Techniques for Inverted Index Compression

Giulio Ermanno Pibiri and Rossano Venturini
Ca'Foscari University, Universita di Pisa

Published December 2020

This is a survey paper!

Index Compressors
A Single Integer

List Compressors
Many Integers Together (Inverted List)

Index Compressors
Many Lists Together (Whole Invertex Index)

+ Experiments!

Motivation: A Tour of the Terms

Terms

Forward Index	**Doc 1**	**Doc 2**	**Doc 3**
	Set(Term1, Term2)	*Set(Term2, Term3)*	*Set(Term1, Term3)*

Inverted Index	**Term 1**	**Term 2**	**Term 3**
	Doc [1, 3]	*Doc [1, 2]*	*Doc [2, 3]*

Motivation

HUGE

MUST Compress

FIND “chicken sandwich recipes” in DB

FIND friends with pet chickens

Well-Studied Problem

1949	Shannon-Fano [32, 93]
1952	Huffman [43]
1963	Arithmetic [1][1]
1966	Golomb [40]
1971	Elias-Fano [30, 33]; Rice [87]
1972	Variable-Byte and Nibble [101]
1975	Gamma and Delta [31]
1978	Exponential Golomb [99]
1985	Fibonacci-based [6, 37]
1986	Hierarchical bit-vectors [35]
1988	Based on Front Coding [16]
1996	Interpolative [65, 66]
1998	Frame-of-Reference (For) [39]; modified Rice [2]
2003	SC-dense [11]
2004	Zeta [8, 9]
2005	Simple-9, Relative-10, and Carryover-12 [3]; RBUC [60]
2006	PForDelta [114]; BASC [61]
2008	Simple-16 [112]; Tournament [100]
2009	ANS [27]; Varint-GB [23]; Opt-PFor [111]
2010	Simple8b [4]; VSE [96]; SIMD-Gamma [91]
2011	Varint-G8IU [97]; Parallel-PFor [5]
2013	DAC [12]; Quasi-Succinct [107]
2014	Partitioned Elias-Fano [73]; QMX [103]; Roaring [15, 51, 53]
2015	BP32, SIMD-BP128, and SIMD-FastPFor [50]; Masked-VByte [84]
2017	Clustered Elias-Fano [80]
2018	Stream-VByte [52]; ANS-based [63, 64]; Opt-VByte [83]; SIMD-Delta [104]; general-purpose compression libraries [77]
2019	DINT [79]; Slicing [78]

Previous Paper!

Integer Compressors

Inverted Index

Term 1

Doc [1, 3]

Term 2

Doc [1, 2]

Term 3

Doc [2, 3]

Goals

- Map each integer to a uniquely-decodable variable-length binary code
    - Assign the smallest codeword possible
- Prefix-Free
- *Lexicographic Assignment* (same lexicographic order as integers they represent, fast decodability)
- Shannon's Theorem: $|C(x)| = \log_2(1 / P(x))$ where $P(x)$ is the probability of x occurring

Encoding and Decoding Prefix-Free Codes

Table 2. Example Prefix-Free Code for the Integers 1..8, Along with Associated Codewords, Codeword Lengths, and Corresponding Left-Justified, 7-Bit Integers

(a)

x	Codewords	*Lengths*	*Values*
1	0	1	0
2	100	3	64
3	101	3	80
4	11000	5	96
5	11001	5	100
6	11010	5	104
7	11011	5	108
8	1110000	7	112
–	–	–	127

4 <= 6 < 8

(b)

Lengths	*First*	*Values*
1	1	0
2	2	64
3	2	64
4	4	96
5	4	96
6	8	112
7	8	112
–	9	127

The codewords are left-justified to better highlight their lexicographic order. In (b), the compact version of the table in (a), used by the encoding/decoding procedures coded in Figure 1. The "values" and "first" columns are padded with a sentinel value (in gray) to let the search be well defined.

Enc(6)

L = 5

Offset = 6 - First[L] = 6 - 4 = 2

Jump = (M - L) = 9 - 5 = 4

Value = (values[L] + offset * jump) = 96 + 2 * 4 = 104

Encoding and Decoding Prefix-Free Codes

Table 2. Example Prefix-Free Code for the Integers 1..8, Along with Associated Codewords, Codeword Lengths, and Corresponding Left-Justified, 7-Bit Integers

(a)

x	Codewords	*Lengths*	*Values*
1	0	1	0
2	100	3	64
3	101	3	80
4	11000	5	96
5	11001	5	100
6	11010	5	104
7	11011	5	108
8	1110000	7	112
–	–	–	127

(b)

Lengths	*First*	*Values*
1	1	0
2	2	64
3	2	64
4	4	96
5	4	96
6	8	112
7	8	112
–	9	127

The codewords are left-justified to better highlight their lexicographic order. In (b), the compact version of the table in (a), used by the encoding/decoding procedures coded in Figure 1. The "values" and "first" columns are padded with a sentinel value (in gray) to let the search be well defined.

