Index Compression

Dr. Subrat Kumar Nayak
Associate Professor
Dept. of CSE, ITER, SOADU

What to be Discussed?

BRUTUS	⟶	1	2	4	11	31	45	173	174	
CAESAR	⟶	1	2	4	5	6	16	57	132	...
CALPURNIA	⟶	2	31	54	101					

- Collection statistics in more detail (with RCV1)
  - How big will the dictionary and postings be?
- Dictionary compression
- Postings compression

Why compression (in general)?

- Use less disk space
  - Saves a little money
- Keep more stuff in memory
  - Increases speed
- Increase speed of data transfer from disk to memory
  - [read compressed data | decompress] is faster than [read uncompressed data]
  - Premise: Decompression algorithms are fast
    - True of the decompression algorithms we use

Why compression for inverted indexes?

- Dictionary
  - Make it small enough to keep in main memory
  - Make it so small that you can keep some postings lists in main memory too
- Postings file(s)
  - Reduce disk space needed
  - Decrease time needed to read postings lists from disk
  - Large search engines keep a significant part of the postings in memory.
    - Compression lets you keep more in memory
- We will devise various IR-specific compression schemes

Recall Reuters RCV1

symbol	statistic	value
N	documents	800,000
L	avg. # tokens per doc	200
M	terms (= word types)	~400,000
	avg. # bytes per token (incl. spaces/punct.)	6
	avg. # bytes per token (without spaces/punct.)	4.5
	avg. # bytes per term	7.5
	non-positional postings	100,000,000

Index parameters vs. what we index

size of	word types (terms)			non-positional postings			positional postings		
	dictionary			non-positional index			positional index		
	Size (K)	∆%	cumul %	Size (K)	∆ %	cumul %	Size (K)	∆ %	cumul %
Unfiltered	484			109,971			197,879		
No numbers	474	-2	-2	100,680	-8	-8	179,158	-9	-9
Case folding	392	-17	-19	96,969	-3	-12	179,158	0	-9
30 stopwords	391	-0	-19	83,390	-14	-24	121,858	-31	-38
150 stopwords	391	-0	-19	67,002	-30	-39	94,517	-47	-52
stemming	322	-17	-33	63,812	-4	-42	94,517	0	-52

Lossless vs. lossy compression

- Lossless compression: All information is preserved.
  - What we mostly do in IR.
- Lossy compression: Discard some information
- Several of the preprocessing steps can be viewed as lossy compression: case folding, stop words, stemming, number elimination.
- Chap/Lecture 7: Prune postings entries that are unlikely to turn up in the top *k* list for any query.
  - Almost no loss quality for top *k* list.

Vocabulary vs. collection size

- How big is the term vocabulary?
  - That is, how many distinct words are there?
- Can we assume an upper bound?
  - Not really: At least $70^{20} = 10^{37}$ different words of length 20
- In practice, the vocabulary will keep growing with the collection size
  - Especially with Unicode ☺

Vocabulary vs. collection size

- Heaps' law: $M = kT^b$
- M is the size of the vocabulary, T is the number of tokens in the collection
- Typical values: $30 \leq k \leq 100$ and $b \approx 0.5$
- In a log-log plot of vocabulary size M vs. T, Heaps' law predicts a line with slope about ½
  - It is the simplest possible relationship between the two in log-log space
  - An empirical finding ("empirical law")

Heaps' Law

- For RCV1, the dashed line
- $\log_{10}M = 0.49 \log_{10}T + 1.64$ is the best least squares fit.
- Thus, $M = 10^{1.64}T^{0.49}$ so $k = 10^{1.64} \approx 44$ and $b = 0.49$.
- Good empirical fit for Reuters RCV1 !
- For first 1,000,020 tokens,
- law predicts 38,323 terms;
- actually, 38,365 terms

