Self-Indexing Inverted Files for Fast Text Retrieval

by Alistair Moffat, Justin Zobel

Onur Taşar, Murat Yusuf Taze

Overview

- *Background Information*
- Query Processing – Boolean and Ranking
- Compression
- *Motivation*
- *Fast Inverted Index*
- *Skipping*
- *Implementation, Experimental Results*
- *Conclusion*

Indexes

- *Indexes* are data structures designed to make search faster
- Text search has unique requirements, which leads to unique data structures
- Most common data structure is *inverted index*
  - general name for a class of structures
  - “inverted” because documents are associated with words, rather than words with documents

Inverted Index

- Each index term is associated with an *inverted list*
  - Contains lists of documents, or lists of word occurrences in documents, and other information
  - Each entry is called a *posting*
  - The part of the posting that refers to a specific document or location is called a *pointer*
  - Each document in the collection is given a unique number
  - Lists are usually *document-ordered* (sorted by document number)

Example "Collection"

S_1 Tropical fish include fish found in tropical environments around the world, including both freshwater and salt water species.

S_2 Fishkeepers often use the term tropical fish to refer only those requiring fresh water, with saltwater tropical fish referred to as marine fish.

S_3 Tropical fish are popular aquarium fish, due to their often bright coloration.

S_4 In freshwater fish, this coloration typically derives from iridescence, while salt water fish are generally pigmented.

Four sentences from the Wikipedia entry for *tropical fish*

Example “Inverted Index”

Simple Inverted Index

and	1	only	2
aquarium	3	pigmented	4
are	3 4	popular	3
around	1	refer	2
as	2	referred	2
both	1	requiring	2
bright	3	salt	1 4
coloration	3 4	saltwater	2
derives	4	species	1
due	3	term	2
environments	1	the	1 2
fish	1 2 3 4	their	3
fishkeepers	2	this	4
found	1	those	2
fresh	2	to	2 3
freshwater	1 4	tropical	1 2 3
from	4	typically	4
generally	4	use	2
in	1 4	water	1 2 4
include	1	while	4
including	1	with	2
iridescence	4	world	1
marine	2		
often	2 3		

Example “Inverted Index”

Inverted Index with counts

- supports better ranking algorithms

and	1:1			
aquarium	3:1			
are	3:1	4:1		
around	1:1			
as	2:1			
both	1:1			
bright	3:1			
coloration	3:1	4:1		
derives	4:1			
due	3:1			
environments	1:1			
fish	1:2	2:3	3:2	4:2
fishkeepers	2:1			
found	1:1			
fresh	2:1			
freshwater	1:1	4:1		
from	4:1			
generally	4:1			
in	1:1	4:1		
include	1:1			
including	1:1			
iridescence	4:1			
marine	2:1			
often	2:1	3:1		
only	2:1			
pigmented	4:1			
popular	3:1			
refer	2:1			
referred	2:1			
requiring	2:1			
salt	1:1	4:1		
saltwater	2:1			
species	1:1			
term	2:1			
the	1:1	2:1		
their	3:1			
this	4:1			
those	2:1			
to	2:2	3:1		
tropical	1:2	2:2	3:1	
typically	4:1			
use	2:1			
water	1:1	2:1	4:1	
while	4:1			
with	2:1			
world	1:1			

Example "Inverted Index"

Inverted Index with positions

- supports proximity matches

and	1,15				
aquarium	3,5				
are	3,3	4,14			
around	1,9				
as	2,21				
both	1,13				
bright	3,11				
coloration	3,12	4,5			
derives	4,7				
due	3,7				
environments	1,8				
fish	1,2	1,4	2,7	2,18	2,23
			3,2	3,6	4,3
			4,13		
fishkeepers	2,1				
found	1,5				
fresh	2,13				
freshwater	1,14	4,2			
from	4,8				
generally	4,15				
in	1,6	4,1			
include	1,3				
including	1,12				
iridescence	4,9				

