A Probabilistic Analysis of the Reduction Ratio in the Suffix-Array IS-Algorithm

CPM'15, Ischia Island, Italy

Cyril Nicaud

LIGM – Université Paris-Est & CNRS
Marne-la-Vallée

July 2015

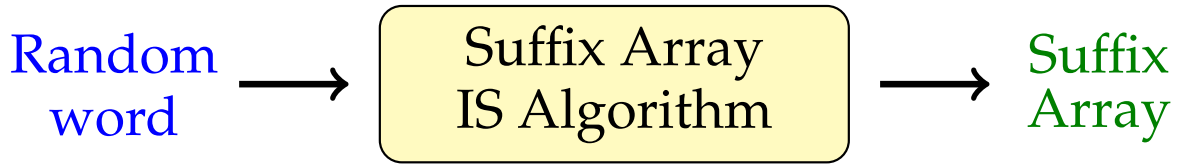
Average case analysis of algorithms

- $\mathcal{I}_n$ is the set of inputs of size n
- Worst case: $\max_{I \in \mathcal{I}_n} C(I)$
- Average case: $\mathbb{E}_n[C] = \sum_{I \in \mathcal{I}_n} p_n(I)\, C(I)$

Techniques for average case analysis of algorithms:

- Discrete probabilities: concentration inequalities, Markov Chains, ...
- Analytic combinatorics: generating series, singularity analysis, ...

Analysis of a suffix-array algorithm



Settings:

- We consider a specific suffix array algorithm
- The input is a random word, either produced by a memoryless or markovian source
- We analyze the typical reduction ratio of the algorithm: it recursively computes the suffix array of an input of size γn, γ is the reduction ratio

Techniques:

- Discrete probabilities: Markov Chains

Words and random words 1/3

Words:

- the alphabet is $A = \{a_1, \ldots, a_k\}$ or $A = \{a, b, c, \ldots\}$
- a word $u = u_0 \cdots u_{n-1}$ is a finite sequence of letters

Memoryless source:

- Let p be a probability mass on A: $\sum_{a \in A} p(a) = 1$
- The probability mass on A^n is

$$p_n(u_0 \cdots u_{n-1}) = p(u_0) \cdots p(u_{n-1})$$

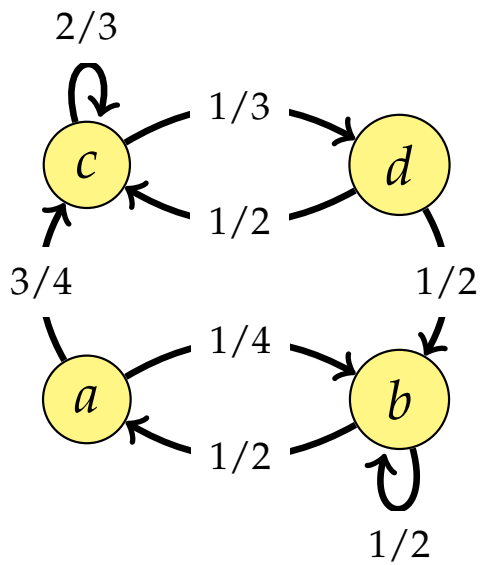
- Each letter is drawn independently following p
- if $p(a) = \frac{1}{k}$ for every $a \in A$, this yields the uniform distribution

Words and random words 2/3

Markovian source (of order 1):

- Markov chain M of set of vertices A
- Initial probability π_0
- The probability mass on $u = u_0 \cdots u_{n-1}$ is

$$p(u) = \pi_0(u_0) M(u_0, u_1) \cdots M(u_{n-2}, u_{n-1}).$$



- $\pi_0(a) = \pi_0(b) = \pi_0(c) = \pi_0(d) = \frac{1}{4}$
- $p(accdb) = \frac{1}{4}\frac{3}{4}\frac{2}{3}\frac{1}{3}\frac{1}{2} = \frac{1}{48}$

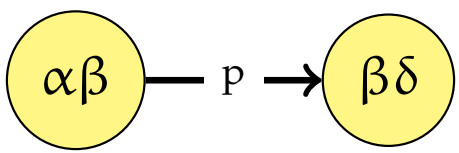
Words and random words 3/3

Markovian source (of order 2):

- Markov chain M of set of vertices $A \times A$
- Initial probability π_0 defined on $A \times A$
- The probability mass on $u = u_0 \cdots u_{n-1}$ is

$$p(u) = \pi_0(u_0u_1)M(u_0u_1, u_1u_2) \cdots M(u_{n-3}u_{n-2}, u_{n-2}u_{n-1}).$$