Encode(x) :
- determine ℓ such that $first[\ell] \le x < first[\ell + 1]$
- $offset = x - first[\ell]$
- $jump = 1 \ll (M - \ell)$
- Write$((values[\ell] + offset \times jump) \gg (M - \ell), \ell)$

Decode() :
- determine ℓ such that $values[\ell] \le buffer < values[\ell + 1]$
- $offset = (buffer - values[\ell]) \gg (M - \ell)$
- $buffer = ((buffer \ll \ell)\ \&\ \text{MASK}) + \text{Take}(\ell)$
- **return** $first[\ell] + offset$

Encodings

Table 3. Integers 1..8 as Represented with Several Codes

x	U(x)	B(x)	$\gamma(x)$	$\delta(x)$	$G_2(x)$	Exp$G_2(x)$	$Z_2(x)$
1	0	0	0.	0.	0.0	0.00	0.0
2	10	1	10.0	100.0	0.1	0.01	0.10
3	110	10	10.1	100.1	10.0	0.10	0.11
4	1110	11	110.00	101.00	10.1	0.11	10.000
5	11110	100	110.01	101.01	110.0	10.000	10.001
6	111110	101	110.10	101.10	110.1	10.001	10.010
7	1111110	110	110.11	101.11	1110.0	10.010	10.011
8	11111110	111	1110.000	11000.000	1110.1	10.011	10.1000

The "." symbol highlights the distinction between different parts of the codes and has a purely illustrative purpose: it is not included in the final coded representation.

Unary Encoding

Table 3. Integers 1..8 as Represented with Several Codes

x	$\mathrm{U}(x)$	$\mathrm{B}(x)$	$\gamma($			$\mathrm{ExpG}_2(x)$	$\mathrm{Z}_2(x)$
1	0	0	0.			0.00	0.0
2	10	1	10.0			0.01	0.10
3	110	10	10.1	100.1	10.0	0.10	0.11
4	1110	11	110.00	101.00	10.1	0.11	10.000
5	11110	100	110.			10.000	10.001
6	111110	101	110.			10.001	10.010
7	1111110	110	110.			10.010	10.011
8	11111110	111	1110.000	11000.000	1110.1	10.011	10.1000

The "." symbol highlights the distinction between different parts of the codes and has a purely illustrative purpose: it is not included in the final coded representation.

$$\mathrm{U}(x) = 1^{x-1}0$$

Con: size is linear in x

Binary Encoding

Table 3. Integers 1..8 as Represented with Several Codes

x	$U(x)$	$B(x)$	$\gamma(x)$	[illegible]	[illegible]	[illegible]	$Z_2(x)$
1	0	0	0.	[illegible]	[illegible]	[illegible]	0.0
2	10	1	10.0	[illegible]	[illegible]	[illegible]	0.10
3	110	10	10.1	[illegible]	[illegible]	[illegible]	0.11
4	1110	11	110.00	101.00	10.1	0.11	10.000
5	11110	100	110.01	[illegible]	[illegible]	[illegible]	10.001
6	111110	101	110.10	[illegible]	[illegible]	[illegible]	10.010
7	1111110	110	110.11	[illegible]	[illegible]	[illegible]	10.011
8	11111110	111	1110.000	[illegible]	[illegible]	[illegible]	10.1000

The “.” symbol highlights the distinction betw [illegible] urely illustrative purpose: it is not included in the final coded r [illegible]

B(x) = bin(x-1)

Pro: Size is log(x)

Con: NOT uniquely decodable

Gamma Encoding

U(|bin(x)|) + (|bin(x)| - 1 LSB of bin(x))

Table 3. Integers 1..8 as Represented w

x	U(x)	B(x)	$\gamma(x)$	$\delta(x)$	G$_2(x)$	ExpG$_2(x)$	Z$_2(x)$
1	0	0	0.	0.			
2	10	1	10.0	100.0			
3	110	10	10.1	100.1			
4	1110	11	110.00	101.00	1		
5	11110	100	110.01	101.01	1		
6	111110	101	110.10	101.10	1		
7	1111110	110	110.11	101.11	1		
8	11111110	111	1110.000	11000.000	1		

Th ... tween different parts of
pu ... d representation.

Pro: Size is log(x) and uniquely decodable
Optimal for $P = 1/2x^2$

Example: 3
bin(3) = 11
|bin(3)| = 2
U(2) = 10
|bin(x)|-1 LSB of bin(3) = 1
=> 10.1

Con: can get big for big values

Delta Encoding

γ(|bin(x)|) + (|bin(x)| - 1 LSB of bin(x))

Table 3. Integers 1..8 as Represented w

x	$U(x)$	$B(x)$	$\gamma(x)$	$\delta(x)$	$G_2(x)$	$ExpG_2(x)$	$Z_2(x)$
1	0	0	0.	0.	0.0	0.00	0.0
2	10	1	10.0	100.0	0.1	0.01	0.10
3	110	10	10.1	100.1	10.0	0.10	0.11
4	1110	11	110.00	101.00	1		
5	11110	100	110.01	101.01	1		
6	111110	101	110.10	101.10	1		
7	1111110	110	110.11	101.11	1		
			000	11000.000	1		

n between different parts of

coded representation.

Pro: doesn't grow as fast!