marine	2,22				
often	2,2	3,10			
only	2,10				
pigmented	4,16				
popular	3,4				
refer	2,9				
referred	2,19				
requiring	2,12				
salt	1,16	4,11			
saltwater	2,16				
species	1,18				
term	2,5				
the	1,10	2,4			
their	3,9				
this	4,4				
those	2,11				
to	2,8	2,20	3,8		
tropical	1,1	1,7	2,6	2,17	3,1
typically	4,6				
use	2,3				
water	1,17	2,14	4,12		
while	4,10				
with	2,15				
world	1,11				

Information Retrieval

- Two main mechanisms for retrieving documents
  - Boolean Queries
    - a set of query terms connected by the logical operators AND, OR, and NOT
  - Range Queries
    - matching an informal query to the documents
    - allocating scores to documents according to their degree of similarity to the query

Query Processing

- inverted lists are read from disk
- the lists are merged,
- taking the intersection of the sets of document numbers for AND operations, the union for OR, and the complement for NOT

Example

$$
\begin{aligned}
I_{\text{“index”}} &= \langle 5, 8, 12, 13, 15, 18, 23, 28, 29, 40, 60\rangle \\
I_{\text{“compression”}} &= \langle 10, 11, 12, 13, 28, 29, 30, 36, 60, 62, 70\rangle \\
I_{\text{“algorithm”}} &= \langle 13, 44, 48, 51, 55, 60, 93\rangle \ ,
\end{aligned}
$$

- their conjunction are documents 13 and 60
  - Terms are connected by AND operator.

Ranking vs Boolean

- More memory is required because in a ranked query there are usually many candidates
  - In a conjunctive Boolean query the answers lie in the intersection of the inverted lists, but in a ranked query, they lie in the union
  - In a conjunctive Boolean query, the number of candidates need never be greater than the frequency of the least common query term
- More time is required because conjunctive Boolean queries typically have a small number of terms, perhaps 3–10, whereas ranked queries usually have far more

Compression

- for space efficiency, the inverted lists are stored compressed
  - For example, the list
  - 5, 8, 12, 13, 15, 18, 23, 28, 29, 40, 60
  - corresponding d-gaps:
  - 5, 3, 4, 1, 2, 3, 5, 5, 1, 11, 20 (good for variable-length encoding)
- Without compression, an inverted file can easily be as large or larger than the text it indexes

Compression

- Advantage
  - net space reduction of as much as 80% of the inverted file size
- Disadvantage
  - even with fast decompression it involves a substantial overhead on processing time

Motivation

- Problem: How to reduce these space and time costs if we compress indexes.
- Solution: A mechanism called ***Self-Indexing***
- For typical conjunctive Boolean queries processing time is reduced by a factor of about five.
- the overhead in terms of storage space is small, typically under 25% of the inverted file, or less than 5% of the complete stored retrieval system

FAST INVERTED FILE PROCESSING

Skipping

Consider the set of $\langle d, f_{d,t} \rangle$

- <5, 1><8, 1><12, 2><13, 3><15, 1><18, 1>...

- Stored as d-gaps:
- <5, 1><3, 1><4, 2><1, 3><2, 1><3, 1>...

Skipping continued

Synchronization points

Skip over every three pointers:

- <<5, a2>><5, 1><3, 1><4, 2><<13,a3>><1,3> <2,1> <3,1>...
- Still redundancy, code differently:
- <<5, a2>><1><3, 1><4, 2><<8, a3-a2>><3> <2,1><3,1>...
- Find the correct block

