Static Pruning of Terms in Inverted Files

Roi Blanco and Álvaro Barreiro

IRLab, Computer Science Department
University of Corunna, Spain
rblanco@udc.es, barreiro@udc.es

Abstract. This paper addresses the problem of identifying collection dependent stop-words in order to reduce the size of inverted files. We present four methods to automatically recognise stop-words, analyse the tradeoff between efficiency and effectiveness, and compare them with a previous pruning approach. The experiments allow us to conclude that in some situations stop-words pruning is competitive with respect to other inverted file reduction techniques.

1 Introduction

Inverted files are the data structures employed by most modern retrieval systems [14] to associate index terms (words, stems, phrases, bigrams, etc...) with document occurrences. Indexes are organised into posting lists containing several pointers which carry the correspondence information. Fast query evaluation is normally done by repeatedly accessing the on-disk index file and fetching the information for every query term. Disk accessing times are the bottleneck for most retrieval systems, and there had been many solutions to improve query evaluation times without affecting retrieval effectiveness, such as lossless compression techniques [7]. More recently, a new family of lossy compression algorithms, namely *pruning*, has emerged to try to improve the efficiency while retaining high effectiveness values. Pruning techniques aim at removing unnecessary information by determining a set of non relevant pointers in each posting list and ruling them out of the retrieval. If the pointer set is dependent on each query, it is called *dynamic pruning* [13], whereas if the pruning can be made off-line it is said to be *static*. Recent works demonstrated that static pruning can produce very compact indices whilst not suffering from an unacceptable precision loss [2]. Also, this technique has been applied in web retrieval [4].

This paper presents several techniques for reducing the size of the inverted file by identifying a stop-words set dependent on the collection. The main difference between this method and the one described in [2] (hereinafter *Carmel's* method) is that the whole term is removed from the index instead of deleting single occurrences. We introduce several techniques based on the terms' *informativeness value*, in particular inverse document frequency (idf) and residual inverse document frequency ($ridf$), and a novel method based on the *term discriminative value*. Discarding a whole term determines that the index term is

not useful in every possible context (query). Although this claim may seem too aggressive (or naive), except for a predetermined and well-known set of function words, we found out that in some scenarios these algorithms prove to be competitive or even better than the methods based on the pruning of term-document occurrences. Other works ([2],[4]) size the amount of pruning as the percentage of pointers removed from the inverted file, and in [2] Carmel et al. advanced that it is not known how static pruning would behave in conjunction with the traditional lossless compression methods, and that further research was needed in order to clarify it. This paper also presents the experiments and results assessing the relationship between the amount of pointers and the real space savings, for five well known coding algorithms. We advance a good and stable behaviour of the static pruning methods for every coding scheme tested. Experiments also report on query times in a real retrieval platform.

The rest of the paper is organised as follows: section 2 describes Carmel's method, section 3 introduces the term pruning methods, the experiments and results are presented in section 4 and the paper ends with a conclusions and further work section.

2 Static index pruning of posting entries

Carmel et al. in [2] proposed and successfully tested a method for removing information from an inverted file. The algorithm operates in a per-term basis, selecting the *less necessary* information from every single posting list in order to reduce the total index size.

There are two parameters involved in the so-called top-k pruning algorithm: k and ϵ. The procedure to select which postings are removed from the index is as follows. First, for every term in the lexicon, the algorithm computes the contribution of every document occurrence to the final score using the score function of the retrieval system. Then it retrieves the k-th highest score z_t and sets a threshold $\tau_t = \epsilon * z_t$. Finally, every document occurrence which score is lower than τ_t is dropped out from the posting list.

It is worth to point out that this is an *idealised pruning algorithm*, as the top k documents scores for a query with less than $\frac{1}{\epsilon}$ terms are guaranteed to be the same, within an error of ϵ, when the original or pruned inverted file is used. However, the algorithm has the problem of obtaining negligible pruning levels. In order to obtain any significant index reduction it is necessary to shift every document occurrence score in the term lists, by subtracting a global minimum score to every document score. The real procedure is to apply the pruning algorithm after this ad-hoc modification of the inverted file. This accomplishes excellent results but the aforementioned property is not proved to hold. As well, there is another variation of the algorithm, namely δ-top answers, that consists of keeping the entries whose score value under a query q is at least δ times the highest score of all the documents under q. The implementation we employed here considered the BM25 score [11] instead of Smart's tf-idf (used in [2]) and we

decided to skip any shifting implying that higher pruning levels were obtained by setting a higher ϵ value.

