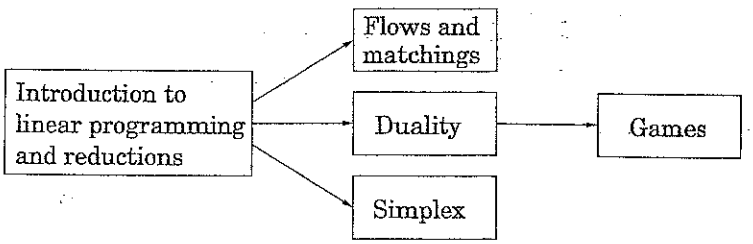
Chapter 7

Linear programming and reductions

Many of the problems for which we want algorithms are *optimization* tasks: the *shortest* path, the *cheapest* spanning tree, the *longest* increasing subsequence, and so on. In such cases, 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*. It turns out an enormous number of problems can be expressed in this way.

Given the vastness of its topic, this chapter is divided into several parts, which can be read separately subject to the following dependencies.



7.1 An introduction to 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.

7.1.1 Example: profit maximization

A boutique chocolatier has two products: its flagship assortment of triangular chocolates, called *Pyramide*, and the more decadent and deluxe *Pyramide Nuit*. How much of each should it produce to maximize profits? Let's say it makes x_1 boxes of Pyramide per day, at a profit of \$1 each, and x_2 boxes of Nuit, at a more substantial profit of \$6 apiece; x_1 and x_2 are unknown values that we wish to determine. But this is not all; there are also some constraints on x_1 and x_2 that must be accommodated (besides the obvious one, $x_1, x_2 \geq 0$). First, the daily demand for these exclusive chocolates 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 of chocolate per day. What are the optimal levels of production?

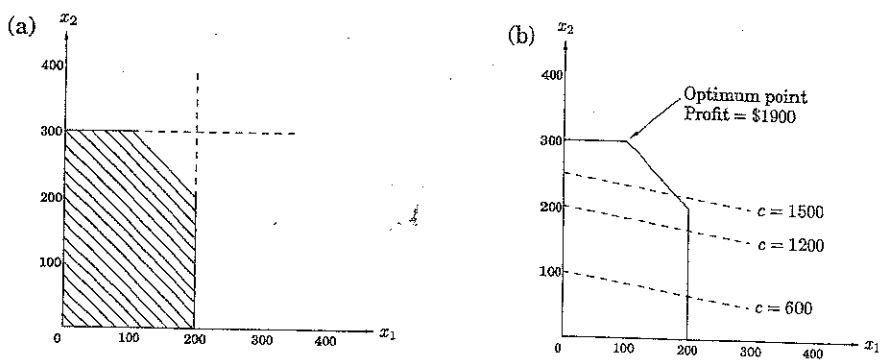
We represent the situation by a *linear program*, as follows.

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

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, shown in Figure 7.1.

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 $-1/6$ and is shown in Figure 7.1 for selected values of c. As c increases, this "profit line" moves parallel to itself, up and to the right. Since the goal

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.



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, as shown in the figure. If the slope of the profit line were different, then its last contact with the polygon could be an entire edge rather than a single vertex. In this case, the optimum solution would not be unique, but there would certainly be an optimum vertex.

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,

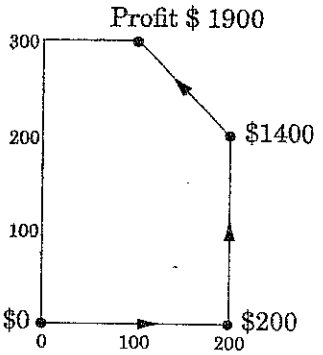
$$x \leq 1, \quad x \geq 2.$$

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

$$\max \; x_1 + x_2$$
$$x_1, x_2 \geq 0.$$

Solving linear programs

Linear programs (LPs) can be solved by the *simplex method*, devised by George Dantzig in 1947. We shall explain it in more detail in Section 7.6, but 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.

