RANDOMIZED ALGORITMS AND COMMUNICATIONS

Jozef Gruska

Faculty of Informatics, Masaryk University
Brno, Czech Republik

February 20, 2013

WEB PAGE of the LECTURE

http://www.fi.muni.cz/usr/gruska/random13

Exercises: Tasks distributio: Tuesedays Personal discussions: Thursday 8.00 or 8.30 for one hour

Exams: 5 questions - 3 to choose to answer

CONTENTS

LITERATURE

- R. Motwami, P. Raghavan: Randomized algorithms, Cambridge University Press, 1995
- J. Gruska: Foundations of computing, 1997
- J. Hromkovič: Design and analysis of randomized algorithms; Springer, 2005
- N. Alon, J. H. Spencer: The probabilistic method, Willey-Interscience, 2008

Chapter 1. INTRODUCTION

The main aim of the first chapter and lecture is:

1. **to present several interesting examples of simple randomized algorithms;**
2. **to demonstrate advantages of randomized algorithms and methods of their analysis.**

The second aim of this chapter is to introduce main complexity classes for randomized algorithms and to discus their relation to complexity classes for deterministic computations.

WHY RANDOMIZED ALGORITHMS?

Randomized algorithms are such algorithms that make random choices (coin-tossing) during their executions. As a consequence, their outcomes do not depend only on their (external) inputs.

Advantages: Randomized algorithms are often simpler, often more efficient, or both, than deterministic algorithms for the same problem!! ? !!

Why are randomized algorithms more efficient? Two simplified explanations:

(1) A systematic search for a solution has often to go through a time-consuming computation paths corresponding to some (few) very unlikely pathological cases. A randomized search for a solution can often avoid, with a sufficiently large probability, such time-consuming paths.

(2) For some algorithmic problems P, for each deterministic algorithm for P one can find bad inputs that force the algorithm to do very long computations. However, for P there is a set of deterministic algorithms such that for any input most of these algorithms are fast and a random choice of one of the algorithms from such a set very likely provides fast an output.

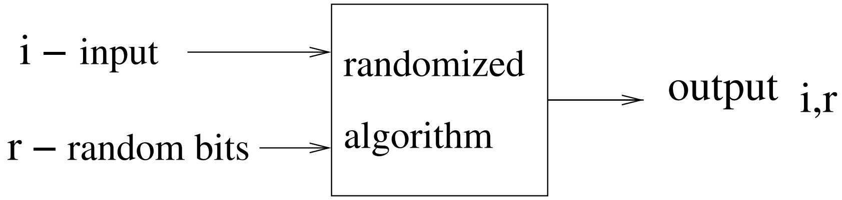
Moreover, **quantum computations** are, in principle, randomized.

Randomized complexity classes offer also a plausible way to extend the very important *feasibility* concept.

ANOTHER VIEW of RANDOMIZED ALGORITHMS

A randomized algorithm $\mathcal{A}$ is an algorithm that at each new run receives, in addition to its input i, a new stream/string r of random bits which are then used to specify outcomes of the subsequent random choices (or coin tossing) during the execution of the algorithm.

Streams r of random bits are assumed to be independent of the input i for the algorithm $\mathcal{A}$.



Important comment: Repeated runs of a randomized algorithm with fixed input data will not, in general, produce the same results. Outcomes, $\mathcal{A}(i, r)$, depend not only on i, but also on r.

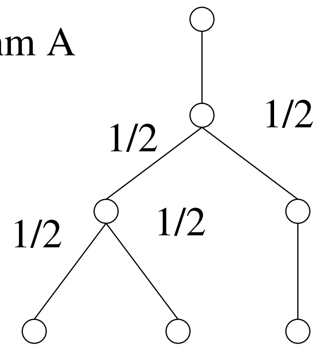
MODELS of RANDOMIZED ALGORITHMS I.

A randomized algorithm can be seen as follows:

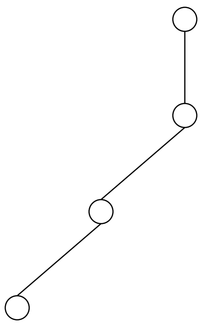
- **As an algorithm that may, from time to time, toss a coin, or read a (next) random bit from its special input stream of random bits, and then to proceed depending on the outcome of the coin tossing (or chosen random bit).**
- **As a nondeterministic-like algorithm which has a probability assigned to each possible transition.**
- **As a probability distribution on a set of deterministic algorithms - $\{\mathcal{A}_i, p_i\}_{i=1}^{n}$.**

RANDOMIZED ALGORITHM as a PROB. DISTR. on DETERMINISTIC ALG

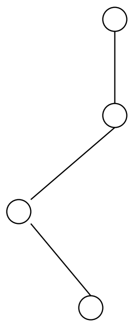
randomized algorithm A



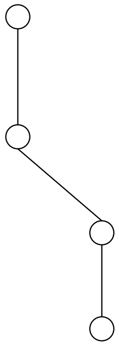
as a probabilistic distribution on three deterministic algorithms B, C, D



(B, 1/4)

