CS 530 Advanced algorithms

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

Péter Gács

Computer Science Department
Boston University

Spring 2018

The class structure

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

Graphs, shortest paths

(Review of material from Data Structures and Algorithms courses.)
Graph $G = (V, E)$. V is the set of vertices (points), E is the set of edges (lines). Directed or undirected graph.
Paths.
Algorithms to

- find a shortest (directed) path from a point s to another, t.
- find all points reachable from point s.

Method:

- breadth-first search
- remember with each newly reached point v the point u from which you arrived at it.

If edges have lengths, a little more complex algorithm: Dijkstra's.

Efficient algorithms

- Asking for an algorithm, we generally understand an efficient algorithm: better than brute-force.
- We measure the efficiency of an algorithm generally by the running time as a function of the length L of its input data (number of bits representing it).
  A "brute-force" algoritm has running time that is typically exponential in L, that is of the form 2^{cL}.
  We are looking for algorithms with running time that is polynomial in L, that is of the form $O(L^c)$.

Example 2.1 When looking for a shortest path from s to t, searching through all possible paths leaving s would be a brute-force algorithm.
Breadth-first search is, on the other hand, polynomial.

Examples 2.2

1. n workers and n jobs. Each worker is capable of performing some of the jobs. Is it possible to assign each worker to a different job, so that workers get jobs they can perform?
2. At a dance party, with 300 students, every boy knows 50 girls and every girl knows 50 boys. Can they all dance simultaneously so that only pairs who know each other dance with each other?

It depends. If each worker is familiar only with the same one job (say, digging), then no.

- Bipartite graph: left set L (of girls), right set R (of boys).
- Matching, perfect matching.

Theorem 2.3 If every node of a bipartite graph has the same degree $d \geq 1$ then it contains a perfect matching.

Examples showing the (local) necessity of the conditions:

- Bipartiteness is necessary, even if all degrees are the same.
- Bipartiteness and positive degrees is insufficient.

Maximum matching

Let us just look for a maximum-size matching, even if it might not be perfect. What is the best we can hope for?

Definition 2.4 A set W of vertices in a graph is called a vertex cover if every edge has at least one end in S.

If our graph has a vertex cover W of size k then clearly there is no matching of size $k + 1$. So the maximum size of a matching is at most as large as the minimum size of vertex cover. Surprisingly, there is an equality.

Theorem 2.5 (Vertex cover, by König-Egerváry) In a bipartite graph, the size of a maximum matching is equal to the size of a minimum vertex cover.

We will obtain the proof along with an algorithm to find the maximum matching.

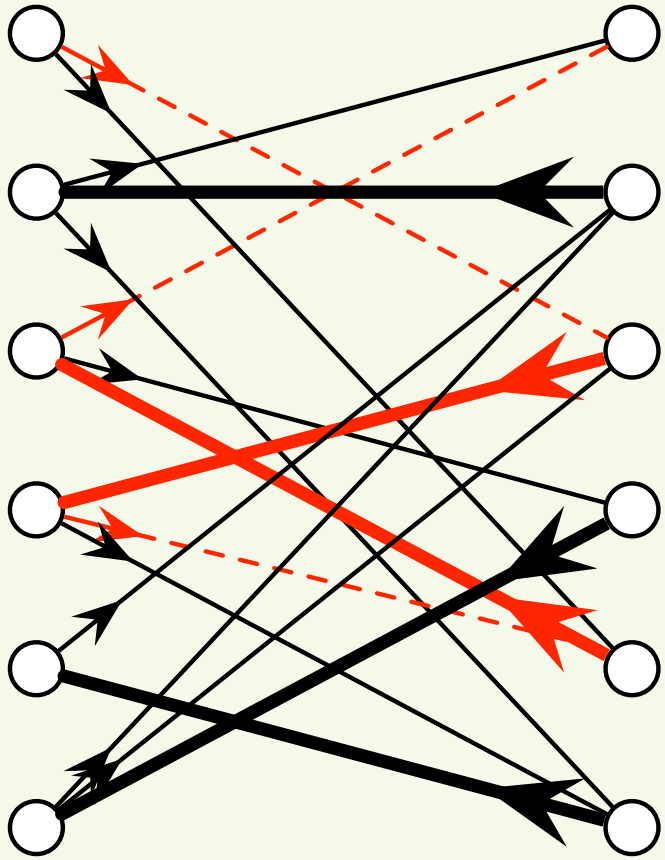
Enlarging a matching

Greedy matching method: just keep adding edges to M as long as we can. We may get stuck with a maximal (unextendable) matching that is not perfect, does not have the maximum number of edges.

Augmenting paths

New way to enlarge a matching M:

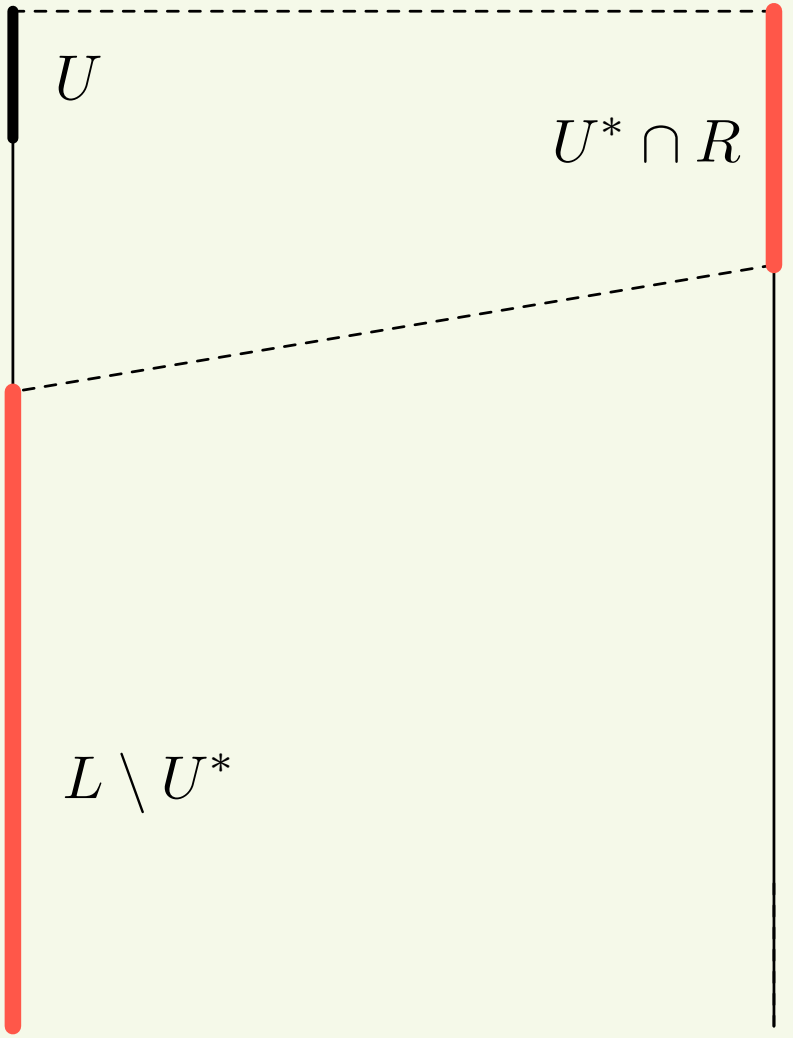
- G_M is a graph obtained from G by directing the edges of M to left, the others to right.
- Augmenting path: a directed path of G_M that starts in L, ends in R, both outside M. To augment, switch the M and non-M edges.



Augment a matching M

From the set U of unmatched points in L, gradually build:
set S reachable on directed paths in G_M.
function $f : S \setminus U \to S$ where $f(s)$ = previous point on path.
$S \leftarrow U, f \leftarrow$ the empty function
while not stopped **do**
    Look for an edge sr between $s \in S \cap L$ and $r \in R \setminus S$
    if there is none **then**
        M is a maximum matching, **return**
    else
        $S \leftarrow S \cup \{r\}, f(r) \leftarrow s$
        if r matched to some $q \in L$ **then**
            $S \leftarrow S \cup \{q\}, f(q) \leftarrow r$
        else
            trace back path P from r to U using $f(\cdot)$
            switch edges on P to increase M
            return

Lemma 2.6 If M has no augmenting path, then there is a vertex cover of size $|M|$.



Proof. U := the unmatched points of L,
U^* := points reachable in G_M from U.
Claim: $(U^* \cap R) \cup (L \setminus U^*)$ is

- a vertex cover
- has size $|M|$.

Check both statements! □

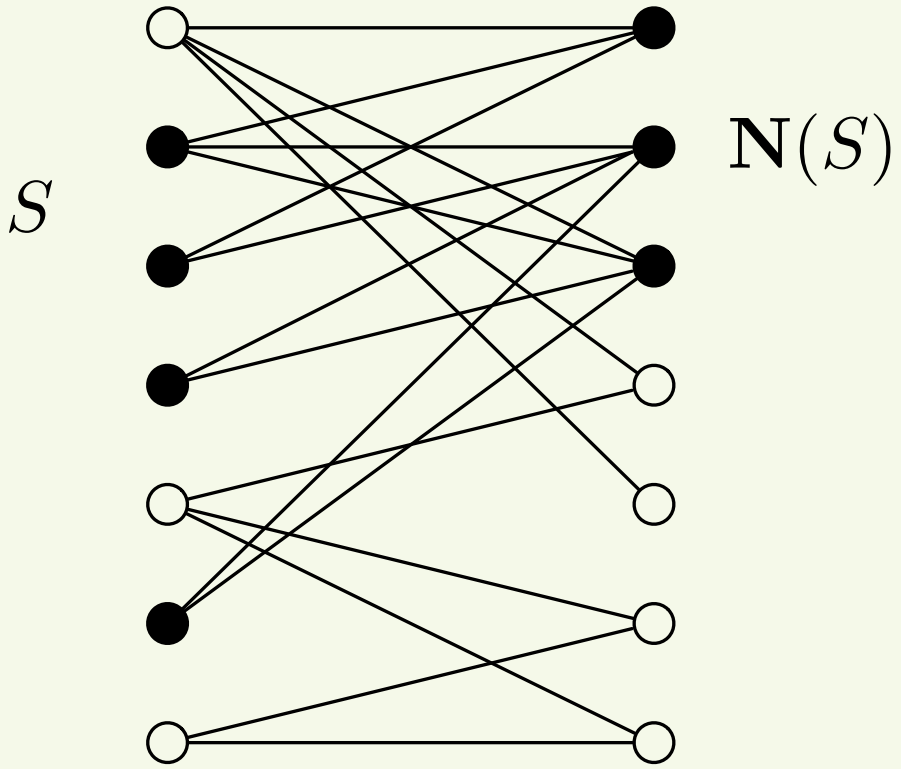
Let us get a criterion for perfect matching. For $S \subseteq L$ let

$$\mathbf{N}(S) \subseteq R$$

be the set of all neighbors of the nodes of S.

Theorem 2.7 (The Marriage Theorem) A bipartite graph has a perfect matching if and only if $|L| = |R|$ and for every $S \subseteq L$ we have $|\mathbf{N}(S)| \geq |S|$.

Proof. The condition is obviously necessary.
Now assume that there is no perfect matching; then the Vertex Cover Theorem gives a vertex cover W of size $< n$. Let $S = L \setminus W$, then it is easy to see that $|\mathbf{N}(S)| < |S|$. □



There is no perfect matching here, since $|S| = 4$, $|\mathbf{N}(S)| = 3$.

Example 2.8 6 tribes partition an island into hunting territories of 100 square miles each. 6 species of tortoise, with disjoint habitats of 100 square miles each.

Can each tribe pick a tortoise living on its territory, with different tribes choosing different totems?

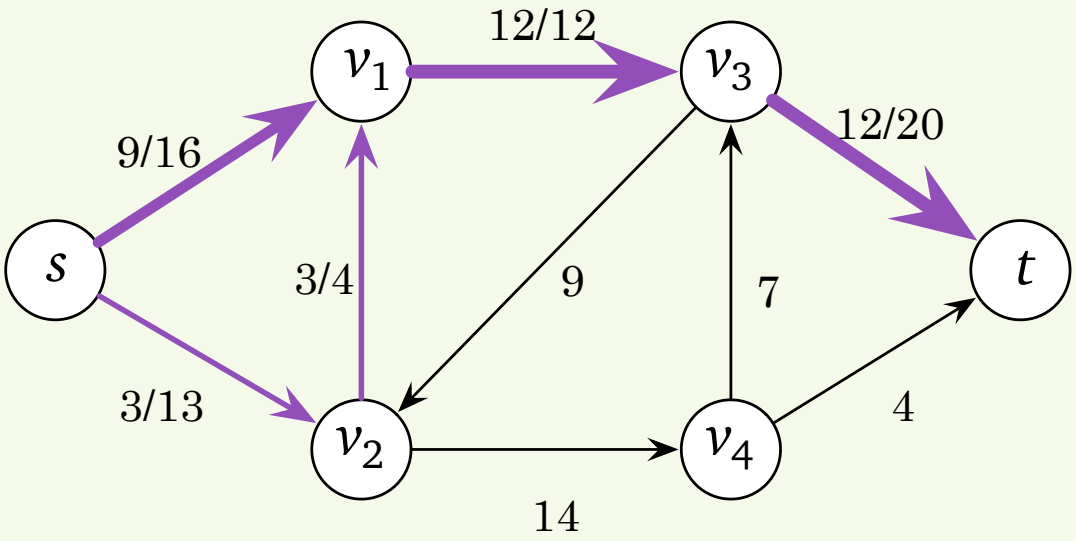
Yes, by the Marriage Theorem. Indeed, the combined hunting area of any k tribes intersects with at least k tortoise habitats.

