Partitioned Elias-Fano Indexes

Giuseppe Ottaviano
ISTI-CNR, Pisa

Rossano Venturini
Università di Pisa

Inverted indexes

Docid

Document

1: [it is what it is not]

2: [what is a]

3: [it is a banana]

Posting list

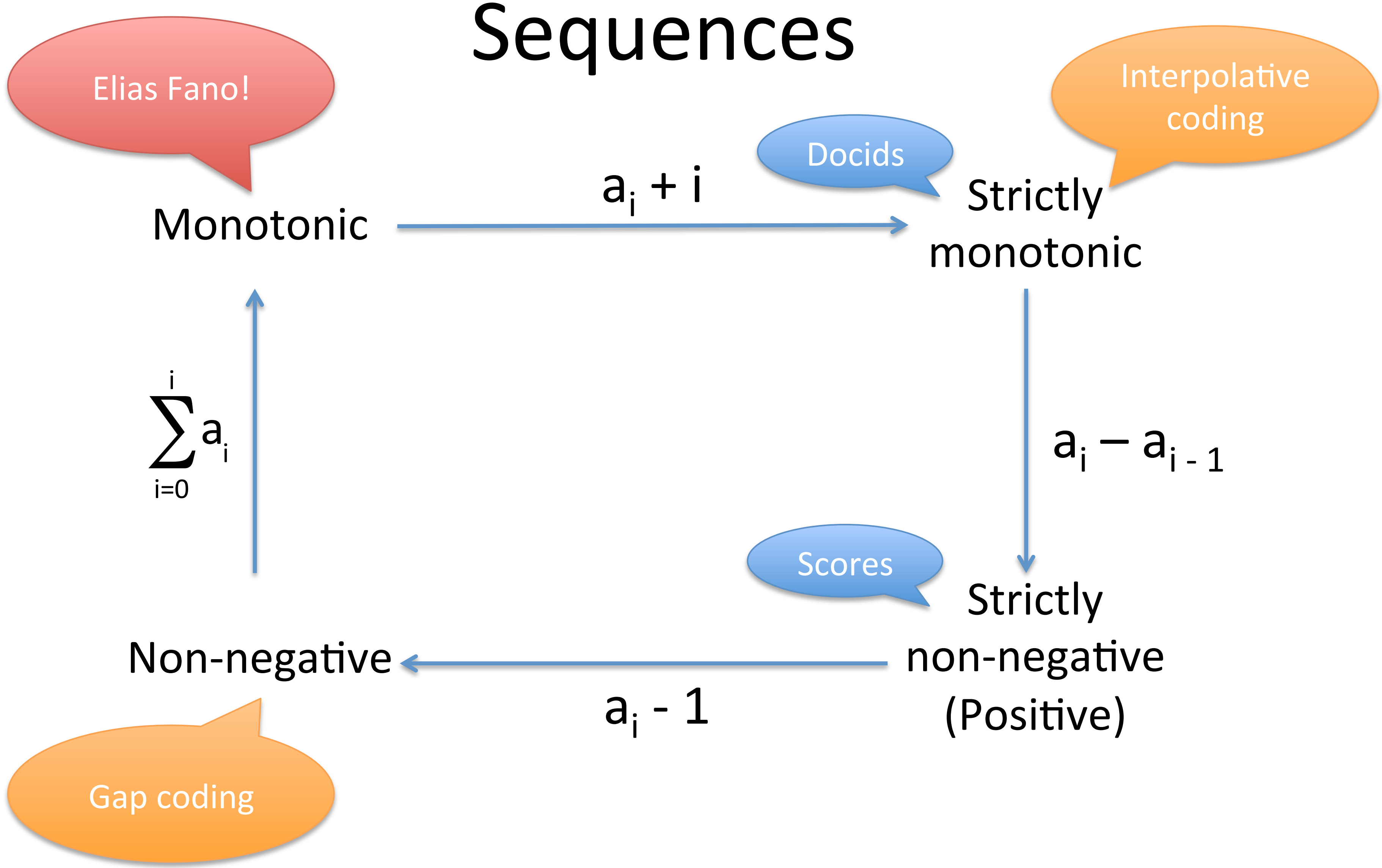
a	
banana	3
is	1, 2, 3
it	1, 3
not	1
what	1, 2

- Core data structure of Information Retrieval
- We seek fast and space-efficient encoding of posting lists (index compression)

