On Universal Codes for Integers

Wallace Tree, Elias Omega and Beyond

Lloyd Allison, Arun S. Konagurthu, Daniel F. Schmidt

Faculty of Information Technology, Monash University, Melbourne, ...

DCC 2021

... Australia

Motivation

- Officially:

  compress integers

  inductive inference $\{H_1, H_2, ...\}, \ \Pr(H_i) = \Pr(i) = 2^{-|\mathsf{CW}(i)|}$

  (fun)

Motivation

- Officially:

  compress integers

  inductive inference $\{H_1, H_2, ...\}, \ \Pr(H_i) = \Pr(i) = 2^{-|\mathsf{CW}(i)|}$

  (fun)

- Actually:

  fun

  inductive inference

  compress integers

Motivation

- Officially:

  compress integers

  inductive inference $\{H_1, H_2, ...\}, \ \Pr(H_i) = \Pr(i) = 2^{-|\mathsf{CW}(i)|}$

  (fun)

- Actually:

  fun

  inductive inference*

  compress integers

- * (i) would like $\Pr(i)$ and $\Pr(i+1)$ to be similar,

Motivation

- Officially:

  compress integers

  inductive inference $\{H_1, H_2, ...\},\ \Pr(H_i) = \Pr(i) = 2^{-|\mathsf{CW}(i)|}$

  (fun)

- Actually:

  fun

  inductive inference*

  compress integers

- * (i) would like $\Pr(i)$ and $\Pr(i+1)$ to be similar,
  (ii) sometimes want $\Pr(n) \sim 1/n$ (and pdf$(x) \sim 1/x$), giving $|\mathsf{CW}(n)| \sim \log(n)$; impossible of course.

... a fancy

$\int_1^\infty \frac{1}{x}, \quad \int_2^\infty \frac{1}{x \log x}, \quad \ldots,$ are infinite

but ...

... a fancy

$$\int_1^\infty \frac{1}{x}, \quad \int_2^\infty \frac{1}{x \log x}, \quad ..., \quad \text{are infinite}$$

but ...

$$\int_1^\infty \frac{1}{x^{1+\delta}}, \quad \int_2^\infty \frac{1}{x(\log x)^{1+\delta}}, \quad ..., \quad \text{are finite,}$$

... a fancy

$\int\limits_1^\infty \frac{1}{x}, \quad \int\limits_2^\infty \frac{1}{x \log x}, \quad \ldots,$ are infinite

but ...

$\int\limits_1^\infty \frac{1}{x^{1+\delta}}, \quad \int\limits_2^\infty \frac{1}{x(\log x)^{1+\delta}}, \quad \ldots,$ are finite,

and $\log(x(\log x)^{1+\delta})) = \log x + (1+\delta) \log \log x$

Plan

Some Universal Codes for Integers:

omega (i.e., Elias ω)

omega$_p(s)$

  omega^2 = omega$_p$(omega)

omega$_r(t)$

  omega* = omega$_r$(omega)

? Is there an ultimate code ?

WTC (Wallace's tree-based code)

Comparisons

Elias omega (ω) (1975)

n=1	"1"
2	"010"
3	"011"
4	"00100"
...	...

Elias omega (ω) (1975)

n=1	"1"
2	"010"
3	"011"
4	"00100"
...	...

- e.g., n = 36

Elias omega (ω) (1975)

n=1	"1"
2	"010"
3	"011"
4	"00100"
...	...

- e.g., n = 36

$$1\ 0\ 0\ 1\ 0\ 0 \quad = 36, \quad \text{len}=6$$

Elias omega (ω) (1975)

n=1	"1"
2	"010"
3	"011"
4	"00100"
...	...

- e.g., n = 36

$$\begin{array}{lll} \quad\quad 1\ 0\ 0\ 1\ 0\ 0 & = 36, & \text{len}=6 \\ 1\ 0\ 1 & = 5, & \text{len}=3 \end{array}$$

Elias omega (ω) (1975)

n=1	"1"
2	"010"
3	"011"
4	"00100"
...	...

