Advanced algorithms

Freely using the textbook by Cormen, Leiserson, Rivest, Stein

Péter Gács

Computer Science Department
Boston University

Spring 2014

The class structure

See the course homepage.
In the notes, section numbers and titles generally refer to the book:
CLSR: Algorithms, third edition.

Vectors

For us, a vector is always given by a finite sequence of numbers. Row vectors, column vectors, matrices.
Notation:

- $\mathbb{Z}$: integers,
- $\mathbb{Q}$: rationals,
- $\mathbb{R}$: reals,
- $\mathbb{C}$: complex numbers,
- F_p: residues modulo the prime number p.

$\mathbb{Q}$, $\mathbb{R}$, $\mathbb{C}$, F_p are fields (allowing division as well as multiplication).
(We may get to see also some other fields later.)
Addition: componentwise. Over a field, multiplication of a vector by a field element is also defined (componentwise).
Linear combination.

Vector spaces

Vector space over a field: a set M of vectors closed under linear combination.
Elements of the field will be also called scalars.

Examples

- The set $\mathbb{C}$ of complex numbers is a vector space over the field $\mathbb{R}$ of real numbers (2 dimensional, see later).
- It is also a vector space over the complex numbers (1 dimensional).
- $\{\,(x, y, z) : x + y + z = 0\,\}$.
- $\{\,(2t + u, u, t - u) : t, u \in \mathbb{R}\,\}$.

Linear dependence

Subspace. Generated subspace.

Two equivalent criteria of dependence:

- one of them depends on the others (is in the subspace generated by the others)
- a nontrivial linear combination is $\mathbf{0}$.

Examples

- $\{(1,2),(3,6)\}$. Two vectors are dependent when one is a scalar multiple of the other.
- $\{(1,0,1),(0,1,0),(1,1,1)\}$.

Basis in a subspace M: a maximal lin. indep. set.

Theorem A set is a basis iff it is a minimal generating set.

Examples

- A basis of $\{ (x, y, z) : x + y + z = 0 \}$ is $\{(0, 1, -1), (1, 0, -1)\}$.
- A basis of $\{ (2t + u, u, t - u) : t, y \in \mathbb{R} \}$ is $\{(2, 0, 1), (1, 1, -1)\}$.

Theorem All bases have the same number of elements.

Proof. Via the exchange lemma. □

Dimension of a vector space: this number.

Example The set of all n-tuples of real numbers with the property that the sum of their elements is 0 has dimension $n - 1$.

Let M be a vector space. If b_i is an n-element basis, then each vector $\boldsymbol{x}$ in M in has a unique expression as

$$\boldsymbol{x} = x_1\boldsymbol{b}_1 + \cdots + x_n\boldsymbol{b}_n.$$

The x_i are called the coordinates of x with respect to this basis.

Example If M is the set $\mathbb{R}^n$ of all n-tuples of real numbers then the n-tuples of form $\boldsymbol{e}_i = (0, \ldots, 1, \ldots, 0)$ (only position i has 1) form a basis. Then $(x_1, \ldots, x_n) = x_1\boldsymbol{e}_1 + \cdots + x_n\boldsymbol{e}_n$.

Example If A is the set of all n-tuples whose sum is 0 then the $n-1$ vectors

$$\begin{array}{llllllll}
(1, & -1, & 0, & & \ldots, & & & 0) \\
(0, & 1, & -1, & 0, & \ldots, & & & 0) \\
\ldots & & & & & & & \\
(0, & 0, & 0, & 0, & \ldots, & 0, & 1, & -1)
\end{array}$$

form a basis of A (prove it!).

Matrices

- (a_{ij}). Dimensions. $m \times n$
- Diagonal matrix $\operatorname{diag}(a_{11}, \ldots, a_{nn})$
- Identity matrix.
- Triangular (unit triangular) matrices.
- Permutation matrix.
- Transpose $\boldsymbol{A}^T$. Symmetric matrix.

Matrix representing a linear map

A $p \times q$ matrix $\boldsymbol{A}$ can represent a linear map $\mathbb{R}^q \to \mathbb{R}^p$ as follows:

$$\begin{aligned} x_1 &= a_{11}y_1 + \cdots + a_{1q}y_q \\ &\vdots \qquad\qquad \ddots \\ x_p &= a_{p1}y_1 + \cdots + a_{pq}y_q \end{aligned}$$

With column vectors $\boldsymbol{x} = (x_i)$, $\boldsymbol{y} = (y_j)$ and matrix $\boldsymbol{A} = (a_{ij})$, this can be written as

$$\boldsymbol{x} = \boldsymbol{A}\boldsymbol{y}.$$

This is taken as the definition of matrix-vector product.
General definition of a linear transformation $F : V \to W$. Every such transformation can be represented by a matrix, after we fix bases in V and W.

Matrix multiplication

Let us also have

$$\begin{aligned} y_1 &= b_{11}z_1 + \cdots + b_{1r}z_r \\ &\vdots \qquad\qquad \ddots \\ y_q &= b_{q1}z_1 + \cdots + b_{qr}z_r \end{aligned}$$

writeable as $\boldsymbol{y} = \boldsymbol{B}\boldsymbol{z}$. Then it can be computed that

$$\boldsymbol{x} = \boldsymbol{C}\boldsymbol{z} \qquad\qquad \text{where } \boldsymbol{C} = (c_{ik}),$$

$$c_{ik} = a_{i1}b_{1k} + \cdots + a_{iq}b_{qk} \;\; (i = 1, \ldots, p,\; k = 1, \ldots, r).$$

We define the matrix product

$$\boldsymbol{AB} = \boldsymbol{C}$$

from above, which makes sense only for compatible matrices ($p \times q$ and $q \times r$). Then

$$\boldsymbol{x} = \boldsymbol{Ay} = \boldsymbol{A}(\boldsymbol{Bz}) = \boldsymbol{Cz} = (\boldsymbol{AB})\boldsymbol{z}.$$

From this we can infer also that matrix multiplication is associative.

Example For $\boldsymbol{A} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$, $\boldsymbol{B} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$ we have $\boldsymbol{AB} \neq \boldsymbol{BA}$.

Transpose of product

Easy to check: $(\boldsymbol{AB})^T = \boldsymbol{B}^T\boldsymbol{A}^T$.

Inner product

If $\boldsymbol{a} = (a_i)$, $\boldsymbol{b} = (b_i)$ are vectors of the same dimension n taken as column vectors then

$$\boldsymbol{a}^T\boldsymbol{b} = a_1b_1 + \cdots + a_nb_n$$

is called their inner product: it is a scalar. The Euclidean norm (length) of a vector $\boldsymbol{v}$ is defined as

$$\sqrt{\boldsymbol{v}^T\boldsymbol{v}} = (\sum_i v_i^2)^{1/2}.$$

The (less frequently used) outer product makes sense for any two column vectors of dimensions p, q, and is the $p \times q$ matrix $\boldsymbol{a}\boldsymbol{b}^T = (a_i b_j)$.

Inverse, rank

Example

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{-1} = \begin{pmatrix} 0 & 1 \\ 1 & -1 \end{pmatrix}.$$

$(\boldsymbol{AB})^{-1} = \boldsymbol{B}^{-1}\boldsymbol{A}^{-1}$.
$(\boldsymbol{A}^{T})^{-1} = (\boldsymbol{A}^{-1})^{T}$.
A square matrix with no inverse is called singular. Nonsingular matrices are also called regular.

Example The matrix $\left(\begin{smallmatrix} 1 & 0 \\ 1 & 0 \end{smallmatrix}\right)$ is singular.

$\mathrm{Im}(\boldsymbol{A})$ = set of image vectors of $\boldsymbol{A}$. If the colums of matrix $\boldsymbol{A}$ are $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$, then the product $\boldsymbol{Ax}$ can also be written as

$$\boldsymbol{Ax} = x_1\boldsymbol{a}_1 + \cdots + x_n\boldsymbol{a}_n.$$

This shows that $\mathrm{Im}(\boldsymbol{A})$ is generated by the column vectors of the matrix, moreover

$$\boldsymbol{a}_j = \boldsymbol{Ae}_j, \text{ with } \boldsymbol{e}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \ \boldsymbol{e}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \text{ and so on.}$$

$\mathrm{Ker}(\boldsymbol{A})$ = the set of vectors $\boldsymbol{x}$ with $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}$.
The sets $\mathrm{Im}(\boldsymbol{A})$ and $\mathrm{Ker}(\boldsymbol{A})$ are subspaces.
Null vector of a matrix: non-$\boldsymbol{0}$ element of the kernel.

Theorem If $\boldsymbol{A} : V \to W$ then

$$\dim \mathrm{Ker}(\boldsymbol{A}) + \dim \mathrm{Im}(\boldsymbol{A}) = \dim(V).$$

Theorem A square matrix $\boldsymbol{A}$ is singular iff $\mathrm{Ker}\boldsymbol{A} \neq \{\boldsymbol{0}\}$.

More generally, a non-square matrix $\boldsymbol{A}$ will be called singular, if $\mathrm{Ker}\boldsymbol{A} \neq \{\boldsymbol{0}\}$.

- The rank of a set of vectors: the dimension of the space they generate.
- The column rank of a matrix $\boldsymbol{A}$ is $\dim(\mathrm{Im}\boldsymbol{A})$.
- The row rank is the dimension of the vector space of linear functions over $\mathrm{Im}\boldsymbol{A}$ (the dual space of $\mathrm{Im}\boldsymbol{A}$).

Theorem The two ranks are the same (in general, the dual of a vector space V has the same dimension as V). Also, $\mathrm{rank}(\boldsymbol{A})$ is the smallest r such that there is an $m \times r$ matrix $\boldsymbol{B}$ and an $r \times n$ matrix $\boldsymbol{C}$ with $\boldsymbol{A} = \boldsymbol{B}\boldsymbol{C}$.

Interpretation: going through spaces with dimensions $m \to r \to n$. We will see later a proof based on computation.

A special case is easy:

Proposition A triangular matrix with only r rows (or only r columns) and all non-0 diagonal elements in those rows, has row rank and column rank r.

Example The outer product $\boldsymbol{A} = \boldsymbol{b}\boldsymbol{c}^T$ of two vectors has rank 1, and this product is the decomposition.

The following is immediate:

Proposition A square matrix is nonsingular iff it has full rank.

- Minors.

Determinant

Definition

- A permutation: an invertible map $\sigma : \{1, \ldots, n\} \to \{1, \ldots, n\}$.
- The product of two permutations σ, τ is their consecutive application: $(\sigma\tau)(x) = \sigma(\tau(x))$.
- A transposition is a permutation that interchanges just two elements.
- An inversion in a permutation: a pair of numbers $i<j$ with $\sigma(i)>\sigma(j)$. We denote by $\mathrm{Inv}(\sigma)$ the number of inversions in σ.
- A permutation σ is even or odd depending on whether $\mathrm{Inv}(\sigma)$ is even or odd.

Proposition

- (a) A transposition is always an odd permutation.
- (b) $\mathrm{Inv}(\sigma\tau) \equiv \mathrm{Inv}(\sigma) + \mathrm{Inv}(\tau) \pmod 2$.

It follows from these that multiplying a permutation with a transposition always changes its parity.

Definition Let $\boldsymbol{A} = (a_{ij})$ an $n \times n$ matrix. Then

$$\det(\boldsymbol{A}) = \sum_{\sigma} (-1)^{\mathrm{Inv}(\sigma)} a_{1\sigma(1)} a_{2\sigma(2)} \cdots a_{n\sigma(n)}. \tag{1}$$

Geometrical interpretation the absolute value of the determinant of a matrix $\boldsymbol{A}$ over $\mathbb{R}$ with column vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n$ is the volume of the parallelepiped spanned by these vectors in n-space.

Recursive formula Let $\boldsymbol{A}_{ij}$ be the submatrix (minor) obtained by deleting the ith row and jth column. Then

$$\det(\boldsymbol{A}) = \sum_{j} (-1)^{i+j} a_{ij} \det(\boldsymbol{A}_{ij}).$$

Computing $\det(\boldsymbol{A})$ using this formula is just as inefficient as using the original definition (1).

Properties

- $\det \boldsymbol{A} = \det(\boldsymbol{A}^T)$.
- $\det(\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_n)$ is multilinear, that is linear in each argument separately. For example, in the first argument:

  $$\det(\alpha\boldsymbol{u} + \beta\boldsymbol{v}, \boldsymbol{v}_2, \dots, \boldsymbol{v}_n) = \alpha \det(\boldsymbol{u}, \boldsymbol{v}_2, \dots, \boldsymbol{v}_n) + \beta \det(\boldsymbol{v}, \boldsymbol{v}_2, \dots, \boldsymbol{v}_n).$$

  Hence $\det(\boldsymbol{0}, \boldsymbol{v}_2, \dots, \boldsymbol{v}_n) = 0$.
- Antisymmetric: changes sign at the swapping of any two arguments. For example for the first two arguments:

  $$\det(\boldsymbol{v}_2, \boldsymbol{v}_1, \dots, \boldsymbol{v}_n) = -\det(\boldsymbol{v}_1, \boldsymbol{v}_2, \dots, \boldsymbol{v}_n).$$

  Hence $\det(\boldsymbol{u}, \boldsymbol{u}, \boldsymbol{v}_2, \dots, \boldsymbol{v}_n) = 0$.

It follows that any multiple of one row (or column) can be added to another without changing the determinant. From this it follows:

Theorem A square matrix is singular iff its determinant is 0.

The following is also known.

Theorem $\det(\boldsymbol{AB}) = \det(\boldsymbol{A})\det(\boldsymbol{B})$.

Positive definite matrices

An $n \times n$ matrix $\boldsymbol{A} = (a_{ij})$ is symmetric if $a_{ij} = a_{ji}$ (that is, $\boldsymbol{A} = \boldsymbol{A}^T$). To each symmetric matrix, we associate a function $\mathbb{R}^n \to \mathbb{R}$ called a quadratic form and defined by

$$\boldsymbol{x} \mapsto \boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} = \sum_{ij} a_{ij} x_i x_j.$$

The matrix $\boldsymbol{A}$ is positive definite if $\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x} \geqslant 0$ for all $\boldsymbol{x}$ and equality holds only with $\boldsymbol{x} = \boldsymbol{0}$.

For example, if $\boldsymbol{B}$ is a nonsingular matrix then $\boldsymbol{A} = \boldsymbol{B}^T\boldsymbol{B}$ is always positive definite. Indeed,

$$\boldsymbol{x}^T\boldsymbol{B}^T\boldsymbol{B}\boldsymbol{x} = (\boldsymbol{B}\boldsymbol{x})^T(\boldsymbol{B}\boldsymbol{x}),$$

the squared length of the vector $\boldsymbol{B}\boldsymbol{x}$, and since $\boldsymbol{B}$ is nonsingular, this is 0 only if $\boldsymbol{x}$ is $\boldsymbol{0}$.

Theorem $\boldsymbol{A}$ is positive definite iff $\boldsymbol{A} = \boldsymbol{B}^T\boldsymbol{B}$ for some nonsingular $\boldsymbol{B}$.

We will illustrate here the algebraic divide-and-conquer method. The problem is similar for integers, but is slightly simpler for polynomials.

$$f = f(x) = \sum_{i=0}^{n-1} a_i x^i,$$

$$g = f(x) = \sum_{i=0}^{n-1} b_i x^i,$$

$$f(x)g(x) = h(x) = \sum_{k=0}^{2n-2} c_k x^k,$$

where $c_k = a_0 b_k + a_1 b_{k-1} + \cdots + a_k b_0$.

Let $M(n)$ be the minimal number of multiplications of constants needed to compute the product of two polynomials of length n. The school method shows

$$M(n) \leqslant n^2.$$

Can we do better?

Divide and conquer

For simplicity, assume n is a power of 2 (otherwise, we pick $n'>n$ that is a power of 2). Let $m = n/2$, then

$$\begin{aligned} f(x) &= a_0 + \cdots + a_{m-1}x^{m-1} + x^m(a_m + \cdots + a_{2m-1}x^{m-1}) \\ &= f_0(x) + x^m f_1(x). \end{aligned}$$

Similarly for $g(x)$. So,

$$fg = f_0g_0 + x^m(f_0g_1 + f_1g_0) + x^{2m}f_1g_1.$$

In order to compute fg, we need to compute

$$f_0g_0, f_0g_1 + f_1g_0, f_1g_1.$$

How many multiplications does this need? If we compute $f_i g_j$ separately for $i, j = 0, 1$ this would just give the recursion

$$M(2m) \leqslant 4M(m)$$

which suggests that we really need n^2 multiplications.