3 Static index pruning of term posting lists

Traditionally, stop word removal aims at identifying noisy terms that may hurt precision, and to the best of our knowledge it has not been used for efficiency purposes.

It is clear that removing high-frequency terms from an uncompressed inverted file may lead to substantial space savings, as they tend to engross most of the occurrences (according to Zipf's law). How this may affect to compressed inverted files is disccussed in [14]. The claim is that the higher the frequency of the word, the better a parametrised compression model such as Golomb will adapt to it, so the less space it will consume in a compressed form. In general, it is a commonly accepted idea that stop-words should be in the inverted file since removing high-frequency words would result in very small space savings. However, we believe that if it is possible to obtain a good ranking of terms according to their *importance*, it would be interesting to establish the tradeoff between retrieval accuracy and the index reduction implied by the removal of the *less important* terms. In fact, some authors [10] report that building a manual extended stop-list speeds searches. We propose to study this effect with techniques that obtain *informativeness* (3.1) and *discriminative* (3.2) rankings.

3.1 Stop-words list based on idf and ridf

The inverse document frequency is a term informativeness measure, therefore it can be used to produce a ranking of bad terms (those with lower idf values). We used a common *idf* normalisation introduced by Robertson and Sparck-Jones in [9] that performed well for identifying dynamic stop-words in [6]. If D is the total number of documents in the collection, and df the number of documents the term t appears in (document frequency), then the idf for term t is:

$$idf = \log\left(\frac{D - df - 0.5}{df + 0.5}\right) \tag{1}$$

Residual idf is defined in [3] as the difference between the observed idf (IDF) and the idf expected under the assumption that the terms follow an independence model, such as Poisson ($I\hat{D}F$). To the best of our knowledge it has not been used for identifying collection-dependent stop-words, although in [8] it is employed successfully for named entity recognition. If tf is the total number of tokens for a term t, then the ridf devised by a Poisson distribution is

$$RIDF = IDF - I\hat{D}F = -\log(\frac{df}{D}) + \log(1 - e^{-\frac{tf}{D}}) \tag{2}$$

Church and Gale [3] claim that the more a term deviates from Poisson, the more dependent on hidden variables, and more useful the term is to discriminate between documents containing it on the basis of the hidden dependencies.

In order to compute the *idf* and *ridf* values for every term appearing in the collection, it is only necessary to traverse the lexicon file once.

3.2 Stop-words list based on Salton's Term Discrimination Model

Salton's Term Discrimination Model (TDM) [12] is one of the first computationally attractive attempts to find an effective ranking of words, based on the analysis of the *Discriminative Value* (DV) of a term and it was used for automatic indexing. The model is embodied into the vector-space framework for Information Retrieval and its use has been limited to small collections (Cranfield, Medlars, Time). However, the usefulness of the model has not been clearly stated in the following years, nor it has been applied in large TREC collections. This paper proposes to revisit the original model and to determine to which extend it may be worthy as a tool for finding stop-words.

The Term Discrimination Model measures the importance of every index term based on the influence it has on a document space. The main assumption is that a document space with distant vectors is preferable for retrieval. A good document space is one that maximises the average separation between every pair of vectors, because it would be easier to distinguish among the retrieved documents. Under this claim, and given that terms act as dimensions of the document space, it is possible to rank the index terms according to how much each term affects the *density* of the vector space, i.e. how good as *discriminators* they are. The DV of a term t is defined as how much the removal of t from the vector space decreases the total space density.

Let $\{t_1 \dots t_T\}$ and $\{d_1 \dots d_D\}$ be the term and the document set respectively, where every document d_i is represented by a term frequency component vector $\langle tf_{i1}, tf_{i2} \dots tf_{iT} \rangle$. The calculation of every document-to-document distance as a measure of the space density is computationally unaffordable for very large collections. One possible variation could be a definition of the density measure related to documents-to-centroid distances. In this case, the DV for a term t_k is

$$DV_k = \sum_{i=1}^{D} \text{distance}(d_i^k, c^k) - \sum_{i=1}^{D} \text{distance}(d_i, c) = Q_k - Q, \tag{3}$$

where Q is the space density, Q_k is the space density after the term t_k is removed, d_i^k is the document obtained after removing the term t_k from d_i, c is the document centroid and c^k is the document centroid resulting after the removal of the term t_k.

