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.

Paper, slides and code:

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

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 useless entries from the index?**

Introduction

Inverted Index

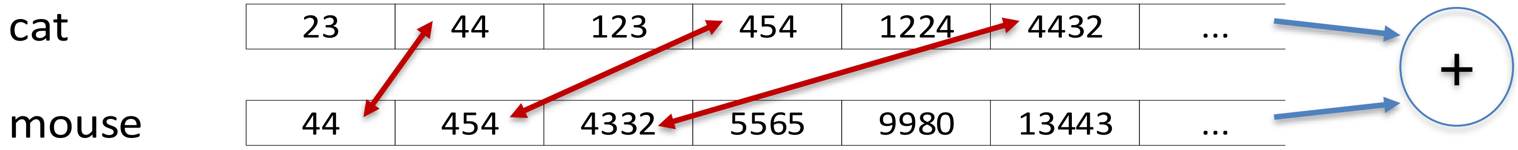
term

posting (inverted) list

posting

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	...

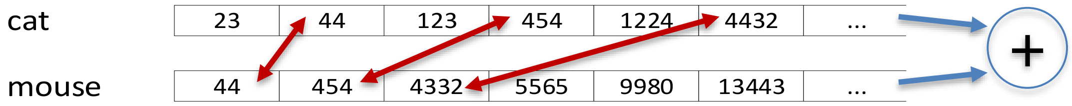
Problem Definition



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



- Many inverted lists are very long, millions of entries long…
- We ***cannot afford*** to traverse and rank all candidates
- Search engines use shortcuts: “early termination techniques”:
  - Index tiering, impact ordering, maxscore/wand etc.
  - *Index pruning* is one early such 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	...

Goal: Remove postings that are (fairly) useless, while keeping quality “almost” the same; make query processing as fast as possible.

Why? Shorter lists lead to faster query processing

Smaller indexes need less RAM or disk accesses

Smaller indexes: move search to devices at the edge

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

Not the same as spam detection/removal

Our Objectives

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 under 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.

Definitions used in this paper

- posting $\boldsymbol{p} = (\boldsymbol{t}, \boldsymbol{d})$ is part of a $top\text{-}k$ result for query q if document d is among the $top\text{-}k$ results for q.

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

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

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

Objective 1. Optimize index quality, given a constraint on size

Approach:

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

Prune Policy

- Select postings based on the global order of predicted values until a 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.

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

- Use the unigram probability 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: 2M**