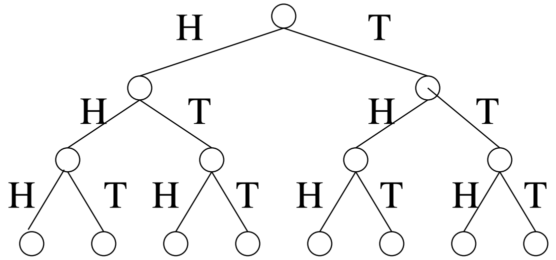


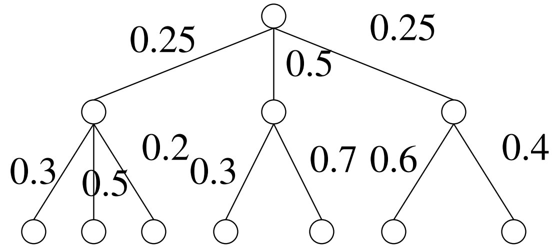
(C, 1/4)

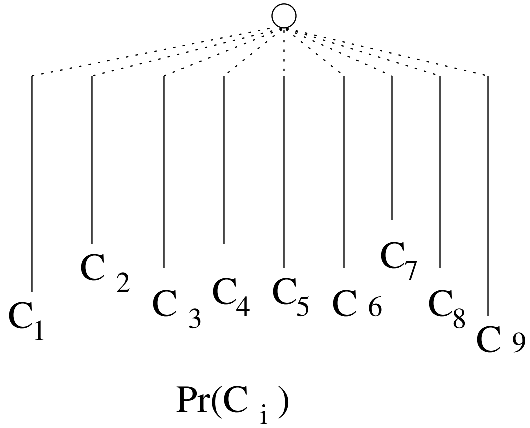


(D, 1/2)

MODELS of RANDOMIZED ALGORITHMS II







C_i are runs of dif. determ. alg.

STORY of RANDOMNESS

DOES RANDOMNESS EXIST? - I

One of the fundamental questions (of science) has been, and actually still is, whether randomness really exist or whether term *randomness* is used only to present objects and events with unknown lawfulness. Two early views are:

The randomness is the unknown and Nature is determined in its fundamentals.

Democritos (460-370 BC)

By Democritos, the order conquered the world and this order is governed by unambiguous laws. By him, the word randomness is used when we have an incomplete knowledge of some phenomena.

The randomness is objective, it is the proper nature of events.

Epikurus (341-270 BC)

By Epikurus, there exists a true randomness that is independent of our knowledge.

Einstein also accepted the notion of randomness only in the relation to incomplete knowledge.

VIEWS on RANDOMNESS in 19th CENTURY

Main arguments, before 20th century, why randomness does not exist:

God-argument: There is no place for randomness in a world created by God.

Science-argument: Success of natural sciences and mechanical engineering in 19th century led to a belief that everything could be discovered and explained by deterministic causalities of cause and resulting effect.

Emotional-argument: Randomness used to be identified with chaos, uncertainty and unpredictability, what was always related to a fear.

There are only two possibilities, either a big chaos conquers the world, or order and law.

Marcus Aurelius

EINSTEIN versus BOHR

God does not roll dice.

Albert Einstein, 1935, a strong opponent of randomness.

The true God does not allow anybody to prescribe what he has to do.

Famous reply by Niels Bohr - one of the fathers of quantum mechanics

DOES GOD PLAY DICE? - NEW VIEWS

God does play even non-local dice.

An observation, due to N. Gisin, on the basis that measurement of entangled states produces shared randomness.

God is not malicious and made Nature to produce, so useful, (shared) randomness.

What the outcomes of theoretical informatics imply.

RANDOMNESS in NATURE

Two big scientific discoveries of 20th century changed the view on usefulness of randomness.

1. **It has turned out that random mutations of DNA have to be considered as a crucial instrument of evolution.**
2. **Quantum measurement yields, in principle, random outcomes.**

RÉNYI's VIEW

Randomness and order do not contradict each other; more or less both may be true at once.

The randomness controls the world
and due to this in the world there are order and law,
which can be expressed in measures of random events
that follow the laws of probability theory.

Alfréd Rényi

RANDOMNESS

- **Randomness is a topic in mathematics studied since 17th century.**
- **Attempts to formalize chance by mathematical laws is somehow paradoxical because, a priori, chance (randomness) is the subject of no law.**
- **There is no proof that randomness exists in the real world.**
- **More exactly, there is no proof that quantum mechanical phenomena can be exploited to provide a satisfactory source of randomness in macroworld.**

KOLMOGOROV COMPLEXITY

- Kolmogorov complexity $K_C(x)$ of a binary string x with respect to a universal computer C is the length of the shortest program for C that produces x.
- The above definition is *basically* independent of the choice of C. Namely, for any other universal computer C' there is a constant $a_{C,C'}$ such that for any string x, $K_{C'}(x) \leq K_C(x) + a_{C,C'}$.
- A string x is considered as **random** if $K_C(x) \approx |x|$, that is if x is incompressible.
- Kolmogorov complexity is not computable.
- It is undecidable whether a given string is random.
- Until Kolmogorov complexity was introduced we had no meaningful way to talk about a given object being random.