Sequences in posting lists

- Generally, a posting lists is composed of
  - Sequence of docids: strictly monotonic
  - Sequence of frequencies/scores: strictly positive
  - ~~Sequence of positions: concatenation of strictly monotonic lists~~
  - ~~Additional occurrence data: ???~~
- We focus on docids and frequencies

Sequences



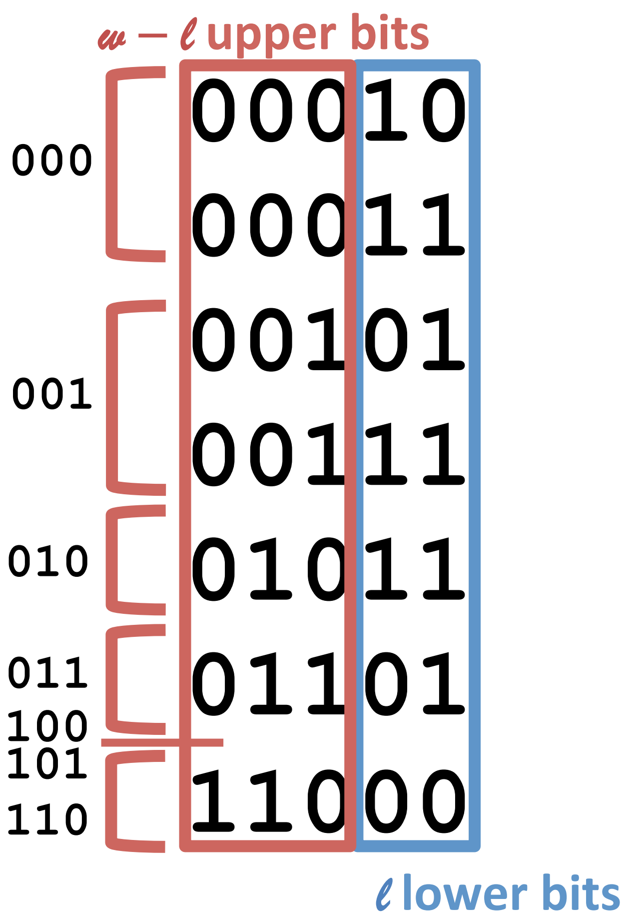
Elias Fano!
Monotonic
$a_i + i$
Docids
Interpolative coding
Strictly monotonic
$a_i - a_{i-1}$
Scores
Strictly non-negative (Positive)
$a_i - 1$
Non-negative
Gap coding
$\sum_{i=0}^{i} a_i$

Elias-Fano encoding

- Data structure from the ’70s, mostly in the succinct data structures niche
- Natural encoding of monotonically increasing sequences of integers
- Recently successfully applied to inverted indexes [Vigna, WSDM13]
  - Used by Facebook Graph Search!

Elias-Fano representation

Example: 2, 3, 5, 7, 11, 13, 24



Count in unary the size of upper bits "buckets" including empty ones

11011010100010

Concatenate lower bits

10110111110100

1101101010001010110111110100

Elias-Fano representation of the sequence

Elias-Fano representation

Example: 2, 3, 5, 7, 11, 13, 24

$w - \ell$ upper bits	ℓ lower bits
000	10
000	11
001	01
001	11
010	11
011	01
110	00

11011010100010 10110111110100

Elias-Fano representation of the sequence

n: sequence length
U: largest sequence value

Maximum bucket: $\lceil U / 2^\ell \rceil$
Example: $\lceil 24 / 2^2 \rceil = 6 =$ 110

Upper bits: one **0** per bucket and one **1** per value

Space

$\lceil U / 2^\ell \rceil + n + n\ell$ bits

Elias-Fano representation

Example: 2, 3, 5, 7, 11, 13, 24

$w - \ell$ upper bits	ℓ lower bits
000	10
000	11
001	01
001	11
010	11
011	01
110	00

Can show that
$\ell = [\log(U/n)]$
is optimal

$[U / 2^{\ell}] + n + n\ell$ bits

$(2 + \log(U/n))n$ bits

U/n is "average gap"