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

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

Experimental Results

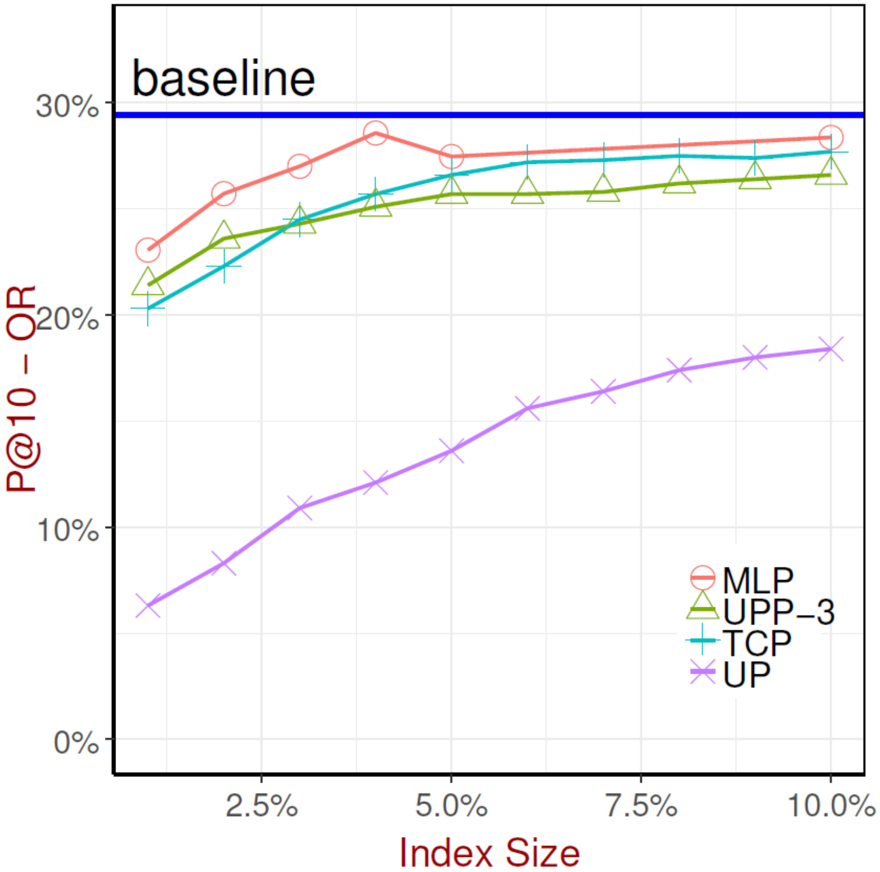
Machine Learned Pruning

- Very high pruning ratios
- Quality metrics

  P@10

  Postings Kept*

  Results Kept*



Objective 2. Trade-Off between Index Size and Query Cost

Benefit of a posting:

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

Cost of a posting: first define a simple model for query processing cost:

$$\sum(lenght\ of\ lists\ for\ disjunctive\ query\ q)$$

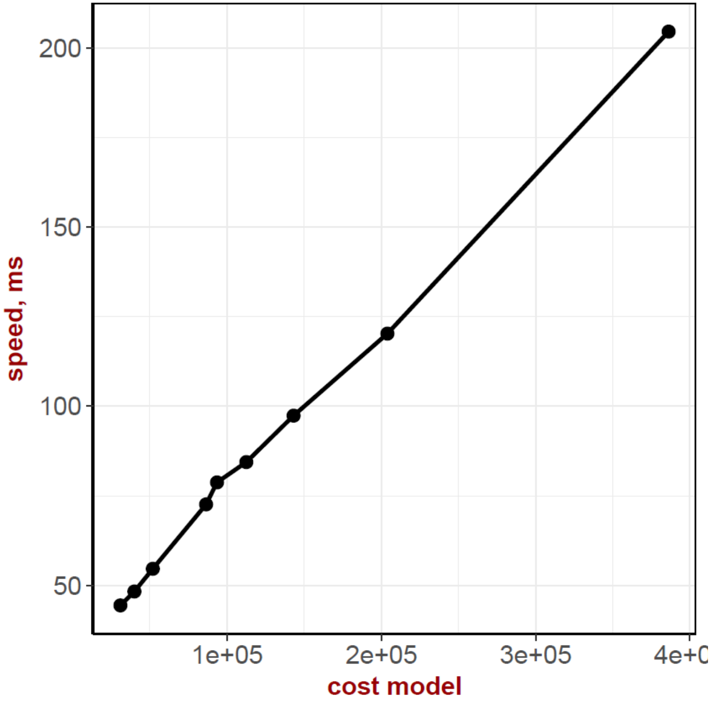
Cost of posting under this model:

$$Pr(p \in q)$$

Cost Model Validation

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

(Exhaustive OR evaluation)



