Chapter 2. DESIGN METHODS

for

RANDOMIZED ALGORITHMS

In this chapter:

1. **main types of randomized algorithms are introduced and illustrated**
2. **main design methods for randomized algorithms are introduced and illustrated.**

BASIC PROBABILITIES FOR RANDOMIZED ALGORITHMS

Let us consider decision problems P.

- **For any randomized algorithm A for P , and any input x, we consider the set $S_{A,x}$ of all runs of A on x as the main sample space for analyzes of A (on x).**
- **If one chooses the random variable V_A that assigns 1 (0) to any run of A with correct (wrong) output, then the expectation value of V_A is exactly the success probability of A on x.**
- **The probability of the complementary event is called the error probability of A on x.**
- **If one takes a random variable that assigns to every computation its complexity (number of steps), then the expectation of this random variable equals the expected time complexity of A on x.**

CLASSIFICATION of RANDOMIZED ALGORITHMS

- **Las Vegas algorithms never produce wrong outcomes, but sometimes "I don't know" outcome (usually denoted as ??), is allowed, with bounded probability only.**
- **A one-sided-error Monte Carlo algorithm (1MC), for a language L, is an algorithm that accepts with probability at least $\frac{1}{2}$ any input $x \in L$ and rejects for sure any input not in L; (The error probability of such algorithms converges to 0 with exponential speed in the number of independent runs executed.)**
- **A bounded-error Monte Carlo algorithm (2MC) A, for a function F, is an algorithm for which there exists a constant $\varepsilon > 0$ such that, for any input x, the algorithm computes the correct output $A(x) = F(x)$, with probability at least $1/2 + \varepsilon$. (The error probability of such an algorithm can be reduced to an arbitrarily small given constant δ using only constantly many (depending on δ) runs.)**

- **A (unbounded error) Monte Carlo algorithm (UMC) is an algorithm that, for any input x, computes the correct output with probability at least $1/2$. (To reduce the error probability of such an algorithm below a given constant, exponentially many runs may be needed.)**

RANDOMIZED ALGORITHMS DESIGN PARADIGMS

- **Foiling the adversary**: One finds a set of deterministic algorithms such that, for each input most of these algorithms compute correct results efficiently and one takes as the randomized algorithm a probability distribution over such a set of deterministic algorithms. (The idea is to overcome such a situation when for each deterministic algorithm there exist bad inputs (for which the algorithm gives wrong result or computes inefficiently.)

- **Abundance of witnesses. A witness y for an input x and problem $P$is information with which one can solve the problem P for input x more efficiently than without it. If one finds a set such that at least half of its elements are witnesses, then a random choice of an element in this set leads to a witness with reasonable probability.**
- **Fingerprinting. For solving various problems it can be much more efficient to work with very small fingerprints (hashes) of the large objects than with such large objects directly.**
- **Random sampling If there are in a set sufficiently many objects we are looking for, then a random sampling in that set can provide such an object with sufficiently high probability. This method is also called probabilistic method.**

- **Random rounding.: A hard to solve optimization problem is transferred to an easy to solve optimization problem by increasing the size of the solution space. Outcomes of a new problem are then used to create a randomized algorithm to solve the original problem.**
- **An amplification of the success probability can be achieved by repeating independent computations on the same input (but with different random auxiliary input).**

STRINGS EQUALITY PROBLEM

Notation. For a binary vector/string $x = x_1x_2\dots x_n$ let

$$\mathbf{Number}(x) = \sum_{i=1}^{n} x_i 2^{n-i}.$$

Problem There are two parties, say A and B. Each of them has one n-bit string. By communication they have to decide whether their strings are equal. How to do that efficiently?

Each deterministic protocol clearly requires sending n bits in the worth case.

Initial situation.
Party A has a binary string $x = x_1x_2\dots x_n$.
Party B has a binary string $y = y_1y_2\dots y_n$.

Randomized protocol

1. **Alice chooses randomly a prime p such that $2 \leq p \leq n^2$ and sends to Bob p and binary representation of the number**
$$s = \textbf{Number}(x) \bmod p;$$
2. **Bob computes**
$$t = \textbf{Number}(y) \bmod p$$
and declares $x = y$ iff $s = t$.

The protocol requires sending at most
$$2\lceil \lg n^2 \rceil \leq 4\lceil \lg n \rceil \textbf{ bits.}$$

For $n = 10^{16}$ the protocol requires sending at most 256 bits.

ERROR ANALYSIS

Let us say that a prime $2 \leq p < n^2$ is bad for a pair $(x, y), x \neq y$, if the above protocol for such an input pair (x, y) yields a wrong answer.

Notation Prim(m) = number of primes smaller than m.

Error probability for an input (x, y) is

$$\frac{\textbf{number of bad primes for } (x,y)}{\textbf{Prim}(n^2)}.$$

Since, by Prime number theorem, Prim$(m) > m/\ln m$ for $m > 69$, we have that for $n \geq 9$

$$\textbf{Prim}(n^2) > \frac{n^2}{2\ln n}.$$

The so-called Prime number theorem says that there are approximately $\frac{n}{\ln n}$ primes among the first n integers.