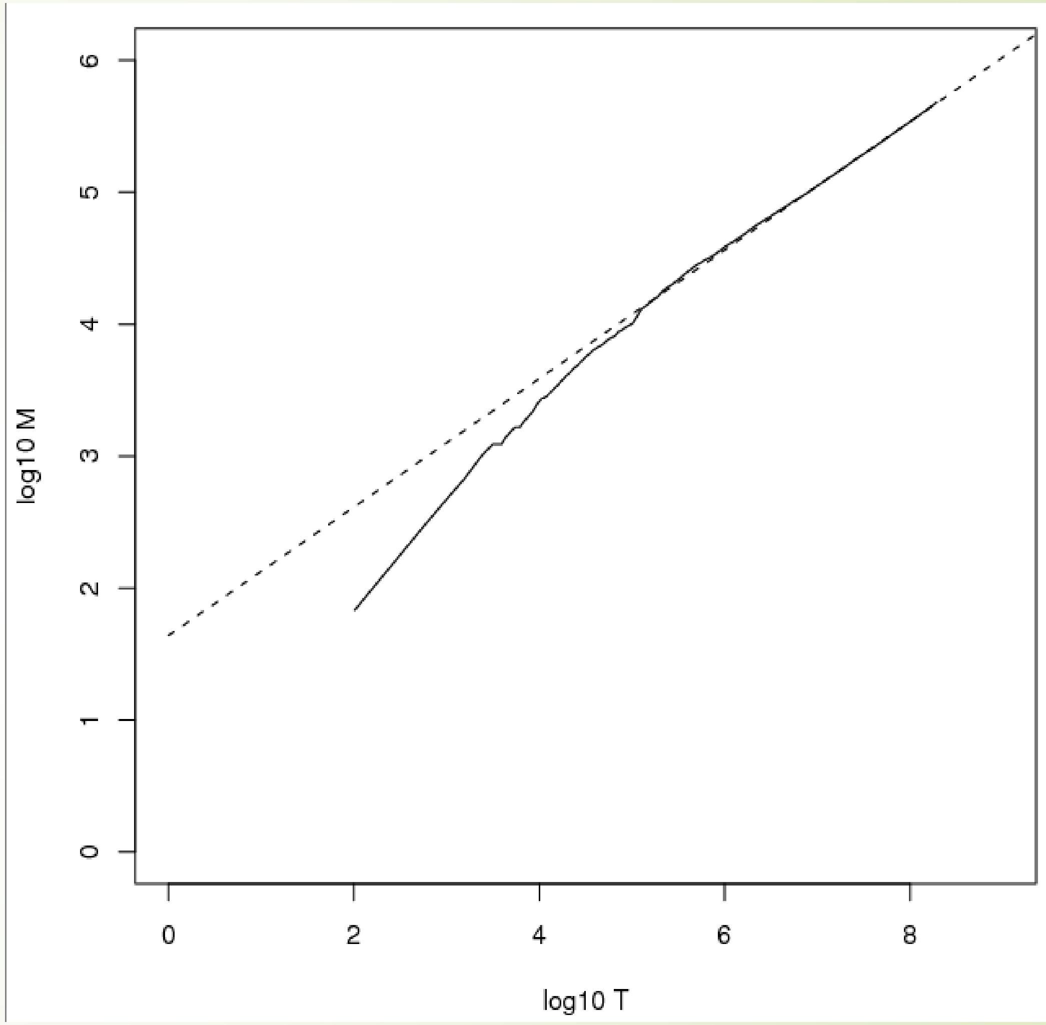


Fig 5.1 p81

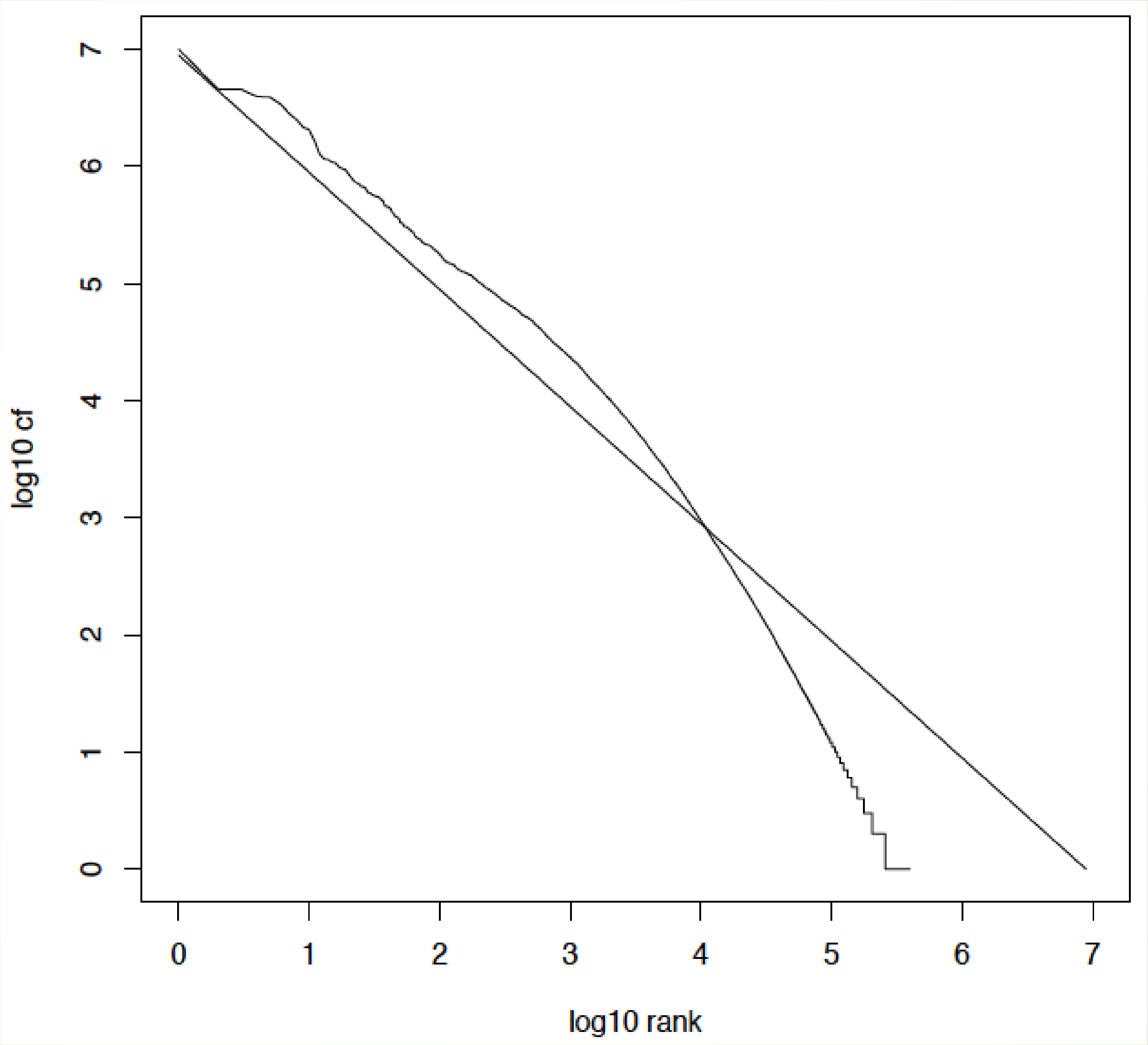
Zipf's law

- Heaps' law gives the vocabulary size in collections.
- We also study the relative frequencies of terms.
- In natural language, there are a few very frequent terms and very many very rare terms.
- Zipf's law: The *i*th most frequent term has frequency proportional to $1/i$.
- $cf_i \propto 1/i = K/i$ where K is a normalizing constant
- cf_i is <u>collection frequency</u>: the number of occurrences of the term t_i in the collection.

Zipf consequences

- If the most frequent term (*the*) occurs cf_1 times