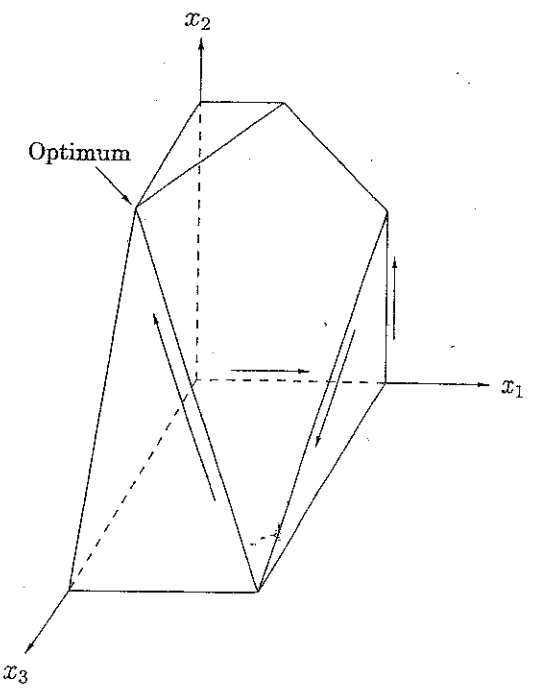
More products

Encouraged by consumer demand, the chocolatier decides to introduce a third and even more exclusive line of chocolates, called *Pyramide Luxe*. One box of these will bring in a profit of \$13. Let x_1, x_2, x_3 denote the number of boxes of each chocolate produced daily, with x_3 referring to Luxe. The old constraints on x_1 and x_2 persist, although the labor restriction now extends to x_3 as well: the sum of all three variables can be at most 400. What's more, it turns out that Nuit and Luxe require the same packaging machinery, except that Luxe uses it three times as much, which imposes another constraint $x_2 + 3x_3 \leq 600$. What are the best possible levels of production?

Here is the updated linear program.

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

Figure 7.2 The feasible polyhedron for a three-variable linear program.



The space of solutions is now three-dimensional. Each linear equation defines a 3D plane, and each inequality a half-space on one side of the plane. The feasible region is an intersection of seven half-spaces, a polyhedron (Figure 7.2). Looking at the figure, can you decipher which inequality corresponds to each face of the polyhedron?

A profit of c corresponds to the plane $x_1 + 6x_2 + 13x_3 = c$. As c increases, this profit-plane moves parallel to itself, further and further into the positive orthant until it no longer touches the feasible region. The point of final contact is the optimal vertex: (0, 300, 100), with total profit \$3100.

How would the simplex algorithm behave on this modified problem? As before, it would move from vertex to vertex, along edges of the polyhedron, increasing profit steadily. A possible trajectory is shown in Figure 7.2, corresponding to the following sequence of vertices and profits:

$$\underset{\$0}{(0,0,0)} \longrightarrow \underset{\$200}{(200,0,0)} \longrightarrow \underset{\$1400}{(200,200,0)} \longrightarrow \underset{\$2800}{(200,0,200)} \longrightarrow \underset{\$3100}{(0,300,100)}$$

Finally, upon reaching a vertex with no better neighbor, it would stop and declare this to be the optimal point. Once again by basic geometry, if all the vertex's neighbors lie on one side of the profit-plane, then so must the entire polyhedron.

A magic trick called duality

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?

In Section 7.4 we'll 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! But we are getting far ahead of our story.

What if we add a fourth line of chocolates, or hundreds more of them? Then the problem becomes high-dimensional, and hard to visualize. Simplex continues to work in this general setting, although we can no longer rely upon simple geometric intuitions for its description and justification. We will study the full-fledged simplex algorithm in Section 7.6.

In the meantime, we can rest assured in the knowledge that there are many professional, industrial-strength packages that implement simplex and take care of all the tricky details like numeric precision. In a typical application, the main task is therefore to correctly express the problem as a linear program. The package then takes care of the rest.

7.1.4 Variants of linear programming

As evidenced in our examples, 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.

We will now show that these various LP options *can all be reduced to one another* via simple transformations. Here's how.