NUMBER Of BAD PRIMES

Lemma Number of bad primes for $x \neq y$ is at most $n-1$.

Proof. A prime p is clearly bad for a pair $(x, y), x \neq y$, if and only if p divides the number

$$w = |\mathbf{Number}(x) - \mathbf{Number}(y)| < 2^n$$

w can be uniquely factorized as

$$w = p_1^{e_1} p_2^{e_2} \ldots p_k^{e_k}$$

where $p_1 < p_2 < \ldots < p_k$ are primes.

We can show that $k \leq n-1$. Indeed, if $k \geq n$, then

$$w \geq 1 \cdot 2 \cdot 3 \ldots n = n! > 2^n$$

what is a contradiction.

Probability of the error of the equality protocol is therefore:

$$\frac{n-1}{\mathbf{Prim}(n^2)} \leq \frac{n-1}{n^2/\ln n^2} \leq \frac{\ln n^2}{n} = \frac{2\ln n}{n}$$

which is at most $0.369 \cdot 10^{-14}$ in case $n = 10^{16}$.

PROBABILITY AMPLIFICATION

In case the above protocol is repeated 10 times, each time with a different prime, and the answer is $x = y$ each time, then the probability of an error is at most

$$\left(\frac{\ln n^2}{n}\right)^{10}$$

and therefore, for $n = 10^{16}$, the error is at most

$$0.47 \cdot 10^{-141}.$$

Reminder Probability of correct output is at least

$$1 - \frac{2\ln n}{n}.$$

LAS VEGAS ALGORITHM WITH ??

Problem Alice has 10 strings $x_i \in \{0,1\}^n, n > 500$ and Bob has also 10 strings $y_i \in \{0,1\}^n$. The task is to determine whether $x_i = y_i$ for some i.

One can show that each deterministic protocol has to exchange in the worth case $10n$ bits. Therefore no deterministic algorithm is essentially more efficient than sending all bits from Alice to Bob and then to let Bob to make comparisons.

We show the existence of a Las Vegas algorithm to solve the problem (to decide whether such i exists) with communication complexity

$$n + \mathcal{O}(\lg n).$$

PROTOCOL

- **Alice randomly chooses 10 primes $p_1, \ldots, p_{10}$, each smaller than n^2, computes all**

$$s_i = \mathbf{Number}(x_i) \bmod p_i, i \in \{1, .2, \ldots, 10\}$$

and sends to Bob

$$p_1, \ldots, p_{10}, s_1, \ldots, s_{10}.$$

- **Bob computes all**

$$r_i = \mathbf{Number}(y_i) \bmod p_i$$

and compares elements in all pairs (s_i, r_i).

If $s_i \neq r_i$ for all i, Bob outputs NO (0);

otherwise, if j is the smallest integer such that $s_j = r_j$, then Bob sends Alice the pair (j, y_j).

- **If $x_j = y_j$, Alice outputs YES (1);**
otherwise she outputs ?? (because it may exist other k with $x_k = y_k$).

COMPLEXITY ANALYSIS of the PROTOCOL

The above protocol exchanges $20\lceil \lg n^2 \rceil + n + 4$ bits - $\{$4 bits are needed to send $j\}$.

Proof that the protocol is a Las Vegas protocol

- If $x_i \neq y_i$ for all i, then probability that

$$\mathrm{Number}(x_i) \bmod p_i \neq \mathrm{Number}(y_i) \bmod p_i$$

for all i, and therefore probability that the above protocol produces correct output 0 is at least

$$\left(1 - \frac{2\ln n}{n}\right)^{10}$$

because, according to the analysis of the strings equality algorithm, probability that $\mathrm{Number}(x_i) \bmod p_i \neq \mathrm{Number}(y_i) \bmod p_i$ is at least $1 - \frac{2\ln n}{n}$

and it can be shown that

$$\left(1 - \frac{2\ln n}{n}\right)^{10} \geq 1 - \frac{20\ln n}{n} \geq \frac{1}{2}$$

If there exists a j such that

$$\mathrm{Number}(x_i) \bmod p_i = \mathrm{Number}(y_i) \bmod p_i$$

then protocol outputs ??, what is O.K.

- Let there is an j such that $x_j = y_j$ and j_0 be the smallest such j.

  Protocol then accepts the input iff

  $$\text{Number}(x_i) \bmod p_i \neq \text{Number}(x_i) \bmod p_i$$

  for all $i < j_0$. Let us denote such an event by E_{j_0}.

  If $j_0 = 1$, then the protocol accepts the input with certainty. If $j_0 > 1$, then, as discussed before, probability of the event E_{j_0} is at least

  $$\left(1 - \frac{2\ln n}{n}\right)^{j_0 - 1} \geq 1 - \frac{2(j_0 - 1)\ln n}{n}$$

  for sufficiently large n.

  The above expression has its minimum for $j_0 = 10$ and therefore the protocol produces the output YES (1) with probability at least $1 - \frac{18\ln n}{n}$, which is larger than $\frac{1}{2}$ for all $n \geq 189$.

