Detecting co-derivative documents in large text collections

Jan Pomikálek, Pavel Rychlý

Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno, Czech Republic
{xpomikal, pary}@fi.muni.cz

Abstract

We have analyzed the SPEX algorithm by Bernstein and Zobel (2004) for detecting co-derivative documents using duplicate n-grams. Although we totally agree with the claim that not using unique n-grams can greatly increase the efficiency and scalability of the process of detecting co-derivative documents, we have found serious bottlenecks in the way SPEX finds the duplicate n-grams. While the memory requirements for computing co-derivative documents can be reduced to up to 1 % by only using duplicate n-grams, SPEX needs about 40 times more memory for computing the list of duplicate n-grams itself. Therefore the memory requirements of the whole process are not reduced enough to make the algorithm practical for very large collections. We propose a solution for this problem using an external sort with the suffix array in-memory sorting and temporary file compression. The proposed algorithm for computing duplicate n-grams uses a fixed amount of memory for any input size.

1. Introduction

Large text collections usually contain duplicate and co-derivative documents[1]. Identifying duplicates and co-derivatives is an important task, which enables both the detection of plagiarism and eliminating undesirable duplicate data from text collections in order to make them more useful for information retrieval tasks, different kinds of processing and analysis.

While full-duplicates can be easily detected using simple fingerprinting methods, detecting near-duplicate and co-derivative documents is a challenging task. Many researchers tackle the problem with algorithms based on shared word n-grams – contiguous sequences of (n) words. The idea is that if two documents share a number of n-grams of a non-trivial length (say five or more), it is very unlikely that they have been created independently.

Broder (2000) defined the document similarity formally with the resemblance measure. Let S_D be a set of all n-grams in a document D. Then the resemblance $r(A, B)$ of documents A and B is defined as

$$r(A, B) = \frac{|S_A \cap S_B|}{|S_A \cup S_B|}$$

A naive approach to co-derivative documents detection which computes resemblance for all document pairs in a collection is infeasible for large collection as the number of all unique pairs is enormous.

A more efficient approach involves using the inverted index of n-grams. For each n-gram (or its fingerprint), the list of documents is remembered in which the n-gram occurs. Using this structure, for an input document D, it is possible to efficiently determine which documents in the collection share at least one n-gram with D and compute their resemblances with D. The number of these documents should be reasonably small if the n (the length of the n-grams) is large enough. This technique is referred to as full-fingerprinting if all the n-grams in the collection are used. The problem is that for large n the number of unique n-grams is proportional to the number of words in the collection and therefore for large collections the index can easily exceed the normal capacity of RAM.

In order to reduce the memory requirements of this technique, different algorithms have been described for selecting small n-gram samples from documents, which can be used for estimating the resemblance – an overview is given in e.g. (Hoad and Zobel, 2003). Intuitively, if two documents share a significant number of n-grams then if we take a sample of each according to the same criteria, such as the k lowest in an arbitrary ordering of the n-grams, it is likely that the samples will share a number of n-grams too. The key question here is which n-grams should be selected for the sample. Different approaches to this problem exist, such as selecting random n-grams, taking each k-th n-gram, using modulo heuristic, etc. N-gram samples can be effectively used for detecting co-derivative documents which are close to duplicates since the probability that the samples will match is high. However, document pairs which differ somewhat more despite sharing significant pieces of text can easily be overlooked.

2. SPEX algorithm

Bernstein and Zobel (2004) introduced a lossless algorithm for n-gram selection called SPEX, which is equivalent to full-fingerprinting in terms of effectiveness. The key idea is as follows. The n-grams which occur only once in a collection are not of any use for detecting duplicates and therefore they can be safely removed from the index. More importantly, unique n-grams constitute a major part in text collections and removing them reduces the size of the index significantly.

Bernstein and Zobel (2004) analyzed the n-grams of size eight in the LATimes newswire collection. The analysis revealed that only 907,981 8-grams out of 67,808,917 were duplicated, i.e. less then 1.5 %. We have observed similar results on the data collections we used in our experiments.

The question remains how to identify the duplicate n-grams efficiently. The SPEX is based on a simple idea. If an n-grams S is unique in the collection, then an $(n + 1)$-gram

[1] Co-derivative documents are pairs of documents which share a significant amount of text that has been derived from one into the other, or derived from a third document into both.

n	N_n^{unq}	N_n^{prn}	T_n^{dup}	N_n^{dup}	M_n
1	191,299	0	278,079	93,751,313	469,378
2	10,130,676	373,298	4,603,645	83,807,882	14,639,102
3	40,876,464	17,291,952	8,061,474	53,058,040	**36,249,631**
4	70,777,744	55,301,874	5,881,395	23,152,706	29,418,739
5	85,278,904	80,918,395	2,945,432	8,647,492	13,187,336
6	90,085,317	89,244,690	1,555,553	3,837,025	5,341,612
7	91,442,538	91,309,145	1,080,612	2,475,750	2,769,558
8	91,864,665	91,842,834	917,628	2,049,569	2,020,071
9	92,047,840	92,041,981	844,990	1,862,340	1,768,477
10	92,158,604	92,155,864	800,768	1,747,522	1,648,498