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

2a. To turn an inequality constraint like $\sum_{i=1}^{n} a_i x_i \leq b$ into an equation, introduce a new variable s and use

$$\sum_{i=1}^{n} a_i x_i + s = b$$
$$s \geq 0.$$

This s is called the *slack variable* for the inequality. As justification, observe that a vector $(x_1, \ldots, x_n)$ satisfies the original inequality constraint if and only if there is some $s \geq 0$ for which it satisfies the new equality constraint.

2b. To change an equality constraint into inequalities is easy: rewrite $ax = b$ as the equivalent pair of constraints $ax \leq b$ and $ax \geq b$.

3. Finally, to deal with a variable x that is unrestricted in sign, do the following:
   - Introduce two nonnegative variables, $x^+, x^- \geq 0$.
   - 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.

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.

7.4 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.

To understand what duality is about, recall our introductory LP with the two types of chocolate:

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

Simplex declares the optimum solution to be $(x_1, x_2) = (100, 300)$, with objective value 1900. Can this answer be checked somehow? Let's see: suppose we take the first inequality and add it to six times the second inequality. We get

$$x_1 + 6x_2 \leq 2000.$$

This is interesting, because it tells us that it is impossible to achieve a profit of more than 2000. Can we add together some other combination of the LP constraints and bring this upper bound even closer to 1900? After a little experimentation, we find that multiplying the three inequalities by 0, 5, and 1, respectively, and adding them up yields

$$x_1 + 6x_2 \leq 1900.$$

So 1900 must indeed be the best possible value! 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's investigate the issue by describing 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	$x_1 + x_2 \leq 400$

To start with, these y_i's must be nonnegative, for otherwise they are unqualified to multiply inequalities (multiplying an inequality by a negative number would flip the $\leq$ to $\geq$). 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 $x_1 + 6x_2$ 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

$$x_1 + 6x_2 \leq 200y_1 + 300y_2 + 400y_3 \quad \text{if} \quad \left\{ \begin{array}{l} y_1, y_2, y_3 \geq 0 \\ y_1 + y_3 \geq 1 \\ y_2 + y_3 \geq 6 \end{array} \right\}.$$

We can easily find y's that satisfy the inequalities on the right by simply making them large enough, for example $(y_1, y_2, y_3) = (5, 3, 6)$. But these particular multipliers would tell us that the optimum solution of the LP is at most $200 \cdot 5 + 300 \cdot 3 + 400 \cdot 6 = 4300$, a bound that is far too loose to be of interest. What we want is a bound that is as tight as possible, so we should minimize $200y_1 + 300y_2 + 400y_3$ subject to the preceding inequalities. *And this is a new linear program!*

Therefore, finding the set of multipliers that gives the best upper bound on our original LP is tantamount to solving a new LP:

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

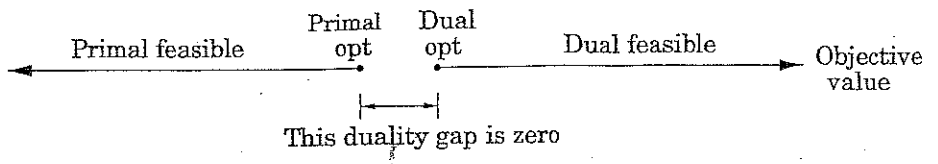
By design, any feasible value of this *dual* LP is an upper bound on the original *primal* LP. So if we somehow find a pair of primal and dual feasible values that are equal, then they must both be optimal. Here is just such a pair:

$$\text{Primal} : (x_1, x_2) = (100, 300); \quad \text{Dual} : (y_1, y_2, y_3) = (0, 5, 1).$$

They both have value 1900, and therefore they certify each other's optimality (Figure 7.9).

