Linear Programming[1]

Fei Li

March 5, 2012

[1] With references of "Algorithms" by S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani.

Many of the problems for which we want algorithms are *optimization tasks*. We seek a solution that

1. satisfies certain constraints (for instance, the path must use edges of the graph and lead from s to t, the tree must touch all nodes, the subsequence must be increasing); and
2. is the best possible, with respect to some well-defined criterion, among all solutions that satisfy these constraints.

Linear programming describes a broad class of optimization tasks in which both the constraints and the optimization criterion are linear functions.

Definition (Linear Programming)

In a linear programming problem we are given a set of variables, and we want to assign **real values** to them so as to (1) *satisfy a set of linear equations and/or linear inequalities* involving these variables and (2) *maximize or minimize a given linear objective function*.

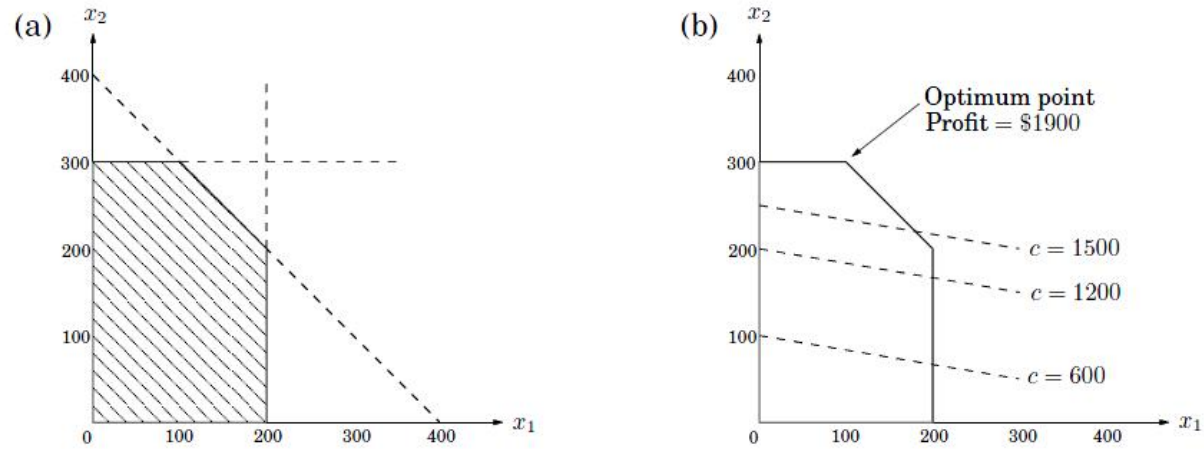
An Example: Geometrical Interpretation

There are two products: Pyramide and Nuit. How much of each should it produce to maximize profits? Let us say it makes x_1 boxes of Pyramide per day, at a profit of \$1 each, and x_2 boxes of Nuit, at profit of \$6 apiece; x_1 and x_2 are unknown values that we wish to determine. There are also some constraints on x_1 and x_2. First, the daily demand for these products is limited to at most 200 boxes of Pyramide and 300 boxes of Nuit. Also, the current workforce can produce a total of at most 400 boxes per day. What are the optimal levels of production?

$$
\begin{aligned}
\text{max} \quad & x_1 + 6x_2 \\
\text{subject to} \quad & x_1 \leq 200 \\
& x_2 \leq 300 \\
& x_1 + x_2 \leq 400 \\
& x_1,\ x_2 \geq 0
\end{aligned}
$$

A linear equation in x_1 and x_2 defines a line in the two-dimensional (2D) plane, and a linear inequality designates a *half-space*, the region on one side of the line. Thus the set of all feasible solutions of this linear program, that is, the points $(x_1,\ x_2)$ which satisfy all constraints, is the intersection of five half-spaces. It is a convex polygon.

Figure 7.1 (a) The feasible region for a linear program. (b) Contour lines of the objective function: $x_1 + 6x_2 = c$ for different values of the profit c.