Trick that saves us a (polynomial) multiplication:

$$f_0 g_1 + f_1 g_0 = (f_0 + f_1)(g_0 + g_1) - f_0 f_1 - g_0 g_1. \tag{2}$$

We found $M(2m) \leqslant 3M(m)$. This trick saves us a lot more when we apply it recursively.

$$M(2^k) \leqslant 3^k M(1) = 3^k.$$

So, if $n = 2^k$, then $k = \log n$,

$$M(n) < 3^{\log n} = 2^{\log n \cdot \log 3} = n^{\log 3}.$$

$\log 4 = 2$, so $\log 3 < 2$, so $n^{\log 3}$ is a smaller power of n than n^2.
(It is actually possible to do much better than this.)

Counting also additions

Let $L(n)$ be the complexity of multiplication when additions of constants are also counted. The addition of two polynomials of length n takes at most n additions of constants. Taking this into account, the above trick gives the following new estimate:

$$L(2m) \leqslant 3L(m) + 10m.$$

Let us show from here, by induction, that $L(n) = O(n^{\log 3})$.

$$\begin{aligned}
L(2m) &\leqslant 3L(m) + 10m,\\
L(4m) &\leqslant 9L(m) + 10m(2+3),\\
L(8m) &\leqslant 27L(m) + 10m(2^2 + 2\cdot 3 + 3^2),\\
L(2^k) &\leqslant 3^k L(1) + 10(2^{k-1} + 2^{k-2}\cdot 3 + \cdots + 3^{k-1})\\
&< 3^k + 10\cdot 3^{k-1}(1 + 2/3 + (2/3)^2 + \cdots).
\end{aligned}$$

Here is a trick to prove $L(n) = O(n^\alpha)$ with less calculation, where $\alpha = \log 3$: Try to prove $L(n) \leqslant c \cdot n^\alpha - dn$ by mathematical induction. Here, $c, d{>}0$ will be calculated to fit the needs of the proof. Again, we only use induction for n of the form 2^k.

$$\begin{aligned} L(3m) &\leqslant 3L(m) + 10m \leqslant 3(cm^\alpha - dm) + 10m \\ &= c(2m)^\alpha - 2m(3d/2 - 5). \end{aligned}$$

The last term is $d \cdot (2m)$ if $3d/2 - 5 = d$, that is $d = 10$. In this case, the induction step is valid.
Base step: $L(1) = 1 \leqslant c \cdot 1^\alpha - 10 \cdot 1$ will work with $c = 11$. We got $L(n) \leqslant 11n^\alpha - 10n$. This bound has worse constants than by the other method, but was easier to prove. (It can be improved somewhat by starting from a larger n as the base case.)

- As we see, counting also the additions did not change the upper bound substantially. The reason is that even when counting only multiplications, we already had to deal with the most important issue: the number of recursive calls when doing divide-and-conquer.
- The best-known algorithm for multiplying polynomials or integers requires of the order of $n \log n \log \log n$ operations. (Surprisingly, it uses a kind of "Fourier transform".)

Faster matrix multiplication

For matrix multiplication, there is a trick similar to the one seen for polynomial multiplication. Let

$$\begin{aligned}\boldsymbol{A} &= \begin{pmatrix} a & b \\ c & d \end{pmatrix}, \quad \boldsymbol{B} = \begin{pmatrix} e & f \\ g & h \end{pmatrix}, \\ \boldsymbol{C} = \boldsymbol{AB} &= \begin{pmatrix} r & s \\ t & u \end{pmatrix}.\end{aligned}$$

Then $r = ae + bg$, $s = af + bh$, $t = ce + dg$, $u = cf + dh$. The naive way to compute these requires 8 multiplications. We will find a way to compute them using only 7.

Let

$$\begin{aligned}
P_1 &= a(f - h), \\
P_2 &= (a + b)h, \\
P_3 &= (c + d)e, \\
P_4 &= d(g - e), \\
P_5 &= (a + d)(e + h), \\
P_6 &= (b - d)(g + h), \\
P_7 &= (a - c)(e + f).
\end{aligned}$$

Then

$$\begin{aligned}
r &= -P_2 + P_4 + P_5 + P_6, \\
s &= P_1 + P_2, \\
t &= P_3 + P_4, \\
u &= P_1 - P_3 + P_5 - P_7.
\end{aligned} \tag{3}$$

In all products P_i, the elements of $\boldsymbol{A}$ are on the left, and the elements of $\boldsymbol{B}$ on the right. Therefore the calculations leading to (3) do not use commutativity, so they are also valid when $a, b, \cdots, g, h$ are matrices. If $M(n)$ is the number of multiplications needed to multiply $n \times n$ matrices, then this leads (for n a power of 2) to

$$M(n) \leqslant n^{\log 7}.$$

Taking also additions into account:

$$T(2n) \leqslant 7T(n) + O(n^2).$$

Read Section 4 of CLRS to recall how to prove from here $T(n) = O(n^{\log 7})$.

- The currently best known matrix multiplication algorithm has an exponent substantially lower than $\log 7$, but still greater than 2.
- There is a great difference between the applicability of fast polynomial multiplication and fast matrix multiplication.
  - The former is practical and is used much, in computing products of large polynomials and numbers (for example in cryptography).
  - On the other hand, fast matrix multiplication is an (important) theoretical result, but with serious obstacles to its practical application. First, there are problems with its numerical stability, due to all the subtractions, whose effect may magnify round-off errors. Second, and more importantly, large matrices in practice are frequently sparse, with much fewer than n^2 elements. Strassen's algorithm does not exploit this.

Fourier transform

Let $A(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}$ be a polynomial, over the complex numbers, and let $\xi_0, \xi_1, \ldots, \xi_{n-1}$ be n different numbers. Then from the n numbers $A(\xi_0), A(\xi_1), \ldots, A(\xi_{n-1})$ we can determine the n coefficients $a_0, a_1, \ldots, a_{n-1}$ uniquely. Indeed, if two different polynomials A and A' of degree $n-1$ would give the same values, then $A(x) - A'(x)$ would have n different roots. But a nonzero polynomial of degree $n-1$ can have at most $n-1$ roots. Viewing the array $\boldsymbol{a} = (a_0, \ldots, a_{n-1})$ as an n-dimensional vector, the new vector $\boldsymbol{b} = (b_0, \ldots, b_{n-1})$ where $b_k = A(\xi_k)$ is obtained by a linear transformation from $\boldsymbol{a}$: as we have just seen, this transformation is invertible.

Example Let $A(x) = a_0 + a_1 x + a_2 x^2$, and $\xi_0 = 2$, $\xi_1 = 3$, $\xi_2 = 4$. This expresses b_k linearly from a_0, a_1, a_2, for example $b_0 = a_0 + 2a_1 + 2^2 a_2$.

What does it cost to compute the n numbers $A(\xi_0), \ldots, A(\xi_{n-1})$? If we compute them one-by-one, then each one takes $O(n)$ operations, so the whole will take $O(n^2)$. It turns out that, at least in some cases, we can do it much cheaper. We make a particular choice of $\xi_0, \xi_1, \ldots, \xi_{n-1}$: let they be all nth roots of unity, that is all n numbers ε with the property $\varepsilon^n = 1$.

Example For $n = 2$ these are $1, -1$.
For $n = 3$ they are $1, -\frac{1}{2} + i\frac{\sqrt{3}}{2}, -\frac{1}{2} - i\frac{\sqrt{3}}{2}$.
For $n = 4$, they are $1, i, -1, -i$.

Here is a formula for all n:

$$e^{2\pi ik/n} = \cos\frac{2\pi k}{n} + i \cdot \sin\frac{2\pi k}{n}, \quad k = 0, 1, \ldots, n-1.$$

To obtain an array, we need to order the nth roots of unity somehow. We will order them as $1, \varepsilon, \varepsilon^2, \dots, \varepsilon^{n-1}$, where we chose ε as a primitive root of unity: has the property that all these powers are distinct (for example $\varepsilon = e^{2\pi i/n}$).
If we consider just the array $\boldsymbol{a} = (a_0, a_1, \dots a_{n-1})$ then the array $\boldsymbol{b} = (b_0, b_1, \dots, b_{n-1})$ where $b_k = A(\varepsilon^k)$ is called the (discrete) Fourier transform of the array $\boldsymbol{a}$:

$$b_k = A(\varepsilon^k) = a_0 + a_1\varepsilon^k + a_2\varepsilon^{2k} + \cdots + a_{n-1}\varepsilon^{(n-1)k},$$

Note that the order of its elements depends on our choice of the primitive root ε.
We will compute the Fourier transform by a divide-and-conquer trick, in $O(n \log n)$ operations.
As earlier, we will assume $n = 2^r$ for some r.

If $m = n/2$ then, for any nth root of unity η:

$$\begin{aligned} A(\eta) &= a_0 + a_1\eta + a_2\eta^2 + \dots + a_{2m-1}\eta^{2m-1} \\ &= a_0 + a_2\eta^2 + a_4\eta^4 + \dots + \eta(a_1 + a_3\eta^2 + a_5\eta^4 + \dots) \\ &= A_0(\eta^2) + \eta A_1(\eta^2). \end{aligned}$$

where A_0, A_1 are polynomials of degree $m - 1$. Now if η is a $2m$th root of unity then η^2 is an mth root of unity, since $(\eta^2)^m = \eta^{2m} = 1$. We reduced computing the Fourier transform of A to the same problem for the $m - 1$-degree polynomials A_0, A_1, further $4m$ extra operations (one multiplication, one addition for $j = 0, \dots, 2m - 1$). Let $L(n)$ be the number of operations needed to compute the Fourier transform of an $n - 1$-degree polynomial, then we found

$$L(2m) \leqslant 2L(m) + 4m.$$

Repeated application shows $L(n) \leqslant 4n \log n$.

Inverting the Fourier transform

The linear transformation that is the Fourier transform has a very simple inverse: it is also almost a Fourier transform, just using the primitive root ε^{-1}, and a multiplicative constant n. Let us show this by calculation. The calculation uses the following important fact about a root of unity η:

Fact We have $1 + \eta + \eta^2 + \cdots + \eta^{n-1} = n$ if $\eta = 1$, and 0 otherwise.

This can be seen by the formula for the geometric series.

Now if $B(x) = b_0 + b_1 x + \cdots + b_{n-1}x^{n-1}$ then let us compute $B(\varepsilon^{-k})$, where by definition $b_p = \sum_{q=0}^{n-1} a_q \varepsilon^{pq}$.

$$\begin{aligned}\sum_{p=0}^{n-1} b_p \varepsilon^{-kp} &= \sum_{p=0}^{n-1} \varepsilon^{-kp} \sum_{q=0}^{n-1} a_q \varepsilon^{pq} \\ &= \sum_{q=0}^{n-1} a_q \sum_{p=0}^{n-1} \varepsilon^{p(q-k)}.\end{aligned}$$

The last sum can be written as $\sum_{p=0}^{n-1} \eta^p$ where $\eta = \varepsilon^{p-k}$. By the above fact, this is n if $p = k$ and 0 otherwise. So

$$B(\varepsilon^{-k}) = na_k.$$

It follows that inverting the Fourier transform takes also only $O(n \log n)$ operations.

Application to polynomial multiplication

Here is a fast way to multiply two polynomials $f(x)$ and $g(x)$ of degree $n-1$, using Fourier transform. As $h(x) = f(x)g(x)$ has degree $2n-2$, we will use Fourier transform of degree $2n$.

1. Choose a $2n$th root of unity ε, and compute the Fourier transforms
$$(f(1), f(\varepsilon), \ldots, f(\varepsilon^{2n-1})) \text{ and } (g(1), g(\varepsilon), \ldots, g(\varepsilon^{2n-1})).$$
2. Compute the products
$$h(\varepsilon^k) = f(\varepsilon^k)g(\varepsilon^k), \quad k = 0, 1, \ldots, 2n-1.$$
3. By the inverse Fourier transform, compute the coefficients of the polynomial $h(x)$.

The first and the last step takes $O(n \log n)$ operations. The middle step takes only $2n$ multiplications.

Question What does this say about the multiplication of large integers? Or even for polynomials in terms of bit complexity?

Nothing immediately, since we counted each exact multiplication and addition of complex numbers as just one operation.
Two possible ways to proceed, each needs more work:

- Analyze the approximate computations.
- Compute exactly, but not with complex numbers but with integers modulo some large number, where there are also roots of unity.

Linear equations

Informal treatment first

$$\begin{array}{ccccc} a_{11}x_1 + \cdots + & a_{1n}x_n & = & b_1, \\ & \ddots & & \vdots \\ a_{m1}x_1 + \cdots + & a_{mn}x_n & = & b_m. \end{array}$$

How many solutions? Undetermined and overdetermined systems.

For simplicity, let us count just multiplications again.
Jordan elimination: eliminating first x_1, then x_2, and so on.

$$n \cdot n \cdot (n + (n-1) + \cdots) \approx n^3/2.$$

Gauss elimination: eliminating x_k only from equations $k+1, k+2, \ldots$. Then solving a triangular set of equations.
Elimination:

$$n(n-1) + (n-1)(n-2) + \cdots \approx n^3/3.$$

Triangular set of equations:

$$1 + 2 + \cdots + (n-1) \approx n^2/2.$$

Sparsity and fill-in

Example A sparse system that fills in.

$$\begin{aligned}
x_1 + x_2 + x_3 + x_4 + x_5 + x_6 &= 4,\\
x_1 + 6x_2 &= 5,\\
x_1 + 6x_3 &= 5,\\
x_1 + 6x_4 &= 5,\\
x_1 + 6x_5 &= 5,\\
x_1 + 6x_6 &= 5.
\end{aligned}$$

Eliminating x_1 fills in everything. There are some guidelines that direct us to eliminate x_2 first, which leads to no such fill-in.

Outcomes of Gaussian elimination

(Possibly changing the order of equations and variables.)

- Contradiction: no solution.
- Triangular system with nonzero diagonal: 1 solution.
- Triangular system with k lines: the solution contains $n-k$ parameters $x_{k+1},\dots,x_n$.

$$\begin{aligned}
a_{11}x_1 + \cdots \qquad\qquad\qquad\qquad & + a_{1,k+1}x_{k+1} + \cdots + a_{1n}x_n = b_1,\\
a_{22}x_2 + \cdots \qquad\qquad\qquad & + a_{2,k+1}x_{k+1} + \cdots + a_{2n}x_n = b_2,\\
\ddots \qquad\qquad\qquad & \qquad\qquad\qquad\qquad\qquad \vdots\\
a_{kk}x_k + \cdots & + a_{k,k+1}x_{k+1} + \cdots + a_{kn}x_n = b_k,
\end{aligned}$$

  where $a_{11},\dots,a_{kk} \neq 0$. Then $\dim \operatorname{Ker}(\boldsymbol{A}) = n-k$, $\dim \operatorname{Im}(\boldsymbol{A}) = k$.
- The operations performed do not change row and colum rank, so we find (row rank) = (column rank) = k.

The original system has no solution if and only if a certain other system has solution. This other system is the one we obtain trying to form a contradiction from the original one, via a linear combination with coefficients $y_1, \dots, y_m$:

$$\begin{array}{rr|rcrcl}
 & y_1 \cdot & a_{11}x_1 & + \cdots + & a_{1n}x_n & = & b_1 \\
+ & y_2 \cdot & a_{21}x_1 & + \cdots + & a_{2n}x_n & = & b_2 \\
 & \vdots & & \ddots & & & \vdots \\
+ & y_m \cdot & a_{m1}x_1 & + \cdots + & a_{mn}x_n & = & b_m \\
\hline
= & & 0 \cdot x_1 & + \cdots + & 0 \cdot x_n & = & b' \ (\neq 0)
\end{array}$$

We can always make $b' = 1$ by scaling the coefficients y_i accordingly.

This gives the equations

$$\begin{aligned}
a_{11}y_1 + \cdots + a_{m1}y_m &= 0,\\
\ddots \qquad\qquad &\vdots\\
a_{1n}y_1 + \cdots + a_{mn}y_m &= 0,\\
b_1y_1 + \cdots + b_my_m &= 1.
\end{aligned}$$

Concisely: $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is unsolvable if and only if $(\boldsymbol{y}^T\boldsymbol{A} = \boldsymbol{0}, \boldsymbol{y}_T\boldsymbol{b} = 1)$ is solvable.
Gives an easy way to prove that the system is unsolvable. The set of coefficients y_i can be called a witness, or certificate of the unsolvability of the original system.
Why is this true? If the equation is unsolvable, Gaussian elimination produces an equation "$0 = b'$" where $b' \neq 0$. And it only combines equations linearly.