Elias-Fano representation

Example: 2, 3, 5, 7, 11, 13, 24

$w - \ell$ upper bits	ℓ lower bits
000	10
000	11
→ 001	01
→ 001	11
010	11
011	01
110	00

nextGEQ(6) = 7

$\lfloor 6 / 2^2 \rfloor = 1 = 001$

Find the first GEQ bucket

= find the 1-th 0 in upper bits

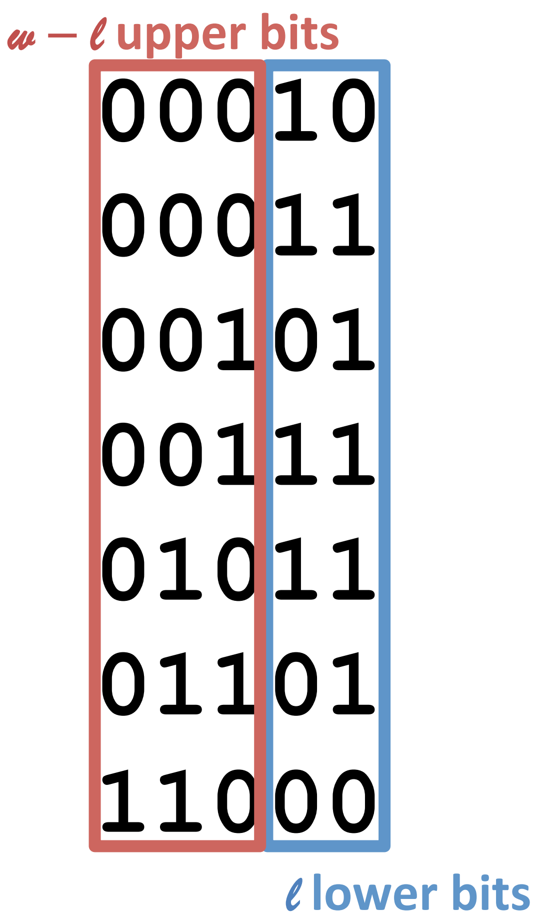
11011010100010

With additional data structures and broadword techniques -> O(1)

Linear scan inside the (small) bucket

Elias-Fano representation

Example: 2, 3, 5, 7, 11, 13, 24



11011010100010 10110111110100

Elias-Fano representation of the sequence

(2 + log(U/n))n-bits space
independent of values distribution!

… is this a good thing?

Term-document matrix

- Alternative interpretation of inverted index

a	2, 3
banana	3
is	1, 2, 3
it	1, 3
not	1
what	1, 2

	1	2	3
a		**X**	**X**
banana			**X**
is	**X**	**X**	**X**
it	**X**		**X**
not	**X**		
what	**X**	**X**	

- Gaps are distances between the **X**s

Gaps are *usually* small

- Assume that documents from the same domain have similar docids

	…	unipi.it/	unipi.it/ students	unipi.it/ research	unipi.it/…/ ottaviano	…	sigir.org/	sigir.org/ venue	sigir.org/ fullpapers	…
…										
pisa		X	X	X	X				X	
…										
sigir					X		X	X	X	
…										

"Clusters" of docids

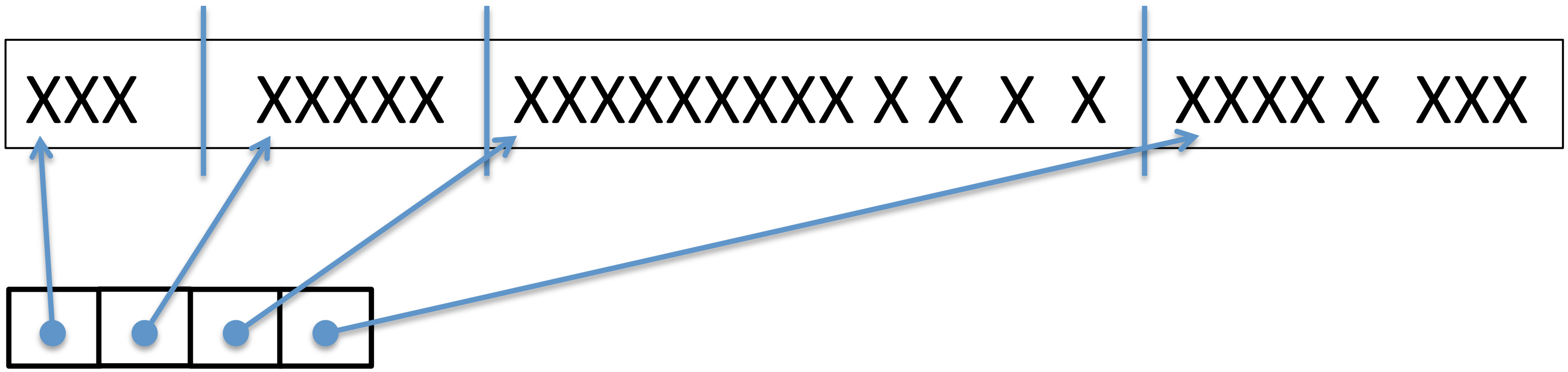
Posting lists contain long runs of very close integers