Optimal for $P(x) = 1/(2x(\log_2 x)^2)$)

Example: 3

γ(x) = 100

bin(3) = 1**1**

|bin(x)|-1 LSB of bin(3) = 1

=> 100.1

Golomb Encoding

$q = \text{floor}((x-1) / b)$
$r = x - q*b - 1$
$\mathbf{G_b = U(q+1) \mid bin(r)}$

Table 3. Integers 1–8 as Represented with Several Codes

x	$U(x)$			(x)	$G_2(x)$	$ExpG_2(x)$	$Z_2(x)$
1	0				0.0	0.00	0.0
2	10				0.1	0.01	0.10
3	110				10.0	0.10	0.11
4	1110				10.1	0.11	10.000
5	11110				110.0	10.000	10.001
6	111110	101	110.10	101.10	110.1	10.001	10.010
7	1111110	110	110.11	101.11	1110.0	10.010	10.011
8	11111110	111	1110.000	11000.000	1110.1	10.011	10.1000

Example: 3
$q = \text{floor}((3-1) / 2) = 1$
$r = 3 - 1*2 - 1 = 0$
$G_b = U(1+1) \mid bin(0) =$ **10.0**

...on between different parts of the codes and has a purely illustrative ... coded representation.

Pro: Optimal for $P(x) = p(1-p)^{x-1}$

Rice Encoding

Golomb with b = 2^k, k > 0

Table 3. Integers 1–8 as Represented with Several Codes

Example: 3
q = floor((3-1) / 2) = 1
r = 3 - 1*2 - 1 = 0
G_b = U(1+1) | bin(0) = **10.0**

x	U(x)			x)	$G_2(x)$	$ExpG_2(x)$	$Z_2(x)$
1	0				0.0	0.00	0.0
2	10			0	0.1	0.01	0.10
3	110			1	10.0	0.10	0.11
4	1110			00	10.1	0.11	10.000
5	11110			01	110.0	10.000	10.001
6	111110	101	110.10	101.10	110.1	10.001	10.010
7	1111110	110	110.11	101.11	1110.0	10.010	10.011
8	11111110	111	1110.000	11000.000	1110.1	10.011	10.1000

The "." symbol highlights the distinction between different parts of the codes and has a purely illustrative purpose: it is not included in the final coded representation.

Exponential Golomb Encoding

h = bucket index

U(h) | (bin rep of x in range [0, B[h+1] - B[h] - 1])

Buckets

$$B = \left[0, 2^k, \sum_{i=0}^{1} 2^{k+i}, \sum_{i=0}^{2} 2^{k+i}, \sum_{i=0}^{3} 2^{k+i}, \ldots\right], \text{ for some } k \geq 0$$

d with Several Codes

					$G_2(x)$	$\mathrm{ExpG}_2(x)$	$Z_2(x)$
					0.0	0.00	0.0
					0.1	0.01	0.10
3	110	10	10.1	100.1	10.0	0.10	0.11
				101.00	10.1	0.11	10.000
				101.01	110.0	10.000	10.001
				101.10	110.1	10.001	10.010
				101.11	1110.0	10.010	10.011
				11000.000	1110.1	10.011	10.1000

ion between different parts of the codes and has a purely illustrative
al coded representation.

Example: 3
bucket_idx = 0
bin rep in bucket = 11
ExpG_2 = 0.11

Zeta Encoding

ExpGolomb relative to vector buckets [0, 2^k-1, 2^{2k}-1, 2^{3k}-1, ...]

Table 3. Integers 1..8 as Represented with Several Codes

x	U(x)	B(x)	$\gamma(x)$	$\delta(x)$	$G_2(x)$	$ExpG_2(x)$	$Z_2(x)$
1	0	0	0.	0.	0.0	0.00	0.0
2	10	1	10.0	100.0	0.1	0.01	0.10
3	110	10	10.1	100.1	10.0	0.10	0.11
			.00	101.00	10.1	0.11	10.000
			01	101.01	110.0	10.000	10.001
			10	101.10	110.1	10.001	10.010
			11	101.11	1110.0	10.010	10.011
			.000	11000.000	1110.1	10.011	10.1000

ion between different parts of the codes and has a purely illustrative
al coded representation.

Example: 3
bucket_idx = 0
bin rep in bucket = 11
Z_2 = 0.11

Variable-Byte

- Byte-aligned (simpler and faster to write into memory)
- Bin(x) is divided into the 7-bit sequences. Then you write a continuation bit (1 to continue, 0 to the termination of the sequence) at the start
- Nibble – 3 bits instead of 7
- Allows for more SIMD parallelism (ex: Varint-G8IU)

Example: 65790
bin(65790) = 10000000011111110
Chunk into groups of 7: 0000100 | 0000001 | 1111110
Add Control Bits: **0**0000100 | **1**0000001 | **1**1111110
VB(65790) = **0**0000100**1**0000001**1**1111110

SC-Dense

- Variable-Byte but with generalized stoppers
- SC(s, c, x) with k(x) >= 1 such that $s\frac{c^{k(x)-1}-1}{c-1} \leq x < s\frac{c^{k(x)}-1}{c-1}$.
  - If k(x)=1, the stopper is x-1
  - y = floor((x-1) / x)
  - $x' = x - sc^{k(x)-1} - s / (c-1)$
  - Representation has k(x) -1 continuers
  - Repeat continuers for k(x)-2 times followed by continuer s + ((y-1) mod c) and final stopper (x' - 1) mod s

