Randomized Algorithms

Wolfgang Merkle
Heidelberg University
Institute of Computer Science

Table of Contents

Table of Contents

Links and References

The most recent version of these transparencies can be found on the webpage for the most recent corresponding course.

Older versions of the transparencies and outdated lecture notes for a minor part of the course can be obtained at
HTTP://MATH.UNI-HEIDELBERG.DE/LOGIC/SKRIPTEN.HTML.
As usual with lecture notes, in what follows we give references neither for the literature containing original work nor for other related publications.

A lot of material covered in the course can be found in the two following standard references on randomized algorithms.

- Michael Mitzenmacher und Eli Upfal, *Probability and Computing*, Cambridge University Press, 2017.
- Rajeev Motwani und Prabhakar Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.

Example 1 Half-duplex channel

Consider two network nodes A and B that are connected by a half-duplex channel, i.e., it is possible to send in both directions, but only in one direction at a time. In case A and B are sending at the same time they will recognize that the other node is sending, too, but they will not get any information about the data send by the other side.

In order to avoid a deadlock due to A and B trying to start sending repeatedly at the same time, one may use a protocol where

(a) if one side is already sending, the other side will never start sending,

(b) if both sides start sending at the same time, both stop and each side waits for a random time interval before trying to send again (while obeying Rule a).

Example 2 Randomized Quicksort

Recall that during the recursive calls of the Quicksort algorithm an element of the list to be processes is chosen as a pivot element, and then this list is split into the elements that are smaller and the elements that are larger than the pivot element,
Consider the two following variants of the Quicksort algorithm:

- deterministic Quicksort, where the pivot element is chosen according to some determinsitic procedure, e.g., is always equal to the first element, or to the median of the first, last, and middle elements.
- randomized Quicksort, where the pivot element is chosen uniformly at random from all elements in the current list.

Example 2 Quicksort

For both variants of the Quicksort algorithm there are cases where the chosen pivot elements are bad in the sense that the tree of partitions degenerates (and in worst case the number of comparisons required for a list with n elements is roughly n^2 instead of the average number $\mathrm{O}(n \log n)$):

- with deterministic Quicksort, there are rare bad inputs that always result in a bad choice of the pivot elements,
- with randomized Quicksort, all inputs are equally good, however, for any given input with small probability the random choices of the algorithm may result in a bad choice of the pivot elements.

Example 3 The copy game

In certain game-theoretical situations, the ability to pursue a randomized strategy makes a big difference.

The copy game

At the beginning of each round, two players A and B commit secretly to a value x_A and x_B, respectively, from $\{0, 1\}$.

Player A wins if the two values are distinct.
Player B wins if both values are the same.

Example 3 The copy game

- In case Player A follows a deterministic strategy that is known to (or can be learned) and can be simulated by Player B, then Player B can simply duplicate the value.
  Consequently, Player A will lose all the time.
- In case Player A follows a randomized strategy where her secret values are determined by independent tosses of a fair coin that are not known to the other player (more precisely, that are independent of the behavior of Player B).

  Then, on average, each player wins exactly half of the rounds.

  The latter holds no matter how dumb Player A and how smart or computationally powerful Player B is.

Excursus on the parity function $\oplus$

- The n-ary parity function $\oplus_n$ is the Boolean function on n arguments that yields 1 exactly when an odd number of the arguments is equal to 1.
- The 2-ary parity function $\oplus_2$, which we also denote by $\oplus$, is just the Exclusive-Or function (XOR function, for short), i.e.,

$$a \oplus b \quad = \quad \begin{cases} 1 & \text{if } a \text{ and } b \text{ are distinct,} \\ 0 & \text{otherwise.} \end{cases}$$

- A Boolean expression that contains only the binary parity function $\oplus$ and n Boolean arguments always evaluates to the value of the n-ary parity function applied to the same arguments. E.g. $(a_1 \oplus a_2) \oplus (a_1 \oplus (a_3 \oplus a_1))$ is equal to $\oplus_5(a_1, a_2, a_1, a_3, a_1)$.

Example 4 One-time pad

- Consider the situtation where $\mathbf{A}$ wants to send to $\mathbf{B}$ a word

$$w = w_1 \dots w_n, \qquad w_i \in \{0,1\} ,$$

while at the same time they try to prevent an eavesdropping adversary $\mathbf{E}$ from getting a clue what this message might be.

Suppose $\mathbf{A}$ and $\mathbf{B}$ both know an otherwise secret random word

$$r = r_1 \dots r_n$$

that has been obtained by independent tosses of a fair coin.

- Then $\mathbf{A}$ can simply send the message

$$w \oplus r = (w_1 \oplus r_1) \dots (w_n \oplus r_n) .$$

Example 4 One-time pad

- Even knowing the word $w \oplus r$, it is impossible to get *any* information about w as long as one has no relevant information about r.

  Indeed, for any fixed word w, the encrypted message $w \oplus r$ attains any possible word of length n with the same probability because r is determined by independent tosses of a fair coin.
- The technique of XOR-ing the message with a random string is called *one-time pad* for the following reason.

  The random word r must not be used with more than one message. For example, if r is used with w_0 and w_1, then from the two encrypted messages $w_0 \oplus r$ and $w_1 \oplus r$ one can easily obtain

$$w_0 \oplus w_1 = (w_0 \oplus r) \oplus (w_1 \oplus r) ,$$

  hence one can tell whether w_0 and w_1 are the same.

Example 5 Dining cryptographers

Three cryptographers A, B, and C eat at a restaurant. When they find out that their bill has already been paid, they wonder whether one of them has paid or somebody else (e.g., the NSA, which they would dislike).
They want to apply a protocol that will determine whether one of them has paid such that after the protocol has been executed,

- all three know whether one of them or somebody else has paid,
- anyone who has not paid will obtain no information at all about whom of the two others might have paid.

It is assumed that all three collaborate in the protocol, i.e., behave honestly and follow the rules of the protocol.

Example 5 Dining cryptographers

- Any two of the cryptographers toss a fair coin in order to obtain a common random bit, which is unknown to the third one.

  For example, A and B determine a random bit $r_{\mathrm{A,B}}$ that is unknown to C.
- For any cryptographer X let u_{X} be the parity of the two random bits obtained by X, e.g., let $u_{\mathrm{A}} = r_{\mathrm{A,B}} \oplus r_{\mathrm{A,C}}$
- Then any cryptographer X publishes u_{X} in case X has not paid, and publishes the complement of u_{X}, otherwise.
- Then $u_{\mathrm{A}} \oplus u_{\mathrm{B}} \oplus u_{\mathrm{C}}$ is equal to 1 if and only if one of the three cryptographers has paid (where we assume that at most one person can pay the bill).

Discrete probability measures

- Consider a chance experiment with outcomes in a set Ω where Ω is finite or countably infinite, i.e., Ω is of the form

$$\Omega = \{\omega_1, \ldots, \omega_n\} \qquad \text{or} \qquad \Omega = \{\omega_1, \omega_2, \ldots\} \,.$$

- The probabilities are given by a real-valued function

$$\mathrm{Prob} : \Omega \to [0,1]$$

such that the values $\mathrm{Prob}[\omega]$ add up to 1.
Prob is a *discrete probability distribution*, (Ω, Prob) is a *discrete probability space*.

- A subset of Ω is called an *event*. The function Prob can be extended to all events E by letting

$$\mathrm{Prob}[E] \;=\; \sum_{\omega \in E} \mathrm{Prob}[\omega] \,.$$

Uniform measure

- The *uniform measure* on a finite set Ω is given by $\mathrm{Prob}[\omega] = \frac{1}{|\Omega|}$ for all $\omega \in \Omega$.

  E.g., a cast of a fair die can be modelled by the *uniform distribution* on $\Omega = \{1, \ldots, 6\}$.
- On a countably infinite set there is no uniform measure.

Random variables

- A *random variable* is a mapping $X : \Omega \to \mathbb{R}$.
- Any random variable X that is defined on a discrete probability space (Ω, Prob) defines a discrete probability measure Prob_X on its range $\Omega_X = \{X(\omega) : \omega \in \Omega\}$ where

$$\mathrm{Prob}_X[x] \; = \sum_{\{\omega\in\Omega : X(\omega)=x\}} \mathrm{Prob}[\omega]\;.$$

  Prob_X is called the *distribution* of X.
- We write $\mathrm{Prob}[X = x]$ instead of $\mathrm{Prob}_X[x]$, and also use notation such as $\mathrm{Prob}[X \geq x]$, $\mathrm{Prob}[X \in S], \ldots$.

Indicator variables

- The *indicator variable* for a set $A \subseteq \Omega$ is the random variable $X : \Omega \to \{0, 1\}$ such that $X(\omega) = 1$ holds if and only if $\omega \in A$.

Example: $\Omega = \{1, \ldots, 6\}$, $\mathrm{Prob}[\omega] = \frac{1}{6}$ for all $\omega \in \Omega$.

Consider the indicator variable $X : \Omega \to \mathbb{R}$ for the set of primes less than or equal to 6

$$X(i) = \begin{cases} 1 & \text{in case } i \text{ is prim} \\ 0 & \text{otherwise} \end{cases}$$

$\mathrm{Prob}[X = 0] = \mathrm{Prob}[\{1, 4, 6\}] = 1/2,$
$\mathrm{Prob}[X = 1] = \mathrm{Prob}[\{2, 3, 5\}] = 1/2.$

Joint distribution

- Let $X_1, \ldots, X_m$ be random variables on the same discrete probability space Ω.
- The *joint distribution* $\mathrm{Prob}_{X_1,\ldots,X_m}$ is

$$\mathrm{Prob}_{X_1,\ldots,X_m}[r_1, \ldots, r_m] = \sum_{\{\omega\in\Omega : X_1(\omega)=r_1,\ldots,X_m(\omega)=r_m\}} \mathrm{Prob}[\omega] \, . \quad (1)$$

- We write $\mathrm{Prob}[X_1 = r_1, \ldots, X_m = r_m]$ instead of $\mathrm{Prob}_{X_1,\ldots,X_m}[r_1, \ldots, r_m]$.

Joint distribution

Example: $\Omega = \{1, \ldots, 6\}$, $\mathrm{Prob}[\omega] = \frac{1}{6}$ for all $\omega \in \Omega$.

Consider indicator variables X, Y and Z for the events ω is prime, ω is even, and ω is odd.
X, Y, and Z have the same distribution, the uniform distribution on $\{0, 1\}$. However,

$$\begin{aligned}\mathrm{Prob}[X = 1, Y = 1] &= \frac{1}{6}\,,\\ \mathrm{Prob}[X = 1, Z = 1] &= \frac{2}{6}\,.\end{aligned}$$

So the joint distribution is not determined by the individual distributions.

Mutually independent random variables

- Let $X_1, \ldots, X_m$ be random variables on the same discrete probability space Ω.
- $X_1, \ldots, X_m$ are *mutually independent* if for any combination of values $r_1, \ldots, r_m$ in the range of $X_1, \ldots, X_m$, respectively,

$$\begin{aligned}&\mathrm{Prob}[X_1 = r_1, \ldots, X_m = r_m] \\ &\qquad = \mathrm{Prob}[X_1 = r_1] \cdot \cdots \cdot \mathrm{Prob}[X_m = r_m] .\end{aligned}$$

Example: m tosses of a fair coin

Consider m tosses of a fair coin and let X_i be the indicator variable for the event that the ith toss shows head. Then the $X_1, \ldots, X_m$ are mutually independent and for all $(r_1 \ldots r_m) \in \{0,1\}^m$

$$\mathrm{Prob}[X_1 = r_1, \ldots, X_m = r_m] = \tfrac{1}{2^m}.$$

Pairwise and k-wise independence

- Random variables $X_1, \ldots, X_m$ are *pairwise independent* if all pairs X_i and X_j with $i \neq j$ are mutually independent, i.e., for all $i \neq j$ and all r_i and r_j

$$\begin{aligned}&\mathrm{Prob}[X_i = r_i \text{ and } X_j = r_j] \\ &\qquad = \mathrm{Prob}[X_i = r_i] \cdot \mathrm{Prob}[X_j = r_j]\,.\end{aligned}$$

- The concept of k-wise independence of random variables for any $k \geq 2$ is defined similar to pairwise independence, where now every subset of k distinct random variables must be mutually independent.

Mutual versus pairwise independence

- It can be shown that mutual independence implies pairwise independence.
- For three or more random variables, in general pairwise independence does not imply mutual independence.
- It can be shown for any $k \geq 2$ that $(k+1)$-wise independence implies k-wise independence, whereas the reverse implication is false.
- Examples of random variables that are k-wise independent but are not mutually independent will be constructed in the section on derandomization.
  An even simpler example is the following.

Pairwise and 3-wise independence

Example: pairwise but not 3-wise independence.

Consider a chance experiment where a fair coin is tossed 3 times and let X_i be the indicator variable for the event that coin i shows head. Let

$$Z_1 = X_1 \oplus X_2, \quad Z_2 = X_1 \oplus X_3, \quad Z_3 = X_2 \oplus X_3 \ .$$

The random variables Z_1, Z_2, and Z_3 are pairwise independent because for any pair i and j of distinct indices in $\{1, 2, 3\}$ and any values b_1 and b_2 in $\{0, 1\}$ we have

$$\mathrm{Prob}[Z_i = b_1 \& Z_j = b_2] = 1/4 \ .$$

On the other hand, Z_1, Z_2, and Z_3 are not 3-wise independent because for example we have $Z_1 = Z_2 \oplus Z_3$.

Expectation

- The *expectation of X* is

$$\mathbf{E}[X] = \sum_{\omega \in \Omega} \mathrm{Prob}[\omega] X(\omega) ,$$

  provided that this sum converges absolutely. If the latter condition is satisfied, we say the expectation of X exists.
- Recall that
  - $\sum_{i \in \mathbb{N}} a_i$ converges to s if and only if the partial sums $a_0 + \ldots + a_n$ converge to s,
  - $\sum_{i \in \mathbb{N}} a_i$ converges absolutely if even the sum $\sum_{i \in \mathbb{N}} |a_i|$ converges,
  - absolute convergence is equivalent to convergence to the same value under arbitrary reorderings.
- The condition on absolute convergence ensures that the expectation is the same no matter how we order Ω.
  The condition is always satisfied if Ω is finite or if X is non-negative and the sum converges at all.

Expectation

Example: $\Omega = \{1, \ldots, 6\}$, $\mathrm{Prob}[\omega] = \frac{1}{6}$ for all $\omega \in \Omega$.

If we let X be the identity mapping on Ω, then

$$\mathbf{E}[X] = \sum_{i \in \{1,\ldots,6\}} \mathrm{Prob}[i]\, X(i) = \frac{1}{6} + \frac{2}{6} + \ldots + \frac{6}{6} = \frac{21}{6} = 3.5 \,.$$

Example: $\Omega = \mathbb{N}$, $\mathrm{Prob}[i] = \frac{1}{2^{i+1}}$.

The expectation of the random variable

$$X : i \mapsto 2^{i+1}$$

does not exist because the corresponding sum does not converge

$$\sum_{i \in \mathbb{N}} \mathrm{Prob}[i]\, X(i) = \sum_{i \in \mathbb{N}} \frac{1}{2^{i+1}} 2^{i+1} = 1 + 1 + \ldots = +\infty \,.$$

Linearity of Expectation

- Let X and $X_1, \ldots, X_n$ be random variables such that their expectations all exist.
- Expectation is linear.

  For any real number r, the expectation of rX exists and it holds that

  $$\mathbf{E}[rX] = r\mathbf{E}[X] .$$

  The expectation of $X_1 + \ldots + X_m$ exists and it holds that

  $$\mathbf{E}[X_1 + \cdots + X_n] \;=\; \mathbf{E}[X_1] + \cdots + \mathbf{E}[X_n] .$$
- If the $X_1, \ldots, X_n$ are *mutually independent*, then the expectation of $X_1 \cdot \ldots \cdot X_m$ exists and

  $$\mathbf{E}[X_1 \cdot \;\cdots\; \cdot X_n] \;=\; \mathbf{E}[X_1] \cdot \;\cdots\; \cdot \mathbf{E}[X_n] .$$

Number of fixed points of a random permutation

- Suppose that n tokens $T_1, \ldots, T_n$ are distributed at random among n persons $P_1, \ldots, P_n$ such that each person gets exactly one token and all such assignments of tokens to persons have the same probability
(i.e., the tokens are assigned by choosing a permutation of $\{1, \ldots, n\}$ uniformly at random).
- What is the expected number of indices i such that P_i gets his or her "own token" T_i?

  If we let X_i be the indicator variable for the event that P_i gets T_i, then $X = \sum_{i=1}^{n} X_i$ is equal to the random number of persons that get their own token.
- By linearity of expectation, the expectation of X is

$$\mathbf{E}\left[X\right] = \mathbf{E}\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} \mathbf{E}\left[X_i\right] = \sum_{i=1}^{n} \left(\frac{n-1}{n}0 + \frac{1}{n}1\right) = 1 \,.$$

Conditional distributions and expectations

- The *conditional probability* of an event E given an event F is

$$\mathrm{Prob}[E|F] = \frac{\mathrm{Prob}[E \cap F]}{\mathrm{Prob}[F]} ,$$

  where this value is undefined in case $\mathrm{Prob}[F] = 0$.
- The *conditional distribution* $\mathrm{Prob}[.|F]$ of a random variable X given an event F is defined by

$$\mathrm{Prob}[X = a|F] = \frac{\mathrm{Prob}[\{\omega \in \Omega \colon X(\omega) = a\} \cap F]}{\mathrm{Prob}[F]} .$$

- The *conditional expectation* $\mathbf{E}[X|F]$ of a random variable X given an event F is the expectation of X with respect to the conditional distribution $\mathrm{Prob}[X|F]$, i.e.,

$$\mathbf{E}[X|F] \;=\; \sum_{a \in \mathrm{range}(X)} a \cdot \mathrm{Prob}[X = a|F] .$$

Markov Inequality

Proposition (Markov Inequality)

Let X be a random variable that assumes only non-negative values. Then for every positive real number r, we have

$$\mathrm{Prob}[X \geq r] \;\leq\; \frac{\mathbf{E}[X]}{r}\,.$$

Proof.

Let (Ω, Prob) be the probability space on which X is defined. Then we have

$$\begin{aligned}\mathbf{E}[X] = & \sum_{\omega\in\Omega} \mathrm{Prob}[\omega] X(\omega) \geq \sum_{\{\omega\in\Omega : X(\omega)\geq r\}} \mathrm{Prob}[\omega] X(\omega) \\ \geq & \; r \sum_{\{\omega\in\Omega : X(\omega)\geq r\}} \mathrm{Prob}[\omega] \geq r\,\mathrm{Prob}[X \geq r]\,.\end{aligned}$$

□

The tenure game

- The tenure game is played by two players Alice and Bob.
- Initially, finitely many tokens are placed at positions that are nonzero natural numbers.
- Then Alice and Bob alternate in their moves where always

  first Bob partitions the tokens in two parts,

  then Alice chooses one part that is removed, while the other tokens move one position closer to position 0.
- Bob wins if and only if some token reaches position 0.
- Picture each token as a nontenured staff member who gets tenure at position 0; Alice is head of department and wants to prevent staff from getting tenure, whereas Bob is dean and wants to give tenure to as many people as possible.

The tenure game

Initial configuration of the tenure game.

Finitely many tokens $1, \ldots, m$ are placed at positions $d_1, \ldots, d_m$ where the d_i are arbitrary nonzero natural numbers.

The moves in the tenure game.

I Partition.
Bob partitions the current set I of tokens that are not at position 0 into two sets I_0 and I_1
(i.e., I is the disjoint union of I_0 and I_1).

II Selection.
Alice determines a bit r.

III Removal and promotion.
The tokens in I_r are removed.
The tokens in I_{1-r} are moved on step closer to position 0
Tokens that were already at position 0 just stay there.

Winning condition for the tenure game

Termination of the tenure game.

The tenure game ends when each token either has been removed or has reached position 0.

Winning condition for the tenure game.

Bob wins if some token reaches position 0, otherwise, Alice wins.

- Observe that a tenure game is decided and hence may be stopped as soon as some token has reached position 0 because then the token will stay at position 0 and Bob will win.

  However, the version of the game where the play goes on until the termination condition holds is equivalent and somewhat easier to analyze.

Winning strategies for the tenure game

Observation

For any given initial configuration of the tenure game, either Alice or Bob has a winning strategy.

The observation holds because the tenure game is a finite two-person zero-sum game with complete information, see the excursus on such games at the end of this section.

Question

Given an initial configuration of the tenure game, who has a winning strategy, Alice or Bob?

Games that Alice win and games that Bob win

Examples of initial configurations for the tenure game.

(i) 2^k-1 tokens at position k.

(ii) 2^k tokens at position k.

(iii) One token at each position 1 through k.

(iv) One token at each position 1 through $k-1$
and two tokens at position k.

(v) Two tokens at each position 2 through k.

(vi) Two tokens at each position 2 through $k-1$
and four tokens at position k.

It is not so hard to see that the odd-numbered configurations are winning for Alice and the other configurations are winning for Bob.

Question

What is the general pattern?

Winning configurations for Alice

A randomized strategy for Alice

Alice determines each bit r by tossing of a fair coin.

No matter what the strategy of Bob is and how Bob partitions, when Alice plays the randomized strategy, then

each time Alice selects, any single token will be removed or promoted with equal probabilities of $1/2$.

Furthermore, since the coin tosses are independent, a token that is initially at position d, will reach position 0 with probabilty $1/2^d$.

Winning configurations for Alice

- Consider the tenure game where initially tokens $1, \ldots, m$ are placed at positions $d_1, \ldots, d_m$, respectively.
- Fix any strategy for Bob.

  Assume Alice plays the randomized strategy.
- Define indicator variables $X_1, \ldots, X_m$ where X_i is 1 if and only if token i reaches position 0.

  Let $X = X_1 + \ldots + X_m$ be the number of tokens that reach the origin.
- The expectation of X is

$$\mathbf{E}[X] = \mathbf{E}[X_1 + \cdots + X_m] = \mathbf{E}[X_1] + \cdots + \mathbf{E}[X_m] = \frac{1}{2^{d_1}} + \cdots + \frac{1}{2^{d_m}} \ .$$

  For any configuration with m tokens at positions $d_1, \ldots, d_m$, we call $\frac{1}{2^{d_1}} + \cdots + \frac{1}{2^{d_m}}$ the *potential* of the configuration.

Winning configurations for Alice

- Consider a tenure game where the potential $\frac{1}{2^{d_1}} + \cdots + \frac{1}{2^{d_m}}$ of the intial configuration is strictly less than 1.
- In this situation, also $\mathbf{E}[X]$, the expected number of tokens that reach position 0, is strictly less than 1

  Hence there must be some sequence of coin tosses where $X < 1$, i.e., such that no token reaches position 0.

  This means that there is a sequence of coin tosses such that Alice wins.
- The argument above does not depend on Bob's strategy.

  That is, no matter what the strategy of Bob is, Alice can win.

  As a consequence, Bob cannot have a winning strategy.

  But then Alice must have a winning strategy.

Winning strategy for Alice

Proposition

If the potential of the initial configuration is strictly less than 1*, then Alice has a winning strategy.*

- How does the winning strategy according to the proposition look like?
- A *terminal configuration* is a configuration where all tokens are at position 0.
  The potential of a terminal configuration is equal to the number of its tokens

Observation

A terminal configuration is winning for Alice if and only if the potential of the configuration is strictly less than 1.

- Idea for a strategy: Alice tries to maintain the invariant that the potential of the current situation is strictly less than 1.

Winning strategy for Alice

- If the initial configuration has potential strictly less than 1, then Alice can maintain this property by the following strategy.

Strategy for Alice

Choose r such that when comparing the potentials of the two configurations that correspond to I_0 and I_1, the potential that corresponds to I_r is higher.

- Assume that no token has yet reached position 0, and consider the partition of the set I of current tokens into I_0 and I_1.

  If we let p, p_0, and p_1 be equal to the potentials that correspond to I, I_0 and I_1, respectively, we have $p = p_0 + p_1$.
- After the tokens in I_{1-r} have been promoted, the potential of the new configuration is $2p_{r-1}$.

  By choice of r we have $2p_{r-1} \leq p_0 + p_1 = p$.

Winning strategy for Bob

- What's about initial configurations with potential of 1 or more?

Observation

A terminal configuration is winning for Bob if and only if the potential of the configuration is at least 1.

- Bob wins if he can maintain the invariant that the potential is at least 1.
- Suppose Bob partitions the set of current tokens into sets I_0 and I_1 where both correspond to a potential of at least $1/2$. Then the subsequent promotion step ensures that the potential will be at least 1 again.

Strategy for Bob

Partition the set of current tokens into sets I_0 and I_1 such that both sets correspond to a potential of at least $1/2$.

Bob can partition equally

- By the following lemma, the strategy for Bob is feasible .

Lemma

Let $d_1 \leq \ldots \leq d_m$ be a nondecreasing sequence of nonzero natural numbers where $\frac{1}{2^{d_1}} + \cdots + \frac{1}{2^{d_m}} \geq \frac{1}{2}$. Then there is an index t such that

$$\frac{1}{2^{d_1}} + \cdots + \frac{1}{2^{d_t}} = \frac{1}{2} .$$

Proof.

- By assumption on the d_i, let t be minimum such that $s_t = \frac{1}{2^{d_1}} + \cdots + \frac{1}{2^{d_t}} \geq \frac{1}{2}$.
- In case $s_t = 1/2$ we are done.

  Otherwise, s_t and $1/2$ are both multiples of $\frac{1}{2^{d_t}}$ but differ by less than $\frac{1}{2^{d_t}}$, a contradiction.

Excursus on game theory

In a finite two-person game with perfect information

two players alternate in specifying moves,

the player whose turn it is to specify the next move knows the sequence of previous moves,

there are always at most finitely many admissible moves,

any sequence of admissible moves ends after a finite number of moves with one player winning and the other player losing.

- The situation reached in such a game after a certain sequence of moves is called a *configuration*.

  Configurations can be assumed to be finitely represented because in any case one can specify a configuration by the sequence of moves that led to the configuration.

Excursus on game theory

The game tree of a finite two-person game with perfect information is

- a labeled tree where each node is labeled with a configuration of the game,
- the root of the tree is labeled with the initial configuration,
- the children of a node are labeled in a one-to-one fashion with the configurations that can be reached by an admissible move from the this nodes configurations.

Observation

Game trees of finite two-person games with perfect information are always finite.

The observation is an immediate consequence of König's Lemma.

König's Lemma

Any finitely branching infinite tree has an infinite path.

Proof.

Let T be any finitely branching infinite tree.
We construct inductively an infinite path $v_0, v_1, \ldots$ on T.
As an invariant of the construction, any node on the path is chosen such that the subtree of T below this node is infinite.
Initially, let v_0 be equal to the root of T.
In the induction step, given the already constructed initial segment $v_0, \ldots v_i$, let v_{i+1} be equal to the least such child of v_i such that the subtree below this child is infinite.
Such a child exists because the subtree of T below v_i is infinite and v_i has only finitely many children.
Furthermore, the invariant is always true, hence the construction will not terminate and yields an infinite path on T.

Excursus on game theory

Definition

Consider a two-person game with players $\mathbf{A}$ and $\mathbf{B}$.

A *strategy* for Player $\mathbf{A}$ is a function that determines the next move of Player $\mathbf{A}$ whenever it is $\mathbf{A}$'s turn to specify a move.

A *winning strategy* for Player $\mathbf{A}$ is a strategy such that Player $\mathbf{A}$ always wins when using this strategy, no matter what strategy Player $\mathbf{B}$ uses.

Strategies and winning strategies for Player $\mathbf{B}$ are defined likewise in the obvious way.

Definition

A two-person game with perfect information is *determined* if one of the two players has a winning strategy.

Excursus on game theory

Theorem

Finite two-person games with perfect information are determined.

Proof. Fix any finite two-person game with perfect information.

We have already seen that the game tree T of this game is finite.
We show for every node v of T that the "subgame" that starts at v is determined; we show this by induction on the height of the subtree of T with root v
If the height of the subtree is 0, the node is a leave of the tree and one of the players wins immediately.
In the induction step consider a node v where the subtree has height $h > 0$ and assume that at v Player $\mathbf{A}$ specifies a move.
By the induction hypothesis, for all children of v the corresponding subgames are determined. In case $\mathbf{A}$ has a winning strategy for one of these subgames, then $\mathbf{A}$ has winning strategy from v; otherwise $\mathbf{B}$ has a winning strategy from v.

Excursus on game theory

- Consider a two-person game with perfect information played by $\mathbf{A}$ and $\mathbf{B}$.
  If $\mathbf{A}$ has a winning strategy, then $\mathbf{A}$ always wins when using this strategy, no matter what strategy $\mathbf{B}$ uses.
  If $\mathbf{A}$ has no winning strategy, this does not imply directly that $\mathbf{B}$ has a winning strategy, but only that for any strategy of $\mathbf{A}$ there is some strategy of $\mathbf{B}$ such that $\mathbf{B}$ will win.

Remark

Infinite two-person games with perfect information in general are not determined.

- Consider an infinite two-person game where two players alternate in specifying the next bit of an infinite sequence and $\mathbf{A}$ wins if and only if the resulting sequence is contained in a certain set C of sequences, otherwise $\mathbf{B}$ wins.
  Using the axiom of choice, one can show that there are certain sets C for which this game is not determined.

A randomized algorithm for finding a cut

- In what follows, "graph" refers to a finite undirected graph.
- A *cut* of a graph $G = (V, E)$ is a partition of V into two disjoint subsets V_0 and V_1.
  The *weight of a cut* (V_0, V_1) is the number of edges between V_0 and V_1, i.e., the weight is

$$|\{ \{u, v\} \in E : u \in V_0, v \in V_1\}| \, .$$

Algorithm Cut

```
Input: A graph $G = (V, E)$ where $V = \{1, \dots, n\}$.
    Choose random bits $r_1, \dots, r_n$ by independent tosses
                                                  of a fair coin.
    Let $V_0 = \{i : r_i = 0\}$.
    Let $V_1 = \{i : r_i = 1\}$.
Output: The cut $(V_0, V_1)$.
```

The Algorithm Cut

Proposition

Let G be a graph with m edges. On input G, the expected weight of the cut returned by Algorithm Cut *is $m/2$.*
In particular, the graph G has a cut with weight at least $m/2$.

Proof

- Let $G = (V, E)$ be a graph with m edges $e_1, \ldots, e_m$.
  Introduce indicator variables $\widehat{e_i}$ where $\widehat{e_i} = 1$ iff
  edge $e_i = \{u_i, v_i\}$ crosses the cut returned by Algorithm Cut.
- The weight $\widehat{\mathrm{w}}_G$ of the cut returned by Algorithm Cut is just the sum of the $\widehat{e_i}$, hence by linearity of expectation we have

$$\mathbf{E}\left[\widehat{\mathrm{w}}_G\right] = \mathbf{E}\left[\widehat{e_1} + \ldots + \widehat{e_m}\right] = \mathbf{E}\left[\widehat{e_1}\right] + \ldots + \mathbf{E}\left[\widehat{e_m}\right] .$$

Verification of the Algorithm Cut

Proof (continued)

- So it suffices to show $\mathbf{E}[\widehat{e_i}] = 1/2$. Indeed, we have

$$\begin{aligned}
\mathbf{E}[\widehat{e_i}] &= 1 \cdot \mathrm{Prob}[\widehat{e_i} = 1] + 0 \cdot \mathrm{Prob}[\widehat{e_i} = 0] \\
&= \mathrm{Prob}[u_i \in V_0 \text{ and } v_i \in V_1] + \mathrm{Prob}[u_i \in V_1 \text{ and } v_i \in V_0] \\
&\overset{(*)}{=} \mathrm{Prob}[u_i \in V_0]\,\mathrm{Prob}[v_i \in V_1] \\
&\qquad + \mathrm{Prob}[u_i \in V_1]\,\mathrm{Prob}[v_i \in V_0] \\
&= \frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2}
\end{aligned}$$

Equation $(*)$ holds because the assignments of nodes to the two sides of the cut are mutually and hence pairwise independent.

