Randomness in Algorithms

- Introducing randomness in an algorithm can lead to improved efficiencies
  - Random sampling can provide probabilistically good results with relatively few samples
- Many random algorithms use stochastic simulation as part of their computation – Monte Carlo Methods
  - Exploit randomness to obtain statistical sample of outcomes
- Monte Carlo methods are particularly useful to study
  - Nondeterministic systems
  - Deterministic systems that are too complicated to model
  - Deterministic problems too high dimensional for discretization
Truly random numbers in proper digital logic do not exist.
- There are specialized chips these days that can generate more or less true randoms by observing some physical quantity in the environment.
- We can use algorithms that generate sequences of numbers that have statistical properties as if they were generated by a uniform distribution (pseudo-random and quasi-random number generators).
- Pseudo-random number generators generate a random looking integer between 1 and MAX_RAND.
  - They are actually sequences of numbers, where the seed is the first number in the series.
- This lack of true randomness is actually great for computer simulations: our experiments can be repeated deterministically. We set the seed usually once at the beginning. Different experiments are created by setting different seeds.
Randomness

- Randomness is often defined in terms of
  - Uncompressability
    - The random sequence is the shortest description of itself
  - Unpredictability
    - The next random number is not predictable from the previous ones
  - Not repeatable
    - Random sequences do not repeat (might not always be desirable or achievable)

Random Number Generators

- To be used, the computer needs access to random numbers
  - True random number generators
    - To generate true random numbers, physical processes can be used
      - Radioactive decay
    - Tables with true random sequences can be (and have been) used
  - Pseudo-random number generators
    - Random numbers are generated using a deterministic algorithm
    - Sequence of numbers appears random without knowledge of the algorithm
      - Pseudo-random numbers are predictable if the algorithm is known
      - Pseudo-random numbers are repeatable and reproducible
      - Pseudo-random number sequences will eventually repeat
  - Quasi-random number generators
    - Quasi-random numbers sacrifice randomness of points and focuses on the uniformity of the sample sequence
Pseudo-random Numbers

- A range of algorithms for pseudo-random number generators are used, including:
  - Congruential random number generator
    - Use a very simple equation to calculate the next pseudo-random number (as a Natural number) based on the previous pseudo-random number:
      \[ x_{k+1} = (ax_k + c) \mod m \]
    - Once a number repeats, the entire sequence repeats
  - Fibonacci generator
    - Next pseudo-random number is generated directly as a real number based on two previous pseudo-random numbers (as product, sum, difference, ...):
      \[ x_{k+1} = (x_{k-c_1} \times x_{k-c_2} + c_3) \mod m \]
    - Where \(c_1, c_2, c_3\), and \(m\) are constants.

Linear Congruential PRNG

- LCG-PRNG-s are the most common type of generators:
  \[ x_{k+1} = (ax_k + c) \mod m \]
- Performance depends on the choice of parameters \(a\), \(c\), and \(m\)
  - \(m\) determines the range of numbers that the random number generator can generate. \(m\) is often chosen as the maximum representable number (to minimize repetition)
  - Non-careful choice of \(a\), \(c\), and \(m\) can lead to statistically biased random number sequences
    - One example of this is the randu generator used in early IBM computers: \(a=65539, c=0, m=2^{31}\)
  - For various good \(a,c\), and \(m\) values see: [http://en.wikipedia.org/wiki/Linear_congruential_generator](http://en.wikipedia.org/wiki/Linear_congruential_generator)
Lagged Fibonacci PRNG

- In a LF-PRNG the next pseudo-random number is generated directly as a real number based on two previous pseudo-random numbers (as product, sum, difference, ...)

\[ x_{k+1} = (x_{k-1} \times x_{k-2} + c_3) \mod m \]

- Where \( c_1, c_2, c_3 \), and \( m \) are constants.
- The operator \( \ast \) can represent any operator, e.g.,: addition (ALF), multiplication (MLF), subtraction, XOR.
- The performance strongly depends on the choice of the constants.
- Performance also strongly depends on the seeds(!). Initialization is a very complex problem.