Implementation

Storage

Let L be the value of k

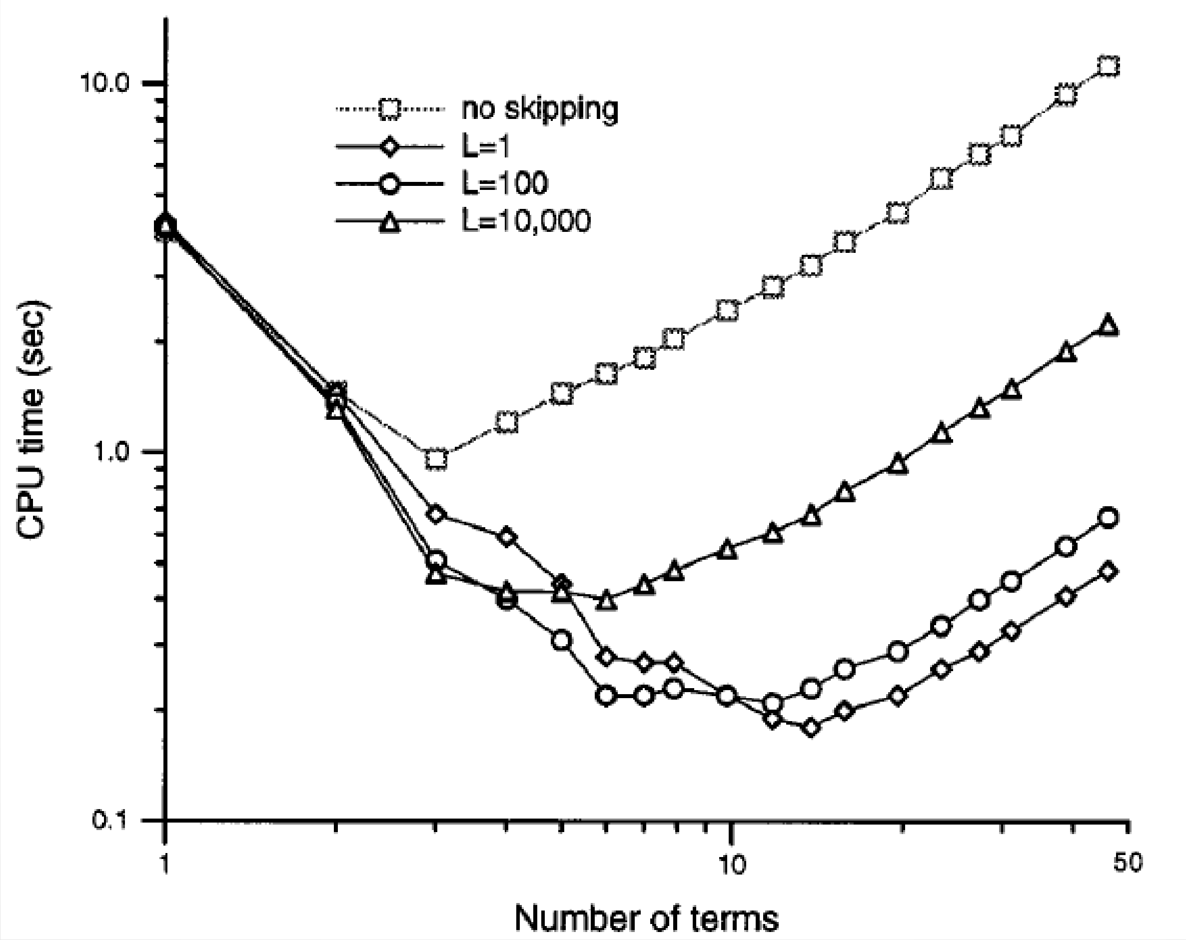
Size of skipped inverted files for a dataset becomes:

Size of Skipped Inverted Files

Parameter	Size MB	Size %
No Skipping	184.36	100
$L = 1$	186.14	101
$L = 10$	188.95	102
$L = 100$	194.74	106
$L = 1{,}000$	205.38	111
$L = 10{,}000$	220.33	120
$L = 100{,}000$	230.21	125