We want to find the point in this polygon at which the objective function — the profit — is maximized. The points with a profit of c dollars lie on the line $x_1 + 6x_2 = c$, which has a slope of $-1/6$. As c increases, this "profit line" moves parallel to itself, up and to the right. Since the goal is to maximize c, we must move the line as far up as possible, while still touching the feasible region. The optimum solution will be the very last feasible point that the profit line sees and must therefore be a vertex of the polygon.

It is a general rule of linear programs that the optimum is achieved at a vertex of the feasible region. The only exceptions are cases in which there is no optimum; this can happen in two ways:

1. *The linear program is infeasible*; that is, the constraints are so tight that it is impossible to satisfy all of them. For instance,

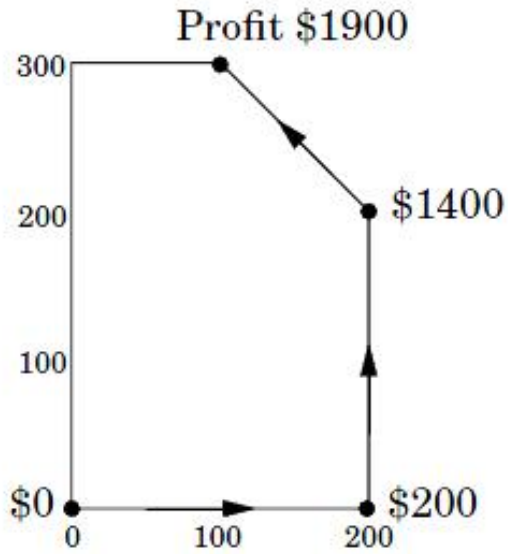
$$x \leq 1, \ x \geq 2.$$

2. *The constraints are so loose that the feasible region is unbounded*, and it is possible to achieve arbitrarily high objective values. For instance,

$$\begin{aligned} \max \quad & x_1 + x_2 \\ & x_1, \ x_2 \geq 0 \end{aligned}$$

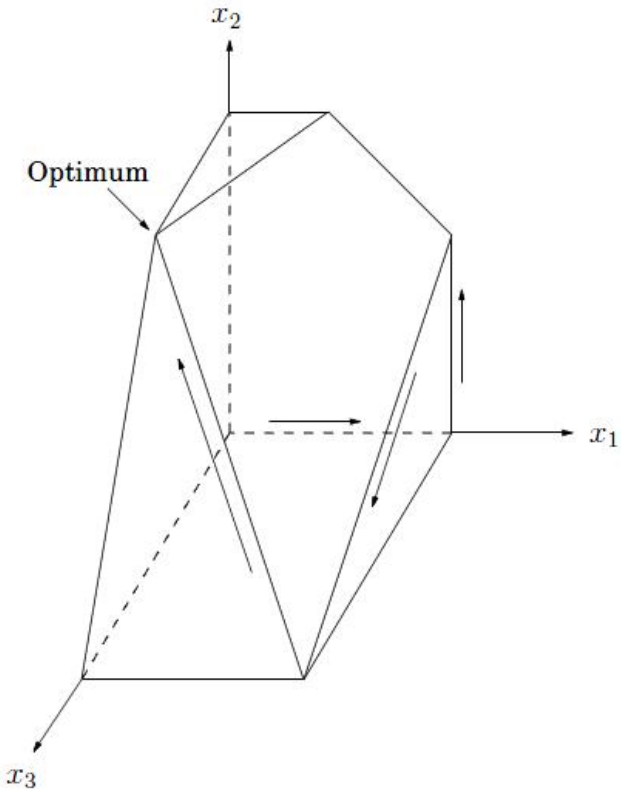
Simplex Method

Linear programs can be solved by the simplex method, devised by **George Dantzig** in 1947. Briefly, this algorithm starts at a vertex, in our case perhaps (0, 0), and repeatedly looks for an adjacent vertex (connected by an edge of the feasible region) of better objective value. In this way it does *hill-climbing* on the vertices of the polygon, walking from neighbor to neighbor so as to steadily increase profit along the way. Here's a possible trajectory.