Flow networks

- Directed graph. Source s, sink t. Every vertex is on some path from s to t.
- Flow function: $f(u, v)$ on all edges (u, v) showing the amount of material going from u to v. We are only interested in the net flow $\hat{f}(u, v) = f(u, v) - f(v, u)$: then $\hat{f}(v, u) = -\hat{f}(u, v)$. So we simply require

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

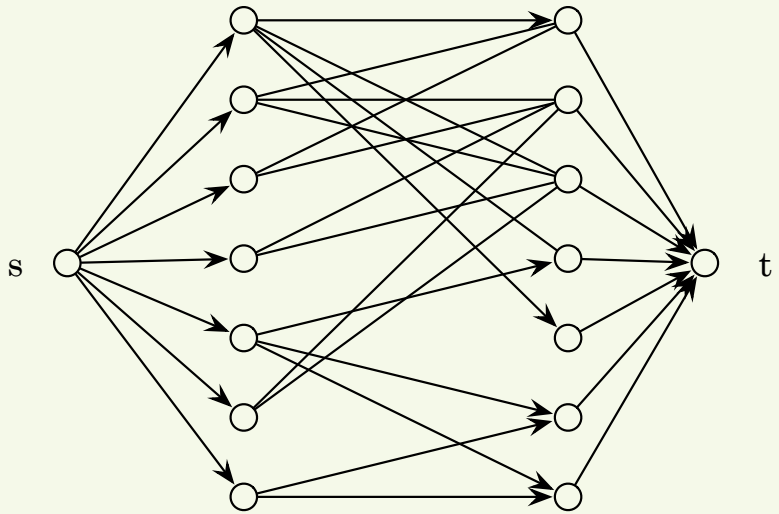
- Excess at point v: $e_f(v) = \sum_u f(u, v)$. Its negative is called the deficit.
- Each edge (u, v) imposes a capacity $c(u, v) \geq 0$ on the flow: $f(u, v) \leq c(u, v)$. (We may have $c(u, v) \neq c(v, u)$.)



The notation f/c means flow f along an edge with capacity c.

- The flow function is called an s-t pre-flow if all excesses other than at s are nonnegative.
  The value $|f|$ of the pre-flow is the excess $e_f(t)$.
- If $e_f(v) > 0$ only for $v = t$ then f is called an s-t flow.
- Our goal is to maximize the value $|f|$ of an s-t flow.
  (It is also the maximum s-t pre-flow.)

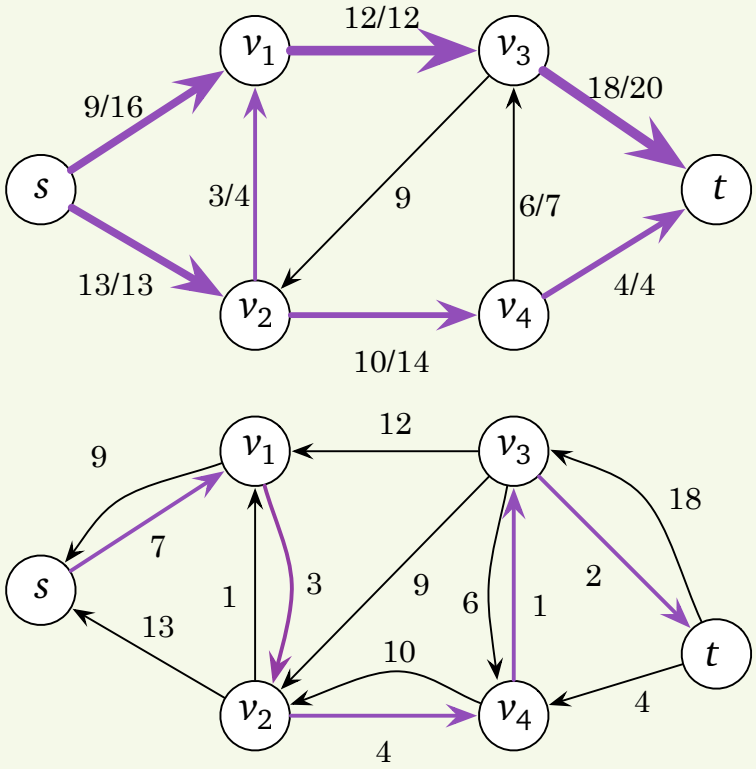
Application to matching



- n points on left, n on right. Edges directed to right, with unit capacity from s to A and from B to t. and any capacity ≥ 1 (even ∞) between A and B.
- Perfect matching $\rightarrow$ flow of value n.
- Flow of value $n \rightarrow$ perfect matching? Not always, but fortunately (as will be seen), there is always an integer maximum flow.

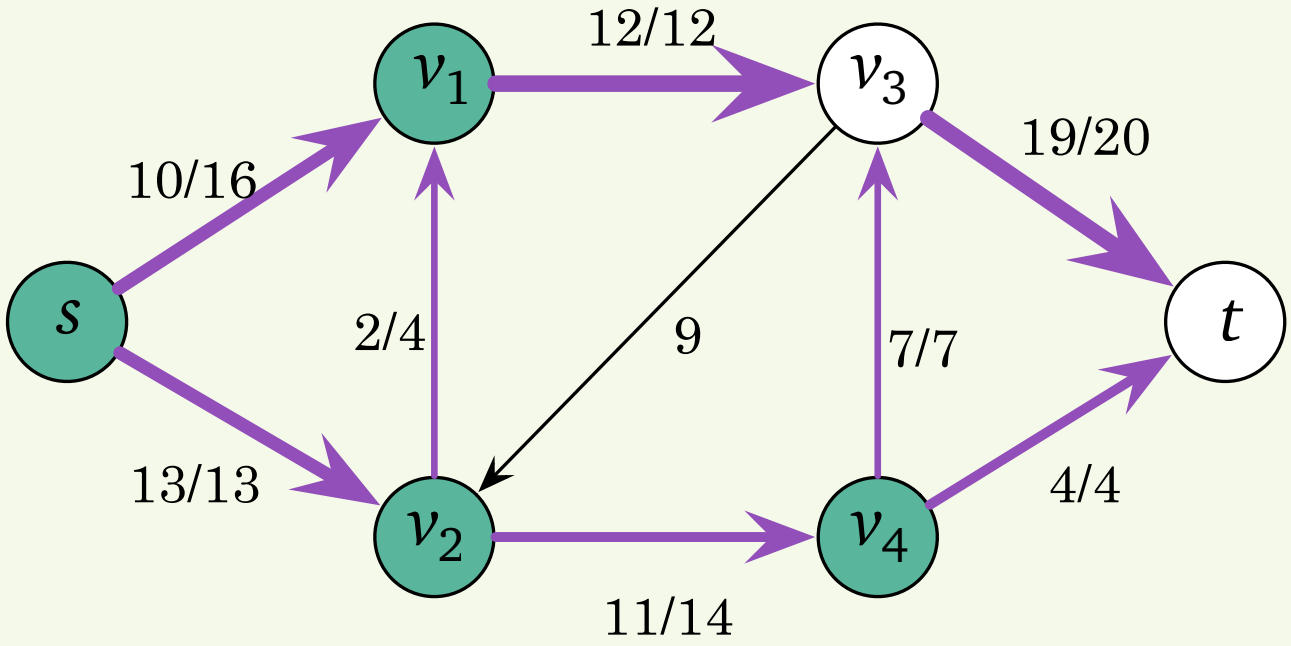
Residual network, augmenting path

Given a pre-flow f, residual capacity $c_f(u, v) = c(u, v) - f(u, v)$. Makes sense even with negative $f(u, v)$. The residual network G_f may have edges (with positive capacity) that were not in the original network. An augmenting path is an s-t path in G_f (with some flow along it). (How does it change the original flow?)



G_f has the edges along which flow can still be sent.
If $f(u, v) > 0$ then sending from v to u means decreasing $f(u, v)$.

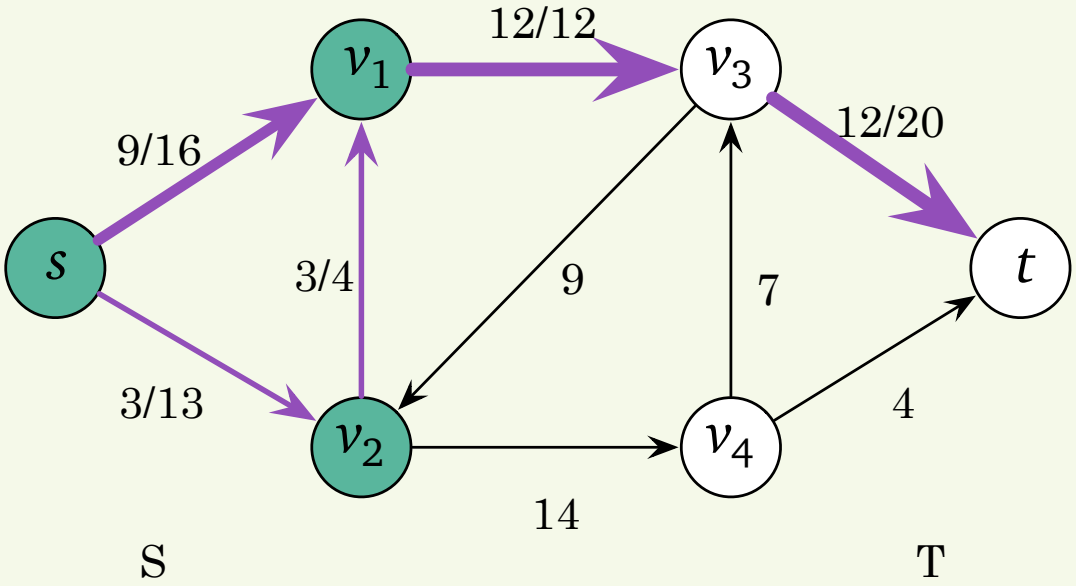
We obtained:



This cannot be improved: look at the cut (S, T) with $T = \{v_3, t\}$.

Cuts

Cut (S, T) is a partition of V with $s \in S$, $t \in T$.
Net flow $f(S, T) = \sum_{u \in S, v \in T} f(u, v)$.
Capacity $c(S, T) = \sum_{u \in S, v \in T} c(u, v)$. Obviously, $f(S, T) \leq c(S, T)$.



In this example, $c(S, T) = 26$, $f(S, T) = 12$.

Lemma 2.9 $f(S, T) = |f|$, the value of the flow.

Corollary 2.10 The value of any flow is bounded by the capacity of any cut.

Theorem 2.11 (Max-flow, min-cut) The following properties of a flow f are equivalent.

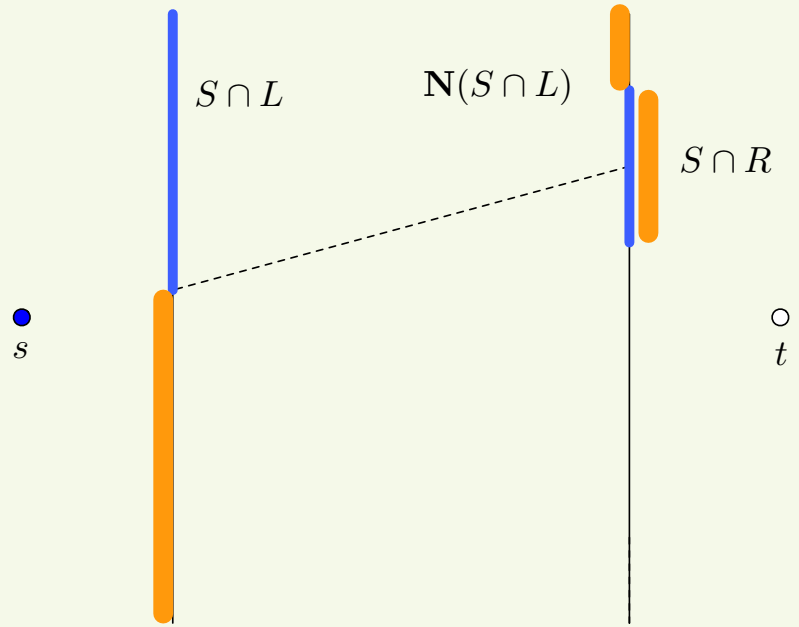
1. $|f| = c(S, T)$ for some cut (S, T).
2. f is a maximum flow.
3. There are no augmenting paths to f.