Verification of the Algorithm Cut

Proof (continued)

- If all possible outcomes of the coin tosses resulted in a cut with weight strictly less than $m/2$, then $\mathbf{E}\left[\widehat{\mathrm{w}}_G\right]$ would be strictly less than $m/2$.

  Hence there is a sequence of coin tosses that yield a cut with weight at least $m/2$.

  Hence there is a cut with weight at least $m/2$. $\square$

Derandomization by conditional expectation

- Let $G = (V, E)$ be a graph with n nodes and m edges and let α be any word of length $s \leq n$.
- Let Cut_α be the algorithm that works like Algorithm Cut, except that $r_1 \dots r_s$ is set equal to α.
- Let $\widehat{\mathrm{w}}_G(\alpha)$ be the weight of the random cut returned by Algorithm Cut_α on input G.

  Note that $\mathbf{E}[\widehat{\mathrm{w}}_G(\lambda)] = \mathbf{E}[\widehat{\mathrm{w}}_G] = \frac{m}{2}$ (λ is the empty word).
- Algorithm Cut_α, with probability of $1/2$ each,

  sets r_{s+1} to 0, i.e., works like $\mathrm{Cut}_{\alpha 0}$,
  sets r_{s+1} to 1, i.e., works like $\mathrm{Cut}_{\alpha 1}$.

  In term of the expected weight of the returned cut, this means

$$\mathbf{E}[\widehat{\mathrm{w}}_G(\alpha)] = \frac{1}{2}\mathbf{E}[\widehat{\mathrm{w}}_G(\alpha 0)] + \frac{1}{2}\mathbf{E}[\widehat{\mathrm{w}}_G(\alpha 1)],$$

  Hence at least one of the expected values on the right-hand side is at least as large as $\mathbf{E}[\widehat{\mathrm{w}}_G(\alpha)]$.

Derandomization by conditional expectation

Derandomized Algorithm Cut (conditional expectation)

```
Input: A graph $G = (V, E)$                    $(V = \{1, \ldots, n\})$.
  For $s = 1, \ldots, n$
    If    $\mathbf{E}[\widehat{\mathrm{w}}_G(r_1 \ldots r_{s-1}0)] \geq \mathbf{E}[\widehat{\mathrm{w}}_G(r_1 \ldots r_{s-1}1)]$   $(*)$
    then $r_s = 0$ else $r_s = 1$.
  Let $V_0 = \{i : r_i = 0\}$.
  Let $V_1 = \{i : r_i = 1\}$.
Output: The cut $(V_0, V_1)$.
```

- The algorithm returns a cut of weight $w_G \geq \frac{m}{2}$ because of

$$\begin{aligned}\frac{m}{2} &= \mathbf{E}[\widehat{\mathrm{w}}_G(\lambda)] \leq \mathbf{E}[\widehat{\mathrm{w}}_G(r_1)] \leq \mathbf{E}[\widehat{\mathrm{w}}_G(r_1 r_2)] \leq \ldots \\ &\leq \mathbf{E}[\widehat{\mathrm{w}}_G(r_1, \ldots, r_{n-1})] \leq \mathbf{E}[\widehat{\mathrm{w}}_G(r_1, \ldots, r_n)] = \mathrm{w}_G \ .\end{aligned}$$

Derandomization by conditional expectation

- Is the derandomized version of Cut efficient?
  How complex is it to evaluate Condition $(*)$?
- Consider iteration s of the for-loop of the algorithm, assuming that $r_1 \dots r_{s-1}$ have already been defined.
  Partition E into four sets E_1, E_2, E_3^0, and E_3^1 defined by

$$\begin{aligned} E_1 &= \{\ \{j_1, j_2\} \in E\ : j_1 < s \text{ and } j_2 < s\}\ , \\ E_2 &= \{\ \{j_1, j_2\} \in E\ : j_1 > s \text{ or } j_2 > s\}\ , \\ E_3^i &= \{\ \ \{j, s\} \in E\ \ : j < s \text{ and } r_j = i\}\ . \end{aligned}$$

- For the edges in E_1 and E_2, the choice of r_s does not matter.
  The constructed cut will contain from E_1 exactly the edges in

$$E_1' = \{\{j_1, j_2\} \in E_1 : r_{j_1} \neq r_{j_2}\}\ .$$

  The constructed cut will contain each edge in E_2 with probability $1/2$.

Derandomization by conditional expectation

- Concerning the sets E_3^i, the choice of r_s does matter.
  The constructed cut contains all edges from $E_3^{1-r_s}$ but no edge from $E_3^{r_s}$, hence has an expected weight of

$$\mathbf{E}\left[\widehat{\mathrm{w}}_G(r_1 \dots r_{s-1} r_s)\right] \;=\; |E_1'| + \frac{1}{2}|E_2| + |E_3^{1-r_s}|\;.$$

- The expected weight is maximized by maximizing $|E_3^{1-r_s}|$, i.e., by maximizing the number of edges between node s and the nodes $1, \dots, s-1$.
- This means that we let $r_s = 0$ in case

$$|\{j < s : \{j,s\} \in E \text{ and } r_j = 1\}|$$

  is at least as large as

$$|\{j < s : \{j,s\} \in E \text{ and } r_j = 0\}|\;,$$

  and, otherwise, we let $r_s = 1$.

Derandomization by conditional expectation

Derandomized Algorithm Cut (conditional expectation)

```
Input: A graph $G = (V, E)$                    $(V = \{1, \ldots, n\})$.
  For $s = 1, \ldots, n$
     If $|\{j < s : \{j, s\} \in E \text{ and } r_j = 1\}|$
                 $\geq |\{j < s : \{j, s\} \in E \text{ and } r_j = 0\}|$
     then $r_s = 0$ else $r_s = 1$ .
  Let $V_0 = \{i : r_i = 0\}$.
  Let $V_1 = \{i : r_i = 1\}$.
Output: The cut $(V_0, V_1)$.
```

- We end up with an extremely simple deterministic algorithm.
- The randomized version of the algorithm may be used for verifying that the deterministic version works as required.

Derandomization by pairwise independence

- *Trivial derandomization* refers to simulating a randomized algorithm for every possible value of its random source.
- Consider a randomized algorithm that on any input of size n runs for at most $t(n)$ steps.
  The algorithm might use up to $t(n)$ random bits, hence its trivial derandomization runs for about $2^{t(n)}t(n)$ steps.
- Now assume that the algorithm uses only $\log t(n)$ many random bits. Then its trivial derandomization runs for a number of steps that is about

$$2^{\log t(n)}t(n) = t(n) \cdot t(n) = t(n)^2 \ .$$

Derandomization method of small sample spaces

First transform a randomized algorithm into one that uses only few random bits, then apply trivial derandomization.

Derandomization by pairwise independence

- For suitable randomized algorithms, the sample space can be made small by working with k-wise instead of mutually independent random bits.
- The verification of Algorithm Cut relied solely on the fact that each bit $r_1, \ldots, r_n$ is uniformly distributed and that these bits are *pairwise* independent.
- How many mutually independent uniformly distributed random bits are required to specify such n pairwise independent bits?

  That is, how small can we choose the sample space?

  About **log *n*** bits are sufficient!

  This yields a sample space of size about n.

  Trivial derandomization then yields an algorithm that runs in polynomial time.

Derandomization by pairwise independence

Construction of n pairwise independent random bits with uniform distribution from a small sample space

For given n, let $t = \lceil \log(n+1) \rceil$ and let $I = \{1, \ldots, t\}$.

Let random bits $b_1, \ldots, b_t$ be obtained by tosses of a fair coin.

Let $J_1, \ldots, J_n$ be pairwise distinct nonempty subsets of I.

For $i = 1, \ldots, n$ let $r_i = \bigoplus_{j \in J_i} b_i$.

In connection with this construction observe that the parameter t is chosen as the unique natural number where

$$2^{t-1} < n + 1 \leq 2^t \ ,$$

hence there are at least n sets J_i as required since $n \leq 2^t - 1$. Furthermore, the size 2^t of the sample space is in $\{n+1, \ldots, 2n\}$.

Derandomization by pairwise independence

Lemma (Verification of the construction)

The random bits $r_1, \ldots, r_n$ obtained by the construction above are pairwise independent and each bit is uniformly distributed.

Proof.

Fix any $t > 0$ and assume that random bits $b_1, \ldots, b_t$ are obtained by tosses of a fair coin.
Fix any nonempty set $J \subseteq \{1, \ldots, t\}$ and let $r = \oplus_{j \in J} b_j$.
For any fixed index $s \in J$ we have,

$$r = \oplus_{j \in J} b_j = (\oplus_{j \in J \setminus \{s\}} b_j) \oplus b_s.$$

That is, if the b_j have already been determined for all j in $J \setminus \{s\}$, then depending on the value of b_s, the value of r will either agree with or will differ from $\oplus_{j \in J \setminus s} b_j$. But b_s attains its two possible values with probability $1/2$ each and the b_j are mutually independent, hence r will be uniformly distributed in $\{0, 1\}$.

Derandomization by pairwise independence

Proof (continued).

For any two distinct nonempty subsets J and J' of $\{1, \dots, t\}$, let

$$r = \bigoplus_{j \in J} b_j \quad \text{and} \quad r' = \bigoplus_{j \in J'} b_j \ .$$

Since the sets J and J' are distinct, there is some index s that is contained in one of the sets but not in the other; w.l.o.g. we assume $s \in J$.
Similarly to the proof of uniformity, we can then argue that

$$r = (\bigoplus_{j \in J \setminus \{s\}} b_j) \oplus b_s \qquad \text{and} \qquad r' = \bigoplus_{j \in J' \setminus \{s\}} b_j \ ,$$

and that hence if the b_j have already been determined for all j except s, then depending on the value of b_s, i.e., with probability $1/2$, the value of r will agree with or will differ from r', consequently r and r' are mutually independent.

Derandomization by pairwise independence

Algorithm $\mathrm{Cut}_{\mathrm{pi}}$ (Using pairwise independent random bits)

```
Input:  A graph $G = (V, E)$                    $(V = \{1, \ldots, n\})$.
    Let $t = \lceil \log(n+1) \rceil$.
    Let $I = \{1, \ldots, t\}$.
    Choose random bits $b_1, \ldots, b_t$ by tosses of a fair coin.
    Let $J_1, \ldots, J_n$ be pairwise distinct nonempty subsets of $I$.
    For $i = 1, \ldots, n$, let $r_i = \bigoplus_{j \in J_i} b_j$.
    Let $V_0 = \{i : r_i = 0\}$.
    Let $V_1 = \{i : r_i = 1\}$.
Output: The cut $(V_0, V_1)$.
```

- The expected weight of the cut returned by Algorithm $\mathrm{Cut}_{\mathrm{pi}}$ is $\frac{m}{2}$, the analysis is essentially the same as for Algorithm Cut.
- If we let $w^1, \ldots, w^n$ be the lexicographically least n words of length t that differ from 0^t, then we can simply let $J_i = \{\ell \colon w^i_\ell = 1\}$, where $w^i = w^i_1 \ldots w^i_t$, $w^i_j \in \{0, 1\}$.

Derandomization by pairwise independence

Derandomized Algorithm Cut (pairwise independence)

```
Input:   A graph $G = (V, E)$                          $(V = \{1, \dots, n\})$.
    Let $t = \lceil \log(n+1) \rceil$ and let $m = 2^t$.
    Let $b^1, \dots, b^m$ be the $m$ words of length $t$, $b^i = b^i_1 \dots b^i_t$.
    Let $w^1, \dots, w^n$ be the least $n$ words of length $t$ that
                                        differ from $0^t$, $w^i = w^i_1 \dots w^i_t$.
    For $i = 1, \dots, m$   (i.e., for all choices of the random bits),
      For $j = 1, \dots, n$,
        Let $r_{i,j} = \oplus_{\ell=1,\dots,t}(b^i_\ell \wedge w^j_\ell)$.     ($\wedge$ is the And-Operator)
      Let $V^i_0 = \{j : r_{i,j} = 0\}$.
      Let $V^i_1 = \{j : r_{i,j} = 1\}$.
Output:  A cut $(V^i_0, V^i_1)$ of maximum weight.
```

- The $(m \times n)$-matrix R with entries $r_{i,j}$ can be viewed as the product of the $(m \times t)$-matrix B with rows $b^1, \dots, b^m$ and the $(t \times n)$-matrix W with columns $w^1, \dots, w^n$.

Derandomization of Algorithm $\mathrm{HyperCut}$

- A *hypergraph* is a pair (V, E) where V is the set of nodes and E is a set of subsets of V.
  The subsets in E are called *hyperedges*.
- A hypergraph is *k-regular* if all its hyperedges contain k nodes.
- A *cut* of a hypergraph $G = (V, E)$ is a partition of V into two disjoint subsets V_0 and V_1.
  The *weight of a cut* (V_0, V_1) is the number of hyperedges that intersect both V_0 and V_1.

Algorithm $\mathrm{HyperCut}$

```
Input:   A hypergraph $G = (V, E)$                    $(V = \{1, \dots, n\})$.

   Choose random bits $r_1, \dots, r_n$ by independent
                                    tosses of a fair coin.

   Let $V_0 = \{i : r_i = 0\}$.
   Let $V_1 = \{i : r_i = 1\}$.

Output:  The cut $(V_0, V_1)$.
```

Derandomization of Algorithm HyperCut

Proposition

Let G be a hypergraph that is k-regular for some $k \geq 2$ and has m edges. The expected weight of the cut returned by Algorithm HyperCut *on input G is*

$$\left(1 - \frac{2}{2^k}\right) m \,.$$

In particular, the graph G has a cut of at least this weight.

Proof.

For each of the edges $e_1, \ldots, e_m$ of G introduce an indicator variable $\widehat{e_i}$ where $\widehat{e_i} = 1$ iff edge e_i crosses the returned cut. Then $\mathbf{E}\left[\widehat{e_i}\right] = (1 - \frac{2}{2^k})$, because the probability that e_i does not cross the cut, i.e., that e_i is either contained in V_0 or V_1 is $2/2^k$. The weight of the returned cut is the sum over the $\widehat{e_i}$, hence the proposition follows by linearity of expectation. $\square$

Derandomization of Algorithm $\mathrm{HyperCut}$

- Let $G = (V, E)$ be a hypergraph with n nodes and m edges.
- Let α be any word of length $s \leq n$

  Let $\mathrm{HyperCut}_\alpha$ be the algorithm that works like Algorithm $\mathrm{HyperCut}$, except that $r_1 \dots r_s$ is set equal to α.
- Let $\widehat{\mathrm{w}}_G(\alpha)$ be the weight of the random cut returned by Algorithm $\mathrm{HyperCut}_\alpha$ on input G.

$$\mathbf{E}\left[\widehat{\mathrm{w}}_G(\lambda)\right] \;=\; \mathbf{E}\left[\widehat{\mathrm{w}}_G\right] \;=\; (1 - 2/2^k)m\,.$$

- With probability of $1/2$ each, $\mathrm{HyperCut}_\alpha$
  - sets r_{s+1} to 0, i.e., works like $\mathrm{HyperCut}_{\alpha 0}$,
  - sets r_{s+1} to 1, i.e., works like $\mathrm{HyperCut}_{\alpha 1}$.

  In term of the expected weight of the returned cut, this means

$$\mathbf{E}\left[\widehat{\mathrm{w}}_G(\alpha)\right] \;=\; \frac{1}{2}\mathbf{E}\left[\widehat{\mathrm{w}}_G(\alpha 0)\right] \;+\; \frac{1}{2}\mathbf{E}\left[\widehat{\mathrm{w}}_G(\alpha 1)\right]\,,$$

  Hence at least one of the expected values on the right-hand side is at least as large as $\mathbf{E}\left[\widehat{\mathrm{w}}_G(\alpha)\right]$.

Derandomization of Algorithm HyperCut

Derandomized Algorithm HyperCut (Conditional expectation)

```
Input:   A hypergraph $G = (V, E)$          $(V = \{1, \dots, n\})$.
  For $s = 1, \dots, n$
    If   $\mathbf{E}[\widehat{\mathrm{w}}_G(r_1 \dots r_{s-1}0)] \geq \mathbf{E}[\widehat{\mathrm{w}}_G(r_1 \dots r_{s-1}1)]$   $(*)$
    then $r_s = 0$ else $r_s = 1$
  Let $V_0 = \{i : r_i = 0\}$.
  Let $V_1 = \{i : r_i = 1\}$.
Output:  The cut $(V_0, V_1)$.
```

For the weight w_G of the cut returned by Algorithm HyperCut on input G we have

$$(1 - \frac{2}{2^k})m = \mathbf{E}[\widehat{\mathrm{w}}_G(\lambda)] \leq \mathbf{E}[\widehat{\mathrm{w}}_G(r_1)] \leq \dots$$

$$\dots \leq \mathbf{E}[\widehat{\mathrm{w}}_G(r_1, \dots, r_{n-1})] \leq \mathbf{E}[\widehat{\mathrm{w}}_G(r_1, \dots, r_n)] = \mathrm{w}_G \; .$$

Derandomization of Algorithm $\mathrm{HyperCut}$

- Is the derandomized version of $\mathrm{HyperCut}$ efficient?

  How complex is it to evaluate Condition $(*)$?
- Let $\widehat{\mathrm{e}}_i(\alpha)$ be the indicator variable for the event that edge e_i crosses the cut returned by Algorithm $\mathrm{HyperCut}_\alpha$.
- By linearity of expectation, the expected value $\mathbf{E}\left[\widehat{\mathrm{w}}_G(r_1 \ldots r_{s-1} i)\right]$ is just the sum over the $\mathbf{E}\left[\widehat{\mathrm{e}}_i(r_1 \ldots r_{s-1} i)\right]$.

  The latter expected values are easy to compute.

  Partition any edge e_i into a set of small nodes of size at most s and a set of large nodes of size strictly larger than s.

  If there are two small nodes that have already been assigned to different sides of the cut, then the expected value is 1.

  Otherwise, all small nodes have been assigned to the same side. If there are k_ℓ large nodes, then the expected values is equal to the probability $1 - 1/2^{k_\ell}$ that at least one of the large nodes is assigned to the other side.

Derandomization of Algorithm $\mathrm{HyperCut}$

- Recall from the introduction, that random variables $X_1, \ldots, X_n$ are *k-wise independent* if every subset of k of theses variables is mutually independent.
- The verification of Algorithm $\mathrm{HyperCut}$ relied solely on the fact that the bits $r_1, \ldots, r_n$ have been chosen according to the uniform distribution on $\{0, 1\}$ and that they are *k-wise independent*.
- The construction of pairwise, i.e., 2-wise independent, random bits in connection with the derandomization of Algorithm Cut can be extended to an, albeit more involved, construction of k-wise independent random bits for arbitrarily large k.
- In what follows, we present another standard construction of k-wise independent random bits that uses algebraic methods.

Derandomization of Algorithm HyperCut

Proposition

Let p be any prime number and let the numbers $\widehat{a}$ and $\widehat{b}$ be chosen uniformly and independently from $\{0, \ldots, p-1\}$.
Then the p numbers

$$\begin{aligned}
\widehat{a}\,0 + \widehat{b} &\qquad \bmod p,\\
\widehat{a}\,1 + \widehat{b} &\qquad \bmod p,\\
&\vdots\\
\widehat{a}\,(p-1) + \widehat{b} &\qquad \bmod p
\end{aligned}$$

are uniformly distributed in $\{0, \ldots, p-1\}$ and are pairwise independent.

(Equivalently, the proposition could be stated in terms of the finite field with p elements.)

Derandomization of Algorithm HyperCut

Proof.

- The random variable $\widehat{a}i + \widehat{b}$ is uniformly distributed.
  If $\widehat{a}i$ is already determined, by choosing $\widehat{b}$ uniformly, $\widehat{a}i + \widehat{b}$ assumes any value in $\{0, \ldots, p-1\}$ with probability $\frac{1}{p}$.
- Any pair of random variables $\widehat{a}i + \widehat{b}$ and $\widehat{a}j + \widehat{b}$ where $i \neq j$ are mutually independent.
  Fix any numbers m_1, m_2 in $\{0, \ldots, p-1\}$. Then the system of equations
  $$\begin{aligned} ai + b &= m_1 \mod p, \\ aj + b &= m_2 \mod p \end{aligned}$$
  has a unique solution (a_0, b_0) with a_0, b_0 in $\{0, \ldots, p-1\}$.
  Hence
  $$\begin{aligned} &\mathrm{Prob}[\widehat{a}i + \widehat{b} \mod p = m_1, \widehat{a}j + \widehat{b} \mod p = m_2] \\ &\qquad = \mathrm{Prob}[\widehat{a} = a_0, \widehat{b} = b_0] = \frac{1}{p^2}\ . \end{aligned}$$

□

Derandomization of Algorithm HyperCut

Algorithm $\mathrm{Cut}_{\mathrm{pi}}$ (Pairwise independence)

```
Input:  A graph $G = (V, E)$ $(V = \{1, \dots, n\})$.
        The least prime $p \geq n$.

    Choose $\widehat{a}$ and $\widehat{b}$ uniformly and
                        independently in $\{0, \dots, p-1\}$ .
    For $i = 1, \dots, n$, let $r_i = (\widehat{a}i + \widehat{b}) \mod p$.
    Let $V_0 = \{i : r_i \text{ is even }\}$.
    Let $V_1 = \{i : r_i \text{ is odd }\}$.

Output: The cut $(V_0, V_1)$.
```

- The expected weight of the cut returned by Algorithm $\mathrm{Cut}_{\mathrm{pi}}$ is close to $\frac{m}{2}$.
- The analysis is basically the same as for Algorithm Cut. (We omit some minor technical details that relate to the fact that now the nodes are assigned to the two parts of the cut with probability $\frac{p-1}{2p}$ and $\frac{p+1}{2p}$.)

Derandomization of Algorithm HyperCut

Derandomized Algorithm Cut (Pairwise independence)

```
Input:  A graph $G = (V, E)$                    $(V = \{1, \dots, n\})$.
  Let $p$ be the least prime where $n \leq p$.
  For all pairs $(a, b)$ in $\{0, \dots, p-1\}$
      For $i = 1, \dots, n$, let $r_i = (ai + b) \mod p$.
      Let $V_0^{(a,b)} = \{i : r_i \text{ is even }\}$.
      Let $V_1^{(a,b)} = \{i : r_i \text{ is odd }\}$.
Output: A cut $(V_0^{(a,b)}, V_1^{(a,b)})$ of maximum weight.
```

- The algorithm returns a cut of weight at least close to $\frac{m}{2}$.
- By Bertrand's postulate the least prime $p > n$ is not larger than $2n$ and checking any number $n, \dots, 2n$ for primality requires time polynomial in $\log n$.
- A number in the range between n and $2n$ can be specified by at most $1 + \lceil \log n \rceil$ bits, hence in the algorithm at most $16n^2$ pairs (a, b) have to be considered.

Derandomization of Algorithm HyperCut

Proposition

Let p be any prime number and let the numbers $\widehat{a}_0, \ldots, \widehat{a}_{k-1}$ be chosen uniformly and independently from $\{0, \ldots, p-1\}$.
Let $r_1, \ldots, r_p$ be equal to the unique sequence of p numbers in $\{0, \ldots, p-1\}$ such that

$$\begin{aligned}
\widehat{a}_{k-1}0^{k-1} + \widehat{a}_{k-2}0^{k-2} + \quad \cdots \quad + \widehat{a}_0 &= r_1 \mod p,\\
\widehat{a}_{k-1}1^{k-1} + \widehat{a}_{k-2}1^{k-2} + \quad \cdots \quad + \widehat{a}_0 &= r_2 \mod p,\\
\widehat{a}_{k-1}2^{k-1} + \widehat{a}_{k-2}2^{k-2} + \quad \cdots \quad + \widehat{a}_0 &= r_3 \mod p,\\
&\vdots\\
\widehat{a}_{k-1}(p-2)^{k-1} + \quad \cdots \quad + \widehat{a}_0 &= r_{p-1} \mod p,\\
\widehat{a}_{k-1}(p-1)^{k-1} + \quad \cdots \quad + \widehat{a}_0 &= r_p \mod p.
\end{aligned}$$

Then the random numbers $r_1, \ldots, r_p$ are uniformly distributed in $\{0, \ldots, p-1\}$ and are k-wise independent.

Derandomization of Algorithm HyperCut

Proof.

Each r_i is uniformly distributed in $\{0, \ldots, p-1\}$ because assuming that the values of $\widehat{a}_1$ through $\widehat{a}_{k-1}$ have already been determined, the p equally probable choices for the value of $\widehat{a}_0$ will result in p pairwise distinct values for r_i.

In order to demonstrate that the r_i are k-wise independent, it suffices to show that for pairwise distinct numbers $i_1, \ldots, i_k$ in $\{1, \ldots, p\}$ and any numbers $b_1, \ldots, b_k$ in $\{0, \ldots, p-1\}$, we always have

$$\mathrm{Prob}[r_1 = b_1, \ldots, r_k = b_k] = \frac{1}{p^k} .$$

Proof (continued).

By definition of the r_j, this amounts to show that the sytem of equations

$$\begin{pmatrix} i_1^{k-1} & i_1^{k-2} & \ldots\ldots & i_1^0 \\ & & \vdots & \\ i_{k-1}^{k-1} & i_{k-1}^{k-2} & \ldots\ldots & i_{k-1}^0 \\ i_k^{k-1} & i_k^{k-2} & \ldots\ldots & i_k^0 \end{pmatrix} \begin{pmatrix} a_{k-1} \\ \vdots \\ a_1 \\ a_0 \end{pmatrix} = (b_1, \ldots, b_k)$$

has a unique solution (where now the numbers appearing in the system of equations are identified in the obvious way with elements of the field with p elements).

Observe that the matrix in this system of equations is a Vandermonde matrix.

Derandomization of Algorithm HyperCut

Proof (continued).

For the Vandermonde matrix M, its determinant is given by

$$\prod_{1 \le j < j' \le k} (i_{j'} - i_j) \ .$$

Since the i_j are pairwise distinct, the determinant differs from 0, hence the matrix is invertible, and one obtains for the a_j the unique solution $M^{-1}(b_1, \ldots, b_k)$ modulo p. □

Derandomization of Algorithm HyperCut

Algorithm HyperCut (k-wise independence)

```
Input:  A hypergraph $G = (V, E)$                    $(V = \{1, \ldots, n\})$.
  Let $p$ be the least prime where $n \leq p$.
  Choose $a_0, \ldots, a_{k-1}$ in $\{0, \ldots, p-1\}$
                                   uniformly and independently.
  For $i = 1, \ldots, n$, let
        $r_i = a_{k-1} i^{k-1} + a_{k-2} i^{k-2} + \ldots + a_0 \mod p$ .
  Let $V_0 = \{i : r_i \text{ is even }\}$.
  Let $V_1 = \{i : r_i \text{ is odd }\}$.
Output:  The cut $(V_0, V_1)$.
```

The algorithm returns a cut of expected weight of approximately $\left(1 - \frac{2}{2^k}\right) m$.

Derandomized Algorithm HyperCut (k-wise independence)

```
Input:  A hypergraph $G = (V, E)$        ($V = \{1, \ldots, n\}$).
 Let $p$ be the least prime where $n \leq p$.
 For all $k$-tuples $\vec{a} = (a_0, \ldots, a_{k-1})$ over $\{0, \ldots, p-1\}$
     For $i = 1, \ldots, n$, let

         $r_i = a_{k-1} i^{k-1} + a_{k-2} i^{k-2} + \ldots + a_0 \mod p$ .

     Let $V_0^{\vec{a}} = \{i : r_i \text{ is even }\}$.
     Let $V_1^{\vec{a}} = \{i : r_i \text{ is odd }\}$.
Output:  A cut $(V_0^{\vec{a}}, V_1^{\vec{a}})$ of maximum weight.
```

The algorithm returns a cut of weight that is approximately at least $\left(1 - \frac{2}{2^k}\right) m$.

Independent sets in hypergraphs

- A *hypergraph* is a pair (V, E) where V is the set of nodes and E is a set of subsets of V. The subsets in E are called *hyperedges*.
  A hypergraph is 3-*regular* if all its hyperedges contain exactly 3 nodes.

Definition

In a hypergrpaph $G = (V, E)$, a subset of V is called *independent* if it does not contain any edge in E.

Theorem

A 3-regular hypergraph with n nodes and $m \geq n/3$ hyperedges has an independent set U where

$$|U| \geq \frac{n\sqrt{n}}{3\sqrt{m}} \; .$$

Independent sets in hypergraphs

- The randomized Algorithm $\mathrm{IndependentSet}$ computes an independent set U of expected size of at least $|U| \geq \frac{n\sqrt{n}}{3\sqrt{m}}$.

Algorithm $\mathrm{IndependentSet}$

```
Input:  A hypergraph $G = (V, E)$                    $(V = \{1, \ldots, n\})$.
        A parameter $p$ with $0 \leq p \leq 1$.

   Choose random bits $r_1, \ldots, r_n$ by tosses of a biased coin
                                        such that $\mathrm{Prob}[r_i = 1] = p$.
   Let $T = \{i : r_i = 1\}$.
   Let $Y = \{\min e : e \in E$ and $e \subseteq T\}$
   $U = T \setminus Y$.

Output: The set $U$.
```

Independent sets in hypergraphs

- The set T is a candidate for an independent set.
- The set Y contains the least node from each hyperedge contained in T.
- The set U is independent.
- Y is a subset of T, hence $|U| = |T| - |Y|$.

  By linearity of expectation, we have

$$\mathbf{E}[|U|] = \mathbf{E}[|T|] - \mathbf{E}[|Y|]$$

- The expected size of T is $\mathbf{E}[|T|] = np$.

Independent sets in hypergraphs

- Let $D = \{e \in E : e \subseteq T\}$.
  Then $|Y| \leq |D|$, hence $\mathbf{E}[|Y|] \leq \mathbf{E}[|D|]$.
- For each e in E, let $\widehat{e}$ be the indicator variable for the event that e is contained in T.

  The size of D is just the sum over the random variables $\widehat{e}$.

  By linearity of expectation, the expected size of D is the sum of the expected values $\mathbf{E}[\widehat{e}]$.

  For any hyperedge e in E, by construction

$$\begin{aligned}\mathbf{E}[\widehat{e}] &= \mathrm{Prob}[e \subseteq T] \cdot 1 + \mathrm{Prob}[e \not\subseteq T] \cdot 0 \\ &= \mathrm{Prob}[e \subseteq T] = p^3 \ . \\ \mathbf{E}[|D|] &= m \cdot p^3 \ .\end{aligned}$$

- In summary, we have

$$\mathbf{E}[|U|] = \mathbf{E}[|T|] - \mathbf{E}[|Y|] \geq \mathbf{E}[|T|] - \mathbf{E}[|D|] = n \cdot p - m \cdot p^3 \ .$$

Independent sets in hypergraphs

- We have seen
$$\mathbf{E}\left[|U|\right] = \mathbf{E}\left[|T|\right] - \mathbf{E}\left[|Y|\right] \geq n \cdot p - m \cdot p^3 \ .$$
- For which $p \in [0,1]$ is the term $n \cdot p - m \cdot p^3$ maximum?
  Assuming $n \leq 3m$, the term is maximum for
$$p = \sqrt{\frac{n}{3m}} \ .$$
  Consider the function $p \mapsto n \cdot p - m \cdot p^3$.
  Its first derivative $n - 3mp^2$ is 0 for this value of p, while its second derivative is negative. For $p = 0$, the function is 0. For $p = 1$, the function evaluates to $n - m < 0$.
- For $p = \sqrt{\frac{n}{3m}}$, we then obtain
$$\begin{aligned}\mathbf{E}\left[|U|\right] &\geq n \cdot p - m \cdot p^3 \\ &= \left(\frac{1}{\sqrt{3}} - \frac{1}{3\sqrt{3}}\right)\frac{n\sqrt{n}}{\sqrt{m}} \geq \frac{1}{3}\frac{n\sqrt{n}}{\sqrt{m}}\end{aligned}$$

Independent sets in hypergraphs

- By the usual argument, the expected value of $|U|$ must be attained for some choice of the coint tosses in Algorithm $\mathrm{IndependentSet}$, hence any 3-regular graph with $n \leq 3m$ contains an independent set of the required size.
- Algorithm $\mathrm{IndependentSet}$ can be deradomized
  - by the method of conditional expectations, where it is arranged in the deradomized algorithm that the potential function $\mathbf{E}[|T|] - \mathbf{E}[|D|]$ does not decrease, hence is always at least as large as the initial value $\frac{1}{3}\frac{n\sqrt{n}}{\sqrt{m}}$.
    How difficult is it to determine the next random bit?
  - by the method of small sample spaces, using 3-wise independent random variables that attain the value 1 with probability p, then applying trivial derandomization.
    How can such 3-wise independent random variables be obtained (where the probability p for the value 1 may only be approximated)?