  In the complementary case, when there is an $l < j_0$ such that

  $$\text{Number}(x_l) \bmod p_l = \text{Number}(x_l) \bmod p_l$$

  the protocol produces as output ??.

The above protocol is therefore indeed a Las Vegas protocol.

TWO TYPES of LAS VEGAS ALGORITHMS

There are two types of Las Vegas algorithms:

- **Algorithms that never produce ??.**
- **Algorithms that may produce ??.**

Note: Las Vegas algorithms may not terminate for some inputs!!!

Claim Any Las Vegas algorithm A_1 can be converted to a Las Vegas algorithm A_2 that solves the same problem and never produces ??.

Construction of A_2 is simple. Each time A_1 is to produce the output ??, what can be done only with bounded probability, a new run of A_1 is initialized with the same input.

AMPLIFICATION of 1MC ALGORITHMS

Error probability of 1MC algorithms decreases exponentially with the number of computation repetitions.

Indeed, if we have k independent runs of the algorithm and one output is 0 (rejection), then the input is rejected with certainty.

If all outputs are 1 (acceptance), then the output will be YES (acceptance) and the error probability is at most $\left(\frac{1}{2}\right)^k$.

Because of the exponential decrease of error probability using repeated applications, 1MC algorithms are very popular.

AMPLIFICATION of 2MC ALGORITHMS

Let A be a 2MC algorithm for a function F and $\varepsilon > 0$ such that

$$\mathbf{Prob}(A(x) = F(x)) \geq \frac{1}{2} + \varepsilon.$$

For any integer k let A_k be the algorithm that performs k independent runs of A and if there is an α that appears at least $\lceil \frac{k}{2} \rceil$ times as the output, then A_k produces α as the output; if there is no such an α, A_k produces ?? as the output.

One can show that if an $\delta > 0$ is fixed then

$$\mathbf{Prob}(A_k(x) = F(x)) \geq 1 - \delta$$

if

$$k \geq \frac{2 \ln \delta}{\ln(1 - 4\varepsilon^2)}.$$

If ε and δ are considered as constant, then so is k and therefore

$$\mathbf{Time}_{A_k}(n) = \mathcal{O}(\mathbf{Time}_A(n)).$$

2MC versus UMC algorithms

Basic question: What is the difference between 2MC and UMC algorithms?

Answer: For an UMC algorithm A it may happen that the distance between the error probability and $\frac{1}{2}$ tends to 0 with growing input size.

As a consequence, if we design, given an $\delta > 0$, an algorithm A_k, that performs k independent runs of A and

$$\mathbf{Prob}(A_k(x)) = F(x)) > 1 - \delta$$

then running time of A_k may be exponential in the input length.

TESTING of STRAIGHTLINE PROGRAMS

Given is a straightline program that consists of a sequence of assignments; the first assignment is $a \leftarrow 1$ and in each other assignment two previously designed variables are either added, or subtracted or multiplied.

How to check whether the outcome of such a program will be 0? Example of such a straightline program:

$a = 1$
$b = a + a$
$c = b \times b$
$d = c \times c$
$e = d \times d$
$f = e - a$
$g = d - a$
$h = d + a$
$i = g \times h$
$j = f - i$
(The catch (problem) is that numbers created during such a program can be enormous.)

RANDOMIZED ALGORITHM

1. If number of assignment statements is n, pick a random prime p with n^2 digits.
2. Perform the given program, but do all computations modulo p.

This algorithm runs in time polynomial in n.

If such a randomized algorithm provides a non-zero number as the output, the output of the original program (with full computations, not modulo ones) is also non-zero.

If the output is zero, how confident we can be that also the original program evaluates also to 0?

Let x be the outcome of the original program. Clearly, $|x| \leq 2^{2^n}$.

By Prime Number Theorem, number of primes with n^2 digits is about $\frac{2^{n^2}}{n^2}$ and since this number is much bigger than 2^n, most of those primes can not divide x (unless $x = 0$).

This means that if we pick a random prime and it does not divide x, then we can be very confident (but not certain) that $x = 0$.

SECRET SHARING between TWO

Problem: The task is to "partition" a secret S between two parties P_1 and P_2 in such a way that none of the parties alone has slightest idea what S is, but if they get together they can easily determine S.

Method: A moderator distributes a binary-string secret S, between two parties P_1 and P_2 by choosing a random binary string b, of the same length as s, and sends:

- b to P_1 and
- $s \oplus b$ to P_2.

This way, none of the parties P_1 and P_2 alone has a slightest idea about s, but both together easily recover s by computing

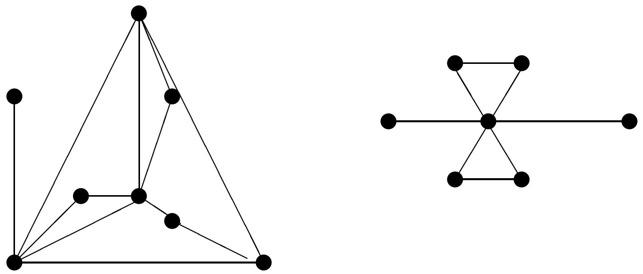
$$b \oplus (s \oplus b) = s.$$

PERFECT MATCHING ALGORITHM

Let $G = \langle V, E \rangle$ be an undirected graph. A subset $X \subseteq E$ is said to be a *matching* of G if no two edges in X have a common node.