- Edges are of the form



where p is the probability of producing δ when the last two produced letters are $\alpha\beta$

Models using sources

snhhy”ee etrhsaetretntre ulihPrrb, iItpebyclpEelodHs sam”guta oxeyosrHo sn- giou h,tlen’”eetotne npo”rFgAs detelo y euoItAodba c temIsye nniig

Memoryless

hes ong y-thicooner t a deny kerexirerede. n’t me, d yopppock sthemburoisiskedad, he It, ” llidered Cl, te iech ” R ps eshen-thed frra are oun wos ” s!”waneoicur! We

First order

cas exactis the ableat hims Sir sudy. ”I agan’t the torriblet-tingreme. Amois hissughtfamew musirother capaut a lack.” Poirothe I dow, colitit. Yousee?” ”Ah!” Haske

Second order

azing his fact her jewels arried to speaking, of Dr. Graham, Richard, as if your guest nighter, and drank you want me tin. ”I’ll she fore the man of the did not

Fourth order

Suffix array 1/2

- u is a word that ends with the letter \$
- \$ is a sentinelle, it is smaller than any other letter
- suffixes are identified by there starting position in u:

$$s_i = u_i u_{i+1} \cdots u_{n-1}$$

- Computing the suffix array SA of u consists in sorting the (nonempty) suffixes of u for the lexicographic order

$$u_{SA[0]} < u_{SA[1]} < \ldots < u_{SA[n-1]}$$

i	s	c	h	i	a	\$
0	1	2	3	4	5	6

SA	6	5	2	3	4	0	1
	0	1	2	3	4	5	6

Suffix array 2/2

- Suffix arrays are fundamental data structures
- Used for indexing of a text:
  - A factor of u is a prefix of a suffix
  - Once the suffix array is computed, binary search is possible in $\mathcal{O}(|\text{pattern}| + \log n)$.
- Used to compute the Burrows-Wheeler transform of u
  - data compression: `bzip2`
  - FM-index
- ...

Theorem (03)

The suffix array can be computed *directly* in linear time.

[Kärkkäinen, Sanders ICALP'03] [Kim, Sim, Park, Park CPM'03] [Ko, Aluru CPM'03]

Some definitions

Definition

Let u be a word that ends with \$. A position $0 \leq i \leq n-1$ is

- of type S if $i = n-1$ or $s_i <_{\text{lex}} s_{i+1}$
- of type L if $i \neq n-1$ and $s_i >_{\text{lex}} s_{i+1}$

Definition

A position $1 \leq i \leq n-1$ is an LMS-position (LeftMost S) iff i is of type S and $i-1$ is of type L.

Definition

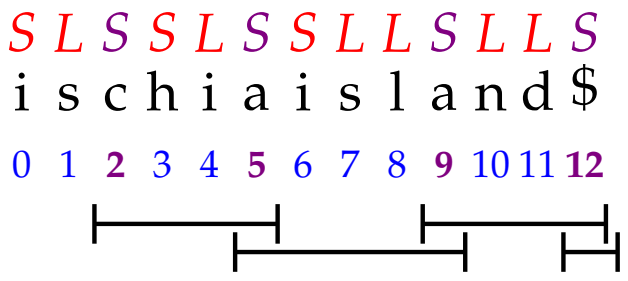
An LMS-factor of u is either \$ or $u[i,j] = u_i \cdots u_j$ where i and j are two consecutive LMS-positions.

An example

Definitions (short version):

- L when larger than the next suffix (for the lex. order)
- S when smaller than the next suffix (for the lex. order)
- LMS when S and previous position is L
- LMS-factor factor between two LMS-positions

Example:



The LMS-factors are:

chia	**aisla**
and$	$

IS-ALGORITHM [Nong, Zhang, Chan DCC'09]

Key observations:

- If we know the relative order of LMS-suffixes, then we can compute where each L-position is in SA in $\mathcal{O}(n)$ time
- Once the L-positions are placed in SA, the S-positions can be determined in $\mathcal{O}(n)$ time

Ordering the LMS-suffixes:

- Order the LMS-factors
- Rename the LMS-factors with integers, respecting the order
- Recursively call **IS-Algorithm** on the sequence of integers

Running time:

- The number of LMS-factors is at most $n/2$
- $T(n) \leq T(n/2) + \mathcal{O}(n)$
- Running time in $\mathcal{O}(n)$

Reduction ratio

- Done using just one scan from left to right or from right to left:
  - Marking each position as L or S, computing the LMS-factors
  - Placing the L-positions in SA from the ordered LMS-suffixes
  - Placing the S-positions in SA from the L-positions
- More complicated:
  - Sorting the LMS-factors $\leftarrow$ clever algorithm in [NZC 09]
  - Recursive call to **IS-Algorithm**

Definition

The reduction ratio of a word u of length n is its number of LMS-positions divided by n.