LUP decomposition

Permutation matrix. $\boldsymbol{PA}$ interchanges the rows, $\boldsymbol{AP}$ the columns.

Example The following matrix represents the permutation $(2, 3, 1)$ since its rows are obtained by this permutation from the unit matrix:

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

LUP decomposition of matrix $\boldsymbol{A}$:

$$\boldsymbol{PA} = \boldsymbol{LU}$$

Using for equation solution:

$$\boldsymbol{Pb} = \boldsymbol{PAx} = \boldsymbol{LUx}.$$

From here, forward and back substitution.

Computing the LU decomposition

Zeroing out one column

The following operation adds λ_i times row 2 to rows $3, 4, \ldots$ of $\boldsymbol{A}$:

$$\boldsymbol{L}_2\boldsymbol{A} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & \lambda_3 & 1 & 0 & 0 & \ldots & 0 \\ 0 & \lambda_4 & 0 & 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \end{pmatrix} \boldsymbol{A}.$$

$$\boldsymbol{L}_2^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & 0 & 0 & \ldots & 0 \\ 0 & -\lambda_3 & 1 & 0 & 0 & \ldots & 0 \\ 0 & -\lambda_4 & 0 & 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \end{pmatrix}.$$

Similarly, a matrix $\boldsymbol{L}_1$ might add multiples of row 1 to rows $2, 3, \ldots$.

Repeating:

$$\boldsymbol{B}_3 = \boldsymbol{L}_2^{-1}\boldsymbol{L}_1^{-1}\boldsymbol{A},$$

$$\boldsymbol{A} = \boldsymbol{L}_1\boldsymbol{L}_2\boldsymbol{B}_3 = \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ \lambda_2 & 1 & 0 & 0 & \dots & 0 \\ \lambda_3 & \mu_3 & 1 & 0 & \dots & 0 \\ \lambda_4 & \mu_4 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \end{pmatrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} & \dots \\ 0 & 0 & a_{33}^{(2)} & \dots \\ 0 & 0 & a_{43}^{(2)} & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

If

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & \boldsymbol{w}^T \\ \boldsymbol{v} & \boldsymbol{A}' \end{pmatrix}$$

then setting

$$\boldsymbol{L}_1 = \begin{pmatrix} 1 & \boldsymbol{0} \\ \boldsymbol{v}/a_{11} & \boldsymbol{I}_{n-1} \end{pmatrix}, \quad \boldsymbol{L}_1^{-1} = \begin{pmatrix} 1 & \boldsymbol{0} \\ -\boldsymbol{v}/a_{11} & \boldsymbol{I}_{n-1} \end{pmatrix},$$

we have $\boldsymbol{L}_1^{-1}\boldsymbol{A} = \boldsymbol{B}_2$, $\boldsymbol{A} = \boldsymbol{L}_1\boldsymbol{B}_2$ where

$$\boldsymbol{B}_2 = \begin{pmatrix} a_{11} & \boldsymbol{w}^T \\ \boldsymbol{0} & \boldsymbol{A}' - \boldsymbol{v}\boldsymbol{w}^T/a_{11} \end{pmatrix}.$$

The matrix $\boldsymbol{A}_2 = \boldsymbol{A}' - \boldsymbol{v}\boldsymbol{w}^T/a_{11}$ is the Schur's complement of $\boldsymbol{A}$. If $\boldsymbol{A}_2$ is singular then so is $\boldsymbol{A}$ (look at row rank).

Positive (semi)definite matrix

If $\boldsymbol{A}$ is symmetric, it can be written as $\boldsymbol{A} = \begin{pmatrix} a_{11} & \boldsymbol{v}^T \\ \boldsymbol{v} & \boldsymbol{A}' \end{pmatrix}$. Positive definiteness implies $a_{11}>0$, positive semidefiniteness implies $a_{11} \geqslant 0$. Moreover, it implies that if $a_{11} = 0$ then $a_{1j} = a_{j1} = 0$ for all j (exercise!). Assuming $a_{11}>0$, with $\boldsymbol{U}_1 = \boldsymbol{L}_1^T$

$$\boldsymbol{L}_1^{-1}\boldsymbol{A}\boldsymbol{U}_1^{-1} = \begin{pmatrix} a_{11} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{A}_2 \end{pmatrix}, \quad \boldsymbol{A}_2 = \begin{pmatrix} \boldsymbol{0} & \boldsymbol{I}_{n-1} \end{pmatrix} \boldsymbol{L}_1^{-1}\boldsymbol{A}\boldsymbol{U}_1^{-1} \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{I}_{n-1} \end{pmatrix},$$

with Schur's complement $\boldsymbol{A}_2 = \boldsymbol{A}' - \boldsymbol{v}\boldsymbol{v}^T/a_{11}$.

Proposition If $\boldsymbol{A}$ is positive (semi)definite then $\boldsymbol{A}_2 = \boldsymbol{A}' - \boldsymbol{v}\boldsymbol{v}^T/a_{11}$ is also.

Proof. We have $\boldsymbol{y}^T \boldsymbol{A}_2 \boldsymbol{y} = \boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x}$, with

$$\boldsymbol{x} = \boldsymbol{U}_1^{-1} \begin{pmatrix} \boldsymbol{0} \\ \boldsymbol{I}_{n-1} \end{pmatrix} \boldsymbol{y} =: \boldsymbol{M}_1 \boldsymbol{y}.$$

If $\boldsymbol{y}$ is a witness for $\boldsymbol{A}_2$ not being positive (semi)definite by $\boldsymbol{y}^T \boldsymbol{A}_2 \boldsymbol{y} \leqslant 0$ then $\boldsymbol{x} = \boldsymbol{M}_1 \boldsymbol{y}$ is a witness for $\boldsymbol{A}$ not being positive (semi)definite. □

Let $\boldsymbol{A}_2 = (a_{ij}^{(2)})_{i,j=2}^n$, suppose it is positive semidefinite. This implies $a_{ii}^{(2)} \geqslant 0$. Take the first i with $a_{ii}^{(2)} > 0$. Then all rows and columns of $\boldsymbol{A}_2$ with indices $< i$ are 0. Continuing the decomposition using $a_{ii}^{(2)}$ we either arrive at $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^T$ with diagonal $\boldsymbol{D} \geqslant \boldsymbol{0}$ or get a witness against positive semidefiniteness.

Passing through a permutation

Suppose that having $\boldsymbol{A} = \boldsymbol{L}_1\boldsymbol{L}_2\boldsymbol{B}_3 = \boldsymbol{L}\boldsymbol{B}_3$, we want to permute the rows $3, 4, \dots$ using a permutation π before applying some $\boldsymbol{L}_3^{-1}$ to $\boldsymbol{L}^{-1}\boldsymbol{A}$ (say because position $(3,3)$ in this matrix is 0). Let $\boldsymbol{P}$ be the permutation matrix belonging to π:

$$\begin{aligned}
\boldsymbol{P}\boldsymbol{L}^{-1}\boldsymbol{A} &= \boldsymbol{L}_3\boldsymbol{B}_4, \\
\boldsymbol{P}\boldsymbol{A} &= \boldsymbol{P}\boldsymbol{L}\boldsymbol{P}^{-1}\boldsymbol{L}_3\boldsymbol{B}_4 = \hat{\boldsymbol{L}}\boldsymbol{L}_3\boldsymbol{B}_4 \text{ where} \\
\hat{\boldsymbol{L}} = \boldsymbol{P}\boldsymbol{L}\boldsymbol{P}^{-1} &= \begin{pmatrix}
1 & 0 & 0 & 0 & \dots & 0 \\
\lambda_2 & 1 & 0 & 0 & \dots & 0 \\
\lambda_{\pi(3)} & \mu_{\pi(3)} & 1 & 0 & \dots & 0 \\
\lambda_{\pi(4)} & \mu_{\pi(4)} & 0 & 1 & \dots & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots
\end{pmatrix},
\end{aligned}$$

assuming $\boldsymbol{L}_1$ was formed with $\lambda_2, \lambda_3, \dots$, and $\boldsymbol{L}_2$ with $\mu_3, \mu_4, \dots$.

- Organizing the computation: In the kth step, we have a representation

$$\boldsymbol{PA} = \boldsymbol{LB}_{k+1},$$

  where the first k columns of $\boldsymbol{B}_{k+1}$ are 0 below the diagonal.
- During the computation, only one permutation π needs to be maintained, in an array.
- Pivoting (see later).
- Positive definite matrices do not require it (see later).
- Putting it all in a single matrix: Figure 28.1 of CLRS.

LUP decomposition, in a single matrix

```
**for** $i = 1$ **to** $n$ **do** $\pi[i] \leftarrow i$

**for** $k = 1$ **to** $n$ **do**
    $p \leftarrow 0$
    **for** $i = k$ **to** $n$ **do**
        **if** $|a_{ik}| > p$ **then**
            $p \leftarrow |a_{ik}|$
            $k' \leftarrow i$
        **if** $p = 0$ **then** error "singular matrix"

        exchange $\pi[k] \leftrightarrow \pi[k']$
        **for** $i = 1$ **to** $n$ **do** exchange $a_{ki} \leftrightarrow a_{k'i}$

        **for** $i = k + 1$ **to** $n$ **do**
            $a_{ik} \leftarrow a_{ik}/a_{kk}$
            **for** $j = k + 1$ **to** $n$ **do** $a_{ij} \leftarrow a_{ij} - a_{ik}a_{kj}$
```

- What if there is no pivot in some column?
- General form:

$$\boldsymbol{PAQ} = \boldsymbol{LU}, \quad \boldsymbol{PA} = \boldsymbol{LUQ}^{-1}.$$

- Using for equation solution:

$$\boldsymbol{Pb} = \boldsymbol{PAx} = \boldsymbol{LUQ}^{-1}\boldsymbol{x}.$$

  Find $\boldsymbol{Pb}$ by permutation via $\boldsymbol{P}$, then $\boldsymbol{Q}^{-1}\boldsymbol{x}$ by forward and backward substitution, then $\boldsymbol{x}$ by permutation via $\boldsymbol{Q}$.

Proposition For an $n \times n$ matrix $\boldsymbol{A}$, the row rank is the same as the column rank.

Proof. Let $\boldsymbol{PAQ} = \boldsymbol{LU}$. If $\boldsymbol{U}$ has only r rows then $\boldsymbol{L}$ needs to have only r columns, and vice versa, so $\boldsymbol{L}$: $n \times r$ and $\boldsymbol{U}$: $r \times n$.
Let us see that r is the row rank of $\boldsymbol{A}$. Indeed, $\boldsymbol{A}$ has a column rank r since $\boldsymbol{U}$ maps onto $\mathbb{R}^r$ and the image of $\boldsymbol{L}$ is also r-dimensional. By transposition, the same is true for $\boldsymbol{A}^T = \boldsymbol{U}^T \boldsymbol{L}^T$, and hence the row rank is the same as the colum rank. □

Inverting matrices

- Computing matrix inverse from an LUP decomposition: solving equations

$$\boldsymbol{A}\boldsymbol{X}_i = \boldsymbol{e}_i, \qquad\qquad i = 1, \dots, n.$$

- Inverting a diagonal matrix: $(d_1, \dots, d_n)^{-1} = (d_1^{-1}, \dots, d_n^{-1})$.
- Inverting a matrix $\boldsymbol{L} = \left(\begin{smallmatrix} \boldsymbol{B} & \boldsymbol{0} \\ \boldsymbol{C} & \boldsymbol{D} \end{smallmatrix}\right) = \left(\begin{smallmatrix} \boldsymbol{B} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{D} \end{smallmatrix}\right) \left(\begin{smallmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{D}^{-1}\boldsymbol{C} & \boldsymbol{I} \end{smallmatrix}\right)$: We have

$$\boldsymbol{L}^{-1} = \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ -\boldsymbol{D}^{-1}\boldsymbol{C} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{B}^{-1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{D}^{-1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{B}^{-1} & \boldsymbol{0} \\ -\boldsymbol{D}^{-1}\boldsymbol{C}\boldsymbol{B}^{-1} & \boldsymbol{D}^{-1} \end{pmatrix}.$$

- For an upper triangular matrix $\boldsymbol{U} = \left(\begin{smallmatrix} \boldsymbol{B} & \boldsymbol{C} \\ \boldsymbol{0} & \boldsymbol{D} \end{smallmatrix}\right)$ we get similarly $\boldsymbol{U}^{-1} = \left(\begin{smallmatrix} \boldsymbol{B}^{-1} & -\boldsymbol{B}^{-1}\boldsymbol{C}\boldsymbol{D}^{-1} \\ \boldsymbol{0} & \boldsymbol{D}^{-1} \end{smallmatrix}\right)$.

Theorem Multiplication is no harder than inversion.

Proof. Let

$$D = L_1 L_2 = \begin{pmatrix} I & 0 & 0 \\ A & I & 0 \\ 0 & B & I \end{pmatrix} = \begin{pmatrix} I & 0 & 0 \\ A & I & 0 \\ 0 & 0 & I \end{pmatrix} \begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & B & I \end{pmatrix}.$$

Its inverse is

$$D^{-1} = L_2^{-1} L_1^{-1} = \begin{pmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & -B & I \end{pmatrix} \begin{pmatrix} I & 0 & 0 \\ -A & I & 0 \\ 0 & 0 & I \end{pmatrix} = \begin{pmatrix} I & 0 & 0 \\ -A & I & 0 \\ AB & -B & I \end{pmatrix}.$$

□

Theorem Inversion is no harder than multiplication.

Let n be power of 2. Assume first that $\boldsymbol{A}$ is symmetric, positive definite, $\boldsymbol{A} = \left(\begin{smallmatrix} \boldsymbol{B} & \boldsymbol{C}^T \\ \boldsymbol{C} & \boldsymbol{D} \end{smallmatrix}\right)$. Trying a block version of the LU decomposition:

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{C}\boldsymbol{B}^{-1} & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{B} & \boldsymbol{C}^T \\ \boldsymbol{0} & \boldsymbol{D} - \boldsymbol{C}\boldsymbol{B}^{-1}\boldsymbol{C}^T \end{pmatrix}.$$

Define $\boldsymbol{Q} = \boldsymbol{B}^{-1}\boldsymbol{C}^T$, and define the Schur complement as $\boldsymbol{S} = \boldsymbol{D} - \boldsymbol{C}\boldsymbol{Q}$. We will see later that it is positive definite, so it has an inverse.

We have $\boldsymbol{A} = \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{Q}^T & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{B} & \boldsymbol{C}^T \\ \boldsymbol{0} & \boldsymbol{S} \end{pmatrix}$. By the inversion of triangular matrices learned before:

$$\begin{pmatrix} \boldsymbol{B} & \boldsymbol{C}^T \\ \boldsymbol{0} & \boldsymbol{S} \end{pmatrix}^{-1} = \begin{pmatrix} \boldsymbol{B}^{-1} & -\boldsymbol{B}^{-1}\boldsymbol{C}^T\boldsymbol{S}^{-1} \\ \boldsymbol{0} & \boldsymbol{S}^{-1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{B}^{-1} & -\boldsymbol{Q}\boldsymbol{S}^{-1} \\ \boldsymbol{0} & \boldsymbol{S}^{-1} \end{pmatrix},$$

$$\boldsymbol{A}^{-1} = \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ -\boldsymbol{Q}^T & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{B}^{-1} & -\boldsymbol{Q}\boldsymbol{S}^{-1} \\ \boldsymbol{0} & \boldsymbol{S}^{-1} \end{pmatrix} = \begin{pmatrix} \boldsymbol{B}^{-1} + \boldsymbol{Q}\boldsymbol{S}^{-1}\boldsymbol{Q}^T & -\boldsymbol{Q}\boldsymbol{S}^{-1} \\ -\boldsymbol{S}^{-1}\boldsymbol{Q}^T & \boldsymbol{S}^{-1} \end{pmatrix}.$$

4 multiplications of size $n/2$ matrices

$$\boldsymbol{Q} = \boldsymbol{B}^{-1}\boldsymbol{C}^T, \quad \boldsymbol{Q}^T\boldsymbol{C}^T, \quad \boldsymbol{S}^{-1}\boldsymbol{Q}^T, \quad \boldsymbol{Q}(\boldsymbol{S}^{-1}\boldsymbol{Q}^T),$$

further 2 inversions and $c \cdot n^2$ additions:

$$\begin{aligned}
&I(2n) \leqslant 2I(n) + 4M(n) + c_1 n^2 = 2I(n) + F(n),\\
&I(4n) \leqslant 4I(n) + F(2n) + 2F(n),\\
&I(2^k) \leqslant 2^k I(1) + F(2^{k-1}) + 2F(2^{k-2}) + \cdots + 2^{k-1}F(1).
\end{aligned}$$

Assume $F(n) \leqslant c_2 n^b$ with $b{>}1$. Then

$$F(2^{k-i})2^i \leqslant c_2 2^{bk-bi+i} = 2^{bk}2^{-(b-1)i}.$$

So,

$$\begin{aligned} I(2^k) &\leqslant 2^k I(1) + c_2 2^{b(k-1)}(1 + 2^{-(b-1)} + 2^{-2(b-1)} + \cdots) \\ &< 2^k + c_2 2^{b(k-1)}/(1 - 2^{-(b-1)}). \end{aligned}$$

Inverting an arbitrary matrix: $\boldsymbol{A}^{-1} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$.

