University of Ljubljana, Faculty of Computer and Information Science

Probabilistic Analysis and Randomized Algorithms

Prof Marko Robnik-Šikonja

Analysis of Algorithms and Heuristic Problem Solving
Edition 2024

Finding maximum

```
findMax(n) {
  fbest = -∞ ;
  for (i=1 ; i <= n ; i++) {
    fi = check(A[i]) ;
    if (fi > fbest) {
       fbest = fi ;
       process(A[i]) ;
     }
  }
}
```

- $O(n \cdot c_{check} + m \cdot c_{process})$
- worst case analysis
- probabilistic analysis
- randomization

Probabilistic analysis

- assumptions about the input distributions
- indicator random variables

Randomization

- to avoid “bad” input sequences, we intentionally randomize the input

```
void findMax(n) {
   randomly shuffle elements in A
   fbest = 0 ;
   for (i=1 ; i <= n ; i++) {
      fi = check(A[i]) ;
      if (fi > fbest) {
          fbest = fi ;
          process(A[i]) ;
       }
   }
}
```

Randomize the input

```
PERMUTE-BY-SORTING(A)
n = A.length
let P[1..n] be a new array
for i = 1 to n
        P[i] = RANDOM(1, n^3)
sort A, using P as sort keys
```

Randomize the input

```
RANDOMIZE-IN-PLACE(A)
n = A.length
for i = 1 to n
        swap A[i] with A[RANDOM(i, n)]
```

On-line maximum

- on-line maximum: elements arrive one by one, randomly shuffled; we can check them but we can select only one

Find online maximum

```
findMaxOnline(k, n) {
  fbest = -∞ ;
  for (i=1 ; i <= k ; i++) {
    if (score(i) > fbest)
       fbest = fi ;
  }
 for (i=k+1 ; i <= n ; i++) {
    if (score(i) > fbest)
       return(i) ;
  }
  return(n) ;
}
```

- How to select k, that we shall select the best one with the largest probability?
- What is the probability that we select the best one using this strategy?

Summation bounds

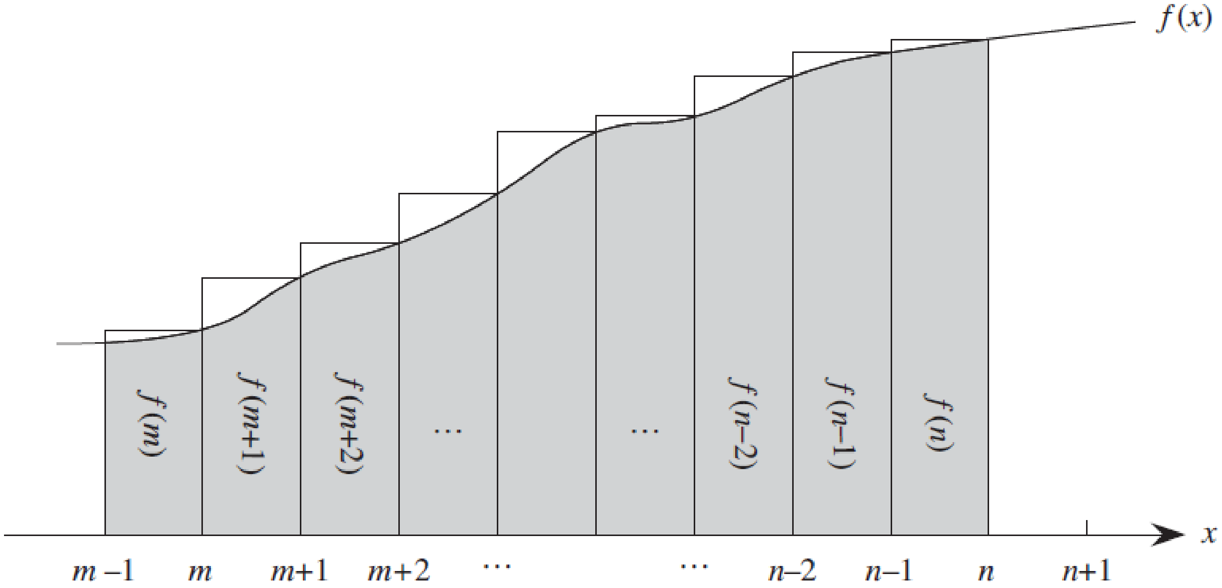
- The sumation $\sum_{k=m}^{n} f(k)$ of monotonously increasing function f(x)