  - then the second most frequent term (*of*) occurs $cf_1/2$ times
  - the third most frequent term (*and*) occurs $cf_1/3$ times …
- Equivalent: $cf_i = ci$^k, so
  - $\log cf_i = \log c + k \log i = \log c - \log i$
  - Linear relationship between $\log cf_i$ and $\log i$
- Another power law relationship

Zipf's law for Reuters RCV1



Compression

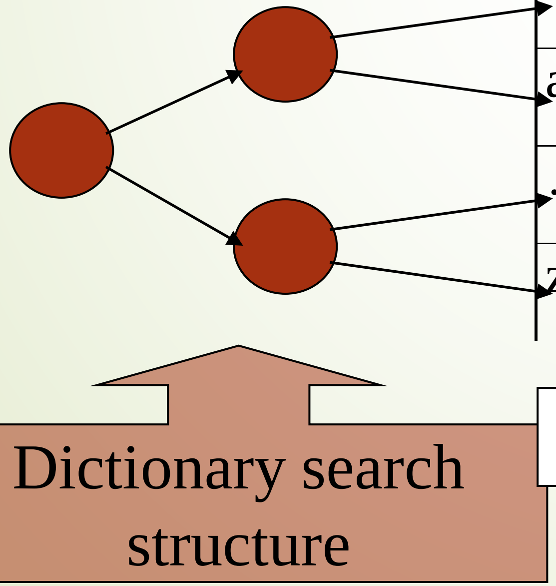
- Now, we will consider compressing the space for the dictionary and postings
  - Basic Boolean index only
  - No study of positional indexes, etc.
  - We will consider compression schemes

Why compress the dictionary?