Generating Random Variates

- So, we can generate uniform random integers between 1 and MAX_RAND.
- Thus we can have a semi-continuous random between (0,1).
- How can we use such a random variate to generate random numbers seemingly coming of different distributions?
  - We need to obtain the cdf of the distribution and invert it. Then substituting the (0,1) uniform random into the inverse cdf we get the properly distributed random number.
  - As an example, the exponential distribution’s inverse cdf is:
    \[ F(X)^{-1} = \frac{-\log(1-u)}{\lambda} \] (where \( u \) is the (0,1) random)
How about Generating a Normal?

- Unfortunately, the normal distribution does not have a closed form cdf and thus no closed-form inverse cdf (and the normal is not the only such distribution)
- However, we could sum up a large number (>30) of uniform random variates and scale it to the proper mean and variance.
- There are other approximations:
  - Box-Muller: $X = \sqrt{-2\ln(u)\cos(2\pi v)}$ where $u$, and $v$ are (0,1) uniform random variates.
  - Marsaglia polar: $X = u \sqrt{-\frac{2\ln(u^2+v^2)}{u^2+v^2}}$ where $u$ and $v$ are from (-1,1) uniform and $u^2+v^2<1$ (otherwise regenerate)

Resolution of Random Numbers and Their Impact

- When we create rand()/MAX_RAND, we end up with a floating point number. However not only is this number a rational number (we cannot have irrational numbers in computers), its values are never closer than $1$/MAX_RAND.
- Thus, when substituting into the inverse cdf, the shape of the inverse cdf will determine how close the final random variates’ values will be.
- This is especially significant of a problem for heavy tailed distributions, where theoretically there are larger chances to create very small or very large numbers, which may be prohibited by the resolution of the (0,1) random.
Monte Carlo Methods

- Monte Carlo methods randomly draw samples from a distribution and determining values for each sample
  - Monte Carlo for ratio problems
    - Sample from a distribution and determine the ratio of valid vs. invalid samples to compute the desired ratio
  - Monte Carlo for expected value problems
    - Sample from the distribution and average the function values at the samples to get the expected value over the given distribution
- Monte Carlo methods provide increasingly precise solutions as the number of samples increases but require
  - Knowledge of relevant probability distribution and function
  - Access to good random numbers
Monte Carlo Ratios

• “The usual” rain falls in a square example:
  • If the rainfall is uniform then the number of drops inside the circle vs. the number of total drops gives an estimate for the circle’s area and thus for $\pi$.

• Determining area of a function
  • Similarly, area (integral) of a function can be determined by a Monte-Carlo ratio method.

Simple Rejection Sampling

• A “Monte Carlo Ratios” method.
  • Let’s say we have a random variable $X$ with an “ugly” probability density function $f(X)$.
  • We want to model this variable, i.e., draw sample from an $f(X)$ distribution.
  • Draw two random uniform numbers:
    • $u$ from $U(0,c)$
    • $x$ from $U(a,b)$
  • Accept $x$ as a sample from $X$ iff $u<f(x)$ (reject otherwise)
Generalized Rejection Sampling

- We assumed that \( f(x) \) can be bounded by a rectangle. What if this is not true?
- We need to find a pdf \( g(x) \) that bounds \( f(x) \)
  \[ f(x) \leq g(x) \cdot c \quad \text{for } \forall x \]
- Generate a sample \( x \) from a random variable with a pdf of \( g(x) \).
- Generate a uniform random sample \( u \) from \( U(0,1) \)
  accept \( x \) if \( u \leq \frac{f(x)}{c \cdot g(x)} \)
  reject otherwise

Uses of Monte Carlo Expected Value: Monte Carlo Integration

- Monte Carlo Integration:
  - Recall, that if we sample from a random variable \( X \); according to the strong law of large numbers:
    \[ \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} x_i = E[X] \]
  - We want to integrate an “ugly” function \( g(x) \) from \( a \) to \( b \).
    - Let us introduce a uniform random variable \( U \) with a domain from \( a \) to \( b \). Then:
      \[ \int_{a}^{b} g(x)dx = (b - a)E_U[g(U)] \]
    - Put this two together to have an algorithm for numerical integration with adjustable precision.
Uses of Monte Carlo Expected Value: Importance Sampling