Amazingly, this is not just a lucky example, but a general phenomenon. To start with, the preceding construction—creating a multiplier for each primal constraint;

Figure 7.9 By design, dual feasible values ≥ primal feasible values. The duality theorem tells us that moreover their optima coincide.



writing a constraint in the dual for every variable of the primal, in which the sum is required to be above the objective coefficient of the corresponding primal variable; and optimizing the sum of the multipliers weighted by the primal right-hand sides—can be carried out for any LP, as shown in Figure 7.10, and in even greater generality in Figure 7.11. The second figure has one noteworthy addition: if the primal has an equality constraint, then the corresponding multiplier (or *dual variable*) need not be nonnegative, because the validity of equations is preserved when multiplied by negative numbers. So, the multipliers of equations are unrestricted variables. Notice also the simple symmetry between the two LPs, in that the matrix $A = (a_{ij})$ defines one primal constraint with each of its *rows*, and one dual constraint with each of its *columns*.

By construction, any feasible solution of the dual is an upper bound on any feasible solution of the primal. But moreover, their optima coincide!

Duality theorem: *If a linear program has a bounded optimum, then so does its dual, and the two optimum values coincide.*

When the primal is the LP that expresses the max-flow problem, it is possible to assign interpretations to the dual variables that show the dual to be none other than the minimum-cut problem (Exercise 7.25). The relation between flows and cuts is therefore just a specific instance of the duality theorem. And in fact, the proof of this theorem falls out of the simplex algorithm, in much the same way as the max-flow min-cut theorem fell out of the analysis of the max-flow algorithm.

Figure 7.10 A generic primal LP in matrix-vector form, and its dual.

Primal LP:	Dual LP:
$\max \mathbf{c}^T\mathbf{x}$	$\min \mathbf{y}^T\mathbf{b}$
$\mathbf{Ax} \leq \mathbf{b}$	$\mathbf{y}^T\mathbf{A} \geq \mathbf{c}^T$
$\mathbf{x} \geq 0$	$\mathbf{y} \geq 0$

7.6 The simplex algorithm

The extraordinary power and expressiveness of linear programs would be little consolation if we did not have a way to solve them efficiently. This is the role of the simplex algorithm.

At a high level, the simplex algorithm takes a set of linear inequalities and a linear objective function and finds the optimal feasible point by the following strategy:

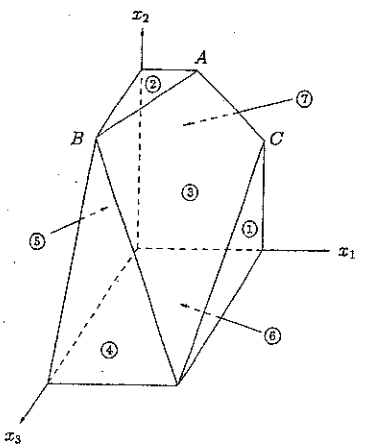
```
let v be any vertex of the feasible region
while there is a neighbor v' of v with better objective value:
  set v = v'
```

In our 2D and 3D examples (Figure 7.1 and Figure 7.2), this was simple to visualize and made intuitive sense. But what if there are n variables, $x_1, \ldots, x_n$?

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.



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

7.6.1 Vertices and neighbors in n-dimensional space

Figure 7.12 recalls an earlier example. Looking at it closely, we see that *each vertex is the unique point at which some subset of hyperplanes meet*. Vertex A, for instance, is the sole point at which constraints ②, ③, and ⑦ are satisfied with equality. On the other hand, the hyperplanes corresponding to inequalities ④ and ⑥ do not define a vertex, because their intersection is not just a single point but an entire line.

Let's make this definition precise.

> *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.[3]

A notion of *neighbor* now follows naturally.

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

In Figure 7.12, for instance, vertices A and C share the two defining inequalities {③, ⑦} and are thus neighbors.

7.6.2 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!

First let's see why the origin is so convenient. Suppose we have some generic LP

$$\begin{aligned} \max\ & \mathbf{c}^T\mathbf{x} \\ & \mathbf{A}\mathbf{x} \le \mathbf{b} \\ & \mathbf{x} \ge 0 \end{aligned}$$

where x is the vector of variables, $\mathbf{x} = (x_1, \ldots, x_n)$. Suppose the origin is feasible. Then it is certainly a vertex, since it is the unique point at which the n inequalities $\{x_1 \ge 0, \ldots, x_n \ge 0\}$ are *tight*. Now let's solve our two tasks. Task 1:

> The origin is optimal if and only if all $c_i \le 0$.