  on an interval from m to n can be bounded by integrals

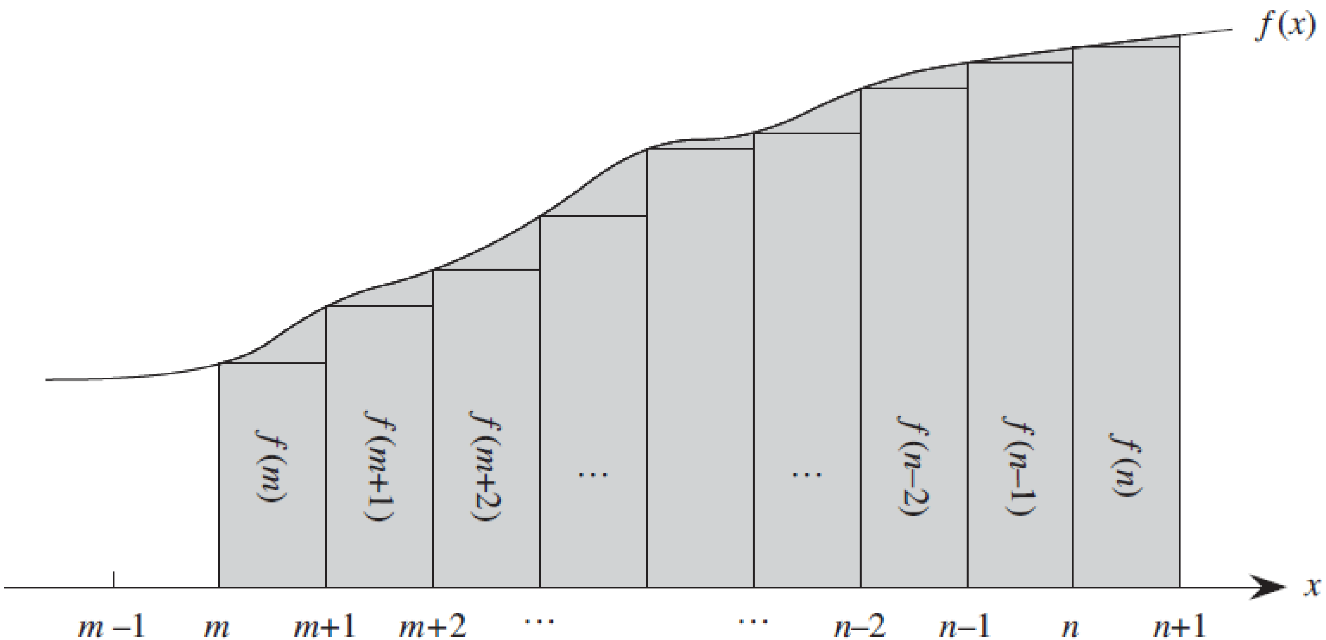
$$\int_{m-1}^{n} f(x)\,dx \leq \sum_{k=m}^{n} f(k) \leq \int_{m}^{n+1} f(x)\,dx$$

- The following figures give an explanation

Lower bound



Upper bound



Monotonically decreasing function

- Similarly to monotonically increasing function, we can show the following relation for monotonically decreasing function

$$\int_{m}^{n+1} f(x)\,dx \le \sum_{k=m}^{n} f(k) \le \int_{m-1}^{n} f(x)\,dx$$

Bounding harmonic series

- In our proof we used harmonic series which is monotonically decreasing therefore

$$\int_{k}^{n} \frac{1}{x}\,dx \le \sum_{i=k}^{n-1} \frac{1}{i} \le \int_{k-1}^{n-1} \frac{1}{x}\,dx$$

Graph min-cut

Contraction algorithm:

repeat {
    select random edge *e=(u,v)*
    contract *e*:
        replace *u* and *v* with super-node *w*
        keep connections of *u* and *v* also for *w*
        keep parallel edges, but not loops
}
until (graph has only two nodes v_1 and v_2)
return cut defined by v_1

