

Generating Random Permutations in Networks

Artur Czumaj

Department of Computer Science &
DIMAP (Centre for Discrete Maths and its Applications)
University of Warwick

BAD Bristol Feb 2010

Random Permutations

- Task:

Input: a set of n numbers $\{1, 2, \dots, n\}$

Output: permutation π of $\{1, 2, \dots, n\}$ ($\pi \in \mathcal{S}_n$)

such that for every $\pi' \in \mathcal{S}_n$

$$\Pr[\pi = \pi'] \approx 1/n!$$

Random Permutations

- Task:

Input: a set of n numbers $\{1, 2, \dots, n\}$

Output: permutation π of $\{1, 2, \dots, n\}$ ($\pi \in S_n$)

such that for every $\pi' \in S_n$

$$\Pr[\pi = \pi'] \approx 1/n!$$

Classical problem
Lots of applications

Random Permutations

- Classical algorithm:

```
Initially,  $\pi = \text{id}$   
for  $i=1$  to  $n$  do  
    choose  $j$  independently and uniformly at random from  $\{i, \dots, n\}$   
     $\pi(i) \Leftrightarrow \pi(j)$ 
```

Random Permutations

- Classical algorithm:

```
Initially,  $\pi = \text{id}$   
for  $i=1$  to  $n$  do  
    choose  $j$  independently and uniformly at random from  $\{i, \dots, n\}$   
     $\pi(i) \Leftrightarrow \pi(j)$ 
```

Claim: Permutation π is random

Algorithm runs in $O(n)$ time
($O(n \log n)$ bit complexity)

What does it mean an almost random permutation?

- Total variation distance is small:

$$\sum_{\pi \in S_n} \left| \Pr[\pi] - \frac{1}{n!} \right| \leq 1/n^9$$

Read: almost all permutations will have the probability of being output $\sim 1/n!$

Can we generate Random Permutations in parallel?



- Modified classical algorithm (random transpositions):



Can we generate Random Permutations in parallel?



- Modified classical algorithm (random transpositions):

Initially, $\pi = \text{id}$

Repeat until it's needed:

choose i, j independently and uniformly at random from $\{1, \dots, n\}$

$\pi(i) \Leftrightarrow \pi(j)$



Can we generate Random Permutations in parallel?



- Modified classical algorithm (random transpositions):

Initially, $\pi = \text{id}$

Repeat until it's needed:

choose i, j independently and uniformly at random from $\{1, \dots, n\}$

$\pi(i) \Leftrightarrow \pi(j)$

$O(n \log n)$ rounds suffice!



Sequential algorithms/processes

Top in at Random

Initially, $\pi = \text{id}$

Repeat as long as needed:

choose j independently and uniformly at random from $\{1, \dots, n\}$

“put $\pi(1)$ into j th position”

{ $x = \pi(1)$
for $i=1$ to $j-1$ do $\pi(i) = \pi(i+1)$ }

Sequential algorithms/processes

Top in at Random

Initially, $\pi = \text{id}$

Repeat as long as needed:

choose j independently and uniformly at random from $\{1, \dots, n\}$

“put $\pi(1)$ into j th position”

{ $x = \pi(1)$
for $i=1$ to $j-1$ do $\pi(i) = \pi(i+1)$ }

$O(n \log n)$ steps

Sequential algorithms/processes

Top in at Random

Initially, $\pi = \text{id}$

Repeat as long as needed:

choose j independently and uniformly at random from $\{1, \dots, n\}$

“put $\pi(1)$ into j th position”

Sequential algorithms/processes

Top in at Random

Initially, $\pi = \text{id}$

Repeat as long as needed:

choose j independently and uniformly at random from $\{1, \dots, n\}$

“put $\pi(1)$ into j th position”

Proof: Follow the last card

Stays at the bottom until $j=n$ for the first time; $O(n)$ steps;

Stays as the second bottom until $j=n-1$ for 1st time; $O(n/2)$ steps;

Property: card below it are randomly permuted!

Stays as the 3rd from the bottom until $j=n-2$... $O(n/3)$ steps

...