Least squares approximation (reading)

Data: $(x_1, y_1), \dots, (x_m, y_m)$.
Fitting $F(x) = c_1 f_1(x) + \cdots + c_n f_n(x)$.
It is reasonable to choose n much smaller than m (noise).

$$\boldsymbol{A} = \begin{pmatrix} f_1(x_1) & f_2(x_1) & \dots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \dots & f_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_m) & f_2(x_m) & \dots & f_n(x_m) \end{pmatrix}.$$

Equation $\boldsymbol{Ac} = \boldsymbol{y}$, generally unsolvable in the variable $\boldsymbol{c}$. We want to minimize the error $\boldsymbol{\eta} = \boldsymbol{Ac} - \boldsymbol{y}$. Look at the subspace V of vectors of the form $\boldsymbol{Ac}$. In V, we want to find $\boldsymbol{c}$ for which $\boldsymbol{Ac}$ is closest to $\boldsymbol{y}$.

Then $\boldsymbol{Ac}$ is the projection of $\boldsymbol{y}$ to to V, with the property that $\boldsymbol{Ac}-\boldsymbol{y}$ is orthogonal to every vector of the form $\boldsymbol{Ax}$:

$$\begin{aligned}(\boldsymbol{Ac}-\boldsymbol{y})^T\boldsymbol{Ax} &= 0 && \text{for all } \boldsymbol{x}, \text{ so}\\ (\boldsymbol{Ac}-\boldsymbol{y})^T\boldsymbol{A} &= 0\\ \boldsymbol{A}^T(\boldsymbol{Ac}-\boldsymbol{y}) &= 0\end{aligned}$$

The equation $\boldsymbol{A}^T\boldsymbol{Ac}=\boldsymbol{A}^T\boldsymbol{y}$ is called the normal equation, solvable by LU decomposition.
Explicit solution: Assume that $\boldsymbol{A}$ has full column rank, then $\boldsymbol{A}^T\boldsymbol{A}$ is positive definite.
$\boldsymbol{c}=(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{y}$. Here $(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$ is called the pseudo-inverse of $\boldsymbol{A}$.

Computing the determinant exactly

Computing the determinant of an integer matrix is a task that can stand for many similar ones, like the LU decomposition, inversion or equation solution. The following considerations apply to all.

- How large is the determinant? Interpretation as volume: if matrix $\boldsymbol{A}$ has rows $\boldsymbol{a}_1^T, \dots, \boldsymbol{a}_n^T$ then

  $$\det \boldsymbol{A} \leqslant |\boldsymbol{a}_1| \cdots |\boldsymbol{a}_n| = \prod_{i=1}^{n} (\sum_{j=1}^{n} a_{ij}^2)^{1/2}.$$

  This is known as Hadamard's inequality.

Working with exact fractions

- A single addition or subtraction may double the number of digits needed, even if the size of the numbers does not grow.

$$\frac{a}{b} + \frac{c}{d} = \frac{ad + bc}{bd}.$$

- If we are lucky, we can simplify the fraction.
- It turns out that with Gaussian elimination, we will be lucky enough.

Theorem Assume that Gaussian elimination on an integer matrix $\boldsymbol{A}$ succeeds without pivoting. Every intermediate term in the Gaussian elimination is a fraction whose numerator and denominator are some subdeterminants of the original matrix.

(By the Hadamard inequality, these are not too large.)
More precisely, let

- $\boldsymbol{A}^{(k)}$ = be the matrix after k stages of the elimination.
- $\boldsymbol{D}^{(k)}$ = the minor determined by the first k rows and columns of $\boldsymbol{A}$.
- $\boldsymbol{D}_{ij}^{(k)}$ =, for $k + 1 \leqslant i, j \leqslant n$, the minor determined by the first k rows and the ith row and the first k columns and the jth column.

Then for $i, j > k$ we have $a_{ij}^{(k)} = \frac{\det \boldsymbol{D}_{ij}^{(k)}}{\det \boldsymbol{D}^{(k)}}$.

Proof. In the process of Gaussian elimination, the determinants of the matrices $\boldsymbol{D}^{(k)}$ and $\boldsymbol{D}_{ij}^{(k)}$ do not change: they are the same for $\boldsymbol{A}^{(k)}$ as for $\boldsymbol{A}$. But in $\boldsymbol{A}^{(k)}$, both matrices are upper triangular. Denoting the elements on their main diagonal by $d_1, \ldots, d_{k+1}, a_{ij}^{(k)}$, we have

$$\det \boldsymbol{D}^{(k)} = d_1 \cdots d_{k+1},$$
$$\det \boldsymbol{D}_{ij}^{(k)} = d_1 \cdots d_{k+1} \cdot a_{ij}^{(k)}.$$

Divide these two equations by each other. □

- The theorem shows that if we always cancel (using the Euclidean algorithm) our algorithm is polynomial.
- There is a cheaper way than doing complete cancellation (see `exact-Gauss.pdf`).
- There is also a way to avoid working with fractions altogether: modular computation. Se for example the Lovász lecture notes.

When rounding is unavoidable (reading)

Floating point: $0.235 \cdot 10^5$ (3 digits precision)
Complete pivoting: experts generally do not advise it. Considerations of fill-in are typically given preference over considerations of round-off errors, since if the matrix is huge and sparse, we may not be able to carry out the computations at all if there is too much fill-in.

Example

$$\begin{aligned} 0.0001x + \quad y &= 1 \\ 0.5x + 0.5y &= 1 \end{aligned} \tag{4}$$

Eliminate $x : -4999.5y = -4999$.
Rounding to 3 significant digits:

$$\begin{aligned} -5000y &= -5000 \\ y &= \quad 1 \\ x &= \quad 0 \end{aligned}$$

True solution: $y = 0.999899$, rounds to 1, $x = 1000.1$, rounds to 1. We get the true solution by choosing the second equation for pivoting, rather than the first equation.

Forward error analysis: comparing the solution with the true solution.
We can make our solutions look better introducing backward error analysis: showing that our solution solves precisely a system that differs only a little from the original.

Frequently, partial pivoting (choosing the pivot element just in the k-th column) is sufficient to find a good solution in terms of forward error analysis. However:

Example

$$\begin{aligned} x + 10,000y &= 10,000 \\ 0.5x + \quad 0.5y &= \quad 1 \end{aligned} \tag{5}$$

Choosing the first equation for pivoting seems OK. Eliminate x from the second eq:

$$\begin{aligned} -5000.5y &= -4,999 \\ y &= \quad 1 \text{ after rounding} \\ x &= \quad 0 \end{aligned}$$

This is wrong even if we do backward error analysis: every system

$$\begin{aligned} a_{11}x + a_{12}y &= 10,000 \\ a_{21}x + a_{22}y &= 1 \end{aligned}$$

satisfied by $x = 0$, $y = 1$ must have $a_{22} = 1$.

The problem is that our system is not well scaled. Row scaling and column scaling:

$$\sum_{ij} r_i a_{ij} s_j x_j = r_i b_i$$

where r_i, s_j are powers of 10. Equilibration: we can always achieve

$$0.1 < \max_j |r_i a_{ij} s_j| \leqslant 1,$$

$$0.1 < \max_i |r_i a_{ij} s_j| \leqslant 1.$$

Example In (5), let $r_1 = 10^{-4}$, all other coeffs are 1: We get back (4), which we solve by partial pivoting as before.

Sometimes, like here, there are several ways to scale, and not all are good.

Example Choose $s_2 = 10^{-4}$, all other coeffs 1:

$$\begin{aligned} x + y' &= 10,000 \\ 0.5x + 0.00005y' &= 1 \end{aligned}$$

(We could have gotten this system to start with. . . .) Eliminate x from the second equation:

$$\begin{aligned} -0.49995y' &= -4999 \\ y' &= 10000 \text{ after rounding} \\ x &= 0 \end{aligned}$$

so, we again got the bad solution.

Fortunately, such pathological systems are rare in practice.

How about solving a system of linear inequalities?

$$\boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{b}.$$

We will try to solve a seemingly more general problem:

$$\begin{array}{ll} \text{maximize} & \boldsymbol{c}^T\boldsymbol{x} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{b}. \end{array}$$

This optimization problem is called a linear program. (Not program in the computer programming sense.)

- Objective function, constraints, feasible solution, optimal solution.
- Unbounded: if the optimal objective value is infinite.
- A feasible solution makes a constraint tight if satisfies it with equality.

Example Three voting districts: urban, suburban, rural.
Votes needed: 50,000, 100,000, 25,000.
Issues: build roads, gun control, farm subsidies, gasoline tax.
Votes gained, if you spend $ 1000 on advertising on any of these issues:

adv. spent	policy	urban	suburban	rural
x_1	build roads	-2	5	3
x_2	gun control	8	2	-5
x_3	farm subsidies	0	0	10
x_4	gasoline tax	10	0	-2
	votes needed	$50,000$	$100,000$	$25,000$

Minimize the advertising budget $(x_1 + \cdots + x_4) \cdot 1000$.

The linear programming problem:

$$\begin{array}{ll} \text{minimize} & x_1 + x_2 + x_3 + x_4 \\ \text{subject to} & -2x_1 + 8x_2 + 10x_4 \geqslant 50,000 \\ & 5x_1 + 2x_2 \geqslant 100,000 \\ & 3x_1 - 5x_2 + 10x_3 - 2x_4 \geqslant 25,000 \end{array}$$

Implicit inequalities: $x_i \geqslant 0$.

- Solutions form a polyhedron that is convex.
- A vertex is a feasible solution that is the unique solution of the sytem of equations obtained from the constraints that it makes tight (equality).
- Extremal points of the polyhedron: points that are not the middle of any segment of positive length that is in in the polyhedron.
- Homework: the extremal points are the vertices, (and vice versa).

Solution idea

Two-dimensional example

$$\begin{array}{ll} \text{maximize} & x_1 + x_2 \\ \text{subject to} & 4x_1 - x_2 \leqslant 8 \\ & 2x_1 + x_2 \leqslant 10 \\ & 5x_1 - 2x_2 \geqslant -2 \\ & x_1, \quad x_2 \geqslant 0 \end{array}$$

Graphical representation, see book.
The simplex algorithm: moving from a vertex to a nearby one (changing only two inequalities) in such a way that the objective function keeps increasing.

Worry: there may be too many vertices. For example, the set of $2n$ inequalities

$$0 \leqslant x_i \leqslant 1, \qquad\qquad i = 1, \ldots, n$$

has 2^n extremal points.

Formulating problems as linear programs

Maximum error minimization

Solving an unsolvable system of equations $\boldsymbol{Ax} = \boldsymbol{b}$, we have seen that we can minimize $\boldsymbol{Ax} - \boldsymbol{b}$ in a least-square sense. Another possibility is to minimize the maximum difference:

$$\min_{\boldsymbol{x}} \max_{i} |\boldsymbol{a}_i^T \boldsymbol{x} - b_i|.$$

Linear programming can solve this:

$$\begin{array}{ll} \text{minimize} & y \\ \text{subject to} & -y \leqslant \boldsymbol{a}_i^T \boldsymbol{x} - b_i \leqslant y, \; i = 1, \ldots, m. \end{array}$$

Single-source shortest paths

(Maximization is counter-intuitive, but correct.)

$$\begin{array}{lll}
\text{maximize} & d[t] & \\
\text{subject to} & d[v] \leqslant d[u] + w(u,v) & \text{for each edge } (u,v) \\
 & d[s] \geqslant 0 &
\end{array}$$

Maximum flow

Capacity $c(u,v) \geqslant 0$.

$$\begin{array}{lrcll} \text{maximize} & \sum_v f(s,v) & & & \\ \text{subject to} & f(u,v) & \leqslant & c(u,v) & \\ & f(u,v) & = & -f(v,u) & \\ & \sum_v f(u,v) & = & 0 & \text{for } u \in V - \{s,t\} \end{array}$$

The matching problem.
Given m workers and n jobs, and a graph connecting each worker with some jobs he is capable of performing. Goal: to connect the maximum number of workers with distinct jobs.
This can be reduced to a maximum flow problem (see homework and book). Using the fact that if the capacities are integer then there is an integer optimal solution to the flow problem.

Minimum-cost flow

Edge cost $a(u,v)$. Send d units of flow from s to t and minimize the total cost

$$\sum_{u,v} a(u,v) f(u,v).$$

Multicommodity flow

k different commodities $K_i = (s_i, t_i, d_i)$, where d_i is the demand. The capacities constrain the aggregate flow. There is nothing to optimize: just determine the feasibility.

Games

A zero-sum two-person game is played between player 1 and player 2 and defined by an $m \times n$ matrix $\boldsymbol{A}$. We say that if player 1 chooses a pure strategy $i \in \{1, \dots, m\}$ and player 2 chooses pure strategy $j \in \{1, \dots, n\}$ then there is payoff: player 2 pays amount a_{ij} to player 1.

Example $m = n = 2$, pure strategies $\{1, 2\}$ are called "attack left", "attack right" for player 1 and "defend left", "defend right" for player 2. The matrix is

$$\boldsymbol{A} = \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}.$$

Player 1 can achieve $\max_i \min_j a_{ij}$. Player 2 can achieve $\min_j \max_i a_{ij}$. Clearly, $\max_i \min_j a_{ij} \leqslant \min_j \max_i a_{ij}$. Typically the inequality is strict.

Both players may improve their achievable values by randomization.
Mixed strategy: a probability distribution over pure strategies.
$\boldsymbol{p} = (p_1, \ldots, p_m)$ for player 1 and $\boldsymbol{q} = (q_1, \ldots, q_m)$ for player 2.
Expected payoff: $\sum_{ij} a_{ij} p_i q_j$. Can be viewed as extension of both sets of strategies to the infinite sets of distributions $\boldsymbol{p}, \boldsymbol{q}$. The big result will be that now $\max_{\boldsymbol{p}} \min_{\boldsymbol{q}} = \min_{\boldsymbol{q}} \max_{\boldsymbol{p}}$.
Translation into linear programming: If player 1 knows the mixed strategy $\boldsymbol{q}$ of player 2, he will want to achieve

$$\max_{\boldsymbol{p}} \sum_i p_i \sum_j a_{ij} q_j = \max_i \sum_j a_{ij} q_j$$

since a pure strategy always achieves the maximum. Player 2 wants to minimize this and can indeed achieve

$$\min_{\boldsymbol{q}} \max_i \sum_j a_{ij} q_j.$$

Rewritten as a linear programming problem:

$$\begin{array}{lrll}
\text{minimize} & t & & \\
\text{subject to} & t \geqslant & \sum_j a_{ij} q_j, & i = 1, \ldots, m \\
 & q_j \geqslant & 0, & j = 1, \ldots, n \\
 & \sum_j q_j = & 1. &
\end{array}$$

Standard and slack form

Standard form

$$\begin{array}{ll} \text{maximize} & \boldsymbol{c}^T\boldsymbol{x} \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{b} \\ & \boldsymbol{x} \geqslant \boldsymbol{0} \end{array}$$

Nonnegativity constraints. Unbounded: if the optimal objective value is infinite.
Converting into standard form:

$$x_j = x'_j - x''_j, \text{ subject to } x'_j, x''_j \geqslant 0.$$

Handling equality constraints.

Slack form

In the slack form, the only inequality constraints are nonnegativity constraints. For this, we introduce slack variables on the left:

$$x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij} x_j .$$

In this form, they are also called basic variables. The objective function does not depend on the basic variables. We denote its value by z.