A matching is said to be a *perfect matching* if it covers all nodes of G.

Example Which of the graphs in the next figure has a perfect matching?



There are polynomial time algorithms to decide whether a given graph has perfect matching, but none is so simple as the randomized algorithm based on so called Tutte theorem presented bellow.

Basic concept: Tutte matrix of a graph

Let $G = \langle V, E \rangle$ be a bipartite graph with nodes $V = \{1, 2, \ldots, n\}$ The *Tutte matrix* $A_G = \{a_{ij}\}_{i,j=1}^{n}$ of G is defined by (x_{ij} are variables, all different for different i, j)

$$a_{ij} = \begin{cases} x_{ij} & \text{if } (i,j) \in E,\ i < j; \\ -x_{ij} & \text{if } (i,j) \in E,\ i > j; \\ 0 & \text{if } (i,j) \notin E \end{cases}$$

TUTTE THEOREM

Example For the graph of six nodes at which node 1 is connected with nodes 4 and 5, node 2 with nodes 5 and 6 and the node 3 is connected with nodes 5 and 6 the Tutte matrix has the form:

$$\begin{pmatrix} 0 & 0 & 0 & x_{14} & x_{15} & 0 \\ 0 & 0 & 0 & 0 & x_{25} & x_{26} \\ 0 & 0 & 0 & 0 & x_{35} & x_{36} \\ -x_{14} & 0 & 0 & 0 & 0 & 0 \\ -x_{15} & -x_{25} & -x_{35} & 0 & 0 & 0 \\ 0 & -x_{26} & -x_{36} & 0 & 0 & 0 \end{pmatrix}$$

Tutte theorem A graph $G = \langle V, E \rangle$ has a perfect matching iff the determinant of the corresponding Tutte matrix is not identically zero.

Proof The determinant of A_G equals $\sum_\pi \sigma_\pi \prod_{i=1}^n a_{i\pi(i)}$ where π are permutations of $\{1, 2, \ldots, n\}$ and $\sigma_\pi = 1$ ($\sigma_\pi = -1$) if π is a product of an even (odd) number of transpositions.

Observation 1 For a permutation π, $\prod_{i=1}^{n} a_{i\pi(i)} \neq 0$ iff $G_\pi = \{(i, \pi(i)),\ 1 \leq i \leq n\}$ is a subgraph of G.

Observation 2 Permutations π with at least one odd cycle do not contribute at all to the determinant of A, because to each such permutation π there is a permutation π' such that $\prod_{i=1}^{n} a_{i\pi(i)} = -\prod_{i=1}^{n} a_{i\pi'(i)}$

Observation 3 It is sufficient to consider permutations π such that G_π consists only of even cycles. Let permutation π^r be obtained from π by reversing all cycles.

Notation For a perfect matching E' let $t_{E'}$ denote the product of the $a's$ corresponding to the edges of E'.

CASE ANALYSIS

Case I $\pi = \pi^r \Rightarrow G_\pi$ consist of the cycles of length 2, π corresponds to a perfect matching E' such that $\prod_{i=1}^{n} a_{i\pi(i)} = (t_{E'})^2$.

Case II $\pi \neq \pi^r$ In this case both π and π' correspond to the union of two perfect matching E' E'' obtained by alternatively selecting edges within the cycles so that

$$\prod_{i=1}^{n} a_{i\pi(i)} + \prod_{i=1}^{n} a_{i\pi^r(i)} = 2t_E t_{E'}.$$

Conclusion

$$det(A_G) = (t_{E_1'} + \ldots + t_{E_k'})^2$$

where E_i' denotes i-th perfect matching.

RANDOM SELECT

Problem Given is a set $S = \{a_1, \ldots, a_n\}$ of $n > 0$ different numbers, and $1 \leq k \leq n$, find the k-th smallest number from S.

A naive way to solve the problem is to sort at first S. This requires $\mathcal{O}(n \lg n)$ comparisons. The following randomized algorithm RSELECT can do that in $\mathcal{O}(n)$ steps

Algorithm RSELECT(S, k))

1. If $n = 1$ output a_1.
2. Otherwise choose $i \in_r \{1, 2, \ldots, n\}$ and

   2.1. compute
$$S_< = \{b \in S \,|\, b < a_i\}$$
$$S_> = \{b \in S \,|\, b > a_i\}$$

   2.2. **if** $|S_<| \geq k$ **then** RSELECT$(S_<, k)$;
   else if $|S_<| = k - 1$ **then** output a_i;
   else RSELECT$(A_>, k - |A_<| - 1)$

This is clearly a Las Vegas algorithm.

CLASSIFICATION of RANDOMIZED OPTIMIZATION ALGORITHMS

Classification of the randomized algorithms we had so far was based on the frequency of outputs and it is suited only for classification of algorithms for decision problems and for computation of functions, but not for optimization problems.