NYU TANDON SCHOOL OF ENGINEERING

Size Cost Trade-Off

n = billions or more of entries

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

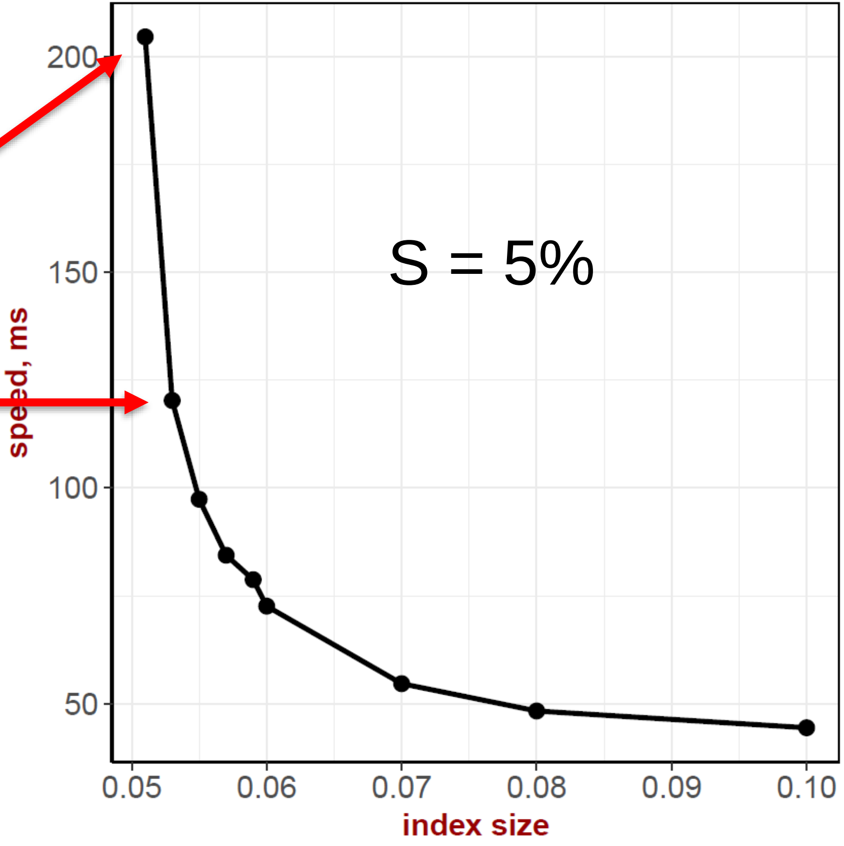
$$min(\sum^{1000} x_i \cdot c_i)$$

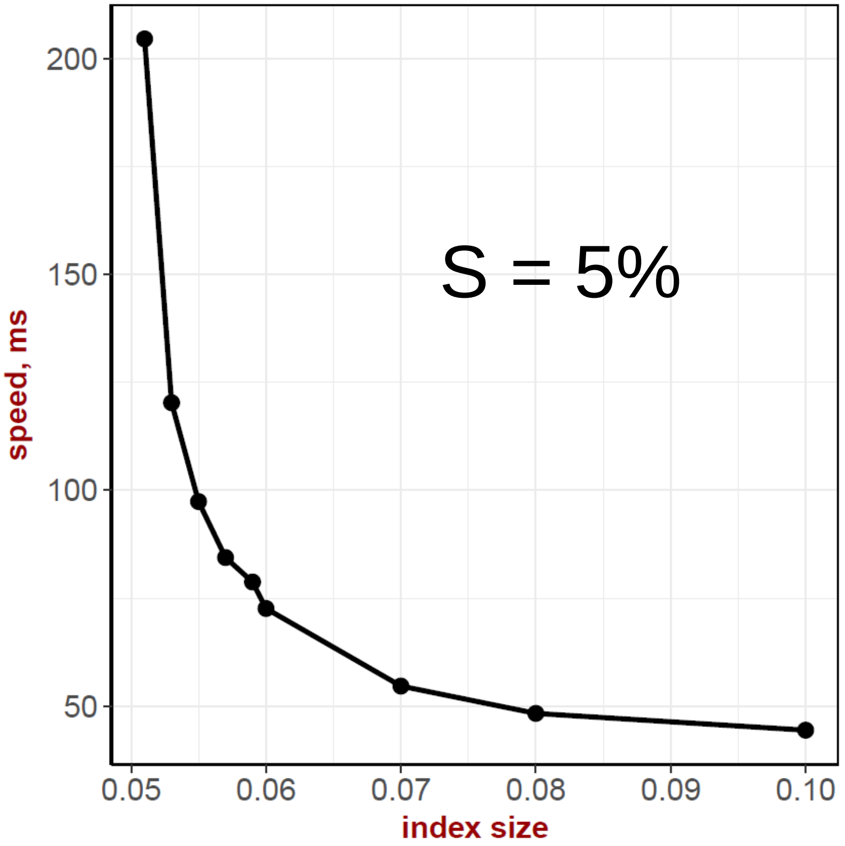
Size Cost Trade-Off Results

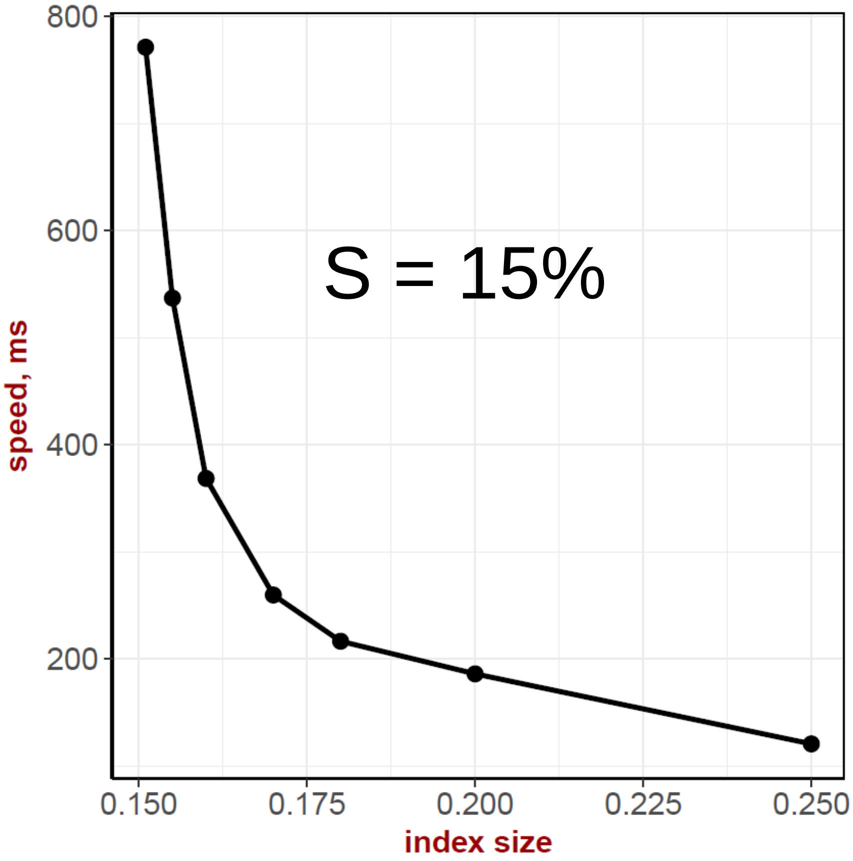
S = 5% of full index

S = 5.5% of full index

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




S = 5%
speed, ms
200
150
100
50
0.05
0.06
0.07
0.08