The equivalence of the first two statements says that the size of the maximum flow is equal to the size of the minimum cut.

Proof: 1⇒2 and 2⇒3 are obvious. The crucial step is 3⇒1. Given f with no augmenting paths, we construct (S, T): let S be the nodes reachable from s in the residual network G_f.

Cut to vertex cover

- Edges from $T \cap L$ meet cut $(\{s\}, T)$.
- Edges to $\mathbf{N}(S \cap L) \cap T$ in cut $(S \cap L, T)$.
- Edges to $S \cap R$ meet cut $(S, \{t\})$.



So $(T \cap L) \cup \mathbf{N}(S \cap L) \cup (S \cap R)$ is a vertex cover, size is $\leq c(\{s\}, T) + c(S \cap L, T) + c(S \cap R, \{t\}) = c(S, T)$.

Dinic-Edmonds-Karp algorithm

- Does the Ford-Fulkerson algorithm terminate? Not necessarily (if capacities are not integers), unless we choose the augmenting paths carefully.
- Integer capacities: always terminates, but may take exponentially long.
  Network derived from the bipartite matching problem: each capacity is 1, so we terminate in polynomial time.
- Dinic-Edmonds-Karp: use breadth-first search for the augmenting paths. Why should this terminate?

Lemma 2.12 In the Edmonds-Karp algorithm, the shortest-path distance $\delta_f(s, v)$ increases monotonically with each augmentation.

Proof: Let $\delta_f(s, u)$ be the distance of u from s in G_f, and let f' be the augmented flow. Assume, by contradiction $\delta_{f'}(s, v) < \delta_f(s, v)$ for some v: let v be the one among these with smallest $\delta_{f'}(s, v)$. Let $u \longrightarrow v$ be be a shortest path edge in $G_{f'}$, and

$$d := \delta_f(s, u)(= \delta_{f'}(s, u)), \text{ then } \delta_{f'}(s, v) = d + 1.$$

Edge (u, v) is new in $G_{f'}$; so (v, u) was a shortest path edge in G_f, giving $\delta_f(s, v) = d - 1$. But $\delta_{f'}(s, v) = d + 1$ contradicts $\delta_{f'}(s, v) < \delta_f(s, v)$.

An edge is said to be critical, when it has just been filled to capacity.

Lemma 2.13 Between every two times that an edge (u, v) is critical, $\delta_f(s, u)$ increases by at least 2.

Proof: When it is critical, $\delta_f(s, v) = \delta_f(s, u) + 1$. Then it disappears until some flow f'. When it reappears, then (v, u) is critical, so

$$\delta_{f'}(s, u) = \delta_{f'}(s, v) + 1 \geq \delta_f(s, v) + 1 = \delta_f(s, u) + 2.$$

Corollary 2.14 We have a polynomial algorithm.

Proof: Just bound the number of possible augmentations, noticing that each augmentation makes some edge critical.

Let $n = |V|$, $m = |E|$. Each edge becomes critical at most $n/2$ times. Therefore there are at most $m \cdot n/2$ augmentations. Each augmentation may take $O(m)$ steps: total bound is

$$O(m^2 n).$$

There are better algorithms: Goldberg's push-relabel algorithm achieves $O(n^3)$.

Goldberg algorithm

This algorithm works with pre-flows f with the property that

- No augmenting path exists for f.

It keeps adjusting the pre-flows repeatedly while preserving this property, until all excess in nodes other than t is eliminated.
Auxiliary integer labeling function called the height $h(v)$ of nodes.
The following requirements imply that there is no augmenting path: let n be the number of nodes.

Source and sink heights $h(s) \geq n$, $h(t) = 0$.

Steepness bound If an edge (u, v) is in the residual network G_f then $h(u) \leq h(v) + 1$.

The algorithm keeps adjusting both the pre-flow f and the height function h.

It works as follows.

Initialization $h(s) \leftarrow n$, and $h(v) \leftarrow 0$ for all $v \neq s$.
$f(e) \leftarrow c(e)$ for every edge leaving s, and $f(e) \leftarrow 0$ for all other e.

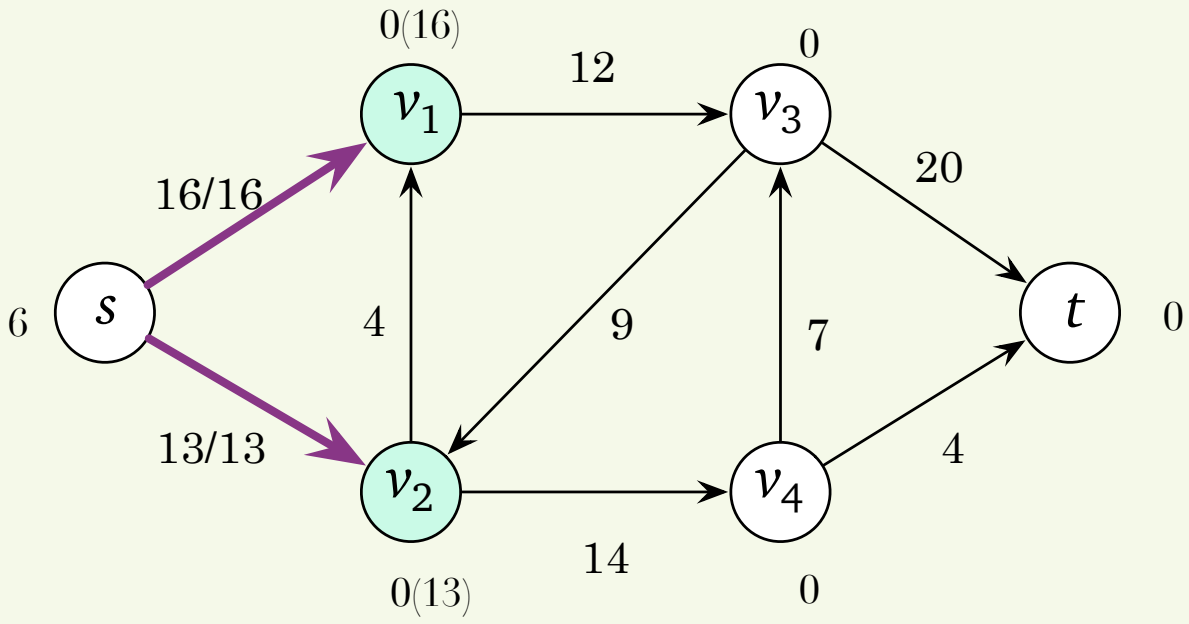
Pushing or relabeling Apply one of the following steps, whichever is applicable:
Choose a node u with $u \neq s, t$ and $e_f(u) > 0$. (For additional efficiency choose the one with the highest $h(u)$.)

if there is an edge (u, v) in the residual network G_f with $h(u) > h(v)$ **then**
push as much of the excess into $f(u, v)$ as $c(u, v)$ allows
else
$h(u) \leftarrow h(u) + 1$

These steps maintain the source and sink heights and the steepness bound.
We will prove that the algorithm terminates in $O(n^3)$ steps.

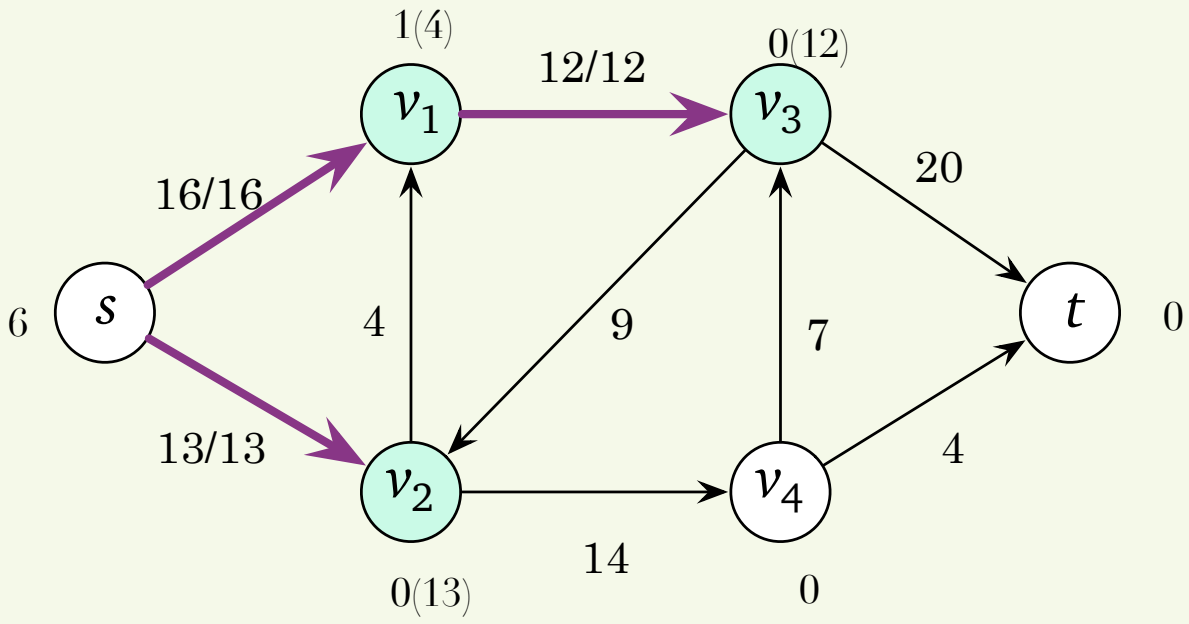
Push-relabel on the example

(Looks better in presentation mode.)



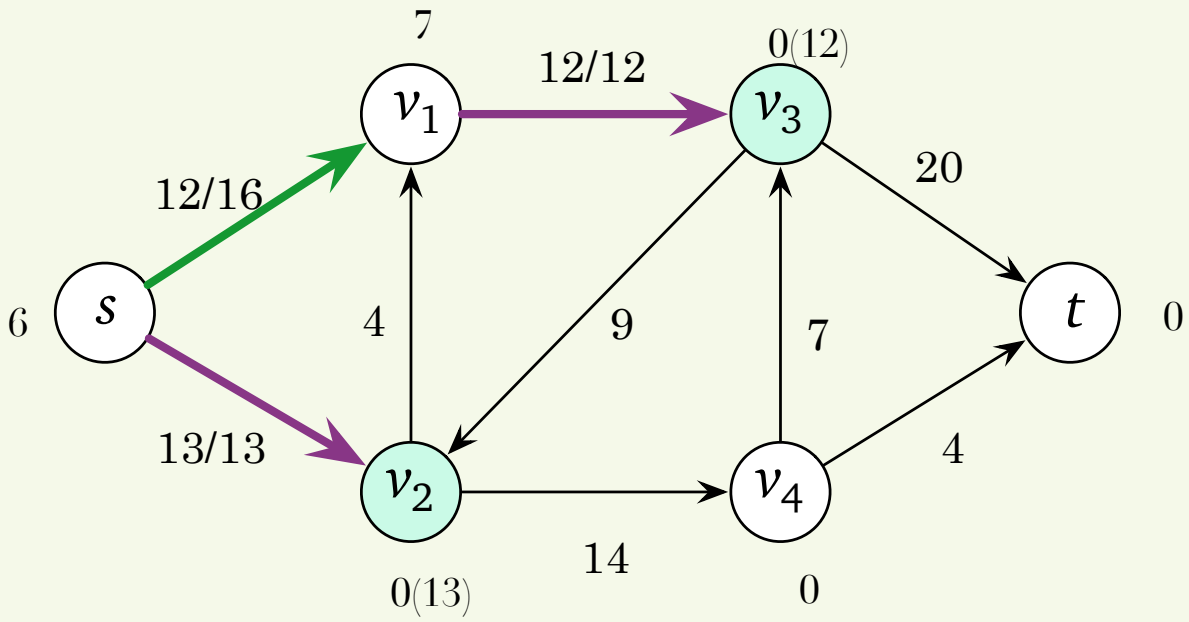
Push-relabel on the example

(Looks better in presentation mode.)