When it becomes the top card, all other cards are randomly permuted

Hence: next step will give a random permutation

$E[\# \text{steps}] = n + n/2 + n/3 + \dots + 1 = \Theta(n \log n)$

Sequential algorithms/processes

Random Transposition with Top

Initially, $\pi = \text{id}$

Repeat as long as needed:

choose j independently and uniformly at random from $\{1, \dots, n\}$

$\pi(1) \Leftrightarrow \pi(j)$

Sequential algorithms/processes

Random Transposition with Top

Initially, $\pi = \text{id}$

Repeat as long as needed:

choose j independently and uniformly at random from $\{1, \dots, n\}$

$\pi(1) \Leftrightarrow \pi(j)$

$O(n \log n)$ steps

Sequential algorithms/processes

Random Adjacent Transpositions

Initially, $\pi = \text{id}$

Repeat as long as needed:

choose j independently and uniformly at random from $\{1, \dots, n-1\}$

$\pi(j) \Leftrightarrow \pi(j+1)$ with probability $1/2$

Sequential algorithms/processes

Random Adjacent Transpositions

Initially, $\pi = \text{id}$

Repeat as long as needed:

choose j independently and uniformly at random from $\{1, \dots, n-1\}$

$\pi(j) \Leftrightarrow \pi(j+1)$ with probability $1/2$

$\Theta(n^3 \log n)$ steps (Wilson)

Took a few decades to give tight analysis

“Fast processes” (parallel)

Initially, $\pi = \text{id}$

Repeat as long as needed:

for every $j=1, \dots, n$

flip a coin and decide if j wants to participate

flip a coin and decide if j wants to go left or right

for every $j=1, \dots, n$

if j wants to participate then

if j wants to go left then

if $j-1$ wants to participate and $j-1$ wants to go right then

$$\pi(j) \Leftrightarrow \pi(j-1)$$

else { j wants to go right }

if $j+1$ wants to participate and $j+1$ wants to go left then

$$\pi(j) \Leftrightarrow \pi(j+1)$$

Random Permutations in Distributed Setting



- Distributed mixing (C et al, SODA'99)

Distributed mixing

- Each person tosses a coin and accordingly decides to be either active or passive during this step
- Each active person chooses i.u.r. a partner
- If A has chosen P, then P and A accept themselves if and only if P is passive and A is the only person who has chosen P
- Each pair of partners that accepted themselves tosses independently a coin and accordingly either exchange the items or keep them

Distributed mixing

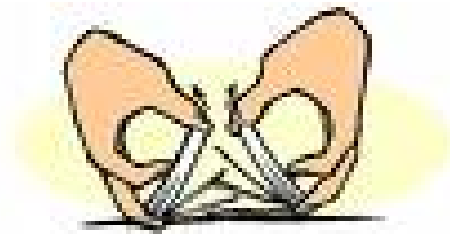
- Each person tosses a coin and accordingly decides to be either active or passive during this step
- Each active person chooses i.u.r. a partner
- If A has chosen P, then P and A accept themselves if and only if P is passive and A is the only person who has chosen P
- Each pair of partners that accepted themselves tosses independently a coin and accordingly either exchange the items or keep them

Theorem: After $O(\log n)$ rounds of Distributed mixing, the input configuration is randomly permuted whp

“Professional shuffling” Riffle Shuffle

Repeat as long as needed:

- cut the deck of card into two approximate halves
- interleave the halves



“Professional shuffling” Riffle Shuffle

Repeat as long as needed:

- cut the deck of card into two approximate halves
- interleave the halves



How many rounds are needed?
(Asked, among others, by American Bridge Association)

“Professional shuffling” Riffle Shuffle

Repeat as long as needed:

- cut the deck of card into two “halves” with the size of the first having binomial distribution
- until both “halves” are empty
- drop the top card from one of the left/right “halves” with the probability proportional to the current “halves” size

“Professional shuffling” Riffle Shuffle

Repeat as long as needed:

- cut the deck of card into two “halves” with the size of the first having binomial distribution
- until both “halves” are empty
- drop the top card from one of the left/right “halves” with the probability proportional to the current “halves” size

Bayer and Diaconis'93
5-7 shuffles suffice for $n=52$

$\sim \log n$

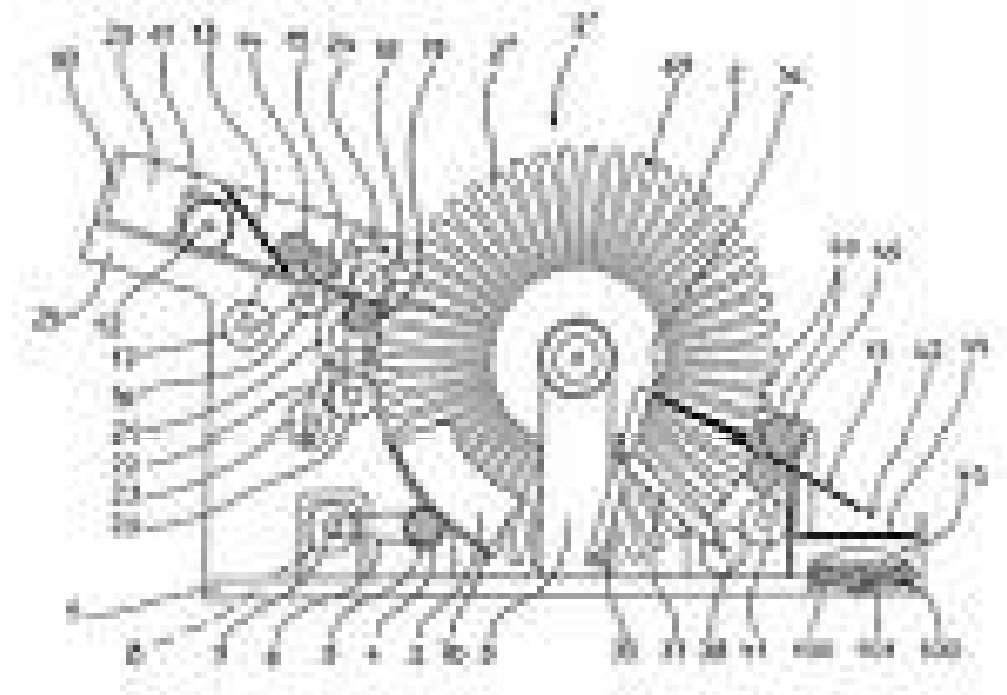
Oblivious shuffling

- In real life (professional or semi-professional games) machines are used
- Machine = oblivious mechanism



Oblivious shuffling

- In real life (professional or semi-professional games) machines are used
- Machine = oblivious mechanism



Thorp shuffle

- Let n be even

Repeat as long as needed:

- cut the deck of card into exactly two halves
- until both halves are empty
- drop the top card from the left/right half according to the outcome of a fair coin flip; then drop from the other half

Thorp shuffle

- Let n be even

Repeat as long as needed:

- cut the deck of card into exactly two halves
- until both halves are empty
- drop the top card from the left/right half according to the outcome of a fair coin flip; then drop from the other half

How many rounds are needed?

Thorp'73

Thorp shuffle

- Aldous & Diaconis [1987] conjectured that after a polylogarithmic number of shuffles the obtained permutation of cards will be almost random
- Conjectured bound: $\Theta(\log^2 n)$

Thorp shuffle

- Aldous & Diaconis [1987] conjectured that after a polylogarithmic number of shuffles the obtained permutation of cards will be almost random
- Conjectured bound: $\Theta(\log^2 n)$
- Only recently Morris (STOC'05) use the method of evolving sets to prove that the mixing time of the Thorp's shuffle on $n = 2^d$ cards is $O(\log^{44} n)$