– That is, long runs of very small gaps

Elias-Fano and clustering

- Consider the following two lists
  - 1, 2, 3, 4, …, $n - 1$, U
  - n random values between 1 and U
- Both have n elements and largest value U
  - Elias-Fano compresses both to the exact same number of bits: $(2 + \log(U/n))n$
- But first list is far more compressible: it is "sufficient" to store n and U: $O(\log n + \log U)$
- Elias-Fano doesn't exploit *clustering*

Partitioned Elias-Fano

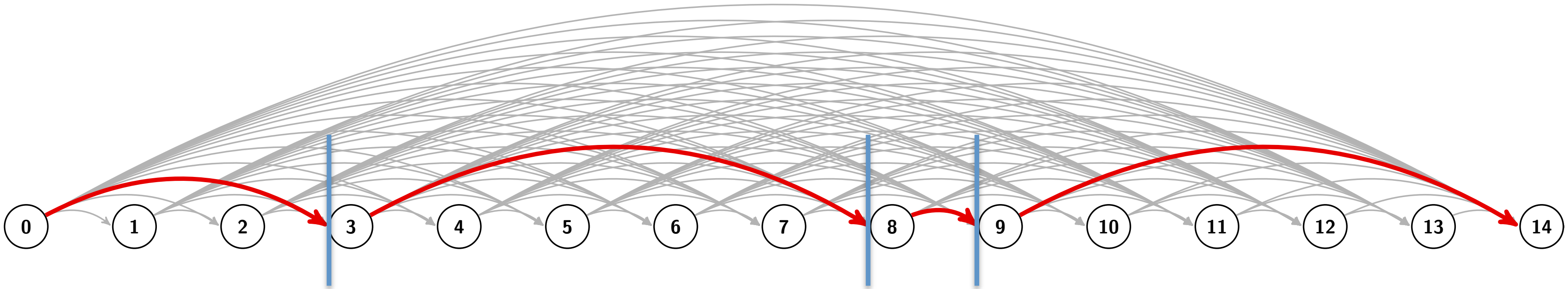


- Partition the sequence into *chunks*
- Add *pointers* to the beginning of each chunk
- Represent each chunk and the sequence of pointers with Elias-Fano
- If the chunks "approximate" the clusters, compression improves

Partition optimization

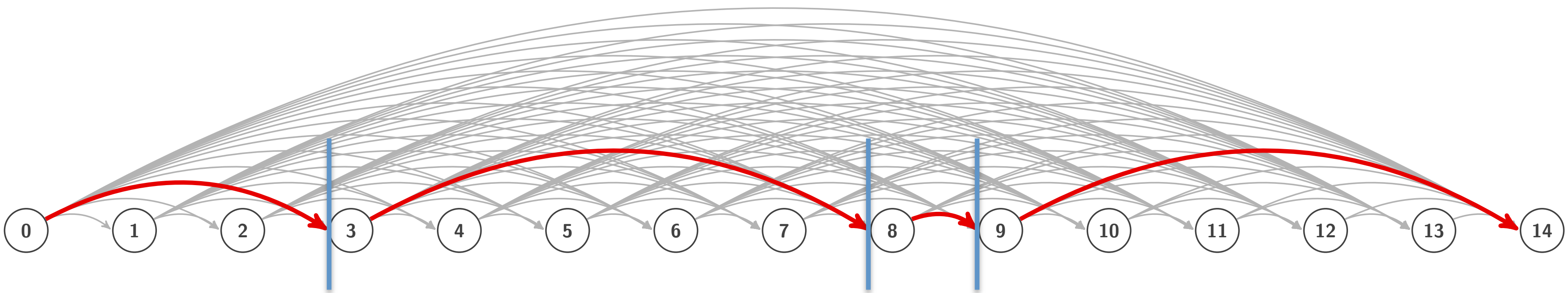
- We want to find, among all the possible partitions, the one that takes up less space
- Exhaustive search is *exponential*
- Dynamic programming can be done *quadratic*
- Our solution: $(1 + \varepsilon)$-approximate solution in linear time $O(n \log(1/\varepsilon)/\log(1 + \varepsilon))$
  - Reduce to a shortest path in a *sparsified* DAG