- e.g., n = 36

$$
\begin{array}{llll}
 & & 1\ 0\ 0\ 1\ 0\ 0 & = 36, \quad \text{len}=6 \\
 & 1\ 0\ 1 & & = 5, \quad \text{len}=3 \\
1\ 0 & & & = 2, \quad \text{len}=2
\end{array}
$$

Elias omega (ω) (1975)

n	
n=1	"1"
2	"010"
3	"011"
4	"00100"
...	...

- e.g., n = 36

1 0 0 1 0 0	= 36,	len=6
1 0 1	= 5,	len=3
1 0	= 2,	len=2
1	= 1,	len=1

Elias omega (ω) (1975)

n	code
n=1	"1"
2	"010"
3	"011"
4	"00100"
...	...

- e.g., n = 36

1 0 0 1 0 0	= 36,	len=6
1 0 1	= 5,	len=3
1 0	= 2,	len=2
1	= 1,	len=1

- 1 10 101 100100 use leading bits for #sections...

Elias omega (ω) (1975)

n	code
n=1	"1"
2	"010"
3	"011"
4	"00100"
...	...

- e.g., n = 36

$$
\begin{array}{llll}
 & 1\,0\,0\,1\,0\,0 & = 36, & \text{len}=6 \\
 & 1\,0\,1 & = 5, & \text{len}=3 \\
 & 1\,0 & = 2, & \text{len}=2 \\
 & 1 & = 1, & \text{len}=1
\end{array}
$$

- 1 10 101 100100 use leading bits for #sections...

  $\underline{0}$ $\underline{0}0$ $\underline{0}01$ $\underline{1}00100$ note, $\underline{0001}$ is unary code for 4

"000001100100"

$\log^*$

$\log^*(n) = c + \log(n) + \log^2(n) + ...$ positive terms,

advocated by J. Rissanen (1983) for use in inference,

$\log^*$

$\log^*(n) = c + \log(n) + \log^2(n) + ...$ positive terms,

advocated by J. Rissanen (1983) for use in inference,
can be seen as an approximation to omega.

omega has a big step when a new section kicks in, e.g.,
15 $\rightarrow$ 0 01 1111 (7 bits); 16 $\rightarrow$ 0 00 000 10000 (11 bits).

omega$_p$

We can use *any* code, s, for #sections – Fibonacci, WTC, etc.

omega$_p$

We can use *any* code, s, for #sections – Fibonacci, WTC, etc.

define omega$_p(s)$ i.e. omega parameterised,
and omega^2 = omega$_p$(omega).

omega$_p$

We can use *any* code, s, for $\#$sections – Fibonacci, WTC, etc.

define omega$_p(s)$ i.e. omega parameterised,
and omega$^2 =$ omega$_p$(omega).

Even a "recursive" version, omega$_r(t)$ that uses itself for $\#$sections ...

$\text{omega}_r(t)$, omega* ...

```
function omega_r_enc(t)
 { function enc(n)
    { var todo = n, CW = "", ...;
      for( nTet = 1; ; nTet ++ )
       { omega's logic but trim off the
         leading bit of each section;

         if( nSections == 1 ) break;
         todo = nSections - 1;      // !
       }//nTet
      return t(nTet) ++ CW;         // !
    }//enc
   return enc;
 }//omega_r_enc

function omega_star_enc(n) = omega_r_enc(omega_enc)(n)
```

... omega*

e.g., n=36

trim(omega(36)) ~~0~~ ~~0~~0 ~~0~~01 ~~1~~00100

... omega*

e.g., n=36

trim(omega(36)) $\not{0}\ \not{0}0\ \not{0}01\ \not{1}00100$

trim(omega(s−1=3)) $\not{0}\ \not{1}1$

... omega*

e.g., n=36

trim(omega(36)) $\not 0\ \not 0 0\ \not 0 01\ \not 1 00100$

trim(omega(s−1=3)) $\not 0\ \not 1 1$

trim(omega(s−1=1)) $\not 1$

... omega*

e.g., n=36

trim(omega(36)) $\not{0}\ \not{0}0\ \not{0}01\ \not{1}00100$
trim(omega(s-1=3)) $\not{0}\ \not{1}1$
trim(omega(s-1=1)) $\not{1}$
3 *tetrations*