Sum-free subsets

Definition

A set Z of integers is *sum-free* if for all x, y, z in Z holds $x + y \neq z$.

Theorem

Every nonempty finite set Z of integers contains a sum-free subset T of size $|T| > \frac{|Z|}{3}$.

Proof.

Fix any set $Z = \{z_1 < \ldots < z_n\}$ of integers.

Let q be a prime number that is of the form $q = 3k + 2$ and that does not divide any number in Z (e.g., choose q strictly larger than the absolute values of all numbers in Z).

Sum-free subsets

Proof (continued).

Pick r in $N_q := \{1, \dots, q-1\}$ uniformly at random .

For $i = 1, \dots, n$, pick $d_i \in N_q$ such that $d_i \equiv z_i \cdot r \bmod q$.

Each d_i is uniformly distributed in N_q because by primality of q, for any fixed z the mapping $r \mapsto z \cdot r$ is a bijection of N_q.

Let $M = \{k+1, \dots, 2k+1\}$ und $T_r = \{z_i : d_i \in M\}$.

The set T_r ist sum-free, because

$$z_{i_1} + z_{i_2} = z_{i_3} \quad \text{implies} \quad d_{i_1} + d_{i_2} \equiv d_{i_3} \bmod q \ ,$$

where the latter cannot hold for $d_{i_1}, d_{i_2}, d_{i_3} \in M$. Furthermore,

$$\mathbf{E}[|T_r|] = \sum_{z_i \in Z} \mathbf{P}[d_i \in M] = |Z| \frac{|M|}{|N_q|} = |Z| \frac{k+1}{3k+1} > \frac{|Z|}{3} \ ,$$

hence for some choice of r we have $|T_r| > \frac{|Z|}{3}$. □

The isolating lemma

Definition

A *weight function* on a set X is a function $w\colon X \to \mathbb{R}$. A weight function w on X extends to subsets of X via $w(S) = \sum_{a \in S} w(a)$.

Theorem (Isolating Lemma)

Let $X = \{a_1, \ldots, a_m\}$ be a set of size $m > 0$ and let $\boldsymbol{F} = \{S_1, \ldots, S_k\}$ be a nonempty set of subsets of X. Let a weight function w on X be determined by choosing uniformly and independently for each x in X a value $w(x)$ in $\{1, \ldots, 2m\}$. Then with probability at least $1/2$, the minimal weight

$$w_{\min} = \min_{S \in \boldsymbol{F}} w(S)$$

is attained by a unique set in $\boldsymbol{F}$.

The isolating lemma

Proof.

Call a set $S \in \boldsymbol{F}$ a *minimum weight set* if $w(S)$ is equal to $w_{\min}$.

Furthermore, call a member a of X *ambiguous* if there are two minimum weight sets S^- and S^+ such that $a \notin S^-$ and $a \in S^+$.

Observe that there is a unique minimum weight set if and only if no member of X is ambiguous.

It suffices to show that any given member of X is ambiguous with probability at most $1/2m$, because then the probability that X has an ambiguous member at all is at most $1/2$.

Fix any a in X and let

$$\boldsymbol{F}^- = \{S \in \boldsymbol{F} \colon a \notin S\}, \qquad \boldsymbol{F}^+ = \{S \in \boldsymbol{F} \colon a \in S\} .$$

If one of the sets $\boldsymbol{F}^-$ or $\boldsymbol{F}^+$ is equal to $\boldsymbol{F}$, then a is never ambiguous, hence we can assume that both sets are nonempty.

The isolating lemma

Proof (continued).

Assuming that $w(x)$ has been determined for all x in $X \setminus \{a\}$, let

$$w^- = \min_{S \in \boldsymbol{F}^-} \sum_{x \in S} w(x) \qquad \text{and} \qquad w^+ = \min_{S \in \boldsymbol{F}^+} \sum_{x \in S \setminus \{a\}} w(x)\,,$$

and let $d = w^- - w^+$ (where d may be negative).

Then $w_{\mathrm{min}} = w^-$ or $w_{\mathrm{min}} = w^+ + w(a)$, and both equations hold simultaneously if and only if $w(a) = d$.

In case $w(a) < d$, all minimum weight sets are in $\boldsymbol{F}^+$.
In case $w(a) > d$, all minimum weight sets are in $\boldsymbol{F}^-$.
In both cases, a cannot be ambiguous by definition of $\boldsymbol{F}^-$ and $\boldsymbol{F}^+$.

Hence a can only be ambiguous in case $w(a) = d$, where the latter has probability at most $1/2m$ because $w(a)$ is chosen uniformly and independently of the other weights from a set of size $2m$. $\square$

Planar graphs

- In what follows, the term graph refers to an undirected graph that is simple, i.e., does neither have loops nor multiple edges.
- A graph is planar if the graph can be embedded into the plane without crossing edges.
  (We will use the notion of an embedding and other notation such as face of an embedding without defining them and refer to Diestel's monograph *Graph Theory*.)
- Furthermore, we will assume certain properties of planar graphs and their embeddings that are intuitively clear without giving formal proofs.
  For example, given an embedding of a planar graph G into the plane, for any cycle (i.e., closed path) in G, part of the plane is inside and part is outside the cycle and no face of the embedding can intersect both parts of the cycle.

Crossing numbers

Theorem (Euler's formula)

Let G be a connected planar graph with n nodes and m edges such that G has an embedding in the plance with f faces (including the outer face). Then holds $f - m + n = 2$.

Sketch of proof.

Use induction on the number of cycles in G.

A graph without cycles is a tree, hence has $m + 1$ edges and a single face and Euler's formula holds.

For a plane graph with $k > 0$ cycles, removing an edge from a cycle decreases both the number of edges and the number of faces by 1, while for the resulting graph Euler's formula holds by the induction hypothesis.

Corollary

All embeddings of a planar graph have the same number of faces.

Crossing numbers

Proposition

Let G be a planar graph with n nodes and $m > 1$ edges.
Then holds $m \leq 3n - 6$.

Proof.

It suffices to prove the assertion for connected G because any nonconnected planar graph can be transformed into a connected planar graph by adding edges.

We can assume $m \geq 3$ because G is simple, hence for $m = 2$ we have $n = 3$ and the assertion is true.

Let f_i be the number of faces of G that are bounded by i edges where edges that are "surrounded" by a face are counted twice.

That is, we view an edge as having two "sides" and when counting the number of edges that bound a face in fact we are counting the sides of edges that bound the face.

Crossing numbers

Proof (continued).

The graph G is simple and $m \geq 3$, hence $f_1 = f_2 = 0$.

By counting the edges in two different ways, we obtain

$$\begin{aligned} f &= \ f_3 + \ f_4 + \ f_5 + \dots \ , \\ 2m &= 3f_3 + 4f_4 + 5f_5 + \dots \ , \end{aligned}$$

hence we have $0 \leq 2m - 3f$ and Euler's formula yields

$$0 \leq 2m - 3f = 2m - 3(m - n + 2) = 3n - 6 - m. \qquad \square$$

Crossing numbers

An embedding of a graph (V, E) into the plane consists of

- $|V|$ pairwise distinct points of the plane, where we idenify these points with the nodes in V,
- a curve with endpoints u and v for each edge $\{u, v\}$ in E, where we identify these curves with the edges in E.

We want to define a notion of crossing edges and, for a given graph, of embedding with a minimum number of crossings.

How should we count crossings of an embedding?
When are two crossings of an embedding identical?

An embedding with a minimum number of crossings shouldn't

- use "tricks" such as an edge intersecting a node that is not an endpoint of the edge,
- have unnecessary crossings such as in the case where two edges cross several times (see below).

Thus we want to define the notion of crossing and to count crossings in a way such that the number of crossings is increased by using such tricks and by unnecessary crossings.

Crossing numbers

Definition

Two edges of an embedding *cross* in a point if this point belongs to both edges but is not a common endpoint.

With an ordering of nodes understood, a *crossing* of an embedding is a set of two contiguous subcurves of two distinct edges such that

- the two edges cross in a point that is a common endpoint of the two subcurves,
- the other endpoints of the subcurves are the two lesser nodes of these edges, respectively.

In the simple case where for every pair of edges there is at most one crossing, we identify crossings with pair of edges.

Crossing numbers

Definition

The *crossing number* $\mathrm{cr}(G)$ of a graph G is the least k such that G has an embedding into the plane with at most k crossings. An embedding of a graph G into the plane is *minimum* if the embedding has at most $\mathrm{cr}(G)$ crossings.

For a minimum embedding, the following assertions are true.

(i) No edge can cross itself.
(ii) Two edges that are incident to a common node cannot cross.
(iii) Two edges cannot cross twice or more.
(iv) Two edges cannot cross in an endpoint of one of the edges.

In any of the situations described in (i) through (iv), the given embedding can be transformed into an embedding of the same graph with strictly less crossings, hence for a minimum embedding, these situations cannot occur.

Crossing numbers

Proposition

For any graph G holds $\mathrm{cr}(G) \geq m - 3n + 6$.

Proof.

Transform a minimum embedding of a graph G into an embedding of a graph G' where each point where two edges of G cross is replaced by a new node, and the edges of the new graph correspond to minimum nonzero subcurves of the old edges that have two nodes of G' as endpoints.

By (i) through (iv), the graph G' is simple and has $n + \mathrm{cr}(G)$ nodes and $m + 2\mathrm{cr}(G)$ edges since every new node is distinct from the old nodes and has degree 4. The proposition shown above then yields

$$3(n + \mathrm{cr}(G)) - 6 \geq m + 2\mathrm{cr}(G), \qquad \text{hence}$$

$$\mathrm{cr}(G) \geq m - 3n + 6.$$

Crossing numbers

Theorem

Let G be a graph with n nodes and m edges and assume $m \geq 4n$. Then the crossing number of G is bounded from below as follows

$$\frac{1}{64}\frac{m^3}{n^2} \leq \mathrm{cr}(G)\ .$$

Proof (Aigner and Ziegler, *Proofs from the book*, Chapter 32).

Let p be a real number between 0 and 1 to be determined later.

Determine a set V_p of nodes of G by tossing a coin for each node of G such that any single node is put into V_p with probability p.

Let G_p be the subgraph of G that is induced by the set V_P.

Crossing numbers

Proof.

Let n_p and m_p be the number of nodes and edges of the graph G_p.

Fix a minimum embedding of G and consider the embedding of G_p that is obtained by restricting the given embedding of G to G_p.

Let x_p be the number of crossings in this embedding of G_p.

Observe that $x_p \geq \mathrm{cr}(G_p)$, hence by the proposition above we have

$$\mathbf{E}[x_p - m_p + 3n_p] \geq 0$$

As usual, we have $\mathbf{E}[n_p] = pn$ and $\mathbf{E}[m_p] = p^2m$. Furthermore,

$$\mathbf{E}[x_p] = p^4\mathrm{cr}(G)\,.$$

because a crossing in the given embedding of G is also in the new embedding if and only if all 4 pairwise distinct endpoints of two corresponding crossing edges are in G_p.

Crossing numbers

Proof (continued).

By linearity of expectation, the preceding discussion yields

$$p^4 \mathrm{cr}(G) - p^2 m + 3pn = \mathbf{E}[x_p] - \mathbf{E}[m_p] + 3\mathbf{E}[x_p] \geq 0, \text{ hence}$$

$$\mathrm{cr}(G) \geq \frac{p^2 m - 3pn}{p^4} .$$

Now let $p = \frac{4n}{m}$. By assumption $p \leq 1$, and we obtain

$$\mathrm{cr}(G) \geq \frac{pm - 3n}{p^3} = \frac{4n - 3n}{p^3} = \frac{1}{64} \frac{m^3}{n^2} . \qquad \square$$

Points on a circle

Proposition

If n points on a circle are chosen uniformly and mutually independently, then with probability $1 - \frac{2n}{2^n}$ the convex hull of these points contains the center of the circle.

Proof.

Fix any circle. For a point x on the circle let the mirror point of x be the unique point that differs from x and is on the line through x and the center of the circle.

If a set P of n points is determined by a chance experiment where

(i) n points on the circle are chosen uniformly and mutually independent,

(ii) for each such point a fair coin is tossed in order to decide whether the point or its mirror point is put into P,

then the points in P are actually chosen uniformly and mutually independently.

Points on a circle

Proof (continued).

Let X be the set of all points chosen in Step i together with their mirror points (we can assume that all these points are pairwise distinct because with probability 1 this is indeed the case).

Call a subset of X a candidate set if the set contains for each point in X either the point or its mirror point.

There are exactly 2^n candidate sets.

The set P is uniformly distributed in the set of candidate sets.

Call a candidate set one-sided if there is a line through the center of the circle such that the candidate set is contained in one of the half-planes determined by the line.

The center of the circle is not in the convex hull of the points in P if and only if P is one-sided.

There are exactly $2n$ candidate sets that are one-sided, hence the probability that P is one-sided is $\frac{2n}{2^n}$. □

Byzantine agreement

The Byzantine agreement problem

- In the Byzantine agreement problem, n processors (or, say, Byzantine generals, ...) communicate with each other in order to reach an agreement on a binary value b.
- There are bad processors that may collaborate with each other in order to prevent an admissible agreement.
  At most a fraction of $1/8$ of all processors are bad.
- Each processor has an initial binary value.
  The agreement must reflect to a certain extent the majority among the initial values. More precisely, the processors must reach an agreement that is *admissible* in the following sense.
  (i) All good processors must agree on the same value b.
  (ii) In case all the good processors have the same initial value, then b must be equal to this value.

Byzantine agreement

The rules for the Byzantine agreement problem

- The communication is done in rounds.
- At the beginning of each round, each processor sends messages to all other processors.
- Processor i sees only the messages sent to Processor i. The messages sent by a processor to different receivers might differ.
- Before each round, the bad processors may agree on an arbitrarily complex pattern of messages for this round.
- At the beginning, the good processors know neither the bad processors nor their strategy.

The rationale for supposing collaboration among the bad processors is that a protocol that succeeds against concerted attacks is likely to succeed in the presence of random or unrelated faults.

Byzantine agreement

Protocol ByzantineAgreement (Processor i)

```
Input: A binary value $b_i$.
    Fix constants $t_0 = \frac{5}{8}n$ and $t_1 = \frac{6}{8}n$ and let $v_i(1) = b_i$.
    For rounds $s = 1, 2, \dots$ do the following.
        Send $v_i(s)$ to all other processors.
        For all $j \neq i$, receive $v_j(s)$ from Processor $j$.
        For $l = 0, 1$, let $c_l = |\{j : v_j(s) = l\}|$.
        If $c_0 \geq c_1$  then  $u(s) = 0$ and $c(s) = c_0$,
                           else  $u(s) = 1$ and $c(s) = c_1$.
        (The most frequent value among $v_1(s), \dots, v_n(s)$ is $u(s)$
         and $c(s)$ is its count.)
        Obtain $\tau(s) \in \{0, 1\}$ by tossing a fair coin.
        (The random bit $\tau(s)$ is the same for all processors.)
        If $c(s) \geq t_{\tau(s)}$, then $v_i(s+1) = u(s)$ else $v_i(s+1) = 0$.
        If $c(s) \geq \frac{7}{8}n$, then assume an agreement on $u(s)$ and
                                   let $v_i(s+1) = v_i(s+2) = \dots = u(s)$.
```

Byzantine agreement

Proposition

Let a group of n processors communicate where all but at most $n/8$ processors obey the Protocol ByzantineAgreement*. Then an admissible agreement is reached with probability 1 and in an expected number of rounds that is constant.*

Remark.

For deterministic protocols to solve the Byzantine agreement problem, matching lower and upper bounds are known:

under the given assumptions, any deterministic protocol will require at least $n/8 + 1$ rounds in worst case,

there is a deterministic protocol that reaches an agreement in at most $n/8 + 1$ rounds.

(for references see the corresponding section in the monograph by Motwani and Raghavan).

The proposition is immediate from the three following claims.

Byzantine agreement

Claim I

In case all good processors have the same initial value, an agreement on this value is reached at the end of the first round.

Proof of Claim I.

In case all good processors have the same initial value b, then in the first round all good processors send b to all other processors. So we have for each good processor $u(1) = b$ and $c(1) = c_b \geq \frac{7}{8}n$, hence the processor assumes an agreement on b. □

Byzantine agreement

Claim II

Let s be minimum such that at the end of round s some good processor assumes an agreement. Then all good processors assume an agreement on the same value at the end of round $s+1$.

Proof of Claim II.

Pick some processor that assumes an agreement on b at the end of round s. In round s, by minimality of s we have for this processor

$$c_b = c(s) \geq 7/8\ n\ ,$$

hence at least $\frac{6}{8}n$ good processors must have sent b.

Again by minimality of s, in round $s+1$ then each good processor will send b, assuming either an agreement on b or no agreement.

Consequently, all good processors will have reached an agreement on b at the end of round $s+1$. □

Byzantine agreement

Claim III

Suppose no good processor assumes an agreement during any of the rounds 1 through $s-1$. Then with probability at least $1/2$ some processor assumes an agreement in round $s+1$.

Proof of Claim III.

If some good processor assumes an agreement at the end of round s, then we are done by Claim II, so we can assume otherwise.

Then it suffices to show that with probability at least $1/2$ in round $s+1$ all good processors send the same bit.

Byzantine agreement

Proof of Claim III (continued).

Let k_0 and k_1 be the number of good processors that send 0 and 1, respectively, during round s and let $k = \max\{k_0, k_1\}$.

We distinguish two cases and in both cases consider round s.

Case A: $k < 5/8\ n$.

We have $k_0, k_1 \leq k < 5/8\ n$.

Hence no matter what messages the at most $n/8$ bad processors send, each processor will receive strictly less than $6/8\ n$ messages containing the same bit.

But with probabity $1/2$ the threshold $t_{\tau(s)}$ will be equal to $t_1 = 6/8\ n$, in which case all good processors send 0 in round $s + 1$.

Byzantine agreement

Proof of Claim III (continued).

Case B: $k \geq 5/8\ n$.

Choose b such that $k = k_b$.

No matter what messages the at most $n/8$ bad processors send, each processor will receive at least $5/8\ n$ messages containing bit b.

But with probabity $1/2$ the threshold $t_{\tau(s)}$ will be equal to $t_0 = 5/8\ n$, in which case all good processors send b in round $s+1$.

□

Stable marriages

Preference lists

Suppose there are n women $F_1, \dots, F_n$ and n men $M_1, \dots, M_n$ such that each person has a prefence list of the persons of the opposite sex, more precisely, there are strict orderings

$$<_{F_1}, \dots, <_{F_n}, <_{M_1}, \dots, <_{M_n}$$

on the set $\{1, \dots, n\}$ such that women F_j prefers man M_k over man M_l if and only if $k <_{F_j} l$ (and similarly for the preferences of men).

Stable marriages

Marriages

Fix a number n and sets $\{F_1, \ldots, F_n\}$ and $\{M_1, \ldots, M_n\}$ of size n.

Then a *marriage* is a bijection π of the set $\{1, \ldots, n\}$

We identify a marriage π with the binary relation

$$H = \{(F_1, M_{\pi(1)}), \ldots, (F_n, M_{\pi(n)})\} .$$

Definition

A marriage H is *unstable* if there are couples (F_i, M_k) and (F_j, M_l) in H such that

$$l <_{F_i} k \qquad \text{and} \qquad i <_{M_l} j .$$

In this situation, the pair of the couples (F_i, M_k) and (F_j, M_l) is called *dissatisfied*. A marriage is *stable* if it is not unstable.

Question   Is there always a stable marriage?

Stable marriages

Theorem (Marriage theorem)

Let $<_{F_1}, \dots, <_{F_n}, <_{M_1}, \dots, <_{M_n}$ be strict orderings on $\{1, \dots, n\}$. Then there is a stable marriage with respect to these orderings.

Proof.

The proposal algorithm below computes a stable marriage.

Remark

The marriage theorem is false in general if the distinction between women and men is dropped, i.e., if one considers an even number $2n$ of persons where

> each person has a preference list of all the other persons and one asks for a partition of the set of persons into sets of size 2 that is stable in a sense similar to the definition above.

The proposal algorithm

Algorithm Proposal (also referred to as proposal algorithm)

```
Input: Strict orderings $<_{F_1}, \ldots, <_{F_n}, <_{M_1}, \ldots, <_{M_n}$ on $\{1, \ldots, n\}$.
  Let $H = \emptyset$.
  While there is an unmarried man.
      Let $k$ be mimimum such that $M_k$ is unmarried.
      Let $i = \min_{<_{M_k}}\{j \colon M_k$ has never proposed to $F_j$ before$\}$.
      If $F_i$ is currently unmarried, then let $H = H \cup \{(F_i, M_k)\}$.
      If $F_i$ is currently married to $M_l$ and $M_k <_{F_i} M_l$,
                    then let $H = (H \setminus \{(F_i, M_l)\}) \cup \{(F_i, M_k)\}$.
Output: a stable marriage $H$.
```

After the least unmarried man M_k has been fixed, he proposes in consecutive iterations of the while loop to all women that have not already rejected or divorced him, in decreasing order of desirability. Eventually M_k is married to the first such woman F_i where F_i is unmarried or is married to a man M_l that in F_i's view is less desirable than M_k (and M_l becomes unmarried again).

The proposal algorithm

Proposition (Verification of the proposal algorithm)

The proposal algorithm terminates and outputs a stable marriage.

Proof.

In each iteration of the while loop, M_k proposes to some woman, i.e., the value i is always defined because

> if there is an unmarried man, then there must also be an unmarried woman, and
> M_k has not already proposed to any unmarried woman, otherwise she had accepted and had then stayed married.

Each man proposes at most once to each woman, thus the while loop is iterated only finitely often.

When the algorithm terminates there is no unmarried man, hence the computed set H is a marriage.

The proposal algorithm

Proof (continued).

Now assume for a proof by contradiction that the computed marriage H is not stable.
Then H contains dissatisfied couples, i.e., contains couples

$$(F_i, M_k) \text{ and } (F_j, M_l) \quad \text{such that} \quad l <_{F_i} k \text{ and } i <_{M_l} j\ .$$

The sequence of partners a woman marries during the execution of the algorithm is strictly increasing in desirability, hence M_l, who in the view of F_i is more desirable than F_i's final partner, is more desirable than all partners of F_i .

Thus M_l never proposes to F_i, otherwise she would marry him.

This contradicts the fact that M_l is married to F_j, hence has proposed to F_j, while M_l surely proposes to F_i before he proposes the first time to F_j. □

The set of all stable marriages

For the following discussion, we fix any instance of the stable marriage problem and consider the set of all stable marriages for this instance.
We define a relation $\preceq_M$ on the set of stable marriages where

$$H \preceq_M H'$$

holds for two stable marriages H and H' if and only if every man has with H either the same wife as with H' or a wife that he prefers to his wife with H' (that is, with H all men do at least as good as with H').

Proposition

The relation $\preceq_M$ is a partial ordering on the set of all stable marriages (i.e., $\preceq_M$ is reflexive, transitive, and antisymmetric).

The set of all stable marriages

Men's best stable marriage

Is there a stable marriage that is a *least* with respect to the $\preceq_M$ relation, i.e., a stable marriage where every man does at least as good as with any other stable marriage?

We say a woman is *possible* for a man, if there is some stable marriage where the man is married to this woman.

Proposition

A stable marriage H is least with respect to the relation $\preceq_M$ if and only if with H every man is married to the most desirable woman among all women who are possible for him.

The set of all stable marriages

The proposal algorithm favors the preferences of men.

Theorem

The output of the proposal algorithm is a stable marriage that is least with respect to the $\preceq_M$ relation.

Proof.

We proof by induction over the proposals made during a run of the proposal algorithm that whenever a woman rejects or divorces a man, then the woman is not possible for this man.

The theorem then follows because the proposals a man makes are chosen without repetition in order of his preference list, hence every man ends up being married to the most desirable woman that did neither reject nor divorce him. By the proposition above, this means that the ouput is least with respect to the $\preceq_M$ relation.

The set of all stable marriages

Proof (continued).

In the induction step, assume that man M_k proposes to woman F_i, who is currently married to M_l.

Case I: M_k is rejected (i.e., F_i prefers M_l to M_k).

There cannot be a stable marriage where M_k is married to F_i.

Assuming otherwise, in such a marriage, M_l could not be married to F_i, while by induction all women more desirable to him than F_i are not even possible for him, so he must be married to a woman F_j who is strictly less desirable to him than F_i.

Consequently, the couples (F_i, M_k) and (F_j, M_l) are dissatisfied, thus contradicting the assumption that the marriage is stable.

The set of all stable marriages

Proof (continued).

Case II: F_i divorces M_l (i.e., F_i prefers M_k to M_l).

There cannot be a stable marriage where M_l is married to F_i.

Assuming otherwise, in such a marriage, M_k could not be married to F_i, while by induction all women more desirable to him than F_i are not even possible, so he must be married to a woman F_j who is strictly less desirable to him than F_i.

Consequently, the couples (F_i, M_l) and (F_j, M_k) are dissatisfied, thus contradicting the assumption that the marriage is stable.

The set of all stable marriages

Remark

The least stable marriage with respect to the relation $\preceq_M$ is the greatest stable marriage with respect to a relation $\preceq_F$ defined like $\preceq_M$ with roles of men and women interchanged.

So the proposal algorithm yields a stable marriage where every man is married to the most preferrable woman possible, whereas every woman is married to the least desirable man possible.

Average-case complexity of the proposal algorithm.

How many proposals are made by the proposal algorithm on average on inputs of order n, where the average is taken over all choices of $2n$ strict orderings $<_{F_1}, \dots, <_{F_n}$, $<_{M_1}, \dots, <_{M_n}$?

The average number of proposals is the same as the expected number of proposals when we choose the $2n$ strict orderings uniformly at random from the set of all strict orderings on $\{1, \dots, n\}$.

We will derive an upper bound on the number of proposals, hence it suffices to consider the case where $<_{F_1}, \dots, <_{F_n}$ are arbitrary but fixed and the strict orderings $<_{M_1}, \dots, <_{M_n}$ are chosen at random.

The average number of proposals

Theorem (Average-case complexity of the proposal algorithm.)

The average number of proposals made by the proposal algorithm on inputs of order n is at most $\mathrm{O}(n \log n)$.

The theorem is an immediate consequence of the following lemma.

Lemma

Let a nonzero natural number n and strict orderings $<_{F_1}, \dots, <_{F_n}$ on $\{1, \dots, n\}$ be given, and let strict orderings $<_{M_1}, \dots, <_{M_n}$ on $\{1, \dots, n\}$ be chosen uniformly and independently at random. Then the expected number of proposals made by the proposal algorithm is at most $\mathrm{O}(n \log n)$.

Proof.

The proof of the lemma is done in three steps.

The average number of proposals

Proof (continued).

Step 1 (replacing random inputs by a random choice).

By the Principle of Deferred Decisions, the probabilities for the various possible runs of proposal algorithm and, in particular, the expected number of proposals remains the same if

> instead of choosing $<_{M_1}, \ldots, <_{M_n}$ in advance and always choosing for the next proposal the woman F_i that is most desirable to M_k and has not already been proposed to by M_k, $\boldsymbol{M_k}$ **chooses a woman** $\boldsymbol{F_i}$ **uniformly at random from all women to whom he has not already proposed.**

Observe that when F_i is determined in an iteration of the proposal algorithm, the restriction of $<_{M_k}$ to the set of all women to whom M_k has not already proposed has not been relevant before, thus by choice of $<_{M_k}$ all possible strict orderings on the latter set of women are equally probable.

The average number of proposals

Proof (continued).

Step 2 (transition to the amnesiac version).

Next consider an *amnesiac version* of the proposal algorithm where in each iteration M_k chooses the next woman to propose to

> not uniformly at random from the set of all women to whom M_k has not proposed before
> **but uniformly at random from the set of all women.**

If an unmarried man proposed another time to the same woman, he will be rejected because this woman must have rejected or divorced him in the past and thus is now married to a more desirable man.

Consequently, the amnesiac and the standard version of the proposal algorithm differ only in so far as with the amnesiac version there may occur additional proposals that are rejected anyway.

In summary, the expected number of proposals for the amnesiac version is at least as large as for the standard version.

The average number of proposals

Proof (continued).

Step 3 (anonymizing men).

With the amnesiac version of the proposal algorithm all men can be viewed as behaving the same, hence for each iteration

> instead of asking which man M_k proposes to which woman and whether he is rejected or not,
> **we may simply ask whether a married or an unmarried woman is chosen.**

The amnesiac version makes exactly one proposal per iteration and the algorithm terminates when the last unmarried woman is chosen.

Consequently the amnesiac version of the proposal algorithm can be analyzed like the coupon collector's problem with n types of coupons (where choosing an unmarried women corresponds to obtaining a new type of coupon), and the theorem follows. $\square$

Excursus on Harmonic numbers

Definition

Then *n*th *harmonic number* is $\mathrm{H}_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}$.

Proposition

For all n we have for the nth harmonic number H_n

$$\ln n \leq \mathrm{H}_n \leq \ln n + 1 \ .$$

Proof.

Let h be the step-like function on the nonnegative reals that attains the value $1/i$ in the interval $[i-1, i)$.

The integral over h from 1 to n is equal to $\mathrm{H}_n - 1$.

Proof (continued).

For $x \geq 1$, the function h is bounded from above by $g^+ \colon x \mapsto 1/x$ and from below by $g^- \colon x \mapsto 1/(x+1)$.

The integral over g^+ from 1 to n is $\ln n$.

The integral over g^- from 1 to n is $\ln(n+1) - \ln 2$.

In summary, we have for all $n \geq 1$

$$\ln n - \ln 2 \leq \ln(n+1) - \ln 2 \leq \mathrm{H}_n - 1 \leq \ln n\ . \qquad \square$$

Excursus on the geometric distribution

Definition (Geometric distribution)

The *geometric distribution with probability* p is the distribution on the set of natural numbers where

$$\mathrm{Prob}[0] = 0 \quad \text{and} \quad \mathrm{Prob}[i] = (1-p)^{i-1}p \text{ for all } i > 0 \,.$$

Remark

Consider a not necessarily fair coin with probability p for heads. The random variable that is equal to the number of times we have to flip the coin in order to obtain outcome heads at least once has a geometric distribution with probability p.

Proposition (Geometric distribution)

Let X be a random variable that is geometrically distributed with nonzero probability p. Then the expected values of X is $1/p$.

Proof.

Let $q = 1 - p$, hence $\mathrm{Prob}[X = i] = q^{i-1}p$ for $i \neq 0$.

The expectation of X is $\mathbf{E}[X] = \sum_{i=1}^{\infty} iq^{i-1}p = p\sum_{i=1}^{\infty} iq^{i-1}$.

For any real x where $-1 < x < 1$, we have

$$1 + x + x^2 + \cdots = \frac{1}{1-x}\,, \quad \text{hence} \quad 1 + 2x + 3x^2 + \cdots = \frac{1}{(1-x)^2}$$

by differentiating both sides of the former equation (recall that for such x the former power series can be differentiated term by term).

Substituting q for x, we obtain $\mathbf{E}[X] = 1/p$. □

The coupon collector's problem

Consider the random experiment where in every round $s = 1, 2, \ldots$ one of n type of coupons is chosen uniformly at random.