In case of optimization problems we do not take as output the most frequent output from several runs, but best output according to some optimization criterion.

Moreover, in case of optimization problems our goal is not always to find an optimal solution. We are usually quite happy to find a (feasible) solution whose cost does not differ much from the cost (quality) of an optimal solution.

All that leads to another classification of randomized approximation algorithms.

ILLUSTRATION

If a randomized algorithm A computes an optimal solution for an input x with probability at least $\frac{1}{|x|}$, then it does not mean that A is not useful.

One can execute $|x|$ independent runs of A on input x, call it the A_x algorithm, and then take the best output. Let us ask what is then the probability of success.

Probability of computing no optimal solution in one run is at most

$$1 - \frac{1}{|x|}$$

and therefore the probability A_x does not find an optimal solution in $|x|$ independent runs is at most

$$(1 - \frac{1}{|x|})^{|x|} < \frac{1}{e}$$

and we have a constant probability $1 - \frac{1}{e}$ of computing an optimal solution.

DEFINITION of OPTIMIZATION PROBLEMS

Definition An **optimization problem** is a 6-tuple
$\mathcal{P} = (\Sigma_I, \Sigma_O, L, \mathcal{M}, \mathbf{cost}, \mathbf{goal})$, where

1. Σ_I is an **input alphabet**;
2. Σ_O is an **output alphabet**;
3. $L \subseteq \Sigma_I^*$ is the language of **feasible inputs** and any $x \in L$ is called a **problem instance of P**;
4. $\mathcal{M}$ is a function from L to $2^{\Sigma_O^*}$, and for each $x \in L$, $\mathcal{M}(x)$ is the set of **feasible solutions for x**;
5. **cost** is a function: $\bigcup_{x\in L}(\mathcal{M}(x) \times x) \rightarrow \mathbf{R}^+$, called **cost function**;
6. **goal** $\in$ (minimum, maximum) is an objective.

SOLVING an OPTIMIZATION PROBLEM

Observe that to solve an optimization problem is more as computing a relation, than as computing a function, because we are usually happy with finding one of optimal solutions or even a solution close to an optimal one.

A feasible solution $\alpha \in \mathcal{M}(x)$ is called optimal for the problem instance x of $\mathcal{P}$ if

$$\mathbf{cost}(\alpha, x) = \mathbf{goal}\{\mathbf{cost}(\beta, x) \,|\, \beta \in \mathcal{M}(x)\}.$$

An algorithm $\mathcal{A}$ is said to solve $\mathcal{P}$ if, for any $x \in L$,

- $\mathcal{A}(x) \in \mathcal{M}(x)$;
- $\mathbf{cost}(\mathcal{A}(x), x) = \mathbf{goal}\{\mathbf{cost}(\beta, x) \,|\, \beta \in \mathcal{M}(x)\}$

If the goal is to find minimum (maximum) we talk about a minimization (maximization) problem.

EXAMPLE - TRAVELING SALESMAN PROBLEM

Input: A weighted complete graph (G, c), where $G = (V, E)$, $V = \{v_1, \ldots, v_n\}$, $E \subset V \times V$ and $c : E \to \mathbf{N}^+$ is a cost function.

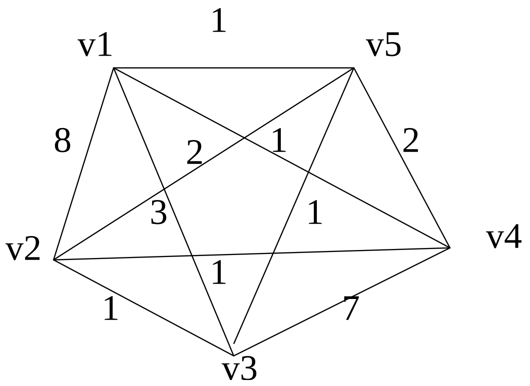
hamiltonian cycles: For any problem instance (G, c), let $\mathcal{M}(G, c)$ be the set of Hamiltonian cycles of G - each such cycle is represented by a sequence of vertices $v_{i_1}, v_{i_2}, \ldots, v_{i_n}, v_{i_1}$, where $i_1, i_2, \ldots, i_n$ is a permutation of $(1, 2, \ldots, n)$.

Costs of Hamiltonian cycles For every Hamiltonian cycle $H = (v_{i_1}, v_{i_2}, \ldots, v_{i_n}, v_{i_1}) \in \mathcal{M}(G, c)$

$$\mathbf{cost}((v_{i_1}, v_{i_2}, \ldots, v_{i_n}, v_{i_1}), (G, c)) = \sum_{j=1}^{n} c((v_{i_j}, v_{i_{j \bmod n+1}})).$$

Goal: minimum - to find a Hamiltonian cycle with minimum cost.