- If the reduction ratio is γ, then the first recursive call is made on a word of length γn
- If $T(n) = T(\gamma n) + \beta n$ then $T(n) \approx \frac{\beta}{1-\gamma} n$

Main focus

- In the original paper [NZC'09]:

> **Theorem 3.15.** *Given the probabilities for each character to be S-type or L-type are i.i.d as 1/2, the mean size of a nonsentinel LMS-substring is four, i.e, the reduction ratio is not greater than 1/3.*

Main focus

- In the original paper [NZC'09]:

> **Theorem 3.15.** *Given the probabilities for each character to be S-type or L-type are i.i.d as 1/2, the mean size of a nonsentinel LMS-substring is four, i.e, the reduction ratio is not greater than 1/3.*

- Can we say something about the typical (or expected) **reduction ratio** for classical model of random words?

Main focus

- In the original paper [NZC'09]:

> **Theorem 3.15.** *Given the probabilities for each character to be S-type or L-type are i.i.d as 1/2, the mean size of a nonsentinel LMS-substring is four, i.e, the reduction ratio is not greater than 1/3.*

- Can we say something about the typical (or expected) **reduction ratio** for classical model of random words?

Theorem

For memoryless or markovian sources the **reduction ratio** can be computed explicitely.

- Let us start with memoryless sources

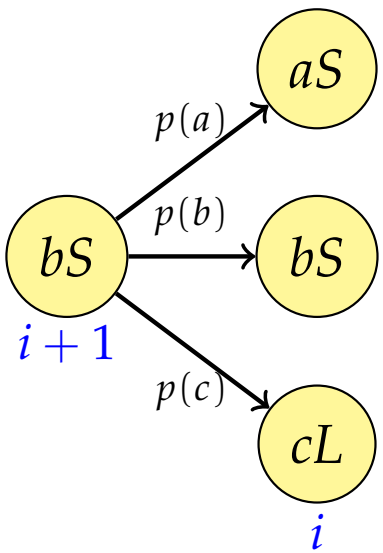
Computing the types

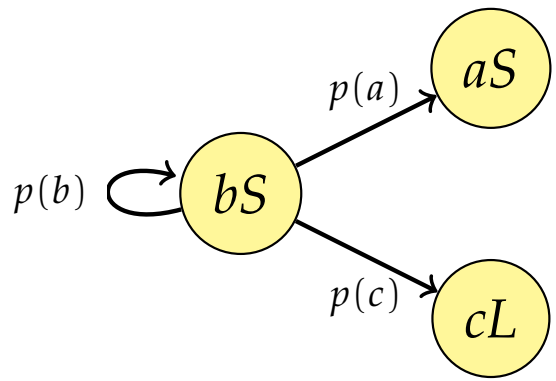
- Computing the types S or $L \leftarrow$ scanning u once from right to left
- Init. $T[n-1] = S$ then for i from $n-2$ down to 1
  - if $u[i] < u[i+1]$ then $T[i] = S$
  - if $u[i] > u[i+1]$ then $T[i] = L$
  - if $u[i] = u[i+1]$ then $T[i] = T[i+1]$

Computing the types

- Computing the types S or $L \leftarrow$ scanning u once from right to left
- Init. $T[n-1] = S$ then for i from $n-2$ down to 1
  - if $u[i] < u[i+1]$ then $T[i] = S$
  - if $u[i] > u[i+1]$ then $T[i] = L$
  - if $u[i] = u[i+1]$ then $T[i] = T[i+1]$