Upon reaching a vertex that has no better neighbor, simplex declares it to be optimal and halts. Why does this *local* test imply *global* optimality? By simple geometry — think of the profit line passing through this vertex. Since all the vertex's neighbors lie below the line, the rest of the feasible polygon must also lie below this line.

The feasible polyhedron for a three-variable linear program.



$$
\begin{aligned}
\max \quad & x_1 + 6x_2 + 13x_3 \\
& x_1 \leq 200 \\
& x_2 \leq 300 \\
& x_1 + x_2 + x_3 \leq 400 \\
& x_2 + 3x_3 \leq 600 \\
& x_1,\ x_2,\ x_3 \geq 0
\end{aligned}
$$

How would the simplex algorithm behave? As before, it would move from vertex to vertex, along edges of the polyhedron, increasing profit steadily. The optimal point is $(0,\ 300,\ 100)$ with a total profit of \$3,100.

$$
\begin{aligned}
\max \quad & x_1 + 6x_2 + 13x_3 \\
& x_1 \leq 200 \\
& x_2 \leq 300 \\
& x_1 + x_2 + x_3 \leq 400 \\
& x_2 + 3x_3 \leq 600 \\
& x_1,\ x_2,\ x_3 \geq 0
\end{aligned}
$$

Here is why you should believe that $(0,\ 300,\ 100)$, with a total profit of \$3100, is the optimum: Look back at the linear program. Add the second inequality to the third, and add to them the fourth multiplied by 4. The result is the inequality $x_1 + 6x_2 + 13x_3 \leq 3100$.

Do you see? This inequality says that no feasible solution (values $x_1,\ x_2,\ x_3$) satisfying the constraints) can possibly have a profit greater than 3100. So we must indeed have found the optimum! The only question is, where did we get these mysterious multipliers $(0,\ 1,\ 1,\ 4)$ for the four inequalities?

We will see that it is always possible to come up with such multipliers by solving another LP! Except that (it gets even better) we do not even need to solve this other LP, because it is in fact so intimately connected to the original one — it is called the *dual* — that solving the original LP solves the dual as well!

Exercises

This time, our company makes handwoven carpets, a product for which the demand is extremely seasonal. Our analyst has just obtained demand estimates for all months of the next calendar year: d_1, d_2, ..., d_{12}. As feared, they are very uneven, ranging from 440 to 920. Here is a quick snapshot of the company. We currently have 30 employees, each of whom makes 20 carpets per month and gets a monthly salary of \$2,000. We have no initial surplus of carpets.
How can we handle the fluctuations in demand? There are three ways:

1. *Overtime*, but this is expensive since overtime pay is 80% more than regular pay. Also, workers can put in at most 30% overtime.
2. *Hiring and firing*, but these cost \$320 and \$400, respectively, per worker.
3. *Storing surplus production*, but this costs \$8 per carpet per month. We currently have no stored carpets on hand, and we must end the year without any carpets stored.

The objective function is to minimize the total cost.

Variants of Linear Programming

A general linear program has many degrees of freedom.

1. It can be either a maximization or a minimization problem.
2. Its constraints can be equations and/or inequalities.
3. The variables are often restricted to be nonnegative, but they can also be unrestricted in sign.

These various LP options can all be reduced to one another via simple transformations.
By applying these transformations we can reduce any LP (maximization or minimization, with both inequalities and equations, and with both nonnegative and unrestricted variables) into an LP of a much more constrained kind that we call the *standard form*, in which the **variables are all nonnegative**, the **constraints are all equations**, and the **objective function is to be minimized**.

Transformation 1

To turn a maximization problem into a minimization (or vice versa), just **multiply the coefficients** of the objective function by -1.

