NYU | TANDON SCHOOL OF ENGINEERING

Exploring Size-Speed Trade-Offs in Static Index Pruning

Juan Rodriguez **Torsten Suel**

Tandon School of Engineering
New York University
Brooklyn, NY

Supported by NSF Grant IIS-1718680 and by Amazon.

Presentation and code available at:

https://github.com/**JC-R/BD2018**

Introduction

- Inverted Index: structure used in search engines
- For each word, we store where it is used
- Inverted list for term t contains index entries (postings)
- Postings have form (did, f) where did is the ID of a document containing t and f is the number of occ.

Introduction

term

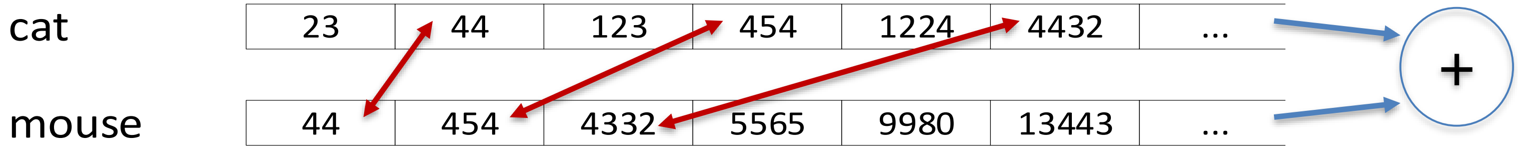
posting (inverted) list

posting

Inverted Index

aardvark	3425	3889	111232	234432			
⋮							
cat	23	44	123	454	1224	4432	...
cataract	1121	22365	33498	100223	122344	15001	...
catering	4434	9845	10554	54665	98001	203400	...
⋮							
mouse	44	454	4332	5565	9980	13443	...
⋮							
zebra	44	123	454	4432	6547	10012	...

Problem Definition



Query processing in a nutshell:

- **Traverse inverted lists for query terms**
- **Apply a ranking function to docs in union or intersection**
- **E.g., cosine, BM25, or more complex ranker**
- **Then return the top-*k* results to the user**

Problem Definition

cat	23	44	123	454	1224	4432	...
mouse	44	454	4332	5565	9980	13443	...

- Many inverted lists are very long
- We ***cannot afford*** to traverse and rank complete lists
- Search engines use shortcuts: “early termination techniques”:
  - Index tiering, impact ordering, maxscore/wand etc.
  - *Index pruning* is one early termination (ET) techn.

Index Pruning

"would anybody search for doc 454 using this term?"

"will this posting ever lead to a top-k result for a likely query?"

If "no", remove this posting from the index

aardvark	3425	3889	111232	234432			
			⋮				
cat	23	44	123	454	1224	4432	...
cataract	1121	22365	33498	100223	122344	15001	...
catering	4434	9845	10554	54665	98001	203400	...
			⋮				
mouse	44	~~454~~	4332	5565	9980	13443	...
			⋮				
zebra	44	123	454	4432	6547	10012	...

Motivation – Index Pruning

Goal: Remove postings that are (fairly) useless, while keeping quality "almost" the same

Why? Shorter lists lead to faster query processing

Smaller indexes: need less RAM or disk accesses

Smaller indexes: search on devices at the edge

Index pruning is a non-safe ET technique (lossy)

Motivation

- Suppose a search engine indexes 4 trillion documents
- 250 index entries per document → 10^15 entries
- 5B queries per day, with 3.4 terms on average
- < 170 billion index entries lead to top-10 in a month

    Actually, far less ...

- **Can we find and remove them from the index?**

Related Work

- **Introduced by Carmel et al (SIGIR 2001)**
- **A lot of subsequent work**
- **Three main approaches:**
    - Algorithmic heuristics: based on impact, rank
    - Hit-based methods: based on past top results
    - Curation-based: build a high-quality subset
- **Not the same as spam detection/removal**

Our Work

1. **Optimize index quality, given a constraint on size**
   Given a bound S on index size, produce a pruned index I' of size at most S that maximizes the average quality of queries un distribution Q.
2. **Trade-off between index size and query cost**
   Given a lower bound S on index size, and a bound R on result quality, produce a pruned index I' that satisfies the constraints while minimizing query cost C.