A straight implementation of eq. 3 is very time consuming. For every term, it requires the computation of the similarities between every document and the centroid, forcing to traverse T times a direct file of D documents. Next it follows a reformulation of eq. 3 that allows to save most of the operations by storing some data in main memory and reducing drastically the total computation time. First, let TF_j^k (TF_j) be the j-th component of the centroid c^k (c):
$TF_j = \frac{1}{D} \sum_{i=1}^{D} tf_{ij}$; $TF_j^k = TF_j$ if $j \neq k$; $TF_j^k = 0$ if $j = k$.

Equation 3 can be rewritten as follows, where tf_{ij}^k is the j-th component of d_i^k.

$$DV_k = \sum_{i=1}^{D}\sum_{j=1}^{T} \frac{tf_{ij}^k \times TF_j^k}{|d_i^k| \times |c^k|} - \sum_{i=1}^{D}\sum_{j=1}^{T} \frac{tf_{ij} \times TF_j}{|d_i| \times |c|}, \tag{4}$$

Let $w_i = \sum_{j=1}^{T} tf_{ij} TF_j$, which is a value that can be precomputed for each d_i, then

$$\sum_{j=1}^{T} tf_{ij}^k \times TF_j^k = \begin{cases} w_i & \text{if } t_k \notin d_i \\ w_i - tf_{ik} TF_k & \text{if } t_k \in d_i, \end{cases} \tag{5}$$

and Q_k can be expressed as

$$Q_k = \sum_{i \setminus t_k \in d_i}^{D} \frac{w_i - tf_{ik} TF_k}{|c^k| \times |d_i^k|} + \sum_{i \setminus t_k \notin d_i}^{D} \frac{w_i}{|c^k| \times |d_i^k|} \tag{6}$$

Taking into account that $|d_i^k| = |d_i|$ if $t_k \notin d_i$, and that $\sum_{i \setminus t_k \notin d_i} \frac{w_i}{|d_i|} = \sum_{i=1}^{D} \frac{w_i}{|d_i|} - \sum_{i \setminus t_k \in d_i} \frac{w_i}{|d_i|}$, then Q_k can be finally rewritten as:

$$Q_k = \frac{1}{|c^k|} \left(\sum_{i \setminus t_k \in d_i}^{D} \left(\frac{w_i - tf_{ik} TF_k}{|d_i^k|} - \frac{w_i}{|d_i|} \right) + \sum_{i=1}^{D} \frac{w_i}{|d_i|} \right) \tag{7}$$

Since Q is constant, the Q_k values will suffice to compute the rank produced by the TDM. The reformulation of Q_k introduced in eq. 7 allows the computation of this rank with just one single pass to a direct file to calculate the w_i and $|d_i|$ values, and another one to the inverted file to recalculate every single term contribution. If we use the cosine normalisation, then $|d_i| = \sqrt{\sum_{j=1}^{T} tf_{ij}^2}$, $|c| = \sqrt{\sum_{j=1}^{T} TF_j^2}$, implying that $|d_i^k| = \sqrt{|d_i|^2 - t_{ik}^2}$, $|c^k| = \sqrt{|c|^2 - TF_k^2}$. Finally we propose another last modification to this model, in which the contribution $\frac{1}{|c^k|} \sum_{i=1}^{D} \frac{w_i}{|d_i|}$ is dropped out from eq. 7. This factor is dominant in the final value of Q_k and very dependent on the $|c^k|$ value. This is a problem in large collections because the method is too biased for high frequency terms (concretely on the factor TF_j appearing on $|c^k|$), ranking them higher.

This efficient implementation of the Term Discrimination Model requires $2|D| + |T|$ extra pointers to store the document lengths, the w_i (for each document) and the TF_j (for each term) values. Considering 16-byte double precision floats, these amounts sum up to approximately 12 MB for the 2 Gigabyte TREC web collection.

The approach described here will be referred as *tdm1* and we denote as *tdm2* another variation that employs a term frequency normalisation factor in the fashion of BM25 [11]:

$$\hat{tf}_{ij} = \frac{(k_1 + 1) tf_{ij}}{tf_{ij} + k_1 \left((1 - b) + b \frac{len(d_i)}{avglen} \right)} \tag{8}$$

In equation 8, $len(d_i)$ stands for the number of tokens in the document d_i, $avglen$ is the average document length in the collection and we used the recommended values for $k_1 = 1.2$ and for $b = 0.75$. In the implementation of *tdm2* we considered the simplification of not recomputing the average document length every time a term is removed from the collections. Once the term frequencies are computed according to eq. 8 the process follows as described for *tdm1*.

