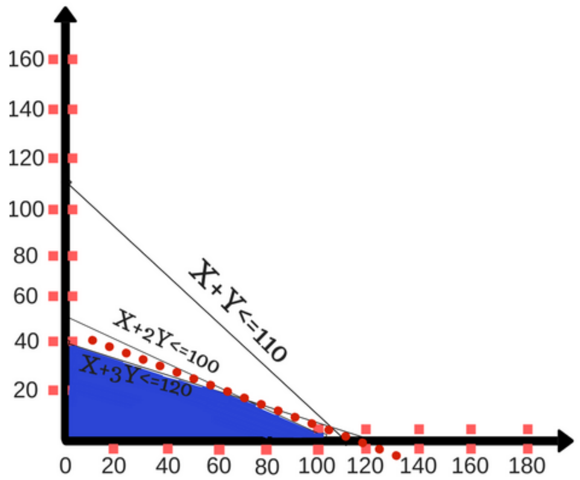

Figure 1. Solving a LP through Graphical method. (https://www.analyticsvidhya.com/blog/2017/02/lintroductory-guide-on-linear-programming-explained-in-simple-english/)

The constraints intersect at the point (60, 20), making this our optimal solution. Plug in 60 and 20 for X and Y to get $Z = 5400$.

Generally, linear programs have more than 2 decision variables. The number of decision variables is always equal to the number of axes. Although 4D, 5D, 6D, etc. data can be visualized in Python and other engines, it is very hard to graph linear programs in more than 3 axes.[3] Thankfully, there are alternative methods to solving linear programs which include the Simplex Algorithm, Ellipsoid Algorithm, and Interior-Point Algorithms.

The idea of the **Simplex Algorithm** (1947) is to start at some "corner" of a feasible region, and then iterate over other corners looking for the optimal position. This method is shown geometrically in Figure 2.

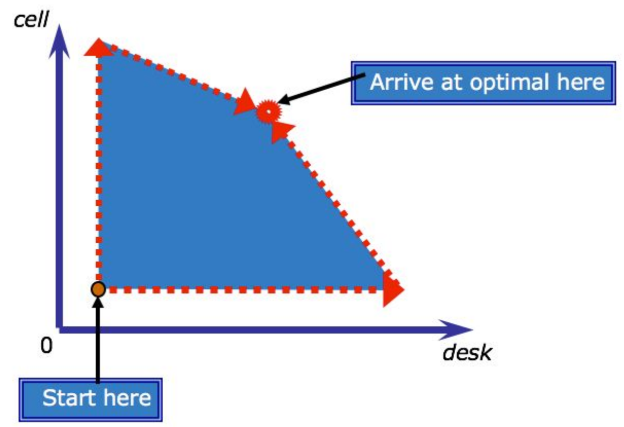

Figure 2. The Simplex Algorithm.
(https://ibmdecisionoptimization.github.io/tutorials/html/Linear_Programming.html)

The **Ellipsoid Algorithm** (1980) is a later model that solves the “feasibility problem” and then enables one to find the solution using a binary search of the objective function. Start with a large ellipse (or ellipsoid, for higher dimensions) that surely contains the feasible region. Continue to shrink the ellipse by a factor of $1 - \frac{1}{n}$ until the center of the ellipse complies with all of the constraints.[4]

Interior-Point Algorithms (1984) work with feasible points in a manner that’s similar to the Simplex Algorithm. However, they actually start within the feasible region, rather than at the corners. One of the more popular algorithms is the **Barrier Optimizer**. The barrier method moves through the interior of a region, using a predictor-corrector to determine its path.[5] The optimizer is part of IBM ILOG CPLEX Optimization Studio, an optimization software package for C++, C#, Java, and Python code, that is one of the leading tools within the field. Python is the go-to language at the moment for machine learning, making this combination efficient towards uncovering new developments in technology.

RELATED RESEARCH

a. Mathematics of the Simplex Algorithm

As discussed earlier, the Simplex Algorithm was the first method to optimize data over a set of given constraints. Starting from a vertex of the feasible region, we iterate over the outer area of the region in order to find the location of the optimal solution.