Transformation 2

To turn an inequality constraint like $\sum_{i=1}^{n} (a_i \cdot x_i) \leq b$ into an equation, introduce a new variable s and use $\sum_{i=1}^{n} (a_i \cdot x_i) + s = b$ and $s \geq 0$
This s is called the *slack variable* for the inequality. To change an equality constraint into inequalities is easy: **rewrite** $a \cdot x = b$ **as the equivalent pair of constraints** $a \cdot x \leq b$ **and** $a \cdot x \geq b$.

Transformation 3

To deal with a variable x that is unrestricted in sign, do the following:

1. Introduce two **nonnegative variables,** $x^+,\ x^- \geq 0$.
2. **Replace** x, wherever it occurs in the constraints or the objective function, **by** $x^+ - x^-$.
   This way, x can take on any real value by appropriately adjusting the new variables. More precisely, any feasible solution to the original LP involving x can be mapped to a feasible solution of the new LP involving $x^+,\ x^-$, and vice versa.

Duality

We have seen that in networks, flows are smaller than cuts, but the maximum flow and minimum cut exactly coincide and each is therefore a certificate of the other's optimality. Remarkable as this phenomenon is, we now generalize it from maximum flow to **any** problem that can be solved by linear programming! It turns out that every linear maximization problem has a **dual** minimization problem, and they relate to each other in much the same way as flows and cuts.

$$\begin{aligned} \max \quad & x_1 + 6x_2 \\ & x_1 \leq 200 \\ & x_2 \leq 300 \\ & x_1 + x_2 \leq 400 \\ & x_1,\ x_2 \geq 0 \end{aligned}$$

Simplex declares the optimum solution to be $(x_1,\ x_2) = (100,\ 300)$, with objective value 1900. Can **this answer be checked** somehow?

1. Suppose we take the first inequality and add it to six times the second inequality. We get $x_1 + 6x_2 \leq 2000$.
2. Multiplying the three inequalities by 0, 5, and 1, respectively, and adding them up yields $x_1 + 6x_2 \leq 1900$.

The multipliers $(0,\ 5,\ 1)$ magically constitute a certificate of optimality!

It is remarkable that such a certificate exists for this LP — and even if we knew there were one, how would we systematically go about finding it? Let us describe what we expect of these three multipliers, call them $y_1,\ y_2,\ y_3$.

Multiplier	Inequality
y_1	$x_1 \leq 200$
y_2	$x_2 \leq 300$
y_3	$x_1 + x_2 \leq 400$

After the multiplication and addition steps, we get the bound:

$$(y_1 + y_3)x_1 + (y_2 + y_3)x_2 \leq 200y_1 + 300y_2 + 400y_3.$$

We want the left-hand side to look like our objective function $\max x_1 + 6x_2$ so that the right-hand side is an upper bound on the optimum solution. For this we need $y_1 + y_3$ to be 1 and $y_2 + y_3$ to be 6. Come to think of it, it would be fine if $y_1 + y_3$ were larger than 1 — the resulting certificate would be all the more convincing. Thus, we get an upper bound

$$x_1 + 6x_2 \leq 200y_1 + 300y_2 + 400y_3, \quad \text{if } \begin{cases} y_1,\ y_2,\ y_3 & \geq 0 \\ y_1 + y_3 & \geq 1 \\ y_2 + y_3 & \geq 6 \end{cases}$$

Primal:

$$\begin{aligned} \max \quad & x_1 + 6x_2 \\ & x_1 \leq 200 \\ & x_2 \leq 300 \\ & x_1 + x_2 \leq 400 \\ & x_1,\ x_2 \geq 0 \end{aligned}$$

Simplex declares the optimum solution to be $(x_1,\ x_2) = (100,\ 300)$, with objective value 1900.

Dual:

$$\begin{aligned} \min \quad & 200y_1 + 300y_2 + 400y_3 \\ & y_1 + y_3 \geq 1 \\ & y_2 + y_3 \geq 6 \\ & y_1,\ y_2,\ y_3 \geq 0 \end{aligned}$$

The multipliers $(0,\ 5,\ 1)$ constitute a certificate of optimality!