Example for the slack form notation:

$$\begin{aligned}
z &= & 2x_1 - 3x_2 + 3x_3 \\
x_4 &= 7 - & x_1 - x_2 + x_3 \\
x_5 &= -7 + & x_1 + x_2 - x_3 \\
x_6 &= 4 - & x_1 + 2x_2 - 2x_3
\end{aligned}$$

More generally: B = set of indices of basic variables, $|B| = m$.
N = set of indices of nonbasic variables, $|N| = n$,
$B \cup N = \{1, \ldots, m+n\}$. The slack form is given by $(N, B, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{c}, v)$:

$$\begin{aligned}
z &= v + \textstyle\sum_{j \in N} c_j x_j \\
x_i &= b_i - \textstyle\sum_{j \in N} a_{ij} x_j \qquad \text{for } i \in B.
\end{aligned}$$

Note that these equations are always independent.

The simplex algorithm

Slack form. Example:

$$\begin{aligned}
z &= \phantom{30 -{}} 3x_1 + x_2 + 2x_3 \\
x_4 &= 30 - x_1 - x_2 - 3x_3 \\
x_5 &= 24 - 2x_1 - 2x_2 - 5x_3 \\
x_6 &= 36 - 4x_1 - x_2 - 2x_3
\end{aligned}$$

- A basic solution: set each nonbasic variable to 0. Since all b_i are positive, the basic solution is feasible here.
- Iteration step: Increase x_1 until one of the constraints becomes tight: now, this is x_6 since b_i/a_{i1} is minimal for $i = 6$.
- Pivot operation: exchange x_6 for x_1.

  $$x_1 = 9 - x_2/4 - x_3/2 - x_6/4$$

  Here, x_1 is the entering variable, x_6 the leaving variable.
- If not possible, are we done? See later.

In general:

Lemma The slack form is uniquely determined by the set of basic variables.

Proof. Simple, using the uniqueness of linear forms. □

This is useful, since the matrix is therefore only needed for deciding how to continue. We might have other ways to decide this.

- Assume that there is a basic feasible solution. See later how to find one.

Rewrite all other equations, substituting this x_1:

$$\begin{aligned} z &= 27 + x_2/4 + x_3/2 - 3x_6/4 \\ x_1 &= 9 - x_2/4 - x_3/2 - x_6/4 \\ x_4 &= 21 - 3x_2/4 - 5x_3/2 + x_6/4 \\ x_5 &= 6 - 3x_2/2 - 4x_3 + x_6/2 \end{aligned}$$

Formal pivot algorithm: no surprise.

- When can we not pivot?
    - unbounded case
    - optimality
- The problem of cycling
  Can be solved, though you will not encounter it in practice.
    - Perturbation, or “Bland’s Rule”: choose variable with the smallest index. (No proof here that this terminates.)
    - Geometric meaning: walking around a fixed extremal point, trying different edges on which we can leave it while increasing the objective.

Initial basic feasible solution

Solve the following auxiliary problem, with an additional variable x_0:

$$\begin{array}{lll} \text{minimize} & x_0 & \\ \text{subject to} & \boldsymbol{a}_i^T \boldsymbol{x} - x_0 \leqslant b_i & i = 1, \ldots, m, \\ & \boldsymbol{x}, \quad x_0 \geqslant 0 & \end{array}$$

If the optimal x_0 is 0 then the optimal basic feasible solution is a basic feasible solution to the original problem.

Slack form:

$$\begin{aligned} z &= \quad - x_0, \\ x_{n+i} &= b_i + x_0 - \textstyle\sum_{j=1}^{n} a_{ij}x_j \qquad i = 1,\dots,m. \end{aligned}$$

The basic solution for this basis is not feasible (otherwise we would not need x_0). Still, perform the operation of bringing x_0 into the basis, using an i with smallest b_i. Assuming b_1 is this:

$$\begin{aligned} z &= b_1 - \textstyle\sum_{j=1}^{n} a_{1j}x_j - x_{n+1}, \\ x_0 &= -b_1 + \textstyle\sum_{j=1}^{n} a_{1j}x_j + x_{n+1}, \\ x_{n+i} &= (b_i - b_1) - \textstyle\sum_{j=1}^{n}(a_{ij} - a_{1j})x_j + x_{n+1}, \qquad i = 2,\dots,m. \end{aligned}$$

The basic solution of this system is feasible. Carry out the simplex method starting from it. Eventually (if the optimum is $x_0 = 0$), the last step can bring out x_0 from the basis again. After this, the basis is a basis of the original problem, with a feasible basic solution.

Complexity of the simplex method

- Each pivot step takes $O(mn)$ algebraic operations.
- How many pivot steps? Can be exponential.
  Does not occur in practice, where the number of needed iterations is rarely higher than $3\max(m, n)$. Does not occur on “random” problems, but mathematically random problems are not typical in practice.
- Spielman-Teng: on a small random perturbation of a linear program (a certain version of) the simplex algorithm terminates in polynomial time (on average).

Polynomial algorithm

Is there a polynomial algorithm for linear programming? Two ways to make the question precise:

- Is there an algorithm with number of algebraic operations and comparisons polynomial in $m + n$? The answer is not known.
- Is there an algorithm with number of bit operations polynomial in the length of input (measured in bits)? The answer is yes. We will see such an algorithm; however, it is rarely competitive in practice.

Duality

Primal (standard form): maximize $\boldsymbol{c}^T\boldsymbol{x}$ subject to $\boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{b}$ and $\boldsymbol{x} \geqslant \boldsymbol{0}$.
Value of the optimum (if feasible): z^*. Dual:

$$\begin{array}{rrcl} & \boldsymbol{A}^T\boldsymbol{y} & \geqslant & \boldsymbol{c} \\ & \boldsymbol{y} & \geqslant & \boldsymbol{0} \\ \min & \boldsymbol{b}^T\boldsymbol{y} & & \end{array} \qquad \begin{array}{rrcl} & \boldsymbol{y}^T\boldsymbol{A} & \geqslant & \boldsymbol{c}^T \\ & \boldsymbol{y}^T & \geqslant & \boldsymbol{0} \\ \min & \boldsymbol{y}^T\boldsymbol{b} & & \end{array}$$

Value of the optimum if feasible: t^*.

Proposition (Weak duality) $z^* \leqslant t^*$, moreover for every pair of feasible solutions $\boldsymbol{x}$, $\boldsymbol{y}$ of the primal and dual:

$$\boldsymbol{c}^T\boldsymbol{x} \leqslant \boldsymbol{y}^T\boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{y}^T\boldsymbol{b} = \boldsymbol{b}^T\boldsymbol{y}. \tag{6}$$

Use of duality. If somebody offers you a feasible solution to the dual, you can use it to upperbound the optimum of the primal (and for example decide that it is not worth continuing the simplex iterations).

Interpretation:

- b_i = the total amount of resource i that you have (kinds of workers, land, machines).
- a_{ij} = the amount of resource i needed for activity j.
- c_j = the income from a unit of activity j.
- x_j = amount of activity j.

$\boldsymbol{Ax} \leqslant \boldsymbol{b}$ says that you can use only the resources you have.
Primal problem: maximize the income $\boldsymbol{c}^T\boldsymbol{x}$ achievable with the given resources.
Dual problem: Suppose that you can buy lacking resources and sell unused resources.

Resource i has price y_i. Total income:

$$L(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{c}^T\boldsymbol{x} + \boldsymbol{y}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) = (\boldsymbol{c}^T - \boldsymbol{y}^T\boldsymbol{A})\boldsymbol{x} + \boldsymbol{y}^T\boldsymbol{b}.$$

Let

$$f(\hat{\boldsymbol{x}}) = \inf_{\boldsymbol{y}\geqslant\boldsymbol{0}} L(\hat{\boldsymbol{x}},\boldsymbol{y}) \leqslant L(\hat{\boldsymbol{x}},\hat{\boldsymbol{y}}) \leqslant \sup_{\boldsymbol{x}\geqslant\boldsymbol{0}} L(\boldsymbol{x},\hat{\boldsymbol{y}}) = g(\hat{\boldsymbol{y}}).$$

Then $f(\boldsymbol{x}) > -\infty$ needs $\boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{b}$. Hence if the primal is feasible then for the optimal $\boldsymbol{x}^*$ (choosing $\boldsymbol{y}$ to make $\boldsymbol{y}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^*) = 0$) we have

$$\sup_{\boldsymbol{x}} f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x}^* = z^*.$$

Similarly $g(\boldsymbol{y}) < \infty$ needs $\boldsymbol{c}^T \leqslant \boldsymbol{y}^T\boldsymbol{A}$, hence if the dual is feasible then we have

$$z^* \leqslant \inf_{\boldsymbol{y}} g(\boldsymbol{y}) = (\boldsymbol{y}^*)^T\boldsymbol{b} = t^*.$$

Complementary slackness conditions:

$$\boldsymbol{y}^T(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) = \boldsymbol{0}, \quad (\boldsymbol{y}^T\boldsymbol{A} - \boldsymbol{c}^T)\boldsymbol{x} = \boldsymbol{0}.$$

Proposition Equality of the primal and dual optima implies complementary slackness.

Interpretation:

- Inactive constraints have shadow price $y_i = 0$.
- Activities that do not yield the income required by shadow prices have level $x_j = 0$.

Theorem (Strong duality) The primal problem has an optimum if and only if the dual is feasible, and we have

$$z^* = \max \boldsymbol{c}^T \boldsymbol{x} = \min \boldsymbol{y}^T \boldsymbol{b} = t^*.$$

This surprising theorem says that there is a set of prices (called shadow prices) which will force you to use your resources optimally. Many interesting uses and interpretations, and many proofs.

Our proof of strong duality uses the following result of the analysis of the simplex algorithm.

Theorem If there is an optimum v then there is a basis $B \subset \{1, \ldots, m+n\}$ belonging to a basic feasible solution, and coefficients $\tilde{c}_i \leqslant 0$ such that

$$\boldsymbol{c}^T \boldsymbol{x} = v + \tilde{\boldsymbol{c}}^T \boldsymbol{x},$$

is an identity for the variable $\boldsymbol{x}$, where $\tilde{c}_i = 0$ for $i \in B$.

For the proof, define the nonnegative variables

$$\tilde{y}_i = -\tilde{c}_{n+i} \qquad\qquad i = 1, \ldots, m.$$

For any $\boldsymbol{x}$, the following transformation holds, where $i = 1, \ldots, m$, $j = 1, \ldots, n$:

$$\begin{aligned}
\sum_j c_j x_j &= v + \sum_j \tilde{c}_j x_j + \sum_i \tilde{c}_{n+i} x_{n+i} \\
&= v + \sum_j \tilde{c}_j x_j + \sum_i (-\tilde{y}_i)(b_i - \sum_j a_{ij} x_j) \\
&= v - \sum_i b_i \tilde{y}_i + \sum_j (\tilde{c}_j + \sum_i a_{ij} \tilde{y}_i) x_j.
\end{aligned}$$

This is an identity for $\boldsymbol{x}$, so the coefficients of the two sides must match: $0 = v - \sum_i b_i \tilde{y}_i$, and also $c_j = \tilde{c}_j + \sum_i a_{ij} \tilde{y}_i$.
Optimality implies $\tilde{c}_j \leqslant 0$, which implies that $\tilde{y}_i$ is a feasible solution of the dual.

Linear programming and linear inequalities

Any feasible solution of the set of inequalities

$$\begin{aligned} \boldsymbol{A}\boldsymbol{x} \qquad\quad &\leqslant \boldsymbol{b} \\ \boldsymbol{A}^T\boldsymbol{y} &\geqslant \boldsymbol{c} \\ \boldsymbol{c}^T\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{y} &= 0 \\ \boldsymbol{x}, \qquad \boldsymbol{y} &\geqslant \boldsymbol{0} \end{aligned}$$

gives an optimal solution to the original linear programming problem.

Theory of alternatives

Theorem (Farkas Lemma, not as in the book) A set of inequalities $\boldsymbol{Ax} \leqslant \boldsymbol{b}$ is unsolvable if and only if a positive linear combination gives a contradiction: there is a solution $\boldsymbol{y} \geqslant \boldsymbol{0}$ to the inequalities

$$\boldsymbol{y}^T \boldsymbol{A} = 0,$$
$$\boldsymbol{y}^T \boldsymbol{b} = 1.$$

For proof, translate the problem to finding an initial feasible solution to standard linear programming.

We use the homework allowing variables without nonnegativity constraints:

$$\begin{array}{ll} \text{maximize} & z \\ \text{subject to} & \boldsymbol{A}\boldsymbol{x} + z \cdot \boldsymbol{e} \leqslant \boldsymbol{b} \end{array} \tag{7}$$

Here, $\boldsymbol{e}$ is the vector consisting of all 1's. The dual is

$$\begin{array}{ll} \text{minimize} & \boldsymbol{y}^T \boldsymbol{b} \\ \text{subject to} & \boldsymbol{y}^T \boldsymbol{A} = \boldsymbol{0} \\ & \boldsymbol{y}^T \boldsymbol{e} = 1 \\ & \boldsymbol{y}^T \geqslant \boldsymbol{0} \end{array} \tag{8}$$

The original problem has no feasible solution if and only if $\max z < 0$ in (7). In this case, $\min \boldsymbol{y}^T \boldsymbol{b} < 0$ in (8). Condition $\boldsymbol{y}^T \boldsymbol{e} = 1$ is not needed. If we drop it then we can scale $\boldsymbol{y}$ to have $\boldsymbol{y}^T \boldsymbol{b} = -1$.

Separating hyperplane

Vectors $\boldsymbol{u}_1, \dots, \boldsymbol{u}_m$ in an n-dimensional space. Let L be the set of convex linear combinations of these points: $\boldsymbol{v}$ is in L if

$$\sum_j y_i \boldsymbol{u}_i = \boldsymbol{v}, \quad \sum_i y_i = 1, \quad \boldsymbol{y} \geqslant 0.$$

Using matrix $\boldsymbol{U}$ with rows $\boldsymbol{u}_i^T$:

$$\boldsymbol{y}^T \boldsymbol{U} = \boldsymbol{v}^T, \quad \sum_i y_i = 1, \quad \boldsymbol{y} \geqslant 0. \tag{9}$$

If $\boldsymbol{v} \notin L$ then we can put between L and $\boldsymbol{v}$ a hyperplane with equation $\boldsymbol{d}^T \boldsymbol{v} = c$. Writing $\boldsymbol{x}$ in place of $\boldsymbol{d}$ and z in place of c, this says that the following set of inequalities has a solution for $\boldsymbol{x}, z$:

$$\boldsymbol{u}_i^T \boldsymbol{x} \leqslant z \quad (i = 1, \dots, m), \quad \boldsymbol{v}^T \boldsymbol{x} > z.$$

Can be derived from the Farkas Lemma.

Complementary slackness, geometrically

Assume that the hyperplane $\boldsymbol{d}^T\boldsymbol{v} = c$ actually touches the set L, that is c is as small as possible (supporting hyperplane). Then there are $\boldsymbol{d}, \boldsymbol{y}, c$ with the properties

$$\begin{aligned} \boldsymbol{d}^T\boldsymbol{u}_i &\leqslant c \\ \boldsymbol{d}^T(\sum_i y_i\boldsymbol{u}_i) &= c, \\ \sum_i y_i &= 1, \\ \boldsymbol{y} &\geqslant 0. \end{aligned}$$

Then for all those constraints $\boldsymbol{d}^T\boldsymbol{u}_i \leqslant c$ that are not tight, the coefficient y_i is 0. In other words, the optimal solution is already a convex combination of those extremal elements of L that are on the hyperplane $\boldsymbol{d}^T\boldsymbol{v} = c$.