Pivoting is a process whereby a matrix can yield various solutions for a system of equations. This is the general equation for a system

$$a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1$$
$$a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2$$
$$\dots$$
$$a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m$$

where $m \le n$.[6] A matrix depicting the linear system would look like this:

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} & b_m \end{bmatrix}$$

Figure 3. Standard Matrix.

A matrix is said to be in **reduced echelon form**, or canonical form, if each pivot (defined as the element where $m = n$) is equal to 1, and other elements in the same column as the pivot have a value of 0.

$$\begin{bmatrix} 1 & 0 & 0 & b_1 \\ 0 & 1 & 0 & b_2 \\ 0 & 0 & 1 & b_3 \end{bmatrix}$$

Figure 4. Example of Reduced Echelon Form

The feasible solutions for the constraints are derived by pivoting in a matrix, and then solving for feasible points. There are often multiple solutions for our domain (disregard negative solutions), and we must find the optimal solution on each corner by plugging back into the equation.

Consider the equation Z = 100X + 50Y - 25Z with the following constraints:

$$\begin{aligned} a_1 + a_2 - a_3 &= 5 \\ 2a_1 - 3a_2 + a_3 &= 3 \\ -a_1 + 2a_2 - a_3 &= -1 \end{aligned}$$

We can form an **augmented tableau** by using the following notation:

$$\begin{array}{ccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & \\ 1 & 0 & 0 & 1 & 1 & -1 & 5 \\ 0 & 1 & 0 & 2 & -3 & 1 & 3 \\ 0 & 0 & 1 & -1 & 2 & -1 & -1 \end{array}$$

Multiply each row by a constant and then add them repeatedly until you obtain an answer in reduced echelon form. You derive the following matrix:

$$\begin{array}{ccccccc} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & \\ 1 & -1 & -2 & 1 & 0 & 0 & 4 \\ 1 & -2 & -3 & 0 & 1 & 0 & 2 \\ 1 & -3 & -5 & 0 & 0 & 1 & 1 \end{array}$$

The solutions are $x_4 = 4$, $x_5 = 2$, and $x_6 = 1$.

These are *possible* solutions of the linear system. There are 4 options available for reduced echelon form. Obtain all of these points and then see which one yields a maximum or a minimum after plugging into the objective function.

b. Mathematics of the Ellipsoid Algorithm

Although the Ellipsoid Algorithm is actually slower than the Simplex Algorithm, it was the first step towards coming up with other approaches to linear programs.[7] It can be thought of as the intermediary between the Simplex Algorithm and later Interior-Point Algorithms.

The problem takes the following form:

maximize $c^T x$
$Ax \leq b$
$x \geq 0$
where A is a m x n real constraint matrix and x, $c \in R^n$.

R^n is an infinite set of all the space available geometrically. A general ellipsoid in R^n can be expressed as $(x-a)^T B(x-a) \leq 1$ where B is a positive matrix. Start by checking $c^T x$ against an estimate of the maximum value c_0. Set $c^T x \geq c_0$. If the system is infeasible, the optimum must be less than c_0. Decrease c_0 by a factor of 2, and repeat until you obtain the feasible region $[c_0/n, c_0)$. This is the easiest way to work with the "feasibility problem" mathematically.

Other variants also use the **separating oracle** to solve this issue. A polynomial time separating oracle either tells that $x \in K$ (this can be thought of as the null hypothesis, H_0) or returns a hyperplane separating x from K (the alternative hypothesis).

Consider a bounding ellipsoid E_0 that takes into account all constraints for a linear program. The minimum volume ellipsoid surrounding a half ellipsoid can be calculated in polynomial time using the following equation:

$$Vol(E_{i+1}) \leq (1 - \tfrac{1}{2n})\ Vol(E_i)$$

Thus, using a *while* loop in a linear program, this repeated equation makes the ellipsoid algorithm very beneficial theoretically. However, this would have to be repeated so many times that the Ellipsoid Algorithm becomes impractical for real-world usage.

c. Instances of Interior-Point Algorithms

Interior-Point Algorithms are faster and more efficient than the Simplex Algorithm or the Ellipsoid Algorithm. They work by formulating large linear programs as nonlinear problems and solving them with various modifications of nonlinear programs.

