MergeShuffle: A very fast, parallel random permutation algorithm

A. Bacher, O. Bodini, A. Hollender, J. Lumbroso

LIPN, Princeton

Part 1: A little story on random permutation samplers

- Laisant-Lehmer procedure (when **Unif([1,n!])** is available)

Part 1: A little story on random permutation samplers

- Laisant-Lehmer procedure (when **Unif([1,n!])** is available)
- Fisher-Yates algorithm (when **Unif([1,n])** is available)

Part 1: A little story on random permutation samplers

- Laisant-Lehmer procedure (when **Unif([1,n!])** is available)
- Fisher-Yates algorithm (when **Unif([1,n])** is available)
- Rao-Sandelius work (when only **random bit** is available)

Part 1: A little story on random permutation samplers

- Laisant-Lehmer procedure (when **Unif([1,n!])** is available)
- Fisher-Yates algorithm (when **Unif([1,n])** is available)
- Rao-Sandelius work (when only **random bit** is available)
- **MergeShuffle** (when only **random bit** is available)

Laisant (1888)-Lehmer (1960)

- **The idea**. Draw uniformly at random a number in $[1, n!]$

Laisant (1888)-Lehmer (1960)

- **The idea**. Draw uniformly at random a number in $[1, n!]$
- Build a constructive bijection between $[1, n!]$ and S_n the permutations of size n.
- For that, use the following unique factorial representation:

$$\forall 0 \leq k < n!, k = c_1(n-1)! + c_2(n-2)! + ...c_{n-1}1!$$

  where $0 \leq c_i < n - i$.
- The first element of the permutation is $c_1 + 1$, we remove it from $L := \{1, ..., n\}$.
- The second is the $c_2 + 1$-st element in L, we remove it,
- And so on...

Laisant (1888)-Lehmer (1960)

- Example for a permutation in S_4.
- We draw $k = 16$

Laisant (1888)-Lehmer (1960)

- Example for a permutation in S_4.
- We draw $k = 16$
- $k = 2 \times 3! + 2 \times 2! + 0 \times 1!$

Laisant (1888)-Lehmer (1960)

- Example for a permutation in S_4.
- We draw $k = 16$
- $k = 2 \times 3! + 2 \times 2! + 0 \times 1!$
- So, the associated permutations is $[3, 4, 1, 2]$

Laisant (1888)-Lehmer (1960)

- Example for a permutation in S_4.
- We draw $k = 16$
- $k = 2 \times 3! + 2 \times 2! + 0 \times 1!$
- So, the associated permutations is $[3, 4, 1, 2]$
- Drawback: very large uniform sampling, a **lot of costly arithmetic** operations.

Fisher-Yates (1948)

- **The idea**. Begin with $C := [1, \ldots, n]$
  Draw uniformly at random a number k in $[1, \ldots, n]$

Fisher-Yates (1948)

- **The idea**. Begin with $C := [1, ..., n]$
  Draw uniformly at random a number k in $[1, ..., n]$
- Swap(C_k, C_n).

Fisher-Yates (1948)

- **The idea**. Begin with $C := [1, ..., n]$
  Draw uniformly at random a number k in $[1, ..., n]$
- Swap(C_k, C_n).
- Draw uniformly at random a number k in $[1, ..., n-1]$

Fisher-Yates (1948)

- **The idea**. Begin with $C := [1, ..., n]$
  Draw uniformly at random a number k in $[1, ..., n]$
- Swap(C_k, C_n).
- Draw uniformly at random a number k in $[1, ..., n-1]$
- Swap(C_k, C_{n-1}).
- And so on...

Fisher-Yates (1948)

```
Input: $c$: an array with $n \geq 2$ elements
Output: A random permutation on $c$
begin
    for $i := n$ downto $2$ by $-1$ do
        $j :=$ Unif$[1, i]$; swap$(c_i, c_j)$
    end
end
```

Fisher-Yates (1948)

- Advantage: Efficient and easy to implement

Fisher-Yates (1948)

- Advantage: Efficient and easy to implement
- Drawback: Need to have a good uniform sampler and **essentially sequential**.

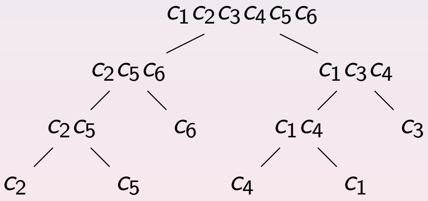
Fisher-Yates (1948): Knuth-Yao

```
Input: a positive integer $n$
Output: $\texttt{Unif}[0, n-1]$
begin
    $u := 1$; $x := 0$;
    while True do
        while $u < n$ do
            $u := 2u$; $x := 2x+$rand-bit();
        $d := u - n$;
        if $x \geq d$ then
            return $x - d$;
        else
            $u := d$;
end
```

Rao (1961) Sandelius (1962)

- **The idea**. Divide and conquer principle
- Partition $\{1, ..., n\}$ in two parts L and R.
- Do it inductively until all part have less than one number.