4 Experiments and Results

4.1 Experimental Setting

We report our empirical findings using the five pruning methods described in sections 2 and 3. The evaluation tries to assess how the mean average precision (MAP) and precision at ten (P@10) vary as the number of deleted occurrences from the inverted file increases. Intentionally, we chose settings that devise high precision values in order to measure the decrease in precision when augmenting the pruning level. We used Porter's algorithm for stemming. BM25 (eq. 9) was selected as the scoring function for every method, as it has proved to be robust in the IR literature:

$$score(d, Q) = \sum_{t \in Q} \log_2 \left(\frac{D - df_t + 0.5}{df_t + 0.5} \right) \frac{(k_1 + 1)tf}{K + tf} \frac{(k_3 + 1)qtf}{k_3 + qtf} \qquad (9)$$

where qtf is the frequency of the term in the query, $K = k_1((1-b) + b\frac{d_l}{avgl})$, and d_l and $avgl$ are the document and average document length respectively. The recommended values [11] are: $k_1 = 1.2$, $k_3 = 1000$ and $b = 0.75$.

We experimented with TREC topics from 401 to 450 in the LATimes and WT2g collections, short queries (title) and long queries (title plus description). Note that the narrative field was discarded as it hurts precision using these settings. Regarding to Carmel's method, the k value was set to 10, and the different pruning levels were obtained by modifying ϵ.

For the TDM-based methods, another condition was taken into account in order to smooth the correlation between the frequency range and the discrimination value. We introduced a document frequency threshold based on the size the collection: only terms with document frequency in the collection greater than 400(2000) where pruned for the LATimes(WT2g) collection.

A second class of experiments try to assess the real tradeoff between the pruning level and the disk space occupied by the inverted file, using different posting-list compression methods. We experimented with five different coding algorithms [7] for the document pointers: three non-parametrised methods (γ, δ, *variable byte*), a local parametrised method (*Golomb coding*), and a context-sensitive method (*interpolative coding*). Within-document term frequencies were coded with unary code, except for the case of variable byte where they were coded with variable bytes as well.

Finally, a third experiment measured the real query time performance of the system for one term-based method (*ridf*) and Carmel's method, to try to determine the final speedup effect of pruning on a retrieval platform.

Indexing and retrieval was carried out using the Terrier IR platform[1] v1.0.0, developed at the University of Glasgow. The pruning and compression program suite was implemented on top of it.

4.2 Precision vs. pruning

Figures 1 to 4 show MAP and P@10 results for the LATimes collection, for both short and long queries. The precision curves end when the number of terms deleted forces any query to be empty. In general, all the methods that prune terms are able to increase initial MAP and P@10 values. Overall, *tdm1* achieved the highest values in precision with a pruning level around 20%-30%. If Fox's stop-list [5] is applied the results are: MAP 0.2695(0.2524) and P@10 0.2933(0.2911) for long(short) queries at a 26.7% pruning level. The best values achieved with the *tdm1*, MAP 0.2839(0.2544) and P@10 0.3224 (0.3022), are better than those attained by Fox's stop-list. Term pruning methods present a good behaviour at certain levels, being ridf remarkably stable and smooth and *tdm1* very good at increasing precision, although at the cost of being too aggressive. The other two methods, *tdm2* and *idf* are very correlated and perform slightly worse than ridf for most of the cases.

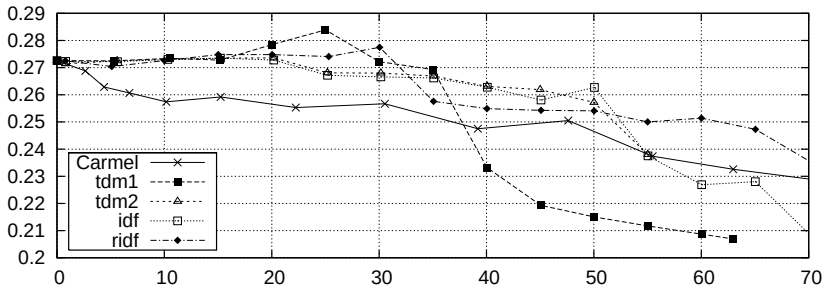

Fig. 1. MAP vs. %pruning for LATimes & long queries

Tables 1 and 2 summarise the results for the WT2g collection. Results are analogous to the ones obtained in the LATimes, although short queries benefit more from precision gains. It is remarkable that Carmel's method is able to improve P@10 values in the WT2g collection at very high pruning levels (short queries only).