Theorem (Duality theorem)

If a linear program has a bounded optimum, then so does its dual, and the two optimum values coincide

Example

One can solve the shortest-path problem by the following "analog" device: Given a weighted undirected graph, build a *physical model* of it in which each edge is a string of length equal to the edge's weight, and each node is a knot at which the appropriate endpoints of strings are tied together. Then to find the shortest path from s to t, just **pull s away from t until the gadget is taut**. It is intuitively clear that this finds the shortest path from s to t.
By pulling s away from t we solve the dual of the shortest-path problem! This dual has a very simple form, with one variable x_u for each node u:

$$\begin{aligned} \max \quad & x_S - x_T \\ & |x_u - x_v| \leq w_{u,v}, \ \forall e = \{u, \ v\} \end{aligned}$$

In words, the dual problem is to stretch s and t as far apart as possible, subject to the constraint that the endpoints of any edge $\{u, \ v\}$ are separated by a distance of at most $w_{u,v}$.

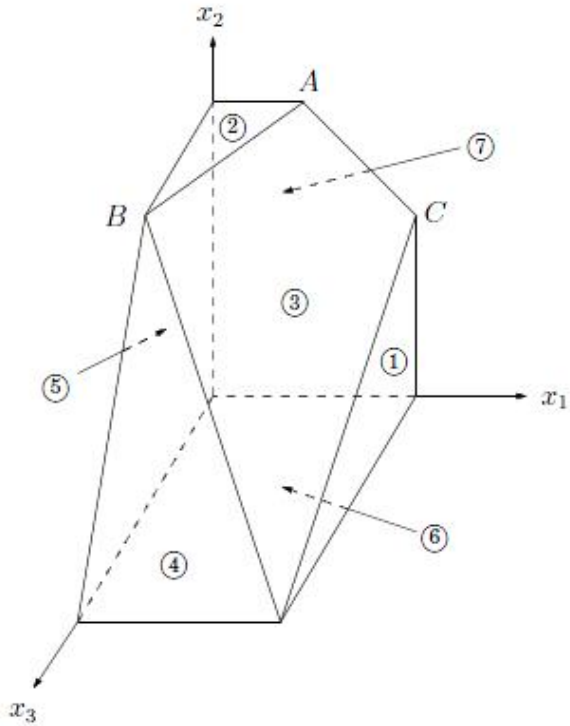
The Simplex Algorithm

1. Let v be any vertex of the feasible region
2. While (there is a neighbor v' of v with better objective value): set $v = v'$.

Assume there are n variables $x_1,\ x_2,\ \ldots,\ x_n$. Any setting of the x_i's can be represented by an n-tuple of real numbers and plotted in n-dimensional space. A linear equation involving the x_i's defines a **hyperplane** in this same space $\mathbb{R}^n$, and the corresponding linear inequality defines a **half-space**, all points that are either precisely on the hyperplane or lie on one particular side of it. Finally, the feasible region of the linear program is specified by a set of inequalities and is therefore the intersection of the corresponding half-spaces, a **convex polyhedron**.

But what do the concepts of vertex and neighbor mean in this general context?

Figure 7.12 A polyhedron defined by seven inequalities.



$$
\begin{aligned}
\max \quad & x_1 + 6x_2 + 13x_3 & \\
& x_1 \leq 200 & ① \\
& x_2 \leq 300 & ② \\
x_1 + x_2 + x_3 & \leq 400 & ③ \\
x_2 + 3x_3 & \leq 600 & ④ \\
& x_1 \geq 0 & ⑤ \\
& x_2 \geq 0 & ⑥ \\
& x_3 \geq 0 & ⑦
\end{aligned}
$$

Definition (Vertex)

Pick a subset of the inequalities. If there is a unique point that satisfies them with equality, and this point happens to be feasible, then it is a *vertex*.

