Static Pruning of Terms In Inverted Files

Roi Blanco and Álvaro Barreiro

IRLab
University of A Corunna, Spain

29th European Conference on Information Retrieval, Rome, 2007

Motivation

- Pruning: to reduce inverted files size with lossy compression
- Static pruning of document pointers has already been addressed by other authors.
- We analyse static pruning of terms in inverted files:
  - Pruning % vs precision
  - Pruning % vs efficiency

Outline

Inverted files

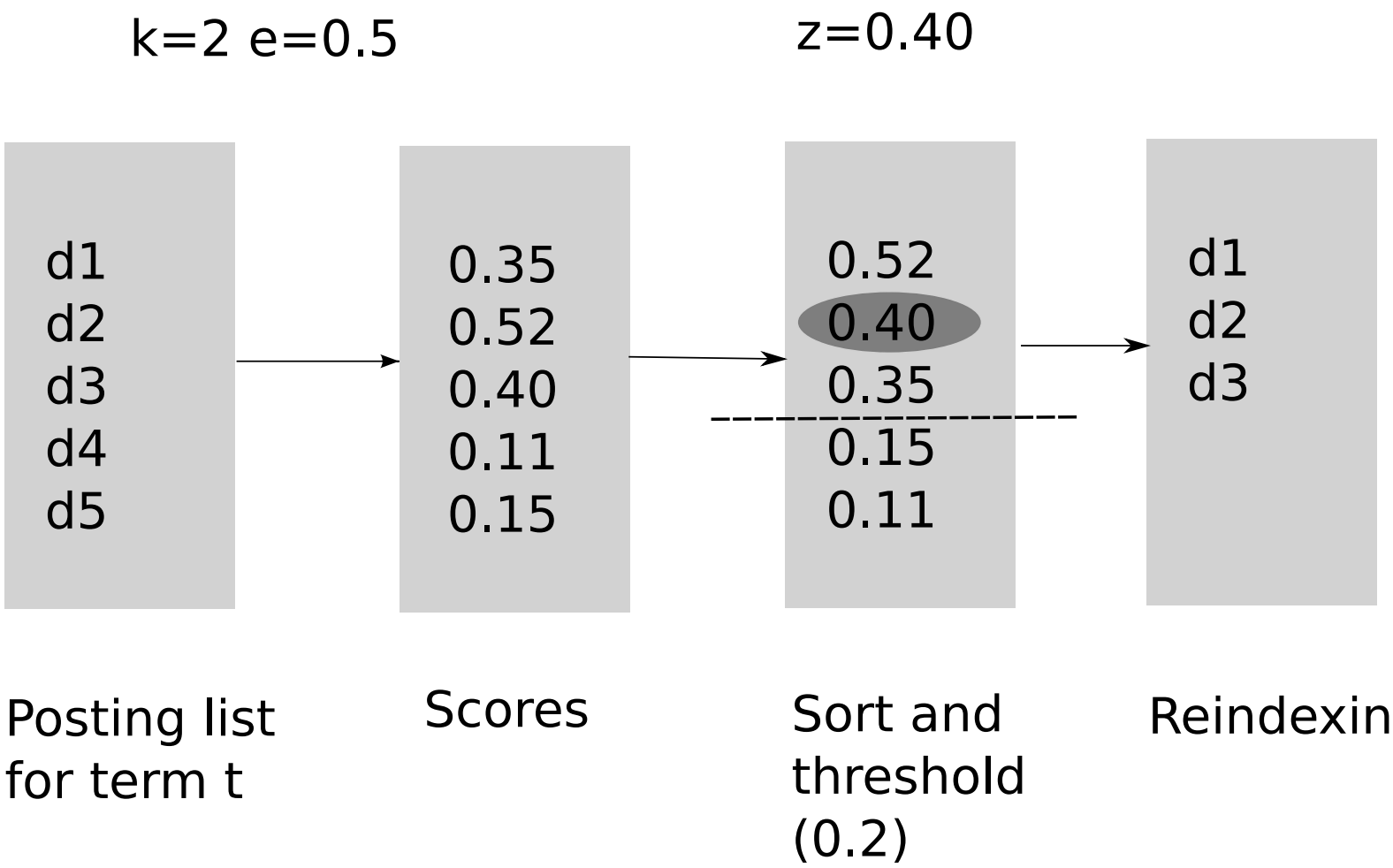
- IR systems use inverted files for efficient retrieval.
- They associate (at least) index terms with document occurrences
- Postings are compressed with static (lossless) coding methods: variable byte, γ, δ, Golomb ...