... omega*

e.g., n=36

trim(omega(36))		$\not{0}$ $\not{0}0$ $\not{0}01$ $\not{1}00100$
trim(omega(s−1=3))		$\not{0}$ $\not{1}1$
trim(omega(s−1=1))		$\not{1}$
3 *tetrations*		
omega(#t=3)	0 11	

... omega*

e.g., n=36

trim(omega(36))				$\not{0}\ \not{0}0\ \not{0}01\ \not{1}00100$
trim(omega(s−1=3))			$\not{0}\ \not{1}1$	
trim(omega(s−1=1))		$\not{1}$		
3 *tetrations*				
omega(#t=3)	0 11			
	0 11	_	1	00100100

"011100100100"

Continuous Improvement ...

- Can there be an "ultimate" code **C**, $\Pr(n) = 2^{-|\mathsf{CW}(n)|}$?

Continuous Improvement ...

- Can there be an "ultimate" code **C**, $\Pr(n) = 2^{-|\mathsf{CW}(n)|}$?
- Define a new code **C'**, $\Pr'(n) = 2^{-|\mathsf{CW}'(n)|}$:–
  $\mathsf{CW}'(1) = \mathsf{CW}(1) ++ "0"$
  $\mathsf{CW}'(2) = \mathsf{CW}(1) ++ "1"$

Continuous Improvement ...

- Can there be an "ultimate" code **C**, $\Pr(n) = 2^{-|\mathsf{CW}(n)|}$?
- Define a new code **C'**, $\Pr'(n) = 2^{-|\mathsf{CW}'(n)|}$:–
  $\mathsf{CW}'(1) = \mathsf{CW}(1) + +"0"$
  $\mathsf{CW}'(2) = \mathsf{CW}(1) + +"1"$
  $\mathsf{CW}'(3) = \mathsf{CW}(2) + +"0"$
  $\mathsf{CW}'(4) = \mathsf{CW}(2) + +"1"$

Continuous Improvement ...

- Can there be an "ultimate" code **C**, $\Pr(n) = 2^{-|\mathsf{CW}(n)|}$?
- Define a new code **C'**, $\Pr'(n) = 2^{-|\mathsf{CW}'(n)|}$:–
  $\mathsf{CW}'(1) = \mathsf{CW}(1) ++ "0"$
  $\mathsf{CW}'(2) = \mathsf{CW}(1) ++ "1"$
  $\mathsf{CW}'(3) = \mathsf{CW}(2) ++ "0"$
  $\mathsf{CW}'(4) = \mathsf{CW}(2) ++ "1"$
  ...
  $\mathsf{CW}'(2n-1) = \mathsf{CW}(n) ++ "0"$
  $\mathsf{CW}'(2n) = \mathsf{CW}(n) ++ "1"$
  ...

… continuous improvement …

- If **C** is a prefix code then so is **C'**

... continuous improvement ...

- If **C** is a prefix code then so is **C'**
- **C'** is a proper code:

  $\Pr'(2n-1) = \Pr'(2n) = \Pr(n)/2$,

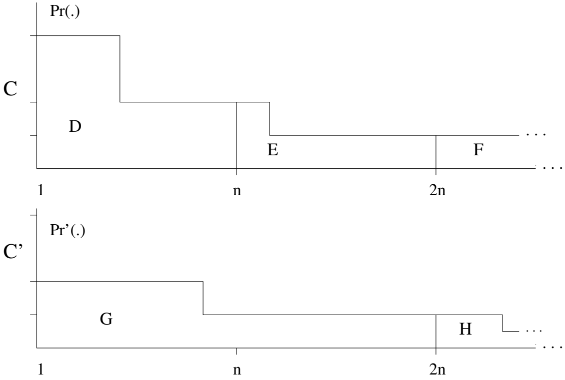
... continuous improvement ...

