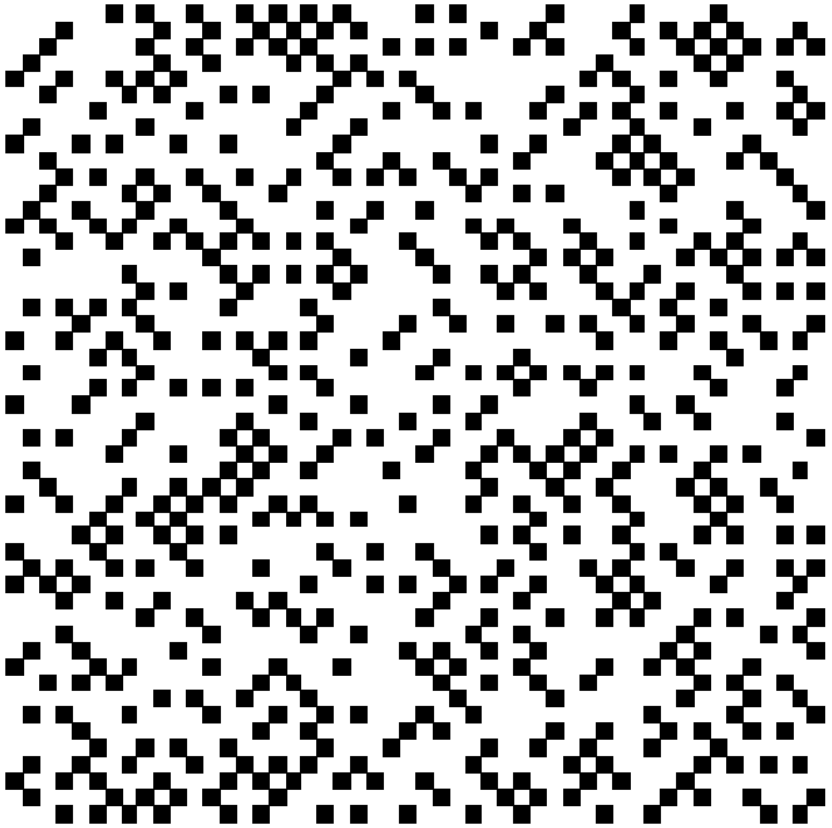
Generalized Fibonacci coding problem using MERW and ANS

for general constraints, multidimensional case

My original 2006 MSc problem:

how to ~optimally store information in 2D lattice $\{0,1\}^{Z^2}$ with no adjacent (↔↕) '1's ?

hard square/**2D Fibonacci** (Markov field)



Maximal entropy random walk (**MERW**)

chosen according to maximal entropy principle

to **find optimal transition probabilities**

now ~200 citations of our 2009 PRL paper

Asymmetric numeral systems (**ANS**)

optimized for any prob. distribution of digits

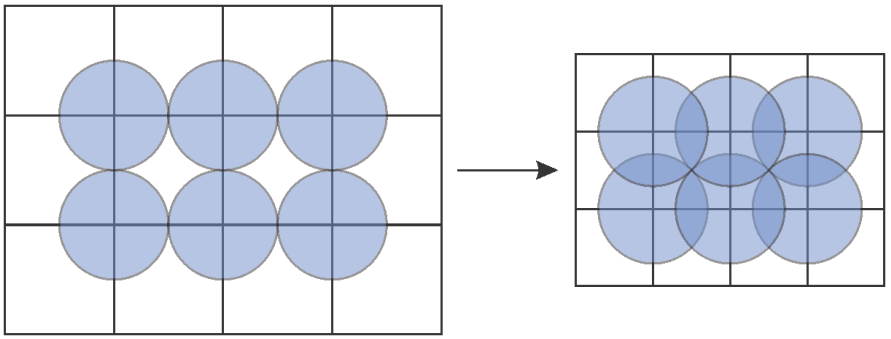
to **encode using these found probabilities**

now widely used in data compressors (links)

replacing Huffman and arithmetic coding

by e.g. Apple, Facebook, Google, Linux, JPEG XL

e.g. to improve capacity of hard disk:

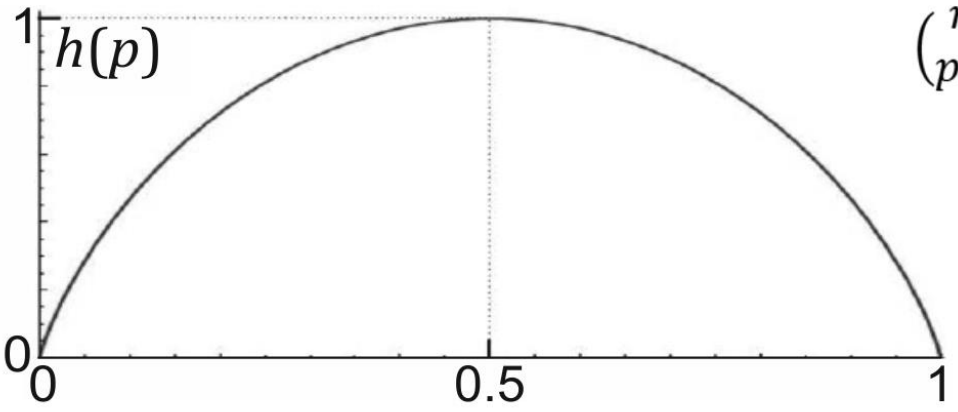


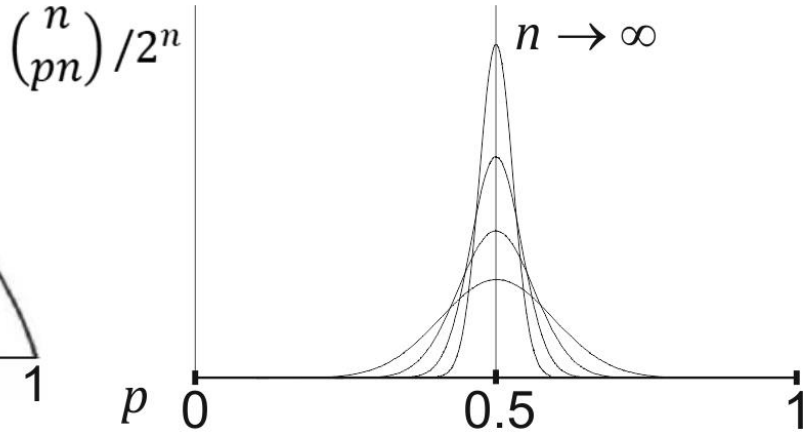
1 bit/node → 2 · 0.58789 ≈ 1.**176**

Jarek Duda, 21.06.2021

We need n bits of information to choose one of 2^n possibilities.