Pruning

- Technique for skipping document pointers during evaluation (dynamic).
- Static pruning removes the document pointers from disk (offline)
- We study the effect of removing collection-dependent stopwords (re-indexing), and compare it to a well-known pruning approach that removes document pointers.

Method of Carmel et al.(2001)

- For every posting list, computes the contribution of each document occurrence using a score function
- Computes a posting-dependent threshold $\tau = z_t * \epsilon$ where z_t is the k-th highest score for term t and ϵ is a parameter
- Removes every entry that scores lower than the threshold
- Under certain conditions, the k top documents retrieved are the same whether the original or the pruned inverted file is used.

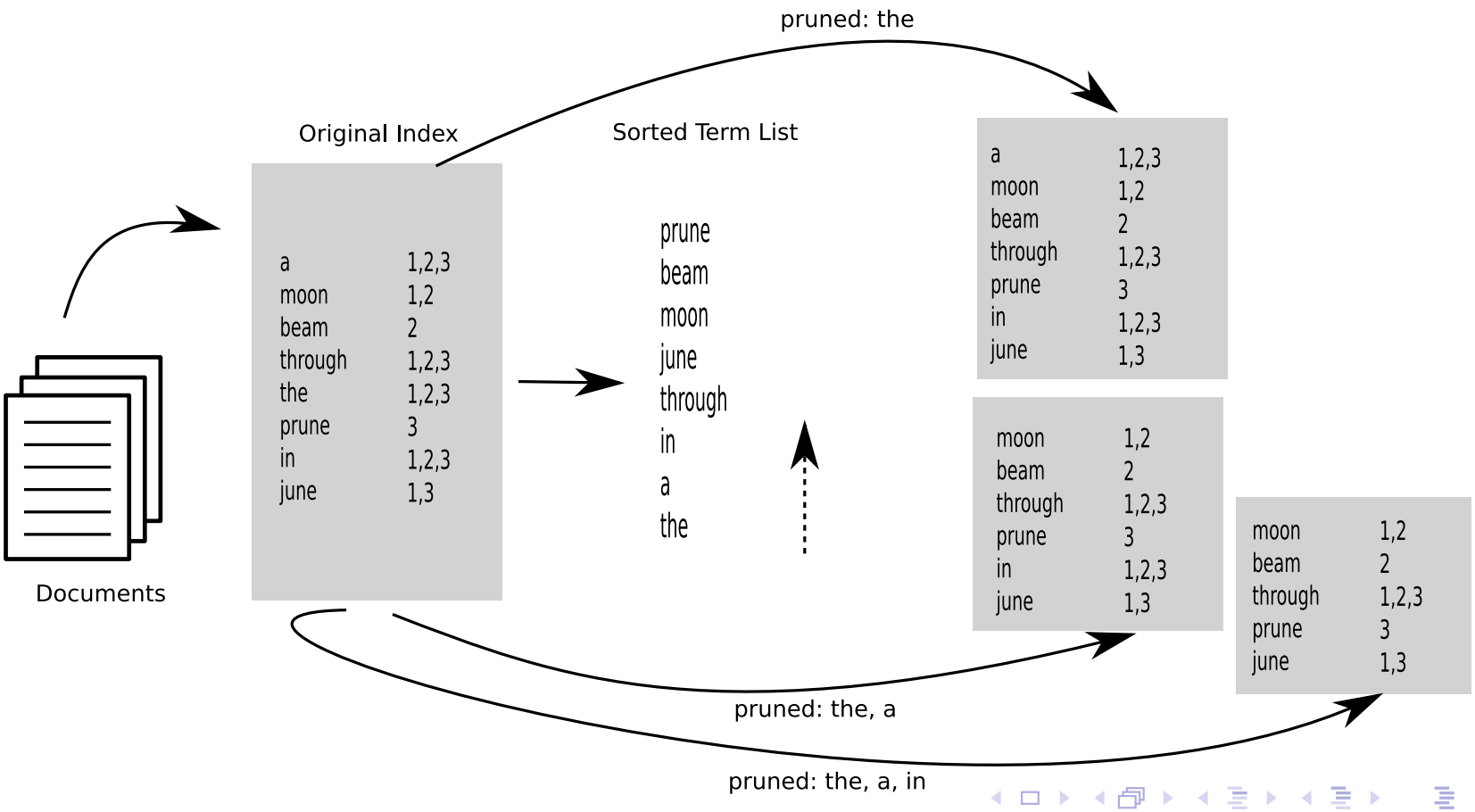

k=2 e=0.5
z=0.40
d1
d2
d3
d4
d5
0.35
0.52
0.40
0.11
0.15
0.52
0.40
0.35
0.15
0.11
d1
d2
d3
Posting list for term t
Scores
Sort and threshold (0.2)
Reindexing

- Carmel's method was evaluated on the LATimes collection using 50 topics and two different query sizes
- The algorithm conveys good results in terms of precision and similarity to the top results of the original unpruned index