Let X be equal to the minimum s such that all n types of coupons have occurred in rounds 1 through s.

The coupon collector's problem asks for the probabilities of the form $\mathrm{Prob}[X = m]$ and for the expectation of X.

Proposition (Expected waiting time of the coupon collector)

In the coupon collector's problem with n types of coupons, the expected number of rounds before all types of coupons have occured at least once is equal to $n\mathrm{H}_n$, hence is at most $\mathrm{O}(n \log n)$.

Excursus on the coupon collector's problem

Proof.

As before, let X be minimum such that all types of coupons occur in round 1 through X.

Let the ith new outcome occur in round s_i (hence $s_1 = 1$, $s_n = X$).

For $i = 1, \ldots, n$, let $I_i = \{s_{i-1} + 1, \ldots, s_i\}$, where we let $s_0 = 0$.

Let $X_i = |I_i| = s_i - s_{i-1}$ and observe $X = X_1 + \cdots + X_n$.

The probability for a new outcome in round $s \in I_i$ is $p_i = \frac{n-i+1}{n}$.

That is, X_i has a geometrical distribution with probability p_i, hence

$$\mathbf{E}[X] = \sum_{i=1}^{n} \mathbf{E}[X_i] = \sum_{i=1}^{n} \frac{1}{p_i} = \sum_{i=1}^{n} \frac{n}{n-i+1} = n \sum_{i=1}^{n} \frac{1}{i} = n\mathrm{H}_n$$

The proposition follows by the upper bound $\ln n + 1$ for the harmonic number H_n. □

Yao's Minimax Principle

Complexity of algorithms

The complexity of an algorithm is usually measured with respect to the *size* of the input, where size may for example refer to

- the length of a binary word describing the input,
- the number of nodes in an input graph,
- the number of elements in an input list.

Furthermore, one may be interested in

- the worst-case complexity, i.e., the complexity on the worst input of a given size, or in
- the average-case complexity, i.e., the average complexity over all inputs of the same size.

Finally, the considered type of complexity may be

- the running time, the number of comparisons made, ...

Yao’s Minimax Principle

Complexity of problems: lower and upper bounds

The complexity of a problem is specified by lower and upper bounds on the complexity of algorithms that solve the problem:

lower bounds are obtained by proving (e.g., using combinatorial arguments) that no algorithm can have a complexity smaller than the lower bound under consideration,
upper bounds are usually obtained by constructing an algorithm for the given problem that is correct and has complexity of at most the upper bound under consideration.

The aim is to find matching lower and upper bounds

If one can derive matching lower and upper bounds for a problem, then the algorithm that yields the upper bound has minimum complexity. Similarly, close lower and upper bounds imply that the complexity of the corresponding algorithm is close to minimum.

Yao's Minimax Principle

Las Vegas and Monte Carlo algorithms

A randomized algorithm is a

Las Vegas algorithm if the algorithm is always correct,
Monte Carlo algorithm if the algorithm may be incorrect.

For a Las Vegas algorithm, the outcomes of the underlying source of randomness have no influence on the correctness of the result, however they may influence the running time or other parameters.

Examples

A typical example of a Las Vegas algorithm is randomized quicksort (i.e., the pivot elements are chosen uniformly at random).

A typical example of a Monte Carlo algorithm is integration by randomized sampling (i.e., in order to obtain an estimate for the area of a geometric figure inside the unit square, pick points in the unit square independently and uniformly at random and let the area be equal to the fraction of points inside the figure).

Yao's Minimax Principle

Worst-case complexity of Las Vegas algorithms

In what follows, we consider Las Vegas algorithms and their expected complexity in worst case, that is, the expected complexity on the worst input of any given size.

On first sight, it appears to be hard to obtain good lower bounds on the expected complexity of a Las Vegas algorithm in worst case.

Yao's Minimax Priniciple, which is discussed in detail below, asserts that for any problem and

> for any fixed probability distribution on the inputs of some given size, any lower bound on the average-case complexity of deterministic algorithms (where the algorithm may depend on the probability distribution)
> is also a lower bound on the expected complexity in worst case of Las Vegas algorithms.

Yao's Minimax Principle

Inputs, programs, and the cost function

We consider problems with a notion of size for the inputs such that for each size n there are

- a finite set $\mathcal{I}_n$ of inputs of this size,
- a finite set $\mathcal{A}_n$ of all correct deterministic algorithms for inputs in $\mathcal{I}_n$,
- a cost function $k_n \colon \mathcal{A} \times \mathcal{I} \to \mathbb{N}$,

where $k_n(A, I)$ is equal to the cost of applying algorithm A to input I.

This rather abstract setup is particularly suited for black-box algorithms like randomized quicksort, which is considered below.

Definition

An algorithm for sorting is a *black-box algorithm* if an input list $\mathrm{x}_1, \dots, \mathrm{x}_n$ can only be accessed by queries of the form "$\mathrm{x}_i < \mathrm{x}_j$?".

Yao's Minimax Principle

Example Randomized quicksort

Randomized quicksort is a black-box algorithm that sorts input lists of pairwise distinct items with respect to a strict linear ordering $<$.

Randomized quicksort works like deterministic quicksort but in the recursive calls the pivot element for splitting the current input list is chosen uniformly at random from the input list.

The size of an input is equal to the number of items in the list and

- $\mathcal{A}_n$ is the set of all deterministic black-box algorithms that correctly sort lists of n items,
- $\mathcal{I}_n$ can be assumed to be equal to the set of all permutations of the set $\{1, \ldots, n\}$ (because of the black-box access),
- $k(A, I)$ is equal to the number of queries of the form "$\mathrm{x}_i < \mathrm{x}_j$?" that algorithm A asks on input I.

The set $\mathcal{A}_n$ can be chosen to be finite by considering only algorithms that never ask the same query twice and by identifying an algorithm with its behaviour as discussed below.

Yao's Minimax Principle

Representing Las Vegas algorithms by distributions

Yao's Minimax Principle is not formulated in terms of Las Vegas algorithms but is about situations where for each n, a deterministic algorithm is chosen randomly from the finite set $\mathcal{A}_n$ of all correct deterministic algorithms according to some probability distribution.

We will apply the principle to Las Vegas algorithms that induce for each n a probability distribution σ_n on the set $\mathcal{A}_n$ of all correct deterministic algorithms in such a way that all relevant features of the algorithm are represented.

Here for each n, the induced probability distribution σ_n determines for each A in $\mathcal{A}_n$ the probability $\mathrm{Prob}_{\sigma_n}[A]$ that on inputs of size n the original Las Vegas algorithm behaves like algorithm A.

As an example, we argue next that a representation by probability distributions on the sets $\mathcal{A}_n$ is possible for all black-box sorting algorithms of Las-Vegas type.

Yao's Minimax Principle

Representing Las Vegas algorithms by distributions

When asking only for the expected number of comparisons that a given black-box sorting algorithm of Las-Vegas type makes when sorting lists of size n, the relevant features of the algorithm can be represented by a probability distribution σ_n on the set $\mathcal{A}_n$ of correct deterministic black-box algorithms for sorting such lists.

First, since we are only interested in the number of comparisons made, any correct deterministic black-box algorithm for sorting can be identified with its *behavior*, i.e.,

> with the mapping that determines for any given situation whether another and, if so, which comparison is made, where by situation we refer to the previously asked queries "$x_i < x_j$?" and their answers.

Note in this connection that the output of a black-box algorithm must be determined by the comparisons made together with the corresponding answers.

Yao's Minimax Principle

Representing Las Vegas algorithms by distributions

Since it suffices to consider black-box algorithms that never ask the same query twice, there are only finitely many possible behaviors of such algorithms, hence we can indeed assume that the set $\mathcal{A}_n$ of correct deterministic black-box algorithms is finite.

Then any Las Vegas algorithm can be identified with the probability distribution σ_n on $\mathcal{A}_n$ where the probability of any algorithm A in $\mathcal{A}_n$ is just the probability that the given Las Vegas algorithm and A behave the same (on all possible inputs).

Conversely, any probability distribution σ_n on $\mathcal{A}_n$ can be viewed as a randomized algorithm $\mathcal{A}_{\sigma_n}$ where on any given input initially some deterministic algorithm is chosen according to σ_n.

When going from a Las Vegas algorithm to the corresponding probability distribution σ_n and then going to the algorithm $\mathcal{A}_{\sigma_n}$, the initial Las Vegas algorithm and $\mathcal{A}_{\sigma_n}$ will have the same probabilities for the various possible behaviors.

Yao's Minimax Principle

Remark

Any sequence of probability distributions $\sigma_0, \sigma_1, \dots$ on $\mathcal{A}_0, \mathcal{A}_1, \dots$ can be viewed as representing a (generalized) Las Vegas algorithm.

In case the sequence of distributions is not given effectively, the corresponding Las Vegas algorithm may not be effective.

As long as we are only interested in lower bounds, considering also such not necessarily effective generalized Las Vegas algorithms is fine.

Yao's Minimax Principle

Remark

Consider the representation of a Las Vegas algorithm by probability distributions σ_n on the set $\mathcal{A}_n$ of correct algorithms. Then for inputs of size n, the expected costs in worst case are given by

$$\max_{I \in \mathcal{I}_n} \sum_{A \in \mathcal{A}_n} \mathrm{Prob}_{\sigma_n}[A] \cdot k_n(A, I).$$

Yao's Minimax Principle

Theorem (Yao's Minimax Principle)

Let $\mathcal{A}$ and $\mathcal{I}$ be nonempty finite sets. Let $k\colon \mathcal{A} \times \mathcal{I} \to \mathbb{N}$ be a cost function, and let σ and τ be probability distributions on $\mathcal{A}$ and $\mathcal{I}$. Let A_σ be a random variable with values in $\mathcal{A}$ and distribution σ, and let I_τ be a random variable with values in $\mathcal{I}$ and distribution τ. Then we have

$$\min_{A \in \mathcal{A}} \mathbf{E}\left[k(A, I_\tau)\right] \leq \max_{I \in \mathcal{I}} \mathbf{E}\left[k(A_\sigma, I)\right] . \tag{2}$$

Remark

In Yao's Minimax Principle, we have in inequality (2) on the

left-hand side the average costs on inputs chosen according to τ for the best deterministic algorithm (which "knows" τ),
right-hand side the expected costs for the randomized algorithm determined by σ on the worst input in $\mathcal{I}$.

Yao's Minimax Principle

Proof of Yao's Minimax Principle.

In order to demonstrate the theorem, we show for

$$\tilde{k} = \sum_{(A,I)\in\mathcal{A}\times\mathcal{I}} \mathrm{Prob}_\sigma[A]\cdot\mathrm{Prob}_\tau[I]\cdot k(A,I)$$

that we have

$$\min_{A\in\mathcal{A}} \mathbf{E}\left[k(A,I_\tau)\right] \le \tilde{k} \le \max_{I\in\mathcal{I}} \mathbf{E}\left[k(A_\sigma,I)\right].$$

While this is not used in what follows, observe that in case the random variables I_τ and A_σ are mutually independent, we have

$$\tilde{k} = \mathbf{E}\left[k(A_\sigma,I_\tau)\right].$$

Yao's Minimax Principle

Proof of Yao's Minimax Principle (continued).

By choice of $\tilde{k}$, we obtain

$$\begin{aligned}
\min_{A\in\mathcal{A}} \mathbf{E}\left[k(A, I_\tau)\right] &\leq \sum_{A\in\mathcal{A}} \mathrm{Prob}_\sigma[A] \cdot \mathbf{E}\left[k(A, I_\tau)\right] \\
&= \sum_{A\in\mathcal{A}} \mathrm{Prob}_\sigma[A] \sum_{I\in\mathcal{I}} \mathrm{Prob}_\tau[I] \cdot k(A, I) \\
&= \tilde{k} \\
&= \sum_{I\in\mathcal{I}} \mathrm{Prob}_\tau[I] \sum_{A\in\mathcal{A}} \mathrm{Prob}_\sigma[A] \cdot k(A, I) \\
&= \sum_{I\in\mathcal{I}} \mathrm{Prob}_\tau[I] \cdot \mathbf{E}\left[k(A_\sigma, I)\right] \leq \max_{I\in\mathcal{I}} \mathbf{E}\left[k(A_\sigma, I)\right].
\end{aligned}$$

□

Yao's Minimax Principle

Example Finding empty columns.

Let $\{0,1\}^{n \times n}$ be the set of Boolean $(n \times n)$ matrices.

A column of a Boolean matrix is said to be empty if all entries in the column are equal to 0.

A black-box algorithm for deciding whether a given quadratic Boolean matrix has an empty column is a correct algorithm for this problem that accesses its input matrix only by queries of the form "Is the entry in row i and column j equal to 0?".

Let the size of an input in $\{0,1\}^{n \times n}$ be equal to n, and let

- $\mathcal{I}_n$ be equal to $\{0,1\}^{n \times n}$,
- $\mathcal{A}_n$ be the set of all deterministic black-box algorithms that decide correctly for inputs of size n whether the input has an empty column,
- $k_n(A, I)$ be equal to the number of entries in I that algorithm A queries, i.e., checks for being 0, on input I.

Yao's Minimax Principle

Proposition (Upper bound)

There is a black-box Las Vegas algorithm that correctly decides whether a Boolean matrix has an empty column such that for matrices of size $(n \times n)$ the expected number of queries in worst case is at most

$$g_{\mathrm{upper}}(n) = \frac{n(n+1)}{2} .$$

Proof.

Consider the algorithm that chooses a permutation π of $\{1, \ldots, n\}$ uniformly at random and then, until enough information has been obtained, successively checks the columns $\pi(1), \pi(2), \ldots$ by checking in each column successively the rows $\pi(1), \pi(2), \ldots$.

On an input that has an empty column, the expected number of checked columns is at most $(n+1)/2$, while on any other input the expected number of checks per column is at most $(n+1)/2$. □

Yao's Minimax Principle

Proposition (Lower bound)

For any black-box Las Vegas algorithm that correctly decides whether a Boolean $(n \times n)$ matrix has an empty column, the expected number of queries in worst case is at least

$$g_{\text{lower}}(n) = \frac{n(n+1)}{2} \, .$$

Proof.

By the discussing above, we can assume that any correct black-box Las Vegas algorithm can be represented by a probability distribution σ_n on $\mathcal{A}_n$, hence the assertion of the proposition can be rephrased as

$$g_{\text{lower}}(n) \leq \max_{I \in \mathcal{I}_n} \mathbf{E}\left[k(A_{\sigma_n}, I)\right] \, .$$

Yao's Minimax Principle

Proof (continued).

By Yao's Minimax Principle, we have for all probability distributions σ_n on $\mathcal{A}_n$ and τ_n on $\mathcal{I}_n$,

$$\min_{A \in \mathcal{A}_n} \mathbf{E}\left[k(A, I_{\tau_n})\right] \leq \max_{I \in \mathcal{I}_n} \mathbf{E}\left[k(A_{\sigma_n}, I)\right] . \tag{3}$$

Thus in order to prove the proposition, it suffices to specify for all n a probability distribution τ_n on $\mathcal{I}_n$ such that the function g_{lower} provides lower bounds for the left-hand side of inequality (3).
In fact, it suffices to specify for any fixed n and for any $\varepsilon > 0$ a probability distribution τ_n^{ε} on $\mathcal{I}_n$ such that

$$(1 - \varepsilon)\, g_{\mathrm{lower}}(n) \leq \min_{A \in \mathcal{A}_n} \mathbf{E}\left[k(A, I_{\tau_n^{\varepsilon}})\right] .$$

Yao's Minimax Principle

Proof (continued).

In order to specify for given n and $\varepsilon > 0$ a probability distribution τ_n^ε on $\mathcal{I}_n$,

> let $\mathcal{D}_n$ be the set of all Boolean $(n \times n)$ matrices that have exactly one entry 1 per column.

Then define τ_n^ε such that

> the subset $\mathcal{D}_n$ of $\mathcal{I}_n$ has probability $1 - \varepsilon$ and the distribution within this set is uniform,
> the set $\mathcal{I}_n \setminus \mathcal{D}_n$ has probability ε and the distribution within this set is again uniform.

Yao's Minimax Principle

Proof (continued).

A correct deterministic algorithm for the empty column problem can reject an input in $\mathcal{D}_n$ only after reading at least one symbol 1 in every column.

For a matrix chosen uniformly at random from $\mathcal{D}_n$, for each column the expected number of entries that have to be read before the single 1 is found is

$$\sum_{i=1}^{n} \frac{i}{n} = \frac{n+1}{2} \ .$$

Hence the expected number of entries that have to be read in total is $\frac{n(n+1)}{2} = g_{\text{lower}}(n)$, where the expectation is with respect to choosing the input uniformly at random from $\mathcal{D}_n$, which is done with probability $(1 - \varepsilon)$. □

Matrix games

A *matrix game* $(\boldsymbol{\mathcal{A}}, \boldsymbol{\mathcal{I}}, \boldsymbol{k})$ is played by two players. The first player selects a strategy A from a finite set $\mathcal{A}$, the second player selects a strategy I from a finite set $\mathcal{I}$, and the first and second player try to minimize and to maximize, respectively, the payoff $k(A, I)$.

A matrix game can be represented by a matrix with $|\mathcal{A}|$ rows and $|\mathcal{I}|$ columns that has entries of the form $k(A, I)$.

As indicated by the notation used above, the situation of Yao's Minimax Principle can be viewed as a matrix game where the first player selects an algorithm from $\mathcal{A}$ and the second player selects an input from $\mathcal{I}$, while the matrix contains entries of the form $k(A, I)$.

In technical terms, matrix games are finite two-player zero-sum games with incomplete information.

Excursus on game theory

Example Two finger morra.

In the game of two finger morra each of two players show simultaneously either one or two fingers.
Let $k \in \{2, 3, 4\}$ be the total count of fingers shown.

In case k is even, Player I has to pay k units to Player II,
In case k is odd, Player I is paid k units by Player II.

The payoff in this game can be represented by the matrix

$$\begin{pmatrix} 2 & -3 \\ -3 & 4 \end{pmatrix},$$

where the rows and columns correspond to the strategies in the set $\mathcal{A} = \mathcal{I} = \{1, 2\}$ of Player I and II, respectively, and the entry $k(i, j)$ in row i and column j is equal to the loss of Player I.

Excursus on game theory

Mixed strategies

In a matrix game $(\mathcal{A}, \mathcal{I}, k)$, the strategies in $\mathcal{A}$ and $\mathcal{I}$ are called *pure strategies* for the first and second player, respectively.

A *mixed strategy* for a player in a matrix game is a probability distribution on the set of strategies for this player, which we identify with a corresponding random variable A_σ or I_τ.

When mixed strategies are allowed, the first and second player try to minimize and maximize, respectively, the expected payoff

$$\mathbf{E}\left[k(A_\sigma, I_\tau)\right] = \sum_{(A,I)\in(\mathcal{A},\mathcal{I})} \mathrm{Prob}_\sigma[A] \cdot \mathrm{Prob}_\tau[I] \cdot k(A, I) \; .$$

Example Two finger morra (continued).

Recall the payoff matrix K for the game of two finger morra

$$K = \begin{pmatrix} 2 & -3 \\ -3 & 4 \end{pmatrix} .$$

Any player who plays a known pure strategy will lose.
In case both players play mixed strategies with probabilities of $1/2$ for each of their two respective pure strategies, the expected payoff of Player I (and then also of Player II) is

$$\begin{aligned} \mathbf{E}\left[k(A_\sigma, I_\tau)\right] &= \sum_{(A,I)\in(\mathcal{A},\mathcal{I})} \mathrm{Prob}_\sigma[A] \cdot \mathrm{Prob}_\tau[I] \cdot k(A, I) \\ &= \frac{1}{4}(2-3-3+4) = 0. \end{aligned}$$

In fact, one of the players has a better strategy (see exercises).

Excursus on game theory

Definition

For a matrix game $(\mathcal{A}, \mathcal{I}, k)$, a mixed strategy σ^* for Player I is *optimal*, if it holds that

$$\max_\tau \mathbf{E}\left[k(A_{\sigma^*}, I_\tau)\right] = \min_\sigma \max_\tau \mathbf{E}\left[k(A_\sigma, I_\tau)\right]. \tag{4}$$

Similarly, a mixed strategy τ^* for Player II is *optimal*, if it holds that

$$\min_\sigma \mathbf{E}\left[k(A_\sigma, I_{\tau^*})\right] = \max_\tau \min_\sigma \mathbf{E}\left[k(A_\sigma, I_\tau)\right]. \tag{5}$$

Remark

In a matrix game, each player has an optimal strategy, which, however, is not necessarily unique. Here it suffices to observe that payoffs, hence also expected payoffs are bounded, and that the set of mixed strategies of each player is compact (in the sense of calculus), hence all minima and maxima in the definition of optimal strategy exist and are attained for appropriate mixed strategies.

Excursus on game theory

Remark

By playing an optimal mixed strategy σ^*, Player I can enforce that the expected payoff is at most

$$k_1^* = \max_{\tau} \mathbf{E}\left[k(A_{\sigma^*}, I_\tau)\right] = \min_{\sigma} \max_{\tau} \mathbf{E}\left[k(A_\sigma, I_\tau)\right],$$

no matter what mixed strategy Player II plays.

Similarly, by playing an optimal mixed strategy τ^*, Player II can enforce that the expected payoff is at least

$$k_2^* = \min_{\sigma} \mathbf{E}\left[k(A_\sigma, I_{\tau^*})\right] = \max_{\tau} \min_{\sigma} \mathbf{E}\left[k(A_\sigma, I_\tau)\right].$$

Remark

Observe that the values k_1^* and k_2^* do not depend on the choice of the optimal strategies σ^* and τ^* that occur in their definitions.

Excursus on game theory

Remark

For any matrix game it holds that $k_2^* \leq k_1^*$.

For a proof, assume that the two players play the optimal strategies σ^* and τ^*, which results in the expected payoff

$$k^* = \mathbf{E}\left[k(A_{\sigma^*}, I_{\tau^*})\right].$$

By optimality of σ^*, the expected payoff k^* is at least k_2^* and by a similar argument using the optimality of τ^* we obtain

$$k_2^* \leq k^* \leq k_1^*.$$

Von Neumann's celebrated Minimax Theorem asserts that for every given matrix game the values k_2^* and k_1^* are the same.

Before we review this theorem (without proof), we derive as an easy consequence that every matrix game has equilibrium points.

Excursus on game theory

An *equilibrium point* of a two-person game is a pair of mixed strategies such that neither player can strictly improve his payoff unilaterally, i.e., by switching to another strategy while the other player keeps his strategy.

Definition (Equilibrium point)

For a matrix game $(\mathcal{A}, \mathcal{I}, k)$, a pair (σ_0, τ_0) of mixed strategies for the first and second player, respectively, is an *equilibrium point* in case

for all mixed strategies σ of Player I it holds that

$$\mathbf{E}\left[k(A_{\sigma_0}, I_{\tau_0})\right] \leq \mathbf{E}\left[k(A_{\sigma}, I_{\tau_0})\right],$$

and for all mixed strategies τ of Player II it holds that

$$\mathbf{E}\left[k(A_{\sigma_0}, I_{\tau})\right] \leq \mathbf{E}\left[k(A_{\sigma_0}, I_{\tau_0})\right].$$

Remark

Assume $k_2^* = k_1^*$ and let σ^* and τ^* be arbitrary optimal strategies for Player I and II. Then the pair (σ^*, τ^*) is an *equilibrium point*.

For a proof, assume that the two players play the strategies σ^* and τ^*, with expected payoff $k^* = \mathbf{E}\left[k(A_{\sigma^*}, I_{\tau^*})\right]$.

Since k^* lies between k_2^* and k_1^*, all three values are the same.

By playing the optimal mixed strategy σ^*, Player *I* enforces the expected payoff to be at most $k_1^* = k^*$.

Accordingly, Player *II* cannot stricly improve the payoff k^* unilaterally by switching to a strategy different from τ^*.

By a similar argument, Player *I* cannot either stricly improve his payoff unilaterally.

Excursus on game theory

Von Neumann's Minimax Theorem

For any matrix game $(\mathcal{A}, \mathcal{I}, k)$, it holds that

$$\max_{\tau} \min_{\sigma} \mathbf{E}\left[k(A_\sigma, I_\tau)\right] = \min_{\sigma} \max_{\tau} \mathbf{E}\left[k(A_\sigma, I_\tau)\right] \qquad (6)$$

The following reformulation of von Neumann's Minimax Theorem relies on the fact that the optimum strategy against a fixed mixed strategy can always be chosen to be a pure strategy.

Variant of von Neumann's Minimax Theorem

For any matrix game $(\mathcal{A}, \mathcal{I}, k)$, it holds that

$$\max_{\tau} \min_{A \in \mathcal{A}} \mathbf{E}\left[k(A, I_\tau)\right] = \min_{\sigma} \max_{I \in \mathcal{I}} \mathbf{E}\left[k(A_\sigma, I)\right] .$$

Excursus on game theory

Proof that the variant is equivalent to the Minimax Theorem.

For arbitrary mixed strategies σ and τ it holds that

$$\begin{aligned}\mathbf{E}\left[k(A_\sigma, I_\tau)\right] &= \sum_{(A,I)\in(\mathcal{A},\mathcal{I})} \mathrm{Prob}_\sigma[A]\cdot\mathrm{Prob}_\tau[I]\cdot k(A,I)\\ &= \sum_{I\in\mathcal{I}} \mathrm{Prob}_\tau[I]\cdot\mathbf{E}\left[k(A_\sigma, I\right) \le \max_{I\in\mathcal{I}}\mathbf{E}\left[k(A_\sigma, I\right)\,,\end{aligned}$$

hence for any given σ, $\max_\tau \mathbf{E}\left[k(A_\sigma, I_\tau)\right] = \max_{I\in\mathcal{I}}\mathbf{E}\left[k(A_\sigma, I)\right]$, where $\geq$ is trivial and $\leq$ holds by the preceding discussion. Then

$$\min_\sigma \max_\tau \mathbf{E}\left[k(A_\sigma, I_\tau)\right] = \min_\sigma \max_{I\in\mathcal{I}} \mathbf{E}\left[k(A_\sigma, I)\right]\,,$$

i.e., the right-hand sides of the equations that asserted in the Minimax Theorem and in its variant have the same value. A similar argument for the left-hand sides then concludes the proof. □

Excursus on game theory

The variant of von Neumann's Minimax Theorem yields as corollary the following reformulation of Yao's Minimax Principle.

Theorem (Equivalent form of Yao's Minimax Principle)

Let $(\mathcal{A}, \mathcal{I}, k)$ be a matrix game and let A_{σ_0} and I_{τ_0} be mixed strategies for the first and second player. Then it holds that

$$\min_{A \in \mathcal{A}} \mathbf{E}\left[k(A, I_{\tau_0})\right] \leq \max_{I \in \mathcal{I}} \mathbf{E}\left[k(A_{\sigma_0}, I)\right] .$$

Proof.

The variant of von Neumann's Minimax Theorem yields

$$\begin{aligned} \min_{A \in \mathcal{A}} \mathbf{E}\left[k(A, I_{\tau_0})\right] &\leq \max_{\tau} \min_{A \in \mathcal{A}} \mathbf{E}\left[k(A, I_{\tau})\right] \\ &= \min_{\sigma} \max_{I \in \mathcal{I}} \mathbf{E}\left[k(A_{\sigma}, I)\right] \leq \max_{I \in \mathcal{I}} \mathbf{E}\left[k(A_{\sigma_0}, I)\right] . \end{aligned}$$

□

Excursus on game theory

Equilibrium points

By von Neumann's Minimax theorem, any matrix game, that is, any finite two-player zero-sum game with incomplete information has at least one equilibrium point.

By a celebrated result of Nash, the existence of equilibrium points remains valid in the more general case where the assumption that the game is zero-sum has been dropped.

For zero-sum games, the expected payoffs of all equilibrium points are the same, whereas this may be false in the more general case.

In the more general case, there may be two equilibrium points such that with one of them the expected payoffs of both players are strictly better of than with the other.

Excursus on game theory

Example Planes on collision course

Consider a situation where two planes are on a collision course without being able to communicate which each other.

Assume further that each plane's crew can decide on either descending or climbing and that the planes do not collide if and only if the two chosen actions are not the same.

Under certain additional simplifying assumptions, we can formalize this situation as a finite two-person game with incomplete information where the payoffs are given by

$$K = \begin{pmatrix} (-100,-100) & (0,0) \\ (0,0) & (-100,-100) \end{pmatrix} .$$

The payoffs -100 and 0 are the costs in case the collision occurs or not, respectively. Both players try to maximize their payoff.

Excursus on game theory

Example Planes on collision course

Recall the formalization of two planes on collision course by a game with payoffs

$$K = \begin{pmatrix} (-100, -100) & (0, 0) \\ (0, 0) & (-100, -100) \end{pmatrix} .$$

The pure strategies are climbing and descending.

A mixed strategy can be identified with its probability for climbing.

There are two equilibrium points $(0, 1)$ and $(1, 0)$, both with an expected payoff of 0 for both players.

Furthermore, there is the equilibrium point $(1/2, 1/2)$ with expected payoff of -50 for each player.

Observe that in case one of the players decides on his pure strategy by tossing a fair coin, no matter what the strategy of the other player is, the expected payoff will always be -50.

Matching lower and upper bounds for sorting

In what follows, we consider black-box Las Vegas algorithms for sorting and obtain essentially matching lower and upper bounds of about $n \log n$ on the expected number of comparisons in worst case (where n is the number of items in the list to be sorted).

The upper bound is obtained by a probabilistic argument that shows that randomized quicksort requires at most 1.4 $n \log n$ comparisons.

The lower bounds is obtained by arguing that

> any deterministic black-box algorithms requires roughly $n \log n$ comparisons on average when the inputs are chosen uniformly at random from the set of all inputs of size n,
> where the lower bound for deterministic algorithms extends to Las Vegas algorithms by Yao's Minimax Principle.

Algorithm $\mathrm{RandQuicksort}$ (Randomized Quicksort)

(Suppose that a strict linear ordering $<$ is understood.)

Input: A list $S = (\mathsf{s}_{\pi(1)}, \ldots, \mathsf{s}_{\pi(n)})$ of n pairwise distinct items ($(\mathsf{s}_1, \ldots, \mathsf{s}_n)$ is the ordered list, π is a permutation).

```
    Pick an item s of $S$ uniformly at random.
    $S_{\mathrm{small}} = (\mathsf{s}_{\pi(i)})_{\mathsf{s}_{\pi(i)} < \mathsf{s}}$
    $S_{\mathrm{large}} = (\mathsf{s}_{\pi(i)})_{\mathsf{s}_{\pi(i)} > \mathsf{s}}$
    If $|S_{\mathrm{small}}| > 1$, then $S_{\mathrm{small}} = \mathrm{RandQuicksort}(S_{\mathrm{small}})$.
    If $|S_{\mathrm{large}}| > 1$, then $S_{\mathrm{large}} = \mathrm{RandQuicksort}(S_{\mathrm{large}})$.
```

Output: $(S_{\mathrm{small}} \circ \mathsf{s} \circ S_{\mathrm{large}})$ ($\circ$ is concatenation).

In order to sort a list S, $\mathrm{RandQuicksort}$ is invoked with input S.

Randomized sorting

Black-box sorting

Recall that we are considering black-box sorting, i.e., the only way an algorithm may obtain information about an item in the input list is to compare the item to another item.

Randomized and deterministic quicksort

- Algorithm RandQuicksort differs from deterministic quicksort precisely by the choice of the pivot elements that are used to split the list that is currently processed.
- For deterministic quicksort there are (rare) bad inputs on which the algorithm always uses about n^2 comparisons.
- There are no particular bad inputs for randomized quicksort, however, on any input, with small probability, randomized quicksort may use about n^2 comparisons.

Randomized sorting: an upper bound

Proposition

When algorithm $\mathrm{RandQuicksort}$ *is run on any list of n pairwise distinct items, the expected number of comparisons required to sort the list is at most $1.4\, n \log n$.*

Proof.