Thorp shuffle

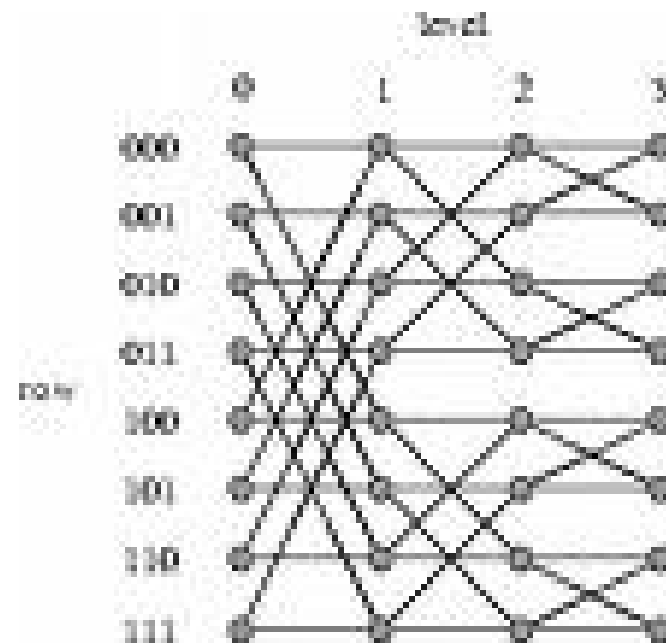
- Aldous & Diaconis [1987] conjectured that after a polylogarithmic number of shuffles the obtained permutation of cards will be almost random
- Conjectured bound: $\Theta(\log^2 n)$
- Only recently Morris (STOC'05) use the method of evolving sets to prove that the mixing time of the Thorp's shuffle on $n = 2^d$ cards is $O(\log^{44} n)$
- Montenegro and Tetali (2006) improved to $O(\log^{29} n)$

Thorp shuffle

- Aldous & Diaconis [1987] conjectured that after a polylogarithmic number of shuffles the obtained permutation of cards will be almost random
- Conjectured bound: $\Theta(\log^2 n)$
- Only recently Morris (STOC'05) use the method of evolving sets to prove that the mixing time of the Thorp's shuffle on $n = 2^d$ cards is $O(\log^{44} n)$
- Montenegro and Tetali (2006) improved to $O(\log^{29} n)$
- Very recently, Morris again improved the analysis and showed the mixing time of $O(\log^4 n)$, without the assumption that n is a power of two

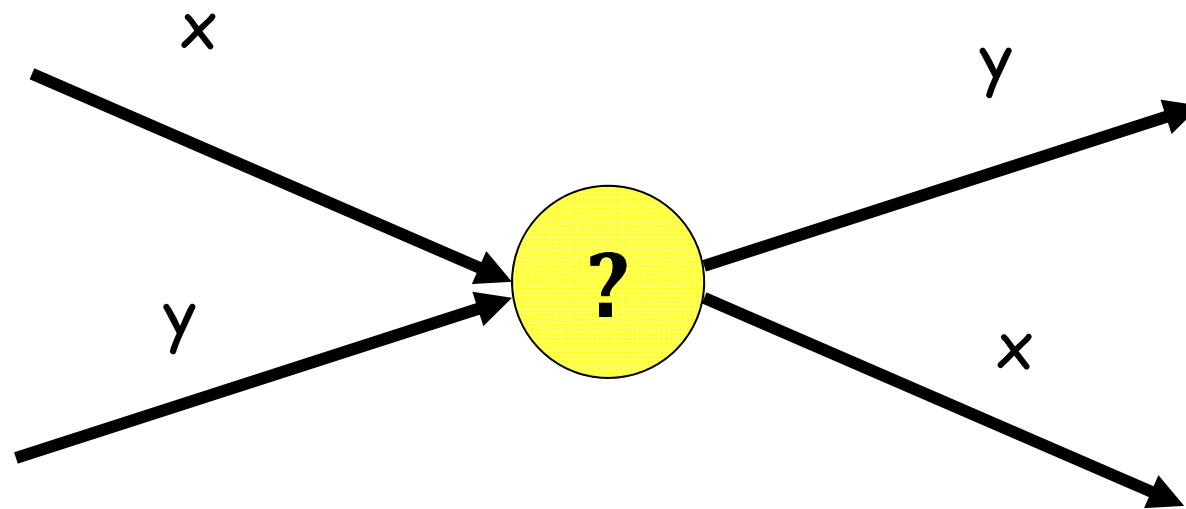
Thorp shuffle and butterfly networks

- If n is a power of 2, then Thorp's shuffle can be modelled by a butterfly network