- There is no experimentation on how the pruning algorithm behaves under different traditional lossless compression methods

Static pruning of terms

- How can we obtain a collection-dependent stop-words list? Are they useful as a pruning tool?
- Traditional stop-words removal uses a fixed term list to avoid indexing *noisy* terms
- High-frequency terms are highly compressible
- How much space are we saving when terms are removed from a compressed inverted file?
- Two options: *Informativeness* and *discriminative* rankings of terms

Static pruning of terms

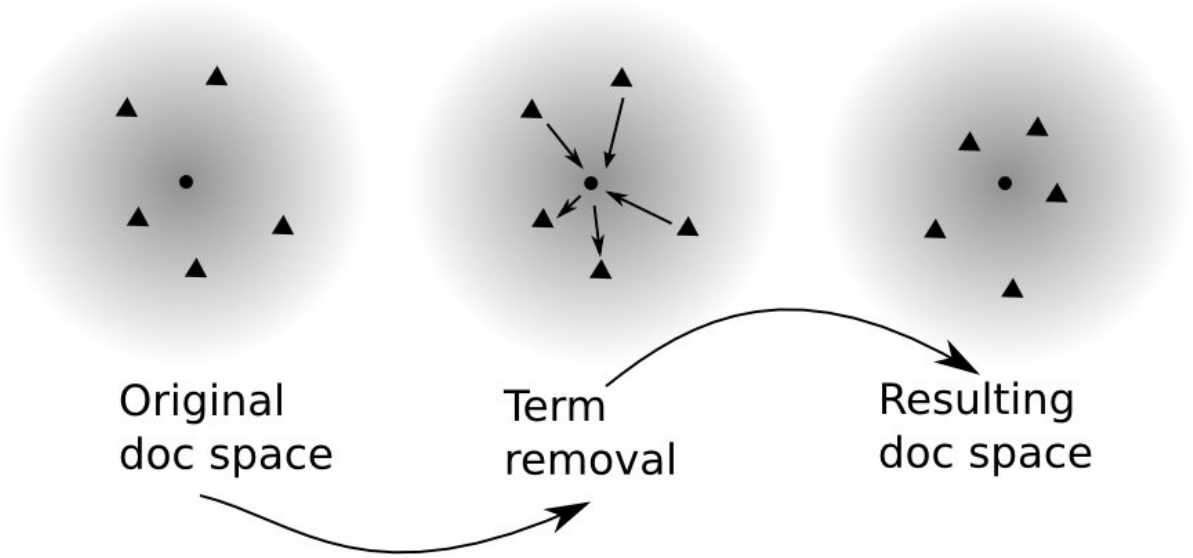


Informativeness rankings

- Inverse document frequency (and variants) is a common term informativeness measure, found in many retrieval methods
- Residual inverse document frequency is the difference between the observed idf and the idf expected if the term followed a Poisson distribution
- Claim: the more the term deviates from Poisson (random independence model) the more informative it is
- Both rankings are very fast to produce (all the information needed is in the lexicon)