- Search begins with the dictionary
- We want to keep it in memory
- Memory footprint competition with other applications
- Embedded/mobile devices may have very little memory
- Even if the dictionary isn't in memory, we want it to be small for a fast search startup time
- So, compressing the dictionary is important

Dictionary storage - first cut

- Array of fixed-width entries
  - ~400,000 terms; 28 bytes/term = 11.2 MB.



Terms	Freq.	Postings ptr.
a	656,265	
aachen	65	
….	….	
zulu	221	

20 bytes

4 bytes each

Fixed-width terms are wasteful

- Most of the bytes in the **Term** column are wasted – we allot 20 bytes for 1 letter terms.
  - And we still can't handle *supercalifragilisticexpialidocious* or *hydrochlorofluorocarbons.*
- Written English averages ~4.5 characters/word.
  - Exercise: Why is/isn't this the number to use for estimating the dictionary size?
- Ave. dictionary word in English: ~8 characters
  - How do we use ~8 characters per dictionary term?
- Short words dominate token counts but not type average.

Compressing the term list: Dictionary-as-a-String

- Store dictionary as a (long) string of characters:
  - Pointer to next word shows end of current word
  - Hope to save up to 60% of dictionary space.

….systilesyzygeticsyzygialsyzygyszaibelyiteszczecinszomo….

Freq.	Postings ptr.	Term ptr.
33		
29		
44		
126		

Total string length = 400K x 8B = 3.2MB

Pointers resolve 3.2M positions: $\log_2 3.2M$ = 22bits = 3bytes

Space for dictionary as a string

- 4 bytes per term for Freq.
- 4 bytes per term for pointer to Postings.
- 3 bytes per term pointer
- Avg. 8 bytes per term in term string
- 400K terms x 19 $\Rightarrow$ 7.6 MB (against 11.2MB for fixed width)

Now avg. 11 bytes/term, not 20.

Blocking

- Store pointers to every *k*th term string.
  - Example below: *k*=4.
- Need to store term lengths (1 extra byte)

….**7*systile*9*syzygetic*8*syzygial*6*syzygy*11*szaibelyite*8*szczecin*9*szomo***….

Freq.	Postings ptr.	Term ptr.
33		
29		
44		
126		
7		

Save 9 bytes on 3 pointers.

Lose 4 bytes on term lengths.

Net

- Example for block size $k = 4$
- Where we used 3 bytes/pointer without blocking
  - 3 x 4 = 12 bytes,

now we use 3 + 4 = 7 bytes.

Shaved another ~0.5MB. This reduces the size of the dictionary from 7.6 MB to 7.1 MB.
We can save more with larger k.

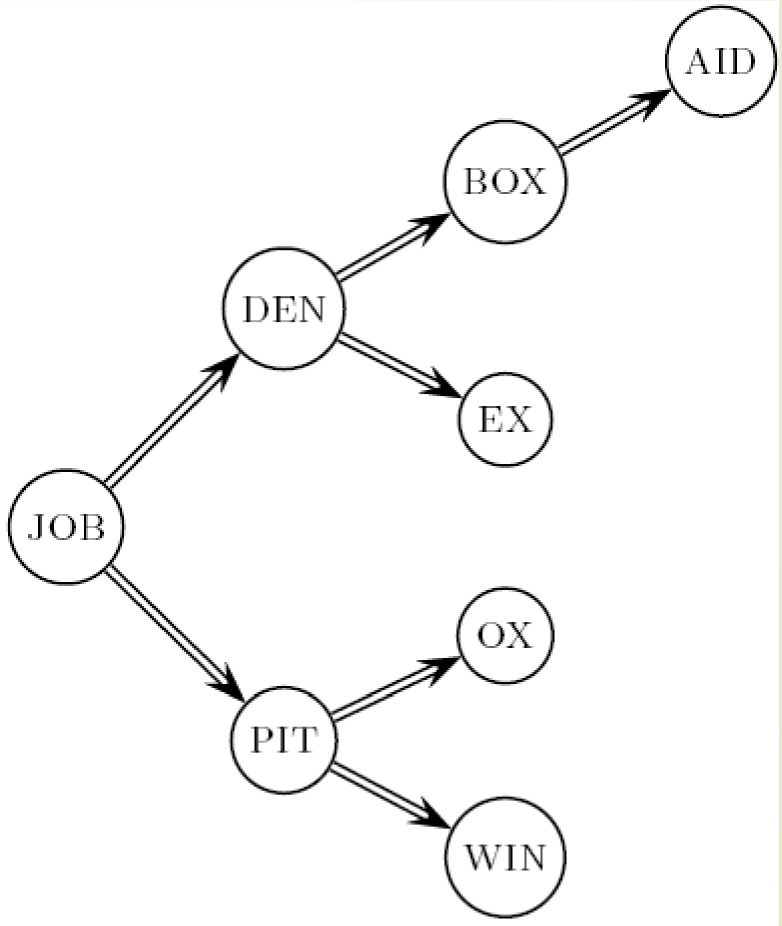
Why not go with larger k?