- randomized algorithm
- probabilistic analysis

Introduction to pseudo-random numbers

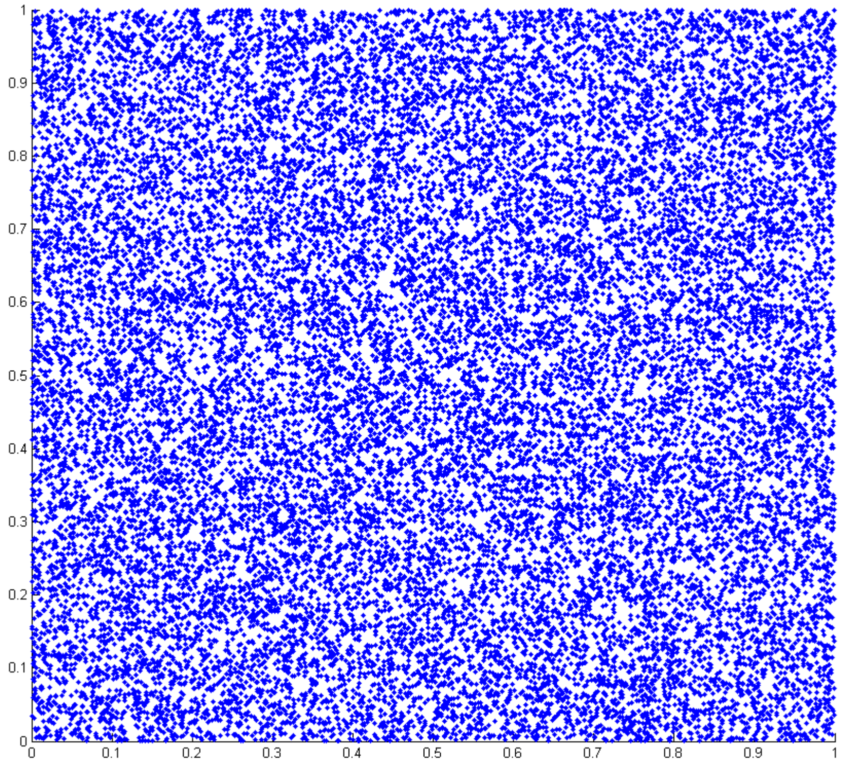
Applications of pseudo random numbers

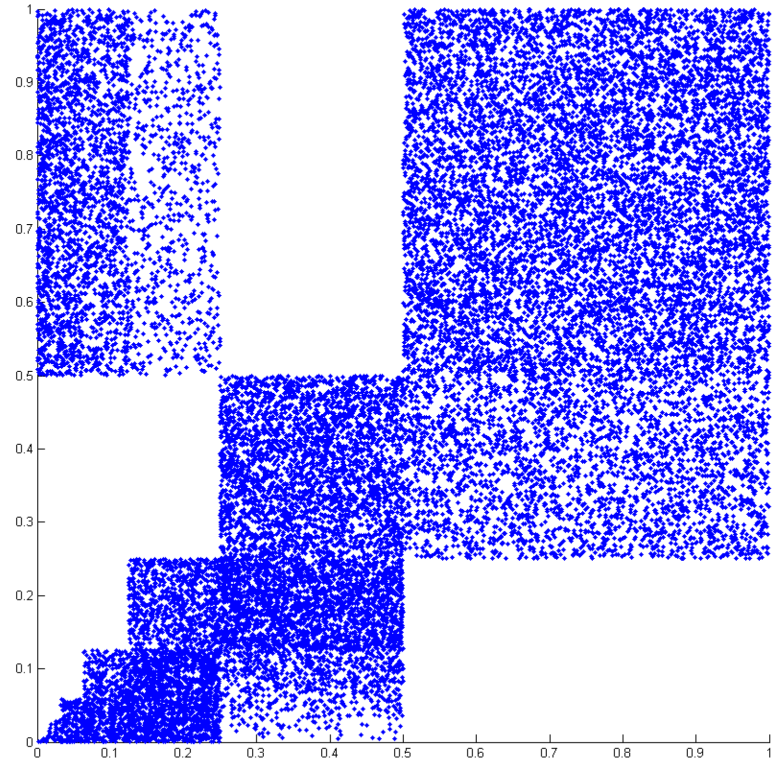
- computer simulations
- cryptography
- statistical sampling and estimation
- Monte Carlo methods
- data analysis and modelling
- computer games
- games of chance

- hardware and software generators
- quality of (pseudo)random numbers: speed and randomness