Discriminative rankings

- Term Discrimination Model (TDM) measures the importance of every index term as the influence it has on the document space (Salton, 75)
- From the early vector space works, based on the discrimination value of a term
- Use limited to small pre-trec collections



- TDM measures the space density before and after removing a dimension (term): discriminative value of a term
- Efficient computation of the discrimination values is possible using a direct and an inverted file
- Two versions: standard cosine document distance + raw tf and standard cosine document distance + BM25's tf

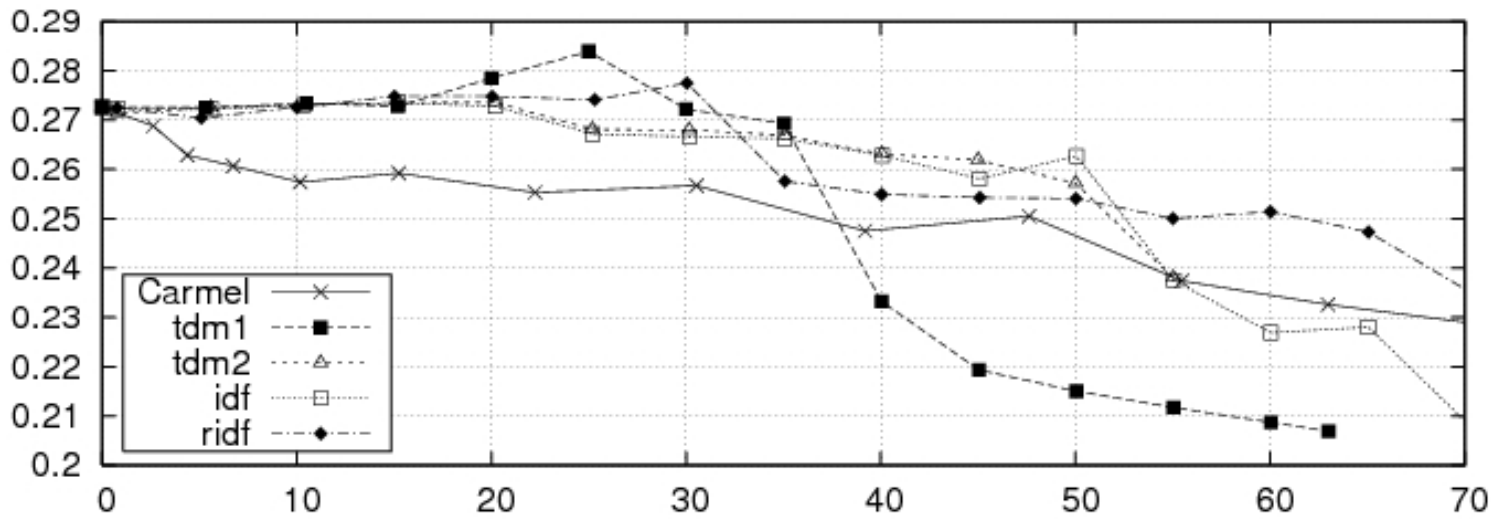
Experiments