EXAMPLE



$$\mathrm{cost}((v1, v2, v3, v4, v5, v1), (G, c)) = 19$$

$$\mathrm{cost}((v1, v5, v3, v2, v4, v1), (G, c) = 5$$

INTEGER LINEAR PROGRAMMING

Given is a system of linear equations and a linear function over variables of this equations. The task is to find a solution to the system of equations such that the value of the linear function is minimized. More formally:

Input **An $m \times n$ matrix and two vectors**

$$A = \{a_{ij}\}_{i,j=1}^{i=m,j=n} \quad b = (b_1, \ldots, b_m)^T \quad c = (c_1, \ldots, c_n)$$

with integer entries.

Set of feasible solutions: $\mathcal{M}(A, b, c) = \{X = (x_1, \ldots, x_n)^T \mid AX = b\}$.

Cost of a solution: **For** $X = (x_1, \ldots, x_n) \in \mathcal{M}(A, b, c)$

$$\textbf{cost}(X, (A, b, c)) = c \cdot X = \sum_{i=1}^{n} c_i x_i.$$

Goal: **minimum**

QUANTIFICATION of "ALMOST OPTIMAL SOLUTIONS"

Many very important optimization problems are **NP**-hard and so we know only exponential time algorithms for finding optimal solutions.

New idea is to jump from exponential to polynomial time by weakening the requirements - to be satisfied with almost optimal solutions. To quantize that tries the next definition.

Definition Let $\mathcal{P} = (\Sigma_i, \Sigma_O, L, \mathcal{M}, \mathrm{cost}, \mathrm{goal})$ be an optimization problem. We say that A is a consistent algorithm for $\mathcal{P}$ if, for every $x \in L$, the output $A(x)$ is a feasible solution for x - that is $A(x) \in \mathcal{M}(x)$.
Let A be a consistent algorithm for $\mathcal{P}$. For every $x \in L$, we define the approximation ratio, $\mathrm{Ratio}_A(x)$, of A on x as

$$\mathrm{Ratio}_A(x) = \max\left\{\frac{\mathrm{cost}(A(x))}{\mathrm{Opt}_{\mathcal{P}}(x)}, \frac{\mathrm{opt}_{\mathcal{P}}(x)}{\mathrm{cost}(A(x))}\right\},$$

where $\mathrm{Opt}_{\mathcal{P}}(x)$ is the cost of an optimal solution for the instance x of $\mathcal{P}$.
For any $\delta > 1$ we say that A is a δ-**approximation algorithm** for $\mathcal{P}$ if, for every $x \in L$, $\mathrm{Ratio}_A(x) \leq \delta$.

WHY TO APPLY RANDOMIZATION in DISCRETE OPTIMIZATIONS

One of the main goals in the area of discrete optimization is to improve the approximation ratio. One tries to design randomized approximation algorithms that produce feasible solutions whose cost (quality) is not very far from the optimal cost with high probability. In the analysis of randomized algorithms we consider therefore the approximation ratio as a random variable and the aim is then either

1. **to estimate the expected value, E(Ratio), or**
2. **to guarantee that a certain approximation ratio is achieved with probability at least $\frac{1}{2}$.**

These two different aims lead to two ways randomized approximation algorithms are defined.

Definition 1 Let $\mathcal{P} = (\Sigma_I, \Sigma_O, L, \mathcal{M}, \mathbf{cost}, \mathbf{goal})$ be an optimization problem. For any $\delta > 1$, a randomized algorithm A is called a **randomized E[δ]-approximation algorithm** for $\mathcal{P}$ if

1. $\mathbf{Prob}(A(x) \in \mathcal{M}(x)) = 1$, and
2. $\mathbf{E}[\mathbf{Ratio}_A(x) \leq \delta] \geq \frac{1}{2}$.

for every $x \in L$.

Definition Let $\mathcal{P} = (\Sigma_I, \Sigma_O, L, \mathcal{M}, \mathbf{cost}, \mathbf{goal})$ be an optimization problem.For any $\delta > 1$, a randomized algorithm A is called a **randomized δ-approximation algorithm** for $\mathcal{P}$ if

1. $\mathbf{Prob}(A(x) \in \mathcal{M}(x)) = 1$, and
2. $\mathbf{Prob}(\mathbf{Ratio}_A(x) \leq \delta) \geq \frac{1}{2}$.

for every $x \in L$.

FACTORIZATION of INTEGERS

The fastest classical algorithm to factor m bit numbers requires time $\mathcal{O}(e^{cm^{1/3}(\lg m)^{2/3}})$.

Shor's factorization algorithm requires $\mathcal{O}(m^2 \lg^2 m \lg\lg m)$ time on a quantum computer and polynomial time on a classical computer.

Factorization of integers can be reduced to solution of several simple algorithmic problems.

FIRST REDUCTION

Lemma 0.1 *If there is a polynomial time deterministic (randomized) [quantum] algorithm to find a nontrivial solution of the modular quadratic equations*

$$a^2 \equiv 1 \pmod{n},$$

then there is a polynomial time deterministic (randomized) [quantum] algorithm to factorize integers.

Proof. Let $a \neq \pm 1$ be such that $a^2 \equiv 1 \pmod{n}$. Since

$$a^2 - 1 = (a+1)(a-1),$$

if n is not prime, then a prime factor of n has to be a prime factor of either $a+1$ or $a-1$.