For length n 0/1 sequences with pn of "1", how many bits we need to choose one?





$$\binom{n}{pn} \approx 2^{nh(p)} \quad \text{for} \quad h(p) := -p\lg(p) - (1-p)\lg(1-p)$$

A sequence of symbols with $(p_s)_{s=0..m-1}$ probability distribution contains asymptotically $\boldsymbol{H = \sum_s p_s \lg(1/p_s)}$ **bits/symbol** $(H \leq \lg(m))$

Seen as weighted average:

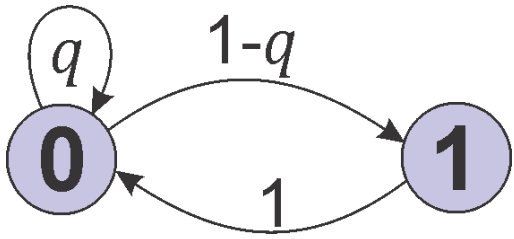
symbol/event of probability p contains $\lg(1/p)$ bits of information

(Jaynes) principle of maximum entropy: while limited knowledge, the best assumption is probability distribution which maximizes entropy.

Fibonacci coding – as a bit sequence with **constraints**: no two neighboring '1's

e.g. 00101010000101010010001 – **each sequence should be equally probable**

What about local statistics: of a single step?



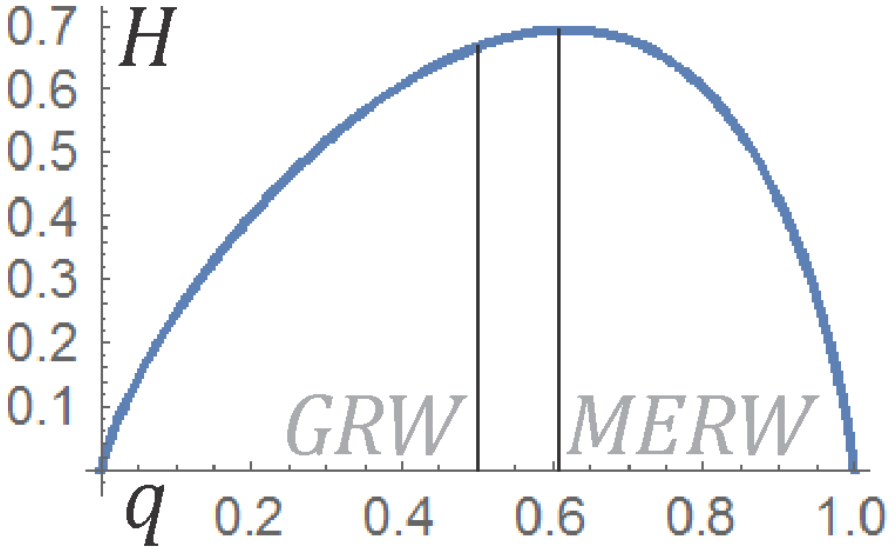
$$M = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \qquad S = \begin{pmatrix} q & 1-q \\ 1 & 0 \end{pmatrix} \qquad q = ?$$

What q should we choose to maximize informational capacity?

Stationary probability: $\pi = (\Pr(0), \Pr(1))^T$

$$\pi S = \pi$$

$$\pi = \left(\frac{1}{2-q}, 1 - \frac{1}{2-q}\right)$$



Entropy – informational content:

$$H = \sum_i \pi_i \sum_j S_{ij} \lg\left(1/S_{ij}\right) = \pi_0 \cdot h(q)$$

$H_{max} \approx 0.694241913$ bits/node ($H = 0.6666\ldots$ for GRW)

for $q = \frac{\sqrt{5}-1}{2} \approx 0.618034$ for MERW ($q = 0.5$ for GRW)

Graph (M)  **stochastic matrix** (S) $\rightarrow$ **stationary probability** (π)

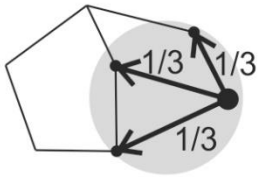
$M_{ab} \in \{0,1\}$ $\quad 0 \leq S_{ab} \leq M_{ab}\ ,\ \forall_a \sum_b S_{ab} = 1$ $\quad \sum_a \pi_a S_{ab} = \pi_b$

Average **entropy** production (rate) per step: $\sum_a \pi_a \sum_b S_{ab} \lg(1/S_{ab})$

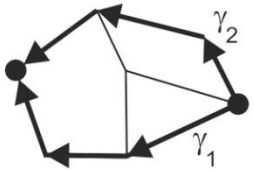
GRW and MERW are equal on regular graphs, but e.g. on **defected 2D lattice**:

MERW – uniform sequence/path ensemble (Anderson localization for electrons)