1. Precision vs pruning
2. Index compression vs pruning.
3. Query evaluation time vs pruning

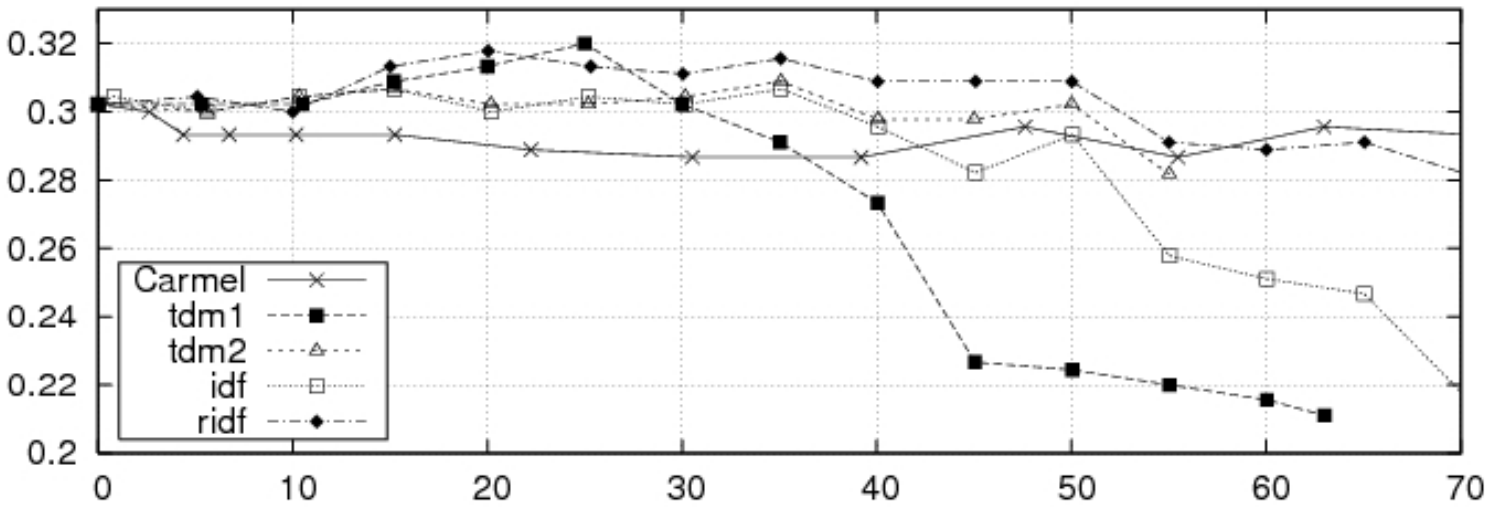
Settings

- BM25 with $b = 0.75$ and $k1 = 1.2$ (recommended settings)
- Experiments with short (title only) and long (title and description) queries
- Topics 401-450 LATimes and WT2g collections
- Five different popular coding schemes (γ, δ, variable byte, Golomb and interpolative coding).
- We measured the pruning level as the percentage of posting entries removed
- Indexing and retrieval using Terrier IR platform v1.0.0., own code for pruning and compression

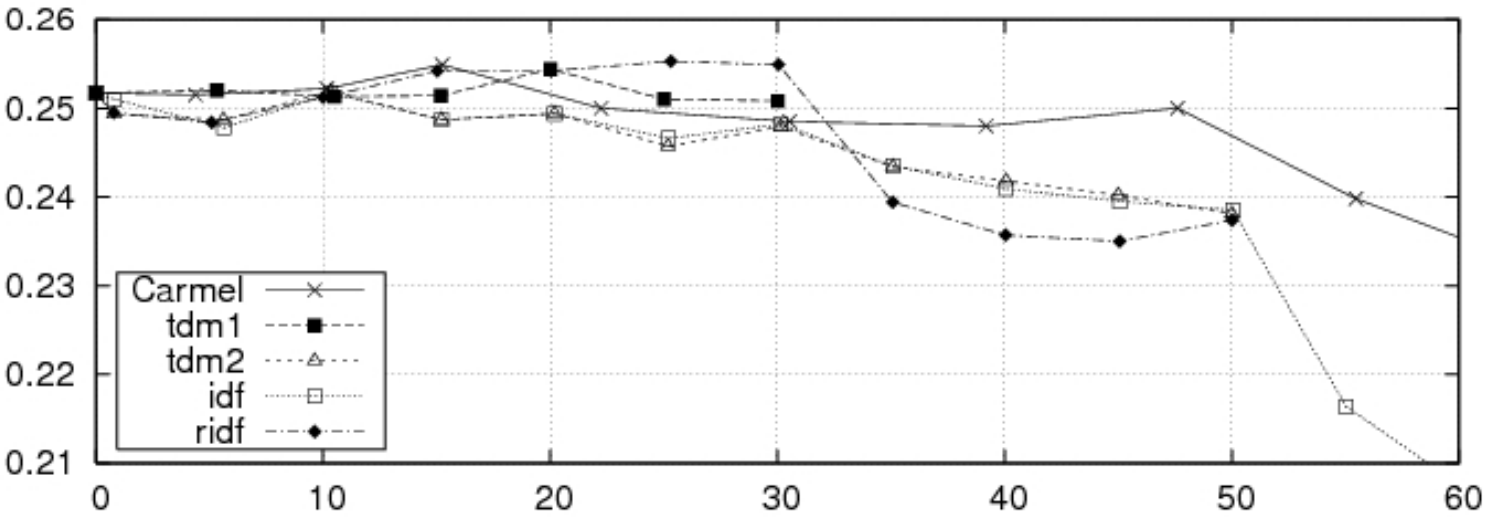
MAP vs % pruning (LATimes, long queries)