Every method presented needs to set some threshold in order to *stop pruning*, be it the ϵ parameter (Carmel's method) or the percentage of pruning (term pruning methods). We carried out a third experiment in order to find an automatic threshold using Fox's stop-list as *relevance* information, i.e. good stopwords. The procedure is as follows: the list of terms is sorted according to a first

[1] http://ir.dcs.gla.uk/terrier

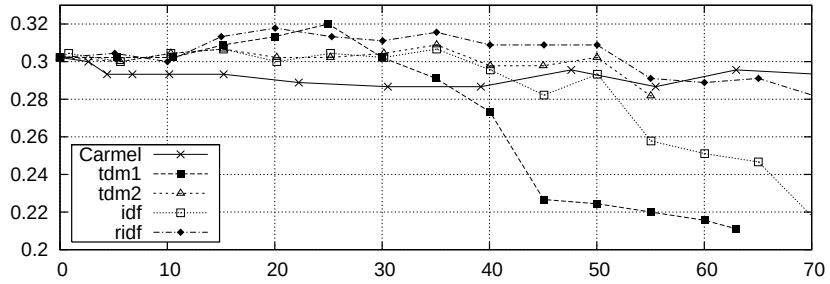

Fig. 2. P@10 vs. %pruning for LATimes & long queries

measure and split into several intervals bounded by the *relevant* (trusted) stopwords. For every term and using a second measure, its informativeness value v_1 and the value of the lower bound of its corresponding interval v_2 are compared. If $v_2 \geq v_1$ the term is pruned. Combining the *ridf* (first) and *tdm2* (second) measures this approach gives, for long(short) queries, MAP values of 0.2685(0.2490) and P@10 values of 0.3044(0.2889) at a 56% pruning level in the LATimes collection. These precision values are obtained automatically and comparable with the ones obtained by Fox's stop-list alone, but at a higher pruning level.

Table 1. Precision vs. %pruning WT2g & long queries

		pruning									
		0%	10%	15%	20%	25%	30%	40%	50%	60%	65%
tdm1	MAP	0.2966	**0.3006**	**0.3062**	**0.3074**	0.2892	0.2704	0.2473	0.2151	0.2054	–
	P@10	0.4780	0.4780	0.4780	**0.4860**	**0.4660**	0.4460	0.3980	0.3440	0.3143	–
tdm2	MAP	0.2966	0.2985	0.2942	0.2925	0.2755	0.2741	0.2602	0.2436	0.2166	0.2054
	P@10	0.4780	0.4620	0.4600	0.4680	0.4360	0.4400	0.4080	0.3780	0.3583	0.3208
idf	MAP	0.2966	0.2987	0.2945	0.2928	0.2749	0.2733	0.2599	0.2410	0.2163	–
	P@10	0.4780	0.4640	0.4620	0.4700	0.4380	0.4380	0.4100	0.3760	0.3204	–
ridf	MAP	0.2966	0.3000	0.3050	0.2970	**0.2922**	**0.2962**	**0.2881**	**0.2625**	**0.2325**	**0.2322**
	P@10	0.4780	**0.4800**	**0.4880**	0.4640	0.4600	**0.4640**	**0.4560**	**0.4320**	**0.3760**	**0.3653**

		pruning									
		0%	9.3%	14.0 %	19.1%	24.2%	30.3%	39.4%	52.1%	58.1 %	66.0%
Carmel	MAP	0.2966	0.2789	0.2779	0.2712	0.2634	0.2591	0.2606	0.2405	0.2283	0.2188
	P@10	0.4780	0.4480	0.4460	0.4540	0.4440	0.4480	0.4400	0.4280	0.4220	0.4200

4.3 Index compression vs. index pruning

Figure 5 shows the real tradeoff between pruning level and disk space usage (WT2g collection). The graphs reflect how the inverted file size decreases when the number of pruned pointers increases using different coding methods. Sizes are relative with respect to the original inverted index except in the last graph, where the size is absolute. Only the posting list file is considered since the space reduction due to the lexicon file is not significant. The behaviour is stable for