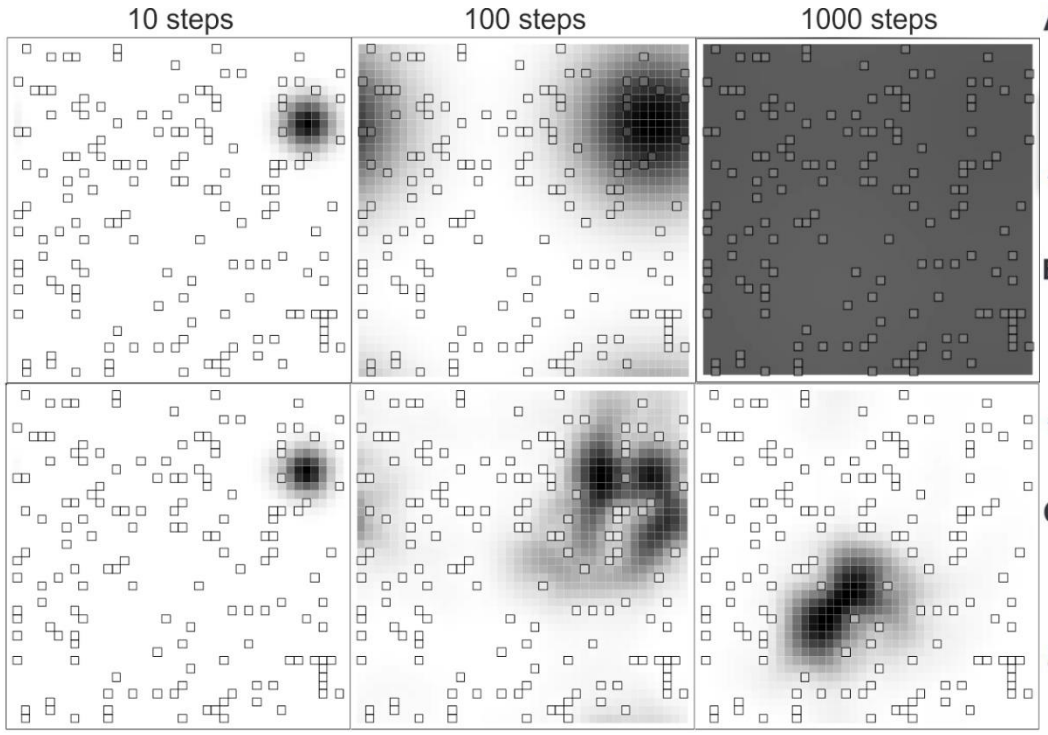
Generic Random Walk (GRW): assume uniform distribution among "the nearest neighbors"

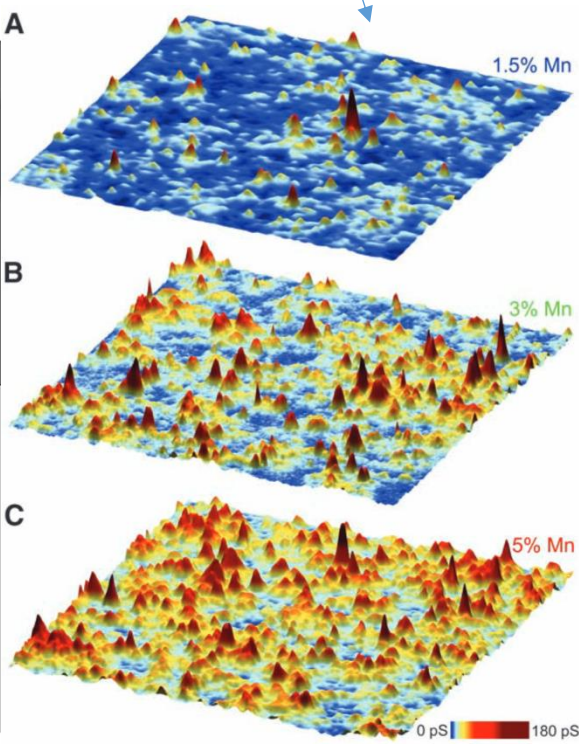


Maximal Entropy Random Walk (MERW): choose that for each two vertices, each path of given length between them is equally probable



$\Pr(\gamma_1)=\Pr(\gamma_2)$





GRW assumes we know exactly the used probabilistic algorithm, has characteristic length

MERW assumes only there are no hidden local probabilistic rules, is scale-free limit of GRW

MERW – uniform probability distribution among **paths**

GRW – all length 1 paths equally probable, **MERW** – all length $t \to \infty$

Assume **unique dominant eigenvalue** (Frobenius-Perron for connected aperiodic)

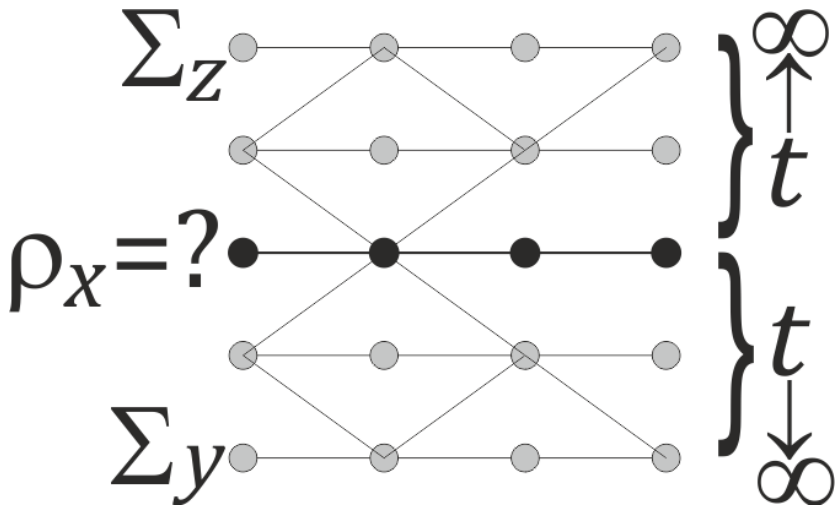
$M\psi = \lambda\psi$ for largest λ

Hubbard: $\mathcal{H} = -\sum_{\text{edge } xy} a_y^+ a_x \equiv -M$

$M^t \approx \lambda^t |\psi\rangle\langle\psi|$ for $t \to \infty$

The number of length $2t$ paths with x in the center:

$$\sum_{yz} (M^t)_{yx}(M^t)_{xz} \approx \lambda^{2t}\psi_x^2$$



$$\boldsymbol{\rho_x \equiv \Pr(x) \propto \psi_x^2} \quad \textbf{– Born rule}$$

Analogously $\Pr(xy) \propto \psi_x M_{xy} \psi_y \Rightarrow \mathbf{S_{xy}} = \frac{\boldsymbol{M_{xy}}}{\boldsymbol{\lambda}} \frac{\boldsymbol{\psi_y}}{\boldsymbol{\psi_x}}$ stochastic matrix