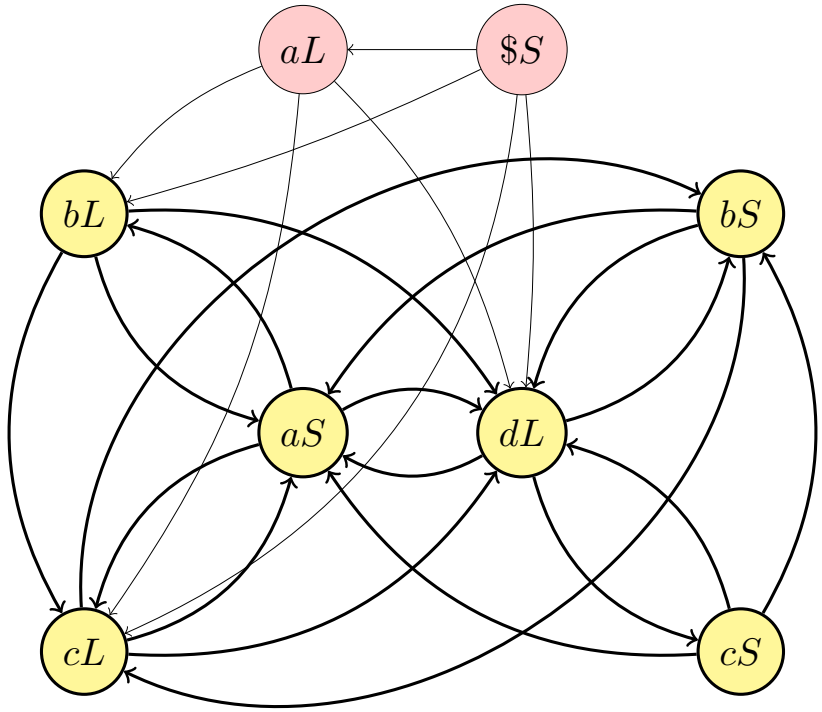
Producing the word backward (from right to left):





A Markov chain problem

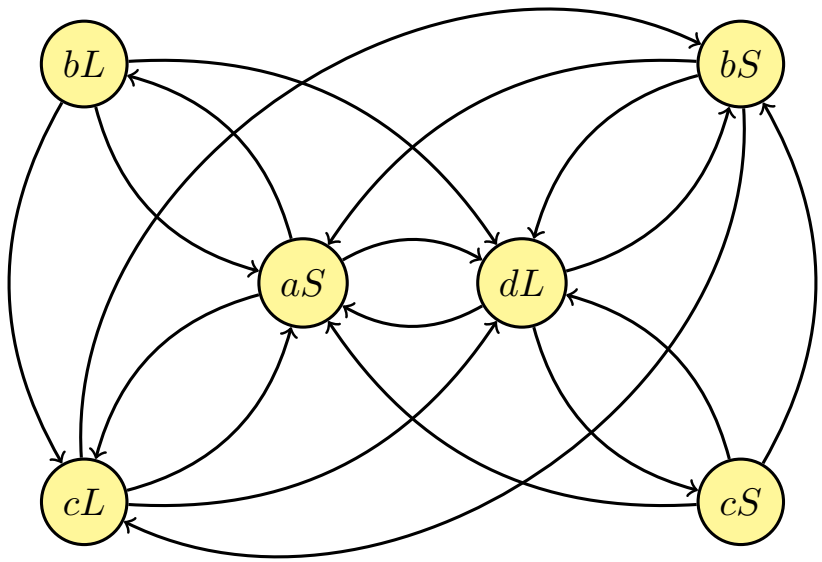
- Generating the letters backward while computing their type is exactly described by a Markov chain on the pairs (letter,type)
- We have an LMS-position when using an edge $xS \to yL$



self-loops missing

A Markov chain problem

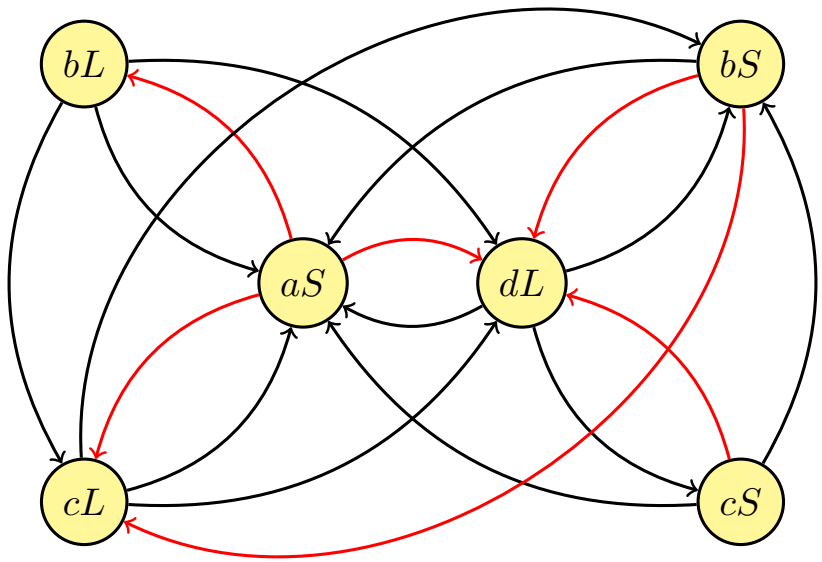
- Generating the letters backward while computing their type is exactly described by a Markov chain on the pairs (letter,type)
- We have an LMS-position when using an edge $xS \to yL$