Rao (1961) Sandelius (1962)



$$\sigma = [2, 5, 6, 4, 1, 3]$$

Rao (1961) Sandelius (1962)

```
Input: $c$: a sequence with $n$ elements
Output: A random permutation on $c$
begin
    if $n \leq 1$ then
        return $c$
    end
    if $n = 2$ then
        if rand-bit() $= 1$ then
            return $(c_2, c_1)$
        else
            return $(c_1, c_2)$
        end
    end
    Let $A_0$ and $A_1$ be two empty arrays
    for $i := 1$ to $n$ do
        add $c_i$ into $A_{rand-bit()}$
    end
    return RS$(|A_0|, A_0)$,RS$(|A_1|, A_1)$
end
```

Rao (1961) Sandelius (1962)

- Advantage: Can be done in place, use only random-bit, easy to parallelize.

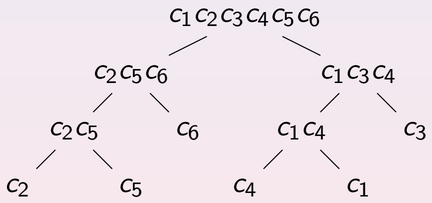
Rao (1961) Sandelius (1962)

- Advantage: Can be done in place, use only random-bit, easy to parallelize.
- Drawback: The **execution time is an unbounded** random variable over $\mathbb{R}^+$.

MergeShuffle (2018)

- **The idea**. Divide and conquer principle
- Partition $\{1, ..., n\}$ in two **balanced** parts L and R.
- Do it inductively until all part have less than one number.
- Merge the results.

MergeShuffle (2018)



$$\sigma = [2, 5, 6, 4, 1, 3]$$

MergeShuffle: How to merge ?

```
Input: $T$, $s$, $n_1$, $n_2$
Output: A random fusion in $T$
begin
    $i \leftarrow s$, $j \leftarrow s + n_1$, $n \leftarrow s + n_1 + n_2$
    while True do
        if rand-bit() = 0 then
            if $i = j$ then Break;
            else
                if $j = n$ then Break;
                Swap($i$, $j$)
                $j \leftarrow j + 1$
            end
            $i \leftarrow i + 1$
        end
    end
    while $i < n$ do
        Draw a random integer $m \in \{s, \dots, i\}$, Swap($i$, $m$), $i \leftarrow i + 1$
    end
end
```

Complexities

Q. How they interact together?

A. Certainly depends on computer evolution !

- Time complexity

Complexities

Q. How they interact together?
A. Certainly depends on computer evolution !

- Time complexity
- Space complexity (not crucial here !)

Complexities

Q. How they interact together?
A. Certainly depends on computer evolution !

- Time complexity
- Space complexity (not crucial here !)
- Random bit complexity
  Low bound using Shannon entropy:
  $\log_2(n!) = n \log_2 n - \frac{n}{\ln(2)} + O(\ln(n))$.

Complexities

Q. How they interact together?

A. Certainly depends on computer evolution !

- Time complexity
- Space complexity (not crucial here !)
- Random bit complexity
  Low bound using Shannon entropy:
  $\log_2(n!) = n \log_2 n - \frac{n}{\ln(2)} + O(\ln(n))$.

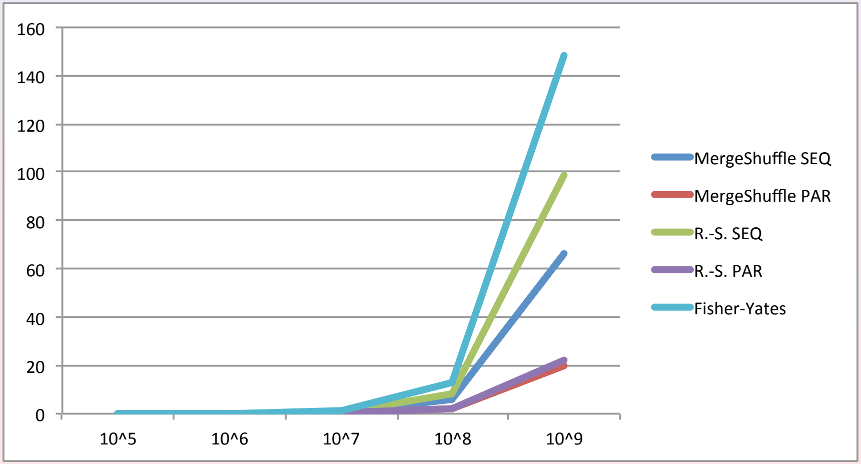
Random bit Complexity

Algorithm	Mean
RS	$n \log_2 n + (0.25 \pm \epsilon)n$
FYKY	$n \log_2 n - (0.33 \pm \epsilon)n$
MergeShuffle	$n \log_2 n + O(n)$

Algorithm	Variance
RS	$(1.83 \pm \epsilon)n$
FYKY	$(1.56 \pm \epsilon)n$

n	10^5	10^6	10^7	10^8
Fisher-Yates	1 631 434	19 550 941	229 329 728	2 628 248 831
MergeShuffle	1 636 560	19 686 051	231 641 075	2 650 387 993
Rao-Sandelius	1 631 519	19 550 449	229 327 120	2 628 251 036

Time Complexity



Time Complexity

On another PC:

n	FYKY	RS	Parallel RS
10^5	4.84ms	4.59ms	4.18ms
10^6	51.1ms	51.6ms	18.5ms
10^7	712ms	623ms	121ms
10^8	12.5s	7.26s	1.04s
10^9	145s	81.7s	10.3s

Time Complexity

THANK YOU