Primarily, we should understand **notions**. For any function f : $R^n \rightarrow R$ the gradient ∇f of f is defined as the vector of its partial derivatives.[8]

$$\nabla f(x) := \begin{bmatrix} \dfrac{\partial f(x)}{\partial x_1} \\ \vdots \\ \dfrac{\partial f(x)}{\partial x_n} \end{bmatrix}$$

Figure 5. Vector of Partial Derivatives

The matrix of second partial derivatives (**Hessian Matrix**) takes on the following form:

$$\nabla^2 f(x) = H(x) := \begin{bmatrix} \dfrac{\partial^2 f(x)}{\partial x_1 \partial x_1} & \cdots & \dfrac{\partial^2 f(x)}{\partial x_1 \partial x_2} \\ \cdots & & \cdots \\ \dfrac{\partial^2 f(x)}{\partial x_n \partial x_1} & \cdots & \dfrac{\partial^2 f(x)}{\partial x_n \partial x_n} \end{bmatrix}$$

Figure 6. Matrix of Second Partial Derivatives

Assume that any function f is twice continuously differentiable for the gradient and matrix to be defined. Let the class of the function be equal to C^2.

Notions become significant through **Newton's Method,** where orbits of greater degree converge linearly, forming the basis for a solution using interior points. The linear approximation of a function *f* can be represented using the formula:

$$\bar{f}(x) = f(x_0) + \nabla f(x_0)^T (x - x_0)$$

Similarly the quadratic approximation of a function *f* can be represented using:

$$g(x) = f(x_0) + \nabla f(x_0)^T (x - x_0) + \tfrac{1}{2}(x - x_0)^T H(x_0)(x - x_0)$$

Newton's Method starts with a point (x_0) on all curves and then evaluates new points (x_1, x_2, … , x_n) until it finds convergence. At this point there is a solution to the linear program.

This solution still has to be evaluated for relevance (does it make sense in the context of the problem?) and time (efficiency). If the appropriate conditions are met, then this point is a solution to the linear system.

There are other ways to solve Interior-Point Algorithms. Many of these methods are complicated and only suitable for proficient mathematicians with a background in multivariable calculus. These include artificial primal and dual linear programs.[9] They are noticeably more challenging than the Simplex Algorithm or Ellipsoid Algorithm, which may explain why they took so many more years to implement.

d. Importance of Technology for Solving Linear Programs

Linear programs are solvable in a variety of programming languages and solvers, including but not limited to Pyomo, AMPL, LINDO, MATLAB, and OptimJ. For this paper, we will be focusing on the CPLEX Optimizer for solving linear programs. We will look at 3 methods, the **Simplex Optimizer**, **Dual-Simplex Optimizer**, and **Barrier Optimizer**. The aim is to show the importance of technology for linear programming, so we do not go into too much depth about what this means on the computer science side of things.

The Simplex Optimizer works identically to the Simplex Algorithm described in the previous section of this paper. The Dual-Simplex Optimizer uses an important mathematical concept, **duality**, to solve linear programs. Every linear program has a dual problem: for maximization problems, this is the minimization problem. On the other hand, for minimization problems this is the issue of maximization. The Dual Simplex Algorithm implicitly uses the dual to arrive at an optimal solution faster than the original problem.

Primal (P):

$$\begin{array}{l}
\max z = c_1x_1 + c_2x_2 + \ldots + c_nx_n \\
\text{s.t.} \\
\quad a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\
\quad a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\
\qquad \vdots \qquad\quad \vdots \qquad\quad \vdots \\
\quad a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m \\
\quad x_j >= 0 \ (j = 1, 2, \ldots, n)
\end{array}$$

Dual (D):

$$\begin{array}{l}
\min w = b_1y_1 + b_2y_2 + \ldots + b_my_m \\
\text{s.t.} \\
\quad a_{11}y_1 + a_{21}y_2 + \ldots + a_{m1}y_m \geq c_1 \\
\quad a_{12}y_1 + a_{22}y_2 + \ldots + a_{m2}x_m \geq c_2 \\
\qquad \vdots \qquad\quad \vdots \qquad\quad \vdots \\
\quad a_{1n}y_1 + a_{2n}y_2 + \ldots + a_{mn}y_m \geq c_n \\
\quad y_i >= 0 \ (i = 1, 2, \ldots, m)
\end{array}$$

Figure 7. Primal and Dual Pair.
(https://ibmdecisionoptimization.github.io/tutorials/html/Linear_Programming.html#Algorithms-for-solving-LPs)

Finally, the Barrier Optimizer is an example of an Interior-Point Method. It uses a predictor-corrector algorithm to constantly adjust its movement throughout the center of a feasible region. CPLEX algorithms give the user power to implement different techniques to solve linear problems, and it’s the programmers decision how to incorporate them.