Table 5. Integers 1..20 as Represented by SC(4, 4)- and SC(5, 3)-Dense Codes, Respectively

x	$SC(4,4,x)$	$SC(5,3,x)$
1	000	000
2	001	001
3	010	010
4	011	011
5	100.000	100
6	100.001	101.000
7	100.010	101.001
8	100.011	101.010
9	101.000	101.011
10	101.001	101.100

x	$SC(4,4,x)$	$SC(5,3,x)$
11	101.010	110.000
12	101.011	110.001
13	110.000	110.010
14	110.001	110.011
15	110.010	110.100
16	110.011	111.000
17	111.000	111.001
18	111.001	111.010
19	111.010	111.011
20	111.011	111.100

Overview of the Encoders

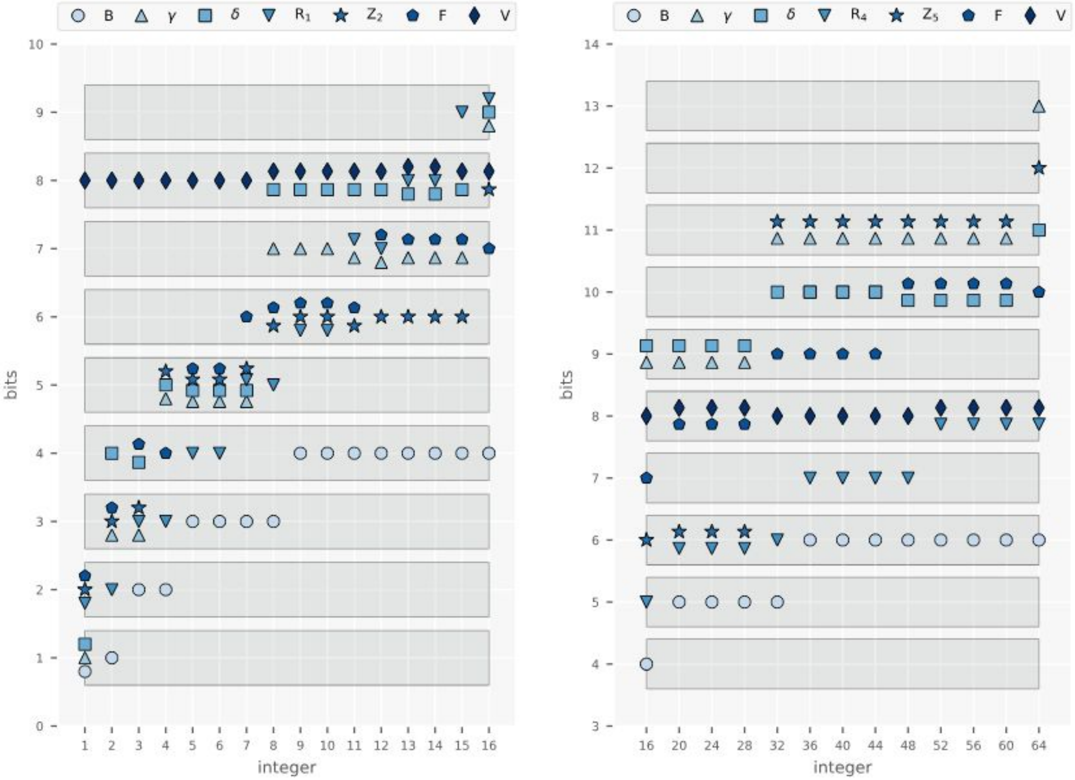


Fig. 3. Comparison between several codes described in Section 2 for the integers 1..64.

List Compressors

Inverted Index

Term 1

Doc [1, 3]

Term 2

Doc [1, 2]

Term 3

Doc [2, 3]

Binary Packing

- Partition into blocks and encode each block separately
  - If blocks has clusters of close integers, the values in a block are likely be of similar magnitude
- Binary Packing
  - Use bit width $\text{ceil}(\log_2(\max + 1))$ of the maximum element then represent all integers in the block with b-bit codewords
- Can also do a variable amount

Simple

- Split the sequence into fixed-memory units and ask how many integers can be packed in a unit
  - 4 bit selector code followed by the integers (for example, in 32-bit word)
- Typically good compression and decoding speed

4-Bit Selector	Integers	Bits per Integer	Wasted Bits
0000	28	1	0
0001	14	2	0
0010	9	3	1
0011	7	4	0
0100	5	5	3
0101	4	7	0
0110	3	9	1
0111	2	14	0
1000	1	28	0

Read: packs 28
1-bit integers

PForDelta

- Problem with previous implementations: LOTS of wasted space if there’s one big value and then rest are small
- Choose k such that >= 90% of integers can be represented using k bits per integer
    - If they do not fit, they are **exceptions** and encoded using another compressor (Variable-Byte or Simple)

Elias-Fano

- Let S(n, U) indicate a sorted sequence S[1..n]
- Break each S[i] into two parts
  - Low bits: L = floor($\log_2$(U/n))
  - High bits: ceil($\log_2$U) - L
- Low bits are encoded separately with a bit-vector L of n ceil($\log_2$(U/n)) bits
- High bits are encoded separately with a bit-vector of <= 2n bits by setting the bit in position h_i + i for all i = 1, ..., n
- Concat these two representations

Elias-Fano

Table 7. Example of Elias-Fano Encoding Applied to the Sequence $\mathcal{S} = [3, 4, 7, 13, 14, 15, 21, 25, 36, 38, 54, 62]$