Table 1: SPEX iterations on BNC

which contains S must be unique too. For example, if the word *nascent* exists only once in the collection then the bigram *nascent effort* cannot be duplicated in the same collection. SPEX builds the lists of duplicate n-grams iteratively, starting from a list of duplicate unigrams (single words). In each iteration a number of n-grams which would otherwise have had to be remembered can be forgotten based on the results of the previous iteration – we know that they cannot be duplicated. The authors claim that this keeps the memory requirements constantly low throughout all the iterations. Unfortunately, this is not entirely true as we show in the next section.

3. SPEX analysis on BNC

In order to get a rough idea of the memory requirements of the SPEX algorithm on large, real data collections, we have analyzed its iterations on the British National Corpus (BNC). BNC is a 100 million words corpus of English texts. For each iteration n, we computed the following values:

- N_n^{unq} – Number of unique n-grams (this is equal to the number of unique n-gram types).
- N_n^{dup} – Number of duplicate n-grams (each n-gram being computed as many times as it occurs in the collection).
- T_n^{dup} – Number of duplicate n-gram types.
- N_n^{prn} – Number of (unique) n-grams which could be pruned based on the results of the previous iteration.
- M_n – Number of n-gram types which had to be remembered in this iteration. This includes the duplicate n-grams, the unique n-grams which could not be pruned and the duplicate $(n-1)$-grams from the previous iteration. Therefore $M_n = T_n^{dup} + (N_n^{unq} - N_n^{prn}) + T_{n-1}^{dup}$.

The results are presented in Table 1. For 8-grams, there are 917,628 duplicate types out of 93,782,293 in BNC, i.e. about 1 %. Slightly less than the SPEX authors reported for the LATimes collection, this is probably due to the fact that BNC contains no duplicate documents. At any rate, the results confirm the claim that the number of duplicate n-grams is much lower than the number of unique n-grams in collections of texts (containing a reasonable amount of duplicate documents), especially for large n-s.

It can also be seen that the number of n-grams to be remembered by the algorithm (M_n) decreases as n grows. This indicates that more iterations could be made without having to sacrifice additional memory resources. However, on close inspection, Table 1 reveals that the third iteration constitutes a serious bottleneck in the whole process. Although the number of duplicate triple types is reasonably low (8,061,474), the problem here is that not enough unique triples are pruned. This is obviously due to the fact that many unique trigrams contain duplicated bigrams. As a result, 36,249,631 n-grams must be remembered at this stage of the algorithm. This means that compared to computing an inverted index directly, for large n-grams such as 8-grams or 10-grams, using the SPEX algorithm the memory requirements cannot be reduced to values as low as around 1 % but only to about 40 %. Similar results have also been observed in other data, which we have used in our experiments.

4. Finding duplicate n-grams

The main idea of the SPEX algorithm (using only the duplicate n-grams) is very good and can indeed reduce the size of the index to as little as 1 %. However, computing the list of duplicate n-grams with SPEX is too expensive. We herby propose a more efficient algorithm.

The core idea of our algorithm is to use an external sort method to generate all n-grams together with their counts directly, in one step. The program splits the input text into chunks which fit into a fixed amount of memory. For each chunk, a sorted list of n-grams is generated and saved to a temporary file on disk. The final phase joins all temporary files and outputs any n-gram with total count higher than one. This is a typical external sort method which would require a huge amount of disk space and a lot of runs (chunks) to process the whole input text.

We use two novel techniques: suffix array in-memory sorting to decrease the number of chunks and temporary file compression to decrease the required disk space. Both techniques use word enumeration to work with integer numbers instead of character strings. A word lexicon of the whole input text is built during incremental processing of chunks. The lexicon maps words as character strings into unique ID

Stop-list size	Input size					
	1 million		10 million		100 million	
0	19797	100.0 %	303681	100.0 %	5916799	100.0 %
15	14871	75.1 %	225589	74.2 %	4465588	75.4 %
30	13872	70.0 %	203530	67.0 %	4052601	68.4 %
100	12449	62.8 %	173888	57.2 %	3482515	58.8 %
300	10861	54.8 %	146818	48.3 %	2961468	50.0 %
600	9585	48.4 %	126372	41.6 %	2581827	43.6 %

Table 2: Number of 10-gram duplicate types depending on input size and stop-list size

numbers during the first phase and IDs back to words during the final output of duplicate n-grams. Temporary file compression uses Elias codes (delta and gamma (Witten et al., 1999)) to store word IDs and omitting the longest common prefix with the previous n-gram. Using this technique, the size of temporary files containing all 10-grams with its counts is about three times bigger than the size of the input text.