- If **C** is a prefix code then so is **C'**
- **C'** is a proper code:

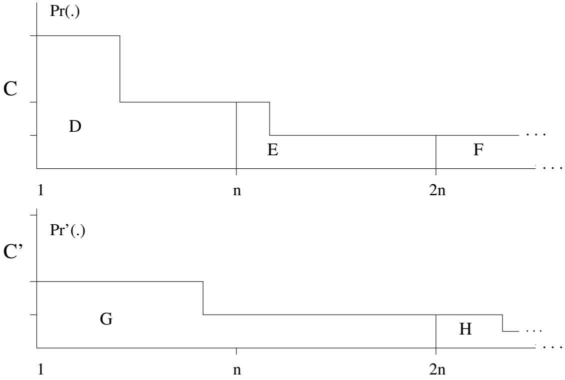
  $\mathrm{Pr}'(2n-1) = \mathrm{Pr}'(2n) = \mathrm{Pr}(n)/2,$

  $\sum_{n\geq 1} \mathrm{Pr}'(n) = 2\sum_{n\geq 1} \mathrm{Pr}(n)/2 = 1.$

... continuous improvement.

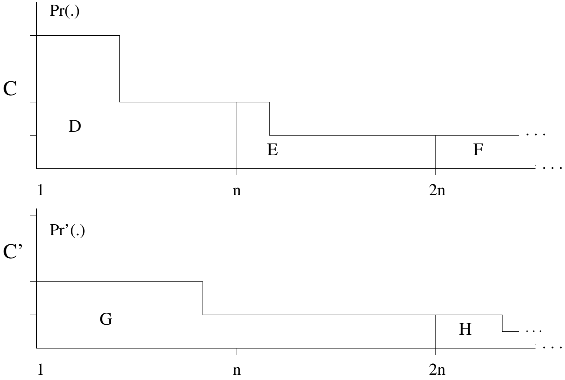


... continuous improvement.



- $D + E + F = G + H = 1,$

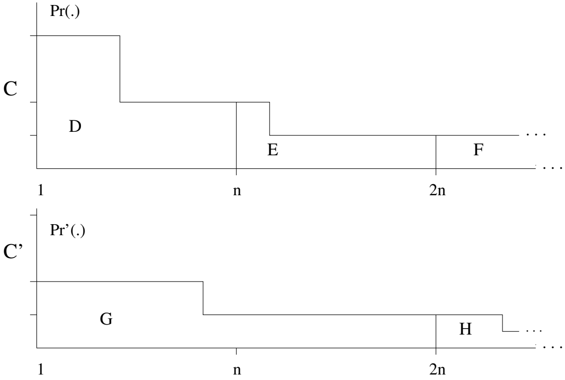
... continuous improvement.



- $D + E + F = G + H = 1$,

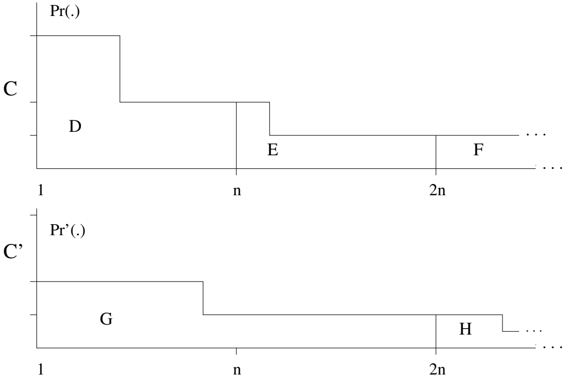
  now $D = G$, so $E + F = H$,

... continuous improvement.



- $D + E + F = G + H = 1$,
  now $D = G$, so $E + F = H$,
  so $H > F$.

... continuous improvement.



- $D + E + F = G + H = 1$,

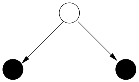
  now $D = G$, so $E + F = H$,

  so $H > F$.
- **C'** is "better" than **C**, *asymptotically*.

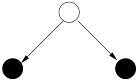
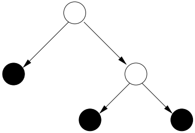
WTC ...