Static Index Pruning

- posting $p = (d, f)$ is part of a top-k result for query q if document d is among the top-k results for q.

- $\Pr(p \in top\text{-}k)$ is the probability that p is part of a top-k result for query q.

- $\Pr(p \in q)$ is the probability the term associated with p occurs in query q.

- $\Pr(p \in top\text{-}k) = \Pr(p \in q) \cdot \Pr(p \in top\text{-}k | p \in q)$

Static Index Pruning

1. Optimize index quality, given a constraint on size

Approach:

- Estimate $Pr(p \in q)$
- Machine learn $Pr(p \in top\text{-}k | p \in q)$
- Predict $Pr(p \in top - k | p \in q)$ for all postings

Prune Policy

- Global order on prediction, select postings until desired size is achieved

Estimating $\boldsymbol{Pr(p \in q)}$

- Language model from $100k$ TREC 01-09 and 2002-2009 relevance and ad-hoc track queries.

- Interpolated with 5M random-sampled pages from the corpus.

- Unigram as $\Pr(p \in q)$.

Machine learning: $Pr(p \in top\text{-}k | p \in q)$

Family	Feature	Description
term	tf	term frequency in the corpus
	tl	term list size
	p_bm25	partial BM25 score
	$Pr[p \in q]$	probability of posting in a random query
document	docSize	# words in document
	docTerms	# unique terms in document
	xdoc	promise of document
dochits	bins 1 - 17	dochit ranges, quantized by rank
	top10	top10 dochits from language model queries
	top1k	top1k dochits from language model queries
posthits	bins 1 - 17	posthit ranges, quantized by rank
	top10	top10 posthits from language model queries
	top1k	top1k posthits from language model queries

- **Random Forest**
- **Training size: 1M**

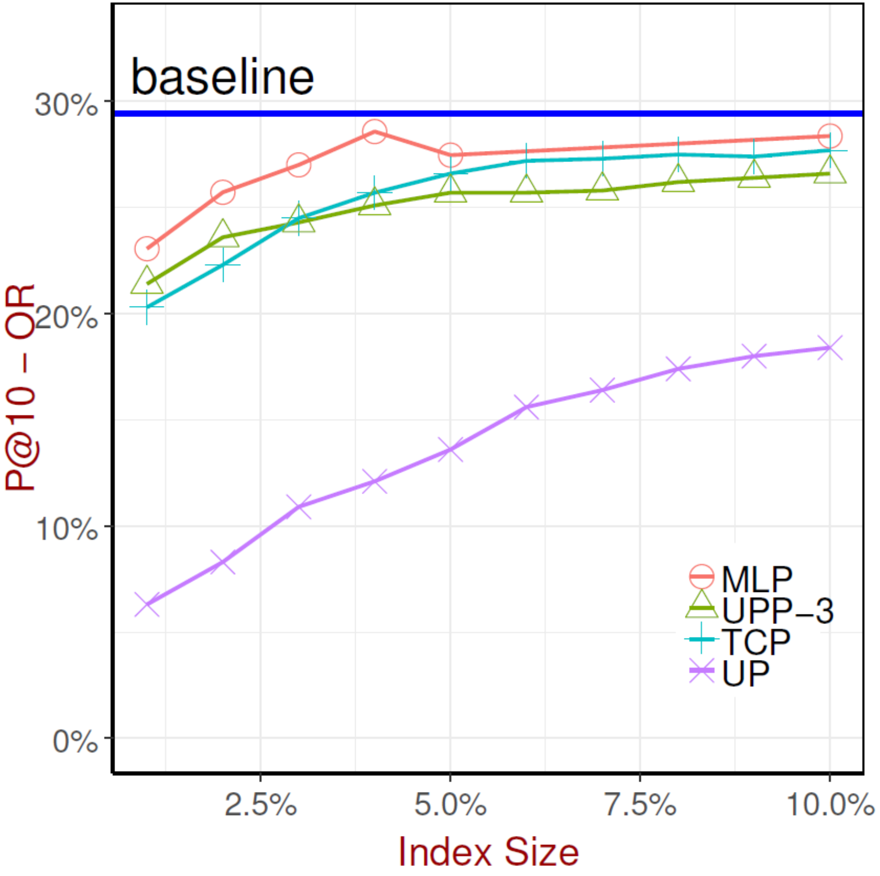
- **Predict on entire index**
  16 billion postings