How many equations are needed to uniquely identify a point? When there are n variables, we need at least n linear equations if we want a unique solution. On the other hand, having more than n equations is redundant: at least one of them can be rewritten as a linear combination of the others and can therefore be disregarded. In short,

- Each vertex is specified by a set of n inequalities.

A notion of **neighbor** now follows naturally.

- Two vertices are **neighbors** if they have $n - 1$ defining inequalities in common.

For instance, vertices A and C share the two defining inequalities $\{(3),\ (7)\}$ and are thus neighbors.

The Algorithm

On each iteration, simplex has two tasks:

1. Check whether the current vertex is optimal (and if so, halt).
2. Determine where to move next.

As we will see, both tasks are easy if the vertex happens to be at the origin. And if the vertex is elsewhere, we will transform the coordinate system to move it to the origin!
Suppose we have some generic LP

$$\begin{aligned} \max \quad & c^T x \\ & Ax \leq b \\ & x \geq 0 \end{aligned}$$

where x is the vector of variables, $x = (x_1,\ x_2,\ \ldots,\ x_n)$. Suppose the origin is feasible. Then it is certainly a vertex, since it is the unique point at which the n inequalities $\{x_1 \geq 0,\ \ldots,\ x_n \geq 0\}$ are *tight*.

Theorem (Task 1: Check whether the current vertex is optimal (and if so, halt).)

The original is optimal if and only if all $c_i \leq 0$.

Theorem (Task 2: Determine where to move next.)

We can move by increasing some x_i for which $c_i > 0$. How much can we increase it? ***Until we hit some other constraint.*** *That is, we release the tight constraint $x_i \geq 0$ and increase x_i until some other inequality, previously loose, now becomes tight. At that point, we again have exactly n tight inequalities, so we are at a new vertex.*

$$
\begin{aligned}
\max \quad & 2x_1 + 5x_2 \\
& 2x_1 - x_2 \leq 4 && (1)\\
& x_1 + 2x_2 \leq 9 && (2)\\
& -x_1 + x_2 \leq 3 && (3)\\
& x_1 \geq 0 && (4)\\
& x_2 \geq 0 && (5)
\end{aligned}
$$

Simplex can be started at the origin, which is specified by constraints (4) and (5) . To move, we release the tight constraint $x_2 \geq 0$. As x_2 is gradually increased, the first constraint it runs into is $-x_1 + x_2 \leq 3$, and thus it has to stop at $x_2 = 3$, at which point this new inequality is tight. The new vertex is thus given by (3) and (4).

What if Our Current Vertex u is Elsewhere?

Approach

The trick is to transform u into the origin, by shifting the coordinate system from the usual $(x_1, \ldots, x_n)$ to the "local view" from u. These local coordinates consist of (appropriately scaled) distances $y_1, \ldots, y_n$ to the n hyperplanes (inequalities) that define and enclose u.

Specifically, if one of these enclosing inequalities is $a_i \cdot x \leq b_i$, then the distance from a point x to that particular "wall" is

$$y_i = b_i - a_i \cdot x.$$

The n equations of this type, one per wall, define the y_i's as linear functions of the x_i's, and this relationship can be inverted to express the x_i's as a linear function of the y_i's. Thus we can rewrite the entire LP in terms of the y's. This does not fundamentally change it (for instance, the optimal value stays the same), but expresses it in a different coordinate frame.

The revised "local" LP has the following three properties:

1. It includes the inequalities $y \geq 0$, which are simply the transformed versions of the inequalities defining u.
2. u itself is the origin in y-space.
3. The cost function becomes $\max c_u + \tilde{c}^T y$, where c_u is the value of the objective function at u and $\tilde{c}$ is a transformed cost vector.