MONOPOLIST GAME

Game Given are n active players each having w one dollar coins. They play, in rounds, the following game until all but one players become bankrupt:

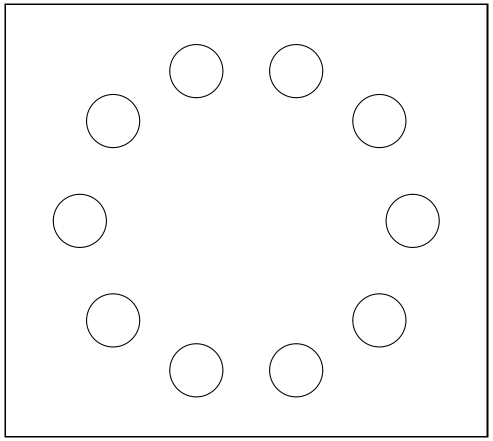
1. In each round every active player puts $1 on a table and roulette wheel is spined to determine the winner who then take all money on the table.
2. A player who looses all his money declares bankruptcy and becomes inactive.

Will the game end? It can be shown that it ends almost always in approximately $(nw)^2$ steps.

EXAMPLE 2 - ELECTION of a LEADER

In some cases randomization is the only way to solve the problem.

Example **Let n identical processors, connected into a ring, have to choose one of them to be a "leader", under the assumption that each of the processors knows n.**



Algorithm (Election of a leader - a symmetry breaking protocol)

1. Each processor sets its local variable V to n and starts to be *active*.
2. Each active processor chooses, randomly and independently, an integer between 1 and V and put it into V.
3. Those processors that choose 1 (if any), send one-bit message around the ring – clockwise - with the speed of one processor per time unit.
4. After $n-1$ steps each processor knows the number l of processors that chosen 1. If $l=1$, the election ends and the leader introduces himself; if $l=0$, election continues by repeating Step 2. If $l>1$, the only processors remaining active will be those that have chosen 1 in Step 2. They set $V \leftarrow l$ and election continues with Step 2.

CLASSICAL versus QUANTUM RANDOMIZATION

- Exact solvability of the leader election problem for regular graphs with identical node-processors is a celebrated unsolvable problem of classical distributed computing.
- It can be shown that this problem cannot be solved exactly and in bounded time on classical computers even in the case processors know number of nodes (n) and topology of the network.
- However, there is quantum algorithm that runs in $\mathcal{O}(n^3)$ time, its communication complexity is $\mathcal{O}(n^4)$, and it can solve this problem exactly for any network topology, provided parties are connected by quantum communication links.

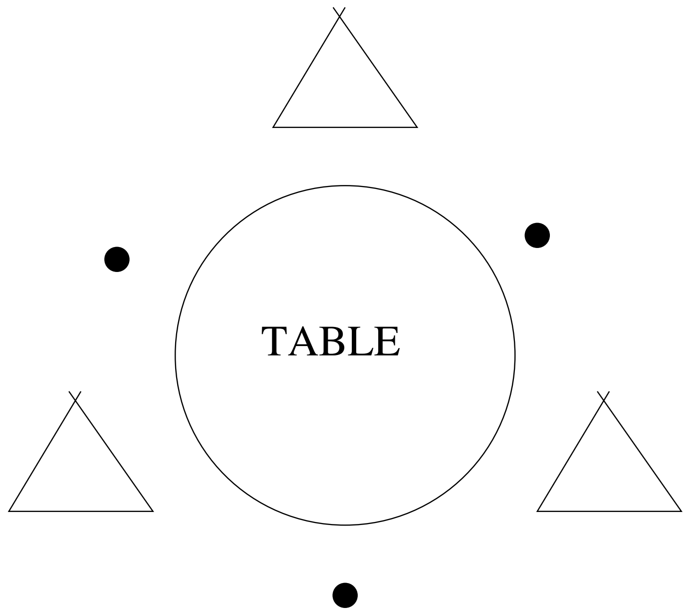
THE DINING CRYPTOGRAPHERS PROBLEM

- **Three cryptographers have dinner at a round table of a 5-star restaurant.**
- **Their waiter tells them that an arrangement has been made that bill will be paid anonymously - either by one of them, or by NSA.**
- **They respect each other's right to make an anonymous payment, but they wonder if NSA has payed the dinner.**
- **How should they proceed to learn whether one of them paid the bill without learning which on e - for other two?**

DINNING CRYPTOGRAPHERS - SOLUTION

- Protocol
  - Each cryptographer flips a perfect coin between him and the cryptographer on his right, so that only two of them can see the outcome.
  - Each cryptographer who did not pay dinner states aloud whether the two coins he see - the one he flipped and the one his right-hand neighbour flipped - fell on the same side or not.
  - The cryptographer who paid the dinner states aloud the opposite what he sees.
- Correctness:
  - Odd number of differences uttered at the table implies that that a cryptographer paid the dinner.
  - Even number of differences uttered at the table implies that NSA paid the dinner.
  - In a case a cryptographer paid the dinner the other two cryptographers would have no idea he did that.