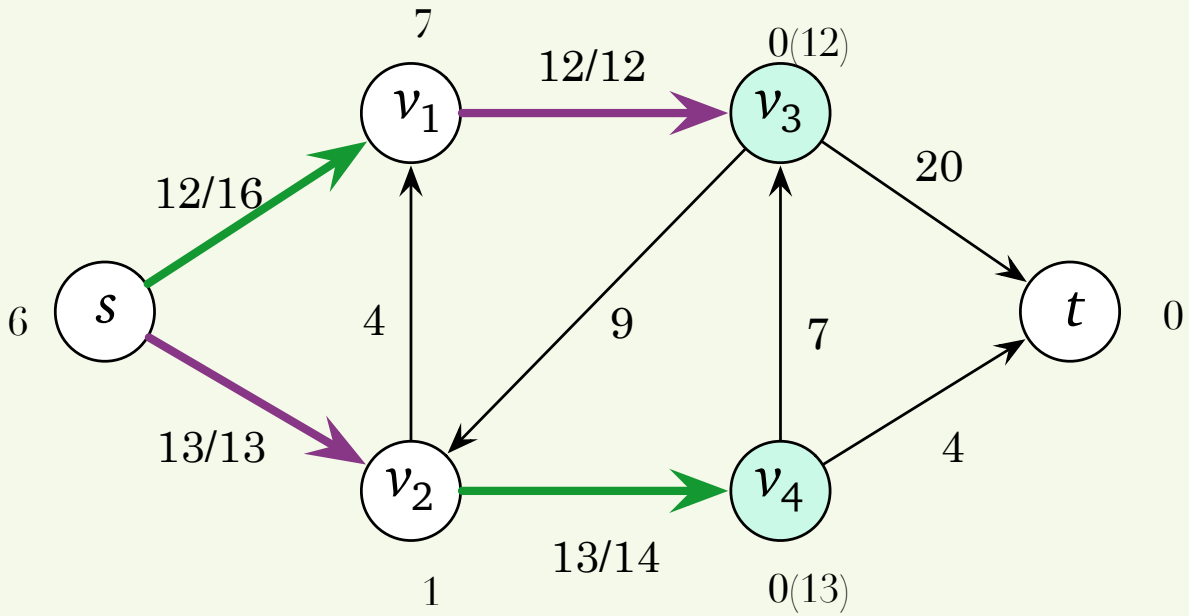
Push-relabel on the example

(Looks better in presentation mode.)



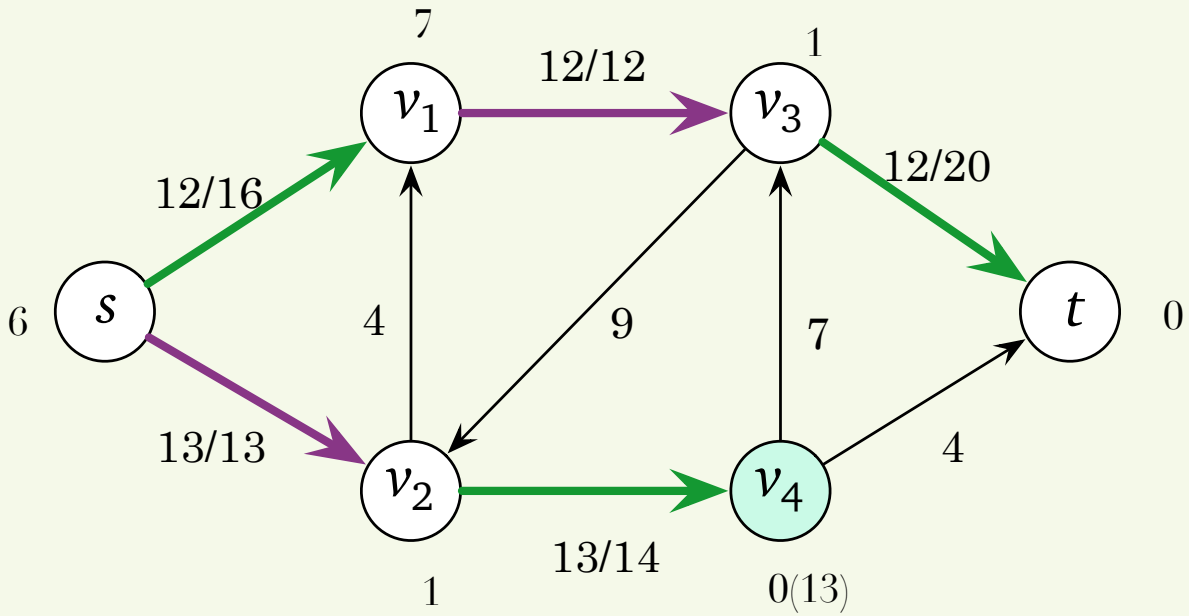
Push-relabel on the example

(Looks better in presentation mode.)



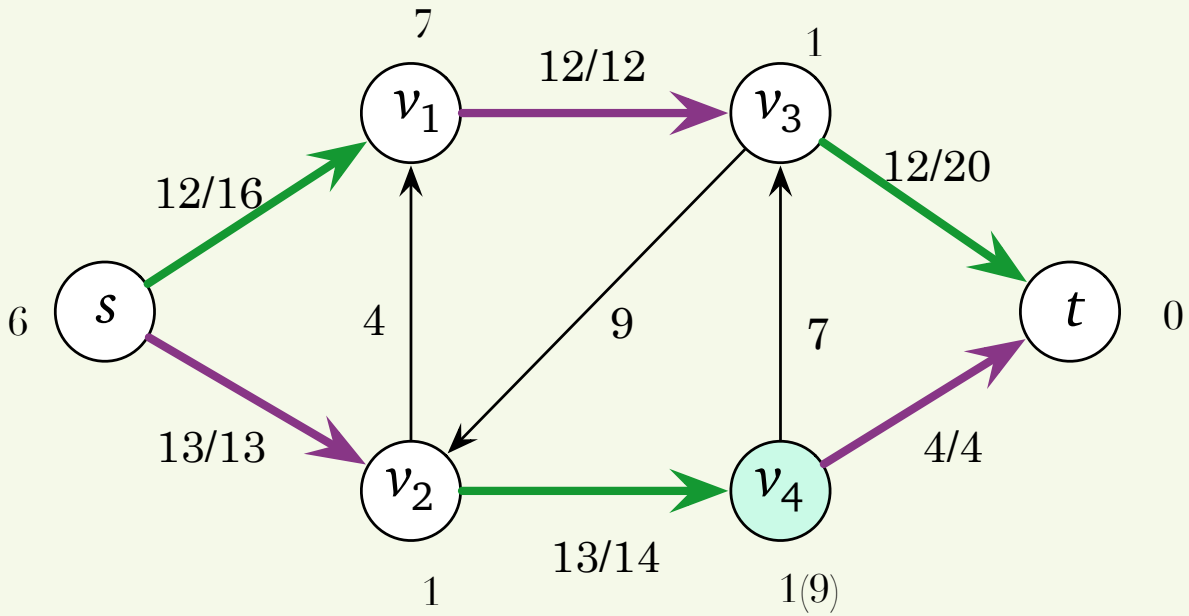
Push-relabel on the example

(Looks better in presentation mode.)



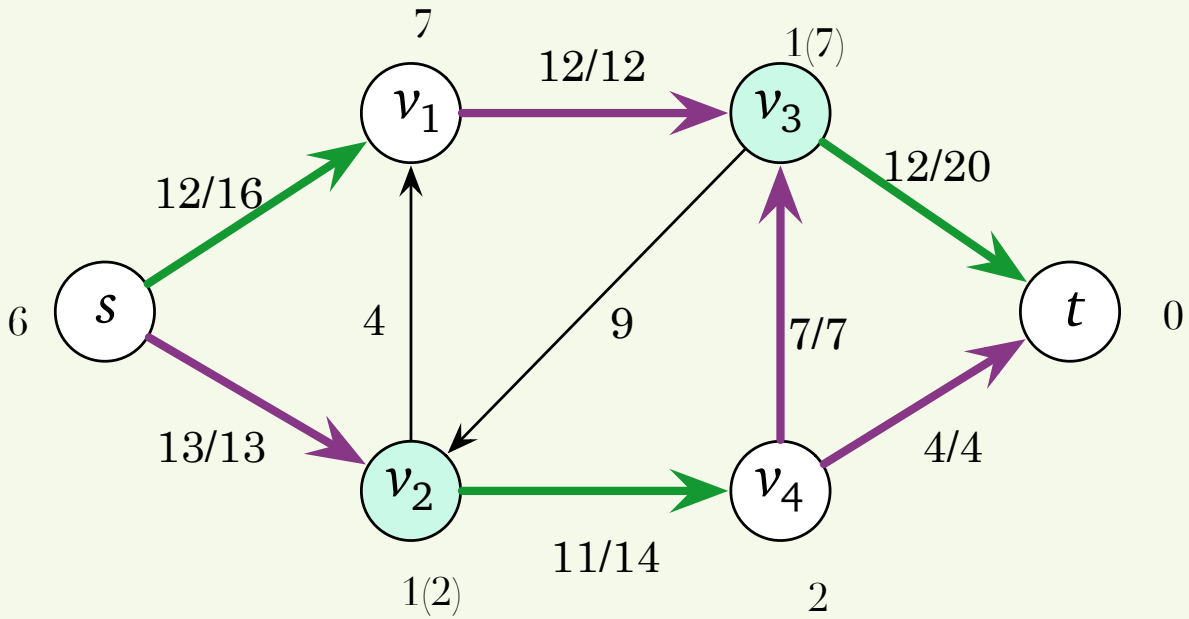
Push-relabel on the example

(Looks better in presentation mode.)



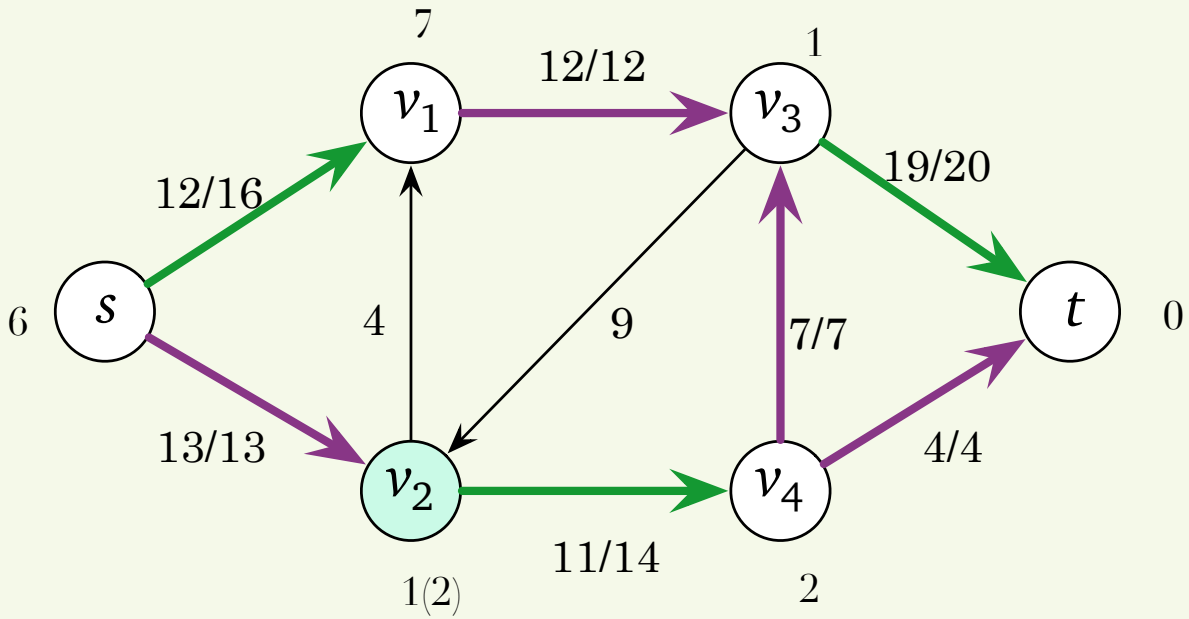
Push-relabel on the example

(Looks better in presentation mode.)



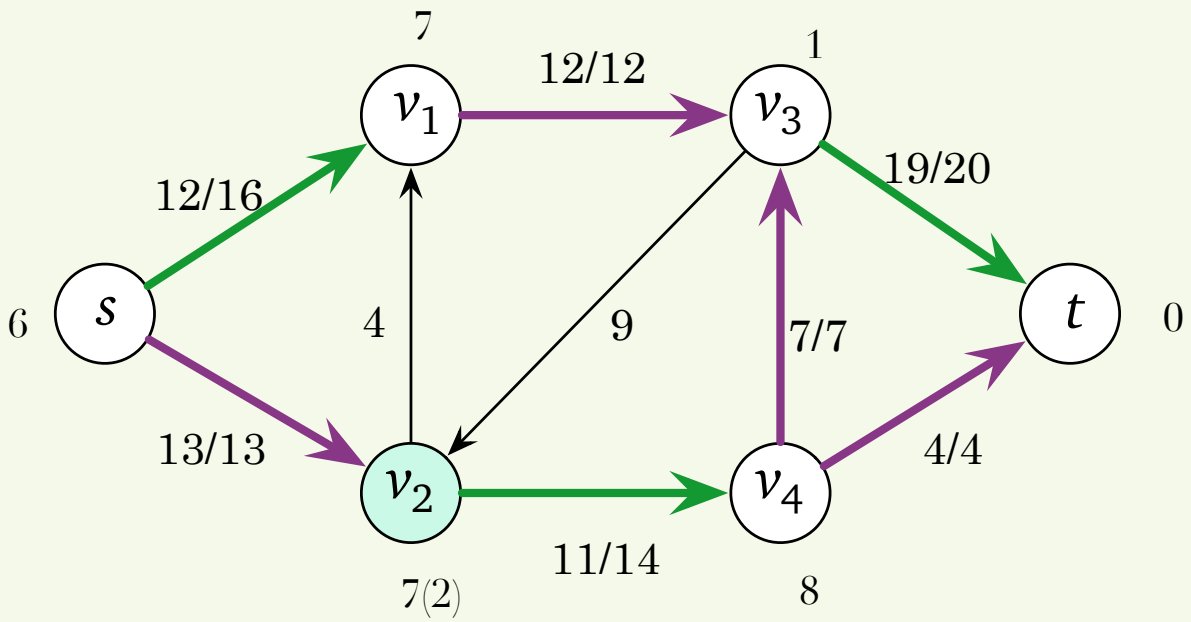
Push-relabel on the example

(Looks better in presentation mode.)