$\mathcal{S}$	3	4	7	13	14	15	21	25	36	38		54	62
	0	0	0	0	0	0	0	0	1	1	1	1	1
high	0	0	0	0	0	0	1	1	0	0	0	1	1
	0	0	0	1	1	1	0	1	0	0	1	0	1
	0	1	1	1	1	1	1	0	1	1		1	1
low	1	0	1	0	1	1	0	0	0	1		1	1
	1	0	1	1	0	1	1	1	0	0		0	0
H	1110			1110			10	10	110		0	10	10
L	011.100.111			101.110.111			101	001	100.110			110	110

Elias-Fano – Random Access

- Add auxiliary data structures to make this possible
- For $\text{Select}_b(i)$, use a bit-vector to get the position of the ith bit set to b
  - Requires $o(n)$ additional bits
- **O(1)** runtime

Elias-Fano – Successor Queries

- $\text{Select}_{,b}(x)$
- h_x = high bits of x
- Binary Search from $\text{Select}_b(h_x) - h_x + 1$ to $\text{Select}_b(h_x+1) - h_x$
- **O(1 + log(U/n))** runtime

Elias-Fano – Partitioning by Universe

- **Roaring:** partition $U(2^{32})$ into chunk spanning 2^{16} values each
  - If a chunk is sparse (less than 2^{12} elements), encode as a sorted array of 16-bit integers.
  - If a chunk is dense (more than 2^{12} elements), encode as a bitmap.
  - If a chunk is full (2^{16} elements), encode implicitly.

- **Slicing:** Roaring but continue encoding recursively if the chunk is sparse (at most 2^8 blocks of 2^8 elements each)

Interpolative

- Represents a sorted integer sequence without requiring the computation of its gaps
- Exploit the order of the already-encoded elements to compute the number of bits needed to represent the elements that will be encoded next
- Idea: recursively divide, encode middle element with minimum bits

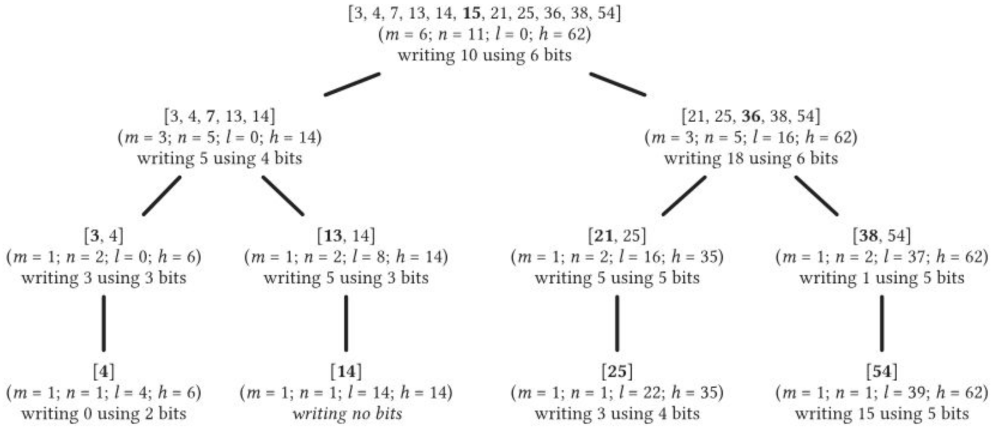


Fig. 4. The recursive calls performed by the Binary Interpolative Coding algorithm when applied to the sequence [3, 4, 7, 13, 14, 15, 21, 25, 36, 38, 54] with initial knowledge of lower and upper bound values $l = 0$ and $h = 62$. In bold font, we highlight the middle element being encoded.

Directly-Addressable Codes

- Representation for a list of integers that supports random access to individual integers
  - Problem of random access reduced to one of **ranking over a bitmap**

Entropy Coding

- Huffman → too large alphabet (H_0 <= L <= H_0 + 1 where H_0 is the entropy)
- Arithmetic → takes nH_0 + 2 bits to encode sequence S of length n with H_0 entropy
- Asymmetric Numeral Systems (ANS) → represent sequence of symbols with a natural number x

Table 8. Two Examples of the ANS Encoding Table with Frame $f[1..6] = [aaabbc]$ and $f[1..4] = [caba]$, Respectively

(a)

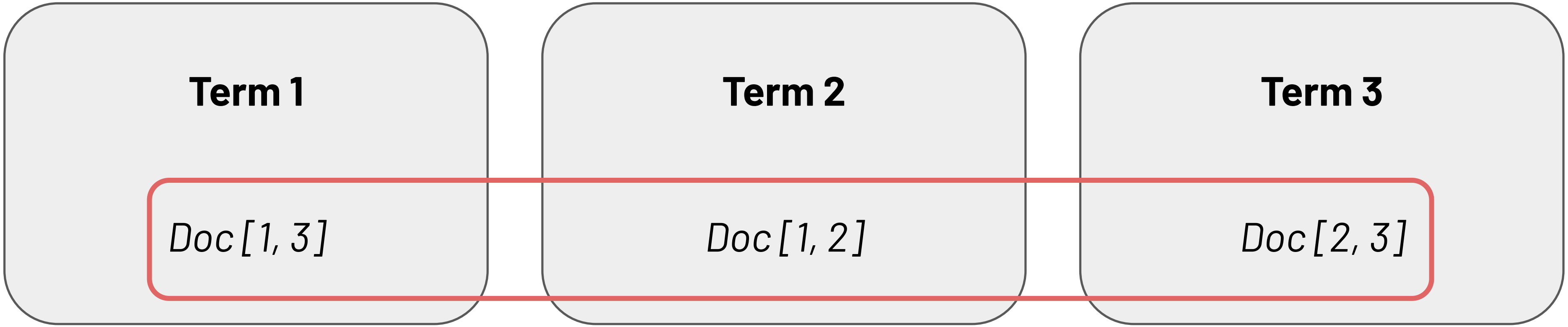
Σ	$\mathbb{P}$	Codes									
a	1/2	1	2	3	7	8	9	13	14	15	19
b	1/3	4	5	10	11	16	17	22	23	28	29
c	1/6	6	12	18	24	30	36	42	48	54	60
		0	1	2	3	4	5	6	7	8	9