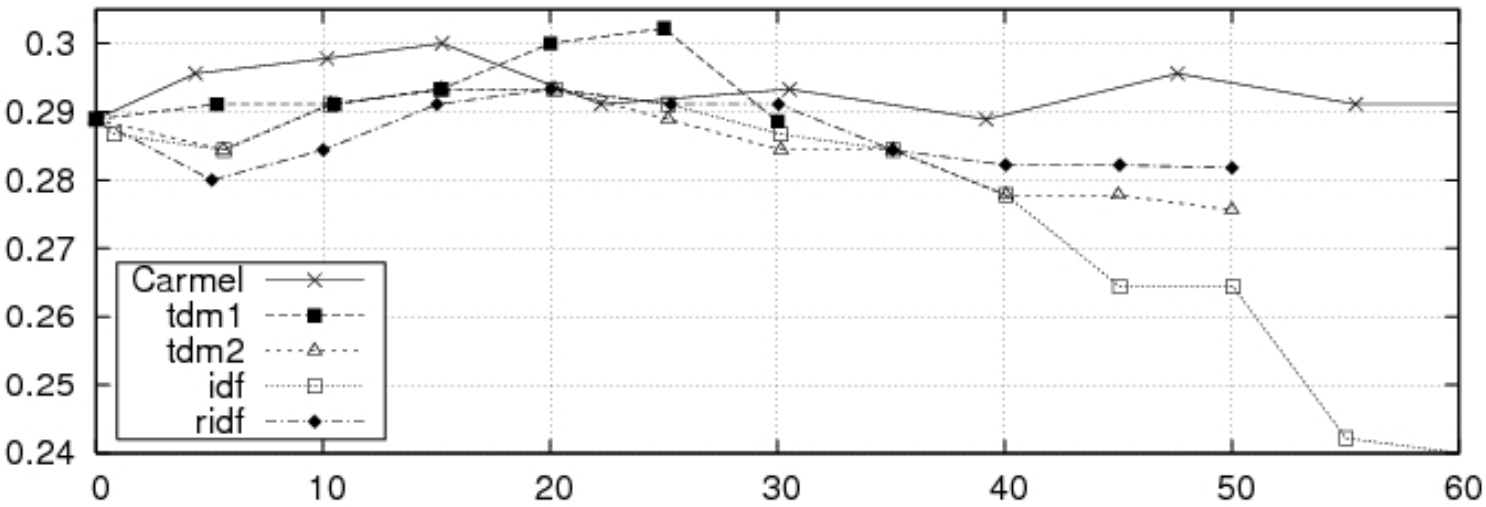
P@10 vs % pruning (LATimes, long queries)



MAP vs % pruning (LATimes, short queries)



P@10 vs % pruning (LATimes, short queries)