Let $S = (\mathsf{s}_1, \ldots, \mathsf{s}_n)$ be any *ordered* list of n pairwise distinct items, and consider the application of $\mathrm{RandQuicksort}$ to any permutation $(\mathsf{s}_{\pi(1)}, \ldots, \mathsf{s}_{\pi(n)})$ of S.

Any pair of items is compared at most once because

the recursive calls to $\mathrm{RandQuicksort}$ are always for lists of items that have not yet been compared,
during such a call, any pair of items is compared at most once.

The number of comparisons is just the number of pairs $(\mathsf{s}_i, \mathsf{s}_j)$ with $i < j$ that are compared at all.

Randomized sorting: an upper bound

Proof (continued).

For any pair i, j where $1 \leq i < j \leq n$, consider the event that s_i is ever compared to s_j, and let p_{ij} be the probability of this event.

Furthermore, let X_{ij} be the corresponding indicator variable, (i.e., $X_{ij} = 1$ if the event occurs and $X_{ij} = 0$, otherwise).

The number of comparisons is just the sum over the X_{ij}, where

$$\mathbf{E}[X_{ij}] \;=\; p_{ij} \cdot 1 + (1 - p_{ij}) \cdot 0 \;=\; p_{ij} \ .$$

Hence the expected number of comparisons is

$$\mathbf{E}\left[\sum_{1 \leq i < j \leq n} X_{ij}\right] = \sum_{1 \leq i < j \leq n} \mathbf{E}[X_{ij}] = \sum_{1 \leq i < j \leq n} p_{ij} \ .$$

It remains to bound the sum of the probabilities p_{ij}.

Randomized sorting: an upper bound

Proof (continued).

Fix any pair of indices i and j where $i < j$ and consider the $j - i + 1$ items $\mathsf{s}_i, \dots, \mathsf{s}_j$.

During the recursive calls, these items always stick together until for the first time one of these items is picked for splitting.

Each of these items has the same chance of being picked first.

In case the item from this list that is picked first

- differs from s_i and s_j, the two latter items are assigned to different sublists and are never compared.
- is equal to s_i or s_j, the two items are compared.

In summary, the probability that s_i and s_j are compared at all is

$$p_{ij} = \frac{2}{j - i + 1} .$$

Randomized sorting: an upper bound

Proof (continued).

The expected number of comparisons then is equal to

$$\begin{aligned}\sum_{1\le i<j\le n} p_{ij} &= \sum_{1\le i<j\le n} \frac{2}{j-i+1} = 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\frac{1}{j-i+1}\\ &= 2\sum_{i=1}^{n-1}\sum_{k=2}^{n-i+1}\frac{1}{k} \le 2n\sum_{k=2}^{n}\frac{1}{k} \le 2n(\mathrm{H}_n - 1),\end{aligned}$$

where $\mathrm{H}_n = 1/1 + 1/2 + \ldots + 1/n$ is the nth Harmonic number. Recall from the section on stable marriages that for all $n \ge 2$,

$$\mathrm{H}_n - 1 \le \ln n = \ln 2 \cdot \log n < 0.7 \log n.$$

Hence the expected number of comparisons of $\mathrm{RandQuicksort}$ on any fixed input of size n is strictly less than $1.4\, n \log n$. □

Randomized sorting: a lower bound

The upper bound of $1.4 n \log n$ comparisons for randomized Quicksort is essentially matched by the following lower bound.

Proposition

For any real number $\varepsilon > 0$ and for almost all n, when sorting lists of n pairwise distinct items by any black-box Las Vegas algorithm, in worst case the expected number of comparisons is at least

$$(1 - \varepsilon) n \log n \, .$$

Proof.

Fix any $\varepsilon > 0$ and n. By the discussion of black-box Las Vegas algorithms for sorting preceding Yao's Minimax Principle,

there is a finite set $\mathcal{I}$ of inputs of size n,
there is a finite set $\mathcal{A}$ of all correct deterministic black-box algorithms for sorting inputs of size n.

Randomized sorting: a lower bound

Proof (continued).

Furthermore, any black-box Las Vegas algorithm that correctly sorts all inputs in $\mathcal{I}$ can be identified with a probability distribution σ on $\mathcal{A}$, where then we refer to the algorithm by A_σ.

If we let $k(A, I)$ be the number of comparisons that algorithm A makes on input I, we have to show that for almost all n,

$$(1-\varepsilon) n \log n \leq \max_{I \in \mathcal{I}} \mathbf{E}\left[k(A_\sigma, I)\right] . \tag{7}$$

By Yao's Minimax Principle, we have for any probability distribution σ on $\mathcal{A}$ and for the uniform distribution τ on $\mathcal{I}$,

$$\min_{A \in \mathcal{A}} \mathbf{E}\left[k(A, I_\tau)\right] \leq \max_{I \in \mathcal{I}} \mathbf{E}\left[k(A_\sigma, I)\right] . \tag{8}$$

So we are done by the next propostion, which shows that for almost all n the left-hand side of (7) is a lower bound for the left-hand side of (8). □

Randomized sorting: a lower bound

Proposition

For any real number $\varepsilon > 0$ and for almost all n, when sorting inputs that are chosen uniformly at random from all permutations of the set $\{1, \dots, n\}$, for any correct deterministic black-box algorithm the average number of comparisons is at least

$$(1 - \varepsilon) n \log n \ .$$

Proof.

Fix any $\varepsilon > 0$ and any natural number $n > 0$, and let $N = n!$.

We identify the set $\mathcal{I}$ of input lists with n elements with the set $\{S_1, \dots, S_N\}$ of all permutations of the set $\{1, \dots, n\}$.

Fix any deterministic black-box algorithm A that correctly sorts all lists in $\mathcal{I}$.

Randomized sorting: a lower bound

Proof (continued).

Let the word w_i be equal to the sequence of "answer bits" that algorithm A receives on input S_i when successively asking queries of the form "x < y?".

Then the average number of comparisons is $(|w_1| + \cdots + |w_N|)/N$ and it suffices to show that this value is at least $(1 - \varepsilon)n \log n$.

Claim 1 The set $\{w_1, \ldots, w_N\}$ has size N and is prefix-free.

For a proof, first observe that there cannot be indices $i \neq j$ such that $w_i = w_j$. For such indices, the distinct inputs S_i and S_j could not be distinguished by A, hence at least one of these inputs would not be sorted correctly.

Similarly, there cannot be indices i and j where w_i is a proper prefix of w_j. For such indices, the distinct inputs S_i and S_j cannot be distinguished by A based on the first $|w_i|$ queries, whereas A asks further queries on input S_i but not on input S_j, a contradiction.

Randomized sorting: a lower bound

Proof (continued).

We say a prefix-free set $\{w_1, \ldots, w_N\}$ of size N has minimum length sum if the sum $|w_1| + \cdots + |w_N|$ is minimum among all prefix-free sets of size N.

Claim 2 There is a prefix-free set of size N that has minimum length sum such that the lengths of any two words in the set differ at most by 1.

For a proof, it suffices to consider a prefix-free set $\{w_1, \ldots, w_N\}$ of size N that has minimum sum $\sum_{i=1}^{N} 2^{-|w_i|}$ among all prefix-free sets of size N that have minimum length sum.

If this set contains words w and w' such that $|w'| - |w| \geq 2$, then replacing w and w' by $w0$ and $w1$ yields as a contradiction another prefix-free set of size N that has minimum length sum and where the sum of the terms $2^{-|w_i|}$ has become strictly smaller due to

$$2^{-|w|} + 2^{-|w'|} > 2^{-|w|} = 2^{-|w0|} + 2^{-|w1|}\ .$$

Randomized sorting: a lower bound

Proof (continued).

By Claim 2, choose a prefix-free set $\{w_1, \ldots, w_N\}$ of size N that has mimimum length sum and where in addition all the w_i have length t or $t+1$ for some natural number t.

Accodingly, the minimum length sum is at least Nt and in order to show the proposition, it suffices to show $t \geq (1-\varepsilon)n \log n$.

There are 2^t and 2^{t+1} words of length t and $t+1$, respectively, and whenever the chosen prefix-free set contains a string w of length t, it contains neither $w0$ nor $w1$, hence 2^{t+1} is an upper bound for the size N of the set, i.e.,

$$t + 1 \geq \log N = \log n! \ . \tag{9}$$

Randomized sorting: a lower bound

Proof (continued).

For any natural number $k \geq 1$, among the n factors of $n!$ there are at most n/k that are less than or equal to n/k, hence we have

$$n! > (n/k)^{(n-n/k)} = (n/k)^{(1-1/k)n} . \qquad (10)$$

Putting together (3) and (4), we get for any $k \geq 1$

$$t + 1 \geq \log n! > \log(n/k)^{(1-1/k)n} = (1-1/k)n(\log n - \log k) .$$

Now pick k where $1/k < \varepsilon/2$. Then it holds for all sufficiently large n that

$$\begin{aligned} t &> (1-1/k)\, n\, (\log n - \log k) - 1 \\ &\geq (1-\varepsilon/2)\, n \log n - 1 \geq (1-\varepsilon) n \log n . \qquad \square \end{aligned}$$

In the proof above, in place of inequality (4) one could also work with approximations of $n!$ or $\log n!$ related to Stirling's formula.

Fingerprinting

Problem

Check the identity of two large files in a situation where comparing them bit by bit is not feasible.

This problem arises for example when we want to check whether

(i) a large file has been transmitted correctly over a possibly noisy channel,

(ii) the current and an older version of a file are the same.

Solution

Compare *fingerprints* of the files, not the whole file

Idea: the fingerprint is considerable smaller than the file itself, while distinct files have distinct fingerprints with high probability.

Fingerprinting

We assume that files are given as words over the alphabet $\{0, 1\}$.

Computing a fingerprint of a word

With k understood, map a given binary word $w = w_1 \dots w_n$ first to

$$f(w) = 2^{n-1} w_1 + 2^{n-2} w_2 + \dots + 2w_{n-1} + w_n ,$$

and then let $\quad f_k(w) = f(w) \mod k \quad$ be the fingerprint of w.

The fingerprint $f_k(w)$ requires approximately $\log k$ bits of storage, where $\log k$ is chosen much smaller than n.

The fingerprints $f_k(u)$ and $f_k(v)$ are the same if and only if $f(u)$ and $f(v)$ leave the same remainder modulo k, i.e., if

$$k \ \text{ divides } \ |f(u) - f(v)| \ .$$

Using a fixed value of k might be good enough for detecting random errors, but choosing k deterministically fails against deliberate changes by a malign adversary.

Fingerprinting

Lemma

For all sufficiently large t, there are at least t prime numbers less than or equal to $t \log t$.

Proof.

By the *prime number theorem*,

$$\lim_{m\to\infty} \frac{|\{p \leq m : p \text{ is prime }\}|}{m/\ln m} \quad = \quad 1 \ ,$$

i.e., for large m, the number of primes below m is at least $\frac{4}{5}\frac{m}{\ln m}$. For all sufficiently large t and for $m = t \log t$, the number of primes below m is then at least

$$\frac{4}{5}\frac{m}{\ln m} = \underbrace{\frac{4}{5\ln 2}}_{>1} \cdot \underbrace{\frac{\log t}{\log t + \log\log t}}_{\to 1} \cdot t \geq t \ . \qquad \square$$

Fingerprinting

Algorithm Fingerprint

(Parameter: a natural number t)

Input: Two words u and v of length n.

```
    Choose $p$ uniformly at random among the prime numbers
                                              below $tn \log tn$.
    If $f_p(u) = f_p(v)$ then accept, else reject.
```

A uniformly distributed prime below $tn \log tn$ can be obtained by picking random numbers in this range and running an efficient primality test on them, where the probability of not finding a prime can be made so small that this case can be neglected.

By the lemma above, there are tn primes in this range, hence the probability of not hitting a prime when picking r numbers

is at most $(1 - \frac{1}{r})^r \leq 1/\mathrm{e} < 1/2$ for $r = \log tn$,
and thus is at most 2^{-k} for $r = k \log tn$.

Fingerprinting

Proposition

Let Algorithm Fingerprint *be applied to words u and v of length n where the parameter t is so large that the lemma above applies. In case both words are identical, the algorithm accepts with probability 1, in case the two words differ, the algorithm accepts with probability at most $1/t$.*

Proof.

If u and v are identical, then $f_p(u) = f_p(v)$ for all values of p, hence the algorithm accepts with probability 1.

Next assume that u and v differ and let $d = |f(u) - f(v)|$.

The algorithm accepts if and only if p divides d.

We have $1 \leq d \leq 2^n$, hence d differs from 0 and has at most n distinct prime factors.

By the lemma above, there are at least tn primes below $tn \log tn$, thus the probability that p divides d is at most $\frac{n}{tn} = \frac{1}{t}$. □

Pattern matching

Consider the problem of comparing a given word $w = w_1 \dots w_n$, the *pattern*, to all the (consecutive) subwords of length n of a sequence

$$a = a_1 a_2 \dots a_m \ ,$$

the *text*, where the text is considerably longer than the pattern.

Formally, we are looking for the least index s such that the subword $a_s a_{s+1} \dots a_{s+n-1}$ of the text a is equal to the pattern w (alternatively, we may look for all such indices s).

Instead of comparing the pattern directly to all relevant subwords, we may compare the corresponding fingerprints.

The fingerprint of the pattern has to be computed only once.

The fingerprints of the possible subwords are computed successively, where the fingerprint of every next subword can be computed easily from the fingerprint of the preceding subword.

Computing successive fingerprints

Recall that the fingerprint of a word w of length n is

$$f_p(w) = (2^{n-1}w_1 + 2^{n-2}w_2 + \ldots + 2w_{n-1} + w_n) \mod p\,,$$

for some prime p chosen at random. Then for successive subwords

$$u = a_i a_{i+1} \ldots a_{i+n-1} \quad \text{and} \quad u' = a_{i+1} a_{i+2} \ldots a_{i+n}$$

their fingerprints are given by

$$\begin{aligned} f_p(u) &= (2^{n-1}a_i + 2^{n-2}a_{i+1} + \cdots + 2a_{i+n-2} + a_{i+n-1}) \mod p, \\ f_p(u') &= (2^{n-1}a_{i+1} + 2^{n-2}a_{i+2} + \cdots + 2a_{i+n-1} + a_{i+n}) \mod p, \end{aligned}$$

hence we have $f_p(u') = 2(f_p(u) - 2^{n-1}a_i) + a_{i+n} \mod p$.

Pattern matching

Expected number of false matches

Consider any subword u of length n of the text.

If the pattern and the subword u are the same, then their fingerprints are the same, too.

If the pattern and the subword u differ, then an analysis similar to the one of algorithm $\mathrm{Fingerprint}$ applies.

In case the prime p is chosen among the first $tn \log tn$ primes, the probability of matching fingerprints for any fixed subword that differs from the pattern is at most $1/t$.

By linearity of expectation, the expected frequency of such false matches is at most $1/t$ with respect to the random choice of p.

Pattern matching

Monte Carlo and Las Vegas version

In case the algorithm for pattern matching described above immediately returns any index where the subword at the corresponding position and the pattern have matching finger prints, the algorithm has a certain probability for being incorrect, i.e., we obtain a Monte Carlo algorithm.

In case the algorithm, when encountering matching fingerprints assumes a match only after having verified the match by comparing the subword and the pattern, the output is always correct, i.e., we obtain a Las Vegas algorithm.

Note that there are deterministic algorithms for finding a pattern in a text that run as fast but are based on other principles than the randomized algorithms above.

Bad inputs

By linearity of expectation, for any pattern and any text, the expected frequency of false matches is at most $1/t$.

There are bad inputs of a pair of pattern and text where for certain choices of p the frequency of false matches is larger than $1/t$.

For example, consider pattern $w = 0^n$ and text $a = 1^m$.
Then for p that divides the difference between $f(0^n)$ and $f(1^n)$, every subword of the text will be a false match.

Consider a variant of the Las Vegas version of the algorithm where a new prime p is picked at random every time a false match is detected. Then large deviations from the expected number of false matches

> don't occur any more with rather high probability on certain bad inputs,
> but may occur with small probability on all inputs.

Self-correcting programs

Correctness

Existing methods for ensuring correctness of hard- or software are not fully satisfactory.

Formal methods for checking the correctness exists but often are considered to be too complicated or to complex for being applied.

The extensive runtime checks with test data that are done in practice cannot guarantee to find all errors.

The Pentium division bug (1994)

The Pentium division bug resulted in incorrect computations for certain rare inputs (according to some sources, division was incorrect for a fraction of roughly 10^{-10} of all arguments).

The division bug indicates that even the most basic functions of highly tested (and hopefully also verified) products may be incorrect, and that in particular rare errors are hard to detect.

Self-checking and self-correcting programs

Testing can ensure that results are correct for most inputs.

Is there a general scheme for avoiding incorrect results at all in case a high fraction of results is correct?

Idea: relate the results for given inputs to the results for

- inputs that are "less complex" and hence can be assumed to be correct,
- inputs, that are randomly chosen and hence can be assumed to be correct with high probability.

This idea can be used

- for checking results (i.e., for detecting errors),
- for correcting results.

We obtain programs that check or even correct themselves.

Self-correcting programs

Algorithm MultiplicationCheck

(Parameter: a natural number t)

Input: three natural numbers a, b and c of size at most 2^n.

```
    Choose $p$ uniformly at random among the
                                   first $tn \log tn$ prime numbers.
    Let $d_1 = c \mod p$.
    Let $d_2 = ((a \mod p)(b \mod p)) \mod p$.
    If $d_1 = d_2$ then accept, else reject.
```

Algorithm MultiplicationCheck accepts an input a, b, c

- with probability 1 in case $ab = c$,
- with probability at most $1/t$ in case $ab \neq c$.

The analysis is similar to the one of Algorithm Fingerprint.

Self-correcting programs

Algorithm MultiplicationCorrection

Input: two natural numbers a and b, each represented by n bits.

```
    Determine $n$-bit natural numbers $r_1$ and $r_2$
                              by $2n$ independent tosses of a fair coin.
    Let $c = (a + r_1)(b + r_2) - r_1(b + r_2) - r_2(a + r_1) + r_1 r_2$    (*)
```

Output: c (where the intended value of c is ab).

The right-hand side of equation $(*)$ evaluates to the required value ab in case all arithmetical operations are correct.

Algorithm MultiplicationCorrection turns a multiplication procedure that *on certain rare inputs is always incorrect* into a multiplication procedure that *on any input is correct with high probability*.

Self-correcting programs

Proposition

Assume that Algorithm $\mathrm{MultiplicationCorrection}$ *is run on a processor where addition and subtraction are always correct, and*

> *multiplication is incorrect for a fraction of at most* ε *of all pairs of* $(n+1)$*-bit numbers.*

Then for any pair a and b of n-bit numbers, the probability that the algorithm fails to compute the product of a and b correctly is at most 16ε*.*

Proof.

It suffices to show that each of the four multiplications in $(*)$ is incorrect with probability at most 4ε.

Observe that for all four multiplication both arguments are $(n+1)$-bit natural numbers (where n-bit numbers are identified with appropriate $(n+1)$-bit numbers).

Self-correcting programs

Proof (continued).

We just consider the evaluation of the term $r_1(b + r_2)$, and omit the very similar consideration for the three remaining terms.

The numbers r_1 and r_2 are chosen uniformly and independently from all n-bit numbers, and the mapping $x \mapsto b + x$ is injective.

Consequently, the pair $(r_1, (b + r_2))$ is chosen uniformly from a set P of 2^{2n} pairs of $(n + 1)$-bit numbers.

By assumption, multiplication is not correct

> for at most an ε-fraction of all pairs of $(n + 1)$-bit numbers,
> that is, for at most $\varepsilon 2^{2(n+1)} = 4\varepsilon 2^{2n}$ such pairs,
> hence for at most an 4ε-fraction of all pairs in P. $\square$

The Lovasz Local Lemma

Avoiding certain events simultaneously

Suppose that $E_1, \dots, E_n$ are events that correspond to errors or other outcomes that we want to avoid.

In what follows, we investigate conditions on the underlying probability distribution that imply that the events E_i can be avoided simultaneously, in the sense of implying nonzero probability for the event

$$G = \bigcap_{i=1,\dots,n} \overline{E_i} \; .$$

The Lovasz Local Lemma

Sum of probabilities

If the sum $\mathrm{Prob}[E_1] + \cdot + \mathrm{Prob}[E_n]$ is strictly less than 1, then the event G has nonzero probability.

A necessary condition for G having nonzero probability is that all E_i have probability strictly smaller than 1, where the latter condition is in fact equivalent in case of mutual independence.

Independence

If the events $E_1, \ldots, E_n$ are mutually independent, then G has nonzero probability if and only if $\mathrm{Prob}[E_i] < 1$ for all i.

For a proof, it suffices to observe that by independence we have

$$\mathrm{Prob}[G] = \mathrm{Prob}\left[\bigcap_{i=1}^{n} \overline{E_i}\right] = \prod_{i=1}^{n} \mathrm{Prob}[\overline{E_i}] = \prod_{i=1}^{n} (1 - \mathrm{Prob}[E_i]) \, .$$

The Lovasz Local Lemma

So $E_1, \ldots, E_n$ can be simultaneously avoided with nonzero probability

> in general, if each event has probability strictly less than $1/n$,
> in case the events are mutually independent, if each event has probability strictly less than 1 .

In case each event depends on at most d other events (in a sense to be explained in a minute), by the Lovasz Local Lemma it suffices to require that each event has probability of at most $\mathrm{O}(1/d)$.

Definition

Let $X_1, \ldots, X_n$ be random variables with ranges $A_1, \ldots, A_n$, respectively. The random variable X_i is *mutually independent* of the random variables $X_{i_1}, \ldots, X_{i_t}$ if for all $a_i \in A_i$ and all $a_{i_j} \in A_{i_j}$

$$\mathrm{Prob}[X_i = a_i | X_{i_1} = a_{i_1} \& \cdots \& X_{i_t} = a_{i_t}] = \mathrm{Prob}[X_i = a_i] \ .$$

In this situation, we say that the random variable X_i *depends* among $X_1, \ldots, X_n$ *only* on the set of all random variables X_j where j is in the set $D = \{1, \ldots, n\} \setminus \{i, i_1, \ldots, i_t\}$.

The Lovasz Local Lemma

The notation above is extended to events in the natural way.

Definition

Let $E_1, \ldots, E_n$ be events. The event E_i is *mutually independent* of the events $E_{i_1}, \ldots, E_{i_t}$, if for all events $L_{i_1}, \ldots, L_{i_t}$ such that each L_j is equal either to E_j or to $\overline{E_j}$ it holds that

$$\mathrm{Prob}[E_i | L_{i_1} \& \cdots \& L_{i_t}] = \mathrm{Prob}[E_i] \ .$$

In this situation, we say that the event E_i *depends* among the events $E_1, \ldots, E_n$ *only* on the set of all events E_j where j is in the set $D = \{1, \ldots, n\} \setminus \{i, i_1, \ldots, i_t\}$.

This way E_i is *mutually independent* of the events $E_{i_1}, \ldots, E_{i_t}$ and, accordingly, E_i depends only on the events E_j with j in the set D as above, if corresponding statemens are true for the indicator variables of the events $E_i, E_{i_1}, \ldots, E_{i_t}$.

The Lovasz Local Lemma

Events $E_1, \dots, E_n$ are mutually independent if and only if each E_i is mutually independent of the set of all other events.

The following example indicates that the notion of an event depending only on certain other events lacks certain properties suggested by its naming.

Example: depending only on certain variables

Suppose a fair coin is tossed 3 times and let X_i be the indicator variable for the event that toss i shows head. Furthermore, let

$$X_4 = X_1 \oplus X_2, \quad X_5 = X_1 \oplus X_3, \quad X_6 = X_2 \oplus X_3 \ .$$

Then the random variable X_1 is mutually independent of the random variables X_4, X_5, X_6, as well as of the random variables X_2, X_3, X_6. Accordingly, X_1 depends only on the set $\{X_2, X_3\}$, but also depends only on the set $\{X_4, X_5\}$. , X_1 depends on X_2 and X_4 in the strong sense that $X_1 = X_2 \oplus X_4$.

The Lovasz Local Lemma

Theorem (Lovasz Local Lemma)

Let $E_1, \ldots, E_n$ be events and let $D_1, \ldots, D_n$ be subsets of $\{1, \ldots, n\}$ such that for $i = 1, \ldots, n$, the event E_i depends among $E_1, \ldots, E_n$ only on the set of all events E_j where $j \in D_i$. Furthermore, assume that there are real numbers $x_1, \ldots, x_n$ in the interval $[0, 1]$ where

$$\mathrm{Prob}[E_i] \leq x_i \prod_{j \in D_i} (1 - x_j)\ .$$

Then it holds for the event $G = \bigcap_{i=1,\ldots,n} \overline{E_i}$ that

$$\mathrm{Prob}[G] \geq \prod_{i=1,\ldots,n} (1 - x_i)\ .$$

For a proof of the lemma see the appendix to this chapter.

The Lovasz Local Lemma

Corollary (Basic form of the Lovasz Local Lemma)

Let $E_1, \ldots, E_n$ be events where $\mathrm{Prob}[E_i] \leq p$ for some real $p < 1$. Furthermore, assume that there is a constant d such that, first, for Euler's number $\mathrm{e} = 2.71\ldots$ it holds that

$$\mathrm{e}(d+1)p \leq 1$$

and, second, there are sets $D_1, \ldots, D_n$ of size at most d such that for $i = 1, \ldots, n$, the event E_i depends among $E_1, \ldots, E_n$ only on the set of all events E_j where $j \in D_i$.
Then it holds for the event $G = \bigcap_{i=1,\ldots,n} \overline{E_i}$ that

$$\mathrm{Prob}[G] > 0 \ .$$

The Lovasz Local Lemma

Proof of the corollary from the local lemma.

For $d = 0$, the events E_i are mutually independent, hence the conclusion of the corollary is immediate by $\mathrm{Prob}[E_i] < 1$.

So assume $d > 0$ and let $x_i = 1/(d+1)$.
Then the assumption of the Local Lemma is satisfied because it holds for $i = 1, \ldots, n$ that

$$x_i \prod_{j \in D_i} (1 - x_j) \geq \frac{1}{d+1} \left(1 - \frac{1}{d+1}\right)^d \geq \frac{1}{d+1} \cdot \frac{1}{\mathrm{e}} \geq p \geq \mathrm{Prob}[E_i] \,,$$

where the second inequality holds because $(1 - \frac{1}{d})^d$ converges nonincreasingly to $1/\mathrm{e}$. The Local Lemma then yields

$$\mathrm{Prob}[G] \geq \prod_{i=1,\ldots,n} (1 - x_i) = \left(1 - \frac{1}{d+1}\right)^n > 0.$$

$\square$

The Lovasz Local Lemma

The proof of the following proposition provides an example for an application of the Lovasz Local Lemma.

Recall that a Boolean formula φ is in k conjunctive normal form (k-CNF) if φ is a conjunction of (disjunctive) clauses where each clause contains exactly k literals with mutually distinct variables.

Note that there are more liberal definitions of k-CNF where, for example, it is just required that each clause contains at most k literals.

Proposition

Let φ be a Boolean formula in k-CNF such that every variable occurs in at most $2^{k-\log k-2}$ clauses of φ. Then φ is satisfiable.

The Lovasz Local Lemma

Proof.

Consider the random experiment where the variables in φ are assigned truth values by independent tosses of a fair coin.

Let E_i be the event that the ith clause of φ is not satisfied.

By construction, we have $p = \mathrm{Prob}[E_i] = 1/2^k$.

Each E_i is mutually independent of the set of all E_j such that the ith and the jth clause of φ do not have a variable in common.

Thus by assumption on φ, any event E_i depends among the events E_j only on a set of at most $d = k2^{k-\log k-2} - k$ events, where

$$\mathrm{e}(d+1)p \le \mathrm{e}k2^{k-\log k-2}2^{-k} < 1 \,.$$

Hence the basic form of the Lovasz Local Lemma applies and φ will be satisfied with nonzero probability. □

Appendix: Proof of the Lovasz Local Lemma

We demonstrate the Lovasz Local Lemma stated above.

Theorem (Lovasz Local Lemma)

Let $E_1, \ldots, E_n$ be events and let $D_1, \ldots, D_n$ be subsets of $\{1, \ldots, n\}$ such that for $i = 1, \ldots, n$, the event E_i depends among $E_1, \ldots, E_n$ only on the set of all events E_j where $j \in D_i$. Furthermore, assume that there are real numbers $x_1, \ldots, x_n$ in the interval $[0, 1]$ where

$$\mathrm{Prob}[E_i] \leq x_i \prod_{j \in D_i} (1 - x_j) \,.$$

Then it holds for the event $G = \bigcap_{i=1,\ldots,n} \overline{E_i}$ that

$$\mathrm{Prob}[G] \geq \prod_{i=1,\ldots,n} (1 - x_i) \,.$$

Appendix: Proof of the Lovasz Local Lemma

Proof: We have to show

$$\mathrm{Prob}[G] \geq \prod_{i=1,\ldots,n} (1 - x_i) \ .$$

By definition of G and the chain rule formula we obtain

$$\begin{aligned}\mathrm{Prob}[G] &= \mathrm{Prob}\left[\cap_{i=1,\ldots,n}\overline{E_i}\right] \\ &= \mathrm{Prob}[\overline{E_1}] \cdot \mathrm{Prob}[\overline{E_2}|\overline{E_1}] \cdot \mathrm{Prob}[\overline{E_3}|\overline{E_1} \cap \overline{E_2}] \\ &\qquad \cdot \mathrm{Prob}[\overline{E_4}|\overline{E_1} \cap \overline{E_2} \cap \overline{E_3}] \cdot \quad \cdots \quad \cdot \mathrm{Prob}[\overline{E_n}|\overline{E_1} \cap \cdots \cap \overline{E_{n-1}}] \ ,\end{aligned}$$

where in turn we have for $i = 1, \ldots, n$

$$\mathrm{Prob}[\overline{E_i}|\overline{E_1} \cap \cdots \cap \overline{E_{i-1}}] = 1 - \mathrm{Prob}[E_i|\overline{E_1} \cap \cdots \cap \overline{E_{i-1}}] \ .$$

So it suffices to show $\mathrm{Prob}[E_i|\overline{E_1} \cap \cdots \cap \overline{E_{i-1}}] \leq x_i$ for $i = 1, \ldots, n$.

Appendix: Proof of the Lovasz Local Lemma

Proof (continued): In order to show $\mathrm{Prob}[E_i|\overline{E_1} \cap \cdots \cap \overline{E_{i-1}}] \leq x_i$, we demonstrate

$$\mathrm{Prob}[E_i| \bigcap_{j \in S} \overline{E_j}] \leq x_i \; ,$$

for all $S \subseteq \{1, \ldots, n\}$ by induction over the size of S and simultaneously for all i.