(b)

Σ	$\mathbb{P}$	Codes									
a	1/2	2	4	6	8	10	12	14	16	18	20
b	1/4	3	7	11	15	19	23	27	31	35	39
c	1/4	1	5	9	13	17	21	25	29	33	37
		0	1	2	3	4	5	6	7	8	9

Index Compressors

Inverted Index



Clustered

- **Idea: Group lists into clusters and encode from a reference list**
- Inverted lists grouped into clusters of similar lists (sharing as many integers as possible)
- Reference List for each cluster → all lists in cluster encoded with respect to the reference list
- Improvement: each intersection can be rewritten in a much smaller universe

ANS Based

- **Idea: Preprocessing to reduce alphabet size then apply ANS**
- Option 1:
  - Preprocess with Variable-Byte to reduce input list to a sequence of bytes
  - Apply ANS
- Option 2:
  - Preprocess with Simple
  - Apply ANS
- Option 3:
  - Packed processing
  - Apply ANS

Dictionary Based

- **Idea: gaps are repetitive, abstract it into a dictionary**
- Dictionary stores the most frequent 2^b patterns ($b > 0$)

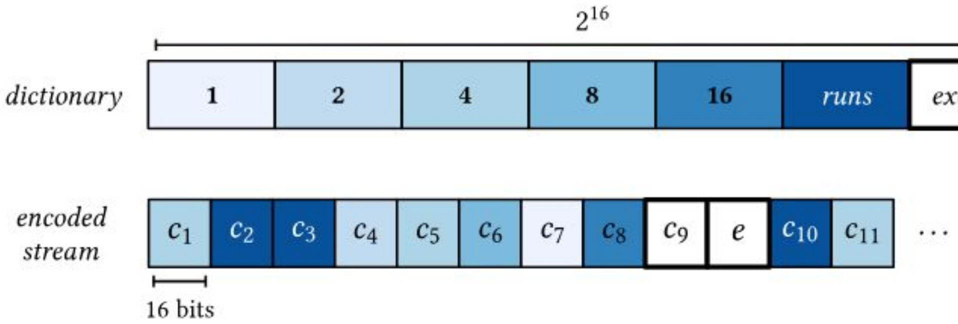


Fig. 6. A dictionary-based encoded stream example, where dictionary entries corresponding to $\{1, 2, 4, 8, 16\}$-long integer patterns, runs, and exceptions are labeled with different shades. Once provision has been made for such a dictionary structure, a sequence of gaps can be modeled as a sequence of codewords $\{c_k\}$, each being a reference to a dictionary entry, as represented with the *encoded stream* in the picture. Note that, for example, codeword c_9 signals an exception, and therefore the next symbol e is decoded using an escape mechanism.

Experiments!

Metrics

Space Effectiveness

Average number of bits for the representation of a document identifier

Time Efficiency

Time needed to perform sequential decoding, intersection, and union of inverted lists

Experimental Setup

Tests random 1k of each number of queries per intersection/union

(a) Basic Statistics

	Gov2	ClueWeb09	CCNews
Lists	39,177	96,722	76,474
Universe	24,622,347	50,131,015	43,530,315
Integers	5,322,883,266	14,858,833,259	19,691,599,096
Entropy of the gaps	3.02	4.46	5.44
$\lceil \log_2 \rceil$ of the gaps	1.35	2.28	2.99

(b) TREC 2005/06 Queries

	Gov2	ClueWeb09	CCNews
Queries	34,327	42,613	22,769
2 terms	32.2%	33.6%	37.5%
3 terms	26.8%	26.5%	27.3%
4 terms	18.2%	17.7%	16.8%
5+ terms	22.8%	22.2%	18.4%

Method	Partitioned by	SIMD	Alignment	Description
VByte	Cardinality	Yes	Byte	Fixed-size partitions of 128
Opt-VByte	Cardinality	Yes	Bit	Variable-size partitions
BIC	Cardinality	No	Bit	Fixed-size partitions of 128
δ	Cardinality	No	Bit	Fixed-size partitions of 128
Rice	Cardinality	No	Bit	Fixed-size partitions of 128
PEF	Cardinality	No	Bit	Variable-size partitions
DINT	Cardinality	No	16-bit word	Fixed-size partitions of 128
Opt-PFor	Cardinality	No	32-bit word	Fixed-size partitions of 128
Simple16	Cardinality	No	64-bit word	Ffixed-size partitions of 128
QMX	Cardinality	Yes	128-bit word	Fixed-size partitions of 128
Roaring	Universe	Yes	byte	Single span
Slicing	Universe	Yes	byte	Multi-span