self-loops missing

A Markov chain problem

- Generating the letters backward while computing their type is exactly described by a Markov chain on the pairs (letter,type)
- We have an LMS-position when using an edge $xS \to yL$



self-loops missing

Classical results on Markov chains

Theorem (stationary vector)

Let M be an irreducible and aperiodic finite Markov chain. There exists a unique stationary vector π, which is the positive probabilistic vector such that $\pi \times M = \pi$.

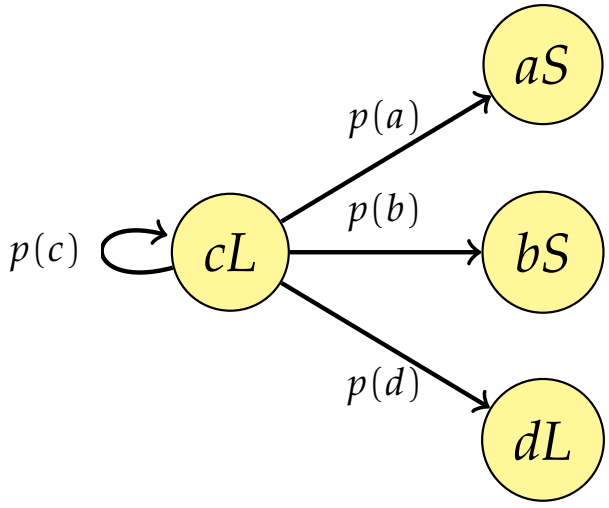
As $n \to \infty$, the probability to be on vertex v after n steps tends to $\pi(v)$ exponentially fast.

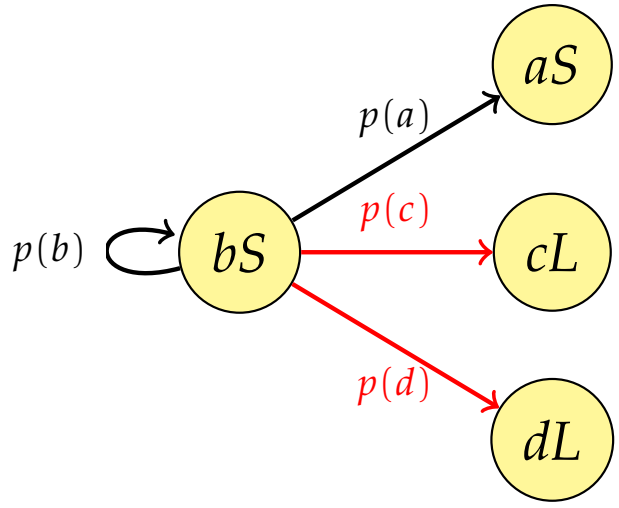
Theorem (informal ergodic theorem)

Let f be a real-valued mapping on the set of vertices V. Let (v_i) be a sequence of vertices met during one trajectory of M, then

$$\frac{1}{n}\sum_{i=1}^{n} f(v_i) \approx \mathbb{E}_\pi[f] = \sum_{v \in V} \pi(v) f(v)$$

Using the ergodic theorem 1/2





$f(cL) = 0$

$f(bS) = p(c) + p(d)$

- Let f be the mapping defined by

$$f(\alpha S) = p_{<\alpha} := \sum_{\beta<\alpha} p(\beta); \qquad f(\alpha L) = 0$$

- Then $\sum_{i=1}^{n} f(v_i)$ "counts" the number of red edges taken

Using the ergodic theorem 2/2

- Then $\sum_{i=1}^{n} f(v_i)$ "counts" the number of red edges taken
- **Ergodic Theorem:** $\frac{1}{n} \sum_{i=1}^{n} f(v_i) \approx \sum_v \pi(v) f(v)$

First formula

The expected number of LMS-factors is asymptotically equivalent to $n \sum_{\alpha \in A} \pi(\alpha S)\, p_{>\alpha}$

- We now need to compute the stationary probability vector π

Computing the stationary vector 1/3

- M viewed as a matrix
- $\pi \times M = \pi$, hence π is a **right eigenvector** of M