Partition optimization



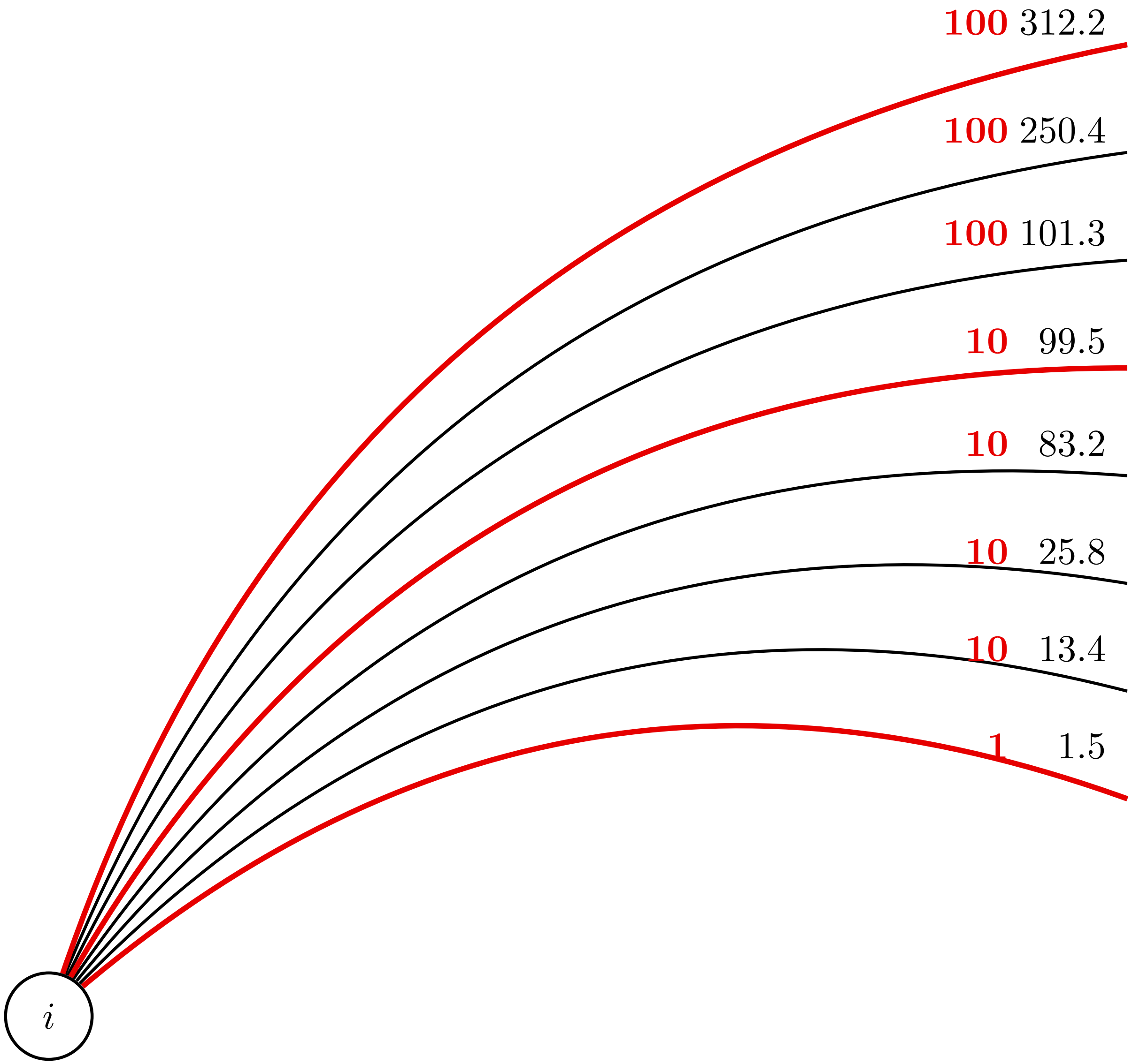
- Nodes correspond to sequence elements
- Edges to potential chunks
- Paths = Sequence partitions

Partition optimization



- Each edge weight is the cost of the chunk defined by the edge endpoints
- Shortest path = Minimum cost partition
- Edge costs can be computed in O(1)...
- ... but number of edges is quadratic!