By using Euclid's algorithm to compute

$$gcd(a+1, n) \text{ and } \quad gcd(a-1, n)$$

we can find, in $\mathcal{O}(\lg n)$ steps, a prime factor of n.

SECOND REDUCTION

The second key concept is that of **period** of the functions

$$f_{n,x}(k) = x^k \bmod n.$$

It is the smallest integer r such that

$$f_{n,x}(k+r) = f_{n,x}(k)$$

for any k, i.e. the smallest r such that

$$x^r \equiv 1 \pmod{n}.$$

AN ALGORITHM TO SOLVE EQUATION $x^2 \equiv 1 \pmod{n}$.

1. Choose randomly $1 < a < n$.
2. Compute $gcd(a,n)$. If $gcd(a,n) \neq 1$ we have a factor.
3. Find period r of function $a^k \bmod n$.
4. If r is odd or $a^{r/2} \equiv \pm 1 \pmod{n}$,then go to step 1; otherwise stop.

If this algorithm stops, then $a^{r/2}$ is a non-trivial solution of the equation

$$x^2 \equiv 1 \pmod{n}.$$

EXAMPLE

Let $n = 15$. Select $a < 15$ such that $gcd(a, 15) = 1$.
$\{$The set of such a is $\{2, 4, 7, 8, 11, 13, 14\}\}$

Choose $a = 11$. Values of 11^x mod 15 are then

$$11, 1, 11, 1, 11, 1$$

what gives $r = 2$.

Hence $a^{r/2} = 11 \pmod{15}$. Therefore

$$gcd(15, 12) = 3, \qquad gcd(15, 10) = 5$$

For $a = 14$ we get again $r = 2$, but in this case

$$14^{2/2} \equiv -1 \pmod{15}$$

and the following algorithm fails.

1. Choose randomly $1 < a < n$.
2. Compute $gcd(a, n)$. If $gcd(a, n) \neq 1$ we have a factor.
3. Find period r of function a^k mod n.
4. If r is odd or $a^{r/2} \equiv \pm 1 \pmod{n}$,then go to step 1; otherwise stop.

EFFICIENCY of REDUCTION

Lemma 0.2 *If $1 < a < n$ satisfying $gcd(n, a) = 1$ is selected in the above algorithm randomly and n is not a power of prime, then*

$$Pr\{r \textit{ is even and } a^{r/2} \not\equiv \pm 1\} \geq \frac{9}{16}.$$

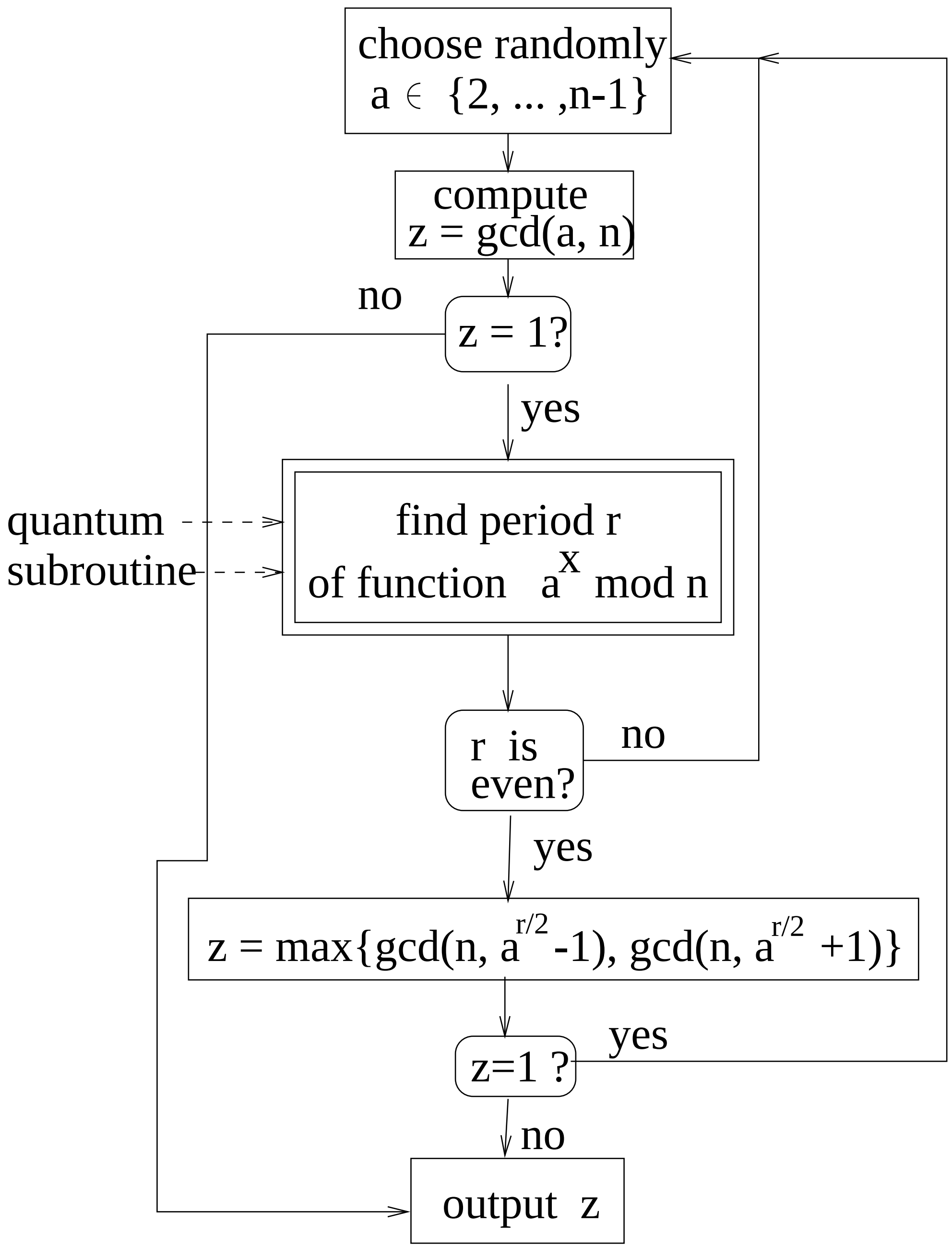
1. Choose randomly $1 < a < n$.
2. Compute $gcd(a, n)$. If $gcd(a, n) \neq 1$ we have a factor.
3. Find period r of function $a^k \bmod n$.
4. If r is odd or $a^{r/2} \equiv \pm 1 \pmod{n}$,then go to step 1; otherwise stop.