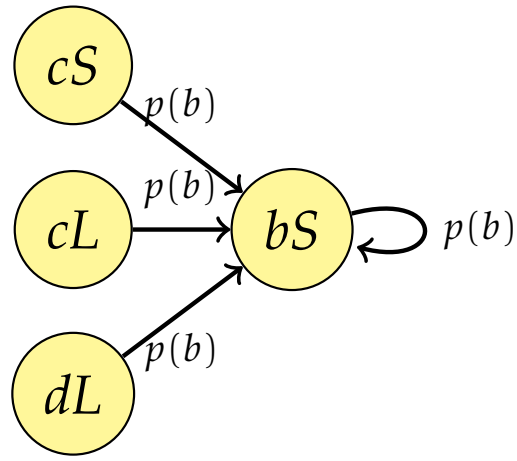
	aS	bS	bL	cS	cL	dL
aS	p_a	0	p_b	0	p_c	p_d
bS	p_a	p_b	0	0	p_c	p_d
bL	p_a	0	p_b	0	p_c	p_d
cS	p_a	p_b	0	p_c	0	p_d
cL	p_a	p_b	0	0	p_c	p_d
dL	p_a	p_b	0	p_c	0	p_d

- We know that π is unique: we can try to guess

Computing the stationary vector 2/3

- $\pi \times M = \pi$: if one pick up a vertex following π then make one step in the Markov chain, then the result is distributed as π
- We must have $\pi(\alpha S) + \pi(\alpha L) = p(\alpha)$

Edges ending in bS:



- We must have:

$$\pi(bS) = p(b)\big(\pi(bS) + \pi(cS) + \pi(cL) + \pi(dL)\big)$$
$$(1 - p(b))\pi(bS) = p(b)\big(p(c) + p(d)\big) = p(b)\, p_{>b}$$

Computing the stationary vector 3/3

Theorem

The stationary vector π is given by

$$\pi(\alpha S) = \frac{p(\alpha)\, p_{>\alpha}}{1 - p(\alpha)} \qquad \text{and} \qquad \pi(\alpha L) = \frac{p(\alpha)\, p_{<\alpha}}{1 - p(\alpha)}$$

- **Proof:** verify that $\sum_v \pi(v) = 1$ and $\pi \times M = \pi$

Result for memoryless sources

Theorem (memoryless sources)

For a memoryless source the **typical** reduction ratio γ is

$$\gamma_p = \sum_{\alpha \in A} \frac{p(\alpha)\, p_{>\alpha}^2}{1 - p(\alpha)}, \qquad p_{>\alpha} = \sum_{\beta > \alpha} p(\beta)$$

Corollary (uniform distribution)

For the uniform distribution on words on an alphabet with k letters,

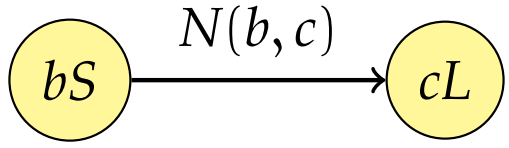
$$\gamma_k = \frac{2k-1}{6k}$$

In particular $\gamma_2 = \frac{1}{4}$ and $\gamma_k \to \frac{1}{3}$ as $k \to \infty$

Markovian sources 1/2

- We assume now that u is generated backward using a Markov chain N of stationary vector (and initial probability vector) τ
- ("backward" is not an issue in our settings)

The probabilities depend on the starting state also:



- Everything works as for memoryless sources, but we have to guess the new stationary vector π

Markovian sources 2/2

Theorem (markovian sources)

For a backward markovian source (N, τ) the **typical** reduction ratio γ is

$$\gamma_N = \sum_{\alpha \in A} \pi(\alpha S) \sum_{\beta > \alpha} N(\alpha, \beta),$$

with

$$\pi(\alpha S) = \frac{\sum_{\beta > \alpha} \tau(\beta) N(\beta, \alpha)}{1 - N(\alpha, \alpha)}$$

Conclusions

File	$\|A\|$	size	γ	uniform	memoryless	Markov
`bible`	63	4047392	**0.3113**	0.3307	0.3230	0.3251
`world192`	93	2408281	**0.2838**	0.3315	0.3256	0.2997
`Chr_22`	4	35033745	**0.2717**	0.2917	0.2928	0.2715

Future directions:

- Analyze the whole algorithm (the sequence of reduction ratios)
- Other Suffix-Array algorithms

Thanks!
Grazie!