- Our goal is to find the expected value of a function $g(x)$ with respect to a pdf $f(x)$. (This could be done if both $g(x)$ and $f(x)$ “are game”.)

\[ E_f[g(x)] = \int_{-\infty}^{\infty} g(x)f(x) \, dx \]

- However, even if we can sample from $f(x) \{x_1, x_2, \ldots x_N\}$, then:

\[ \int_{-\infty}^{\infty} g(x)f(x) \, dx \to \frac{1}{N} \sum_{i=1}^{N} g(x_i) \]

- What happens if we cannot draw samples from $f(x)$? We introduce another pdf, the sampling distribution: $q(x)$ from which we can draw samples $x_i$:

\[ \int_{-\infty}^{\infty} g(x)f(x) \, dx \to \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{q(x_i)} g(x_i) \]

where \( \frac{f(x_i)}{q(x_i)} \) is the importance weight $w_i$ for sample $x_i$.

Convergence Rate of MC

- As we are calculating the sum of random variables divided by $N$:

\[ E \left[ \frac{x_1 + \cdots + x_N}{N} \right] = \frac{N \mu}{N} = \mu \]

- However, the variance (stdev) is really the one determining how fast we are converging:

\[ Var \left[ \frac{x_1 + \cdots + x_N}{N} \right] = \frac{1}{N^2} N \sigma^2 = \frac{\sigma^2}{N} \]

- The standard deviation (in this case the standard error) is $\frac{\sigma}{\sqrt{N}}$ thus implying $O \left( \frac{1}{\sqrt{N}} \right)$ for the convergence rate.
But, Does it Converge?

\[ E[g(x)] = \frac{1}{N} \sum_{i=1}^{N} f(x_i) \rightarrow \frac{1}{N} \sum_{i=1}^{N} q(x_i) g(x_i) = E[\hat{g}(x)] \]

- As the number of samples \( N \) grows:
  \[ E[g(x) - \hat{g}(x)] = E[g(x)] - E[\hat{g}(x)] \rightarrow 0 \]
  and the variance of the final estimate will decrease.
- However the distribution of the importance weight has a huge impact on the convergence.
- We cannot just look at the convergence of \( E[g(x)] \) to \( E[\hat{g}(x)] \) we need to consider what the variance of \( \hat{g}(x) \) looks like.
  \[ \text{Var}_q[\hat{g}(x)] = \text{Var}_q[g(x)] = \int_{-\infty}^{\infty} g(x)^2 f(x)^2 \frac{dx}{q(x)} \]
- What happens if the nominator has a heavier tail than \( q(x) \)? The selection of \( q(x) \) heavily influences the convergence. Rule of thumb: if \( f(x) \) is not zero \( q(x) \) should not be zero!

Quasi Random Numbers

- Quasi-Random number generators generate numbers that uniformly cover the space but do not individually appear random
  - Consecutive numbers are not unbiased
- An example quasi random number generator:
  - **Base-p low-discrepancy sequence (p is prime).** Have a sequence number \( i \) (the index). For the \( i \)th number use the base-p representation mirrored, and put after a decimal point:
  - **Base-3 is called the Halton sequence**

<table>
<thead>
<tr>
<th>i</th>
<th>B-2 rep</th>
<th>Mirror</th>
<th>In dec</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.1</td>
<td>0.5</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>0.01</td>
<td>0.25</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>0.11</td>
<td>0.75</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>0.001</td>
<td>0.125</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>0.101</td>
<td>0.625</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>0.011</td>
<td>0.375</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>0.111</td>
<td>0.875</td>
</tr>
</tbody>
</table>
**Quasi vs. Pseudo Randoms**

![Pseudo Uniform Random Scatter vs. Quasi Random Scatter](figures.png)

*Figures from Mathworks documentation*

**Impact of Quasi Random Generation on MC**

- For MC random number does not matter as much as that all in all if samples are drawn from a uniform distribution they should look uniform!
- The error of MC ratio methods is:
  \[ Error \propto \frac{1}{N^{1/2+1/(2d)}} \]
- The error of MC expected value methods is:
  \[ Error \propto \frac{(\ln(N))^d}{N} \]
- \( d \) is the “dimensionality” of the quasi random number.