Implementation

Performance on Boolean Queries



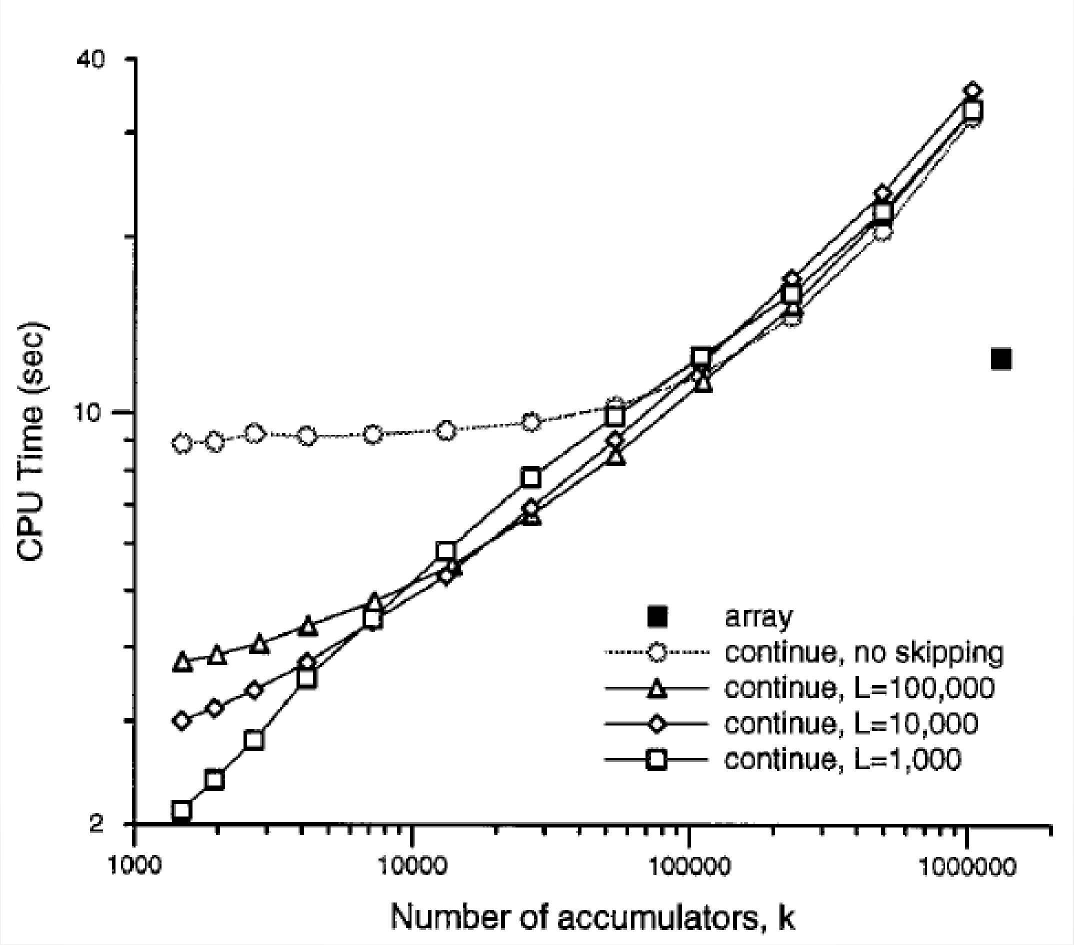
Implementation

Ranked Queries

- Any document containing any of the terms is considered as a candidate.
- We need to restrict the number of accumulators
- Two algorithms:
- Quit
- Continue

Experimental Result

Top 200 documents are returned



Conclusions

Advantages:

- CPU time is reduced
- Only compressing the pointers save the space but increase the processing time
- The idea can be applied to both the boolean queries and the ranked queries

References

- Addison Wesley, 2008
- G. Salton. Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer. Addison-Wesley, Reading, Massachusetts, 1989.
- G. Salton and M.J. McGill. Introduction to Modern Information Retrieval. McGraw-Hill, New York, 1983.