Algorithms (XII)

Yijia Chen
Shanghai Jiaotong University

Review of the Previous Lecture

Chapter 7. Linear programming and reductions

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*.

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?

- Every box of Pyramide has a a profit of \$1.
- Every box of Nuit has a profit of \$6.
- The daily demand is limited to at most 200 boxes of Pyramide and 300 boxes of Nuit.
- The current workforce can produce a total of at most 400 boxes of chocolate per day.

LP formulation

Objective function	$\max\ x_1 + 6x_2$
Constraints	$x_1 \leq 200$
	$x_2 \leq 300$
	$x_1 + x_2 \leq 400$
	$x_1, x_2 \geq 0$

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.

The optimal solution

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.

The optimal solution (cont'd)

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,

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

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

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

Solving linear programs

Linear programs (LPs) can be solved by *the simplex method*, devised by George Dantzig in 1947.

This algorithm starts at a vertex, 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.

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

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 $x_2 + 3x_3 \leq 600$.

LP

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

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*.

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? A possible trajectory:

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

A magic trick called duality

Here is why you should believe that $(0, 300, 100)$, with a total profit of \$3100, is the optimum:

Recall

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

Add the second inequality to the third, and add to them the fourth multiplied by 4.

The result is the inequality

$$x_1 + 6x_2 + 13x_3 \leq 3100.$$

Integer linear programming

The optimum solution might turn out to be *fractional*.

This number would have to be **rounded** to either 10 or 11 in order to make sense, and the overall cost would then increase correspondingly.
In the present example, most of the variables take on fairly large (double-digit) values, and thus rounding is unlikely to affect things too much.

There are other LPs, however, in which rounding decisions have to be made very carefully in order to end up with an integer solution of reasonable quality.

In general, there is a tension in linear programming between *the ease of obtaining fractional solutions* and *the desirability of integer ones*.

As we shall see in Chapter 8, finding the optimum integer solution of an LP is an important but very hard problem, called **integer linear programming**.

Reductions

We want to solve Problem P.

We already have an algorithm that solves Problem Q.

If any subroutine for Q can also be used to solve P, we say P **reduces to** Q. Often, P is solvable by *a single call to Q's subroutine*, which means any instance x of P can be transformed into an instance y of Q such that $P(x)$ can be deduced from $Q(y)$.

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.

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

Variants of linear programming (cont'd)

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.

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 $x^+ - x^-$.

Standard form

Therefore, 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**:

- the variables are all nonnegative,
- the constraints are all equations,
- and the objective function is to be minimized.

$$\begin{array}{c} \max\ x_1 + 6x_2 \\ x_1 \leq 200 \\ x_2 \leq 300 \\ x_1 + x_2 \leq 400 \\ x_1, x_2, x_3 \geq 0 \end{array} \quad \Longrightarrow \quad \begin{array}{c} \min\ -x_1 - 6x_2 \\ x_1 + s_1 = 200 \\ x_2 + s_2 = 300 \\ x_1 + x_2 + s_3 = 400 \\ x_1, x_2, s_1, s_2, s_3 \geq 0 \end{array}$$

Flows in networks

Shipping oil

We have a network of pipelines along which oil can be sent. The *goal* is to ship as much oil as possible from the **source** s and the **sink** t.

Each pipeline has a *maximum capacity* it can handle, and there are no opportunities for storing oil en route.

Maximizing flow

The networks consist of a directed graph $G = (V, E)$; two special nodes $s, t \in V$, which are, respectively, a **source** and **sink** of G; and **capacities** $c_e > 0$ on the edges.

We would like to send as much oil as possible from s to t without exceeding the capacities of any of the edges.
A particular shipping scheme is called a **flow** and consists of a variable f_e for each edge e of the network, satisfying the following two properties:

1. It doesn't violate edge capacities: $0 \leq f_e \leq c_e$ for all $e \in E$.
2. For all nodes u except s and t, the amount of folw entering u equals the amount leaving

$$\sum_{(w,v)\in E} f_{wu} = \sum_{(u,z)\in E} f_{uz}$$

In other words, flow is *conserved*.

Maximizing flow (cont'd)

The **size** of a flow is the total quantity sent from s to t and, by the *conservation principle*, is equal to the quantity leaving s:

$$\texttt{size}(f) = \sum_{(s,u)\in E} f_{su}.$$

In short, our goal is to assign values to $\{f_e \mid e \in E\}$ that will satisfy a set of linear constraints and maximize a linear objective function.
But this is a linear program! The maximum-flow problem reduces to linear programming.

A closer look at the algorithm

All we know so far of the simplex algorithm is the vague *geometric intuition* that it keeps making local moves on the surface of a convex feasible region, successively improving the objective function until it finally reaches the optimal solution.