WTC0	WTC1	CW(.)	tree
0:	1:	“0”	

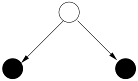
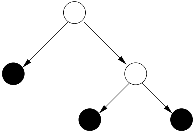
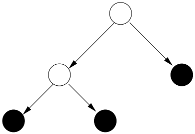
WTC ...

WTC0	WTC1	CW(.)	tree
0:	1:	"0"	
1:	2:	"100"	

WTC ...

WTC0	WTC1	CW(.)	tree
0:	1:	"0"	
1:	2:	"100"	
2:	3:	"10100"	

WTC ...

WTC0	WTC1	CW(.)	tree
0:	1:	"0"	
1:	2:	"100"	
2:	3:	"10100"	
3:	4:	"11000"	

...

... WTC, Catalans, paths

r↑								
5	1	5	14	28	42	42	0	...
4	1	4	9	14	14	0		
3	1	3	5	5	0			
2	1	2	2	0				
1	1	1	0					
0	1	0						
	0	1	2	3	4	5		→c

$paths$

$paths_{r,c} = \#$ Dyck paths from (r, c) to $(-1, 0)$.
$\leftarrow$ "1", $\downarrow$ "0".

... WTC, Catalans, paths

r↑								
5	1	5	14	28	42	42	0	...
4	1	4	9	14	14	0		
3	1	3	5	5	0			
2	1	2	2	0				
1	1	1	0					
0	1	0						
	0	1	2	3	4	5		→c

$paths$

$paths_{r,c} = \#$ Dyck paths from (r, c) to $(-1, 0)$.
$\leftarrow$ "1", $\downarrow$ "0".

$paths_{r,c} = paths_{r-1,c} + paths_{r,c-1}$, and boundary conditions.
$(paths_{f,f} = C_f.)$

... WTC, Catalans, paths

r↑ \ →c	0	1	2	3	4	5		
5	1	5	14	28	↓ 42	← 42	0	...
4	1	↓ 4	← 9	← 14	← 14	0		
3	↓ 1	← 3	5	5	0			
2	↓ 1	2	2	0				
1	↓ 1	1	0					
0	↓ 1	0						

$paths$

$paths_{r,c} = \#$ Dyck paths from (r, c) to $(-1, 0)$.
$\leftarrow$ "1", $\downarrow$ "0".

$paths_{r,c} = paths_{r-1,c} + paths_{r,c-1}$, and boundary conditions.
$(paths_{f,f} = C_f.)$

WTC0(35) = WTC1(36) = "10111010000"

... WTC complexity

- A l o n g compare and (for a "1") subtract give one bit of code, so provided paths$_{r,c}$ and the cummulative Catalans are cached and we have a "binary" representation of n, encode and decode take $O(\log(n))^2)$ time.

... WTC complexity

- A long compare and (for a "1") subtract give one bit of code, so provided $\text{paths}_{r,c}$ and the cummulative Catalans are cached and we have a "binary" representation of n, encode and decode take $O(\log(n))^2)$ time.
- ? slow for data compression, but OK for inference.
- Is there a faster algorithm?

... WTC complexity

- A l o n g compare and (for a "1") subtract give one bit of code, so provided paths$_{r,c}$ and the cummulative Catalans are cached and we have a "binary" representation of n, encode and decode take $O(\log(n))^2)$ time.
- ? slow for data compression, but OK for inference.
- Is there a faster algorithm?
- There is a fast approximation for $|\text{CW}(n)|$: $\log_2(n) + 1.5\log_2(\log_2(n)) + c$.