Matlab example

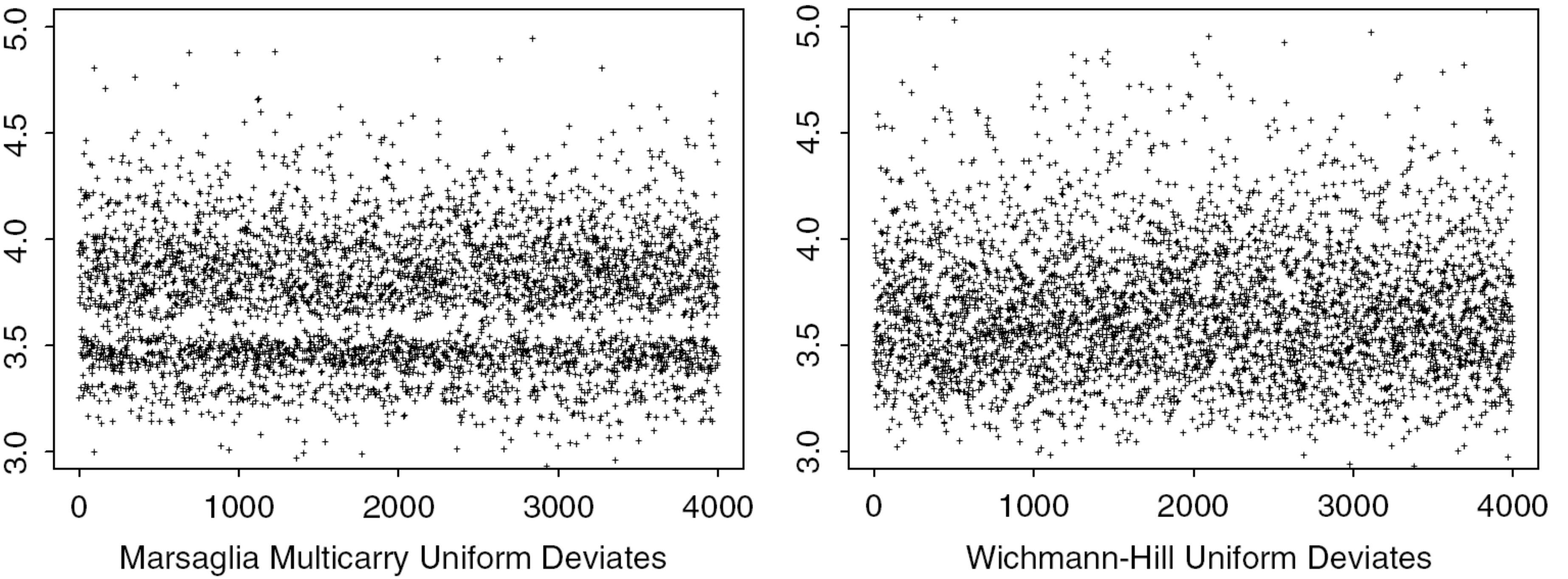
```
Z = rand(28,100000);
condition = Z(1,:) < 1/4;
scatter(Z(16,condition),Z(28,condition),'.');
```





- P. Savicky: A strong nonrandom pattern in Matlab default random number generator. Technical Report, Institute of Computer Science, Academy of Sciences of Czech Republic (2006)

Example



- Value-at-Risk (financial analysis)
  B. D. McCullough: A Review of TESTU01.
  Journal of Applied Econometrics, 21: 677–682 (2006)

Quality criteria

- randomness
- speed of generator
- period

Linear congruential generators

- simplest and most common
  $x_i = (a \cdot x_{i-1} + c) \bmod m \quad u_i = x_i / m$
- A notorious example:
  RANDU:
  $x_i = 65539 \cdot x_{i-1} \bmod 2^{31}$
- simple but bad

MINSTD

- used as a standard for a long time
  $x_i = 16807 \cdot x_{i-1} \bmod (2^{31}-1)$

i	x_i decimal	x_i binary
1	1	1
2	16807	100000110100111
3	282475249	10000110101100011101011110001
4	1622650073	1100000101101111010110011011001
5	984943658	111010101101010000110000101010
6	...	...