The behavior of simplex has an *elementary interpretation:*

Start with zero flow.

Repeat: choose an appropriate path from s to t, and increase flow along the edges of this path as much as possible.

A closer look at the algorithm (cont'd)

There is just one complication.

What if we choose a path that blocks all other paths?

Simplex gets around this problem by also allowing paths to *cancel existing flow*.

To summarize, in each iteration simplex looks for an s-t path whose edges (u, v) can be of two types:

1. (u, v) is in the original network, and is not yet *at full capacity*.
2. The reverse edge (v, u) is in the original network, and there is some flow along it.

If the current flow is f, then in the first case, edge (u, v) can handle up to $c_{uv} - f_{uv}$ additional units of flow, and in the second case, up to f_{vu} additional units (canceling all or part of the existing flow on (v, u)).

A closer look at the algorithm (cont'd)

These flow-increasing opportunities can be captured in a **residual network** $G^f = (V, E^f)$, which has exactly the two types of edges listed, with residual capacities c^f:

$$\begin{cases} c_{uv} - f_{uv} & \text{if } (u,v) \in E \text{ and } f_{uv} < c_{uv} \\ f_{vu} & \text{if } (v,u) \in E \text{ and } f_{vu} > 0. \end{cases}$$

Thus we can equivalently think of simplex as choosing an s-t path in the residual network.

By simulating the behavior of simplex, we get a *direct algorithm* for solving max-flow.

It proceeds in iterations, each time explicitly constructing G^f, finding a suitable s-t path in G^f by using, say, a linear-time breadth-first search, and halting if there is no longer any such path along which flow can be increased.

Cuts

A truly remarkable fact:

> *not only does simplex correctly compute a maximum flow, but it also generates a short proof of the optimality of this flow!*

An (s, t)-**cut** partitions the vertices into two *disjoint* groups L and R such that $s \in L$ and $t \in R$. Its **capacity** is the total capacity of the edges from L to R, and as argued previously, is an *upper bound* on *any* flow:

Pick any flow f and any (s, t)-cut (L, R). Then $\texttt{size}(f) \leq \texttt{capacity}(L, R)$.

A certificate of optimality

Theorem (Max-flow min-cut)

The size of the maximum flow in a network equals the capacity of the smallest (s, t)-cut.

Proof.

Suppose f is the final flow when the algorithm terminates.
We know that node t is no longer reachable from s in the residual network G^f.
Let L be the nodes that are reachable from s in G^f, and let $R = V \setminus L$ be the rest of the nodes. We claim that

$$\texttt{size}(f) = \texttt{capacity}(L, R)$$

To see this, observe that by the way L is defined, any edge going from L to R must be at full capacity (in the current flow f), and any edge from R to L must have zero flow.
Therefore the net flow across (L, R) is exactly the capacity of the cut. □

Efficiency

Each iteration of our maximum-flow algorithm is efficient, requiring $O(|E|)$ time if a DFS or BFS is used to find an s-t path.

But how many iterations are there?

Suppose all edges in the original network have integer capacities $\leq C$.
Then on each iteration of the algorithm, the flow is always an integer and increases by an integer amount. Therefore, since the maximum flow is at most $C|E|$, the number of iterations is at most this much.

If paths are chosen in a sensible manner – in particular, by using a BFS, which finds the path with the fewest edges – then the number of iterations is at most $O(|V| \cdot |E|)$, no matter what the capacities are.

This latter bound gives an overall running time of $O(|V| \cdot |E|^2)$ for maximum flow.

Bipartite matching

The problem

A graph with nodes on the left representing boys and nodes on the right representing girls.

There is an edge between a boy and girl if they like each other.
Is it possible to choose couples so that everyone has exactly one partner, and it is someone they like?

In graph-theoretic jargon, is there a **perfect matching**?

LP formulation

This matchmaking game can be reduced to the maximum-flow problem, and thereby to linear programming!

- Create a new source node s with outgoing edges to all the boys;
- a new sink node t, with incoming edges from all the girls;
- and direct all the edges in the original bipartite graph from boy to girl.
- Finally, give every edge a capacity of 1.

Then there is a perfect matching if and only if this network has a flow whose *size equals the number of couples*.

The maximum-flow problem has the following property: if all edge capacities are integers, then the optimal flow found by our algorithm is *integral*.
We can see this directly from the algorithm, which in such cases would increment the flow by an integer amount on each iteration.

Duality

Recall:

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

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

Can this answer be checked somehow?

We take the first inequality and add it to six times the second inequality:

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

Multiplying the three inequalities by 0, 5, and 1, respectively, and adding them up yields

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

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

These y_i's must be nonnegative, for otherwise they are unqualified to multiply inequalities.

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.

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

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*!