Now imagine what would happen if we eliminated technology for linear programming. Although we would still be able to solve simple problems using graphing or the Simplex Algorithm, complicated problems would become virtually impossible to solve on paper. The Ellipsoid Algorithm uses a while loop to apply a formula over and over. Doing the same calculations by hand would become a tedious process that would take hours upon hours of handiwork. Interior-Point algorithms are ingrained in technology. They are applicable in programming languages and solvers such as the CPLEX Optimizer. Take the optimizers away and linear programs are virtually impossible to solve by hand for many dimensions and constraints.

CONCLUSION

We solved linear equations using the

- Simplex Algorithm
- Ellipsoid Algorithm, and
- Interior-Point algorithms

Higher order linear programs[10] include

- Integral linear programs, and
- Mixed fractions linear programs

Additionally, there are interior-point algorithms that we did not solve.

These should be considered for further research.

Works Cited

1. Analytics Vidhya Content Team. "Introductory guide on Linear Programming for (aspiring) data scientists." *Analytics Vidhya*, 28 Feb. 2017, https://www.analyticsvidhya.com/blog/2017/02/lintroductory-guide-on-linear-programming-explained-in-simple-english/. Accessed 12 Dec. 2019.
2. Weisstein, Eric W. "Traveling Salesman Problem." *Wolfram Mathworld*, http://mathworld.wolfram.com/TravelingSalesmanProblem.html. Accessed 12 Dec. 2019.
3. Ostwal, Prasad. "Multi-dimension plots in Python — From 3D to 6D." *Medium.com*, 28 May, 2019, https://medium.com/@prasadostwal/multi-dimension-plots-in-python-from-2d-to-6d-9a2bf7b8cc74. Accessed 12 Dec. 2019.
4. Kingsford, Carl, and Sleator, Danny. "Lecture #18: LP Algorithms, and Seidel's Algorithm." *Carnegie Mellon University*, 31 Oct. 2017, https://www.cs.cmu.edu/~15451-f17/lectures/lec18-lp2.pdf.
5. "Tutorial: Linear Programming, (CPLEX Part 1)." *IBM Decision Optimization*, https://ibmdecisionoptimization.github.io/tutorials/html/Linear_Programming.html. Accessed 12 Dec. 2019.
6. Luenberger, David G., and Ye, Yinyu. "The Simplex Method." *Linear and Nonlinear Programming,* vol. 4, no. 1, 2008, pp. 49-88. *Universitat Autònoma de Barcelona*, http://mat.uab.cat/~alseda/MasterOpt/310trialtext.pdf.
7. Arora, Sanjeev. "The Ellipsoid Algorithm for Linear Programming." *Princeton University*, Fall 2005, https://www.cs.princeton.edu/courses/archive/fall05/cos521/ellipsoid.pdf.
8. Fukuda, Komei, and Adjiashvili, David. "Chapter 10: Interior-Point Methods for Linear Programming." *ETH Zurich*, Fall 2011, http://www-oldurls.inf.ethz.ch/personal/fukudak/lect/opt2011/aopt11note4.pdf.
9. Kojima, Masakazu, Shinji, Mizuno, and Yoshise, Akiko. "Chapter 2: A Primal-Dual Interior Point Algorithm for Linear Programming." *Progress in Mathematical*

Programming, 1989, pp.29-47. https://www.researchgate.net/profile/Akiko_Yoshise/publication/234810405_A_Primal-Dual_Interior_Point_Algorithm_for_Linear_Programming/links/0c960526a79f5f2ab2000000/A-Primal-Dual-Interior-Point-Algorithm-for-Linear-Programming.pdf.

10. Aneja, Y.P., Chandrasekaran, R., and Nair, K.P.K. "Classes of Linear Programs with Integral Optimal Solutions." *Mathematical Programming Essays in Honor of George B. Dantzig Part I*, vol. 24, no. 1, 2009, pp. 225-237. https://link.springer.com/chapter/10.1007/BFb0121053.