Push-relabel on the example

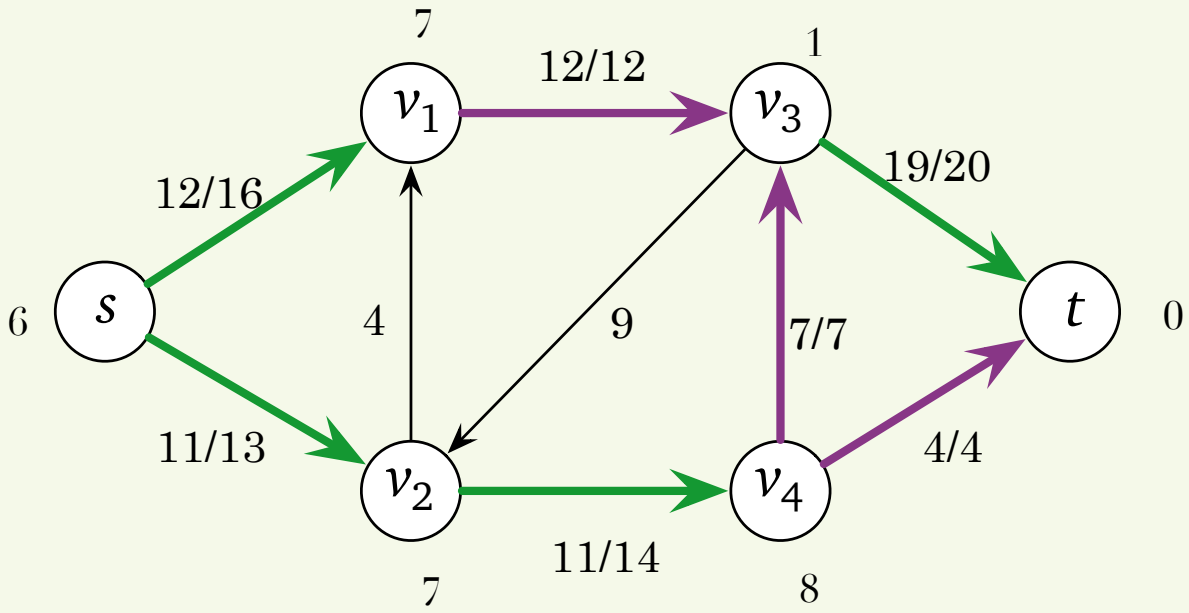
(Looks better in presentation mode.)



(After several back-and-forths between v_2 and v_4.)

Push-relabel on the example

(Looks better in presentation mode.)



Claim 2.15 If $e_f(u) > 0$ then there is an augmenting path from u to s.

Indeed, all the excess in u could only have come in flows from s. (A detailed accounting confirms this.)

- The claim and the steepness bound imply $h(u) \leq h(s) + n - 1$, hence $h(u) \leq 2n - 1$. Hence the number of relabeling operations is bounded by $(n-2)(2n-1) \leq 2n^2$.
- A push operation on edge (u, v) is saturating if it results in $f(u, v) = c(u, v)$.

Claim 2.16 On each edge (u, v) there are at most n saturating pushes. So the total number of saturating pushes is $\leq 2mn$.

Indeed, between each two saturating pushes, the height must increase by at least 2 (at the push in the opposite direction $h(v) > h(u)$). Now recall the bound $2n - 1$ on maximum height.

- We will bound by $4n^3$ the number of non-saturating pushes.

Claim 2.17

(a) At each value H of the maximum height of nodes with excess, from each node u of height H there is at most one non-saturating push.

(b) H changes at most $4n^2$ times.

To prove (a): a nonsaturating push eliminates the excess of u, and u can get a new excess only from a neighbor with height above $h(u)$.
To prove (b): H can also decrease; however, it can increase only by a relabel operation, of which there are $< 2n^2$.

- More sophisticated analysis shows a bound $O(n^2\sqrt{m})$ in place of $O(n^3)$.

Project selection

Network flow theory has many applications. Sometimes it takes ingenuity to apply it: see the following example.

- Set of possible projects to choose from: $P = \{1, 2, \ldots, n\}$.
  Project i brings profit p_i: positive or negative (then it is a cost).
- Dependencies: acyclic directed graph $G = (P, E)$.
  Edge (i, j): project i requires project j, too.
- Not a time ordering. Example:
  4: a wedding shower in which we could collect $p_4 = 1000$ dollars, but then we have to:
  - 5: buy food beforehand for $-p_5 = 200$ dollars, and
  - 8: clean up afterwards for $-p_8 = 250$ dollars.

Goal: A subset $S \subseteq P$ is feasible if with every project in it, it contains all others on which it depends: $u \in S$, $(u, v) \in E \Rightarrow v \in S$.

Find a feasible set S with maximum total profit $p(S) = \sum_{i \in S} p_i$.

Idea: Flow network. Add source and sink s, t. Capacities:

1. Edges $(i, j) \in E$ have capacity $c(i, j) = \infty$.
2. Edges (s, i) with $p_i > 0$ have capacity $c(s, i) = p_i$.
3. Edges (j, t) with $p_j < 0$ have capacity $c(j, t) = -p_j$.

A cut S, T has finite capacity if and only if $S' = S \setminus \{s\}$ is feasible.

If S is feasible:

$$c(S, T) = \sum_{i \in T : p_i > 0} p_i - \sum_{j \in S : p_j < 0} p_j. \tag{2.1}$$

Obvious upper bound on the total profit: $C = \sum_{i : p_i > 0} p_i$.

Claim 2.18 $p(S') = C - c(S, T)$.

Indeed: the first sum of (2.1) is the amount of profits we lose, and the second sum is the amount of the costs we incur.

- To maximize the profit, find a minimum cut.

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

Examples 3.1

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

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 3.2

- $\{(1, 2), (3, 6)\}$. 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 3.3 A set is a basis iff it is a minimal generating set.

Examples 3.4

- 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 3.5 All bases have the same number of elements.

Proof. Via the exchange lemma. □

Dimension of a vector space: this number.

Example 3.6 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 $\mathbf{x}$ in M in has a unique expression as

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

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

Example 3.7 If M is the set $\mathbb{R}^n$ of all n-tuples of real numbers then the n-tuples of form $\mathbf{e}_i = (0, \ldots, 1, \ldots, 0)$ (only position i has 1) form a basis. Then $(x_1, \ldots, x_n) = x_1\mathbf{e}_1 + \cdots + x_n\mathbf{e}_n$.

Example 3.8 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 $\text{diag}(a_{11}, \ldots, a_{nn})$
- Identity matrix.
- Triangular (unit triangular) matrices.
- Permutation matrix.
- Transpose $\mathbf{A}^T$. Symmetric matrix.

Matrix representing a linear map

A $p \times q$ matrix $\mathbf{A}$ 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 $\mathbf{x} = (x_i)$, $\mathbf{y} = (y_j)$ and matrix $\mathbf{A} = (a_{ij})$, this can be written as

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

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 $\mathbf{y} = \mathbf{Bz}$. Then it can be computed that

$$\mathbf{x} = \mathbf{Cz} \qquad\qquad \text{where } \mathbf{C} = (c_{ik}),$$

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

We define the matrix product

$$\mathbf{AB} = \mathbf{C}$$

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

$$\mathbf{x} = \mathbf{Ay} = \mathbf{A}(\mathbf{Bz}) = \mathbf{C}z = (\mathbf{AB})\mathbf{z}.$$

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

Example 3.9 For $\mathbf{A} = \left(\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right)$, $\mathbf{B} = \left(\begin{smallmatrix} 0 & 0 \\ 1 & 0 \end{smallmatrix}\right)$ we have $\mathbf{AB} \neq \mathbf{BA}$.

Transpose of product

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

Inner product

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

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

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

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

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

Example 3.10

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

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

Example 3.11 The matrix $\left(\begin{smallmatrix} 1 & 0 \\ 1 & 0 \end{smallmatrix}\right)$ is singular.

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

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

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

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

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

Theorem 3.12 If $\mathbf{A} : V \to W$ then

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

Theorem 3.13 A square matrix $\mathbf{A}$ is singular iff $\mathrm{Ker}\mathbf{A} \neq \{\mathbf{0}\}$.

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

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

Theorem 3.14 The two ranks are the same (in general, the dual of a vector space V has the same dimension as V). Also, $\mathrm{rank}(\mathbf{A})$ is the smallest r such that there is an $m \times r$ matrix $\mathbf{B}$ and an $r \times n$ matrix $\mathbf{C}$ with $\mathbf{A} = \mathbf{BC}$.

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 3.15 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 3.16 The outer product $\mathbf{A} = \mathbf{b}\mathbf{c}^T$ of two vectors has rank 1, and this product is the decomposition.

The following is immediate:

Proposition 3.17 A square matrix is nonsingular iff it has full rank.

- Minors.

Definition 3.18

- 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 3.19

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

$$\det(\mathbf{A}) = \sum_{\sigma} (-1)^{\mathrm{Inv}(\sigma)} a_{1\sigma(1)} a_{2\sigma(2)} \cdots a_{n\sigma(n)}. \tag{3.1}$$

Geometrical interpretation the absolute value of the determinant of a matrix $\mathbf{A}$ over $\mathbb{R}$ with column vectors $\mathbf{a_1}, \ldots, \mathbf{a_n}$ is the volume of the parallelepiped spanned by these vectors in n-space.

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

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

Computing $\det(\mathbf{A})$ using this formula is just as inefficient as using the original definition (3.1).

Properties

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

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

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

  $$\det(\mathbf{v}_2, \mathbf{v}_1, \ldots, \mathbf{v}_n) = -\det(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n).$$

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

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

Theorem 3.21 A square matrix is singular iff its determinant is 0.

The following is also known.

Theorem 3.22 $\det(\mathbf{AB}) = \det(\mathbf{A})\det(\mathbf{B})$.

Determinants for equations

Given an $n \times n$ square matrix $\mathbf{A}$ and an n-vector $\mathbf{b}$, let $\mathbf{A}[i]\mathbf{b}$ be the matrix obtained by replacing the ith column of $\mathbf{A}$ with $\mathbf{b}$. Then the following theorem of linear algebra will be used frequently:

Theorem 3.23 (Cramer's rule) If the matrix $\mathbf{A}$ is nonsingular then the solution of equation $\mathbf{Ax} = \mathbf{b}$ is a vector $\mathbf{x}$ whose coordinate x_i is

$$\frac{\det(\mathbf{A}[i]\mathbf{b})}{\det(A)}.$$

For example, if $\mathbf{A}$ is 2×2 matrix then

$$x_2 = \frac{a_{11}b_2 - a_{21}b_1}{a_{11}a_{22} - a_{21}a_{12}} = \frac{\begin{vmatrix} a_{11} & b_1 \\ a_{21} & b_2 \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}}.$$

We will use Cramer's rule to estimate the size of solutions. It is not helpful for actual computations: for computing the determinant, the efficient way is still to use Gaussian elimination, the same as for equations.

Positive definite matrices

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

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

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

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

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

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

Theorem 3.24 $\mathbf{A}$ is positive definite iff $\mathbf{A} = \mathbf{B}^T\mathbf{B}$ for some nonsingular $\mathbf{B}$.

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

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 4.1 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 $x_{k+1}, \ldots, x_n$.

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

where $a_{11}, \ldots, a_{kk} \neq 0$. Then $\dim \mathrm{Ker}(\mathbf{A}) = n-k$, $\dim \mathrm{Im}(\mathbf{A}) = k$.