64 GB RAM DDR4
Linux 5
C++, GCC 9.2.1
Indexes in internal memory, data structures on disk
Average across three runs

Results (Space + Decoding Time)

Table 11. Space Effectiveness in Total GiB and Bits per Integer, and Nanoseconds per Decoded Integer

Method	Gov2			ClueWeb09			CCNews		
	GiB	Bits/int	ns/int	GiB	Bits/int	ns/int	GiB	bits/int	ns/int
VByte	5.46	8.81	0.96	15.92	9.20	1.09	21.29	9.29	1.03
Opt-VByte	2.41	3.89	0.73	9.89	5.72	0.92	14.73	6.42	0.72
BIC	1.82	2.94	5.06	7.66	4.43	6.31	12.02	5.24	6.97
δ	2.32	3.74	3.56	8.95	5.17	3.72	14.58	6.36	3.85
Rice	2.53	4.08	2.92	9.18	5.31	3.25	13.34	5.82	3.32
PEF	1.93	3.12	0.76	8.63	4.99	1.10	12.50	5.45	1.31
DINT	2.19	3.53	1.13	9.26	5.35	1.56	14.76	6.44	1.65
Opt-PFor	2.25	3.63	1.38	9.45	5.46	1.79	13.92	6.07	1.53
Simple16	2.59	4.19	1.53	10.13	5.85	1.87	14.68	6.41	1.89
QMX	3.17	5.12	0.80	12.60	7.29	0.87	16.96	7.40	0.84
Roaring	4.11	6.63	0.50	16.92	9.78	0.71	21.75	9.49	0.61
Slicing	2.67	4.31	0.53	12.21	7.06	0.68	17.83	7.78	0.69

Simpler, byte-aligned code, bitmaps, SIMD instructions

Results (Time for AND Queries)

Table 12. Milliseconds Spent per AND Query by Varying the Number of Query Terms

Method	Gov2					ClueWeb09					CCNews				
	2	3	4	5+	avg.	2	3	4	5+	avg.	2	3	4	5+	avg.
VByte	2.2	2.8	2.7	3.3	2.8	10.2	12.1	13.7	13.9	12.5	14.0	22.4	19.7	21.9	19.5
Opt-VByte	2.8	3.1	2.8	3.2	3.0	12.2	13.3	14.0	13.6	13.3	16.0	23.2	19.6	20.3	19.8
BIC	6.8	9.7	10.4	13.2	10.0	31.7	44.2	51.5	53.8	45.3	45.6	79.7	76.9	88.8	72.8
δ	4.6	6.3	6.5	8.2	6.4	20.9	28.3	33.5	34.5	29.3	28.6	50.9	48.0	55.6	45.8
Rice	4.1	5.6	5.8	7.3	5.7	19.2	25.7	30.2	31.1	26.6	26.5	46.5	43.5	50.1	41.6
PEF	2.5	3.1	2.8	3.2	2.9	12.3	13.5	14.4	13.8	13.5	17.2	24.6	21.0	21.9	21.2
DINT	2.5	3.3	3.3	4.1	3.3	11.9	14.6	16.5	17.1	15.0	16.9	27.3	24.6	28.1	24.2
Opt-PFor	2.6	3.5	3.5	4.3	3.5	12.8	15.9	18.0	18.3	16.3	16.6	27.2	24.3	27.1	23.8
Simple16	2.8	3.7	3.7	4.6	3.7	12.8	16.3	18.4	18.9	16.6	17.6	28.8	26.3	29.5	25.5
QMX	2.0	2.6	2.5	3.0	2.5	9.6	11.5	13.0	13.1	11.8	13.3	21.5	18.8	20.8	18.6
Roaring	0.3	0.5	0.7	0.8	0.6	1.5	2.5	3.1	4.3	2.9	1.1	2.0	2.6	4.1	2.5
Slicing	0.3	1.0	1.2	1.6	1.0	1.5	4.5	5.4	6.7	4.5	1.8	4.3	5.1	6.0	4.3

Results (Time for OR Queries)