Generally: $\Pr(\gamma_0 \dots \gamma_l) = \frac{1}{\lambda^l} \frac{\psi_{\gamma 0}}{\psi_{\gamma l}}$ $\qquad \left(S^l\right)_{xy} = \frac{(M^l)_{xy}}{\lambda^l} \frac{\psi_y}{\psi_x}$

Gathered formulas:

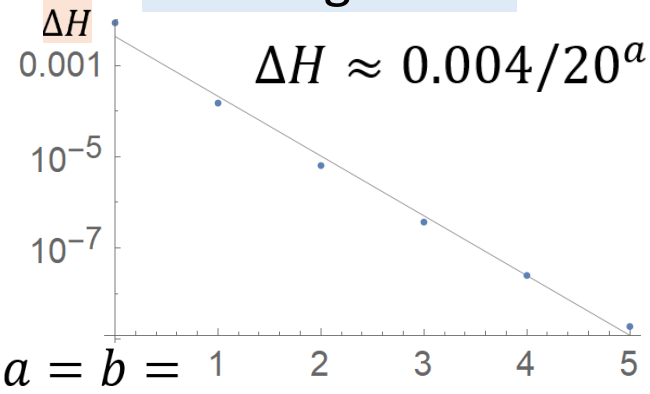
M: adjacency matrix	**GRW**	**MERW**
characteristic length	1	∞
$\boldsymbol{S_{ij}} = \Pr(x_{t+1} = j \mid x_t = i)$	$\frac{M_{ij}}{d_i}$ $(d_i = \sum_j M_{ij})$	$\frac{M_{ij}}{\lambda}\frac{\psi_j}{\psi_i}$ $(M\psi = \lambda\psi)$
$\pi S = \pi$ stat. $\boldsymbol{\pi_i} =$ for symmetric M:	complicated $d_i / \sum_j d_j$	$\varphi_i\psi_i$ $(\varphi M = \lambda\varphi)$ ψ_i^2 (max $\lvert\lambda\rvert$)
$\overline{\boldsymbol{S}}_{ij} = \Pr(x_t = i \mid x_{t+1} = j)$ symmetric M $(\varphi = \psi)$:	$\pi_i M_{ij}/\pi_j$ complicated M_{ij}/d_j	$\frac{M_{ij}}{\lambda}\frac{\varphi_i}{\varphi_j}$
$\boldsymbol{S_{\gamma_0\gamma_1}S_{\gamma_1\gamma_2}\dots S_{\gamma_{l-1}\gamma_l}} =$ for path γ	$\frac{M_{\gamma_0\gamma_1}M_{\gamma_1\gamma_2}\dots M_{\gamma_{l-1}\gamma_l}}{d_{\gamma_0}d_{\gamma_1}\dots d_{\gamma_{l-1}}}$	$\frac{M_{\gamma_0\gamma_1}M_{\gamma_1\gamma_2}\dots M_{\gamma_{l-1}\gamma_l}}{\lambda^l}\frac{\psi_{\gamma_l}}{\psi_{\gamma_0}}$
$\left(\boldsymbol{S^l}\right)_{\boldsymbol{ij}} =$	complicated	$\frac{(M^l)_{ij}}{\lambda^l}\frac{\psi_j}{\psi_i}$
scaling, generally	$S^{GRW(M^l)} \neq \left(S^{\mathrm{GRW}(M)}\right)^l$	$S^{MERW(M^l)} = \left(S^{\mathrm{MERW}(M)}\right)^l$
$\boldsymbol{H(S)} = -\sum_i \pi_i \sum_j S_{ij} \lg S_{ij}$	$\sum_i \frac{d_i \lg(d_i)}{\sum_j d_j}$ $(M = M^T)$	$\lg(\lambda)$

Being able to calculate MERW for any adjacency matrix M, approximate 2D Fibonacci with $w \times \infty$ lattice, find conditional distributions for scanning model

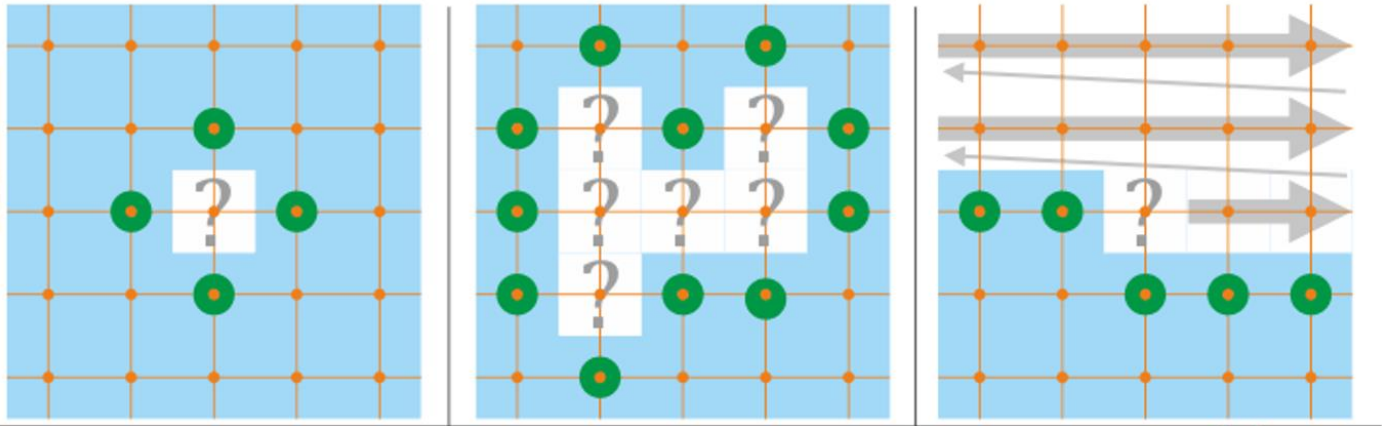
Choose $\Pr(?|\bullet)$ based on $a = b$ neighboring values