- 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 $y_1, \ldots, y_m$:

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

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

This gives the equations

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

Concisely: $\mathbf{Ax} = \mathbf{b}$ is unsolvable if and only if $(\mathbf{y}^T\mathbf{A} = \mathbf{0}, \mathbf{y}_T\mathbf{b} = 1)$ 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.

Permutation matrix. **PA** interchanges the rows, **AP** the columns.

Example 4.2 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 **A**:

$$\mathbf{PA} = \mathbf{LU}$$

Using for equation solution:

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

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, ... of $\mathbf{A}$:

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

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

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

Repeating:

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

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

If

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

then setting

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

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

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

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

Positive (semi)definite matrix

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

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

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

Proposition 4.3 If $\mathbf{A}$ is positive (semi)definite then $\mathbf{A}_2 = \mathbf{A}' - \mathbf{v}\mathbf{v}^T/a_{11}$ is also.

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

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

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

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

Passing through a permutation

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

$$\mathbf{P}\mathbf{L}^{-1}\mathbf{A} = \mathbf{L}_3\mathbf{B}_4,$$

$$\mathbf{P}\mathbf{A} = \mathbf{P}\mathbf{L}\mathbf{P}^{-1}\mathbf{L}_3\mathbf{B}_4 = \hat{\mathbf{L}}\mathbf{L}_3\mathbf{B}_4 \text{ where}$$

$$\hat{\mathbf{L}} = \mathbf{P}\mathbf{L}\mathbf{P}^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & \ldots & 0 \\ \lambda_2 & 1 & 0 & 0 & \ldots & 0 \\ \lambda_{\pi(3)} & \mu_{\pi(3)} & 1 & 0 & \ldots & 0 \\ \lambda_{\pi(4)} & \mu_{\pi(4)} & 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \end{pmatrix},$$

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

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

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

  where the first k columns of $\mathbf{B}_{k+1}$ 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** $\pi[i] \leftarrow i$
$k, l \leftarrow 1$ // Pivot is (k, l).
while $k, l \leq n$ **do**
    $k' \leftarrow k, p \leftarrow 0$
    while $l \leq n$ **do**
        for $i = k$ **to** n **do**
            if $|a_{il}| > p$ **then**
                $p \leftarrow |a_{il}|, k' \leftarrow i$, **break**
        if $p > 0$ **then break**
        else $l \leftarrow l + 1$ // singular
    if $l \leq n$ **then** exchange $\pi[k] \leftrightarrow \pi[k']$ **else break**
    for $j = 1$ **to** n **do** exchange $a_{kj} \leftrightarrow a_{k'j}$
    for $i' = k + 1$ **to** n **do**
        $a_{i'l} \leftarrow a_{i'l}/a_{kl}$
        for $j' = l + 1$ **to** n **do** $a_{i'j'} \leftarrow a_{i'j'} - a_{i'l}a_{kj'}$
    $k \leftarrow k + 1, l \leftarrow l + 1$

Proposition 4.4 For an $n \times n$ matrix $\mathbf{A}$, the row rank is the same as the column rank.

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

Inverting matrices

- Computing matrix inverse from an LUP decomposition: solving equations

$$\mathbf{A}\mathbf{X}_i = \mathbf{e}_i, \qquad i = 1, \ldots, n.$$

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

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

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

Theorem 5.1 Multiplication is no harder than inversion.

Proof. Let

$$\mathbf{D} = \mathbf{L}_1\mathbf{L}_2 = \begin{pmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{A} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} & \mathbf{I} \end{pmatrix} = \begin{pmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{A} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{B} & \mathbf{I} \end{pmatrix}.$$

Its inverse is

$$\mathbf{D}^{-1} = \mathbf{L}_2^{-1}\mathbf{L}_1^{-1} = \begin{pmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & -\mathbf{B} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ -\mathbf{A} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{pmatrix} = \begin{pmatrix} \mathbf{I} & \mathbf{0} & \mathbf{0} \\ -\mathbf{A} & \mathbf{I} & \mathbf{0} \\ \mathbf{AB} & -\mathbf{B} & \mathbf{I} \end{pmatrix}.$$

□

Theorem 5.2 Inversion is no harder than multiplication.

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

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

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

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

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

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

4 multiplications of size $n/2$ matrices

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

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

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

Assume $F(n) \leq c_2 n^b$ with $b > 1$. Then

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

So,

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

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

Least squares approximation (reading)

Data: $(x_1, y_1), \ldots, (x_m, y_m)$.
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).

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

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

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

$$(\mathbf{Ac} - \mathbf{y})^T \mathbf{Ax} = 0 \qquad \text{for all } \mathbf{x}, \text{ so}$$
$$(\mathbf{Ac} - \mathbf{y})^T \mathbf{A} = 0$$
$$\mathbf{A}^T(\mathbf{Ac} - \mathbf{y}) = 0$$

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

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.

Solving linear equations 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 $\mathbf{A}$ has rows $\mathbf{a}_1^T, \ldots, \mathbf{a}_n^T$ then

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

  This is known as Hadamard's inequality.
- So if the inputs are integers then the length of the result in bits is polynomial in the length of the inputs.

The paradox of the determinant

- $\det(\mathbf{A})$ is a polynomial of the elements a_{ij}. But it has $n!$ terms, and this is the number of operations needed in the usual formulas for computing it.
- As seen, the length of the output does not require so many operations.
- Computing $\det(\mathbf{A})$ by Gaussian elimination uses only $O(n^3)$ operations, but these involve divisions.
- Divisions create a dilemma.
  - Rounding introduces errors that may accumulate.
  - Computing exactly with fractions might increase the bit length of the intermediate fractions a/b to exponential size.
- Below we will show that Gaussian elimination is special: it is possible to keep the bit length of the intermediate fractions to polynomial size.

Theorem 7.1 Assume that Gaussian elimination on an integer matrix $\mathbf{A}$ 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

- $\mathbf{A}^{(k)}$ = be the matrix after k stages of the elimination.
- $\mathbf{D}^{(k)}$ = the minor determined by the first k rows and columns of $\mathbf{A}$.
- $\mathbf{D}^{(k)}_{ij}$ =, for $k+1 \le i, j \le n$, 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 $a^{(k)}_{ij} = \dfrac{\det \mathbf{D}^{(k)}_{ij}}{\det \mathbf{D}^{(k)}}$.

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

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

Divide these two equations by each other. □

- A similar argument leads to the theorem called Cramer's Rule, announced earlier.

- The theorem shows that if we always reduce fractions (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. See for example `exact-Gauss.pdf`.

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 7.2

$$\begin{aligned} 0.0001x + \quad y &= 1 \\ 0.5x + 0.5y &= 1 \end{aligned} \tag{7.1}$$

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 7.3

$$\begin{aligned} x + 10,000y &= 10,000 \\ 0.5x + \quad 0.5y &= \quad 1 \end{aligned} \tag{7.2}$$

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

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

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

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

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

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

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

Example 7.4 In (7.2), let $r_1 = 10^{-4}$, all other coeffs are 1: We get back (7.1), which we solve by partial pivoting as before.

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

Example 7.5 Choose $s_2 = 10^{-4}$, all other coeffs 1:

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

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

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

so, we again got the bad solution.

Fortunately, such pathological systems are rare in practice.

How about solving a system of linear inequalities?

$$\mathbf{Ax} \leq \mathbf{b}.$$

We will try to solve a seemingly more general problem:

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{Ax} \leq \mathbf{b}. \end{aligned}$$

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 8.1 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 $ 100 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 $(x_1 + \cdots + x_4) \cdot 100$.

The linear program:

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

Implicit inequalities: $x_i \geq 0$.

- Solutions form a convex polyhedron (intersection of half-spaces).
- 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).

Two-dimensional example

$$
\begin{array}{lrcl}
\text{maximize} & x_1 + & x_2 & \\
\text{subject to} & 4x_1 - & x_2 \leq & 8 \\
& 2x_1 + & x_2 \leq & 10 \\
& 5x_1 - 2x_2 & \geq & -2 \\
& x_1, & x_2 \geq & 0
\end{array}
$$

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

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

has 2^n extremal points.

Formulating problems as linear programs

Maximum error minimization

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

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

Linear programming can solve this:

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

(Maximization is counter-intuitive, but correct.)

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

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

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

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.

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

Example 8.2 $m = n = 2$, pure strategies $\{1, 2\}$ are called "attack left", "attack right" for player 1 and "defend left", "defend right" for player 2. The matrix is

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

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

Both players may improve their achievable values by randomization. Mixed strategy: a probability distribution over pure strategies. $\mathbf{p} = (p_1, \ldots, p_m)$ for player 1 and $\mathbf{q} = (q_1, \ldots, q_m)$ 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 $\mathbf{p}$, $\mathbf{q}$. The big result will be that now $\max_{\mathbf{p}} \min_{\mathbf{q}} = \min_{\mathbf{q}} \max_{\mathbf{p}}$.
Translation into linear programming: If player 1 knows the mixed strategy $\mathbf{q}$ of player 2, he will want to achieve

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

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

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

Rewritten as a linear program:

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

Standard form

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^T\mathbf{x} \\ \text{subject to} \quad & \mathbf{A}\mathbf{x} \le \mathbf{b} \\ & \mathbf{x} \ge \mathbf{0} \end{aligned}$$

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

$$x_j = x_j' - x_j'', \text{ subject to } x_j', x_j'' \ge 0.$$

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
$(N, B, \mathbf{A}, \mathbf{b}, \mathbf{c}, v)$:

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

Note that these equations are always independent.

Lemma 8.3 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.

- A basic solution: set each nonbasic variable to 0.
- Assume that there is a basic feasible solution, that is where all b_i are positive.
  See later how to find one.

Example:

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

- 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.

Rewrite all other equations, substituting this x_1:

$$\begin{aligned} z &= 27 + \tfrac{1}{4}x_2 + \tfrac{1}{2}x_3 - \tfrac{3}{4}x_6 \\ x_1 &= 9 - \tfrac{1}{4}x_2 - \tfrac{1}{2}x_3 - \tfrac{1}{4}x_6 \\ x_4 &= 21 - \tfrac{3}{4}x_2 - \tfrac{5}{2}x_3 + \tfrac{1}{4}x_6 \\ x_5 &= 6 - \tfrac{3}{2}x_2 - 4x_3 + \tfrac{1}{2}x_6 \end{aligned}$$

Formal pivot algorithm: no surprise.

Cases with no increase by pivoting

Let $z = v + \sum_{j \in N} c_j x_j$ at some intermediate stage. Cases:

1. All c_j are ≤ 0: then we are at the optimum.
2. There is a $c_j > 0$ with $a_{ij} > 0$ for all i: then the optimum is ∞, this is the unbounded case.
3. There are j with $c_j > 0$, but for each such j, there are i with $a_{ij} < 0$, and for such i we have $b_i = 0$. Pivoting does not increase the objective function: danger of cycling.

The problem of cycling: Can be solved, though you will not encounter it in practice.

- Bland's Rule: choose entering variable with the smallest, then leaving variable with smallest index. (See handout for proof of termination.)
- Geometry: several minimal sets of constraints define the same vertex—must find one from which we can leave on an edge and increase the objective.

Initial basic feasible solution

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

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

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

Slack form:

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

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 (which is negative). Assuming b_1 is this:

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

The basic solution of this system is feasible. Carry out the simplex method starting from it. Eventually (if the optimum is $x_0 = 0$), bring out x_0 from the basis again (any way you want, will not change the solution as $b_0 = 0$ now). Now you have got a basic feasible solution of the original program.

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.

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

$$\begin{array}{rlrl} & \mathbf{A}^T\mathbf{y} \geq \mathbf{c} & & \mathbf{y}^T\mathbf{A} \geq \mathbf{c}^T \\ & \mathbf{y} \geq \mathbf{0} & & \mathbf{y}^T \geq \mathbf{0} \\ \min & \mathbf{b}^T\mathbf{y} & \min & \mathbf{y}^T\mathbf{b} \end{array}$$

Value of the optimum if feasible: t^*.

Proposition 8.4 (Weak duality) $z^* \leq t^*$, moreover for every pair of feasible solutions $\mathbf{x}$, $\mathbf{y}$ of the primal and dual:

$$\mathbf{c}^T\mathbf{x} \leq \mathbf{y}^T\mathbf{A}\mathbf{x} \leq \mathbf{y}^T\mathbf{b} = \mathbf{b}^T\mathbf{y}. \tag{8.1}$$

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).

$$\begin{aligned} a_{11}x_1 + \ldots + a_{1n}x_n &\le b_1 \\ &\ldots \\ a_{m1}x_1 + \ldots + a_{mn}x_n &\le b_m \\ -x_1 &\le 0 \\ &\ldots \\ -x_n &\le 0 \end{aligned} \tag{8.2}$$

If a nonnegative linear combination of (8.2) gives

$$c_1x_1 + \ldots + c_nx_n \le \mu. \tag{8.3}$$

then the optimum is $\le \mu$. Let $y_1, \ldots, y_m, z_1, \ldots, z_n$ be the coefficients of such a linear combination, then

$$\begin{aligned} y_1a_{1j} + \ldots + y_ma_{mj} + z_j &= c_j, \quad j = 1, \ldots, m, \\ y_1b_1 + \ldots + y_mb_m &= \mu, \end{aligned}$$

that is $\mathbf{y}^T\mathbf{A} \ge \mathbf{c}^T$, $\mathbf{y}^T\mathbf{b} = \mu$. The dual asks for the smallest μ.