Application to games

Primal, with dual variables written in parentheses at end of lines:

$$\begin{array}{lllr}
\text{minimize} & t & & \\
\text{subject to} & t - \sum_j a_{ij} q_j \geqslant 0 & i = 1, \ldots, m & (p_i) \\
 & \sum_j q_j = 1, & & (z) \\
 & q_j \geqslant 0, & j = 1, \ldots, n &
\end{array}$$

Dual:

$$\begin{array}{lll}
\text{maximize} & z & \\
\text{subject to} & \sum_i p_i = 1, & \\
 & -\sum_i a_{ij} p_i + z \leqslant 0, & j = 1, \ldots, n \\
 & p_i \geqslant 0 & i = 1, \ldots, m.
\end{array}$$

Dual for max-flow: min-cut

$$\begin{array}{lrcll}
\text{maximize} & \sum_{v\in V} f(s,v) & & & \\
\text{subject to} & f(u,v) & \leqslant & c(u,v), & u,v \in V, \\
 & f(u,v) & = & -f(v,u), & u,v \in V, \\
 & \sum_{v\in V} f(u,v) & = & 0, & u \in V \setminus \{s,t\}.
\end{array}$$

Two variables associated with each edge, $f(u,v)$ and $f(v,u)$. Simplify. Order the points arbitrarily, but starting with s and ending with t. Leave $f(u,v)$ when $u<v$: whenever $f(v,u)$ appears with $u<v$, replace with $-f(u,v)$.

$$\begin{array}{lrll}
\text{maximize} & \sum_{v>s} f(s,v) & & \\
\text{subject to} & f(u,v) & \leqslant c(u,v), & u<v, \\
& -f(u,v) & \leqslant c(v,u), & u<v, \\
& \sum_{v>u} f(u,v) - \sum_{v<u} f(v,u) = & 0, & u \in V \setminus \{s,t\}.
\end{array}$$

Some constraints disappeared but others appeared, since in case of $u<v$ the constraint $f(v,u) \leqslant c(v,u)$ is written now $-f(u,v) \leqslant c(u,v)$. A dual variable for each constraint. For $f(u,v) \leqslant c(u,v)$, call it $y(u,v)$, for $-f(u,v) \leqslant c(v,u)$, call it $y(v,u)$. For

$$\sum_{v>u} f(u,v) - \sum_{v<u} f(v,u) = 0$$

call it $y(u)$.

Dual constraint for each primal variable $f(u,v)$, $u<v$. Since $f(u,v)$ is not restricted by sign, the dual constraint is an equation. If $u,v \neq s$ then $f(u,v)$ has coefficient 0 in the objective function. The equation for $u \neq s, v \neq t$ is $y(u,v) - y(v,u) + y(u) - y(v) = 0$.
For $u = s, v \neq t$: $y(s,v) - y(v,s) - y(v) = 1$.
For $u \neq s$ but $v = t$, $y(u,t) - y(t,u) + y(u) = 0$.
For $u = s, v = t$: $y(s,t) - y(t,s) = 1$.
Setting $y(s) = -1$, $y(t) = 0$, all these equations can be summarized in

$$y(u,v) - y(v,u) = y(v) - y(u).$$

The objective function to minimize is $\sum_{u \neq v} c(u,v)y(u,v)$

$$= \sum_{u<v} y(v,u)(c(u,v) + c(v,u)) + c(u,v)(y(v) - y(u))$$
$$= \sum_{u<v} y(u,v)(c(u,v) + c(v,u)) + c(v,u)(y(u) - y(v)).$$

For each $u<v$, minimize the corresponding term while keeping $y(v) - y(u)$ fixed. If $y(v) \geqslant y(u)$ then making $y(v,u) = 0$ still leaves $y(u,v) \geqslant 0$. The term becomes $c(u,v)(y(v) - y(u))$.
If $y(v)<y(u)$ then make $y(u,v) = 0$ to get $c(v,u)(y(u) - y(v))$. The objective becomes

$$\sum_{u \neq v} c(u,v)|y(v) - y(u)|^+$$

where $|x|^+ = \max(x, 0)$, subject to $y(s) = -1$ $y(t) = 0$. Require $y(s) = 0$, $y(t) = 1$ instead; the problem remains the same.

Claim There is an optimal solution in which each $y(u)$ is 0 or 1.

Proof. Assume that there is an $y(u)$ that is not 0 or 1. If it is outside the interval $[0, 1]$ then moving it towards this interval decreases the objective function, so assume they are all inside. If there are some variables $y(u)$ inside this interval then move them all by the same amount either up or down until one of them hits 0 or 1. One of these two possible moves will not increase the objective function. Repeat these actions until each $y(u)$ is 0 or 1. □

Let $\boldsymbol{y}$ be an optimal solution in which each $y(u)$ is either 0 or 1. Let

$$S = \{\, u : y(u) = 0 \,\}, \quad T = \{\, u : y(u) = 1 \,\}.$$

Then $s \in S$, $t \in T$. The objective function is

$$\sum_{u \in S, v \in T} c(u,v).$$

This is the value of the "cut" (S,T). So the dual problem is about finding a minimum cut, and the duality theorem implies the max-flow/min-cut theorem.

Maximum bipartite matching

Bipartite graph with left set A, right set B and edges $E \subseteq A \times B$. Interpretation: elements of A are workers, elements of B are jobs. $(a, b) \in E$ means that worker a has the skill to perform job b. Two edges are disjoint if both of their endpoints differ. Matching: a set M of disjoint edges. Maximum matching: a maximum-size assignment of workerst to jobs.

Covering set $C \subseteq A \cup B$: a set with the property that for each edge $(a, b) \in E$ we have $a \in C$ or $b \in C$.

Clearly, the size of each matching is $\leqslant$ the size of each covering set.

Theorem The size of a maximum matching is equal to the size of a minimum covering set.

There is a proof by reduction to the flow problem and using the max-flow min-cut theorem.

Problem (10pts) Show an example of a polyhedron determined by the set of linear inequalitites $\boldsymbol{Ax} \leqslant \boldsymbol{b}$ where $\boldsymbol{A}$ is an $m \times n$ matrix, and

- The simplex algorithm may take an exponential number of steps, as a function of $m + n$.
- Consider just the problem of deciding the feasibility of a set of inequalities

$$\boldsymbol{a}_i^T \boldsymbol{x} \leqslant b_i, \quad i = 1, \ldots, m$$

for $x \in \mathbb{R}^n$. If each entry has at most k digits then the size of the input is

$$L = m \cdot n \cdot k.$$

We want a decision in a number of steps polynomial in L, that is $O(L^c)$ for some constant c.

Ellipsoids

In space $\mathbb{R}^{\boldsymbol{n}}$, for all $r>0$ the set

$$B(\boldsymbol{c}, r) = \{\, \boldsymbol{x} : (\boldsymbol{x} - \boldsymbol{c})^T (\boldsymbol{x} - \boldsymbol{c}) \leqslant r^2 \,\}$$

is a ball with center $\boldsymbol{c}$ and radius r. A nonsingular linear transformation $\boldsymbol{L}$ transforms $B(0, 1)$ into an ellipsoid

$$E = \{\, \boldsymbol{L}\boldsymbol{x} : \boldsymbol{x}^T \boldsymbol{x} \leqslant 1 \,\} = \{\, \boldsymbol{y} : \boldsymbol{y}^T \boldsymbol{A}^{-1} \boldsymbol{y} \leqslant 1 \,\},$$

where $\boldsymbol{A} = \boldsymbol{L}^T \boldsymbol{L}$ is positive definite. A general ellipsoid $E(\boldsymbol{c}, \boldsymbol{A})$ with center $\boldsymbol{c}$ has the form

$$\{\, \boldsymbol{x} : (\boldsymbol{x} - \boldsymbol{c})^T \boldsymbol{A}^{-1} (\boldsymbol{x} - \boldsymbol{c}) \leqslant 1 \,\}$$

where $\boldsymbol{A}$ is positive definite.

Though we will not use it substantially, the following theorem shows that ellipsoids can always be brought to a simple form. A basis $\boldsymbol{b}_1, \dots, \boldsymbol{b}_n$ of the vector space $\mathbb{R}^n$ is called orthonormal if $\boldsymbol{b}_i^T \boldsymbol{b}_j = 0$ for $i \neq j$ and 1 for $i = j$.

Theorem (Principal axes) Let E be an ellipsoid with center $\mathbf{0}$. Then there is an orthonormal basis such that if vectors are expressed with coordinates in this basis then

$$E = \{\, \boldsymbol{x} : \boldsymbol{x}^T \boldsymbol{A}^{-2} \boldsymbol{x} \leqslant 1 \,\},$$

where $\boldsymbol{A}$ is a diagonal matrix with positive elements $a_1, \dots, a_n$ on the diagonal.

In other words, $E = \left\{\, \boldsymbol{x} : \frac{x_1^2}{a_1^2} + \cdots + \frac{x_n^2}{a_n^2} \leqslant 1 \,\right\}$.

In 2 dimensions this gives the familiar equation of the ellipse

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1.$$

The numbers a, b are the lengths of the principal axes of the ellipse, measured from the center. When they are all equal, we get the equation of a circle (sphere in n dimensions).

Volume of an ellipsoid

Let V_n be the volume of a unit ball in n dimensions. It is easy to see that the volume of the ellipsoid

$$E = \left\{ \boldsymbol{x} : \frac{x_1^2}{a_1^2} + \cdots + \frac{x_n^2}{a_n^2} \leqslant 1 \right\}.$$

is $\mathrm{Vol}(E) = V_n a_1 a_2 \cdots a_n$. More generally, if $E = \{ \boldsymbol{x} : x^T (\boldsymbol{A}\boldsymbol{A}^T)^{-1} \boldsymbol{x} \leqslant 1 \}$ then $\mathrm{Vol}(E) = V_n \det \boldsymbol{A}$.

Bounding the set of solutions

The set of solutions is a (possibly empty) polyhedron P. Let

$$N = n^{n/2}10^{2kn}, \quad \delta = \frac{1}{2mN}, \quad \varepsilon = \frac{\delta}{10^k n},$$
$$b'_i = b_i + \delta.$$

In preparation, we will show

Theorem

- (a) There is a ball E_1 of radius $\leqslant N\sqrt{n}$ and center $\mathbf{0}$ with the property that if there is a solution then there is a solution in E_1.
- (b) $\boldsymbol{Ax} \leqslant \boldsymbol{b}$ is solvable if and only if $\boldsymbol{Ax} \leqslant \boldsymbol{b}'$ is solvable and its set of solutions of contains a cube of size 2ε.

Consider the upper bound first. We have seen in homework the following:

Lemma If there is a solution that is a vertex then there is one with $|x_j| \leqslant N$ for all j.

Now, suppose there is a solution $\boldsymbol{z}$. For each j, if $z_j \geqslant 0$ let us introduce a new constraint $x_j \geqslant 0$, while if $z_j<0$ then introduce a constraint $x_j \leqslant 0$. It is easy to see that this new system has a solution that is a vertex.
Now for the lower bound. One of your homeworks has a problem showing the following:

Lemma If $\boldsymbol{Ax} \leqslant \boldsymbol{b}$ has no solution then defining $b'_i = b_i + \delta$, the system $\boldsymbol{Ax} \leqslant \boldsymbol{b}'$ has no solution either.

The following clearly implies ⓑ of the theorem:

Corollary If $\boldsymbol{Ax} \leqslant \boldsymbol{b}'$ is solvable then its set of solutions contains a cube of size 2ε.

Proof. If $\boldsymbol{Ax} \leqslant \boldsymbol{b}'$ is solvable then so is $\boldsymbol{Ax} \leqslant \boldsymbol{b}$. Let $\boldsymbol{x}$ be a solution of $\boldsymbol{Ax} \leqslant \boldsymbol{b}$. Then changing each x_j by any amount of absolute value at most ε changes

$$\boldsymbol{a}_i^T \boldsymbol{x} = \sum_{j=1}^{n} a_{ij} x_j$$

by at most $10^k n\varepsilon \leqslant \delta$, so each inequality $\boldsymbol{a}_i^T \boldsymbol{x} \leqslant b_i'$ still holds. □

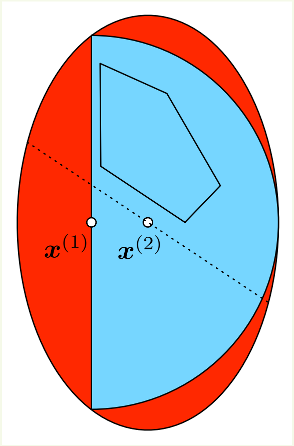
The algorithm

- The algorithm will go through a series $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \dots$ of trial solutions, and in step t learn $P \subseteq E_t$ where our wraps $E_1, E_2, \dots$ are ellipsoids.
- We start with $\boldsymbol{x}^{(1)} = \mathbf{0}$, the center of our ball. Is it a solution? If not, there is an i with $\boldsymbol{a}_i^T \boldsymbol{x}^{(1)} > b_i$. Then P is contained in the half-ball

$$H_1 = E_1 \cap \{\, \boldsymbol{x} : \boldsymbol{a}_i^T \boldsymbol{x} \leqslant \boldsymbol{a}_i^T \boldsymbol{x}^{(1)} \,\}.$$

Shrinking rate

To keep our wraps simple, we enclose H_1 into an ellipsoid E_2 of possibly small volume.



Lemma There is an ellipsoid E_2 containing H_1 with $\mathrm{Vol}(E_2) \leqslant e^{-\frac{1}{2n}} \mathrm{Vol}(E_1)$. This is true even if E_1 was also an ellipsoid.

Note $e^{-\frac{1}{2n}} \approx 1 - \frac{1}{2n}$.

Proof

Assume without loss of generality

- E_1 is the unit ball $E_1 = \{ \boldsymbol{x} : \boldsymbol{x}^T \boldsymbol{x} \leqslant 1 \}$,
- $\boldsymbol{a}_i = -\boldsymbol{e}_1$, $b_i < 0$.

Then the half-ball to consider is $\{ \boldsymbol{x} \in E_1 : x_1 \geqslant 0 \}$. The best ellipsoid's center has the form $(d, 0, \dots, 0)^T$. The axes will be $(1-d), b, b, \dots, b$, so

$$E_2 = \left\{ \boldsymbol{x} : \frac{(x_1 - d)^2}{(1-d)^2} + b^{-2} \sum_{j \geqslant 2} x_j^2 \leqslant 1 \right\}.$$

It touches the ball E_1 at the circle $x_1 = 0$, $\sum_{j \geqslant 2} x_j^2 = 1$:

$$\frac{d^2}{(1-d)^2} + b^{-2} = 1.$$

Hence

$$b^{-2} = 1 - \frac{d^2}{(1-d)^2} = \frac{1-2d}{1-2d+d^2},$$

$b^2 = 1 + \frac{d^2}{1-2d}$. Using $1 + z \leqslant e^z$:

$$\mathrm{Vol}(E_2) = V_n(1-d)b^{n-1} \leqslant V_n e^{-d + \frac{(n-1)d^2}{1-2d}} = V_n e^{-d\frac{1-(n+1)d}{1-2d}}.$$

Choose $d = \frac{1}{2(n+1)}$ to make the numerator $1/2$, then this is $V_n e^{-\frac{1}{2n}}$. This proves the Lemma for the case when E_1 is a ball. When E_1 is an ellipsoid, transform it linearly into a ball, apply the lemma and then transform back. The transformation takes ellipsoids into ellipsoids and does not change the ratio of volumes.

Bounding the number of iterations

Now the algorithm constructs E_3 from E_2 in the same way, and so on. If no solution is found, then r steps diminish the volume by a factor

$$e^{-\frac{r}{2n}}.$$

We know $\mathrm{Vol}(E_1) \leqslant V_n(N\sqrt{n})^n$, while if there is a solution then the set of solutions contains a ball of volume $\geqslant V_n\varepsilon^n$. But if r is so large that

$$e^{-\frac{r}{2n}} < \left(\frac{\varepsilon}{N\sqrt{n}}\right)^n$$

then $\mathrm{Vol}(E_{r+1})$ is smaller than the volume of this small ball, so there is no solution.

It is easy to see from here that r can be chosen to be polynomial in m, n, k.

Swept under the rug

Formula for computing the ellipsoids: Let $\boldsymbol{B}_k$ be the matrix of the kth ellipsoid E_k, with center $\boldsymbol{x}^{(k)}$:
$E_k = \{\, \boldsymbol{x} : (\boldsymbol{x} - \boldsymbol{x}^{(k)})^T \boldsymbol{B}_k^{-1} (\boldsymbol{x} - \boldsymbol{x}^{(k)}) \leqslant 1 \,\}$. Let $\boldsymbol{a}_i^T \boldsymbol{x} \leqslant b_i$ be the violated constraint. Define

$$\boldsymbol{b}_k = \frac{\boldsymbol{B}_k \boldsymbol{a}_i}{\sqrt{\boldsymbol{a}_i^T \boldsymbol{B}_k \boldsymbol{a}_i}}, \qquad \boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \frac{\boldsymbol{b}_k}{n+1},$$

$$\boldsymbol{B}_{k+1} = \frac{n^2}{n^2 - 1}\left(\boldsymbol{B}_k - \frac{2}{n+1} \boldsymbol{b}_k \boldsymbol{b}_k^T\right).$$

It can be shown that this formula is sufficient to compute with a precision that does not ruin the polynomiality of the algorithm.