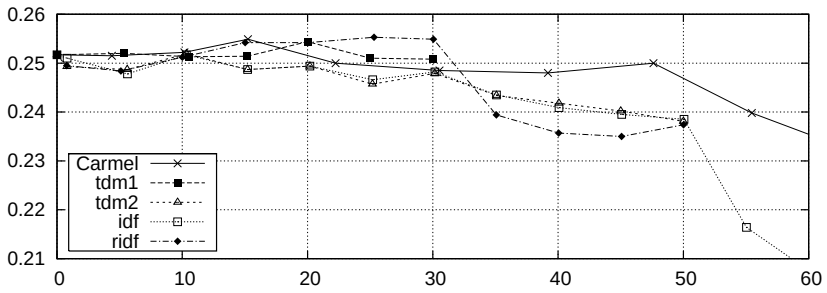

Fig. 3. MAP vs. %pruning LATimes & short queries

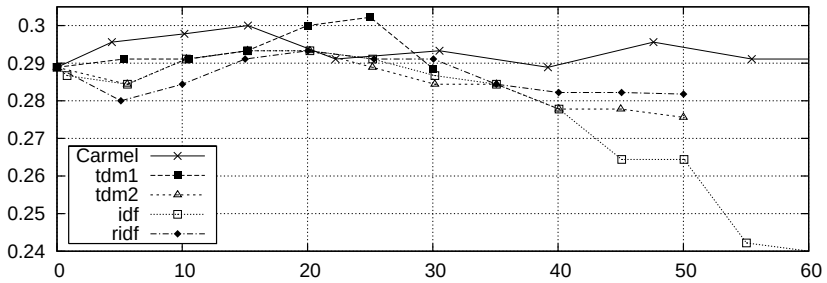

Fig. 4. P@10 vs. %pruning LATimes & short queries

every compression algorithm, which proves that measuring the pruning level as the number of deleted occurrences is a valid indicator of the final compressed file, despite of the coding method used. The best reduction is obtained for the method based on *ridf* although with minor differences. The final figure shows the relative performance of the different coding algorithms, measured in megabytes (pruning values obtained with *ridf*).

It is possible to explain the values in figure 5 as follows. Real coding of posting lists is based on document gaps. A document gap is the difference between two consecutive document identifiers in the same list. For a given term with consecutive document identifiers a, b, c the cost of coding its postings would be $\phi(b-a)+\phi(c-b)$ and for bit-based coding methods $\phi(x) = O(\log(x))$. Carmel's method may prune the document occurrence with identifier b resulting in a coded posting list reduction from $\log(b-a)+\log(c-b)$ to $\log(c-a)$. Methods that prune every term occurrence do not leave this $\log(c-a)$ gap in the posting list when they operate, as they remove the whole list, thus they may yield less average bits per gap values. The first slope in the graphs is due to the fact that the first terms in being pruned are the ones with highest document frequency, which happen to be the ones with the highest within-document term frequencies. When those frequencies are coded in unary ($\phi(x) = x$) the space saved when they are removed is more noticeably. In fact, if the frequencies are coded with gamma,

Table 2. Precision vs. %pruning WT2g & short queries

		pruning									
		0%	10%	15%	20%	25%	30%	35%	40%	50%	55%
tdm1	MAP	0.2540	**0.2688**	**0.2719**	**0.2661**	0.2524	0.2470	–	–	–	–
	P@10	0.4180	**0.4540**	**0.4560**	**0.4620**	**0.4480**	**0.4271**	–	–	–	–
tdm2	MAP	0.2540	0.2635	0.2641	0.2600	0.2498	0.2490	0.2393	0.2351	0.2172	–
	P@10	0.4180	0.4360	0.4360	0.4300	0.4040	0.4060	0.3800	0.3620	0.3553	–
idf	MAP	0.2540	0.2635	0.2644	0.2602	0.2503	0.2495	0.2408	0.2351	0.2172	0.2109
	P@10	0.4180	0.4380	0.4380	0.4300	0.4080	0.4040	0.3780	0.3600	0.3480	0.3163
ridf	MAP	0.2540	0.2619	0.2640	0.2636	**0.2594**	**0.2572**	**0.2524**	**0.2509**	**0.2333**	**0.2254**
	P@10	0.4180	0.4400	0.4360	0.4204	0.4143	0.4204	**0.4020**	**0.3939**	**0.3633**	**0.3653**

		pruning									
		0%	9.27%	14.0%	19.1%	24.2%	31.0%	39.6%	45.2%	52.1%	58.9%
Carmel	MAP	0.2540	0.2634	0.2632	0.2622	0.2606	0.2558	0.2548	0.2526	0.2397	0.2301
	P@10	0.4180	0.4360	0.4360	0.4360	0.4380	0.4420	0.4400	0.4500	0.4580	0.4500