Table 13. Milliseconds Spent per OR Query by Varying the Number of Query Terms

Method	Gov2					ClueWeb09					CCNews				
	2	3	4	5+	avg.	2	3	4	5+	avg.	2	3	4	5+	avg.
VByte	6.8	24.4	54.7	131.7	54.4	20.1	71.3	156.0	379.5	156.7	24.4	94.5	178.8	391.4	172.3
Opt-VByte	11.0	35.7	77.4	176.0	75.0	31.3	101.4	213.4	500.1	211.6	36.4	128.0	232.0	510.4	226.7
BIC	16.7	50.3	105.0	238.8	102.7	49.9	145.3	290.4	668.2	288.4	64.4	193.8	332.6	692.5	320.8
δ	12.6	40.8	87.9	202.5	85.9	34.9	112.9	236.7	557.7	235.6	42.2	144.9	263.8	571.3	255.5
Rice	13.4	43.1	93.3	211.3	90.3	36.8	118.2	248.5	576.6	245.0	43.6	149.3	270.5	585.6	262.2
PEF	10.2	33.0	71.7	164.2	69.8	31.1	99.7	208.5	492.3	207.9	37.6	127.5	232.6	507.1	226.2
DINT	8.5	28.5	63.7	147.6	62.1	24.9	84.1	178.8	424.3	178.0	30.6	109.2	200.4	432.7	193.2
Opt-PFor	8.9	31.1	69.4	161.4	67.7	27.0	90.8	194.0	453.5	191.3	31.3	113.2	209.0	447.2	200.2
Simple16	7.8	26.2	58.3	138.2	57.6	23.7	78.0	165.5	394.7	165.5	28.7	101.5	185.3	397.8	178.4
QMX	6.6	23.8	53.4	128.1	53.0	19.7	70.0	153.2	377.9	155.2	24.0	92.6	175.2	382.4	168.6
Roaring	1.2	2.8	4.3	6.4	3.7	4.7	9.0	12.0	15.7	10.3	3.8	7.6	10.5	15.1	9.2
Slicing	1.3	4.0	6.3	9.2	5.2	5.0	12.8	18.1	25.3	15.3	5.8	12.9	17.3	23.0	14.8

Results (Time/Space)

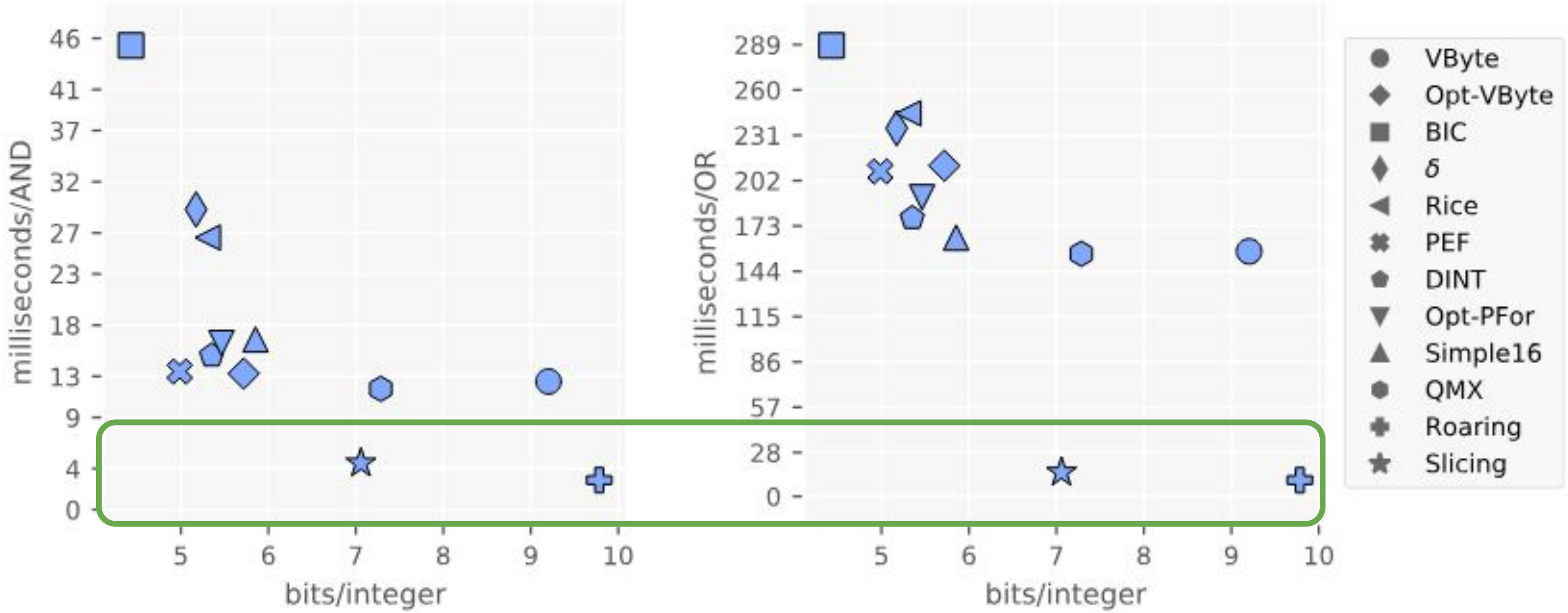


Fig. 7. Space/time trade-off curves for the ClueWeb09 dataset.

Future Directions, Strengths/Weaknesses, Questions

The Punchline: Directions for Future Work

1. **Simpler compression formats** that can be decoded with low-latency instructions (bitwise) and few branches

2. Devising **dynamic and compressed representations** for integer sequences that can support additions and deletions

Strengths/Weaknesses

- Strengths
  - Thorough description and evaluation of the different compressors

- Weakness
  - Nothing really novel contributed except maybe the comparisons(makes sense, it’s a survey paper!)
  - The evaluation could have explored more realistic queries instead of random queries

Discussion Questions

1. What are the benefits of these survey papers?

2. What are some trends you notice with the algorithms?
   a. I'm definitely noticing (1) a shift from theoretically how small can the codes be to how can we make it faster (ex: byte-aligned) and (2) over time, the algorithms seems to get more complex

3. Which encoding schemes would you use in real-life scenarios (compaction vs. performance)? What are interesting areas these compressions schemes can be used beyond inverted index compression?
   a. Could some reasonably be used for audio/video/image compression?