[3] There is one tricky issue here. It is possible that the same vertex might be generated by different subsets of inequalities. In Figure 7.12, vertex B is generated by {②, ③, ④}, but also by {②, ④, ⑤}. Such vertices are called *degenerate* and require special consideration. Let's assume for the time being that they don't exist, and we'll return to them later.

If all $c_i \leq 0$, then considering the constraints $\mathbf{x} \geq 0$, we can't hope for a better objective value. Conversely, if some $c_i > 0$, then the origin is not optimal, since we can increase the objective function by raising x_i.

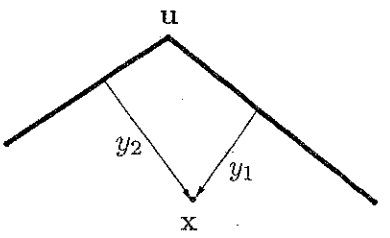
Thus, for task 2, we can move by increasing some x_i for which $c_i > 0$. How much can we increase it? *Until we hit some other constraint.* That is, we release the tight constraint $x_i \geq 0$ and increase x_i until some other inequality, previously loose, now becomes tight. At that point, we again have exactly n tight inequalities, so we are at a new vertex.

For instance, suppose we're dealing with the following linear program.

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

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.

So we know what to do if we are at the origin. But what if our current vertex $\mathbf{u}$ is elsewhere? The trick is to transform $\mathbf{u}$ into the origin, by shifting the coordinate system from the usual $(x_1, \ldots, x_n)$ to the "local view" from $\mathbf{u}$. These local coordinates consist of (appropriately scaled) distances $y_1, \ldots, y_n$ to the n hyperplanes (inequalities) that define and enclose $\mathbf{u}$:



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

$$y_i = b_i - \mathbf{a}_i \cdot \mathbf{x}.$$

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 doesn't 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 $\mathbf{y} \geq 0$, which are simply the transformed versions of the inequalities defining $\mathbf{u}$.
2. $\mathbf{u}$ itself is the origin in y-space.
3. The cost function becomes $\max c_{\mathbf{u}} + \tilde{\mathbf{c}}^T\mathbf{y}$, where $c_{\mathbf{u}}$ is the value of the objective function at $\mathbf{u}$ and $\tilde{\mathbf{c}}$ is a transformed cost vector.

In short, we are back to the situation we know how to handle! Figure 7.13 shows this algorithm in action, continuing with our earlier example.

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've 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 (see footnote 3 in Section 7.6.1), this edge has nonzero length, and so we strictly improve the objective value. Thus the process must eventually halt.

7.6.3 Loose ends

There are several important issues in the simplex algorithm that we haven't yet mentioned.

The starting vertex

How do we find a vertex at which to start simplex? In our 2D and 3D examples we always started at the origin, which worked because the linear programs happened to have inequalities with positive right-hand sides. In a general LP we won't always be so fortunate. However, it turns out that finding a starting vertex *can be reduced to an LP* and solved by simplex!

To see how this is done, start with any linear program in standard form (recall Section 7.1.4), since we know LPs can always be rewritten this way.

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

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:

- Create m new *artificial variables* $z_1, \ldots, z_m \geq 0$, where m is the number of equations.
- Add z_i to the left-hand side of the ith equation.
- 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.

There are two cases. If the optimum value of $z_1 + \cdots + z_m$ is zero, then all z_i's obtained by simplex are zero, and hence from the optimum vertex of the new LP we get a starting feasible vertex of the original LP, just by ignoring the z_i's. We can at last start simplex!

But what if the optimum objective turns out to be positive? Let us think. We tried to minimize the sum of the z_i's, but simplex decided that it cannot be zero. But this means that the original linear program is infeasible: it *needs* some nonzero z_i's to become feasible. This is how simplex discovers and reports that an LP is infeasible.