0.09
0.10
index size

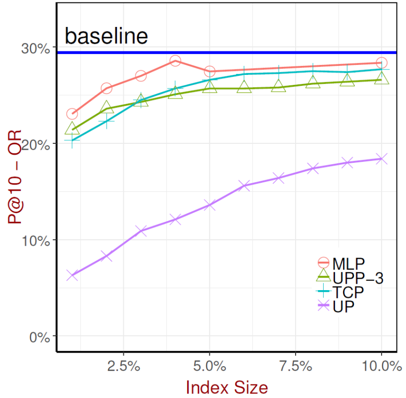

S = 15%
speed, ms
800
600
400
200
0.150
0.175
0.200
0.225
0.250
index size

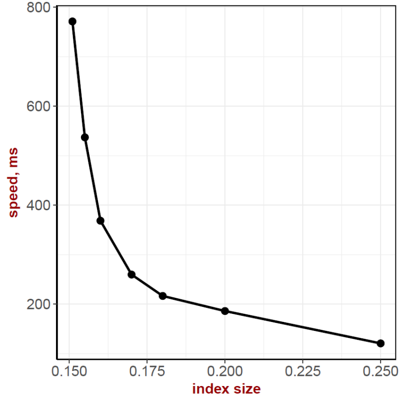
Summary

1. Learning to prune: optimize for size.

2. Significantly **increase query processing speed if we allow a very small increment in index size: optimize for speed**

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





NYU TANDON SCHOOL OF ENGINEERING

Thanks!

Questions?

Acknowledgements