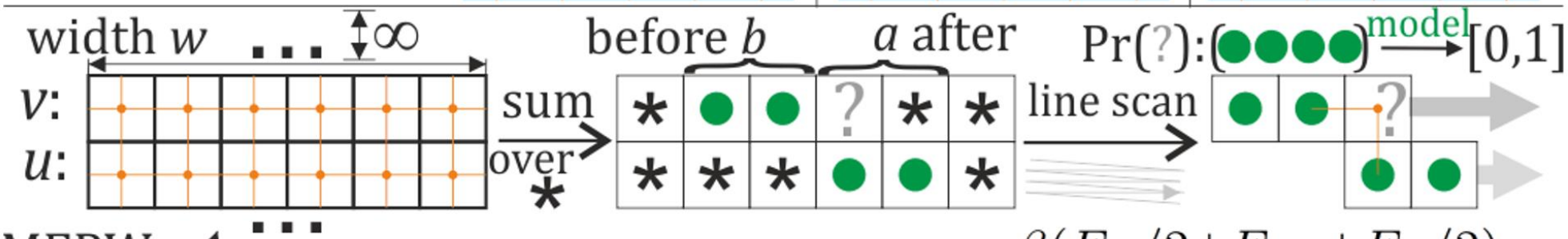
$\Delta H \approx 0.01 \ldots 10^{-9}$ below optimal ≈0.587891 bits/node

(originally to be encoded with first ANS: now uABS variant)



Hammersley-Clifford theorem: **Gibbs field** (Boltzmann distribution) is equivalent with **Markov field**: $\Pr(?|\blacksquare) = \Pr(?|\bullet)$ local, global, scanning:





MERW: $\Pr(uv) = \psi_u \frac{M_{uv}}{\lambda} \psi_v$

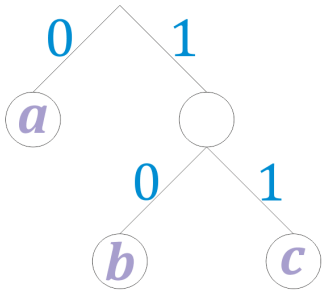
$M_{uv} = e^{-\beta(E_u/2 + E_{uv} + E_v/2)}$

$M\psi = \lambda\psi \quad (\max_{|\lambda|},\ \|\psi\|_2 = 1)$

i.i.d. Pr(0) = Pr(1) complex distr.

Asymmetric Numeral Systems 010001 ↔ abaaabaaa

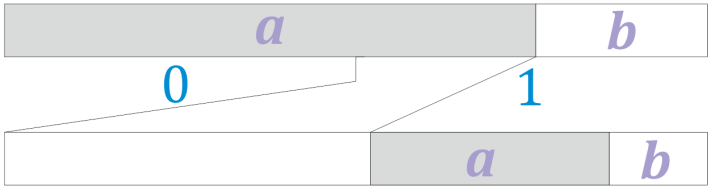
Past: compromise



(1952)

(prefix,) **Huffman coding**
(also unary, Golomb, Elias, etc.)
fast (>300MB/s/core)
no multiplication, needs sorting
but **inaccurate:** Pr(*s*) ~ 2^{-r}
e.g. for Pr(*a*)=0.01, Pr(*b*)=0.99
uses **1** bit/symbol

or ?



(~1976)

arithemtic/range coding
slow (<< 100MB/s/core)
uses multiplication
uses nearly **accurate** Pr(*s*)
e.g. for Pr(*a*)=0.01, Pr(*b*)=0.99
uses ~**0.08** bits/symbol

Now: ANS

(2007)

tANS: **tabled** - no multiplication
"Huffman generalized to fractional bits"
also allows for simultaneus encryption

(2013)

rANS: **range** - direct replacement
of arithmetic/range coding: with
smaller state, less multiplications

mainly used for smaller models, fixed distributions

fast (> 500MB/s/core)
uses nearly **accurate** Pr(*s*)
e.g. for Pr(*a*)=0.01, Pr(*b*)=0.99
uses ~**0.08** bits/symbol

mainly used for larger models, adaptive distributions

e.g. in general compressors (Apple LZFSE, Fb ZSTD), DNA (CRAM), Google Draco ... JPEG XL

10GB large **text** benchmark (2020, i9 9900K), 1GB wiki for 10 languages (ANS):

(combining models for various languages)

10GB -> Size	encoding time	decoding time	
5,034,758,325 bytes,	18.449 sec. -	7.311 sec.,	lz4 -1 (v1.9.2)
4,666,386,317 bytes,	26.686 sec. -	4.827 sec.,	lzturbo -10 -p0 (v1.2)
4,371,496,854 bytes,	46.907 sec. -	7.282 sec.,	lz4x -1 (v1.60)
3,909,521,247 bytes,	32.603 sec. -	11.287 sec.,	lizard -40 (v1.0.0)
3,823,273,187 bytes,	136.146 sec. -	59.070 sec.,	gzip -1 (v1.3.12) "**.zip**"
3,770,151,519 bytes,	34.216 sec. -	26.236 sec.,	brotli -q 0 (v1.0.7)
3,642,089,943 bytes,	28.752 sec. -	10.717 sec.,	**zstd** -1 (v1.4.5) LZ + **tANS**/huf
3,660,882,443 bytes,	767.399 sec. -	7.633 sec.,	lz4x -9 (v1.60)
3,237,812,198 bytes,	392.835 sec. -	53.771 sec.,	gzip -9 (v1.3.12) "**.zip**"
3,095,248,795 bytes,	137.881 sec. -	20.738 sec.,	brotli -q 4 (v1.0.7)
3,078,914,611 bytes,	240.124 sec. -	9.381 sec.,	**zhuff** -c2 -t1 (v0.99beta), LZ4 + **tANS**
3,065,081,662 bytes,	50.724 sec. -	12.904 sec.,	**zstd** -4 --ultra --single-thread (v1.4.5)
2,660,370,879 bytes,	153.103 sec. -	19.993 sec.,	**lzturbo** -32 -p0 (v1.2), LZ + **tANS**
2,639,230,515 bytes,	561.791 sec. -	11.774 sec.,	**zstd** -12 --ultra --single-thread(v1.4.5)
2,357,818,671 bytes,	3,953.092 sec. -	34.300 sec.,	rar -m5 -ma5 -mt1 (v5.80)
2,337,506,087 bytes,	2,411.038 sec. -	11.971 sec.,	**zstd** -18 --ultra --single-thread(v1.4.5)
2,220,027,943 bytes,	7,439.064 sec. -	22.690 sec.,	brotli -q 10 (v1.0.7)
2,080,479,075 bytes,	4,568.550 sec. -	12.934 sec.,	**zstd** -22 --ultra --single-thread(v1.4.5)
2,059,053,547 bytes,	4,909.124 sec. -	55.188 sec.,	7-Zip -t7z -mx9 -mmt1 (v19.02) - LZMA
1,973,568,508 bytes,	6,626.946 sec. -	89.762 sec.,	arc -m9 -mt1 (v0.67)
1,921,561,064 bytes,	17,200.759 sec. -	27.147 sec.,	brotli -q 11 --large_window=30 (v1.0.7)
1,899,403,918 bytes,	1,327.809 sec. -	375.295 sec.,	nz -cO -t1 (v0.09 alpha)
1,722,407,658 bytes,	778.796 sec. -	401.317 sec.,	m99 -b1000000000 -t1 (beta)
1,675,874,699 bytes,	781.839 sec. -	198.309 sec.,	bwtturbo -59 -t0 (v20.2)
1,644,097,084 bytes,	21,097.196 sec. -	93.130 sec.,	**razor** (v1.03.7) - **adaptive 4bit rANS**
1,638,441,156 bytes,	1,030.489 sec. -	640.502 sec.,	bsc -m0 -b1024 -e2 -T (v3.1.0)
1,632,628,624 bytes,	1,146.133 sec. -	1,284.451 sec.,	bcm -9 (v1.40)
1,450,364,034 bytes,	2,701.335 sec. -	2,433.988 sec.,	mcm -x -m11 (v0.83)