The simplex algorithm is now fully defined. It moves from vertex to neighboring vertex, **stopping** when the objective function is locally optimal, that is, **when the coordinates of the local cost vector are all zero or negative**. As we have just seen, a vertex with this property must also be globally optimal. On the other hand, if the current vertex is not locally optimal, then its local coordinate system includes some dimension along which the objective function can be improved, so we move along this direction.along this edge of the polyhedron.until we reach a neighboring vertex. By the nondegeneracy assumption, this edge has nonzero length, and so we strictly improve the objective value. Thus the process must eventually halt.

The starting vertex. Start with any linear program in standard form, since we know LPs can always be rewritten this way.

$$\min c^T x \text{ such that } Ax = b \text{ and } x \geq 0.$$

We first make sure that the right-hand sides of the equations are all nonnegative: if $b_i < 0$, just multiply both sides of the ith equation by -1. Then we create a new LP as follows:

1. Create m new *artificial variables* $z_1, \ldots, z_m \geq 0$, where m is the number of equations.
2. Add z_i to the left-hand side of the ith equation.
3. Let the objective, to be *minimized*, be $z_1 + z_2 + \cdots + z_m$.

For this new LP, it's easy to come up with a starting vertex, namely, the one with $z_i = b_i$ for all i and all other variables zero. Therefore we can solve it by simplex, to obtain the optimum solution.

Unboundedness. In some cases an LP is unbounded, in that its objective function can be made arbitrarily large. If this is the case, simplex will discover it: in exploring the neighborhood of a vertex, it will notice that taking out an inequality and adding another leads to an under-determined system of equations that has an infinity of solutions. And in fact (this is an easy test) the space of solutions contains a whole line across which the objective can become larger and larger. In this case simplex halts and complains.

Simplex is not a polynomial time algorithm. Certain rare kinds of linear programs cause it to go from one corner of the feasible region to a better corner and then to a still better one, and so on for an exponential number of steps.
In 1979, a young Soviet mathematician called Leonid Khachiyan came up with the ***ellipsoid algorithm***, one that is very different from simplex, extremely simple in its conception (but sophisticated in its proof) and yet one that solves any linear program in polynomial time. Instead of chasing the solution from one corner of the polyhedron to the next, Khachiyan's algorithm confines it to smaller and smaller ellipsoids (skewed high-dimensional balls). When this algorithm was announced, it became a kind of "mathematical Sputnik", a splashy achievement that had the U.S. establishment worried, in the height of the Cold War, about the possible scientific superiority of the Soviet Union. The ellipsoid algorithm turned out to be an important theoretical advance, but did not compete well with simplex in practice. The paradox of linear programming deepened: A problem with two algorithms, one that is efficient in theory, and one that is efficient in practice!

A few years later Narendra Karmarkar, a graduate student at UC Berkeley, came up with a completely different idea, which led to another provably polynomial algorithm for linear programming. Karmarkar's algorithm is known as the ***interior point method***, because it does just that: it dashes to the optimum corner not by hopping from corner to corner on the surface of the polyhedron like simplex does, but by cutting a clever path in the interior of the polyhedron. And it does perform well in practice. But perhaps the greatest advance in linear programming algorithms was not Khachiyan's theoretical breakthrough or Karmarkar's novel approach, but an unexpected consequence of the latter: the fierce competition between the two approaches, simplex and interior point, resulted in the development of very fast code for linear programming.

Ellipsoid Algorithm

Problem

Given a bounded convex set $P \in \mathbb{R}^n$, find $x \in P$. (We will see that we can reduce linear programming to finding an x in P.)