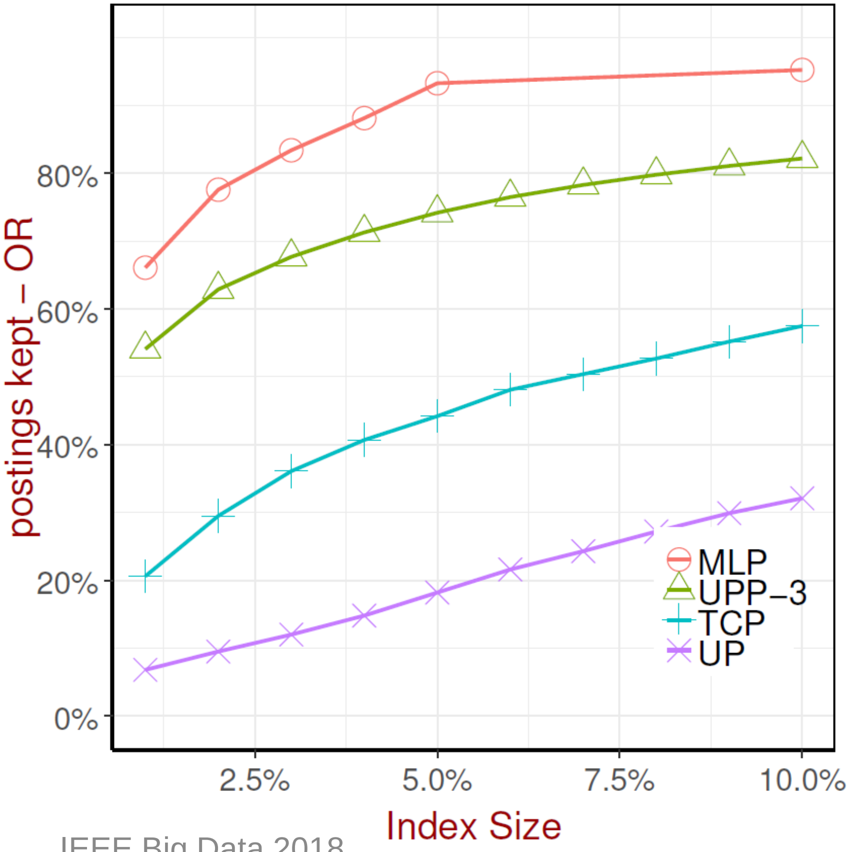
- **Spark cluster on Hadoop**
  1024 Vcores
  3Tb RAM
- **200 Tb HD**