Operating on **fractional number of bits**

The basic ANS concept: store information as a natural number x

Assume **number x contains $\lg(x)$ bits of information**

($\equiv \Pr(x) \sim 1/x$ Zipf law)

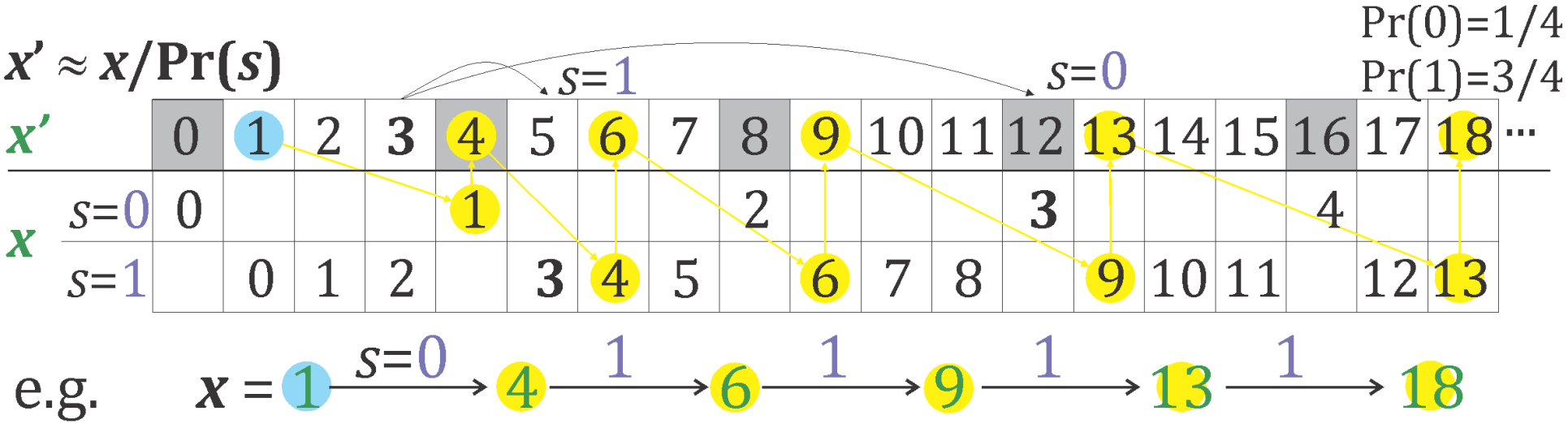
We know **symbol of probability p contains $\lg(1/p)$ bits**

So **new number x' contains**: $\lg(x') \approx \lg(x) + \lg(1/p)$ bits

In other words: $x' \approx x/p$

Like in standard binary system $x' = 2x + s \approx x/0.5$

optimal for $\Pr(s=0) = \Pr(s=1) = 0.5$



ANS: $x \to \approx x/\Pr(s)$ while encoding symbol s

Redefine even/odd subsets according to densities

$x \to$ $x-$th appearance of $'\text{even}'(s=0)$ or $'\text{odd}'$ $(s=1)$

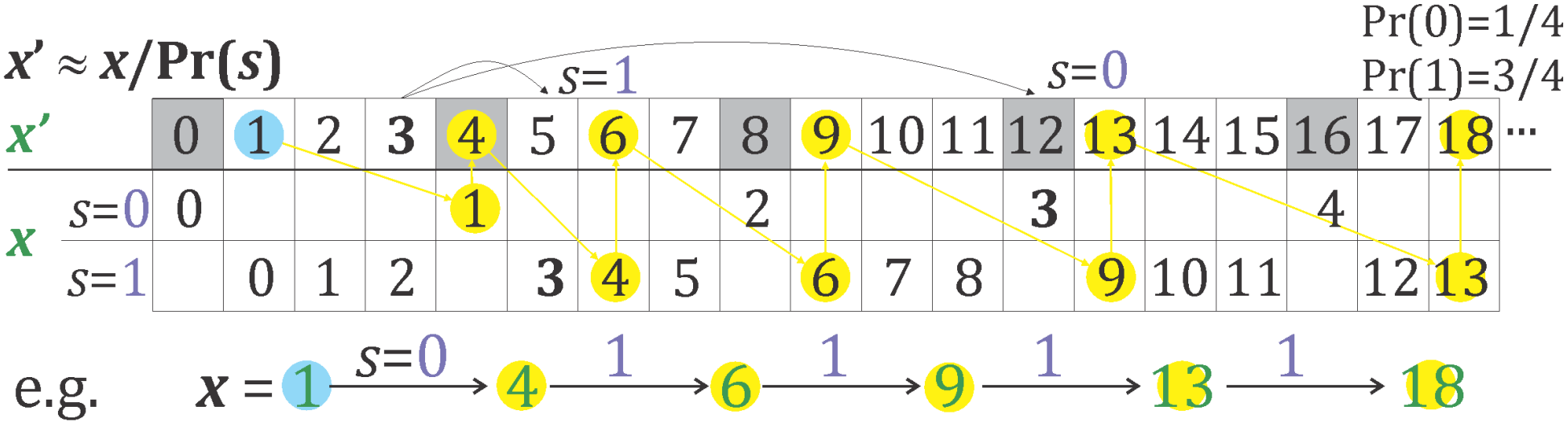
rANS variants: repeating division in ranges