Exercise

- Estimate the space usage (and savings compared to 7.6 MB) with blocking, for block sizes of $k = 4$, 8 and 16.

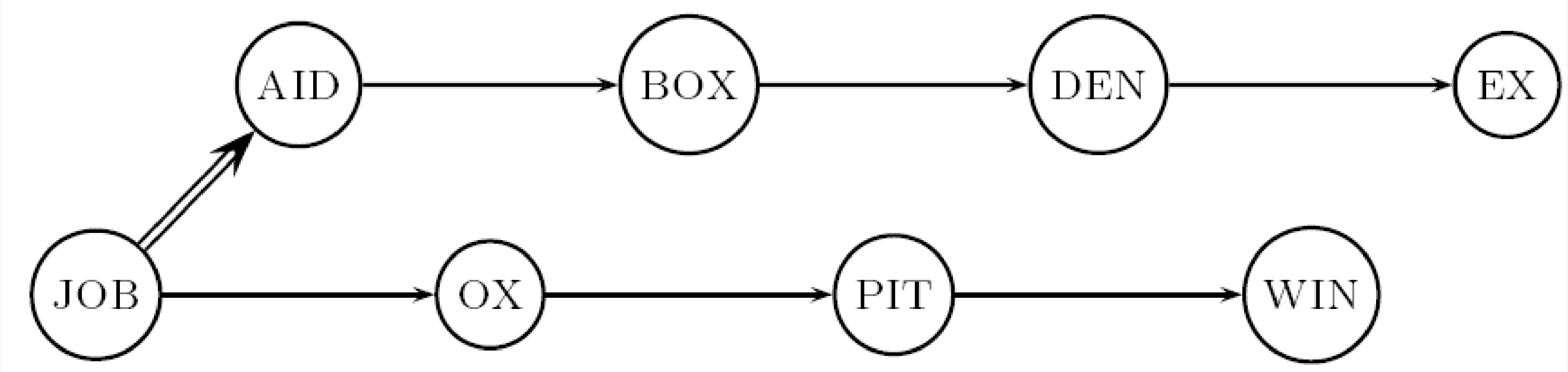
Dictionary search without blocking

- Assuming each dictionary term equally likely in query (not really so in practice!), average number of comparisons = (1+2*2+4*3+4)/8 ~2.6



Exercise: what if the frequencies of query terms were non-uniform but known, how would you structure the dictionary search tree?

Dictionary search with blocking



- Binary search down to 4-term block;
  - Then linear search through terms in block.
- Blocks of 4 (binary tree), avg. = (1+2·2+2·3+2·4+5)/8 = 3 compares

Exercise

Estimate the impact on search performance (and slowdown compared to k=1) with blocking, for block sizes of k = *4*, *8* and *16*.

Front coding

- Front-coding:
  - Sorted words commonly have long common prefix – store differences only
  - (for last *k-1* in a block of *k*)

8***automata***8***automate***9***automatic***10***automation***

→8***automat****a*1◊*e*2◊*ic*3◊*ion*

Encodes ***automat***

Extra length beyond ***automat.***

Begins to resemble general string compression.

RCV1 dictionary compression summary

Technique	Size in MB
Fixed width	11.2
Dictionary-as-String with pointers to every term	7.6
Also, blocking k = 4	7.1
Also, Blocking + front coding	5.9