Economical 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.

$\mathbf{Ax} \leq \mathbf{b}$ says that you can use only the resources you have.
Primal problem: maximize the income $\mathbf{c}^T\mathbf{x}$ achievable with the given resources.

Suppose that resources i have prices $y_i \geq 0$ with the property that

$$\sum_i y_i a_{ij} \geq c_j, \ j = 1, \ldots, n.$$

Then selling the resources for product j you get at least as much as the income c_j from selling the product j: so we can call these discouraging prices. For such a set of prices, the total income $\sum_i b_i y_i$ from selling all your resouces upperbounds any possible income $\sum_j c_j x_j$ from production.
Dual problem: Minimize $\sum_i b_i y_i$ obtainable from discouraging prices.
The optimal discouraging prices are called shadow prices.

You can skip this slide

Suppose that you can buy lacking resources and sell unused resources. Total income:

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

Let

$$f(\hat{\mathbf{x}}) = \inf_{\mathbf{y}\geq\mathbf{0}} L(\hat{\mathbf{x}}, \mathbf{y}) \leq L(\hat{\mathbf{x}}, \hat{\mathbf{y}}) \leq \sup_{\mathbf{x}\geq\mathbf{0}} L(\mathbf{x}, \hat{\mathbf{y}}) = g(\hat{\mathbf{y}}).$$

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

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

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

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

Complementary slackness conditions:

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

Proposition 8.5 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 8.6 (Strong duality) The primal problem has an optimum if and only if the dual is feasible, and we have

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

This surprising theorem says that at the shadow prices, the total value of your resources is equal to the income from optimal production.

Many interesting uses and interpretations, and many proofs. The handout gives a geometric proof not dependent on the simplex algorithm. But the simplex algorithm gives an explicit solution to the dual.

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

Theorem 8.7 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 $\tilde{c}_i \leq 0$ such that

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

is an identity for the variable $\mathbf{x}$, 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 $\mathbf{x}$, 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 $\mathbf{x}$, 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 $\tilde{c}_j \leq 0$, 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

$$\begin{aligned} \mathbf{Ax} &\leq \mathbf{b} \\ \mathbf{A}^T\mathbf{y} &\geq \mathbf{c} \\ \mathbf{c}^T\mathbf{x} - \mathbf{b}^T\mathbf{y} &= 0 \\ \mathbf{x}, \quad \mathbf{y} &\geq \mathbf{0} \end{aligned}$$

gives an optimal solution to the original linear programming problem.

Theorem 8.8 (Farkas Lemma, not as in the book) A set of inequalities $\mathbf{Ax} \leq \mathbf{b}$ is unsolvable if and only if a positive linear combination gives a contradiction: there is a solution $\mathbf{y} \geq \mathbf{0}$ to the inequalities

$$\begin{aligned} \mathbf{y}^T\mathbf{A} &= 0, \\ \mathbf{y}^T\mathbf{b} &= -1. \end{aligned}$$

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

We use the homework allowing variables without nonnegativity constraints:

$$\begin{array}{ll} \text{maximize} & z \\ \text{subject to} & \mathbf{Ax} + z \cdot \mathbf{e} \leq \mathbf{b} \end{array} \tag{8.4}$$

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

$$\begin{array}{ll} \text{minimize} & \mathbf{y}^T\mathbf{b} \\ \text{subject to} & \mathbf{y}^T\mathbf{A} = \mathbf{0} \\ & \mathbf{y}^T\mathbf{e} = 1 \\ & \mathbf{y}^T \geq \mathbf{0} \end{array} \tag{8.5}$$

The original problem has no feasible solution if and only if $\max z < 0$ in (8.4). In this case, $\min \mathbf{y}^T\mathbf{b} < 0$ in (8.5). Condition $\mathbf{y}^T\mathbf{e} = 1$ is not needed. If we drop it then we can scale $\mathbf{y}$ to have $\mathbf{y}^T\mathbf{b} = -1$.

Separating hyperplane

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

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

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

$$\mathbf{y}^T \mathbf{U} = \mathbf{v}^T, \quad \sum_i y_i = 1, \quad \mathbf{y} \geq 0. \tag{8.6}$$

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

$$\mathbf{u}_i^T \mathbf{x} \leq z \quad (i = 1, \ldots, m), \quad \mathbf{v}^T \mathbf{x} > z.$$

Can be derived from the Farkas Lemma.

Complementary slackness, geometrically

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

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

Then for all those constraints $\mathbf{d}^T\mathbf{u}_i \leq c$ 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 $\mathbf{d}^T\mathbf{v} = c$.

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

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

Dual:

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

Dual for max-flow: min-cut

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

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 $-f(u, v)$.

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

Some constraints disappeared but others appeared, since in case of $u < v$ the constraint $f(v,u) \le c(v,u)$ is written now $-f(u,v) \le c(u,v)$. A dual variable for each constraint. For $f(u,v) \le c(u,v)$, call it $y(u,v)$, for $-f(u,v) \le c(v,u)$, 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 $y(v) \geq y(u)$ then making $y(v, u) = 0$ still leaves $y(u, v) \geq 0$. 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 8.9 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 $\mathbf{y}$ 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.

- 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

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

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.

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

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

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

$$E = \{\, \mathbf{Lx} : \mathbf{x}^T\mathbf{x} \leq 1 \,\} = \{\, \mathbf{y} : \mathbf{y}^T\mathbf{A}^{-1}\mathbf{y} \leq 1 \,\},$$

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

$$\{\, \mathbf{x} : (\mathbf{x} - \mathbf{c})^T\mathbf{A}^{-1}(\mathbf{x} - \mathbf{c}) \leq 1 \,\}$$

where $\mathbf{A}$ 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 $\mathbf{b}_1, \ldots, \mathbf{b}_n$ of the vector space $\mathbb{R}^n$ is called orthonormal if $\mathbf{b}_i^T \mathbf{b}_j = 0$ for $i \neq j$ and 1 for $i = j$.

Theorem 9.1 (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

$$E = \{ \mathbf{x} : \mathbf{x}^T \mathbf{A}^{-2} \mathbf{x} \leq 1 \},$$

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

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

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

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

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

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 9.2

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

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

Lemma 9.3 If there is a solution that is a vertex then there is one with $|x_j| \leq N$ for all j.

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

Lemma 9.4 If $\mathbf{Ax} \leq \mathbf{b}$ has no solution then defining $b'_i = b_i + \delta$, the system $\mathbf{Ax} \leq \mathbf{b}'$ has no solution either.

The following clearly implies (b) of the theorem:

Corollary 9.5 If $\mathbf{Ax} \leq \mathbf{b}'$ is solvable then its set of solutions contains a cube of size 2ε.

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

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

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

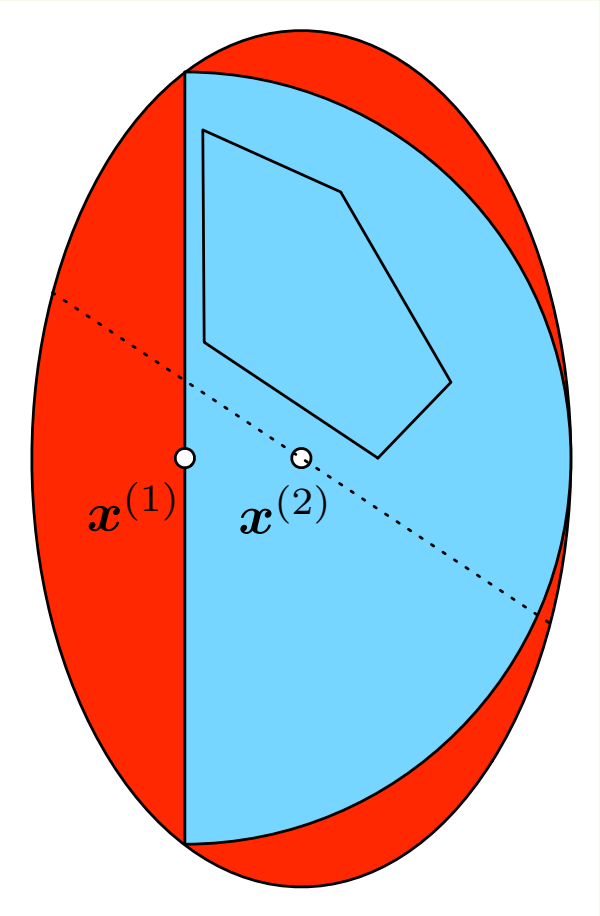
The algorithm

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

$$H_1 = E_1 \cap \{ \mathbf{x} : \mathbf{a}_i^T \mathbf{x} \leq \mathbf{a}_i^T \mathbf{x}^{(1)} \}.$$

Shrinking rate

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



Lemma 9.6 There is an ellipsoid E_2 containing H_1 with $\mathrm{Vol}(E_2) \leq e^{-\frac{1}{2(n+1)}} \mathrm{Vol}(E_1)$. This is true even if E_1 was also an ellipsoid.

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

Assume without loss of generality

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

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

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

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

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

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 $1 + z \leq e^z$:

$$\mathrm{Vol}(E_2) = V_n(1-d)b^{n-1} \leq V_n e^{-d+\frac{(n-1)d^2}{2(1-2d)}} = V_n e^{-d\frac{2-(n+3)d}{2(1-2d)}}.$$

Choose $d = \frac{1}{n+1}$ to make the fraction $1/2$, then this is $V_n e^{-\frac{1}{2(n+1)}}$. 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

$$e^{-\frac{r}{2(n+1)}}.$$

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

$$e^{-\frac{r}{2(n+1)}} < \left(\frac{\varepsilon}{N\sqrt{n}}\right)^n$$

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

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

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

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 $x_j \geq 0$, $x_j \leq 0$). Then solve the equations.

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

Equivalently, f is convex if

$$f(\lambda \mathbf{a} + (1 - \lambda)\mathbf{b}) \leq \lambda f(\mathbf{a}) + (1 - \lambda) f(\mathbf{b})$$

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

Examples 10.2

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

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

Definition 10.3 A convex program is an optimization problem of the form

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

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

$$\mathbf{x} \in H$$

for any convex set H given in some effective way.

Example: Support vector machine

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

$$\mathbf{z}^T \mathbf{u}_i \leq x \text{ for } i = 1, \ldots, k,$$
$$\mathbf{z}^T \mathbf{v}_i \geq y \text{ for } i = 1, \ldots, l.$$

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

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

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

$$\mathbf{a}_i^T \mathbf{x} \leq b_i, \quad i = 1, \ldots, n.$$

  A trial solution $\mathbf{x}^{(t)}$ was always the center of some ellipsoid E_t. If it violated the conditions, it violated one of these: $\mathbf{a}_i^T \mathbf{x}^{(t)} > b_i$. 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 $\mathbf{x}^{(t)} \in H$. How to imitate the ellipsoid algorithm further?

Definition 10.4 Let $\mathbf{a} : \mathbb{Q}^n \to \mathbb{Q}^n$, $b : \mathbb{Q}^n \to \mathbb{Q}$ 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 $\mathbf{x} \in \mathbb{R}^n$, with $\mathbf{a} = \mathbf{a}(\mathbf{x})$, $b = b(\mathbf{x})$ we have:

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

Example 10.5 For the unit ball $H = \{\, \mathbf{x} : \mathbf{x}^T\mathbf{x} \leq 1 \,\}$, the functions $\mathbf{a} = \mathbf{x} \cdot |\mathbf{x}^T\mathbf{x} - 1|^+$, and $b = \mathbf{x}^T\mathbf{x}|\mathbf{x}^T\mathbf{x} - 1|^+$ 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 $(\mathbf{a}(\cdot), b(\cdot))$. 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.

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

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

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

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

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

$$S = \{ i : \mathbf{z}^T \mathbf{u}_i \leq 0 \}.$$

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

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

(draw a picture!).

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

$$\arccos(y)/\pi \geq \alpha(1-y)/2, \quad -1 \leq y \leq 1.$$

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

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

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

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

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

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

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

Separation oracle for semidefiniteness

Please look up the the LU decomposition algorithm in these notes, when applied to positive semidefinite matrices $\mathbf{A}$ (Cholesky decomposition). We structured it in such a way that when it fails it gives a witness $\mathbf{z}$ 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 $\mathbf{A}$ from the positive semidefinite ones. Indeed, for any positive semidefinite matrix $\mathbf{B}$ we have $\sum_{ij} z_i z_j b_{ij} \geq 0$.