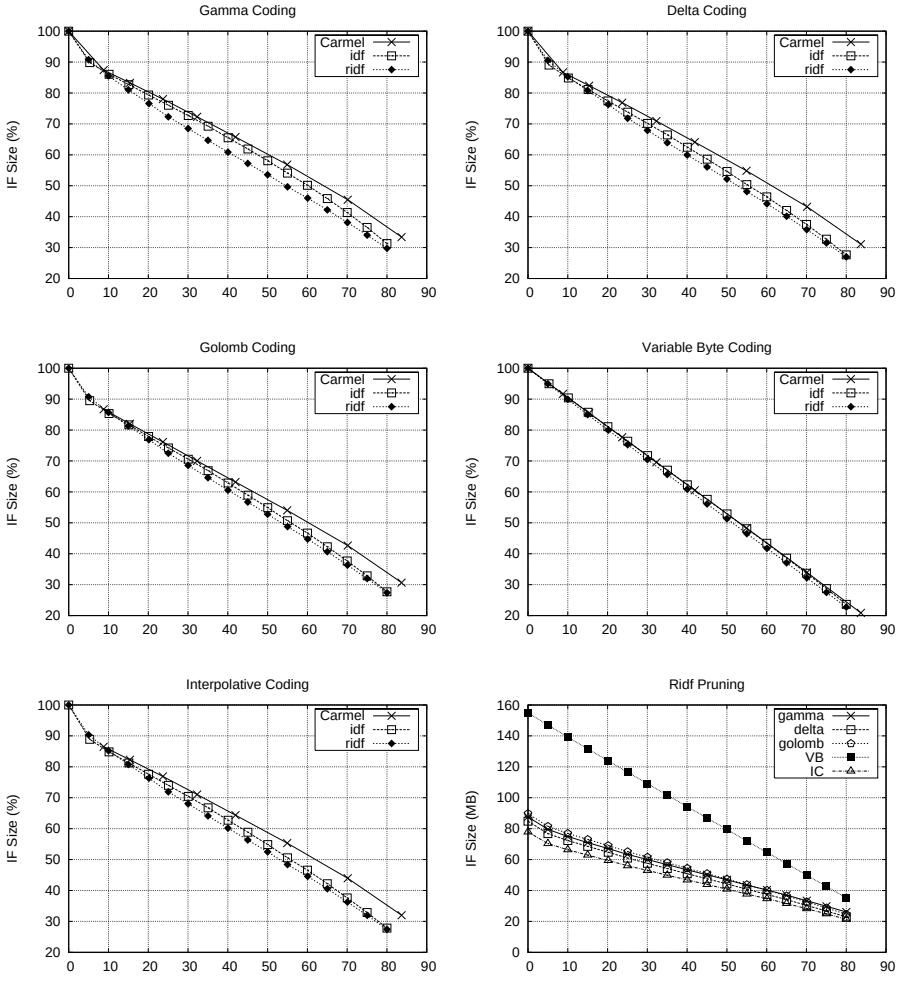

Fig. 5. Effect on Inverted File size vs. %pruning

the slope softens. It is interesting to notice that Carmel's method follows this behaviour too, which indicates that if ϵ is low, it is only able to delete occurrences of terms with high document frequency.

In the case of variable byte coding, 90% of the pointers require just one byte and therefore there is no noticeable difference among the methods. Variable byte is clearly the worst method with respect to inverted file size, although it is interesting because of its faster decompression times.

4.4 Query times vs. pruning

Figure 6 reports on average query times for *ridf* and Carmel's method on the LATimes collection with fifty queries (topics from 401 to 450). There is a query processing time reduction which is more important in the case of long queries. The different behaviour between the methods is due to the number of disk accesses, main bottleneck for query evaluation in retrieval systems. Every query term is processed if the inverted file is pruned with Carmel's method, and this is the reason why query processing time varies smoothly with respect to the pruning level. In the *ridf*-based pruning method, query processing times can be drastically reduced at pruning levels that maintain or even improve the precision values.