Comparisons [www]

n	Fibonacci	ω	ω^*	WTC
1	2	<u>1</u>	<u>1</u>	<u>1</u>
2	<u>3</u>	<u>3</u>	4	<u>3</u>
3	4	<u>3</u>	4	5
4	<u>4</u>	6	7	5
...	...	...	...	...
10^6	30	31	36	<u>27</u>
googol	480	349	354	<u>345</u>

(|CW(n)|)

Comparisons [www]

n	Fibonacci	ω	ω^*	WTC
1	2	<u>1</u>	<u>1</u>	<u>1</u>
2	<u>3</u>	<u>3</u>	4	<u>3</u>
3	4	<u>3</u>	4	5
4	<u>4</u>	6	7	5
...	...	...	...	...
10^6	30	31	36	<u>27</u>
googol	480	349	354	<u>345</u>

(|CW(n)|)

Fibonacci drops behind after $n = 317,811$.

Comparisons [www]

n	Fibonacci	ω	ω^*	WTC
1	2	<u>1</u>	<u>1</u>	<u>1</u>
2	<u>3</u>	<u>3</u>	4	<u>3</u>
3	4	<u>3</u>	4	5
4	<u>4</u>	6	7	5
...	...	...	...	...
10^6	30	31	36	<u>27</u>
googol	480	349	354	<u>345</u>

(|CW(n)|)

Fibonacci drops behind after $n = 317,811$.
WTC generally beats ω until $n = \text{cC}_{848} + 1$ when they $=$.

Comparisons [www]

n	Fibonacci	ω	ω^*	WTC
1	2	$\underline{1}$	$\underline{1}$	$\underline{1}$
2	$\underline{3}$	$\underline{3}$	4	$\underline{3}$
3	4	$\underline{3}$	4	5
4	$\underline{4}$	6	7	5
...	...	...	...	...
10^6	30	31	36	$\underline{27}$
googol	480	349	354	$\underline{345}$

(|CW(n)|)

Fibonacci drops behind after $n = 317,811$.
WTC generally beats ω until $n = \text{cC}_{848} + 1$ when they $=$.
ω beats WTC at $n = \text{cC}_{13,877,006}$ but not permanently.

Comparisons [www]

n	Fibonacci	ω	ω^*	WTC
1	2	$\underline{1}$	$\underline{1}$	$\underline{1}$
2	$\underline{3}$	$\underline{3}$	4	$\underline{3}$
3	4	$\underline{3}$	4	5
4	$\underline{4}$	6	7	5
...	...	...	...	...
10^6	30	31	36	$\underline{27}$
googol	480	349	354	$\underline{345}$

(|CW(n)|)

Fibonacci drops behind after $n = 317,811$.
WTC generally beats ω until $n = \mathrm{cC}_{848} + 1$ when they $=$.
ω beats WTC at $n = \mathrm{cC}_{13,877,006}$ but not permanently.
ω beats WTC permanently from ??? on.

Comparisons [www]

n	Fibonacci	ω	ω^*	WTC
1	2	<u>1</u>	<u>1</u>	<u>1</u>
2	<u>3</u>	<u>3</u>	4	<u>3</u>
3	4	<u>3</u>	4	5
4	<u>4</u>	6	7	5
...	...	...	...	...
10^6	30	31	36	<u>27</u>
googol	480	349	354	<u>345</u>

(|CW(n)|)

Fibonacci drops behind after $n = 317,811$.
WTC generally beats ω until $n = \text{cC}_{848} + 1$ when they $=$.
ω beats WTC at $n = \text{cC}_{13,877,006}$ but not permanently.
ω beats WTC permanently from ??? on.
ω^* beats ω from $n = 2 \uparrow\uparrow 8$ on.

Conclusions

- omega^2 and omega* are fun but not better in practice at compressing typical (ha, tiny!) integers.
- WTC compresses almost all small and "quite large" integers more than omega.
- WTC is nicer than omega and log* for inference,

  a bit slow for "data compression" of integers but

  fast enough for inference and

  there is a fast approximation for $|\mathsf{CW}(n)|$.

Interesting

- A. V. Levenstein, "*in Russian*", 1968, see D. Salomon, *Variable-length Codes for Data Compression*, Springer, p.80, 2007.
- F. Ruskey, *Generating balanced parenthesis strings by prefix shifts*, in CATS, Wollongong, pp.107-115, 2008.
- C. S. Wallace, *Statistical and Inductive Inference by Minimum Message Length*, Springer, 2005.

End