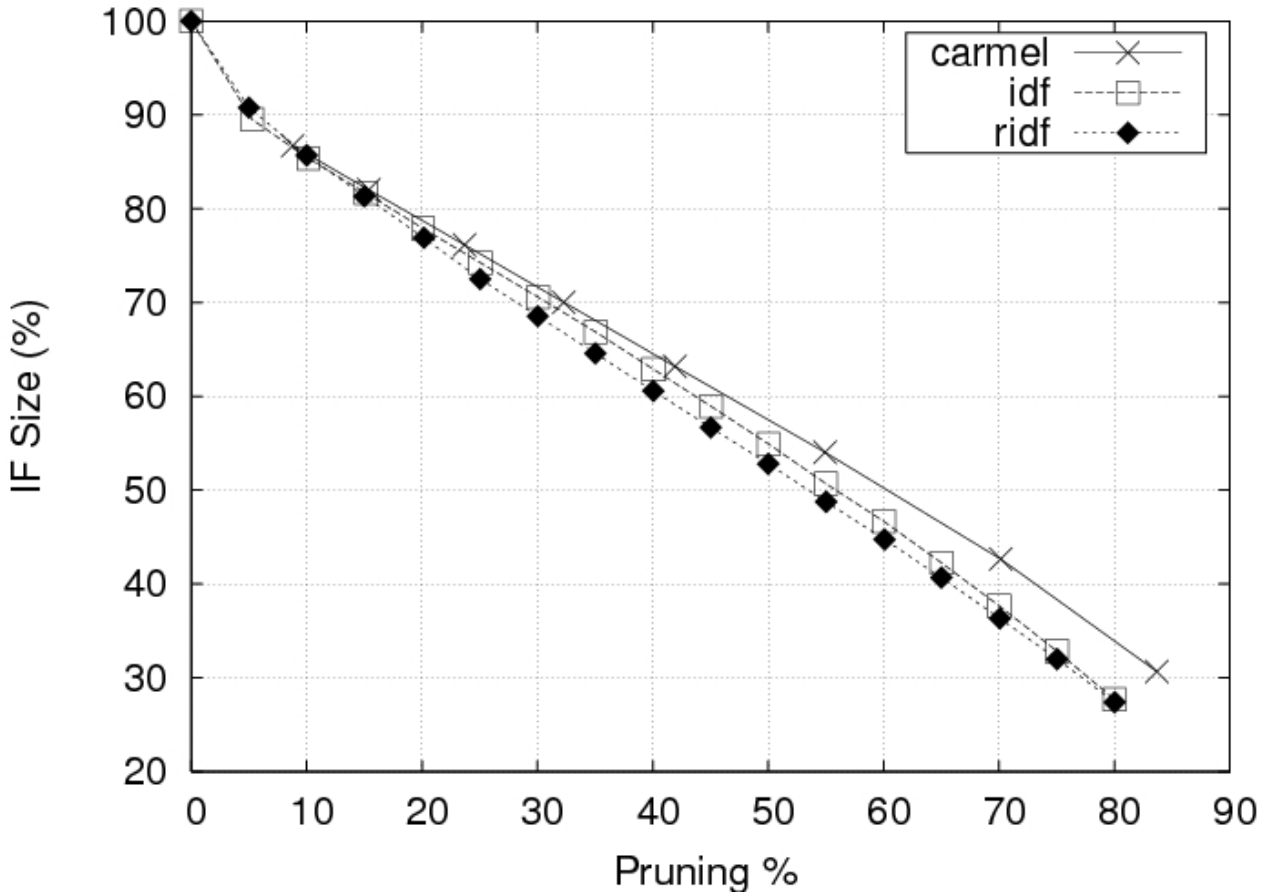
- Overall, term pruning can increase MAP and P@10
- Term pruning seems better suited for MAP whereas pruning of posting entries for P@10
- The ranking based on the original TDM method seems the best at low pruning levels
- Residual idf is very stable at every pruning level
- Idf and the normalised variation of tdm are very correlated
- Results are similar in the WT2g collection, with Carmel's method being extremely good for P@10 and short queries.

Automatic thresholding

- It is possible to combine two different rankings to avoid thresholds
- Use a common stop-words list (Fox's) as *relevance* (trustful) information
- Automatic thresholding brings almost the same precision values as Fox's stoplist alone, but at a higher pruning level.