5 Conclusions and future work

We implemented several pruning techniques based on the *informativeness* and *discriminative value* of terms. We also evaluated the behaviour of precision with respect to pruning, and the final effect in index file reduction and query processing times. Those methods have been compared with the well-known pruning method introduced by Carmel et al. [2]. We found out that *tdm1* is good if only high values of precision are desired, although it is very aggressive, and *ridf* is easy to implement and very stable. In general, pruning whole terms is better for maintaining or improving MAP, and it keeps precision values at high pruning levels with long queries, whereas pruning pointers is better with respect to P@10. In particular, Carmel's method behaved very well for P@10 and short queries in the WT2g collection. Therefore, methods that prune terms could be useful in applications such as indexing collections for PDAs and mobile devices, and desktop search.

One future research line is to design a pointer-based pruning method that operates selectively over posting lists, driven by a global term rank. Another topic of research is to address the problem of pruning while allowing for phrasal queries. None of the methods presented here is appropriate for processing phrasal queries. To tackle these problems it is necessary to develop an explicit pruning method for this purpose [4] or to combine a pruned inverted file with a next-word index [1].

Acknowledgements. The work reported here was co-funded by SEUI and FEDER under project MEC TIN2005-08521-C02 and "Xunta de Galicia" under project

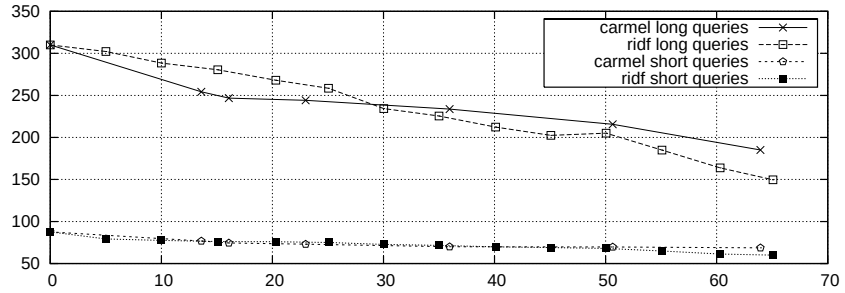

Fig. 6. Average query processing time (ms) vs. %pruning

PGIDIT06PXIC10501PN. Roi Blanco is supported by a grant of DXID of the "Xunta de Galicia". We also thank the support of the "Galician Network of NLP&IR" (2006/03).

References

1. D. Bahle, H. Williams, and J. Zobel. Efficient phrase querying with an auxiliary index. In *Proc. of ACM SIGIR* 2002, pages 215–221.
2. D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, Y. Maarek, and A. Soffer. Static index pruning for information retrieval systems. In *Proc. of ACM SIGIR* 2001, pages 43–50.
3. K. Church and W. Gale. Poisson mixtures. *Natural Language Engineering*, 2(1):163–190, 1995.
4. E. S. de Moura, C. F. dos Santos, D. R. Fernandes, A. S. Silva, P. Calado, and M. A. Nascimento. Improving web search efficiency via a locality based static pruning method. In *Proc. of WWW* 2005, pages 235–244.
5. C. Fox. A stop list for general text. *SIGIR Forum*, 24(1-2):19–21, 1990.
6. R.T.W. Lo, B. He, and I. Ounis. Automatically building a stopword list for an information retrieval system. In *Proc. of DIR'05*, Utrecht, Netherlands, 2005.
7. A. Moffat and A. Turpin. *Compression and Coding Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA, 2002.
8. J. D. M. Rennie and T. Jaakkola. Using term informativeness for named entity detection. In *Proc. of ACM SIGIR* 2005, pages 353–360.
9. S. Robertson and K. Sparck Jones. Relevance weighting of search terms. *JASIS*, 27:129–146, 1976.
10. S. E. Robertson and S. Walker. Okapi/Keenbow at TREC-8. In *Text REtrieval Conference*, pages 151–162, 2000.
11. S. E. Robertson, S. Walker, M. Hancock-Beaulieu, A. Gull, and M. Lau. Okapi at TREC-4. In *Text REtrieval Conference*, pages 21–30, 1996.
12. G. Salton, C. S. Yang, and C. T. Yu. A theory of term importance in automatic text analysis. *JASIS*, 26(1):33–44, 1975.
13. H. Turtle and J. Flood. Query evaluation: Strategies and optimizations. *IP&M*, 31(6):831–850, 1995.
14. I. H. Witten, A. Moffat, and T. C. Bell. *Managing Gigabytes: Compressing and Indexing Documents and Images*. Morgan Kaufmann Publishers, San Francisco, CA, 1999.