Experimental Results

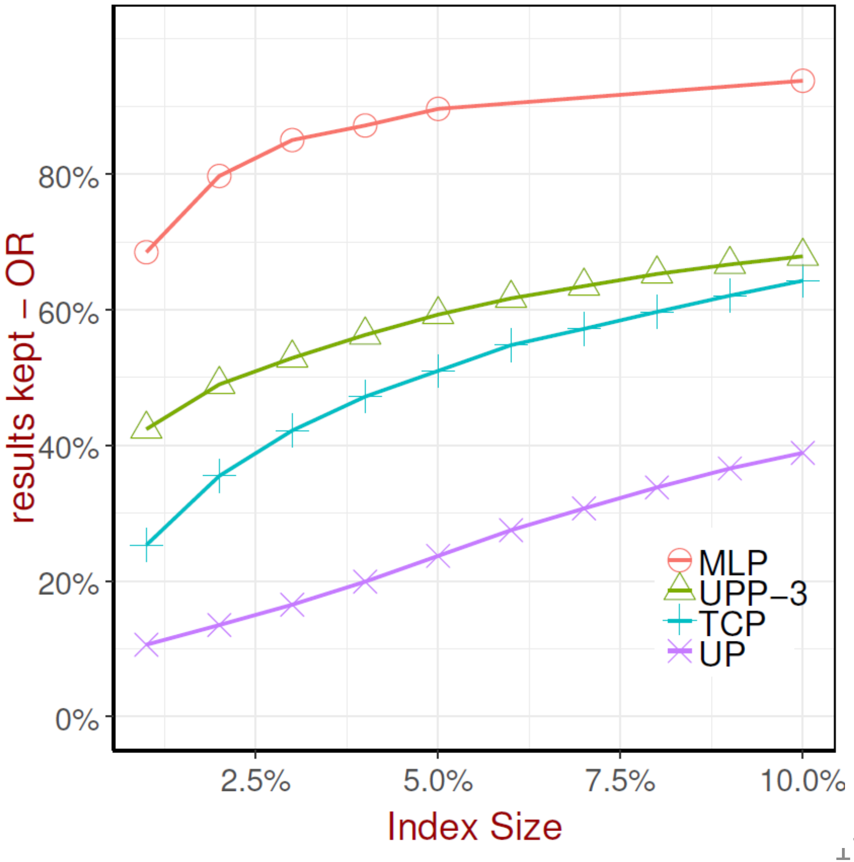
Machine Learned Pruning

- Very high pruning ratios
- Quality metrics
  P@10
  Postings Kept*
  Results Kept*





80%
60%
40%
20%
0%
postings kept – OR
2.5%
5.0%
7.5%
10.0%
Index Size
MLP
UPP–3
TCP
UP



80%
60%
40%
20%
0%
results kept – OR
2.5%
5.0%
7.5%
10.0%
Index Size
MLP
UPP–3
TCP
UP

2. Trading-Off between Index Size and Query Cost

- *Benefit* of a posting:

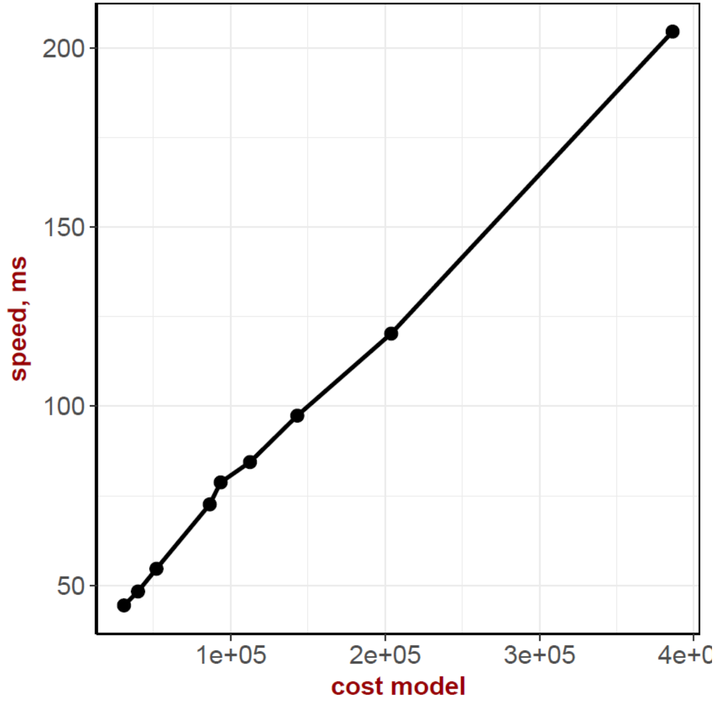
$$Pr(p \in to\text{-}k | p \in q)$$

- Query Cost: A simple model = $\sum$ (length of lists for the terms in disjunctive query q)
  A posting in this model has an *Expected Cost* proportional to:

$$Pr(p \in q)$$

Cost Model Validation

- Actual query speed measurements show near linear relationship to our cost model!