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.

Near-optimal set cover and duality

- Given set $X = \{1, \ldots, m\}$ and a family $\mathcal{F} = \{S_1, \ldots, S_n\}$ 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_j has cost $c_j > 0$. We want a minimum-cost set cover.
- Special case: vertex cover of a graph $G = (V, E)$. $X = E$, and vertex v_j has cost c_j.

Linear programming relaxation

- Let $a_{ij} = 1$ if $i \in S_j$, and 0 otherwise.
- Let $x_j = 1$ mean that set S_j is chosen.
- Condition that i is covered: $\sum_j a_{ij} x_j \geq b_i$ where $b_i = 1$.

$$\begin{aligned} \text{minimize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i, \;\; i = 1, \ldots, m, \\ & x_j \geq 0, \;\; j = 1, \ldots, n. \end{aligned}$$

Self-test: why don't we need to require $x_j \leq 1 (= b_i)$?
This is a relaxation of the original problem (we relax the condition that x_j is integer). Any feasible solution is called a fractional set cover.

The dual: prices for elements

$$\begin{aligned} \text{maximize} \quad & \textstyle\sum_{i=1}^{n} b_i y_i \\ \text{subject to} \quad & \textstyle\sum_{i=1}^{n} a_{ij} y_i \le c_j, \ j = 1, \ldots, n, \\ & y_i \ge 0, \ i = 1, \ldots, m. \end{aligned}$$

Self-test: why don't we need to require $y_i \le c_j$?

- View y_i as the price of element i.
- Constraints: the cost of each set S_j is $\ge$ the total price of its elements.
- Goal: maximize the total price (recall $b_i = 1$).

Set-packing interpretation of the dual (can be skipped)

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{n} b_i y_i \\ \text{subject to} \quad & \sum_{i=1}^{n} a_{ij} y_i \le c_j, \ j = 1, \ldots, n, \\ & y_i \ge 0, \ i = 1, \ldots, m. \end{aligned}$$

- Let $T_i = \{ j : a_{ij} = 1 \}$. View b_i as the worth of set T_i.
- Consider the special case $c_j = 1$ for all j.
- If y_i must be integer then it shows whether set T_i is chosen.
- The constraints say that the chosen sets T_i must be disjoint: they form a packing.
- Goal: to maximize the total worth of the packing.

Feasible solutions are called a fractional packing.

Greedy optimization of the dual

Constraint $\sum_{i=1}^{n} a_{ij} y_i \le c_j$ belonging to set S_j is tight if it is satisfied with equality.

Algorithm: Repeat while you can: pick a variable y_j and increase it until one of the constraints becomes tight.

The set cover: those sets whose constraints became tight.

How good? Multiplicity $m(i) = \sum_j a_{ij}$ of an element i: the number of sets S_j it belongs to. Let $M = \max_i m(i)$. By weak duality and the tightness of sets with $x_j = 1$:

$$\sum_i y_i \le \sum_j x_j c_j = \sum_j x_j \sum_i a_{ij} y_i = \sum_i y_i \sum_j a_{ij} x_j$$
$$\le \sum_i y_i m(i) \le M \sum_i y_i .$$

So this solution is within factor M from the optimum.

Vertex cover: $M = 2$, we got a 2-optimal solution.

Unit-cost vertex cover

- The above algorithm for the unit-cost vertex cover ($c_j = 1$) is particularly simple:
  Repeat while you can: pick an uncovered edge and add both ends to the vertex cover.
- This algorithm is greedy for the dual problem (increasing edge prices), but not greedy for the primal (choosing vertices).

Exercise What is the greedy algorithm for the primal?

Primal-dual schema (can be skipped)

The idea of the above algorithm can be generalized.
For some $\alpha, \beta \geq 1$, formally relax the complementary slackness:
Primal conditions: $x_j > 0 \Rightarrow c_j/\alpha \leq \sum_i a_{ij} y_i \leq c_j$.
Dual conditions: $y_i > 0 \Rightarrow b_i \leq \sum_j a_{ij} x_j \leq \beta b_i$.

Proposition 11.1 If the primal and dual feasible solutions satisfy these conditions, then

$$\mathbf{c}^T \mathbf{x} \leq \alpha \beta \mathbf{b}^T \mathbf{y}.$$

Proof straightforward:

$$\sum_j c_j x_j \leq \alpha \sum_j x_j \sum_i a_{ij} y_i = \alpha \sum_i y_i \sum_j a_{ij} x_j \leq \alpha \beta \sum_i y_i b_i.$$

The primal-dual schema:

- Start from an infeasible integer primal and a feasible dual (typically $\mathbf{x} = \mathbf{0}$, $\mathbf{y} = \mathbf{0}$).
- 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.

We carried out this program for set cover.

Primal complementary slackness conditions for each S, with factor $\alpha = 1$:

$$x_S > 0 \Rightarrow \sum_{e \in S} y_e = c_S .$$

Set S is tight when this holds. Plan: use only tight sets.

Dual complementary slackness conditions: worst possible factor, the maximal number β of sets with a nonempty intersection. Note that $x_S \leq 1$ will always hold for the optimum:

$$y_e > 0 \Rightarrow 1 \leq \sum_{S \ni e} x_S \leq \beta .$$

Greedy set cover

For set $S = S_j$ write $c_S = c_j$, $x_S = x_j$.

Algorithm Greedy-Set-Cover(X, $\mathcal{F}$)

$U \leftarrow X$

$\mathcal{C} \leftarrow \emptyset$ (same as saying $x_S = 0$ for all S)

For the dual variables, initially let $y_e = 0$ for all e

while $U \neq \emptyset$ **do**

select an cF that maximizes $|S \cap U|/c_S$

$U \leftarrow U \setminus S$

$\mathcal{C} \leftarrow \mathcal{C} \cup \{S\}$ (same as saying $x_S \leftarrow 1$)

If element e was covered by set S then let $y_e = \frac{c(S)}{|S \cap U|}$ be its increased price.

return $\mathcal{C}$

This way we cover each element at minimum price (at the moment).

The total cost is $\sum_S c_S x_S = \sum_e y_e$.

The approximation ratio would be 1 (optimal solution) if the prices y_e defined above gave a dual feasible solution. But they may not be. Let $H(n) = 1 + 1/2 + \cdots + 1/n (\approx \ln n)$.

Theorem 11.2 Greedy-Set-Cover has a ratio bound $\max_{S \in \mathcal{F}} H(|S|)$.

The proof uses the next lemma, which bounds the degree to which the prices can be dual infeasible.

Lemma 11.3 For all S in $\mathcal{F}$ we have $\sum_{e \in S} y_e \leq c_S \cdot H(|S|)$.

Proof. Let F_k be the set chosen at the kth step. Let $e \in S \cap F_k \setminus \bigcup_{l<k} F_l$, and $V_k = S \setminus \bigcup_{l<k} F_l$ be the remaining part of S before e will be covered in the greedy cover. By the greedy property,

$$y_e \leq c_S / |V_k|,$$

since S could also have been chosen. 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

$$y_{e_k} = y_{e_{j(k)}} \leq \frac{c_S}{|V_{j(k)}|} = \frac{c_S}{|S| - j(k) + 1} \leq \frac{c_S}{|S| - k + 1}.$$

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.

$$\sum_S c_S x_S = \sum_e y_e \leq \sum_{S \in \mathcal{C}_*} \sum_{e \in S} y_e \leq \sum_{S \in \mathcal{C}_*} c_S \cdot H(|S|) \leq H(|S^*|) \sum_{S \in \mathcal{C}_*} c_S$$

where S^* is the largest set. □

Question 1 Is $\Theta(\log n)$ the best possible factor for set cover?

Yes (by a deep theorem), if NP-complete problems are hard.

Exercise Construct an example graph where result of the greedy vertex cover algorithm is indeed a factor $\Omega(\log d)$ worse than the optimum, and thus worse than our earlier algorithm which gave a factor 2.

Approximation scheme

An algorithm that for every ε, gives an $(1 + \varepsilon)$-approximation.

- 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.

Knapsack problem

Given: integers $b \geq a_1, \ldots, a_n$, and integer weights $w_1 \geq \ldots \geq w_n$.

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

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

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

If the set is empty the minimum is ∞. 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 $(A_k(0), \ldots, A_k(w))$. Namely

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

The optimum is $\max\{\, p : A_n(p) \le b \,\}$.
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 $r > 0$, $w_i'' = \lfloor w_i/r \rfloor$.

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

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

$$\mathbf{w}^T\mathbf{x}''/r \geq (\mathbf{w}'')^T\mathbf{x}'' \geq (\mathbf{w}'')^T\mathbf{x}^* \geq (\mathbf{w}/r)^T\mathbf{x}^* - n,$$
$$\mathbf{w}^T\mathbf{x}'' \geq \mathrm{OPT} - r \cdot n.$$

Let $r = \varepsilon w_1/n$, then

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

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

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

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

Look at the special case of knapsack, with $w_i = a_i$. Here, we just want to fill up the knapsack as much as we can. This is equivalent to minimizing the remainder,

$$b - \sum_i \mathbf{a}^T \mathbf{x}.$$

But this minimization problem is inapproximable.

Polynomial time

We measure the time cost of an algorithm as a function of the length of the input.

Examples 14.1

- The sieve algorithm to decide primality of an integer x is not polynomial-time.
- Multiplying two numbers of length n in time $O(n^2)$.
- Greatest common divisor in polynomial time.
- Length of shortest path betweem two points in a graph with unit edge-length: breadth-first search.
- Shortest path in a graph with integer edge lengths. Simple-minded reduction to breadth-first search leads to an exponential algorithm. Dijkstra's algorithm does it in polynomial time.

Examples 14.2

- 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 14.3 Given a graph G.

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

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.

Abstract problems

Instance. Solution (witness, certificate).

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.

Example 14.4 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 in time that is polynomial 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 14.5 (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 $1 \leq w \leq b(x)$ 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 $x_1, \ldots, x_n$.
  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 $f_C(x_1, \ldots, x_n)$. Circuit satisfiability is the question of satisfiability of this function.
- Assume also that every non-input node has exactly two incoming edges.

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 14.6 Reducing linear programming to linear programming in standard form.

NP-hardness.
NP-completeness.

Theorem 14.7 (Cook-Levin) Circuit satisfiability is NP-complete.

In order to prove this theorem, we must formalize the notion of computing time, and introduce a concrete machine model: the random access machine.
In another course, we would take more time to argue that the notion of polynomial-time computability remains the same also on other reasonable machine models.

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.

Theorem 14.8 If a random access machine computes a function $f(x)$ in time $O(n^c)$ then for each input length n there is a logic circuit of size $O(n^{2c})$ computing $f(x)$ from input x.

The proof builds a separate circuit for each computation step of the machine, and then concatenates these circuits. Each computation step is either a simple addition, decision or table access: any basic computer architecture course will teach you how to make a small (polynomial-size) circuit for it.

Proof sketch for the Cook-Levin theorem. 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 14.9 3-SAT is NP-complete.

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

Theorem 14.10 INDEPENDENT SET is NP-complete.

Reducing SAT to it.

Example 14.11

- Integer linear programming, in particular solving $\mathbf{Ax} = \mathbf{b}$, where the $m \times n$ matrix $\mathbf{A} \geq 0$ and the vector $\mathbf{b}$ consist of integers, and $x_j \in \{0, 1\}$.
  Case $m = 1$ is the subset sum problem.
- Reducing 3SAT to solving $\mathbf{Ax} = \mathbf{b}$.
- Reducing $\mathbf{Ax} = \mathbf{b}$ to $\mathbf{a}^T\mathbf{x} = b$ (subset sum). Let $D = 1 + \max_i |b_i| + \sum_j |a_{ij}|$. Define

$$A_j = \sum_i D^{i-1} a_{ij}, \quad B = \sum_i D^{i-1} b_i.$$

Then $\mathbf{Ax} = \mathbf{b} \Leftrightarrow \sum_j A_j x_j = B$.

Example 14.12 Set cover $\geq$ 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.

In case of problems difficult to solve exactly, 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 $f(x, A(x)) < m(x)\lambda\,.$

Try local improvements as long as you can.

Example 15.1 (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 15.2 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.

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

Recall our earlier treatment of greedy set cover and vertex cover.

- The greedy set cover algorithm gave a logarithmic factor approximation guarantee.
- This also applied to vertex cover. But a non-greedy algorithm (greedy for the dual) gave a better guarantee (factor 2) for vertex cover.

Definition 15.3 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

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

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

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 $X_1, \ldots, X_N$ be i.i.d. random variables with variance σ^2 and expected value μ. Then the inequality says

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

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

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

This will converge slowly if σ/μ is large.

Example 15.4 $X_i = 1$ 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 $\mathsf{P}[I = i] = |S_i|/M$. Pick an element $X \in S_I$ uniformly. Then for each x we have

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

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

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

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