*****



TABLE

TECHNICAL SOLUTION

Let three coin tossings made by cryptographers be represented by bits

$$b_1, b_2, b_3$$

In case none of them payed dinner, what are says are values

$$b_1 \oplus b_2, b_2 \oplus b_3, b_3 \oplus b_1$$

and their parity is

$$(b_1 \oplus b_2) \oplus (b_2 \oplus b_3) \oplus (b_3 \oplus b_1) = 0$$

In case one of them payed dinner, say Cryptographer 2, they say

$$b_1 \oplus b_2, \overline{b_2 \oplus b_3}, b_3 \oplus b_1$$

and

$$(b_1 \oplus b_2) \oplus (\overline{b_2 \oplus b_3}) \oplus (b_3 \oplus b_1) = 1$$

TWO RANDOMIZED ALGORITHMS

Problem: Determine the number, say n, of elements of a bag X, provided you can do, repeatedly, only the following operation: to pick up, randomly, an element of the bag X, to look at it, and to return it back to the bag.

Algorithm:

$k \leftarrow 0$;
do choose randomly an element from X, mark it and return it back; set $k \leftarrow k+1$
until the just chosen element has already been chosen;
$n \leftarrow \left\lfloor \frac{2k^2}{\pi} \right\rfloor$

ZERO POLYNOMIAL TESTING

Problem: Decide whether a given polynomial $p(x_1, \ldots, x_n)$, (given implicitly) with integer coefficients, and with each product of variables being of the degree at most k, is identically 0.

Algorithm:

Compute $p(x_1, \ldots, x_n)$ N times, for sufficiently large N; each time with randomly chosen integer values for $x_1, \ldots, x_n$ from the interval $[0, 2kn]$.

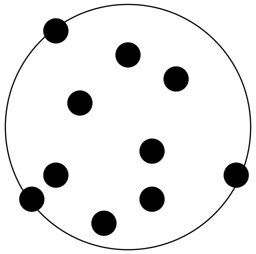
If, at the above process at least once a value different from 0 is obtained, then p is not identically 0.

If all N values obtained are 0, then we can consider p to be identically 0. The probability of error is at most 2^{-N}.

DESIGN of the SMALLEST ENCLOSING DISK

Task Given is a set S of n points on plane. Find the smallest disk (circle) $D(S)$ containing S.

Note $D(S)$ is determined by any three points on its edge.



Naive solution For any three points design a disk/circle passing through them - complexity of such an algorithm is $\mathcal{O}(n^3)$

Random $\mathcal{O}(n)$ algorithm - Welzl

To start consider all points as having weight 1

Algorithm

1. **Choose randomly (taking into considerations weights of points) a set of about 20 points S' and determine, somehow, $D(S')$.**
2. **In case there are points of S that are out of $D(S')$ double their weights and go to Step 1. Otherwise you are done**

RANDOMIZED QUICKSORT

Problem: To sort a set S of n elements we can use the following algorithm.

1. Choose a median y of S.
2. Compare all elements of S with y and divide S into the set S_1 of elements smaller than y and into the set S_2 of the remaining elements.
3. Sort recursively sets S_1 and S_2.

Analysis of the number of comparisons: $T(n)$

$T(n) \leq 2T(\frac{n}{2}) + (c+1)n$

if we can find y in cn steps for some constant c

Solution of the above inequality

$T(n) \leq c'n \lg n$

Asymptotically, the same solution is obtained if we require only that none of the sets S_1, S_2 has more than $\frac{3}{4}n$ elements.

Since there are at least $\frac{n}{2}$ elements y with the last property there is a good chance that if y is always chosen randomly, then we get a good performance.

This way we obtain *random QUICKSORT* or RQUICKSORT.

ANALYSIS of RQUICKSORT (RQS)

Let X be given and let

x_i – be the i-th smallest element of X;

x_{ij} – be a random variable having value 1 if x_i and x_j are being compared (during an execution of the RQS).

Expected number of comparisons of RQS

$$E\left[\sum_{i=1}^{n}\sum_{j=1}^{n} x_{ij}\right] = \sum_{i=1}^{n}\sum_{j=1}^{n} E[x_{ij}]$$

If p_{ij} is the probability that x_i and x_j are being compared during an execution of the algorithm, then $E[x_{ij}] = p_{ij}$.

In order to estimate p_{ij} it is enough to realize that if x_i and x_j are compared during an execution of the RQS, then one of these two elements has to be in the subtree headed by the other element in the comparison tree being created at that execution. Moreover, in such a case all elements between x_i and x_j are still to be inserted into the tree being created. Therefore, at the moment other element (not the one in the root of the subtree), is chosen, it is chosen randomly from at least $|j-i|+1$ elements. Hence $p_{ij} \leq \frac{1}{|i-j|+1}$. Therefore we have (for $H_n = \sum_{i=1}^{n} \frac{1}{i}$):

$$\sum_{i=1}^{n}\sum_{j=1}^{n} p_{ij} \leq \sum_{i=1}^{n}\sum_{j=i}^{n} \frac{2}{j-i+1} \leq \sum_{i=1}^{n}\sum_{k=1}^{n-i+1} \frac{2}{k} \leq$$

$$2\sum_{i=1}^{n}\sum_{k=1}^{n-i+1} \frac{1}{k} \leq 2nH_n = \Theta(n\log n)$$

SATISFIABILITY of BOOLEAN FORMULAS

The following algorithm finds, given a satisfiable Boolean formula F in 3-CNF, with very high probability, a satisfying assignment for F.