Base case $|S| = 0$: By assumption of the Local Lemma it holds for $i = 1, \ldots, n$ that

$$\mathrm{Prob}[E_i| \bigcap_{j \in S} \overline{E_j}] = \mathrm{Prob}[E_i] \leq x_i \prod_{j \in D_i} (1 - x_j) \leq x_i \; .$$

Appendix: Proof of the Lovasz Local Lemma

Proof (continued): Induction step $|S| = k$ to $|S| = k + 1$:

Fix i in $\{1, \ldots, n\}$ and partition S into the sets

$$S_{\mathrm{dep}} = S \cap D_i \qquad \text{and} \qquad S_{\mathrm{ind}} = S \setminus D_i \ .$$

Case I: $S_{\mathrm{dep}} = \emptyset$. By definition of D_i and choice of S_{ind} we have

$$\mathrm{Prob}[E_i | \bigcap_{j \in S} \overline{E_j}] = \mathrm{Prob}[E_i | \bigcap_{j \in S_{\mathrm{ind}}} \overline{E_j}] = \mathrm{Prob}[E_i] \leq x_i \ .$$

Appendix: Proof of the Lovasz Local Lemma

Proof (continued):

Case II: $S_{\mathrm{dep}} \neq \emptyset$. By definition of conditional probability we have

$$\mathrm{Prob}[E_i | \bigcap_{j\in S} \overline{E_j}] = \frac{\mathrm{Prob}[E_i \cap \bigcap_{j\in S} \overline{E_j}]}{\mathrm{Prob}[\bigcap_{j\in S} \overline{E_j}]}$$

$$= \frac{\overbrace{\mathrm{Prob}[E_i \cap \bigcap_{j\in S_{\mathrm{dep}}} \overline{E_j} | \bigcap_{j\in S_{\mathrm{ind}}} \overline{E_j}]}^{(*)} \cdot \mathrm{Prob}[\bigcap_{j\in S_{\mathrm{ind}}} \overline{E_j}]}{\underbrace{\mathrm{Prob}[\bigcap_{j\in S_{\mathrm{dep}}} \overline{E_j} | \bigcap_{j\in S_{\mathrm{ind}}} \overline{E_j}]]}_{(**)} \cdot \mathrm{Prob}[\bigcap_{j\in S_{\mathrm{ind}}} \overline{E_j}]}$$

$$\leq \frac{x_i \prod_{j\in D_i}(1-x_j)}{\prod_{j\in S_{\mathrm{dep}}}(1-x_j)} \leq x_i \prod_{j\in D_i\setminus S_{\mathrm{dep}}} (1-x_j) \leq x_i \ .$$

Appendix: Proof of the Lovasz Local Lemma

Proof (continued):

Case II: $S_{\text{dep}} \neq \emptyset$. By definition of conditional probability we have

$$\mathrm{Prob}[E_i | \bigcap_{j \in S} \overline{E_j}] = \frac{\mathrm{Prob}[E_i \cap \bigcap_{j \in S} \overline{E_j}]}{\mathrm{Prob}[\bigcap_{j \in S} \overline{E_j}]}$$

$$= \frac{\overbrace{\mathrm{Prob}[E_i \cap \bigcap_{j \in S_{\text{dep}}} \overline{E_j} | \bigcap_{j \in S_{\text{ind}}} \overline{E_j}]}^{=(*)} \cdot \mathrm{Prob}[\bigcap_{j \in S_{\text{ind}}} \overline{E_j}]}{\underbrace{\mathrm{Prob}[\bigcap_{j \in S_{\text{dep}}} \overline{E_j} | \bigcap_{j \in S_{\text{ind}}} \overline{E_j}]]}_{=(**)} \cdot \mathrm{Prob}[\bigcap_{j \in S_{\text{ind}}} \overline{E_j}]}$$

$$\le \frac{x_i \prod_{j \in D_i} (1 - x_j)}{\prod_{j \in S_{\text{dep}}} (1 - x_j)} \le x_i \prod_{j \in D_i \setminus S_{\text{dep}}} (1 - x_j) \le x_i \ .$$

Appendix: Proof of the Lovasz Local Lemma

Proof (continued): The value of $(*)$ can bounded as follows

$$
\begin{aligned}
(*) &= \mathrm{Prob}[E_i \cap \bigcap_{j \in S_{\mathrm{dep}}} \overline{E_j} | \bigcap_{j \in S_{\mathrm{ind}}} \overline{E_j}] \\
&\leq \mathrm{Prob}[E_i | \bigcap_{j \in S_{\mathrm{ind}}} \overline{E_j}] = \mathrm{Prob}[E_i] \leq x_i \prod_{j \in D_i} (1 - x_j) \,.
\end{aligned}
$$

In the second line, the three relations hold from left to right because first, the compared probabilities are with respect to two events where one is a subset of the other, second, S_{ind} is a subset of D_i, and, third, by assumption of the Local Lemma.

Appendix: Proof of the Lovasz Local Lemma

Proof (continued): In order to bound $(**)$, recall that by case assumption we have $|S_{\text{dep}}| = r > 0$, say, $S_{\text{dep}} = \{j_1, \ldots, j_r\}$.
By a variant of the chain rule formula we obtain for $F = \bigcap_{j \in S_{\text{ind}}} \overline{E_j}$

$$
\begin{aligned}
(**) &= \mathrm{Prob}[\bigcap_{j \in S_{\text{dep}}} \overline{E_j} | F] \\
&= \mathrm{Prob}[\overline{E_{j_1}} | F] \cdot \mathrm{Prob}[\overline{E_{j_1}} \cap \overline{E_{j_2}} | F \cap \overline{E_{j_1}}] \\
&\qquad \cdot \quad \cdots \quad \cdot \mathrm{Prob}[\overline{E_{j_r}} | F \cap \overline{E_{j_1}} \cap \cdots \cap \overline{E_{j_{r-1}}}] \\
&= \prod_{t=1,\ldots,r} \mathrm{Prob}[\overline{E_{j_t}} | \underbrace{F \cap \overline{E_{j_1}} \cap \cdots \cap \overline{E_{j_{t-1}}}}_{\text{at most } |S|-1 \text{ sets } \overline{E_j}}] \\
&\geq \prod_{t=1,\ldots,r} (1 - x_{j_t}) = \prod_{j \in S_{\text{dep}}} (1 - x_j)\,,
\end{aligned}
$$

where the inequality is immediate by the induction hypothesis. $\square$

Karger's Min-Cut algorithm

Recall from the section on derandomization that

> a *cut* of a graph $G = (V, E)$ is a partition of V into two disjoint subsets V_0 and V_1,
> the *weight of a cut* (V_0, V_1) is the number of edges between V_0 and V_1.

Definition (Min-cut problem)

A cut (V_0, V_1) is *nontrivial* in case V_0 and V_1 are both nonempty.

A *mininum cut* of a given graph G is a nontrivial cut of G that has minimum weight among all nontrivial cuts of G.

The *min-cut problem* asks to find a minimum cut of a given graph.

The size of a minimum cut of a graph G is the minimum number of edges that must be removed from G in order to render G disconnected.

Minimum cuts can be computed in deterministic polynomial time via computing corresponding maximum flows.

Karger's Min-Cut algorithm

In what follows, we will discuss a randomized algorithm for solving the min-cut problem that has been proposed by Karger.

It is suggesting to describe the algorithm and its verification in terms of multigraphs and cuts of multigraphs.

Definition (Multigraph)

A *multigraph* is a pair $G = (V, E)$ where V is a finite set and $E\colon V \times V \to \mathbb{N}$ is a mapping such that for all $u, v \in G$ we have

$$E(u, v) = E(v, u) \quad \text{and} \quad E(u, u) = 0$$

(i.e., our multigraphs are undirected and do not have loops).

For a multigraph $G = (V, E)$, the members of V are called *nodes*, and all sets of the form $\{u, v\}$ for two distinct nodes u and v are called *edges*. The *multiplicity* of an edge $\{u, v\}$ is $E(u, v)$.

The Contract algorithm

A graph can be identified with a multigraph where all multiplicities are either 0 or 1; however note that

for a graph $G = (V, E)$ only the pairs $\{u, v\}$ contained in E are called edges,
in a multigraph $G = (V, E)$ any pair $\{u, v\}$ of two distinct nodes is called an edge of G, even in case the multiplicity of $\{u, v\}$ is 0.

Furthermore, there are contexts where an edge $\{u, v\}$ of a multigraph of multiplicity t is viewed as a set of t different copies of an edge between u and v (of multiplicity 1).

Similar to the latter view, for a multigraph $G = (V, E)$ the *number of edges counted with multiplicities* is

$$\mathrm{m}(E) = \frac{1}{2} \sum_{(u,v)\in V\times V} E(u, v) = \sum_{\{(u,v)\in V\times V :\ u<v\}} E(u, v),$$

where for the last sum term we assume some strict order $<$ on V.

The $\mathrm{Contract}$ algorithm

The notions of cut and trivial cut of a graph extend by literally the same definition to multigraphs.

Definition (Minimum cuts of multigraphs)

Let $G = (V, E)$ be a multigraph. The *weight* of a cut (V_0, V_1) of G is

$$\mathrm{w}(V_0, V_1) = \sum_{u \in V_0, v \in V_1} E(u, v).$$

A cut (V_0, V_1) of G is minimum if the cut is nontrivial and has minimum weight among all nontrivial cuts of G.

The $\mathrm{Contract}$ algorithm

Definition (Contracting of an edge)

Let $G = (V, E)$ be a multigraph and let u and v be any pair of two distinct nodes of G.

The multigraph obtained from G by *contracting the edge* $\{u, v\}$ *towards* u is $(V \setminus \{v\}, E')$ where for all x, y in $V \setminus \{u, v\}$ we have

$$\begin{aligned} E'(x, y) &= E(x, y) \\ E'(u, x) &= E'(x, u) = E(u, x) + E(v, x). \end{aligned}$$

Contracting an edge $\{u, v\}$ towards the node u is also referred to as contracting or collapsing nodes u and v into u.

Edges with arbitrary multiplicity including 0 can be contracted.

The multigraph resulting from contracting an edge does not depend on the multiplicity of the contracted edge.

The Contract algorithm

Algorithm Contract

Input: A graph $G = (V, E)$ (Assume that $V = \{1, \ldots, n\}$).

($G = (V, E)$ is considered as a multigraph.)

Initialization: let $n = |V|$, $V_n = V$, $E_n = E$ and $G_n = (V_n, E_n)$.

```
For $i = n, \ldots, 3$
    Choose edge $\{u, v\} \in E_i$ uniformly at random while taking into
        account the multiplicities of the edges
        (i.e., edge $\{u, v\}$ is chosen with probability $\frac{E_i(u,v)}{\mathrm{m}(E_i)}$).
    Let $G_{i-1} = (V_{i-1}, E_{i-1})$ be the multigraph obtained from $G_i$ by
        contracting the edge $u$ and $v$ into the node $\min\{u, v\}$.
```

Output: The cut (U_0, U_1) where U_0 and U_1 are the sets of nodes that have been contracted into the two nodes of V_2, respectively.

Recall that $\mathrm{m}(E)$ is the sum of the multiplicities as given by E over all edges in a multigraph (V, E).

Verification of the Contract algorithm

Theorem

If algorithm Contract *is run on a graph with n nodes, each minimum cut of the graph is output with probability at least $\frac{2}{n^2}$.*

Proof.

Let G be any graph with n nodes and fix some minimum cut of G of weight k, which we refer to as the desigated cut.

For $i = n, \dots, 2$, the algorithm constructs graphs $G_i = (V_i, E_i)$.

Fix some i and for any set U of nodes of G_i, let $\mathrm{expand}(U)$ be the set of nodes of V that have been contracted into a node of U.

If (U_0, U_1) is a cut of G_i, then $(\mathrm{expand}(U_0), \mathrm{expand}(U_1))$ is a cut of G of the same weight, hence (U_0, U_1) has weight at least k.

For every node v in G_i the weight of the cut $(\{v\}, V_i \setminus \{v\})$ is equal to the degree of v, which is thus at least k. So we obtain

$$|E_i| \geq \frac{k|V_i|}{2} = \frac{ki}{2}.$$

Verification of the Contract algorithm

Proof (continued).

Call G_i good in case during the construction of $G_n, \ldots, G_i$ no edge crossing the designated cut has been contracted.

By definition, G_i is good if and only if the edges of the designated cut correspond to a minimum cut of G_i of weight k.

The algorithm outputs the designated cut if and only if G_2 is good.

Assuming that G_i is good, G_{i-1} is good with probability of at least

$$1 - \frac{k}{|E_i|} \geq 1 - \frac{2k}{ki} = 1 - \frac{2}{i} = \frac{i-2}{i}, \qquad \text{because of } |E_i| \geq \frac{ki}{2}.$$

Recall that the algorithm outputs the designated cut in case G_2 is good, which is true with probability of at least

$$\prod_{i=n}^{3} \left(\frac{i-2}{i} \right) = \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \ \cdots \ \cdot \frac{3}{5} \cdot \frac{2}{4} \cdot \frac{1}{3} = \frac{2}{n(n-1)} > \frac{2}{n^2}.$$

Verification of the Contract algorithm

Proof (continued).

More precisely, the lower bound $\frac{2}{n^2}$ holds because of

$$\begin{aligned}
&\mathrm{Prob}[G_2 \text{ is good}] \\
&\qquad = \mathrm{Prob}[G_2 \text{ is good } \& \ \cdots \ \& \ G_n \text{ is good}] \\
&\qquad = \prod_{i=n}^{3} \mathrm{Prob}[G_{i-1} \text{ is good} | G_i \text{ is good } \& \ \cdots \ \& \ G_n \text{ is good}] \\
&\qquad = \prod_{i=n}^{3} \mathrm{Prob}[G_{i-1} \text{ is good} | G_i \text{ is good}] = \prod_{i=n}^{3} \left(\frac{i-2}{i} \right).
\end{aligned}$$

The graph G_i is good if and only if $G_i, G_{i+1}, \ldots, G_n$ are all good, hence the first and third equation follow.

The second equation holds by the chain rule formula and because G_n is always good.

The last equation holds by the discussion on the previous slide. □

Verification of the Contract algorithm

Corollary

Assume algorithm Contract *is run indepedently for tn^2 many times on a graph with n nodes. Then the probability that the minimum cut of the graph is not found is at most $\frac{1}{2^{2t}}$.*

Proof.

By Theorem 54, the probability that the minimum cut is not found is at most

$$\left(1-\frac{2}{n^2}\right)^{tn^2} = \left(1-\frac{2}{n^2}\right)^{\frac{n^2}{2}\cdot 2t} \leq \left(\frac{1}{\mathrm{e}}\right)^{2t} < \frac{1}{2^{2t}}.$$

□

Learning probably approximately correct

We want to learn an unknown subset of a base set E.

The subset to be learned is called the *target concept* K_T.

The target concept is chosen from a concept class $\mathcal{K}$ of subsets of E, i.e., $\mathcal{K}$ is a subset of the power set 2^E of E.

The data consists of pairs $(x_1, K_T(x_1)), \ldots, (x_s, K_T(x_s))$ where

- the *examples* x_j are chosen randomly and independently from E according to a probability distribution μ on E,
- the bit $K_T(x_j)$ tells whether x_j is a member of the target concept, where a one indicates membership.

In general, a *learner* for a concept class $\mathcal{K}$ maps the given data to a candidate concept in $\mathcal{K}$.

The goal of the learners considered here is to come up with a candidate concept K in $\mathcal{K}$ that does not differ too much from the target concept K_T.

Learning probably approximately correct

Learning probably approximately correct

The discrepancy between the candidate and the target concept is measured with respect to the probability distribution μ on E, more precisely, it is required for some given $\varepsilon > 0$ that

$$\mu(K_T \triangle K_A) \leq \varepsilon\ . \tag{11}$$

The learner must succeed in reaching *precision* ε as stated in (11) with probability at least $1 - \delta$ for some given *error bound* $\delta > 0$.

For the case of a deterministic learner considered here, the success probability refers to the choice of the sample $x_1, \ldots, x_s$.

It is natural to measure the discrepancy with respect to μ because it is hard to distinguish two concepts that differ only in a part of E that is less likely and hence is unlikely to be represented in the sample $x_1, \ldots, x_s$.

PAC learning

Definition PAC learner

Let E be any set and let $\mathcal{K}$ be a subset of the power set 2^E of E. The learner A *learns the concept class* $\mathcal{K}$ *probably approximately correct* (*PAC learns* $\mathcal{K}$, for short) with sample complexity s if

for every probability distribution μ on E,
for all $\varepsilon, \delta > 0$ and for $s = s(\varepsilon, \delta)$,
for all $K_T \in \mathcal{K}$,

in case $x_1, \ldots, x_s$ are randomly chosen from E such that the x_j are mutually independent and distributed according to μ, then A yields on the data

$$(x_1, K_T(x_1)), \ldots, (x_s, K_T(x_s))$$

a concept K_A in $\mathcal{K}$ such that

$$\mathrm{Prob}[\mu(K_T \triangle K_A) \leq \varepsilon] \geq 1 - \delta.$$

Variants of PAC learners

In what follows, we only consider the probability-theoretical and combinatorial aspects of PAC learning.

In particular, we do neither require that learners are effective or are even efficient, nor that the sample complexity $s = s(\varepsilon, \delta)$ obeys certain bounds in ε and δ.

There are more restricted notions of PAC learning where for example it is required in addition that

- the function from data to candidate concept provided by the learner is computable in polynomial time,
- the sample complexity is polynomial in the reciprocals of ε and δ, i.e., $s = p(1/\varepsilon, 1/\delta)$ for some polynomial p.

PAC learning of rectangles

Example Learning axis-parallel rectangles

We consider PAC learning of axis-parallel rectangles.

Let $E = \mathbb{R}^2$ be the Euclidean plane and for

$$\mathrm{R}(a, b, c, d) = \{(x, y) \in \mathbb{R}^2 \colon a \leq x \leq b \text{ and } c \leq y \leq d\}$$

let

$$\mathcal{K} = \{\mathrm{R}(a, b, c, d) \colon a, b, c, d \in \mathbb{R}\}$$

be the concept class of all axis-parallel rectangles (including some degenerated ones like lines or the empty set).

A PAC learner for rectangles

Let A be the learner that on data $((x_1, y_1), b_1), \ldots, ((x_s, y_s), b_s)$ for

$$X = \{x_i \colon b_i = 1\} \quad \text{and} \quad Y = \{y_i \colon b_i = 1\}$$

yields as a candidate concept the rectangle

$$R_A = \mathrm{R}(\min X, \max X, \min Y, \max Y)$$

in case X (and then also Y) is nonempty and, otherwise, yields the empty set.

Note that the candidate concept R_A provided by A is

simply the least rectangle that contains all positive examples in the sample,

is always a subset ot the target concept, and

is consistent with the given data in the sense that for all i we have $b_i = R_A(x_i)$ (consistent learners will be discussed below).

PAC learning of rectangles

We will argue next that A becomes a PAC learner for $\mathcal{K}$ when the sample complexity $s = s(\varepsilon, \delta)$ is chosen appropriately.

What is the required sample complexity?

Fix any probability distribution μ on E, real numbers $\varepsilon, \delta > 0$ and target concept $R_T = \mathrm{R}(a, b, c, d)$.

Case I: $\mu(R_T) \leq \varepsilon$

Since $R_A \subseteq R_T$, we have $R_T \triangle R_A = R_T \setminus R_A \subseteq R_T$, hence

$$\mu(R_T \triangle R_A) \leq \mu(R_T) \leq \varepsilon \, .$$

Case II: $\mu(R_T) > \varepsilon$

We define a new rectangle $R_0 \subseteq R_T$ by removing strips of measure about $\varepsilon/4$ on all four sides of R_T.

What is the required sample complexity? (continued)

In order to define the marginal strips R_1(*right*), R_2(*left*), R_3(*top*), and R_4(*bottom*), let

$$\begin{aligned}
z_1 &= \sup\{z\colon \mu(\mathrm{R}(z,b,c,d) \geq \varepsilon/4\},\\
z_2 &= \inf\{z\colon \mu(\mathrm{R}(a,z,c,d) \geq \varepsilon/4\},\\
z_3 &= \sup\{z\colon \mu(\mathrm{R}(a,b,z,d) \geq \varepsilon/4\},\\
z_4 &= \inf\{z\colon \mu(\mathrm{R}(a,b,c,z) \geq \varepsilon/4\},
\end{aligned}$$

$$\begin{aligned}
R_1 &= \mathrm{R}(z_1,b,c,d), & R_1^- &= R_1 \setminus \{(x,y)\colon x = z_1\},\\
R_2 &= \mathrm{R}(a,z_2,c,d), & R_2^- &= R_2 \setminus \{(x,y)\colon x = z_2\},\\
R_3 &= \mathrm{R}(a,b,z_3,d), & R_3^- &= R_3 \setminus \{(x,y)\colon y = z_3\},\\
R_4 &= \mathrm{R}(a,b,c,z_4), & R_4^- &= R_4 \setminus \{(x,y)\colon y = z_4\}.
\end{aligned}$$

What is the required sample complexity? (continued)

By construction, we then have for $i = 1, \ldots, 4$,

(i) $R_i \subseteq R_T$,
(ii) $\mu(R_i) \geq \varepsilon/4$,
(iii) $\mu(R_i^-) \leq \varepsilon/4$.

Now let $R_0 = \mathrm{R}(z_1, z_2, z_3, z_4) = R_T \setminus (R_1^- \cup R_2^- \cup R_3^- \cup R_4^-)$.

If case the sample contains at least one point in R_1 through R_4, then

$$R_0 \subseteq R_A \subseteq R_T$$

and hence

$$R_T \triangle R_A = R_T \setminus R_A \subseteq R_1^- \cup R_2^- \cup R_3^- \cup R_4^- \ ,$$

and by (iii) we obtain $\mu(R_T \triangle R_A) \leq \varepsilon$.

What is the required sample complexity? (continued)

For $i = 1, \ldots, 4$, by (ii), the rectangle R_i has probability of at least $\varepsilon/4$, hence the probability that among s examples none is in R_i is at most $(1 - \varepsilon/4)^s$. So it suffices to choose s where

$$\begin{aligned} 4(1-\varepsilon/4)^s \leq \delta &\Leftrightarrow 2 + s\log(1-\varepsilon/4) \leq \log\delta \\ &\Leftrightarrow s \geq \frac{-2+\log\delta}{\log(1-\varepsilon/4)} = \frac{2+\log 1/\delta}{-\log(1-\varepsilon/4)} \ . \end{aligned}$$

Inspection of the growth of the binary logarithm function on arguments close to 1 reveals that $x < -\log(1-x)$ holds for $x > 0$.

Applying this inequality to $x = \varepsilon/4$ shows that A becomes a PAC learner in case

$$s \geq \frac{2+\log 1/\delta}{\varepsilon/4} \ .$$

PAC learning of finite concept classes

Definition Consistent learner

Let E be a set and let $K \subseteq E$ be a concept. Then K is *consistent* with data $(x_1, b_1), \ldots, (x_s, b_s)$ if $K(x_j) = b_j$ for $j = 1, \ldots, s$.

A learner for a concept class $\mathcal{K}$ is *consistent* if for any target concept K_T in $\mathcal{K}$ and any data of the form

$$(x_1, K_T(x_1)), \ldots, (x_n, K_T(x_n))$$

the learner yields a candidate concept that is consistent with the given data.

A learner must output a candidate concept in the concept class $\mathcal{K}$ to be learned, thus a learner can never be consistent with data that is not consistent with any concept in $\mathcal{K}$.

PAC learners need not to be consistent because candidate and target concept are just required to be close but not to be the same.

PAC learning of finite concept classes

Theorem

Let E be a set and let $\mathcal{K} \subseteq 2^E$ be a finite concept class over E. Then every consistent learner for $\mathcal{K}$ is a PAC learner for $\mathcal{K}$ with sample complexity

$$s = s(\varepsilon, \delta) = \frac{\log |\mathcal{K}| + \log 1/\delta}{\varepsilon}$$

Proof.

Fix any probability measure μ on E and reals $\varepsilon, \delta > 0$.

By definition of PAC learning, it suffices to show that for any given target concept K_T, our learner outputs a candidate concept K in $\mathcal{K}$ such that $\mu(K_T \triangle K) \geq \varepsilon$ with probability of at most δ .

So it suffices to show that every individual such K is output with probability of at most $\delta/|\mathcal{K}|$, where by assumption on the learner a concept can only be output if it is consistent with the data.

PAC learning of finite concept classes

Proof. (continued)

It suffices to show that for $s = s(\varepsilon, \delta)$ as in the theorem and any target concept K_T, for any concept K in $\mathcal{K}$ such that

$$\mu(K_T \triangle K) \geq \varepsilon \tag{12}$$

the probability that a sample of size at least s will lead to data consistent with K is at most $\delta/|\mathcal{K}|$.

In general, a concept K is consistent with data for a target concept K_T if and only if none of the sample points is in $K_T \triangle K$.

So any K that satisfies (2) will be consistent with a sample of size s with probability of at most $(1-\varepsilon)^s$, hence it suffices to show

$$(1-\varepsilon)^s \leq \delta/|\mathcal{K}|\,.$$

But for s as in the proposition, this follows by essentially the same argument as in the example on learning rectangles, where the factor 2 and the term $\varepsilon/4$ are replaced by $|\mathcal{K}|$ and ε, respectively.

PAC learning and VC dimension

Definition VC dimension

Let E be a set, let $\mathcal{K} \subseteq 2^E$ be a finite concept class over E, and let $X \subseteq E$ be finite. Then X is *shattered* by the concept class $\mathcal{K}$ if

$$\{K \cap X \colon K \in \mathcal{K}\} = 2^X \, ,$$

i.e., if any subset of X can be obtained by intersecting X with a concept K in $\mathcal{K}$.

The *Vapnik-Chernovenkis dimension* or *VC dimension*, for short, of the concept class $\mathcal{K}$ is

$$\mathrm{VC}(\mathcal{K}) = \max\{|X| \colon X \subseteq E \text{ is finite and is shattered by } \mathcal{K}\} \, .$$

Examples

For finite E and $\mathcal{K} = 2^E$, we have $\mathrm{VC}(\mathcal{K}) = |E|$.

For $E = \mathbb{R}$ and $\mathcal{K}$ the set of finite intervals, we have $\mathrm{VC}(\mathcal{K}) = 2$.

PAC learning and VC dimension

Example Axis-parallel rectangles

Let E be the Euclidean plane, i.e., $E = \mathbb{R}^2$, and let $\mathcal{K}$ be equal to the set of axis-parallel rectangles.

The set $\{(0,1),(1,0),(-1,0),(0,-1)\}$ has size four and is shattered by $\mathcal{K}$, hence $\mathrm{VC}(\mathcal{K}) \geq 4$.

On the other hand, we have $\mathrm{VC}(\mathcal{K}) < 5$.

For a proof, fix any five points $(x_1, y_1), \ldots, \mathit{and}\ (x_5, y_5)$ and let

$$\begin{aligned} x_{\min} &= \min\{x_1, \ldots, x_5\}, & x_{\max} &= \max\{x_1, \ldots, x_5\}, \\ y_{\min} &= \min\{y_1, \ldots, y_5\}, & y_{\max} &= \max\{y_1, \ldots, y_5\}. \end{aligned}$$

Now choose four points in this set such that $x_{\min}$ and $x_{\max}$ occur among the x-coordinates of these four points, and similarly, $y_{\min}$ and $y_{\max}$ occur among the y-coordinates.
Then any axis-parallel rectangle that contains these four points must also contain the remaining fifth point.

PAC learning and VC dimension

The result that every consistent learner of a finite concept class is already a PAC learner for this concept class can be extended to concept classes that have finite VC dimension.

Theorem

Let E be a set and let $\mathcal{K} \subseteq 2^E$ be a finite concept class over E with finite Vapnik-Chernovenkis dimension $\mathrm{VC}(\mathcal{K}) = d$.
Then every consistent learner for $\mathcal{K}$ is a PAC learner for $\mathcal{K}$ with sample complexity

$$s = s(\varepsilon, \delta) = \mathrm{O}\left(\frac{d \log 1/\min\{\varepsilon, \delta\}}{\varepsilon} \right) ,$$

where the constant hidden in the O-notation does depend neither on $\mathcal{K}$ nor on ε or δ.

See Rivest, *Lecture Notes on Machine Learning*, 1994, or Blumer, Ehrenfeucht, Haussler, and Warmuth, *Learnability and the Vapnik-Chervonenkis-dimension*, JACM 36(4):929–965, 1989.

The Markov inequality

The next two theorems extend by essentially the same proofs to not necessarily discrete random variables.

Theorem Markov inequality

Let X be a discrete random variable that attains only nonnegative values and such that $\mathbf{E}[X]$ exists. Then for all positive $r \in \mathbb{R}$,

$$\mathrm{Prob}[X \geq r] \leq \frac{\mathbf{E}[X]}{r} .$$

Equivalently, if in addition $\mathbf{E}[X] > 0$, then for all positive $r \in \mathbb{R}$,

$$\mathrm{Prob}[X \geq r\,\mathbf{E}[X]] \leq \frac{1}{r} .$$

Proof.

Let X_r be the indicator variable for the event that X is at least r, i.e., $X_r = 0$ in case $X < r$ and, otherwise, $X_r = 1$. So $X_r \leq X/r$, and hence

$$\mathrm{Prob}[X \geq r] = \mathbf{E}[X_r] \leq \mathbf{E}[X/r] = \frac{\mathbf{E}[X]}{r} .$$

The Chebyshev inequality

Definition Variance

Let X be a discrete random variable such that $\mathbf{E}[X]$ exists.
The *variance* $\mathbf{Var}[X]$ of X is defined as

$$\mathbf{Var}[X] = \mathbf{E}\left[(X - \mathbf{E}[X])^2\right] .$$

Remark

The square root of the variance of a random variable X is a very rough measure how much the values of X will deviate from $\mathbf{E}[X]$, though only rarely $\sqrt{\mathbf{Var}[X]}$ will indeed be equal to the expected deviation.

Equivalently, the variance of a random variable X is a rough measure for how much X is *concentrated* around $\mathbf{E}[X]$, where a small variance means highly concentrated.

The Chebyshev inequality

Theorem Chebyshev inequality

Let X be a random variable where $\mathbf{E}[X]$ and $\mathbf{Var}[X]$ both exist. Then it holds for all positive $r \in \mathbb{R}$,

$$\mathrm{Prob}[|X - \mathbf{E}[X]| \geq r] \leq \frac{\mathbf{Var}[X]}{r^2} .$$

Equivalently, in case in addition $\mathbf{Var}[X] > 0$, for all positive $r \in \mathbb{R}$,

$$\mathrm{Prob}\left[|X - \mathbf{E}[X]| \geq r\sqrt{\mathbf{Var}[X]}\right] \leq \frac{1}{r^2} .$$

Proof.

It suffices to observe that

$$\mathrm{Prob}\left[|X - \mathbf{E}[X]| \geq r\sqrt{\mathbf{Var}[X]}\right] = \mathrm{Prob}\left[|X - \mathbf{E}[X]|^2 \geq r^2\mathbf{Var}[X]\right] ,$$

and to apply the Markov inequality to the random variable $|X - \mathbf{E}[X]|^2$, which is nonnegative and has expectation $\mathbf{Var}[X]$.

The Chernov-Hoeffding bound

Theorem Chernov-Hoeffding bound

Let $X_1, \ldots X_n$ be mutually independent $\{0,1\}$-valued random variables where $\mathrm{Prob}[X_i = 1] = p_i$ for $0 < p_i < 1$.

Furthermore, let $X = X_1 + \cdots + X_n$ and $\mu = \mathbf{E}[X] = \sum p_i$.

a) For all δ where $0 < \delta \leq 1$ it holds that

$$\mathrm{Prob}[X \leq (1-\delta)\mu] < \mathrm{e}^{-\frac{\delta}{2}\mu}\,.$$

b) For all $\delta > 0$ it holds that

$$\mathrm{Prob}[X \geq (1+\delta)\mu] < \left(\frac{\mathrm{e}^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu}\,.$$

(Here $\mathrm{e} = 2.71\ldots$ is Euler's number.)

The Chernov-Hoeffding bound

Majority vote

Suppose for some $p > 1/2$ and for any input, a randomized algorithm decides with probability at least p correctly whether any given input has a certain property or not.

Applying a *majority vote* to independent iterations of the algorithms means that the algorithm is run several times on the same input using mutually independent random sources, and the output is equal to the majority of the outcomes of the iterations

Corollary Probability amplification

Let $X_1, \ldots X_n$ be mutually independent $\{0,1\}$-valued random variables where $\mathrm{Prob}[X_i = 1] = 1/2 + \varepsilon$ for some real $\varepsilon > 0$.

Then for $X = X_1 + \cdots + X_n$, we have

$$\mathrm{Prob}\left[X \leq \frac{n}{2}\right] < 2^{-\frac{\varepsilon}{2}n} .$$

The Chernov-Hoeffding bound

Proof.