Combined linear congruential generator

- combinations of linear congruential generators
- improvements: addition, subtraction, bit mixing
- better randomness, small period

Multiple recursive generators

- higher order recursions
  $x_i = (a_1 \cdot x_{i-1} + \ldots + a_k \cdot x_{i-k}) \bmod m$
  $u_i = x_i / m$

- e.g., (Knuth, 1998):
  $x_i = (271828183 \cdot x_{i-1} + 314159269 \cdot x_{i-2}) \bmod (2^{31}-1)$

- combined multiple recursive generators

Other generators

- combinations
- non-linear generators (quadratic, multiplicative, floating point generators, inverse generators)
- (linear) recursive bit generators (modulo 2, operators)
- cryptographic (ISAAC, AES, BBS,…)
- AES http://en.wikipedia.org/wiki/Advanced_Encryption_Standard

BBS (Blum-Blum-Shrub)

- bit generator
- select two large prime integers p and q (e.g., at least 40 decimal places)
- let m = pq
- $X_i = X_{i-1}^2 \bmod m$
- $b_i = \text{parity}(X_i)$ (0 if even, 1 if odd)
- finding dependency is equivalent to factorization of m (finding multipliers p and q).
- Currently there is no polynomial non-quantum algorithm for integer factorization but it is not proven that such an algorithm does not exist
- the numbers are therefore currently random enough for most uses

Criteria of randomness

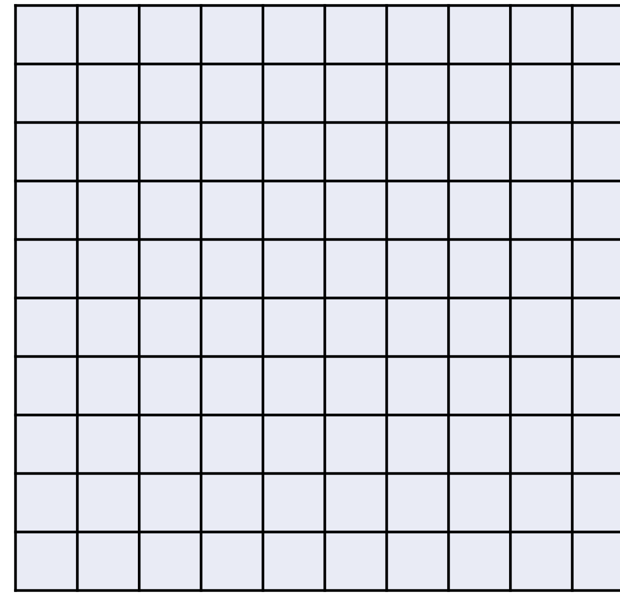
- generate a sequence of t numbers, $u_i \in (0, 1)$
- hypothesis
  $u_0, u_1, \ldots u_{t-1}$ are independent uniformly distributed random variables $U(0,1)$
- equivalent:
  vector $(u_0, u_1, \ldots u_{t-1})$
  is uniformly randomly distributed in unit hypercube $(0,1)^t$
- equivalent: sequence of independent random bits

Statistical tests for randomness

- infinitely many possible tests
- only show dependencies, cannot prove that dependencies do not exists
- increase of trust

- *"The difference between the good and bad RNGs, in a nutshell, is that the bad ones fail very simple tests whereas the good ones fail only very complicated tests that are hard to figure out or impractical to run*."
  L'Ecuyer and Simard, 2007. TestU01: A C Library for Empirical Testing of Random Number Generators*. ACM Transactions on Mathematical Software.*

An example of a test



- Pearson's X^2 goodness-of-fit test
- put generated numbers into k cells (e.g., two-dimensional grid)
- for each cell we know the expected number of elements E_i
- let O_i be the observed number of samples from each cell
- statistics

$$X_0^2 = \sum_{i=1}^{k} \frac{(O_i - E_i)^2}{E_i}$$

- if hypothesis of uniform distribution of numbers is true, the statistics X_0^2 is chi-squared distributed with k-1 degrees of freedom
- we reject the hypothesis if $X_0^2 > X^2_{\alpha,k-p-1}$

Ideas of statistical tests

- one sequence of numbers:
  - tests of groups,
  - gaps,
  - increasing subsequences
- several sequences, hypercube partitioning
  - statistics on partitions
  - statistics on distances
- one sequence of bits
  - cryptographic tests,
  - compressiveness,
  - spectral tests (Fourier),
  - autocorrelation
- several bit sequences

A toolbox of tests

- L'Ecuyer and Simard, 2007. TestU01: A C Library for Empirical Testing of Random Number Generators. *ACM Transactions on Mathematical Software.* [*http://simul.iro.umontreal.ca/testu01/tu01.html*](http://simul.iro.umontreal.ca/testu01/tu01.html)
- results: not many generators pass all tests
- poor results for some popular software (Excel, MATLAB, Mathematica, Java)
- improvements in recent years, e.g., https://www.pcg-random.org/
- hardware generators