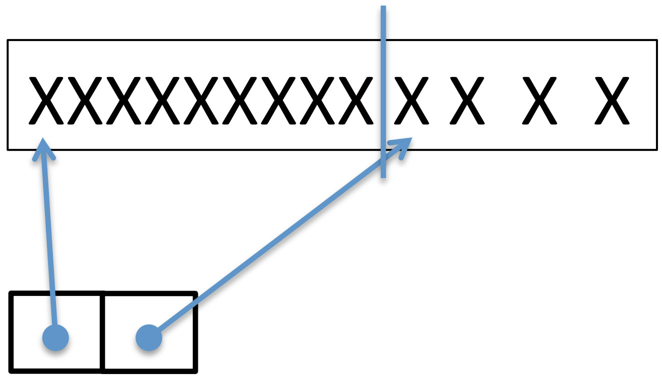
Sparsification: idea n.1



Sparsification: idea n.1

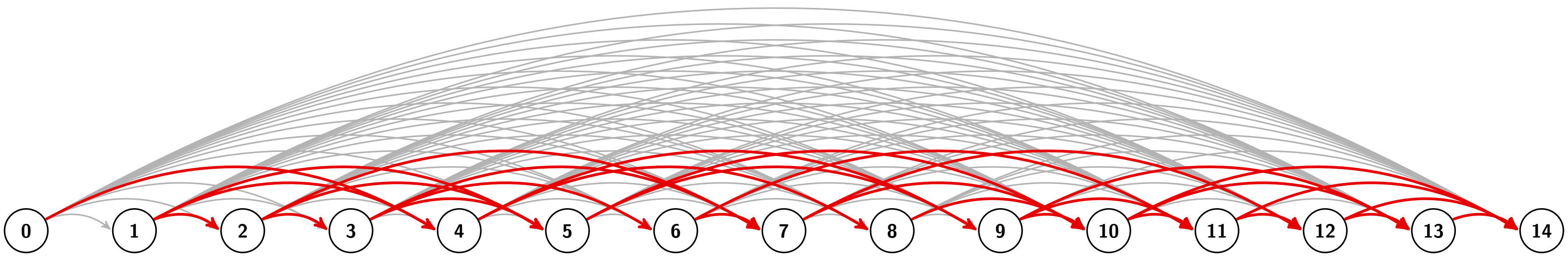
- General DAG *sparsification* technique
- Quantize edge costs in *classes* of cost between $(1 + \varepsilon_1)^i$ and $(1 + \varepsilon_1)^{i+1}$
- For each node and each cost class, keep only one maximal edge
  - $O(\log n / \log (1 + \varepsilon_1))$ edges per node!
- Shortest path in sparsified DAG at most $(1 + \varepsilon_1)$ times more expensive than in original DAG
- Sparsified DAG can be computed *on the fly*