Let $\mu = \mathbf{E}[X] = (1/2 + \varepsilon)n$. For any δ where $0 < \delta < 1$ and

$$\frac{n}{2} \leq (1 - \delta)\mu = (1 - \delta)(1/2 + \varepsilon)n \, , \tag{13}$$

by the Chernov-Hoeffding bound we have

$$\mathrm{Prob}[X \leq \frac{n}{2}] \leq \mathrm{Prob}[X \leq (1 - \delta)\mu] < \mathrm{e}^{-\frac{\delta}{2}\mu}$$

Elementary rearrangements show that (13) becomes true for

$$\delta = \frac{2\varepsilon}{1 + 2\varepsilon} \, ,$$

and plugging this value of δ into the bounds derived above yields

$$\mathrm{Prob}[X \leq \frac{n}{2}] \leq \mathrm{e}^{-\frac{\delta}{2}\mu} = \mathrm{e}^{-\frac{\varepsilon}{1+2\varepsilon}(\frac{1}{2}+\varepsilon)n} = \mathrm{e}^{-\frac{\varepsilon}{2}n} < 2^{-\frac{\varepsilon}{2}n} \, . \qquad \square$$

Appendix: Proof of the Chernov-Hoeffding bound

In the proof, we will use the standard properties of the exponential function $y \mapsto \mathrm{e}^y$, where $\mathrm{e} = 2.71\ldots$ is Euler's number.

In particular, the exponential function is strictly increasing and is equal to its own derivative. Furthermore, we have

$$1 + y < \mathrm{e}^y \quad \text{for any real number } y \neq 0\,, \tag{14}$$

Inequality (14) simply holds true because its two sides are the same at $y = 0$, and the right-hand side grows strictly more slowly than the left-hand side on the negative reals, but grows strictly faster on the positive reals.

Proof.

Let $X_1, \ldots X_n$ be mutually independent $\{0,1\}$-valued random variables where $\mathrm{Prob}[X_i = 1] = p_i$ for $0 < p_i < 1$.

Let $X = X_1 + \cdots + X_n$ and $\mu = \mathbf{E}\,[X] = \sum p_i$.

Appendix: Proof of the Chernov-Hoeffding bound

Proof (continued).

Proof of part a: For given δ where $0 < \delta \leq 1$, we have to show that

$$\mathrm{Prob}[X \leq (1-\delta)\mu] < \mathrm{e}^{-\frac{\delta}{2}\mu} .$$

In case $\delta = 1$ this inqality holds true, as we get by using (14)

$$\mathrm{Prob}[X \leq 0] = \prod (1 - p_i) \leq \prod \overbrace{\mathrm{e}^{-p_i}}^{<1} < \prod \mathrm{e}^{-\frac{p_i}{2}} = \mathrm{e}^{-\frac{\delta}{2}\mu}$$

So we can assume $\delta < 1$. Now for any real $t > 0$ it holds that

$$\begin{aligned} \mathrm{Prob}[X \leq (1-\delta)\mu] &= \mathrm{Prob}[-tX \geq -t(1-\delta)\mu] \\ &= \mathrm{Prob}[\mathrm{e}^{-tX} \geq \mathrm{e}^{-t(1-\delta)\mu}] \leq \frac{\mathbf{E}\left[\mathrm{e}^{-tX}\right]}{\mathrm{e}^{-t(1-\delta)\mu}} , \end{aligned}$$

where the two last relations follow because the exponential function is strictly increasing and by the Markov inequality, respectively.

Appendix: Proof of the Chernov-Hoeffding bound

Proof (continued).

Concerning the numerator in the last term above, we have

$$\mathbf{E}\left[\mathrm{e}^{-tX}\right] = \mathbf{E}\left[\mathrm{e}^{-t\sum_{i=1}^{n} X_i}\right] = \mathbf{E}\left[\prod_{i=1}^{n} \mathrm{e}^{-tX_i}\right] = \prod_{i=1}^{n} \mathbf{E}\left[\mathrm{e}^{-tX_i}\right] ,$$

where the last equation holds because the X_i are mutually independent. Furthermore, using (14) again, we have

$$\mathbf{E}\left[\mathrm{e}^{-tX_i}\right] = p_i \mathrm{e}^{-t} + (1 - p_i) \cdot 1 = 1 + p_i(\mathrm{e}^{-t} - 1) \leq \mathrm{e}^{p_i(\mathrm{e}^{-t}-1)} .$$

Together, the two last chains of relations yield

$$\mathbf{E}\left[\mathrm{e}^{-tX}\right] \leq \prod_{i=1}^{n} \mathrm{e}^{p_i(\mathrm{e}^{-t}-1)} = \mathrm{e}^{\sum_{i=1}^{n} p_i(\mathrm{e}^{-t}-1)} = \mathrm{e}^{(\mathrm{e}^{-t}-1)\mu} .$$

Appendix: Proof of the Chernov-Hoeffding bound

Proof (continued).

Putting together equations and inequalities from last two slides, we get

$$\mathrm{Prob}[X \le (1-\delta)\mu] \le \frac{\mathbf{E}\left[\mathrm{e}^{-tX}\right]}{\mathrm{e}^{-t(1-\delta)\mu}} \le \frac{\mathrm{e}^{(\mathrm{e}^{-t}-1)\mu}}{\mathrm{e}^{-t(1-\delta)\mu}} = \mathrm{e}^{(\overbrace{\mathrm{e}^{-t}-1+t(1-\delta)}^{=:b(t)})\mu}.$$

Since the exponential function is strictly increasing, the upper bound $\mathrm{e}^{b(t)}$ becomes minimum exactly for values of t such that $b(t)$ is minimum.

We have $b'(t) = -\mathrm{e}^{-t} + 1 - \delta$ and $b''(t) = \mathrm{e}^{-t}$, thus for $t > 0$ the function b attains its unique minimum at $t_0 = \ln(1/1-\delta)$.

Appendix: Proof of the Chernov-Hoeffding bound

Proof (continued).

Plugging the value t_0 into the upper bound $\mathrm{e}^{b(t)}$ yields

$$\mathrm{Prob}[X \leq (1-\delta)\mu] \leq \frac{\mathrm{e}^{-\delta\mu}}{(1-\delta)^{(1-\delta)\mu}} = \Bigg(\frac{\mathrm{e}^{-\delta}}{\underbrace{(1-\delta)^{(1-\delta)}}_{>\mathrm{e}^{-\delta+\delta^2/2}}}\Bigg)^{\mu}$$

$$< \mathrm{e}^{(-\delta+\delta-\delta/2)\mu} = \mathrm{e}^{-\frac{\delta^2}{2}\mu}\,.$$

Recall in this connection that for all δ where $|\delta| < 1$, it holds that $\ln(1-\delta) = -\delta - \frac{\delta^2}{2} - \frac{\delta^3}{3} - \ldots$, hence we have for such $\delta > 0$

$$\begin{aligned}\ln(1-\delta)^{(1-\delta)} &= (1-\delta)\ln(1-\delta) = (1-\delta)(-\delta - \frac{\delta^2}{2} - \frac{\delta^3}{3} - \ldots) \\ &= -\delta + (1-\frac{1}{2})\delta^2 + (\frac{1}{2}-\frac{1}{3})\delta^3 + \ldots > -\delta + \delta^2/2\,,\end{aligned}$$

that is $(1-\delta)^{(1-\delta)} > \mathrm{e}^{-\delta+\delta^2/2}$.

Proof (continued).

Proof of part b: The proof of part b resembles the one of part a. For given $\delta > 0$, we have to show that

$$\mathrm{Prob}[X \geq (1+\delta)\mu] < \left(\frac{\mathrm{e}^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu} .$$

For any real $t > 0$ it holds that

$$\begin{aligned}\mathrm{Prob}[X \geq (1+\delta)\mu] &= \mathrm{Prob}[tX \geq t(1+\delta)\mu] \\ &= \mathrm{Prob}[\mathrm{e}^{tX} \geq \mathrm{e}^{t(1+\delta)\mu}] \;\leq\; \frac{\mathbf{E}\left[\mathrm{e}^{tX}\right]}{\mathrm{e}^{t(1+\delta)\mu}} ,\end{aligned}$$

where the two last relations follow because the exponential function is strictly increasing and by the Markov inequality, respectively.

Proof (continued).

Similar to the proof of part a, we infer

$$\mathbf{E}\left[\mathrm{e}^{tX}\right] = \mathbf{E}\left[\mathrm{e}^{t\sum_{i=1}^{n} X_i}\right] = \mathbf{E}\left[\prod_{i=1}^{n} \mathrm{e}^{tX_i}\right] = \prod_{i=1}^{n} \mathbf{E}\left[\mathrm{e}^{tX_i}\right] ,$$

where the last equation holds because the X_i are mutually independent. Furthermore, since $p_i, t > 0$ and using (14), we have

$$\mathbf{E}\left[\mathrm{e}^{tX_i}\right] = p_i\mathrm{e}^t + (1 - p_i) \cdot 1 = 1 + p_i(\mathrm{e}^t - 1) < \mathrm{e}^{p_i(\mathrm{e}^t-1)} .$$

Together, the two last chains of relations yield

$$\mathbf{E}\left[\mathrm{e}^{tX}\right] < \prod_{i=1}^{n} \mathrm{e}^{p_i(\mathrm{e}^t-1)} = \mathrm{e}^{\sum_{i=1}^{n} p_i(\mathrm{e}^t-1)} = \mathrm{e}^{(\mathrm{e}^t-1)\mu} .$$

Appendix: Proof of the Chernov-Hoeffding bound

Proof (continued).

Putting together equations and inequalities from last two slides, we get

$$\mathrm{Prob}[X \geq (1+\delta)\mu] \leq \frac{\mathbf{E}\left[\mathrm{e}^{tX}\right]}{\mathrm{e}^{t(1+\delta)\mu}} < \frac{\mathrm{e}^{(\mathrm{e}^t-1)\mu}}{\mathrm{e}^{t(1+\delta)\mu}} = \mathrm{e}^{(\overbrace{\mathrm{e}^t-1-t(1+\delta)}^{=:b(t)})\mu}.$$

Like in the proof of part a, we obtain a minimum value of the upper bound by minimizing $b(t)$.

Now we have $b'(t) = \mathrm{e}^t - 1 - \delta$ and $b''(t) = \mathrm{e}^t$, thus for $t > 0$ the function b attains its unique minimum at $t_0 = \ln(1+\delta)$.

Plugging the value t_0 into the upper bound $\mathrm{e}^{b(t)}$ yields

$$\mathrm{Prob}[X \geq (1+\delta)\mu] < \frac{\mathrm{e}^{\delta\mu}}{(1+\delta)^{(1+\delta)\mu}} = \left(\frac{\mathrm{e}^{\delta}}{(1+\delta)^{(1+\delta)}}\right)^{\mu}. \quad \square$$

Randomized local search

Boolean formulas in conjunctive normal form

A *literal* is a Boolean variable or a negated Boolean variable.

A *(disjunctive) clause* is a disjunction of literals.

A Boolean formula is in *conjunctive normal form*, or in *CNF*, for short, if the formula is a conjunction of clauses.

A Boolean formula is in *k conjunctive normal form*, or in *k-CNF*, for short, if the formula is in CNF such that each clause contains exactly k mutually distinct variables.

There are more liberal definitions of k-CNF where, for example, it is simply required that each clause has at most k literals.

In what follows, the actual choice of the notion k-CNF does not matter too much, though by using the more restrictive notion the exposition becomes slightly cleaner.

Randomized local search

Satisfying assignments

An *assignment* for a Boolean variable over variables $Z_1, \dots, Z_n$ is a binary word of length n.

The *(hamming) distance* $\mathrm{d}(\sigma, \tau)$ of two words σ and τ of the same length is the number of positions where the words differ.

With variables $Z_1, \dots, Z_n$ are understood, an assignment of length n is identified in the natural way with a mapping that assigns to each Z_i a truth value, where the symbol 1 denotes true and 0 denotes false.

Given an appropriate assignment, a Boolean formula can be evaluated in the usual way. For example, the 3-CNF formula

$$(Z_1 \vee \neg Z_3 \vee Z_4) \wedge (\neg Z_1 \vee Z_4 \vee Z_5)$$

is made true or is *satisfied* by the assignment 10010 but not by the assignment 10000.

Randomized local search

Local search guided by unsatisfied clauses

Consider a Boolean formula φ in n variables in 3-CNF.

Suppose that φ is satisfiable and fix any satisfying assignment σ_0.

Then given any assignment σ and any clause of φ that is not made true by σ, the assignments σ and σ_0 must differ on at least one of the variables in this clause.

Consequently, when choosing a variable uniformly at random from any fixed unsatisfied clause and flipping the value of the assignment σ for this variable, with probability at least $1/3$ one obtains an assignment that is strictly closer to σ_0 than σ.

For example, when starting at an assignment σ where $\mathrm{d}(\sigma, \sigma_0) \leq j$ and iterating such random flips j times or more, one reaches σ_0 with probability of at least 3^{-j}.

Randomized local search

Algorithm $\mathrm{LocalSearch}(\varphi, \sigma, r)$ (Randomized Local Search)

```
Input:  A Boolean formula $\varphi$ in $n$ variables in 3-CNF.
        An assignment $\sigma \in \{0,1\}^n$ and a natural number $r$.
  While $r \geq 1$ and $\varphi(\sigma) = \mathrm{false}$
        Pick the least clause of $\varphi$ that is not made true by $\sigma$.
        Pick a variable in this clause uniformly at random.
        Flip the value of the assignment $\sigma$ at this variable.
        Let $r = r - 1$.
Output: The modified assignment $\sigma$.
```

Suppose φ is satisfiable and σ_0 is any fixed satisfying assignment.

The hamming distance $\mathrm{d}(\sigma, \sigma_0)$ either increases or decreases by 1 during each iteration of the while loop in algorithm $\mathrm{LocalSearch}$.

Furthermore, a decrease occurs with probability of at least $1/3$.

Randomized local search

Consider an invocation $\mathrm{LocalSearch}(\varphi, \sigma, r)$ such that φ has a satisfying assignment σ_0 that has distance $j \leq r$ from σ.

The probability of finding *some* satisfying assignment during this invocation is at least as good as reaching the origin in a

one-dimensional random walk of length r that
starts at position j and
moves left and right with probability $1/3$ and $2/3$, respectively.

Local search and random walks

In the exursus on random walks below, it is demonstrated that the origin is reached in such a random walk with probability of at least $2^{-(j+1)}$ in case $r \geq 27j$.

By invoking $\mathrm{LocalSearch}$ with $r = 27n$, the probability of finding a satisfying assignment is at least $2^{-(j+1)}$, provided that the initial assignment has distance at most j to some satisfying assignment.

Randomized local search

The probability of starting close to a satisfying assignment

In case the an assignment σ is chosen uniformly at random from all assignments in $\{0,1\}^n$, what is the probability for choosing a assignment that has distance j to some fixed satisfying assignment?

Assuming that φ is satisfiable, fix any satisfying assignment σ_0.

Choosing the assigment σ uniformly at random amounts to tossing a fair coin for each bit of σ.

Equivalently, one could toss a fair coin for each position in order to decide whether σ should agree with the corresponding bit of σ_0.

Accordingly, the number of positions where the randomly chosen assignment σ and the fixed assignment σ_0 differ will be distributed according to a Binomial distribution with parameter $1/2$, i.e.,

$$\mathrm{Prob}[\mathrm{d}(\sigma,\sigma_0)=j] = \binom{n}{j}\left(\frac{1}{2}\right)^{j}\left(\frac{1}{2}\right)^{n-j} = \binom{n}{j}\left(\frac{1}{2}\right)^{n} .$$

Success probability of randomized local search

What is the probability of finding a satisfying assignment on an invocation $\mathrm{LocalSearch}(\varphi, \sigma, 27n)$, where φ is satisfiable and σ is chosen uniformly at random?

By putting together the probabilities for an initial assignment to be at distance j from some fixed satisfying assignment and the lower bound of $p_j/2 = 2^{-(j+1)}$ on the success probability in this case, the probability of finding a satisfying assignment is at least

$$\sum_{j=0}^{n} \mathrm{Prob}[\mathrm{d}(\sigma, \sigma_0) = j] \frac{p_j}{2} = \sum_{j=0}^{n} \binom{n}{j} \frac{1}{2^n} \frac{1}{2^{j+1}} = \frac{1}{2^{n+1}} \sum_{j=0}^{n} \binom{n}{j} \frac{1}{2^j}$$

$$= \frac{1}{2^{n+1}} \left(1 + \frac{1}{2}\right)^n = \frac{1}{2} \left(\frac{3}{4}\right)^n .$$

Randomized local search

Independent trials

If a chance experiment with probability $1/n$ for success is repeated independently n times, the probability of obtaining at least one success is at least 0.6.

The probability of obtaining no success is equal to $(1 - 1/n)^n$, which goes increasingly to $1/\mathrm{e} < 0.4$ when n goes to infinity.

Iterated local searches

For a satisfiable formula φ and an assignment σ chosen uniformly at random, an invocation $\mathrm{LocalSearch}(\varphi, \sigma, 27n)$ will return a satisfying assignment with probability of at least $(3/4)^n/2$.

When invoking the algorithm $2(4/3)^n$ times independently, the probability of finding a satisfying assignment is at least 0.6 in case the input formula is indeed satisfiable and, otherwise, is 0.

Randomized local search

By the preceding discussion, the following algorithm, when applied to a satisfiable Boolean formula, will return a satisfying assignment with probability at least 0.6.

The running time of the algorithm is a polynomial in n times the number of invocations of $\mathrm{LocalSearch}$, hence is in $\mathrm{O}(1.34^n)$.

Algorithm $\mathrm{LocalSearch}(\varphi)$ (Iterated Local Search)

```
Input:  A Boolean formula $\varphi$ in $n$ variables in 3-CNF.
   Let $i = 1$ and $\sigma = 0^n$.
   While $i \leq 2 \cdot \left(\frac{4}{3}\right)^n$ and $\varphi(\sigma) = \mathrm{false}$
          Pick an assignment $\sigma$ uniformly at random from $\{0, 1\}^n$.
          Let $\sigma := \mathrm{LocalSearch}(\varphi, \sigma, 27n)$.
          Let $i := i + 1$.
Output: The current assignment $\sigma$.
```

Excursus on random walks

Random walks

Consider the chance experiment where

> a token moves at random on the x-axis such that at times $t = 0, 1, \ldots$ the token is at position X_t in $\{0, 1, \ldots\}$,
> if $X_t = 0$, then $X_t = X_{t+1} = X_{t+2} = \ldots$,
> if $X_t \neq 0$, then X_{t+1} is equal to $X_t - 1$ or $X_t + 1$ with probability α and $1 - \alpha$, respectively.

Such a sequence of random variables $X_0, X_1, \ldots$ is called a *one-dimensional random walk* with the origin as *absorbing point* and uniform *transition probabilities* α and $1 - \alpha$.

The sequence $X_0, X_1, \ldots$ forms a *Markov chain* because

the X_i satisfy the *Markov property*, i.e., for all t the probability distribution of X_{t+1} is determined by the value of X_t alone, in the sense that the probability distribution of X_{t+1} conditioned on X_t and on $X_1, \ldots, X_t$ is the same.

Excursus on random walks

The probability of reaching the origin

For a random walk $X_0, X_1, \dots$ as above and with the value of the parameter α understood, let $p_j = p_j(\alpha)$ denote the probability that the token starting at position j eventually reaches the origin, i.e., the probability that $X_i = 0$ for some $i \geq 0$, given that $X_0 = j$.

Calculating the probabilities p_j

For all $j \geq 0$, it holds that $p_j = p_1^j$.

Trivially, we have $p_0 = 1 = p_1^0$ and $p_1 = p_1^1$.

For $j \geq 2$, reaching the origin from position j amounts to

first reaching position $j-1$, starting at position j,
then reaching position $j-2$, starting at position $j-1$,
and so on until the origin is reached,

where the corresponding events all have probability p_1 and are mutually independent, hence $p_j = p_1^j$.

Calculating the probabilities p_j

In case $\alpha = 1$, we must have $p_j = 1$ for all j, so assume $\alpha < 1$.

For all $j \geq 1$ we have $p_j = \alpha p_{j-1} + (1-\alpha)p_{j+1}$.

For the special case $j = 1$, this yields

$$p_1 = \alpha p_0 + (1-\alpha)p_2 = \alpha + (1-\alpha)p_1^2\ , \text{ which is equivalent to}$$
$$0 = p_1^2 - \frac{1}{1-\alpha}p_1 + \frac{\alpha}{1-\alpha} = (p_1 - 1)(p_1 - \frac{\alpha}{1-\alpha})\ .$$

Case $\alpha \geq 1/2$**.**

For such α, we have $\alpha/(1-\alpha) \geq 1$, hence the second solution for p_1 is at least 1, i.e., either agrees with the first solution or cannot be a probability.

As a consequence, the only admissible solution for the value of the probabillity p_1 is $p_1 = 1$.

By $p_j = p_1^j$, for $\alpha \geq 1/2$ the p_j are all equal to 1.

Calculating the probabilities p_j

The random walk where $\alpha = 1/2$ is called the *symmetric random walk*. The fact that a symmetric random walk starting at any position j with probability 1 will eventually reach the origin is sometimes referred to as gambler's ruin.

Case $\alpha < 1/2$.

We will argue in a minute that in this case the probability p_1 must differ from 1, hence the only admissible solution is

$$p_0 = 1, \quad p_1 = \frac{\alpha}{1-\alpha}, \text{ and } \quad p_j = p_1^j = \left(\frac{\alpha}{1-\alpha}\right)^j \text{ for all } j \geq 0 \,.$$

In case $\alpha = 1/3$ most relevant to us, we obtain $p_j = 1/2^j$

More general, in case $\alpha = 1/k$ for some $k \geq 1$, we have

$$\frac{\alpha}{1-\alpha} = \frac{1/k}{(k-1)/k} = \frac{1}{k-1}, \text{ hence } \quad p_j = \frac{1}{(k-1)^j} \,.$$

Calculating the probabilities p_j

Case $\alpha < 1/2$ (continued). It remains to show that for $\alpha < 1/2$ the probability p_1, and hence $p_2, p_3, \ldots$ are strictly smaller than 1.

If we let R_i be the indicator variable of move i being to the right, then $\mathrm{Prob}[R_i = 1] = 1 - \alpha = 1/2 + \varepsilon$ for some $\varepsilon > 0$.

Reaching the origin eventually implies that for some $t \geq j$ the origin is reachd first after exactly t moves, which in turn implies that at most half of the these t moves have been to the right.

Similar to the discussion on probability amplification, the Chernov-Hoeffding bound then yields for all $j > 0$,

$$p_j \leq \sum_{t=j}^{\infty} \mathrm{Prob}\left[\sum_{i=1}^{t} R_t \leq t/2\right] < \sum_{t=j}^{\infty} (\underbrace{\mathrm{e}^{-\frac{\varepsilon}{2}}}_{<1})^t < \infty \ ,$$

So the p_j are smaller than the tail sums of a converging series, thus become arbitrarily small for large j, hence $p_1 < 1$ due to $p_j = p_1^j$.

Restricting the length of the random walk

The probabilities p_j we have just been calculated give the probability of eventually reaching the origin when starting at position j.

We will see next that for all sufficiently large j, the probability of reaching the origin from position j will become only marginally smaller if the total number of moves allowed is restricted to dj for some appropriate constand d that depends on α.

We will restrict attention to fixed values $\alpha = 1/3$ and $d = 27$. The argument extends naturally to the more general case $\alpha = 1/k$.

For $\alpha = 1/3$, assume that the token does not reach the origin within the first $27j$ moves, and consider the position reached after $27j$ moves. The probability that this position is at most, say, $4j$, is very small. If, on the other hand, the position is larger than $4j$, the probability of still reaching the origin from there is much smaller than for the initial position j.

Restricting the length of the random walk

For a given initial position $X_0 = j$, we define the following events

- E the token reaches the origin eventually,
- F the token reaches the origin within the first $27j$ moves,
- S the token reaches the origin after strictly more than $27j$ moves and $X_{27j} \leq 4j$
- L the token reaches the origin after strictly more than $27j$ moves and $X_{27j} > 4j$.

The last three events partition the first one, hence we have

$$p_j = \mathrm{Prob}[E] = \mathrm{Prob}[F] + \mathrm{Prob}[S] + \mathrm{Prob}[L] \ .$$

We will argue next that $\mathrm{Prob}[S]$ and $\mathrm{Prob}[L]$ are rather small compared to $\mathrm{Prob}[E]$, and that hence $\mathrm{Prob}[F] \geq \mathrm{Prob}[E]/2$.

Restricting the length of the random walk

The event L occurs if and only if

event E occurs, i.e., the origin is eventually reached, and $X_{27j} > 4j$ (note that this implies $X_i > 0$ for all $i \leq 27j$).

Accordingly, we have

$$\begin{aligned}\mathrm{Prob}[L] &= \mathrm{Prob}[E \& X_{27j} > 4j] \\ &= \underbrace{\mathrm{Prob}[E | X_{27j} > 4j]}_{\leq p_{4j}} \underbrace{\mathrm{Prob}[X_{27j} > 4j]}_{\leq 1} \leq p_{4j} = \frac{p_j}{2^{3j}} \,,\end{aligned}$$

where the last inequality holds due to $p_i = 1/2^i$.

Restricting the length of the random walk

In order to bound the probability of S, recall that the event S occurs if and only if the origin is reached but not during the first $27j$ moves *and* $X_{27j} \leq 4j$. Let $\widetilde{S}$ be the event that the origin is not reached during the first $27j$ moves and $X_{27j} \leq 4j$. Then S is a proper subset of $\widetilde{S}$, hence S is less likely than $\widetilde{S}$.

Consider a variant of the random walk where the origin is no longer absorbing, i.e., whenever the token reaches the origin or a negative position, it continues moving left or right with the given probabilities.

Let X'_t be the position of the token after t moves when starting the new random walk at position j and let S' be the event $X'_{27j} \leq 4j$.

The event $\widetilde{S}$ is less likely than S' since for each walk in $\widetilde{S}$ there is a unique corresponding walk in S', whereas S' contains some additional walks that reach the origin during the first $27j$ moves at least once.

Restricting the length of the random walk

Recall that the event S' occurs if and only if $X'_{27j} \leq 4j$.

So S' occurs if and only if among the first $27j$ moves there are at most $15j$ moves to the right.

The expected number of moves to the right is $27j\frac{2}{3} = 18j$.

In order to apply the Chernov-Hoeffding bound, let R_i be the indicator variable of move i being to the right, i.e., $R_i = 1$ in case $X_i - X_{i-1} = 1$ and $R_i = 0$, otherwise.

Let $R = R_1 + \cdots + R_{27j}$ and observe that $\mathbf{E}[R] = 27j\frac{2}{3} = 18j$.

For $\delta = 1/6$, the Chernov-Hoeffding bound yields

$$\begin{aligned}\mathrm{Prob}[S'] &= \mathrm{Prob}[X'_{27j} \leq 4j] = \mathrm{Prob}[R \leq 15j] \\ &= \mathrm{Prob}[R \leq (1-\delta)\mathbf{E}[R]] \leq \mathrm{e}^{-\frac{\delta}{2}\mathbf{E}[R]} = \mathrm{e}^{-\frac{3}{2}j} < 2^{-2j} = \frac{p_j}{2^j},\end{aligned}$$

where the last inequality holds because of $\log \mathrm{e} > 1.44 > 4/3$.

Exhaustive local search

An invocation $\mathrm{LocalSearchExh}(\varphi, \sigma, r)$ results in an exhaustive local search up to depth r on the same tree as for randomized local search, i.e., the tree has root σ and 3 descendants per inner node.

Algorithm $\mathrm{LocalSearchExh}(\varphi, \sigma, r)$ (Exhaustive Local Search)

Input: A Boolean formula φ in n variables in 3-CNF.
An assignment $\sigma \in \{0,1\}^n$ and a natural number r.

(The satisfying assignment found first, if any, during the recursive invocations is returned via the global variable α.)

```
If $\varphi(\alpha) = \mathrm{false}$ and $\varphi(\sigma) = \mathrm{true}$, let $\alpha := \sigma$.
If $r \geq 1$ and $\varphi(\alpha) = \mathrm{false}$
    Pick the least clause of $\varphi$ that is not made true by $\sigma$.
    For all three variables in this clause
        Let $\sigma'$ be obtained from $\sigma$ by flipping the
                                  value of $\sigma$ at this variable.
        $\mathrm{LocalSearchExh}(\varphi, \sigma', r-1)$ .
```

Exhaustive local search

Proposition Verification of $\mathrm{LocalSearchExh}$.

Let φ be a Boolean formula in n variables in 3-CNF. Let σ and σ_0 be any assignments for the variables in φ such that $\mathrm{d}(\sigma, \sigma_0) \leq r$ and σ_0 is a satisfying assignment.

Invoking $\mathrm{LocalSearchExh}(\varphi, \sigma, r)$ yields a satisfying assigment.

Proof. All recursive invocations eventually terminate because

- the parameter r is counted down,
- the conditions of the if-clauses are such that whenever a least unsatisfied clause is picked indeed σ does not make φ true.

In case α is ever set equal to a satisfying assignment, then α will never be changed afterwards and we are done, so assume otherwise.

By induction on r, we obtain a contradiction. For $r = 0$, α is set to the satisfying assignment $\sigma = \sigma_0$. For $r \geq 1$, either $\varphi(\sigma) = \mathrm{true}$ and α is set to σ, or by the induction hypothesis an invocation $\mathrm{LocalSearchExh}(\varphi, \sigma', r-1)$ finds a satisfying assignment since at least one of the assignments σ' is strictly closer to σ_0 than σ. □

Exhaustive local search

From the exhaustive version of local search the following deterministic algorithm for 3-SAT is immediate, which runs faster than an exhaustive search over all assignments.

Example

Let φ be a Boolean formula in n variables in 3-CNF. Then any assignment for φ has distance at most $n/2$ from one of the assignments 0^n and 1^n.

Accordingly, the formula φ is satisfiable if and only if a satisfying assignment is returned by at least one of the invocations

$$\mathrm{LocalSearchExh}(\varphi, 0^n, \frac{n}{2}) \quad \text{and} \quad \mathrm{LocalSearchExh}(\varphi, 1^n, \frac{n}{2})\,.$$

This yields a deterministic algorithm for 3-SAT that runs in time

$$\mathrm{poly}(n)\, 3^{\frac{n}{2}} = \mathrm{poly}(n) \left(\sqrt{3}\right)^n \leq \mathrm{poly}(n)\, 1.74^n\,.$$

Exhaustive local search

Is it possible to further optimize the radius $n/2$ in the last example?

Definition Volume of a discrete ball

The *n-dimensional ball* with center $\sigma \in \{0,1\}^n$ and radius r is

$$\mathrm{B}(\sigma, n, r) = \{\tau \in \{0,1\}^n \colon \mathrm{d}(\sigma, \tau) \leq r\}$$

Since by symmetry the volume of an n-dimensional ball with radius r does not depend on the center σ, we denote this volume by

$$\mathrm{vol}(n, r) = |\mathrm{B}(0^n, n, r)| \,.$$

The members of the ball $\mathrm{B}(0^n, n, r)$ can be uniquely described by the set of $j \leq r$ positions where they differ from 0^n, i.e., where they have a 1. Consequently, the exact volume of the ball $\mathrm{B}(0^n, n, r)$ is

$$\mathrm{vol}(n, r) = \sum_{j=0}^{r} \binom{n}{j} \,.$$

however, the expression on the right-hand side is hard to work with.

Exhaustive local search

Definition Binary entropy function

The volume of a discrete ball can be approximated by using the *binary entropy function* $\mathcal{H}$ defined by

$$\begin{aligned} \mathcal{H}\colon (0,1) &\to [0,1] \\ \rho &\mapsto -\rho\log\rho - (1-\rho)\log(1-\rho) \ . \end{aligned}$$

Proposition Approximate volume of a ball

In order to approximate the volume of the n-dimensional ball with radius r let $\rho = r/n$ and $v(n,r) = 2^{\mathcal{H}(\rho)n}$.

Then for all sufficiently large n and for $r \leq n/2$ it holds that

$$\frac{v(n,r)}{n} \leq \mathrm{vol}(n,r) \leq v(n,r) \ .$$

Exhaustive local search

Comparing exhaustive local search to exhaustive search

Any invocation $\mathrm{LocalSearchExh}(\varphi, \sigma, r)$ where φ is a Boolean formula in k-CNF will find a satisfying assignment for φ if and only if there is a satisfying assignment in the ball of radius r around σ.