Even if an exact solution exists, we only found an approximate solution. To find an exact solution (if exists) in polynomial time, ask one-by-one about each of the constraints whether it can be tight (introduce the opposite inequality), until a vertex is found (add possibly some more of constraints, of the type $x_j \geqslant 0$, $x_j \leqslant 0$). Then solve the equations.

Convex programming

Many methods and results of linear programming generalize to the case when the set of feasible solutions is convex and there is a convex function to minimize.

Definition A function $f : \mathbb{R}^n \to \mathbb{R}$ is convex if the set $\{ (\boldsymbol{x}, y) : f(\boldsymbol{x}) \leqslant y \}$ is convex. It is concave if $-f(\boldsymbol{x})$ is convex.

Equivalently, f is convex if

$$f(\lambda \boldsymbol{a} + (1 - \lambda)\boldsymbol{b}) \leqslant \lambda f(\boldsymbol{a}) + (1 - \lambda) f(\boldsymbol{b})$$

holds for all $0 \leqslant \lambda \leqslant 1$.

Examples

- Each linear function $\boldsymbol{a}^T\boldsymbol{x} + b$ is convex.
- If a matrix $\boldsymbol{A}$ is positive semidefinite then the quadratic function $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}$ is convex.
- If $f(\boldsymbol{x})$, $g(\boldsymbol{x})$ are convex and $\alpha, \beta \geqslant 0$ then $\alpha f(\boldsymbol{x}) + \beta g(\boldsymbol{x})$ is also convex.

If $f(\boldsymbol{x})$ is convex then for every constant c the set $\{\, \boldsymbol{x} : f(\boldsymbol{x}) \leqslant c \,\}$ is a convex set.

Definition A convex program is an optimization problem of the form

$$\begin{aligned} &\min f_0(\boldsymbol{x}) \\ &\text{subject to } f_i(\boldsymbol{x}) \leqslant 0 \text{ for } i = 1, \ldots, m, \end{aligned}$$

where all functions f_i for $i = 0, \ldots, m$ are convex.
More generally, we also allow constraints of the form

$$\boldsymbol{x} \in H$$

for any convex set H given in some effective way.

Example: Support vector machine

- Vectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k$ represent persons known to have ADD (attention deficit disorder). u_{ij} = measurement value of the jth psychological or medical test of person i. $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_l \in \mathbb{R}^n$ represent persons known not to have ADD.
- Separate the two groups, if possible, by a linear test: find vectors $\boldsymbol{z}, x<y$ with

$$\boldsymbol{z}^T \boldsymbol{u}_i \leqslant x \text{ for } i = 1, \ldots, k,$$
$$\boldsymbol{z}^T \boldsymbol{v}_i \geqslant y \text{ for } i = 1, \ldots, l.$$

- For $\boldsymbol{z}, x, y$ to maximize the width of the gap $\frac{y-x}{(\boldsymbol{z}^T\boldsymbol{z})^{1/2}}$, solve the convex program:

$$\begin{array}{ll} \text{maximize} & y - x \\ \text{subject to} & \boldsymbol{u}_i^T \boldsymbol{z} \leqslant x,\ i = 1, \ldots, k, \\ & \boldsymbol{v}_i^T \boldsymbol{z} \geqslant y,\ \ i = 1, \ldots, l, \\ & \boldsymbol{z}^T \boldsymbol{z} \leqslant 1. \end{array}$$

Separation oracle

For the definition of “given in an effective way”, take clue from the ellipsoid algorithm:

- We were looking for a solution to a system of linear inequalities

  $$\boldsymbol{a}_i^T \boldsymbol{x} \leqslant b_i, \quad i = 1, \dots, n.$$

  A trial solution $\boldsymbol{x}^{(t)}$ was always the center of some ellipsoid E_t. If it violated the conditions, it violated one of these: $\boldsymbol{a}_i^T \boldsymbol{x}^{(t)} > b_i$. We could then use this to cut the ellipsoid E_t in half and to enclose it into a smaller ellipsoid E_{t+1}.
- Now we are looking for an element of an arbitrary convex set H. Assume again, that at step t, it is enclosed in an ellipsoid E_t, and we are checking the condition $\boldsymbol{x}^{(t)} \in H$. How to imitate the ellipsoid algorithm further?

Definition Let $\boldsymbol{a} : \mathbb{Q}^n \to \mathbb{Q}^n$, $b : \mathbb{Q}^n \to \mathbb{Q}$ be functions computable in polynomial time and $H \subseteq \mathbb{R}^n$ a (convex) set. These are a separating (hyperplane) oracle for H if for all $\boldsymbol{x} \in \mathbb{R}^n$, with $\boldsymbol{a} = \boldsymbol{a}(\boldsymbol{x})$, $b = b(\boldsymbol{x})$ we have:

- If $\boldsymbol{x} \in H$ then $\boldsymbol{a} = \mathbf{0}$.
- If $\boldsymbol{x} \notin H$ then $\boldsymbol{a} \neq \mathbf{0}$, further $\boldsymbol{a}^T \boldsymbol{y} \leqslant b$ for all $\boldsymbol{y} \in H$ and $\boldsymbol{a}^T \boldsymbol{x} \geqslant b$.

Example For the unit ball $H = \{\, \boldsymbol{x} : \boldsymbol{x}^T \boldsymbol{x} \leqslant 1 \,\}$, the functions $\boldsymbol{a} = \boldsymbol{x} \cdot |\boldsymbol{x}^T \boldsymbol{x} - 1|^+$, and $b = \boldsymbol{x}^T \boldsymbol{x} |\boldsymbol{x}^T \boldsymbol{x} - 1|^+$ give a separation oracle. To find a separation oracle for an ellipsoid, transform it into a ball first.

Suppose that the convex set H allows a separation oracle $(\boldsymbol{a}(\cdot), b(\cdot))$. If the goal is to find an element of H then we can proceed with the ellipsoid algorithm, enclosing the convex set H into ellipsoids of smaller and smaller volume. This sometimes leads to good approximation algorithms.

Semidefinite programs

- If $\boldsymbol{A}, \boldsymbol{B}$ are symmetric matrices then $\boldsymbol{A} \preceq \boldsymbol{B}$ denotes that $\boldsymbol{B} - \boldsymbol{A}$ is positive semidefinite, and $\boldsymbol{A} \prec \boldsymbol{B}$ denotes that $\boldsymbol{B} - \boldsymbol{A}$ is positive definite.
- Let the variables x_{ij} be arranged in an $n \times n$ symmetric matrix $\boldsymbol{X} = (x_{ij})$. The set of positive semidefinite matrices

$$\{\, \boldsymbol{X} : \boldsymbol{X} \succeq \boldsymbol{0} \,\}$$

is convex. Indeed, it is defined by the set of linear inequalities

$$\boldsymbol{a}^T \boldsymbol{X} \boldsymbol{a} \geqslant 0, \text{ that is } \sum_{ij} (a_i a_j) x_{ij} \geqslant 0$$

where $\boldsymbol{a}$ runs through all vectors in $\mathbb{R}^n$.

Example: maximum cut

Recall the maximum cut problem in a graph $G = (V, E, w(\cdot))$ where w_e is the weight of edge e.
New idea:

- Assign a unit vector $\boldsymbol{u}_i \in \mathbb{R}^n$ to each vertex $i \in V$ of the graph.
- Choose a random direction through $\mathbf{0}$, that is a random unit vector $\boldsymbol{z}$. The sign of the projection on $\boldsymbol{z}$ determines the cut:

$$S = \{\, i : \boldsymbol{z}^T \boldsymbol{u}_i \leqslant 0 \,\}.$$

- The probability that $\boldsymbol{z}$ cuts $\boldsymbol{u}_i$ and $\boldsymbol{u}_j$ is

$$\arccos(\boldsymbol{u}_i^T \boldsymbol{u}_j)/\pi$$

  (draw a picture!).

Let $\alpha \approx 0.87856$ be the largest value with

$$\arccos(y)/\pi \geqslant \alpha(1-y)/2, \quad -1 \leqslant y \leqslant 1.$$

Instead of maximizing $\sum_{i \neq j} w_{ij} \arccos(\boldsymbol{u}_i^T \boldsymbol{u}_j)/\pi$, we will just maximize its lower bound

$$\alpha \sum_{i \neq j} w_{ij}(1 - \boldsymbol{u}_i^T \boldsymbol{u}_j)/2.$$

This is at least α times the value of the max cut, since if (S,T) is a cut, then setting $\boldsymbol{u}_i = \boldsymbol{e}$ for $i \in S$ and $\boldsymbol{u}_i = -\boldsymbol{e}$ for $i \in T$ we get exactly the value

$$\sum_{i \neq j} w_{ij}(1 - \boldsymbol{u}_i^T \boldsymbol{u}_j)/2.$$

$$\begin{array}{ll} \text{minimize} & \sum_{i \neq j} w_{ij} \boldsymbol{u}_i^T \boldsymbol{u}_j \\ \text{subject to} & \boldsymbol{u}_i^T \boldsymbol{u}_i = 1, \ i = 1, \ldots, n. \end{array}$$

It is more convenient to work with the variables $x_{ij} = \boldsymbol{u}_i^T \boldsymbol{u}_j$. The matrix $\boldsymbol{X} = (x_{ij})$ is positive semidefinite, with $x_{ii} = 1$, if and only if it can be represented as $x_{ij} = \boldsymbol{u}_i^T \boldsymbol{u}_j$. We arrive at the semidefinite program:

$$\begin{array}{ll} \text{minimize} & \sum_{i \neq j} w_{ij} x_{ij} \\ \text{subject to} & x_{ii} = 1, \ i = 1, \ldots, n, \\ & \boldsymbol{X} \succeq \boldsymbol{0}. \end{array}$$

Separation oracle for semidefiniteness

Please look up the the LU decomposition algorithm in these notes, when applied to positive semidefinite matrices $\boldsymbol{A}$ (Cholesky decomposition). We structured it in such a way that when it fails it gives a witness $\boldsymbol{z}$ with $\sum_{ij} z_i z_j a_{ij} < 0$. The vector $(z_i z_j)_{i,j=1}^{n}$ is the direction of the hyperplane separating the matrix $\boldsymbol{A}$ from the positive semidefinite ones. Indeed, for any positive semidefinite matrix $\boldsymbol{B}$ we have $\sum_{ij} z_i z_j b_{ij} \geqslant 0$.
Warning: All this is inexact without the estimation of the effect of roundoff errors and degree of approximation, in the precise analysis of the ellipsoid algorithm, in the context of convex optimization problems.

NP problems

Examples

- Shortest vs. longest simple paths
- Euler tour vs. Hamiltonian cycle
- 2-SAT vs. 3-SAT. Satisfiability for circuits and for conjunctive normal form (SAT). Reducing sastisfiability for circuits to 3-SAT. Use of reduction in this course: proving hardness.
- Ultrasound test of sex of fetus.

Decision problems vs. optimization problems vs. search problems.

Example Given a graph G.

Decision Given k, does G have an independent subset of size $\geqslant k$?

Optimization What is the size of the largest independent set?

Search Given k, give an independent set of size k (if there is one).

Optimization+search Give a maximum size independent set.

Random access machine

Memory: one-way infinite tape: cell i contains natural number $T[i]$ of arbitrary size.

Program: a sequence of instructions, in the "program store": a (potentially) infinite sequence of labeled registers containing instructions. A program counter.

Instruction types:

$T[T[i]] = T[T[j]]$	random access
$T[i] = T[j] \pm T[k]$	addition
`if` $T[0]>0$ `then jump to` s	conditional branching

The cost of an operation will be taken to be proportional to the total length of the numbers participating in it. This keeps the cost realistic despite the arbitrary size of numbers in the registers.

Polynomial time

Abstract problems

Instance. Solution.

Encodings

Concrete problems: encoded into strings.

Polynomial-time computable functions, polynomial-time decidable sets.

Polynomially related encodings.

Language: a set of strings. Deciding a language.

Polynomial-time verification

Example Hamiltonian cycles.

- An NP problem is defined with the help of a function

$$V(x, w)$$

  with yes/no values that verifies, for a given input x and witness (certificate) w whether w is indeed witness for x.
- It is required that $V(x, w)$ is computable as a function of the length of x. This implies that the length of the witnesses w (taken into account) is bounded polynomially in the length of x.

The same decision problem may belong to very different verification functions (search problems).

Example (Compositeness) Let the decision problem be the question whether a number x is composite (nonprime). The obvious verifiable property is

$$V_1(x,w) \Leftrightarrow (1<w<x) \wedge (w|x).$$

There is also a very different verifiable property $V_2(x,w)$ for compositeness such that, for a certain polynomial-time computable $b(x)$, if x is composite then at least half of the numbers $1 \leqslant w \leqslant b(x)$ are witnesses. This can be used for probabilistic prime number tests.

Satisfiability

- Let us use Boolean variables $x_i \in \{0,1\}$, where 0 stands for false, 1 for true. A logic expression is formed using the connectives $\wedge, \vee, \neg$: for example

$$F(x_1, x_2, x_3, x_4) = (x_1 \vee \neg x_2) \wedge (x_2 \vee \neg x_3 \vee x_4).$$

  Other connectives: say $x \Rightarrow y = \neg x \vee y$.
- An assignment (say $x_1 = 0$, $x_2 = 0$, $x_3 = 1$, $x_4 = 0$) allows to compute a value (in our example, $F(0,0,1,0) = 0$).
- An assignment (a_1, a_2, a_3, a_4) satisfies F, if $F(a_1, a_2, a_3, a_4) = 1$. The formula is satisfiable if it has some satisfying assignment.
- Satisfiability problem: given a formula $F(x_1, \ldots, x_n)$ decide whether it is satisfiable.

Special cases:

- A conjunctive normal form (CNF) $F(x_1, \ldots, x_n) = C_1 \wedge \cdots \wedge C_k$ where each C_i is a clause, with the form $C_i = \tilde{x}_{j_1} \vee \cdots \vee \tilde{x}_{j_r}$. Here each $\tilde{x}_j$ is either x_j or $\neg x_j$, and is called a literal.
  SAT: the satisfiability problem for conjunctive normal forms.
- A 3-CNF is a conjunctive normal form in which each clause contains at most 3 literals—gives rise to 3-SAT.
- 2-SAT: as seen in class, this is solvable in polynomial time.

Logic formulas, can be generalized to logic circuits.

- Acyclic directed graph, where some nodes and edges have labels. Nodes with no incoming edges are input nodes, each labeled by some logic variable $x_1, \dots, x_n$.
  Nodes with no outgoing edges are output nodes.
- Some edges have labels $\neg$. Non-input nodes are labeled $\vee$ or $\wedge$.
- Assume just one output node: the circuit C defines some Boolean function $f_C(x_1, \dots, x_n)$. Circuit satisfiability is the question of satisfiability of this function.
- Assume also that every non-input node has exactly two incoming edges.

Reducibility, completeness

Reduction of problem A_1 to problem A_2 in terms of the verification functions V_1, V_2 and a reduction (translation) function τ:

$$\exists w V_1(x, w) \Leftrightarrow \exists u V_2(\tau(x), u).$$

Example Reducing linear programming to linear programming in standard form.

NP-hardness.
NP-completeness.

Theorem Circuit satisfiability is NP-complete.

Consider a verification function $V(x,w)$. For an x of length n, to a random access machine program computing $V(x,w)$, in cost t, construct a circuit $C(x)$ of polynomial size in n,t, that computes $V(x,w)$ from any input string w. (We translated x to $C(x)$.) Now there is a witness w if and only if $C(x)$ is satisfiable.

Theorem 3-SAT is NP-complete.

Translating a circuit's local rules into a 3-CNF.

Theorem INDEPENDENT SET is NP-complete.

Reducing SAT to it.

Example

- Integer linear programming, in particular solving $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where the $m \times n$ matrix $\boldsymbol{A} \geqslant 0$ and the vector $\boldsymbol{b}$ consist of integers, and $x_j \in \{0, 1\}$.
  Case $m = 1$ is the subset sum problem.
- Reducing 3SAT to solving $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$.
- Reducing $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ to $\boldsymbol{a}^T\boldsymbol{x} = b$ (subset sum).

Example Set cover $\geqslant$ vertex cover $\sim$ independent set.