Corollary 0.3 *If there is a polynomial time randomized [quantum] algorithm to compute the period of the function*

$$f_{n,a}(k) = a^k \bmod n,$$

then there is a polynomial time randomized [quantum] algorithm to find non-trivial solution of the equation $a^2 \equiv 1 \pmod{n}$ (and therefore also to factorize integers).

A GENERAL SCHEME FOR SHOR'S ALGORITHM


choose randomly
a ∈ {2, ... ,n-1}
compute
z = gcd(a, n)
no
z = 1?
yes
quantum
subroutine
find period r
of function a^x mod n
r is
even?
no
yes
z = max{gcd(n, a^r/2 -1), gcd(n, a^r/2 +1)}
z=1 ?
yes
no
output z

GENERATION of (PSEUDO) RANDOMNESS

There are nowadays several tools how to produce sufficiently good randomness:

- **Pseudo-random generators**
- **Extractors of randomness**
- **Quantum measurements - quantum random bits generating devices (that are already produced commercially).**

PSEUDO-RANDOM GENERATORS

Definition - pseudorandom generator Let $l_n : \{0,1\}^n \rightarrow \{0,1\}^{N_n}$ be such that $N_n >> n$ for all n. A **(computationally indistinguishable) pseudorandom generator** with **stretch function** l_n, is an efficient deterministic algorithm which on input of a random n-bit **seed** outputs a N_n-bit sequence which is computationally indistinguishable from a random N_n-bit sequence.

It has been shown that if integer factoring is intractable, then the so-called BBS pseudo-random generator, discussed below, is sufficiently good even for cryptographic purposes.

Let n be an integer such that $n \bmod 4 = 3$. Choose randomly an $x_0 < n$.
For $i \geq 0$ let $x_{i+1} = x_i^2 \bmod n,\ b_i =$ the least significant bit of x_i For each integer j, let $BBS_{n,j}(x_0) = b_0 \dots b_{j-1}$ be the first j bits of the pseudo-random sequence generated from the seed x_0 by the BBS pseudo-random generator.

RANDOMNESS EXTRACTORS

Extractors are algorithms that produce from any long and weakly-random bit-string a shorter, but more random, bit-string.

In other words, an extractor is a mapping which, when applied to high-entropy source generates a shorter yet uniformly distributed output.

In a more general approach an extractor is an algorithm that converts a long weakly random source and a truly random short seed into a uniformly distributed random output (that is longer than the seed and shorter than the source).

An extractor is a certain kind of pseudorandom generator.

No extractor is currently know that has been proven to work when applied to any type of high-entropy source.

Von NEUMANN EXTRACTOR

It is an extractor that keeps taking successive pairs of consecutive bits (non-overlapping) from the input stream and if the two bits are the same, no output is generated; if they are different the first of them is outputed.

For example the input sequence

$$10111100001111000011100110010101$$

is transformed to the sequence

$$1101000$$

The von Neumann extractor can be shown to produce a uniform output, even if the distribution of the input bits is not uniform, so long as each bit has the same probability of being one and there is no correlelation between successive bits.

EXTRACTORS - FORMAL DEFINITION

Definition A (k, ε)-extractor is a mapping

$$\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$$

such that for every distribution X on $\{0,1\}^n$ with $H_\infty(X) \geq k$ the distribution $\mathrm{Ext}(X, s)$ is ε-close to the uniform distribution on $\{0,1\}^m$.

The aim is to have $n > m$ and $d << m$.

By a probabilistic method to be discussed later one can show that there exists a (k, ε) extractor for many k and ε.

Note H_∞ stands for so-called *min-entropy*, which is a measure of the amount of randomness in the worst case.