Theorem (Khachian 79)

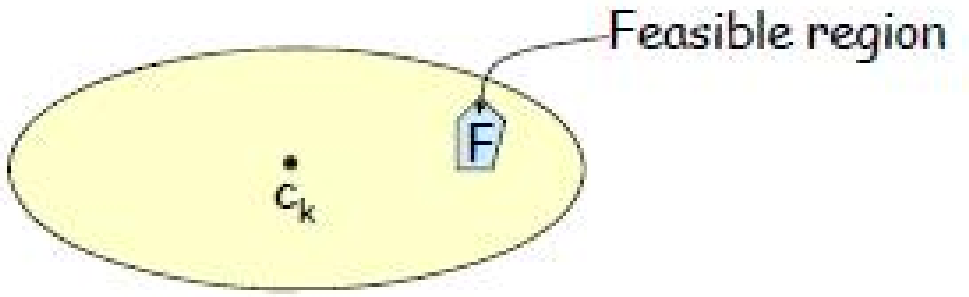
Ellipsoid algorithm is the first polynomial-time algorithm for linear programming. Its running time is $O(n^4 L)$, where L is the number of bits to represent A and b.

Idea

Consider a sequence of smaller and smaller ellipsoids; each with the feasible region inside. For iteration k:

$$c_k = \text{center of } E_k.$$

Eventually c_k has to be inside of F, and we are done.



Reduction from the General Case

Definition (Ellipsoid)

Given a center a, and a positive definite matrix A, the *ellipsoid* $E(a,\ A)$ is defined as

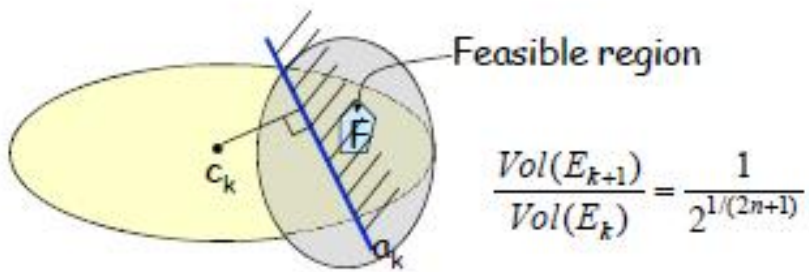
$$\{x \in \mathbb{R}^n :\ (x-a)^T A^{-1}(x-1) \leq 1\}$$

$$\begin{aligned} \max \quad & z = c^T x \\ \text{subject to} \quad & Ax \leq b \\ & x \geq 0. \end{aligned}$$

$$\begin{aligned} \text{find} \quad & x,\ y \\ \text{subject to} \quad & Ax \leq b \\ & -x \leq 0 \\ & -yA \leq -c \\ & -y \leq 0 \\ & -cx + by \leq 0 \end{aligned}$$

We start with a big ellipsoid E that is guaranteed to contain P. We then check if the center of the ellipsoid is in P. If it is, we are done, we found a point in P. Otherwise, we find an inequality $c^T x \leq b_i$ which is satisfied by all points in P (for example, it is explicitly given in the description of P) which is not satisfied by our center.

1. Let E_0 be an ellipsoid containing P.
2. **while** (center a_k of E_k is not in P) **do**
   - 2.1 Let $c^T x \leq c^T a_k$ be such that $\{x : c^T x \leq c^T a_k\} \supseteq P$.
   - 2.2 Let $E_k + 1$ be the minimum volume ellipsoid containing $E_k \bigcap \{x : c^T x \leq c^T a_k\}$.
   - 2.3 $k \leftarrow k + 1$.



The ellipsoid algorithm has the important property that the ellipsoids constructed shrink in volume as the algorithm proceeds.

Lemma

$$\frac{Vol(E_{k+1})}{Vol(E_k)} < e^{-\frac{1}{2(n+1)}}.$$

Interior Point Methods

Idea

Travel through the interior with a combination of (1) An optimization term (moves toward objective) and (2) A centering term (keeps away from boundary).

Remark

Used since 1950s for nonlinear programming. Karmakar proved a variant is polynomial time in 1984.

1. **Affine scaling**: simplest, but no known time bounds
2. **Potential reduction**: $O(nL)$ iterations
3. **Central trajectory**: $O(n^{1/2}L)$ iterations

The time for each iteration involves solving a linear system so it takes polynomial time. The "real world" time depends heavily on the matrix structure.