$\bar{s}(x) = 0$ if $\text{mod}(x,4) = 0$, else $\bar{s}(x) = 1$

to decode or encode 1, localize quadruple ($\lfloor x/4 \rfloor$ or $\lfloor x/3 \rfloor$)

if $\bar{s}(x) = 0$, $D(x) = (0, \lfloor x/4 \rfloor)$ else $D(x) = (1, 3\lfloor x/4 \rfloor + \text{mod}(x,4) - 1)$

$C(0,x) = 4x$ $\quad$ $C(1,x) = 4\lfloor x/3 \rfloor + 1 + \text{mod}(x,3)$



rANS - range variant for large alphabet $\mathcal{A} = \{0, .., m-1\}$

assume $\Pr(s) = f_s/2^n$ $\quad\quad$ $c_s := f_0 + f_1 + \cdots + f_{s-1}$

start with base 2^n numeral system and merge length f_s ranges

for $x \in \{0,1, \dots, 2^n - 1\}$, $\bar{s}(x) = \mathbf{max}\{s: c_s \leq x\}$, mask $= 2^n - 1$

encoding: $C(s,x) = \lfloor x/f_s \rfloor \ll n + \mathrm{mod}(x, f_s) + c_s$

decoding: $s = \bar{s}(x \,\&\, \mathrm{mask})$ (e.g. tabled, alias method)

$$D(x) = (s,\ f_s \cdot (x \gg n) + (\, x \,\&\, \mathrm{mask}) - c_s)$$

Plus renormalization to make for example $x \in \{2^{16}, \dots, 2^{32} - 1\}$, $n = 12$:

Decoding step $(mask = 2^n - 1)$	**Encoding step** $(msk = 2^{16} - 1,\ d = 32\text{-}n)$
s = symbol[x & $mask$]	s = readSymbol();
writeSymbol(s);	if($x \geq$ (f[s]<< d))
x = f[s] (x >> n) + (x & $mask$) – c[s];	{write16bits(x & msk); x >>= 16; }
if($x < 2^{16}$) $x = x$ << 16 + read16bits();	x = [x / f[s]] << n + (x % f[s]) + c[s];

CRAM v3 (2015): order 1 rANS: 256 frequencies depending on the previous symbol

RENORMALIZATION to prevent $x \to \infty$

Enforce $x \in I = \{L, \dots, 2L-1\}$ by

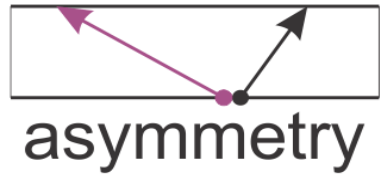
transmitting lowest bits to bitstream

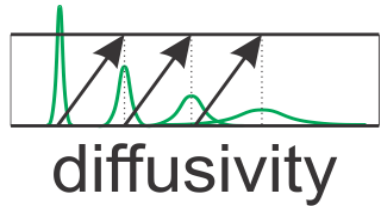
"buffer" x contains $\lg(x)$ bits of information

– produces bits when **accumulated**

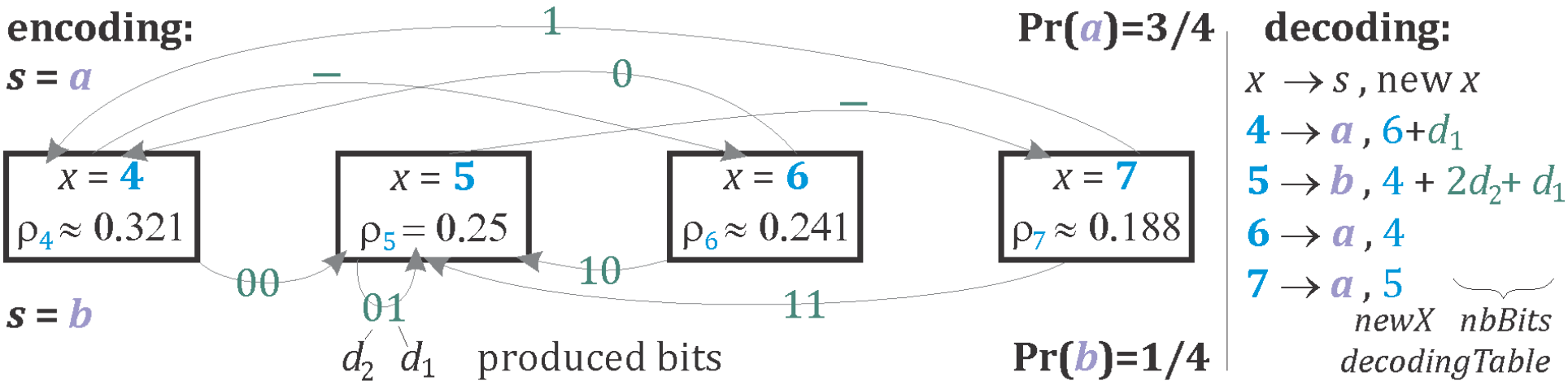
Symbol of $Pr = p$ contains $\lg(1/p)$ bits:

$\lg(x) \to \approx \lg(x) + \lg(1/p)$ modulo 1

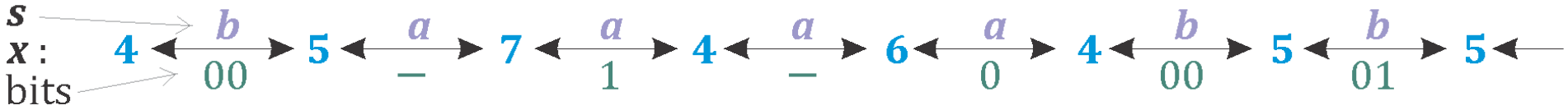




Tabled variant (tABS/tANS) – put everything into a table:



example: encoding → ← decoding



example of tANS construction for L=16 states and size 3 alphabet

t=decodingTable[x]; use(t.symbol); $x \to$ **t.newX + readBits(t.nbBits);**

1. Approximate probabilities

as $p_s \approx L_s / L$

2. Spread symbols: L_s of symbol s

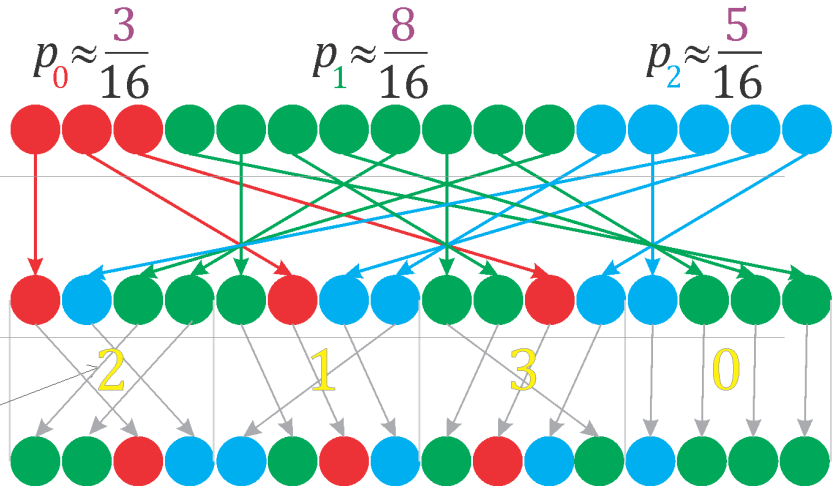
(fast, step = 5)

2*. Scramble

(4 block cycle)

key

secure PRNG



3. Enumerate appearances

from L_s to $2L_s$ - 1

L =16, L_0= 3, L_1= 8, L_2= 5

$C(s,x)$		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
	s=0			3				4			5						
x	s=1	8	9				10			11			12		13	14	15
	s=2				5	6			7			8		9			

4. Renormalize to make x remain in $I = \{L, ..., 2L\text{-}1\}$ range

decodingTable:

(**symbol**,

nbBits,

newX)

x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
symbol	1	1	0	2	2	1	0	2	1	0	2	1	2	1	1	1
pre-renormalization x_{tmp}	8	9	3	5	6	10	4	7	11	5	8	12	9	13	14	15
nbBits to read to return to I	1	1	3	2	2	1	2	2	1	2	1	1	1	1	1	1
newX = x_{tmp} << nbBits	16	18	24	20	24	20	16	28	22	20	16	24	18	26	28	30

5. Endode/decode - e.g. decoding 11|10|00|011|0|10|10|011

bits → 11 → 10 → 0 → 0 → 011 → 0 → 1 → 01 → 0 → 0 → 1 → 1

x: 25 → 23 → 30 → 28 → 18 → 27 → 24 → 23 → 29 → 26 → 16 → 17 → 19

s: 0 → 2 → 1 → 2 → 0 → 1 → 1 → 2 → 1 → 2 → 1 → 1

Which **symbol spread** should we choose? (link)

(using PRNG seeded with cryptkey for encryption)

for example: **ratio loss** for $p = (0.04,\ 0.16,\ 0.16,\ 0.64)$

$L = 16$ states, $q = (1, 3, 2, 10)$, $q/L = (0.0625, 0.1875, 0.125, 0.625)$

method	symbol spread	ΔH/H ratio loss	
-	-	**~0.011**	penalty of **quantizer** itself
Huffman	 **0011222233333333**	**~0.080**	would give **Huffman** decoder
spread_range_i()	**0111223333333333**	**~0.059**	**Increasing** order
spread_range_d()	**3333333333221110**	**~0.022**	**Decreasing** order
spread_fast()	**0233233133133133**	**~0.020**	**fast**, in FSE/Zstd
spread_prec()	**3313233103332133**	**~0.015**	**close to quantization** ΔH/H
spread_**tuned**()	**3233321333313310**	**~0.0046**	**better than quantization** ΔH/H due to also using p
spread_**tuned**_s()	**2333312333313310**	**~0.0040**	L log L complexity (sort)
spread_**tuned**_p()	**2331233330133331**	**~0.0058**	testing $1/(p \ln(1+1/i)) \approx i/p$ approximation

tANS (2007) - fully tabled: Apple LZFSE, Facebook ZSTD, lzturbo

fast: no multiplication (**FPGA**!), less memory efficient (~8kB for 2048 states)

static in ~32kB blocks, costly to update (rather needs rebuilding),

allows for simultaneous encryption (CSPRNG to perturb symbol spread)

tANS decoding step	**Encoding step** (for symbol s)

t = decodingTable[x];
writeSymbol(t.symbol);
x = t.newX + readBits(t.nbBits);
	nbBits = (x + nb[s]) >> r ;
	writeBits(x, nbBits);
	x = encodingTable[start[s] + (x >> nbBits)];

rANS (2013) – needs multiplication – CRAM (DNA), RAZOR, BB-ANS, JPEG XL

more memory effective – especially for large alphabet and precision (CDF only)

better for adaptation ($CDF[s] \le y < CDF[s+1]$ - tabled, alias, binary search/SIMD)

rANS decoding step (mask = $2^n - 1$)	**Encoding step** (s) ($msk = 2^{16} - 1$, d=32-n)
s = **symbol**(x & $mask$); writeSymbol(s); x = f[s] (x >> n) + (x & $mask$) – c[s]; if($x < 2^{16}$) $x = x$ << 16 + read16bits();	if($x \ge (f[s] \ll d)$) {write16bits(x & msk); x >>= 16; } $x = \lfloor x / f[s] \rfloor \ll n + (x \,\%\, f[s]) + c[s]$;