So $\mathrm{LocalSearchExh}(\varphi, \sigma, r)$ checks whether the set $\mathrm{B}(\sigma, n, r)$ of size $\mathrm{vol}(n, r)$ contains a satisfying assignment, by searching exhaustively a tree that has at most

$$\mathrm{a}_k(n, r) := 2k^r \geq k^0 + k^1 + \cdots + k^r$$

many nodes (observe that not all leaves need to have depth k).

The advantage of exhaustive local search to depth r over an exhaustive search over all assignments in $\mathrm{vol}(n, r)$ comes from the fact that for appropriately chosen r, we have $\mathrm{a}_k(n, r) < \mathrm{vol}(n, r)$.

In the example above, for $k = 3$ and $r = n/2$ we had

$$\mathrm{a}_k(n, r) \leq 2 \cdot 1.74^n \quad \text{and} \quad \mathrm{vol}(n, r) \sim 2^{n-1} .$$

Optimum ball size for exhaustive local search

For which choice of r will the ratio of the size of the search tree of depth r and a ball of radius r be minimum?

We formulate the minimization in terms of the parameter $\rho = r/n$.

Approximating the size of the search tree and the volume of the corresponding ball by the already derived upper and lower bound, respectively, the ratio is bounded from above by

$$\frac{\mathrm{a}_k(n,r)}{\frac{1}{n}v(n,r)} = \frac{2nk^r}{2^{\mathcal{H}(\rho)n}} = 2n \cdot 2^{(\rho \log k - \mathcal{H}(\rho))n} \,.$$

The first derivative of $\mathcal{H}(\rho)$ is $-\log(\frac{\rho}{1-\rho})$, and accordingly the exponent is minimized for ρ such that

$$\log k + \log(\frac{\rho}{1-\rho}) = 0 \quad \Leftrightarrow \quad \frac{\rho}{1-\rho} = \frac{1}{k} \quad \Leftrightarrow \quad \rho = \frac{1}{k+1} \,.$$

How big are the benefits from the optimum ball size?

Plugging the optimum value $\rho = \frac{1}{k+1}$, i.e., $r = \frac{n}{k+1}$, into the upper bound for the ratio of tree size and ball volume yields

$$\frac{\mathrm{a}_k(n,r)}{\frac{1}{n}v(n,r)} = \frac{2nk^r}{2^{\mathcal{H}(\rho)n}} = 2n \cdot \left(\frac{k^{\frac{1}{k+1}}}{2^{\mathcal{H}(\frac{1}{k+1})}} \right)^n = 2n \cdot \left(\frac{k}{k+1} \right)^n ,$$

because of

$$\begin{aligned} \frac{k^{\frac{1}{k+1}}}{2^{\mathcal{H}(\frac{1}{k+1})}} &= k^{\frac{1}{k+1}} \cdot 2^{\frac{1}{k+1} \log \frac{1}{k+1} + \frac{k}{k+1} \log \frac{k}{k+1}} \\ &= k^{\frac{1}{k+1}} \cdot \left(\frac{1}{k+1} \right)^{\frac{1}{k+1}} \cdot \left(\frac{k}{k+1} \right)^{\frac{k}{k+1}} \\ &= \left(\frac{k}{k+1} \right)^{\frac{1}{k+1}} \cdot \left(\frac{k}{k+1} \right)^{\frac{k}{k+1}} = \frac{k}{k+1} . \end{aligned}$$

Exhaustive local search starting at a random assignment

Let φ be a Boolean formula in n variables in k-CNF, let σ_0 be some fixed assignment for φ, and let $r \leq n$.

If we choose an assignment σ uniformly at random from the 2^n assignments for φ, then the three following events are all the same

σ is in the ball $\mathrm{B}(\sigma_0, n, r)$,
the distance $\mathrm{d}(\sigma_0, \sigma)$ is at most r,
σ_0 is in the ball $\mathrm{B}(\sigma, n, r)$,

hence all three share the probability $\mathrm{vol}(n, r)/2^n$ of the first one.

As a consequence, in case φ is indeed satisfiable, a single invocation $\mathrm{LocalSearchExh}(\varphi, \sigma, r)$ of exhaustive local search where the initial assignment σ is chosen uniformly at random will find a satisfying assignment with probability of at least

$$\frac{\mathrm{vol}(n, r)}{2^n} .$$

Exhaustive local search

A randomized algorithm based on exhaustive local search

Let φ be a satisfiable Boolean formula in n variables in k-CNF.

In case for some natural number t exhaustive local search is invoked $t2^n/\mathrm{vol}(n,r))$ times where the initial assignments are chosen independently and uniformly at random, then the probability of not finding any satisfying assignment is at most

$$\left(1 - \frac{\mathrm{vol}(n,r)}{2^n}\right)^{\frac{t2^n}{\mathrm{vol}(n,r)}} \leq \mathrm{e}^{-t} < 2^{-t} \, .$$

Trivially, this error bound remains valid with a larger number of independent invocations of exhaustive local search, and we will use the larger but easier to compute number of invocations equal to

$$t\frac{n2^n}{v(n,r)} = \frac{t2^n}{\frac{1}{n}v(n,r)} \geq \frac{t2^n}{\mathrm{vol}(n,r)} \, .$$

A randomized algorithm based on exhaustive local search

The randomized algorithm that uses $tn2^n/v(n,r)$ indpendent invocations of exhaustive local search has probability of at most 2^{-t} of not finding any given satisfying assginment.

Using the optimum value $r = 1/(k+1)$ derived above, this randomized algorithm then explores at most a number of assignments equal to

$$tn2^n \frac{\mathrm{a}_k(n,r)}{v(n,r)} \leq tn2^n 2n \cdot \left(\frac{k}{k+1}\right)^n = 2tn^2 \cdot \left(\frac{2k}{k+1}\right)^n .$$

Example

For the case $k = 3$ and with $t = n^2$, we obtain a randomized algorithm that checks satisfiability of formulas in 3-CNF in time $\mathrm{poly}(n)1.5^n$ with probability of error of at most e^{-n^2}.

Exhaustive local search

The randomized algorithm based on exhaustive local search can be derandomized by using covering codes (see the excursus below), that is, sets of words of a given length n such that the balls around this words of radius r cover the set of all words of length n in the sense that each such word is in at least one of these balls.

In the exursus it is demonstrated that for all sufficiently large n and all $r < n/2$ there is a covering code of size

$$|C| \leq \mathrm{poly}(n)\frac{2^n}{v(n,r)}$$

such that computing the list of all words in C in lexicographical order takes time $\mathrm{poly}(n)|C|$.

Let $\rho = 1/(k+1)$ be the optimum relative radius from above and let $r = n\rho = n/(k+1)$ be the corresponding radius.

Running exhaustive local search for all balls of radius r around all the codewords in such a code then takes time of at most

$$\mathrm{poly}(n)\frac{2^n}{v(n,r)}\mathrm{a}_k(n,r) \leq \mathrm{poly}(n)2^n 2n\left(\frac{k}{k+1}\right)^n \leq \mathrm{poly}(n)\left(\frac{2k}{k+1}\right)^n .$$

Excursus on covering codes

Definition Covering code

A set C of words of length n is a *covering code of radius r* in case the union of the balls with radius r around the words w in C contain all words of length n, i.e.,

$$\{0,1\}^n = \bigcup_{w \in C} \mathrm{B}(w,n,r)\,.$$

Equivalently, a set C of words of length n is a covering code of radius r if for any word u of length n there is some word w in C such that $\mathrm{d}(w,u) \leq r$.

Note that for a covering code of radius r it is not required that the balls of radius r around the code words are mutually disjoint.

Excursus on covering codes

A covering code of codeword length n and radius r must have size of at least $2^n/\mathrm{vol}(n,r)$, and this size can be almost realized.

Theorem

For any n and $r < n$ there is a covering code C of codeword length n and radius r of size

$$|C| \leq \frac{n2^n}{\mathrm{vol}(n,r)} \leq \frac{n^2 2^n}{v(n,r)} \; .$$

Proof. First observe that the second inequality holds independently of C because $v(n,r)/n$ is a lower bound on $\mathrm{vol}(n,r)$.

Suppose a set C of size $n2^n/\mathrm{vol}(n,r)$ is picked independently and uniformly at random from the set of all words of length n.

We will argue next that with probability strictly larger than 0 the set C will be a covering code as required, which then shows that there is a covering code as required.

Excursus on covering codes

Proof (continued). Let E_u be the event that word u is in none of the balls of radius r around the $n2^n/\mathrm{vol}(n,r)$ words in C.

We have seen above that for any given word u, $\mathrm{Prob}[E_u] \leq e^{-n}$.

The sum of the error probabilities $\mathrm{Prob}[E_u]$ over the 2^n words of length n is strictly less than 1, hence with probability strictly larger than 0, the set C is a covering code. □

The size of the covering code asserted by the theorem exceeds the trivial lower bound only by a factor of n.

But the time required for computing such a covering code may be too large to obtain a reasonable derandomization of the randomized algorithm for satisfiability based on exhaustive local search.

In the remainder of this excursus, we will construct covering codes that are slightly larger than the one asserted by the theorem but which can be computed fast enough to yield a deterministic algorithm based on exhaustive local search, which has essentially the same running time as the randomized algorithm.

Excursus on covering codes

Theorem

For any $n > 0$ and radius $r \leq n/2$ there is a covering code C of codeword length n and radius r of size

$$|C| \leq n^3 \frac{2^n}{v(n,r)} \ . \tag{15}$$

such that computing the list of all words in C in lexicographical order takes time $\mathrm{poly}(n)2^{3n}$.

Proof: To ask for a covering code as required is an instance of the minimum set cover problem where

the base set A is the set of all words of length n,
all balls of radius r around the words of length n can be used in the set cover.

Excursus on covering codes

Proof (continued): The greedy algorithm for the minimum set cover problem yields a covering code of size at most $1 + \ln \mathrm{vol}(n, r) \leq n$ times the minimum size of such a code.

There is a covering code of radius r of size at most $n^2 2^n / v(n, r)$.

So the greedy algorithm yields a covering code of the required size.

The greedy algorithmus can be implemented via counters for each word w that count how many words in the ball $\mathrm{B}(w, n, r)$ are not yet covered.

Then each of the at most 2^n iterations of the while loop of the greedy algorithm requires to run twice through all words of length n, first, in order to pick the next set to be put into the cover, second, in order to update the counters.

Updating a single count can be done in time $\mathrm{poly}(n)2^n$, hence the running time per iteration is in $\mathrm{poly}(n)2^{2n}$, and the theorem follows. $\square$

Excursus on covering codes

When applying the greedy algorithm for set cover in order to obtain a covering code, the upper bound on the running time of $\mathrm{poly}(n)2^{3n}$ is still too large with respect to intended applications.

This problem can be easily overcome by working with direct sums of covering codes of small enough length.

Theorem

For all sufficiently large n and any $r < n/2$ there is a covering code C of codeword length n and radius r of size

$$|C| \leq \mathrm{poly}(n)\frac{2^n}{v(n,r)}$$

such that the list of all words in C in lexicographical order can be computed in time

$$\mathrm{poly}(n)\left(\frac{2^{\mathrm{n}}}{v(n,r)}\right).$$

Excursus on covering codes

Proof. Fix any n and $r < n/2$ and let $\rho = r/n$.

Let $d > 0$ be a constant to be specified later.

Consider a word of length n as the concatenation of d words of length $n_0 = \lfloor n/d \rfloor$ and a possibly empty suffix of length $n_0' \leq d$.

Let C_0 be a covering code of codeword length n_0 and radius $r_0 = \lfloor \rho n_0 \rfloor$ that has size at most $n_0^3 2^{n_0}/v(n_0, r_0)$ and can be computed in time $\mathrm{poly}(n_0)2^{3n_0}$.

Let C_0' be equal to $\{0,1\}^{n_0'}$.

Let C be the direct sum of d copies of C_0 and one copy of C_0', i.e., C contains exactly all concatenations of d words in C_0 plus a word in C_0' (where C_0' contains just the empty word in case $n_0' = 0$).

By construction, C is a covering code of codeword length n and radius of at most r.

Excursus on covering codes

Proof (continued). The size of C is at most

$$\left(n_0^3 \frac{2^{n_0}}{v(n_0, r_0)}\right)^d \cdot 2^{n_0'} \leq 2^{n_0'} \left(\frac{n}{d}\right)^{3d} \frac{2^{dn_0}}{2^{d\mathcal{H}(\rho)n_0}} \leq \mathrm{poly}(n) \frac{2^n}{v(n, r)} .$$

The time required to compute C is a polynomial in n times the time required to construct C_0, which is at most a polynomial in n times

$$2^{3n_0} \leq 2^{\frac{3n}{d}} \leq 2^{(1-\mathcal{H}(\rho))n} = \frac{2^n}{v(n, r)} .$$

in case we choose $3/d \leq 1 - \mathcal{H}(\rho)$. □

Excursus on the set cover problem

The weighted minimum set cover problem

Let $S_1, \dots, S_m$ be subsets of some finite set A where each set S_j has a rational weight w_j. The *weighted minimum set cover problem* asks for a subset J of the index set $\{1, \dots, m\}$ that covers A in the sense that

$$\bigcup_{j \in J} S_j = A \tag{16}$$

and, among all such subsets J, has minimum weight sum $\sum_{i \in J} w_j$.

The special case where all the weights are equal to 1 is called the *minimum set cover problem*, i.e., the minimum set cover problem asks for a set J of minimum cardinality that satisfies (16).

The minimum set cover problem is NP-hard

Unless $\mathrm{P} = \mathrm{NP}$, in deterministic polynomial time it is neither possible to solve the minimum set cover problem nor to compute the size of a minimum set cover, hence similar remarks hold for the weighted version.

Excursus on the set cover problem

The greedy algorithm $\mathrm{MinimumSetCover}$ yields an approximation to the solution of the weighted set cover problem that is provably only slightly heavier than the minimum set cover.

Algorithm $\mathrm{MinimumSetCover}(A, S_1, \ldots, S_m, w_1, \ldots, w_m)$

```
Input:  A finite set $A$, subsets $S_1, \ldots, S_m$ of $A$ where $A \subseteq \cup_{j=1}^{m} S_j$,
        and rational weights $w_1, \ldots, w_m$.
  Let $J := \emptyset$, $S := \emptyset$.
  While $A \neq S$ (i.e., while $S$ is not a set cover)
      For all $j \in \{1, \ldots, m\}$ where $S_j \not\subseteq S$ let $\gamma_j := \frac{w_j}{|S_j \setminus S|}$.
      Let $J = J \cup \{j\}$ for some $j$ where $\gamma_j$ is minimum.
      Let $S = \cup_{j \in J} S_j$.
Output: The set cover $J$.
```

Proposition

Consider an instance of the weighted minimum set cover problem where the sets S_i all have size at most n. Then on this input, the algorithm $\mathrm{MinimumSetCover}$ returns a set cover that has weight of at most $1 + \ln n$ times the weight of a minimum set cover.

Proof: We omit the easy proofs that the algorithm always terminates and returns a set cover for the given input.

In order to show that the algorithm computes a set cover of minimum weight, fix any input $A, S_1, \ldots, S_m, w_1, \ldots, w_m$.

Let J^* be any set cover of minimum weight w^* for this input.

Let J_{algo} be the set cover of weight w_{algo} returned by the algorithm.

We refer by the term stage to a single iteration of the while loop.

With a stage understood, the terms S and J refer to the corresponding values at the beginning of the stage.

Excursus on the set cover problem

For any a in A, say a becomes covered via set S_j if a is not in S at the beginning of some stage but a is in S_j and the index j is put into J during this stage.

In case a becomes covered via set S_j, then say that a contributes

$$\mathrm{contr}(a) = \frac{w_j}{|S_j \setminus S|}$$

where the value of S refers to the state when j is put into J.

When the index j is put into J during a stage, then the weight w_j of S_j is equal to the sum of the contributions of all a that become convered at that stage since

$$\sum_{a \in S_j \setminus S} \mathrm{contr}(a) = |S_j \setminus S| \frac{w_j}{|S_j \setminus S|} = w_j \; .$$

Since each a in A becomes covered via a unique set S_j, we have

$$w_{\mathrm{algo}} = \sum_{j \in J_{\mathrm{algo}}} w_j = \sum_{j \in J_{\mathrm{algo}}} \; \sum_{\substack{\{a \in A\colon\, a \text{ becomes} \\ \text{covered via } S_j\}}} \mathrm{contr}(a) = \sum_{a \in A} \mathrm{contr}(a) \; .$$

Excursus on the set cover problem

So we are done by

$$w_{\mathrm{algo}} = \sum_{a \in A} \mathrm{contr}(a) \leq \sum_{j \in J^*} \sum_{a \in S_j} \mathrm{contr}(a) \leq \sum_{j \in J^*} (1 + \ln n) w_j = (1 + \ln n) w^* \ ,$$

where the relations, from left to right, hold by the discussion on the last page, because J^* is a set cover, by the inequality to be shown next, and because w^* is the weight of the cover J^*.

It remains to show for all $j \in J^*$ that

$$\sum_{a \in S_j} \mathrm{contr}(a) \leq (1 + \ln n) w_j \ .$$

Excursus on the set cover problem

We fix any j in J^* and show $\sum_{a \in S_j} \operatorname{contr}(a) \leq (1 + \ln n) w_j$.

Let $a_s, \ldots, a_1$ be the members of S_j in the order in which they become covered during the stages of the algorithm, i.e., a_s becomes covered first, a_{s-1} becomes covered next, and so on.

If a_i becomes covered via set S_ℓ at some stage, we have

$$\operatorname{contr}(a_i) = \frac{w_\ell}{\underbrace{|S_\ell \setminus S|}_{=\gamma_\ell}} \leq \frac{w_j}{\underbrace{|S_j \setminus S|}_{=\gamma_j}} \leq \frac{w_j}{i}$$

by definition of the contribution of a, because at this stage ℓ is put into J, and, finally, by the definition of the order on S_j.

Recalling that the nth Harmonic number H satisfies

$$\mathrm{H}_n = 1 + \frac{1}{2} + \cdots \frac{1}{n} \leq 1 + \ln n\,, \qquad \text{we obtain}$$

$$\begin{aligned}\sum_{a \in S_j} \operatorname{contr}(a) &= \operatorname{contr}(a_1) + \cdots + \operatorname{contr}(a_s) \\ &\leq \frac{w_j}{1} + \cdots + \frac{w_j}{s} \leq \mathrm{H}_n w_j \leq (1 + \ln n) w_j\,. \qquad \square\end{aligned}$$

Symmetric-key and public-key cryptography

Classical cryptographic protocols depend on the use of secret information that has been agreed on in advance.

E.g., in classical protocols for sending encrypted messages over an insecure channel, sender and receiver share a *secret key* that allows

the sender to encrypt the plaintext message,
the receiver to decrypt the received ciphertext message.

One-time pad encryption scheme

$\mathbf{A}$ wants to send to $\mathbf{B}$ a message $w = w_1 \dots w_n$, $w_i \in \{0, 1\}$, over an insecure channel, which can be read by an adversary $\mathbf{E}$.

Using a secure channel, $\mathbf{A}$ and $\mathbf{B}$ exchange in advance a random word $r = r_1 \dots r_n$ obtained by independent tosses of a fair coin.

$\mathbf{A}$ sends to $\mathbf{B}$ the word $w \oplus r = (w_1 \oplus r_1) \dots (w_n \oplus r_n)$, from which $\mathbf{B}$ can recover the message $w = (w \oplus r) \oplus r$.

Symmetric-key and public-key cryptography

A cryptographic protocol is referred to as *symmetric-key protocol*, if the two parties involved share an otherwise secret key (or, which is essentially the same, in case the two parties have different keys that can be easily computed from each other).

Opposed to symmetric-key protocols are *public-key protocols*,

The introduction of public-key techniques in the 1970ies can be seen as the beginning of modern cryptography, while before, in classical cryptography, essentially all cryptographic protocols were symmetric-key protocols.

Private and public keys

In public-key protocols, each party has a *private key* and a *public key*. The private key is kept secret and is only known to the respective party, while the public key is indeed published.

Symmetric-key and public-key cryptography

In a public-key protocol for sending encrypted messages over an insecure channel from sender $\mathbf{B}$ to receiver $\mathbf{A}$,

$\mathbf{B}$ uses $\mathbf{A}$'s public key for encrypting the plaintext message.
$\mathbf{A}$ uses it private key for decrypting the ciphertext message.

Public and private key of each party must match, in fact, usually determine each other.

However, in case it were feasible to obtain the private key from the public key, the protocol would be insecure.

Hardness assumption

In public-key protocols it is assumed that even for a randomized algorithm it is not feasible to compute from a public key any relevant information about the corresponding private key.

More precisely, it is assume that any such computation either has only a negligible chance of success or runs unrealistically long.

Symmetric-key and public-key cryptography

In order to render the hardness assumption plausible, public-key protocols are based on computational tasks that are believed to be infeasible in the strong sense that even a randomized algorithm has either negligible chance of success or must run unrealistically long.

Common choices of such computational tasks are

- *factorization*, i.e., computing the prime factors of a given natural number n,
- *computing discrete logarithms*, i.e., computing the discrete logarithm $\log_d a$ for a given member a of some fixed cyclic group $(G, \cdot)$ and some fixed generator d of G.

In what follows, we will consider public-key protocols that are based on the assumption that for suitable primes p discrete logarithms in the multiplicative group $\mathbb{Z}_p^*$ are hard to compute with respect to any given generator d.

These protocols can also be based on suitable other cyclic groups.

Key agreement

For a start, we consider public-key protocols for the cryptographic primitive *key agreement*.

Consider the situation where $\mathbf{A}$ and $\mathbf{B}$ want to agree on a shared but otherwise secret key K,

> where the channel between $\mathbf{A}$ and $\mathbf{B}$ is not secure against being read by a third party $\mathbf{E}$, and
> without being able to use another, secure channel before or during the protocol,

Note that we assume that $\mathbf{E}$ may read but cannot change the data transferred over the given channel.

The third party $\mathbf{E}$ is referred to as *adversary* or *eavesdropper*, where the latter term refers in particular to somebody just reading the information transmitted over the channel.

Key agreement

The following protocol is meant to allow $\mathbf{A}$ and $\mathbf{B}$ to agree on a secret key without being able to use a secure channel before or during the protocol.

Protocol	Diffie-Hellman key agreement
Initialization	$\mathbf{A}$ and $\mathbf{B}$ select and publish a suitable prime number p and a generator d of $\mathbb{Z}_p^*$.
Step 1	$\mathbf{A}$ and $\mathbf{B}$ choose uniformly and independently (of each other) a secret natural number i_A and i_B, respectively, from $\{1, \dots, p-1\}$.
Step 2	$\mathbf{A}$ sends d^{i_A} to $\mathbf{B}$ and $\mathbf{B}$ sends d^{i_B} to $\mathbf{A}$.
Step 3	$\mathbf{A}$ computes the key $(d^{i_B})^{i_A}$. $\mathbf{B}$ computes the key $(d^{i_A})^{i_B}$.
Result	$\mathbf{A}$ and $\mathbf{B}$ know the secret key $(d^{i_B})^{i_A} = d^{i_A \cdot i_B} = (d^{i_A})^{i_B}$.

Key agreement

The following protocol is very similar to the Diffie-Hellman key agreement protocol but uses a public-key infrastructure.

Protocol	ElGamal key agreement
Initialization	Each user $\mathbf{A}$ selects an appropriate prime p, a generator d of $\mathbb{Z}_p^*$, and a random natural number i_A in $\{1, \ldots, p-1\}$; furthermore, $\mathbf{A}$ keeps i_A secret and publishes (p, d, d^{i_A}).
Step 1	$\mathbf{B}$ obtains a valid copy of $\mathbf{A}$'s public key (p, d, d^{i_A}). $\mathbf{B}$ chooses uniformly at random a secret natural number i_B in $\{1, \ldots, p-1\}$. $\mathbf{B}$ sends d^{i_B} to $\mathbf{A}$ and computes $(d^{i_A})^{i_B}$.
Step 2	$\mathbf{A}$ computes the key $(d^{i_B})^{i_A}$.
Result	$\mathbf{A}$ and $\mathbf{B}$ know the secret key $(d^{i_B})^{i_A} = d^{i_A \cdot i_B} = (d^{i_A})^{i_B}$

Public-key encryption

The public-key infrastructure from the ElGamal key agreement protocol can also be used for directly sending encrypted messages.

Protocol	ElGamal public-key encryption
Goal	$\mathbf{B}$ wants to send to $\mathbf{A}$ a message m in $\{1, \ldots, p-1\}$ over an insecure channel.
Initialization	Each user $\mathbf{A}$ selects an appropriate prime p, a generator d of $\mathbb{Z}_p^*$, and a random natural number i_A in $\{1, \ldots, p-1\}$; furthermore, $\mathbf{A}$ keeps i_A secret and publishes (p, d, d^{i_A}).
Step 1	$\mathbf{B}$ obtains a valid copy of $\mathbf{A}$'s public key (p, d, d^{i_A}). $\mathbf{B}$ chooses uniformly at random a secret natural number i_B in $\{1, \ldots, p-1\}$, computes the values $m \cdot (d^{i_A})^{i_B}$ and d^{i_B}, and sends both values to $\mathbf{A}$.
Step 2	$\mathbf{A}$ computes $m = m \cdot \underbrace{(d^{i_B})^{i_A}}_{=(d^{i_A})^{i_B}} (d^{i_B})^{(p-1)-i_A}$.
Result	$\mathbf{B}$ has securely sent the message m to $\mathbf{A}$.

Excursus on cyclic groups and discrete logarithms

Recall that a group $(G, \circ)$ is a set G with a binary operation $\circ$ on G such that $\circ$ is associative, there is a neutral element e and every $a \in G$ has an inverse a^{-1}, i.e., $a \circ a^{-1} = e$.

Definition Cyclic group

A member d of a finite group $(G, \circ)$ is a *generator* of G in case G can be written as $\{d, d^1, d^2, \ldots, d^{|G|}\}$.
A finite group $(G, \circ)$ is *cyclic* if it contains a generator.

The infinite group $(\mathbb{Z}, +)$ of integers under addition is also called cyclic because $\mathbb{Z}$ can be written as $\{\ldots, 1^{-2}, 1^{-1}, 1^{-0}, 1^1, 1^2, \ldots\}$, where 1^{-k} is equal to $-k$ and 1^k is equal to k. Up to isomorphism, this is the only infinite cyclic group.

Remark

For any finite group $(G, \circ)$ and any $a \in G$ we have $a^{|G|} = e$.

Accordingly, for a cyclic finite group $(G, \circ)$ with generator d, for every $i \in \{0, \ldots, |G|\}$ it holds that $(d^i)^{-1} = d^{|G|-i}$.

Excursus on cyclic groups and discrete logarithms

Standard examples of cyclic groups are given by $\mathbb{Z}_n^*$, the multiplicative group of residues modulo a natural number n that are units, i.e., that have a multiplicative inverse.

For a prime p, all residues except $\overline{0} = \overline{p}$ are units, hence we have

$$\mathbb{Z}_p^* = \{\overline{1}, \ldots, \overline{p-1}\}.$$

In particular, $|\mathbb{Z}_n^*| = p - 1$ and thus $a^{p-1} = \overline{1}$ for all $a \in \mathbb{Z}_p^*$.

Definition

Given a generator d of a cyclic group G of order n and some b in G, we write $\log_d b$ for the *discrete logarithm* of b to base d, which is the unique number x in $\{0, \ldots, n-1\}$ such that $b = d^x$.

Bibliographical notes

The material in this section has been adapted from Menezes et al. [1], an extremely comprehensive and wellreadable monograph on cryptography. In August 2013, the individual chapters of the 5th printing of the handbook were available free of charge at `http://cacr.uwaterloo.ca/hac/`.

[1] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of applied cryptography*, CRC Press, 1997, xxviii+780 pages.

Zero-knowledge protocols

Problem: $\mathbf{A}$ wants to identify herself to $\mathbf{B}$.

This problem arises for example when
- $\mathbf{A}$ wants to log on to computer $\mathbf{B}$,
- $\mathbf{A}$ is a customer, $\mathbf{B}$ an internet bank.

Standard solutions

$\mathbf{A}$ identifies herself by her *secret*, e.g., by password or PIN.
In order to avoid that somebody overhearing the communication may learn about $\mathbf{A}$'s secret, one may use protocols where
- each password is used only once,
- the secret is not revealed but is just used to solve some task.

Ideal solution

Not even $\mathbf{B}$ can obtain any relevant information while communicating with $\mathbf{A}$, even if $\mathbf{B}$ deviates from the protocol.
There are such protocols, which are called
interactive proof systems with the zero-knowledge property.

Zero-knowledge protocols

Definition

Let $G = (V, E)$ be a graph. A *k-coloring* of G is a mapping

$$g : V \to \{1, \ldots, k\} \; .$$

A *coloring* of G is a k-coloring of G for some k. A coloring is *legal* if $g(u)$ and $g(v)$ are distinct for all edges $\{u, v\}$ in E.

$\mathbf{A}$'s secret will be a legal k-coloring of a graph G, where it is assumed that it is infeasible to compute a legal k-coloring of G.

Assumption

For the sake of the argument, assume that there is a randomized procedure such that for sufficiently large inputs n and k,

> the procedure yields a graph G with n nodes together with a legal k-coloring of G,
> knowing only G and k, it is infeasible to compute a legal k-coloring of G.

Zero-knowledge protocols

Protocol Coloring

Assumption:	$\mathbf{A}$ knows a legal k-coloring g of G and has access to a random source not known to $\mathbf{B}$.
Step 1 ($\mathbf{A}$)	Pick uniformly at random a permutation π of $\{1, \ldots, k\}$. *Commit secretly* to the list of colors $\pi(g(1)), \ldots, \pi(g(n))$.
Step 2 ($\mathbf{B}$)	Among all edges of G, pick an edge $\{u, v\}$ uniformly at random and send u and v to $\mathbf{A}$.
Step 3 ($\mathbf{A}$)	Reveal the colors $\pi(g(u))$ and $\pi(g(v))$ to $\mathbf{B}$.
Step 4 ($\mathbf{B}$)	Accept if the two nodes are colored with distinct colors from $\{1, ..., k\}$ and reject, otherwise.

In order to diminish the error probability, the Protocol Coloring is iterated m times, where m is the number of edges of G.

The protocol Coloring achieves the following goals

(i) $\mathbf{A}$ can always verify correctly her identity.

(ii) If at each of the m iterations the colors committed to do not form a legal k-coloring of G, then the probability that $\mathbf{B}$ accepts is at most $1/2$.

(iii) $\mathbf{B}$ is not able to extract any relevant information while communicating with $\mathbf{A}$.

(i): Follows by inspection of the protocol.

(ii): The probability of error is at most $(1 - \frac{1}{m})^m \leq \frac{1}{\mathrm{e}} \leq \frac{1}{2}$.

(iii): $\mathbf{B}$ obtains nothing but mutually independent, uniformly distributed pairs of distinct colors.
In fact, $\mathbf{B}$ could easily produce his own sequence of pairs of colors that has the same distribution as the sequence evolving during the protocol ($=$ definition of zero-knowledge).

Committing secretly

In the Protocol $\mathrm{Coloring}$, $\mathbf{A}$ has to commit secretly to a sequence of colors.

If the protocol were executed in real life, this could be done by placing a corresponding number of colored tokens into opaque containers such that $\mathbf{A}$ cannot change the arrangement afterwards but may reveal the content of any container to $\mathbf{B}$.

In electronic form, one would commit to the individual bits of a word describing the sequence of colors.

Committing to a single bit can then be implemented for example by a one-to-one function f that is easy to compute but where for a given function value $f(n)$ it is infeasible to determine whether n has a certain property or not, say, is even or odd.

Here one has to assume that certain functions, e.g., the discrete logarithm, have these properties.

Table of Contents