Size Cost Trade-Off

Goal: Given a lower bound S on index size, and a bound R on result quality, produce a pruned index I' that satisfies the constraints while minimizing query cost C.

Benefit: $\boldsymbol{Pr(p \in to-k|p \in q)}$

Cost: $\boldsymbol{Pr(p \in q)}$

Size Cost Trade-Off

LP formulation:

$$min(\sum_{i=1}^{n} x_i \cdot c_i)$$

$$(\sum_{i=1}^{n} b_i) \geq B$$

$$\sum_{i=1}^{n} i = S + \Delta$$

x_i = posting i
b_i = benefit for x_i
c_i = cost for x_i
B = total benefit
S = size
Δ = size increment

Size Cost Trade-Off

- Cannot express an LP formula for all postings (billions)
- We quantize all postings using simple logarithmic quantization
- Assign each posting into one of k classes, $k = 1000$ in our work
- Result in 1 million variables, easily solvable and fast

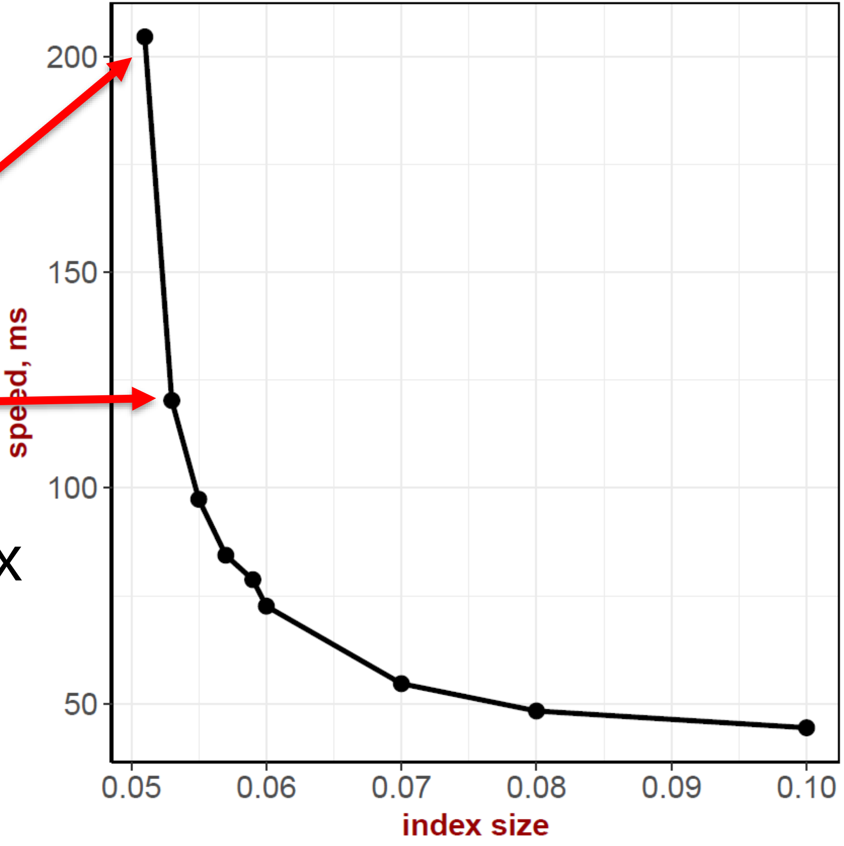