Algorithm:
Choose randomly a truth assignment T for F;
while there is a truth assignment T' that differs from T in
exactly one variable and satisfies more clauses of F than T
do choose such of these T' that satisfy the most clauses and set $T \leftarrow T'$ **od**;
return T

How good is this simple algorithm?

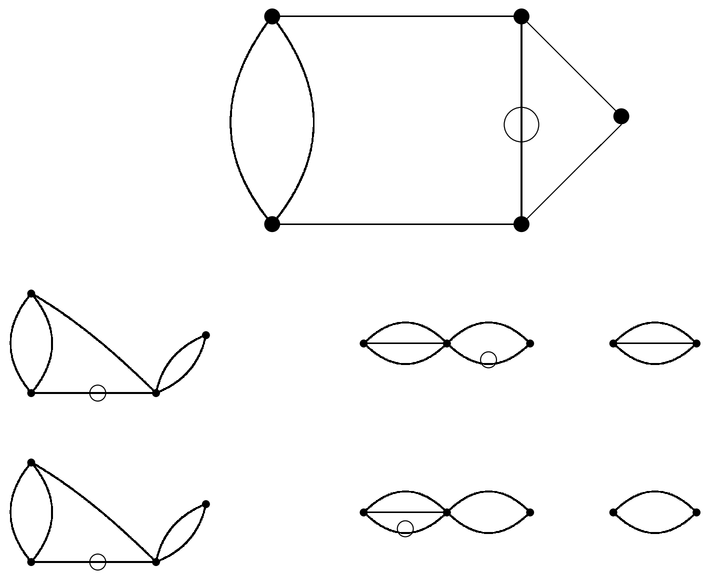
Theorem If $0 < \epsilon < \frac{1}{2}$, then there is a constant c such that for all but a fraction of at most $n2^n e^{-\frac{\epsilon n^2}{2}}$ of satisfiable 3-CNF Boolean formulas with n variables, the probability that the above algorithm succeeds in discovering a truth assignment in each independent trial from a random start is at least $1 - e^{-\epsilon^2 n}$.

CUTS in MULTIGRAPHS - PROBLEM

Given is an undirected and loop-free multigraph G. The task is to find one of the smallest sets C of edges (called a cut) of G such that the removal of edges from C disconnects the multigraph G.

Basic operation is an edge contraction If e is an edge of a loop-free multigraph G, then the multigraph G/e is obtained from G by contracting the edge $e = \{x, y\}$, that is, we identify the vertices x and y and remove all resulting loops.

Example:



CUTS in MULTIGRAPHS - ALGORITHM

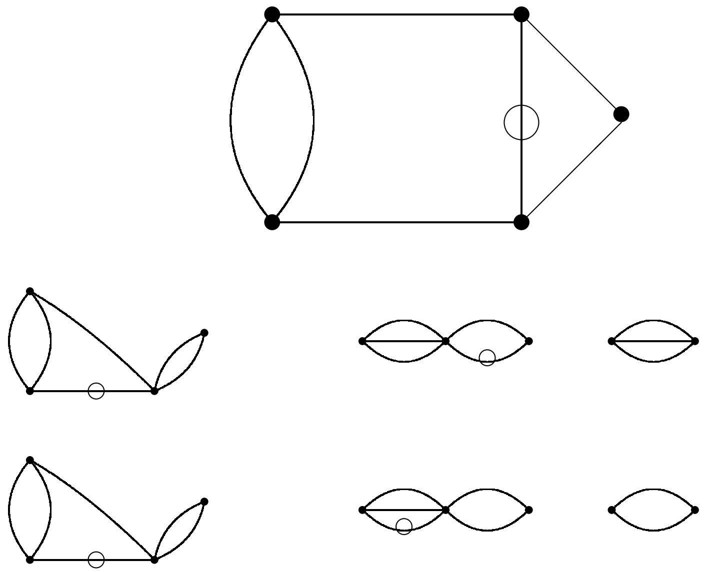
Basic idea of the algorithm given below: An edge contraction of a multigraph does not reduce the size of the minimal cut.

Contract algorithm:

while there are more than 2 vertices in the multigraph
do edge-contraction of a randomly chosen edge **od**

Output the size of the minimal cut of the resulting 2 vertex multigraph.

Example:



In the above example, where two options are explored in the second step, we got once the optimal result, and once a non-optimal result.

HOW GOOD is the ABOVE ALGORITHM?

How probable is that our randomized algorithm produces an incorrect result?

Let G be a multigraph with n nodes and k be the size of its minimal cut;

C - be a particular minimal cut of size k.

Observation: G has to have at least $\frac{kn}{2}$ edges. (Why?)

We derive a lower bound on the probability that no edge of C is ever contracted during an execution of the algorithm.