Sparsification: idea n.2



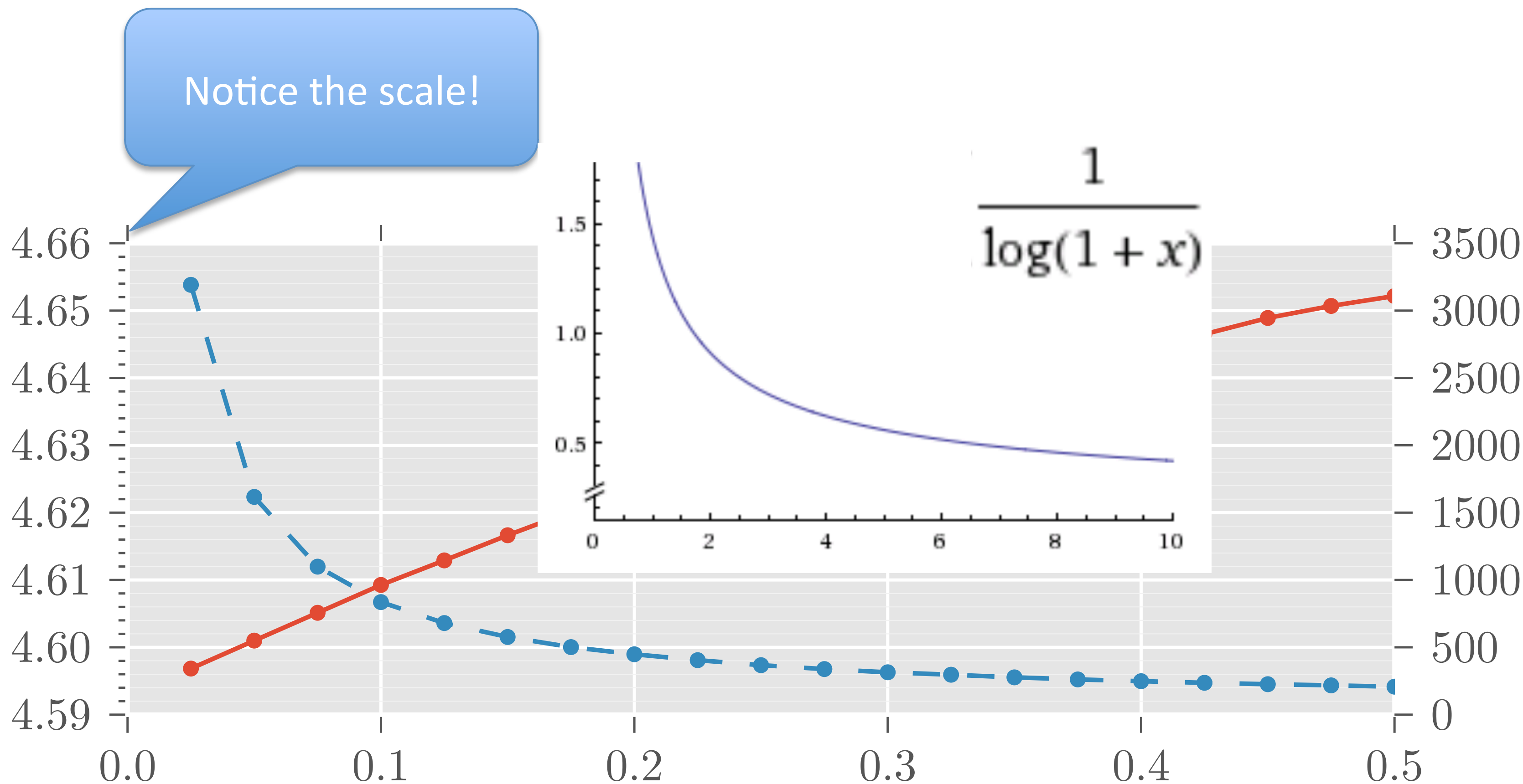
- If we split a chunk at an arbitrary position
  - New cost ≤ Old cost + 1 + cost of new pointer
- If chunk is "big enough", loss is negligible
- We keep only edges with cost $O(1 / \varepsilon_2)$
- At most $O(\log (1 / \varepsilon_2) / \log (1 + \varepsilon_1))$ edges/node

Sparsification



- Sparsified DAG has $O(n \log (1 / \varepsilon_2) / \log (1 + \varepsilon_1))$ edges!
- Fixed ε_i, it is $O(n)$ vs $O(n^2)$ in original DAG
- Overall approximation factor is $(1 + \varepsilon_2)(1 + \varepsilon_1)$