Co-NP

- Definition of the Co-NP class: L is in Co-NP if its complement is in NP. Example: logical tautologies.
- The class NP∩Co-NP. Examples: duality theorems.
- Example of a class that is in NP∩Co-NP, and not known to be in P: derived from the factorization problem.
  Let L be the set of those pairs of integers $x > y > 0$ for which there is an integer $1 < w < y$ with $w | x$. This is clearly in NP. But the complement is also in NP. A witness that there is no w with the given properties is a complete factorization

  $$x = p_1^{\alpha_1} \cdots p_k^{\alpha_k}$$

  of x, along with witnesses of the primality of $p_1, \ldots, p_k$. The latter are known to exist, by an old—nontrivial—theorem that primality is in NP.

Approximations

The setting

In case of NP-complete problems, maybe something can be said about how well we can approximate a solution. We will formulate the question only for problems, where we maximize a positive function. For object function $f(x,y)$ for $x,y \in \{0,1\}^n$, the optimum is

$$M(x) = \max_y f(x,y)$$

where y runs over the possible "witnesses".
For $0<\lambda$, an algorithm $A(x)$ is a λ-approximation if

$$f(x,A(x))>M(x)/\lambda.$$

For minimization problems, with minimum $m(x)$, we require $f(x,A(x))<m(x)\lambda$.

Greedy algorithms

Try local improvements as long as you can.

Example (Maximum cut) Graph $G = (V, E)$, cut $S \subseteq V$, $\overline{S} = V \setminus S$. Find cut S that maximizes the number of edges in the cut:

$$|\{\, \{u,v\} \in E : u \in S, v \in \overline{S} \,\}|.$$

Greedy algorithm:

Repeat: find a point on one side of the cut whose moving to the other side increases the cutsize.

Theorem If you cannot improve anymore with this algorithm then you are within a factor 2 of the optimum.

The unimprovable cut contains at least half of all edges.

Randomized algorithms

Generalize maximum cut for the case where edges e have weights w_e, that is maximize

$$\sum_{u \in S, v \in \overline{S}} w_{uv}.$$

- Question The greedy algorithm brings within factor 2 of the optimum also in the weighted case. But does it take a polynomial number of steps?
- New idea: decide each "$v \in S$?" question by tossing a coin. The expected weight of the cut is $\frac{1}{2}\sum_e w_e$, since each edge is in the cut with probability 1/2.
- We will do better with semidefinite programming.

Less greed is sometimes better

What does the greedy algorithm for vertex cover say?
Better performance guarantee by a less greedy algorithm:
`Approx_Vertex_Cover`(G):

```
    $C \leftarrow \emptyset$
    $E' \leftarrow E[G]$
    **while** $E' \neq \emptyset$ **do**
        let $(u,v)$ be an arbitrary edge in $E'$
        $C \leftarrow C \cup \{u,v\}$
        remove from $E'$ every edge incident on either $u$ or $v$
    **return** $C$
```

Theorem `Approx_Vertex_Cover` has a ratio bound of 2.

Proof. The points of C are endpoints of a matching. Any optimum vertex cover must contain half of them. □

More general vertex cover problem for $G = (V, E)$, with weight w_i in vertex i. Let $x_i = 1$ if vertex x is selected. Linear programming problem without the integrality condition:

$$\begin{array}{ll} \text{minimize} & \boldsymbol{w}^T \boldsymbol{x} \\ \text{subject to} & x_i + x_j \geqslant 1, \ (i, j) \in E, \\ & \boldsymbol{x} \geqslant \boldsymbol{0}. \end{array}$$

Let the optimal solution be $\boldsymbol{x}^*$. Choose $\overline{x}_i = 1$ if $x_i^* \geqslant 1/2$ and 0 otherwise.

Claim Solution $\overline{\boldsymbol{x}}$ has approximation ratio 2.

Proof. We increased each x_i^* by at most a factor of 2. □

The set-covering problem

Given $(X, \mathcal{F})$: a set X and a family $\mathcal{F}$ of subsets of X, find a min-size subset of $\mathcal{F}$ covering X.
Example: Smallest committee with people covering all skills.
Generalization: Set S has weight $w(S)>0$. We want a minimum-weight set cover.

The algorithm $\mathtt{Greedy\_Set\_Cover}(X, \mathcal{F})$:

```
$U \leftarrow X$
$\mathcal{C} \leftarrow \emptyset$
**while** $U \neq \emptyset$ **do**
    select an $S \in \mathcal{F}$ that maximizes $|S \cap U|/w(S)$
    $U \leftarrow U \setminus S$
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{S\}$
**return** $\mathcal{C}$
```

If element e was covered by set S then let price$(e) = \frac{w(S)}{|S \cap U|}$. Then we cover each element at minimum price (at the moment).
Note that the total final weight is $\sum_{k=1}^{n}$ price(e_k).

Analysis

Let $H(n) = 1 + 1/2 + \cdots + 1/n (\approx \ln n)$.

Theorem `Greedy_Set_Cover` has a ratio bound $\max_{S \in \mathcal{F}} H(|S|)$.

Lemma For all S in $\mathcal{F}$ we have $\sum_{e \in S} \text{price}(e) \leqslant w(S)H(|S|)$.

Proof. Let $e \in S \cap S_i \setminus \bigcup_{j<i} S_j$, and $V_i = S \setminus \bigcup_{j<i} S_j$ be the remaining part of S before e will be covered in the greedy cover. By the greedy property,

$$\text{price}(e) \leqslant w(S)/|V_i|.$$

Let $e_1, \ldots, e_{|S|}$ be a list of elements of S in the order in which they are covered (ties are broken arbitrarily), with $e_{j(k)}$ the earliest element covered along with e_k. The above inequality gives

$$\text{price}(e_k) = \text{price}(e_{j(k)}) \leqslant \frac{w(S)}{|S| - j(k) + 1} \leqslant \frac{w(S)}{|S| - k + 1}.$$

Summing for all k proves the lemma. □

Proof of the theorem. Let $\mathcal{C}_*$ be the optimal set cover and $\mathcal{C}$ the cover returned by the algorithm.

$$\sum_{e} \text{price}(e) \leqslant \sum_{S \in \mathcal{C}_*} \sum_{e \in S} \text{price}(e) \leqslant \sum_{S \in \mathcal{C}_*} w(S) H(|S|) \leqslant H(|S^*|) \sum_{S \in \mathcal{C}_*} w(S)$$

where S^* is the largest set. □

Question Is this the best possible factor for set cover?

The answer is not known.

Primal-dual schema

Let us look at the same algorithm and analysis from a different point of view. Primal (a generalized "covering problem"):

$$\begin{array}{ll} \text{minimize} & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} & \sum_{j=1}^{n} a_{ij} x_j \geqslant b_i,\ i = 1, \dots, m, \\ & \qquad x_j \geqslant\ 0,\ j = 1, \dots, n. \end{array}$$

Dual (a "packing problem"):

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^{n} b_i y_i \\ \text{subject to} & \sum_{i=1}^{n} a_{ij} y_i \leqslant c_j,\ \ j = 1, \dots, n. \\ & \qquad y_i \geqslant\ 0,\ i = 1, \dots, m. \end{array}$$

For some $\alpha, \beta \geqslant 1$, formally relax the complementary slackness:
Primal conditions: $x_j > 0 \Rightarrow c_j/\alpha \leqslant \sum_j a_{ij} y_i \leqslant c_j$.
Dual conditions: $y_i > 0 \Rightarrow b_i \leqslant \sum_j a_{ij} x_j \leqslant \beta b_i$.

Proposition If the primal and dual feasible solutions satisfy these conditions, then

$$\boldsymbol{c}^T \boldsymbol{x} \leqslant \alpha\beta \boldsymbol{b}^T \boldsymbol{y}.$$

Proof straightforward.

The primal-dual schema:

- Start from an infeasible integer primal and a feasible dual (typically $\boldsymbol{x} = \mathbf{0}$, $\boldsymbol{y} = \mathbf{0}$).
- Keep improving the feasibility of the primal, keeping it integral, and the optimality of the dual.
  The primal guides the improvements of the dual and vice versa.

Application to set cover

Set cover problem, without integrality condition. Set system $\mathcal{S}$, universe U.

$$\begin{array}{ll} \text{minimize} & \sum_S c(S) x_S \\ \text{subject to} & \sum_{S \ni e} x_S \geqslant 1,\ e \in U, \\ & x_S \geqslant 0,\ S \in \mathcal{S}, \end{array}$$

Dual with variables y_e, $e \in U$:

$$\begin{array}{ll} \text{maximize} & \sum_{e \in U} y_e \\ \text{subject to} & \sum_{e \in S} y_e \leqslant c(S),\ S \in \mathcal{S}, \\ & y_e \geqslant \quad 0,\ e \in U. \end{array}$$

Primal complementary slackness conditions for each S, with factor $\alpha = 1$: $x_S \neq 0 \Rightarrow \sum_{e \in S} y_e = c(S)$.
Set S is tight when this holds. Plan: use only tight sets.

The algorithm

1. Start with $\boldsymbol{x} = \mathbf{0}$, $\boldsymbol{y} = \mathbf{0}$.
2. Repeat, until all elements are covered:
   Pick uncovered element e, raise y_e until some set S goes tight.
   Add S to the set cover.

Since the relaxed complementary slackness conditions hold at the end, we achieved the approximation factor $\alpha\beta = 1$.

Simplicity As opposed to the simplex method, the successive improvements were not accompanied by any linear transformations accompanying a basis change.

Fully approximable version of knapsack

The knapsack problem is defined as follows.
Given: integers $b \geqslant a_1 \geqslant \ldots \geqslant a_n$, and integer weights $w_1 \geqslant \cdots \geqslant w_n$.

$$\begin{aligned} \text{maximize} \quad & \boldsymbol{w}^T\boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{a}^T\boldsymbol{x} \leqslant \quad b, \\ & x_i = 0, 1,\ i = 1, \ldots, n. \end{aligned}$$

Dynamic programming: For $1 \leqslant k \leqslant n$,

$$A_k(p) = \min\{\, \boldsymbol{a}^T \boldsymbol{x} : \boldsymbol{w}^T \boldsymbol{x} \geqslant p,\ x_{k+1} = \cdots = x_n = 0 \,\}.$$

If the set is empty the minimum is ∞, and set $A_k(x) = 0$ for $x \leqslant 0$. Let $w = w_1 + \cdots + w_n$. The vector $(A_{k+1}(0), \ldots, A_{k+1}(w))$ can be computed by a simple recursion from $(A_k(0), \ldots, A_k(w))$.

$$A_{k+1}(p) = \min\{\, A_k(p), a_{k+1} + A_k(p - w_{k+1}) \,\}.$$

The optimum is $\max\{\, p : A_n(p) \leqslant b \,\}$.
Complexity: roughly $O(nw)$ steps.
Why is this not a polynomial algorithm?

Idea for approximation: break each w_i into a smaller number of big chunks, and use dynamic programming. Let $r>0$, $w'_i = \lfloor w_i/r \rfloor$.

$$\begin{array}{ll} \text{maximize} & (\boldsymbol{w}')^T \boldsymbol{x} \\ \text{subject to} & \boldsymbol{a}^T \boldsymbol{x} \leqslant \quad b, \\ & x_i = 0, 1,\ i = 1, \ldots, n. \end{array}$$

For the optimal solution $\boldsymbol{x}'$ of the changed problem, estimate $\frac{\boldsymbol{w}^T\boldsymbol{x}'}{\mathrm{OPT}} = \frac{\boldsymbol{w}^T\boldsymbol{x}'}{\boldsymbol{w}^T\boldsymbol{x}^*}$. We have

$$\begin{aligned} \boldsymbol{w}^T\boldsymbol{x}'/r &\geqslant (\boldsymbol{w}')^T\boldsymbol{x}' \geqslant (\boldsymbol{w}')^T\boldsymbol{x}^* \geqslant (\boldsymbol{w}/r)^T\boldsymbol{x}^* - n, \\ \boldsymbol{w}^T\boldsymbol{x}' &\geqslant \mathrm{OPT} - r \cdot n = \mathrm{OPT} - \varepsilon w_1, \end{aligned}$$

where we set $r = \varepsilon w_1/n$. This gives

$$\frac{(\boldsymbol{w})^T\boldsymbol{x}'}{\mathrm{OPT}} \geqslant 1 - \frac{\varepsilon w_1}{\mathrm{OPT}} \geqslant 1 - \varepsilon.$$

With $w = \sum_i w_i$, the amount of time is of the order of

$$nw/r = n^2 w/(w_1\varepsilon) \leqslant n^3/\varepsilon,$$

which is polynomial in $n, (1/\varepsilon)$.

Classifying approximability

An approximation scheme is an algorithm that for every ε, gives an $(1 + \varepsilon)$-approximation, computable in polynomial time, if ε is fixed.

- A problem is fully approximable if it has a polynomial-time approximation scheme.
  Example: see a version KNAPSACK below.
- It is partly approximable if there is a lower bound $\lambda_{\min} > 1$ on the achievable approximation ratio.
  Example: MAXIMUM CUT, VERTEX COVER, MAX-SAT.
- It is inapproximable if even this cannot be achieved.
  Example: INDEPENDENT SET (deep result). The approximation status of this problem is different from VERTEX COVER, despite the close equivalence between the two problems.

Approximability is model-dependent

Approximability depends much on which function is chosen to optimize. Examples:

- Special case of knapsack, with $w_i = a_i$. Equivalent to minimizing $b - \sum_i \boldsymbol{a}^T \boldsymbol{x}$. The minimization is inapproximable, since the question whether the optimum is 0 is NP-complete.
- Maximum independent set is inapproximable. If k is the size of a maximum independent set, then $n - k$ is the size of a minimum vertex cover, which is partly approximable. And it indeed might happen that $k = n^{1/2}$ and we only find $n^{1/3}$. The quotient is unbounded, while $\frac{n-n^{1/3}}{n-n^{1/2}} \to 1$ as $n \to \infty$.

Counting problems: the class # P

Definition Function f is in $\#P$ if there is a polynomial-time (verifier) predicate $V(x, y)$ and polynomial $p(n)$ such that for all x we have

$$f(x) = |\{\, y : |y| \leqslant p(|x|) \wedge V(x, y) \,\}|.$$

Reduction among $\#P$ problems. The $\#P$-complete problems are all obviously NP-hard.

Repeated tests

How to aproximate a $\#P$ function?
Repeated independent tests will work only if the probability of success is not tiny. More formally, if it is not tiny compared to the standard deviation. Look at Chebysev's inequality, say. Let $X_1, \dots, X_N$ be i.i.d. random variables with variance σ^2 and expected value μ. Then the inequality says

$$\mathbb{P}[|\sum_i X_i/N - \mu| > t\sigma] \leqslant t^{-2}/N.$$

Suppose we want to estimate μ within a factor of 2, so let $t\sigma = \mu/2$, then $t = \mu/(2\sigma)$,

$$\mathbb{P}[|\sum_i X_i/N - \mu| > \mu/2] \leqslant (1/N)(2\sigma/\mu)^2.$$

This will converge slowly if σ/μ is large.

Example $X_i = 1$ with probability p and 0 otherwise. Then $\sigma^2 = p(1-p)$, our bound is $4(1-p)/(pN)$, so we need $N>1/p$ if p is small.

DNF satisfaction

Suppose we want to find the number of satisfying assignments of a disjunctive normal form

$$C_1 \vee \cdots \vee C_m.$$

More generally, suppose we need to estimate $|S|$ where

$$S = S_1 \cup \cdots \cup S_m.$$

Suppose that

- We can generate uniformly the elements of S_i for each i.
- We know (can compute in polynomial time) $|S_i|$.
- For each element x, we know

$$c(x) = |\{\, i : x \in S_i \,\}|.$$

Then we know $M = \sum_i |S_i|$, but we want to know $|S|$.

Pick $I \in \{1, \ldots, m\}$ such that $\mathbb{P}[I = i] = |S_i|/M$. Pick an element $X \in S_I$ uniformly. Then for each x we have

$$\mathbb{P}[X = x] = \sum_{S_i \ni x} \mathbb{P}[I = i] \, \mathbb{P}[X = x | I = i] = \sum_{S_i \ni x} \frac{|S_i|}{M} \frac{1}{|S_i|} = c(x)/M.$$

Let $Y = M/c(X)$, then

$$E(Y) = \sum_{x \in S} \frac{M}{c(x)} \, \mathbb{P}[X = x] = |S|.$$

On the other hand, $0 \leqslant Y \leqslant M$, so $\sigma \leqslant M \leqslant m|S|$, therefore $\sigma/\mu \leqslant m$, so sampling will converge fast.
We found a FPRAS (fully polynomial randomized approximation scheme) for counting the DNF solutions.