Degeneracy

In the polyhedron of Figure 7.12 vertex B is *degenerate*. Geometrically, this means that it is the intersection of more than $n = 3$ faces of the polyhedron (in this case, ②, ③, ④, ⑤). Algebraically, it means that if we choose any one of four sets of three inequalities ({②, ③, ④}, {②, ③, ⑤}, {②, ④, ⑤}, and {③, ④, ⑤}) and solve the corresponding system of three linear equations in three unknowns, we'll get the same solution in all four cases: (0, 300, 100). This is a serious problem: simplex may return a suboptimal degenerate vertex simply because all its neighbors are identical to it and thus have no better objective. And if we modify simplex so that it detects degeneracy and continues to hop from vertex to vertex despite lack of any improvement in the cost, it may end up looping forever.

One way to fix this is by a *perturbation*: change each b_i by a tiny random amount to $b_i \pm \epsilon_i$. This doesn't change the essence of the LP since the ϵ_i's are tiny, but it has the effect of differentiating between the solutions of the linear systems. To see why geometrically, imagine that the four planes ②, ③, ④, ⑤ were jolted a little. Wouldn't vertex B split into two vertices, very close to one another?

Unboundedness

In some cases an LP is unbounded, in that its objective function can be made arbitrarily large (or small, if it's a minimization problem). 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 underdetermined 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, all the way to ∞. In this case simplex halts and complains.

7.6.4 The running time of simplex

What is the running time of simplex, for a generic linear program

$$\max \mathbf{c}^T\mathbf{x} \text{ such that } \mathbf{A}\mathbf{x} \leq 0 \text{ and } \mathbf{x} \geq 0,$$

where there are n variables and $\mathbf{A}$ contains m inequality constraints? Since it is an iterative algorithm that proceeds from vertex to vertex, let's start by computing the time taken for a single iteration. Suppose the current vertex is $\mathbf{u}$. By definition, it is the unique point at which n inequality constraints are satisfied with equality. Each of its neighbors shares $n - 1$ of these inequalities, so $\mathbf{u}$ can have at most $n \cdot m$ neighbors: choose which inequality to drop and which new one to add.

A naive way to perform an iteration would be to check each potential neighbor to see whether it really is a vertex of the polyhedron and to determine its cost.

Finding the cost is quick, just a dot product, but checking whether it is a true vertex involves solving a system of n equations in n unknowns (that is, satisfying the n chosen inequalities exactly) and checking whether the result is feasible. By Gaussian elimination (see the following box) this takes $O(n^3)$ time, giving an unappetizing running time of $O(mn^4)$ per iteration.

Fortunately, there is a much better way, and this mn^4 factor can be improved to mn, making simplex a practical algorithm. Recall our earlier discussion (Section 7.6.2) about the *local view* from vertex $\mathbf{u}$. It turns out that the per-iteration overhead of rewriting the LP in terms of the current local coordinates is just $O((m+n)n)$; this exploits the fact that the local view changes only slightly between iterations, in just one of its defining inequalities.

Next, to select the best neighbor, we recall that the (local view of) the objective function is of the form "max $c_{\mathbf{u}} + \tilde{\mathbf{c}} \cdot \mathbf{y}$" where $c_{\mathbf{u}}$ is the value of the objective function at $\mathbf{u}$. This immediately identifies a promising direction to move: we pick any $\tilde{c}_i > 0$ (if there is none, then the current vertex is optimal and simplex halts). Since the rest of the LP has now been rewritten in terms of the y-coordinates, it is easy to determine how much y_i can be increased before some other inequality is violated. (And if we can increase y_i indefinitely, we know the LP is unbounded.)

It follows that the running time per iteration of simplex is just $O(mn)$. But how many iterations could there be? Naturally, there can't be more than $\binom{m+n}{n}$, which is an upper bound on the number of vertices. But this upper bound is exponential in n. And in fact, there are examples of LPs for which simplex does indeed take an exponential number of iterations. In other words, *simplex is an exponential-time algorithm*. However, such exponential examples do not occur in practice, and it is this fact that makes simplex so valuable and so widely used.

Linear programming in polynomial time

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. For a long time, linear programming was considered a paradox, a problem that can be solved in practice, but not in theory!

Then, 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!