$$min(\sum^{1000} x_i \cdot c_i)$$

Size Cost Trade-Off Results

$S = 5\%$ of full index

$S = 5.5\%$ of full index

$Q' \geq Q$ @ 5% of full index



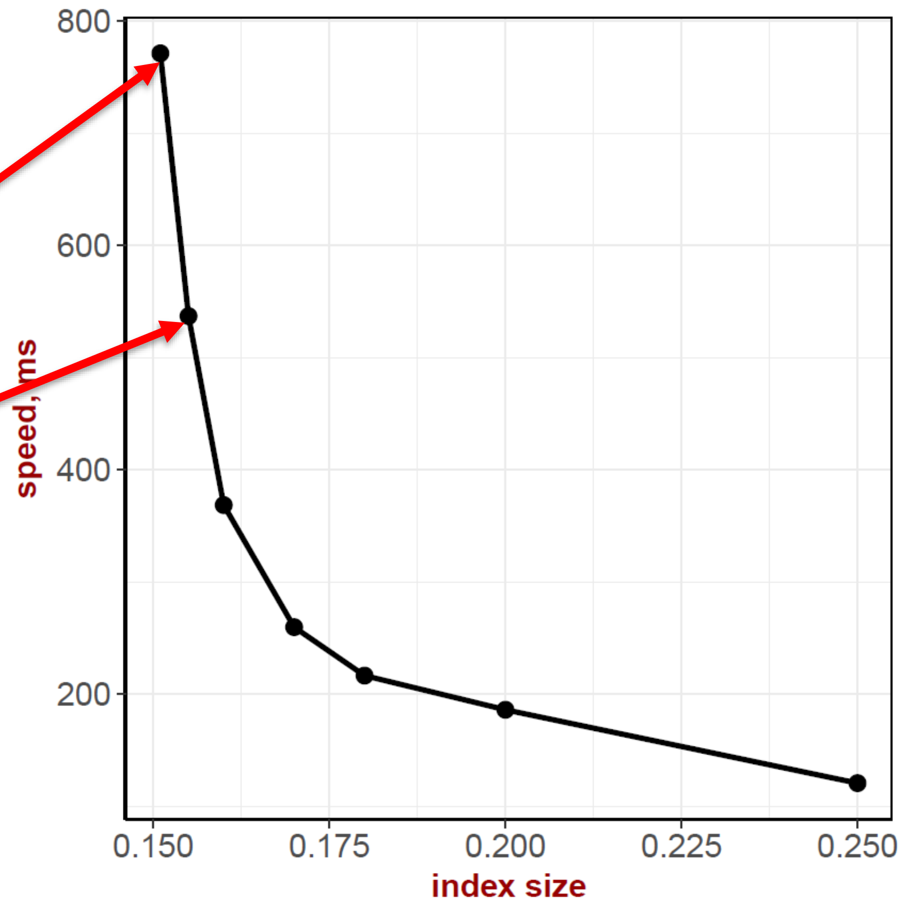
NYU TANDON SCHOOL OF ENGINEERING

Size Cost Trade-Off Results

$S = 15\%$ of full index

$S = 15.5\%$ of full index

$Q' \geq Q$ @ 15% of full index

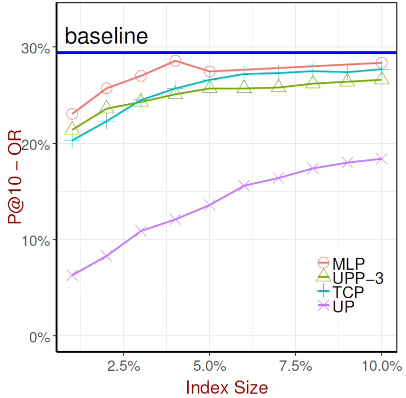


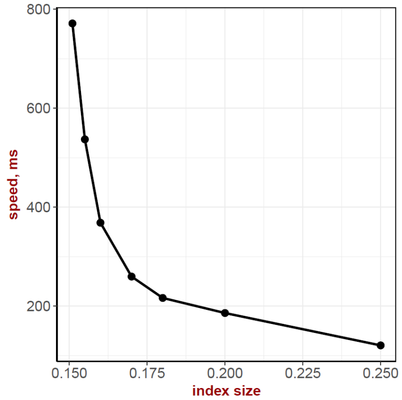
Summary

1. Learning to prune: optimize for size.

***2. Can significantly* increase query processing speed if we allow a very small increment in index size**

- **1 + 2 = optimal size/cost at any index pruning size**





Open Questions

- **We learned to optimize size for a simple ranking function, BM25.**

- **Need to explore learning to prune under complex ranking functions**

Thanks for listening!

Questions?