Random permutation networks

- A layered network, like a sorting network
- Each node (not from input/output layers) has two inputs and two outputs



- Output = random permutation of the two inputs

Thorp shuffle vs butterfly

- For n being a power of 2
- $\log_2 n$ Thorp's shuffles can be modelled by a run on a single butterfly random permutation network
- This gives nice intuitions

Morris result (2009)

- $O(\log^3 n)$ butterflies suffice to random permute

Conjecture of Aldous: $O(\log n)$ should do

What can we do (Czumaj & Voecking; 2010)

- 0-1 random permutations can be obtained by using only $O(\log n)$ random butterflies
- k-partial n-permutation: permutation of a set containing $n-k$ zeros and $\{1, 2, \dots, k\}$
- For every k with $n-k = \Theta(n)$,
random k-partial n-permutations can be obtained by using $O(\log n)$ random butterflies

Applications

- There is a random permutation network of **depth $O(\log^3 n)$**
- One can generate random n -permutations on an EREW PRAM model with $n \log^3 / \log \log n$ processors in **time $O(\log \log n)$**

Think that even computing OR of n bits requires $\Omega(\log n)$ with any number of processors

Shallow random permutation networks

- Use $O(\log n)$ butterflies to select which elements will be in top half, and which will be in the bottom half
 - This is equivalent to permuting $n/2$ zeros and $n/2$ ones, which can be done with $O(\log n)$ butterflies
- “Recursively” permute top half and bottom half independently
 - To maintain the error term (distance from random permutation), each, even the smallest recursive call, needs to have depth $O(\log n)$

Random permutation on EREW PRAM using random permutation networks



- Random permutations are “oblivious”
(idea from C. et al.)

- First, do all random choices for the “switches” in the network
- Once the switches has been set, we only have to find n paths (of length $\text{poly-log}(n)$) from the sources to the destinations
- Standard pointer jumping technique can do it in time $O(\log [\text{depth of the network}])$

The idea of the proof of randomly permuting partial permutations



- Non-Markovian coupling
- Show that if two copies differ by a single “swap”, after $O(\log n)$ butterflies they’ll be the same [can be coupled]
- Key trick:
- Since $\Theta(n)$ elements are indistinguishable, the outcome of the switches between them is irrelevant
- These elements form a buffer to put the misplaced elements in the right position

Open questions Challenges



- Thorp shuffle “must” require $\Theta(\log^2 n)$ shuffles
- 0-1 lemma
- Shallow random permutation networks

Thorp shuffle “must” require $\Theta(\log^2 n)$ shuffles

- ... Or not?
- The following algorithm requires $O(\log \log n)$ rounds:

Repeat until needed

- Randomly permute top half
- Randomly permute bottom half
- for each $i=1, \dots, n/2$
 - with prob. $\frac{1}{2}$, swap the i th element with element $i+n/2$

0-1 Lemma

- It's known that if a sorting network sorts zeros and ones only, it sorts all the numbers
- Is it true for random permutation networks?
 - Most likely no
- Under which conditions it's true?

Shallow random permutation networks

- Can we design a random permutation network of depth $O(\log n)$?
- Should be possible
- Should be very difficult to prove that it works

Typical network will randomly permute 0-1

Theorem [C. & Kutylowski'00]: A randomly chosen random permutation network will randomly permute any input sequence of 0s and 1s.

Earlier, Rackoff and Simon (STOC'93) showed that $\log^{0(1)}n$ depth will suffice

Applications of random permutation networks

- Cryptography
- Cryptographic defense against traffic analysis
(Protecting messages against an eavesdropper.)
- Communication services
- Fault resistant systems