Let E_i be the event of non-choosing an edge of C at the i-th step of the algorithm.

The probability that the edge randomly chosen in the first step is in C is at most $\frac{k}{\frac{nk}{2}} = \frac{2}{n}$ and therefore $Pr(E_1) \geq 1 - \frac{2}{n}$.

If E_1 occurs, then at the second contraction step there are at least $\frac{k(n-1)}{2}$ edges. Hence $Pr(E_2|E_1) \geq 1 - \frac{2}{n-1}$

Similarly, in the i-th step

$$Pr\left[E_i|\bigcap_{j=1}^{i-1} E_j\right] \geq 1 - \frac{2}{n-i+1} = \frac{n-i-1}{n-i+1}$$

PROOF CONTINUATION

Therefore, the probability that no edge of C is ever contracted during an execution of the algorithm, that is that algorithm gives correct output, can be lower bounded by

$$Pr\left[\bigcap_{i=1}^{n-2} E_i\right] \geq \prod_{i=1}^{n-2}\left(1-\frac{2}{n-i+1}\right) = \prod_{i=1}^{n-2}\left(\frac{n-i-1}{n-i+1}\right) = \frac{2}{n(n-1)} = \Omega(\frac{1}{n^2})$$

Hence, the probability of an incorrect result is $\leq 1-\frac{2}{n(n-1)}$.

Moreover, if the above algorithm is repeated $\frac{n^2}{2}$ times, making each time random decisions, then the probability that a minimal cut is not found is at most

$$\left(1-\frac{2}{n^2-n}\right)^{\frac{n^2}{2}} < \left(1-\frac{2}{n^2}\right)^{\frac{n^2}{2}} = \left(1-\frac{1}{\frac{n^2}{2}}\right)^{\frac{n^2}{2}} < \frac{1}{e}$$

Running time of the best deterministic minimum cut algorithm is $\mathcal{O}(nm+n^2\lg n)$, where m is number of edges and n is number of nodes.

REMINDERS

The following facts are well-known from mathematical analysis:

- $(1+\frac{x}{n})^n \leq e^x$;
- $\lim_{n\rightarrow\infty}(1+\frac{x}{n})^n = e^x$

PRIMES RECOGNITION

The fastest known sequential deterministic algorithm to decide whether a given integer n is prime has complexity $O\left((\lg n)^{14}\right)$

A simple randomized Rabin-Miller's Monte Carlo algorithm for prime recognition is based on the following result from the number theory.

Lemma Let $n \in \mathbf{N}$. Denote, for $1 \leq x < n$, by $C(x)$ the condition:

> Either $x^{n-1} \not\equiv 1 (\bmod\ n)$ or there is an $m \in N$, $m = \frac{(n-1)}{2^i}$ for some i, such that $gcd(n, x^m - 1) \neq 1$

Fact: If $C(x)$ holds for some $1 \leq x < n$, then n is not prime. If n is not prime, then $C(x)$ holds for at least half of x between 1 and n.

In other words most of the numbers between 1 and n are witnesses for composability of n.

Rabin-Miller algorithm

- Choose randomly integers $x_1, \ldots, x_m$ such that $1 \leq x_j < n$;
- For each x_j determine whether $C(x_j)$ holds;
- **if** $C(x_j)$ holds for some x_j;
  then n is not prime
  else n is prime, with probability of error 2^{-m}

RANDOMIZED COMPLEXITY CLASSES

In order to model random steps formally, and to study power of randomization, it is enough to consider nondeterministic Turing machines (NTM), that have in each configuration exactly two choices to make and for each input all computations of which have the same length. To define different complexity classes, one just needs to consider different acceptance modes.

RP: A language L is in $\mathbf{RP}$ (**R**andom **P**olynomial time) if there is a polynomial NTM such that:

- if $x \in L$, then *at least half* of all computations of M on x terminate in the accepting state;
- if $x \notin L$, then *all* computations of M terminate in the rejecting state. (So called *Monte Carlo* acceptance or *one-sided Monte Carlo* acceptance).

PP: A language L is in $\mathbf{PP}$ (**P**robabilistic **P**olynomial time) if there is a polynomial NTM such that: $x \in L$ iff *more than half* of computations of M on x terminate in the accepting state. (So called *acceptance by majority*.)

ZPP: A language is in $\mathbf{ZPP}$ (**Z**ero error **P**robabilistic **P**olynomial time) (So called *Las Vegas* acceptance.)

$$\mathbf{ZPP} = \mathbf{RP} \wedge \mathbf{coRP}$$

ANOTHER VIEW of COMPLEXITY CLASSES

BPP: A language is in $\mathbf{BPP}$ ($\mathbf{B}$ounded error away from $\frac{1}{2}$ $\mathbf{P}$robabilistic $\mathbf{P}$olynomial time), if there is a polynomial NTM M such that:

- If $x \in L$, then at least $\frac{3}{4}$ computations of M on x terminate in the accepting state.
- If $x \notin L$, then at least $\frac{3}{4}$ of computations of M on x terminate in the rejecting state.