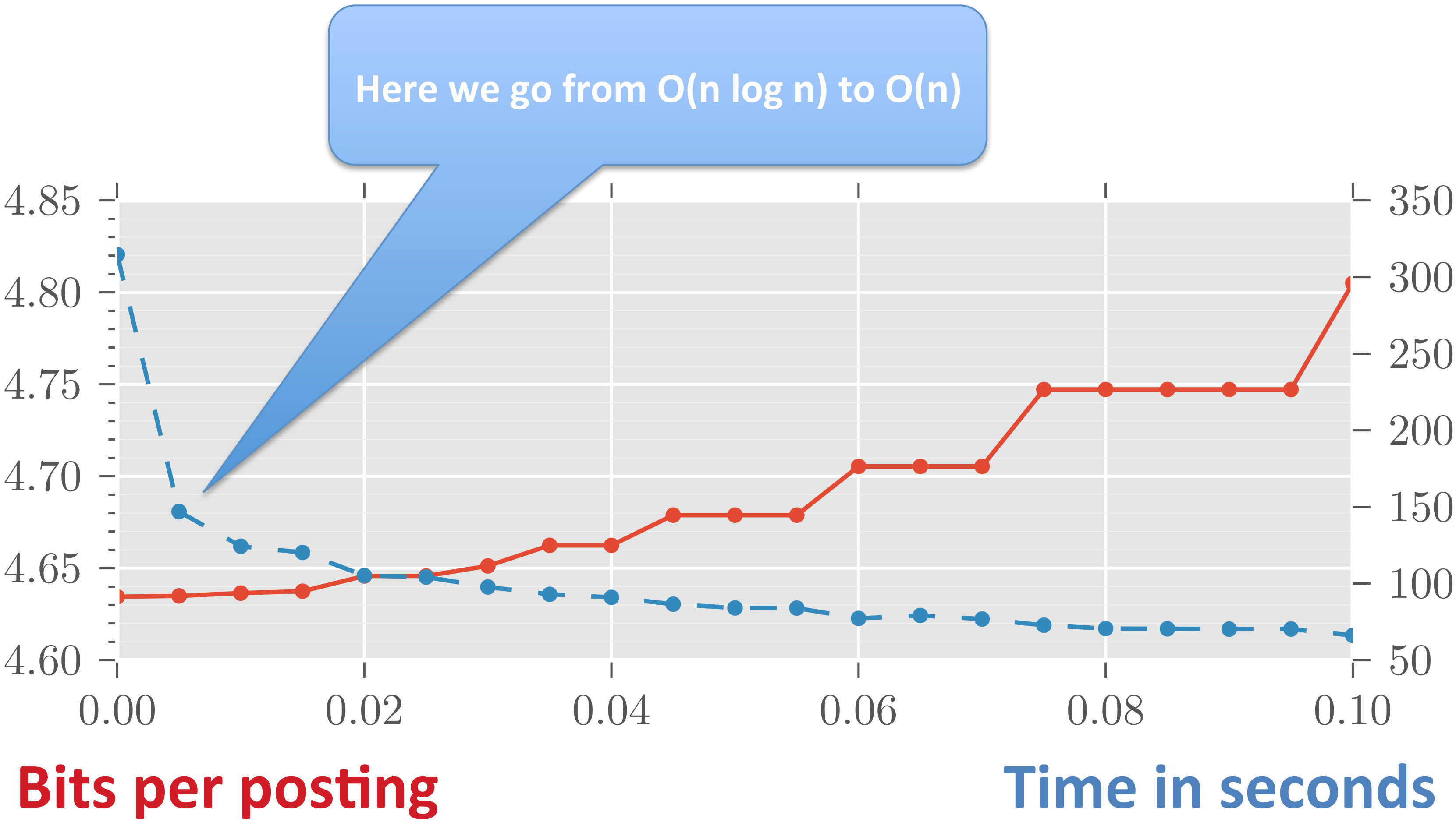
Dependency on ε_1



Bits per posting

Time in seconds

Dependency on ε_2



Results on GOV2 and ClueWeb09

	Gov2			ClueWeb09		
	space GB	doc bpi	freq bpi	space GB	doc bpi	freq bpi
EF single	7.66 (+64.7%)	7.53 (+83.4%)	3.14 (+32.4%)	19.63 (+23.1%)	7.46 (+27.7%)	2.44 (+11.0%)
EF uniform	5.17 (+11.2%)	4.63 (+12.9%)	2.58 (+8.4%)	17.78 (+11.5%)	6.58 (+12.6%)	2.39 (+8.8%)
EF ϵ-optimal	4.65	4.10	2.38	15.94	5.85	2.20

	Gov2		ClueWeb09	
	TREC 05	TREC 06	TREC 05	TREC 06
EF single	80.7 (+8%)	175.0 (+10%)	261.0 (+0%)	444.0 (−2%)
EF uniform	72.1 (−3%)	154.0 (−3%)	254.0 (−3%)	435.0 (−4%)
EF ϵ-optimal	74.5	159.0	261.0	451.0

	Gov2		ClueWeb09	
	TREC 05	TREC 06	TREC 05	TREC 06
EF single	2.1 (+10%)	4.7 (+1%)	13.6 (−5%)	15.8 (−9%)
EF uniform	2.1 (+9%)	5.1 (+10%)	15.5 (+8%)	18.9 (+9%)
EF ϵ-optimal	1.9	4.6	14.3	17.4

OR queries

AND queries

Thanks for your attention!

Questions?