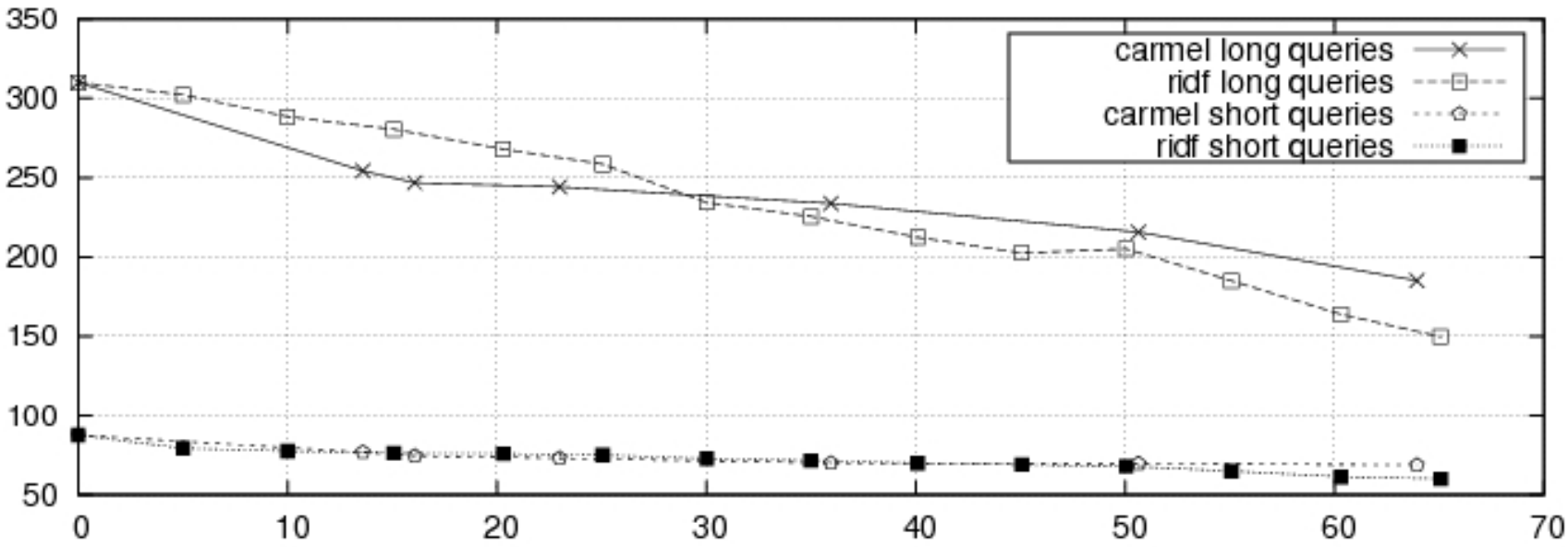
	Short queries		Long queries		Pruning
	MAP	p@10	MAP	p@10	
Automatic	0.2685	0.3044	0.2490	0.2889	56%
Fox	0.2695	0.3224	0.2524	0.3022	26.7%

Pruning and compression



- The behaviour is stable for every compression scheme
- Measuring the pruning level as the number of deleted posting occurrences is a valid indicator of the final IF size
- Term pruning achieves a slightly better space reduction, due to the gap encoding of document pointers

Pruning and querying times



- Average query time (ms) vs. % pruning (LATimes, 50 topics)

- Query processing times reduction are more noticeable in the case of long queries
- For pruning of document pointers, response times vary smoothly with respect to the pruning level
- Pruning document pointers reduces the amount of blocks of disk transferred, whereas removing a term forces evaluation to stop at the lexicon (no disk transfer at all)

Conclusions

- We implemented several pruning techniques based on the *informativeness* and *discriminative value* of terms
- In general, pruning whole terms is better for maintaining or improving MAP (up to a certaing pruning level), whereas pruning pointers is better for p@10
- The pruning based on *tdm1* rank is good if only high precision values are desired, although very aggressive
- Pruning based on *ridf* rank is easy to implement and very stable
- Carmel's method behaved very well for p@10 and short queries in the web collection
- Methods that prune terms can be useful for some kind of applications (indexing on small devices, desktop search...)