Less formally, classes $\mathbf{RP}$, $\mathbf{PP}$ and $\mathbf{BPP}$ can be defined as classes of problems for which there is a randomized algorithm A with the following property:

- **RP:**
  - $x \in L \Rightarrow PR(A(x)$ accepts$) \geq \frac{1}{2}$;
  - $x \notin L \Rightarrow PR(A(x)$ accepts$) = 0$
- **PP:**
  - $x \in L \Rightarrow PR(A(x)$ accepts$) > \frac{1}{2}$;
  - $x \notin L \Rightarrow PR(A(x)$ accepts$) \leq \frac{1}{2}$.
- **BPP:**
  - $x \in L \Rightarrow PR(A(x)$ accepts$) \geq \frac{3}{4}$;
  - $x \notin L \Rightarrow PR(A(x)$ accepts$) < \frac{1}{4}$

BASIC INCLUSIONS between

DETERMINISTIC and RANDOMIZED COMPLEXITY CLASSES

Theorem

$$\mathbf{P} \subseteq \mathbf{ZPP} \subseteq \mathbf{RP} \subseteq \mathbf{NP} \subseteq \mathbf{PP} \subseteq \mathbf{PSPACE}$$

Proof: Since relations $\mathbf{P} \subseteq \mathbf{ZPP} \subseteq \mathbf{RP}$ are obvious, we show first

$$\mathbf{RP} \subseteq \mathbf{NP}$$

If $L \in \mathbf{RP}$ then there is a NTM M accepting L with Monte Carlo acceptance. Hence, $x \in L$ iff M has at least one accepting computation for x. Thus, $L \in \mathbf{NP}$.

$\mathbf{NP} \subseteq \mathbf{PP}$

Let $L \in \mathbf{NP}$ and M be a polynomial NTM for L. Design a NTM M' that for an input w nondeterministicaly chooses and performs one of two steps:

1. M' accepts
2. M' simulates M on input w.

M' can be transformed into an equivalent NTM M'' that always have two choices and all its computations on w have the same length. M'' accepts L by majority $\Rightarrow L \in \mathbf{PP}$. Indeed:
If $w \not\in L$, then exactly half of computations accept – those corresponding to step 1.
If $w \in L$, then there is at least one computation of M that accepts w $\Rightarrow$ more than half of computations of M'' accept. In addition, it holds

$$\mathbf{PP} \subseteq \mathbf{PSPACE}$$

COMPLEXITY CLASS BPP

- Acceptance by clear majority seems to be the most important concept of the randomized computing.
- The number $\frac{3}{4}$ used in the definition of the class $\mathbf{BPP}$ can be replaced by any number larger than $\frac{1}{2}$. In other words, for any $\varepsilon < \frac{1}{2}$ we can say that an BPP-algorithm accepts (rejects) any word from (not from) the underlying language with the probability at least $1 - \varepsilon$
- $\mathbf{BPP}$–algorithms allow to diminish, by a repeated application, the probability of error as much as needed.
- It *seems* that $\mathbf{P} \subsetneqq \mathbf{BPP} \subsetneqq \mathbf{NP}$ and therefore the class $\mathbf{BPP}$ seems to be a reasonable extension of the class $\mathbf{P}$ and as a class of *feasible problems*.

Theorem All languages in $\mathbf{BPP}$ have polynomial size Boolean circuits.

Definition A language $L \subseteq \{0,1\}^\star$ has polynomial size Boolean circuits if there is a family of Boolean circuits $G = \{C_i\}_{i=1}^{\infty}$ and a polynomial p such that size of C_n is bounded by $p(n)$, C_n has n inputs and $x \in L$ iff the output of $C_{|x|}$ is 1 if its i-th input is x_i.

AMPLIFICATION of PROBABILITIES

Let a PTM $\mathcal{M}$ have a probability of error at most $\varepsilon < \frac{1}{2}$. Take k copies of $\mathcal{M}$ and run them for the same input and then make the majority decision. Such a majority voting will be wrong with probability

$$p_{err} \leq \sum_{S\subseteq\{1,\ldots,k\},|S|\leq k/2} (1-\varepsilon)^{|S|}\varepsilon^{k-|S|} \tag{1}$$

$$= ((1-\varepsilon)\varepsilon)^{k/2} \sum_{S\subseteq\{1,\ldots,k\},|S|\leq k/2} \left(\frac{\varepsilon}{1-\varepsilon}\right)^{k/2-|S|} \tag{2}$$

$$< (\sqrt{\varepsilon(1-\varepsilon)})^k 2^k = \lambda^k, \tag{3}$$

where $\lambda = 2\sqrt{\varepsilon(1-\varepsilon)} < 1$, because the above sum is $\leq 2^k$, since $\frac{\varepsilon}{1-\varepsilon} \leq 1$.

In case k is big enough, the effective error probability will be as small as we wish. This process is called **amplification of probability**.

$\mathbf{BPP}$ and $\mathbf{P}$/poly

The result from the last theorem can be stated in a nice form using the concept of *non-uniform* $\mathbf{P}$ complexity class $\mathbf{P}/poly$.

For a Boolean function $f : \{0,1\}^* \longrightarrow \{0,1\}$ and an integer n let f_n be Boolean function defined on $\{0,1\}^n$ by

$$f_n(x_1, \ldots, x_n) = f(x_1 \ldots x_n)$$

and let $c(f_n)$ be the size of the minimal circuit for f_n.

Definition A Boolean function f belongs to the class $\mathbf{P}$/poly if there is a polynomial p such that

$$c(f_n) \leq p(n).$$

MAIN THEOREM

Theorem $\mathbf{BPP} \subseteq \mathbf{P}/poly$.

Proof. Let $f(x)$ be a **BPP**-predicate(language) and $\mathcal{M}$ be a probabilistic TM that determines/decides f with probability at least $1-\varepsilon$. Using repetitive runs we can construct i a polynomial probabilistic TM $\mathcal{M}'$ that decides f with the error probability less than $\varepsilon' = 1/2^n$ for inputs of length n.

$\mathcal{M}'$ uses at each run a random string r (one random bit for each step). For each input x of length n, the fraction of strings r that lead to an incorrect answer is therefore less than $1/2^n$.

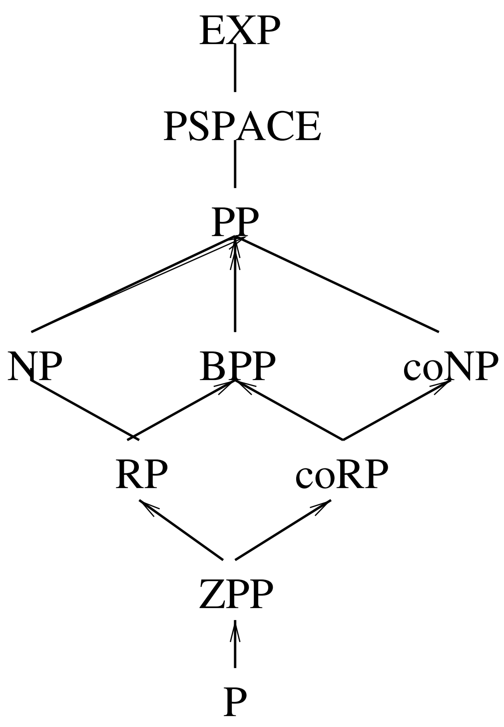
The overall fraction of "bad" pairs (x, r) among all such pairs is therefore less than $1/2^n$. (If one represents the set of all pairs (x, r) as a unit square, the "bad" subset has an area $< 1/2^n$.)

Therefore it has to exist an $r = r_*$ such that the fraction of bad pairs (x, r) is less than $1/2^n$ among all pairs with $r = r_*$. However, there are only 2^n such pairs. Hence, the only way how the fraction of bad pairs (x, r_*) can be smaller than $1/2^n$ is that there are no bad pairs at all!!

Consequently, we have to conclude that that there is a string r_* that produces correct answers for all x of length n,

The machine $\mathcal{M}'$ can therefore be transformed into a polynomial-size circuit with inputs (x, r). Then we fix the value $r = r_*$ and this way we obtain a polynomial size circuit with an input x that decides/computes $f(x)$ for all x of length n.

HIERARCHY of COMPLEXITY CLASSES



CLASS MA

The class $\mathbf{BPP}$ can be seen as a randomized version of the class $\mathbf{P}$. In a similar way the class $\mathbf{MA}$ (Marlin-Arthur) can be seen as a randomized version of the class $\mathbf{NP}$.

$\mathbf{MA}$ is the class of decision problems solvable by a Merlin-Arthur protocol, which goes as follows: Merlin, who has unbounded computational resources, sends Arthur a polynomial-size to-be-proof that the answer to the problem is "yes". Arthur must verify the proof in BPP, so that if the answer to the decision problem is

- "yes", then there exists a proof which Arthur accepts with probability at least $\frac{2}{3}$.
- "no", then Arthur accepts any "to-be-proof" with probability at most $\frac{1}{3}$.

An alternative definition requires that if the answer is "yes", then there exists a proof that Arthur accepts with certainty.

It can be shown that if $\mathbf{P} = \mathbf{BPP}$, then $\mathbf{MA}$=$\mathbf{NP}$.

HOW IMPORTANT is RANDOMNESS

for DESIGN of ALGORITHMS

- The answer depends much on how we define when an algorithm is "efficient".
- If constant factors are of importance, then randomization is clearly of large importance.
- If we consider $\mathcal{O}(n), \mathcal{O}(n^2)$ and also $\mathcal{O}(n^3)$ algorithms as still efficient, but already $\mathcal{O}(n^4)$ algorithms as not, then randomness is still of importance for some problems.
- If "polynomial-time computability" is used for efficiency criterion, we do not know answer yet but we maybe able to claim that randomness is not essential.

- There is a strong evidence that $\mathbf{P} = \mathbf{BPP}$.
- Such assumption is based on results showing that computational hardness of some problems can be used to generate pseudorandom sequences that look random to all polynomial time algorithms.
- Using such techniques Widgerson and Impagliazo showed that $\mathbf{P}{=}\mathbf{BPP}$ if there is a problem computable in an exponential time that requires circuits of exponential size.