Creating a suffix array of tokenized text (sequence of word IDs) is used to generate sorted list of n-grams for each input chunk. We use a linear algorithm for suffix sorting (Larsson and Sadakane, 1999) which requires only 8 bytes per token. Another 4 bytes per token is used for the original token sequence to output the list of n-grams. Both techniques and the final join are very fast, which we show in the next section.

5. Experimental results

In order to evaluate the scalability of the proposed algorithm, we have used the biggest text collection that we have available – the BiWeC corpus. BiWeC (Big Web Corpus) is an ongoing project which aims at creating a collection of English texts exceeding the size of 20 billion words. In order to collect such a large amount of data we use a web crawler. In the first stage of the project we managed to download 13.86 million web pages. We applied the BTE algorithm (Finn et al., 2001) to remove HTML markup and boilerplate from the pages. After filtering out full-duplicates using MD5 checksums we ended up with 6.73 million of plain text documents. This collection contained 9.29 billion tokens.

We took several BiWeC samples of various sizes and ran our algorithm on these samples as well as on the full collection. We used a 2.4 GHz machine with 32 GB of RAM, but no more than 1.5 GB was used by the process even for the whole collection. The results are reported in the Table 3. It can be seen that the computation time is close to linear in the size of the input data. The computation is feasible even for an input of more than 9 billion tokens, though it takes about 18 hours on a single machine. We estimate the running time on the full BiWeC data (20 billion tokens) to be less than 72 hours, which is acceptable for a one-time computation.

We also experimented with using a stop-list. The stop-list we used was represented by the most frequent words in the corpus. We removed stop-list words from each document before extracting n-grams. This led to reducing the number of n-grams to work with and consequently improved the running times of the algorithm. The results from the Table 3 were achieved using a stop-list of the 600 most frequent words.

Though we do not have any evidence, we believe that not using stop-list words should not harm the performance of the deduplication algorithm, assuming the size of the stop-list is reasonable. Rather, even better performance might be achieved as the number of n-grams decreases which occur in two independent documents by chance. It remains to be seen how many stop-list words can be left out before the results of the algorithm get worse. This is a subject of our further research.

Table 2 shows how the number of duplicate n-grams decreases with increasing size of the stop-list. The results reveal that the number of duplicate n-grams can be reduced to a half by using 600 stop-words. This remains true for any size of the input.

Input text size in millions of tokens	Running time	# of 10-gram duplicate types
1	1.4s	9 585
10	13.6s	126 372
100	2m48s	2 581 827
1 000	38m21s	31 266 338
9 292	18h19m	403 920 712

Table 3: Results of duplicate n-grams detection using external sort

6. Conclusion

The SPEX algorithm builds an inverted index for n-grams using only those n-grams which occur at least twice across the whole data collection. This index is as useful for detecting co-derivative documents as the full index is, while being about 100 times smaller. However, we have showed that the memory requirements for computing the list of duplicate n-grams are about 40 % of the size of the full index. This makes the algorithm impractical for very large collections.

We have presented an algorithm which can compute the list of duplicate n-grams using a fixed amount of memory while at the same time requiring a reasonable amount of CPU time. This improvement to the SPEX makes it possible to build an inverted index for duplicated n-grams and consequently identify co-derivative documents efficiently using as little as 1 % of the memory required for building a full inverted index.

We have demonstrated that the proposed algorithm can process more than 9 billion tokens within 18.5 hours on a 2.4 GHz machine with less than 2 GB RAM. As the memory requirements of the algorithm are constant and the time complexity is close to linear the algorithm should be practical even for processing much larger amounts of texts.

7. References

Y. Bernstein and J. Zobel. 2004. A scalable system for identifying co-derivative documents. *Proc. String Processing and Information Retrieval Symp.*, pages 55–67.

A.Z. Broder. 2000. Identifying and filtering near-duplicate documents. *Proceedings of the 11th Annual Symposium on Combinatorial Pattern Matching*, pages 1–10.

A. Finn, N. Kushmerick, and B. Smyth. 2001. Fact or fiction: Content classification for digital libraries. In *DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries*.

T.C. Hoad and J. Zobel. 2003. Methods for identifying versioned and plagiarized documents. *Journal of the American Society for Information Science and Technology*, 54(3):203–215.

N.J. Larsson and K. Sadakane. 1999. Faster suffix sorting. Technical Report LU-CS-TR:99-214, Lund University, Sweden.

I.H. Witten, T.C. Bell, and A. Moffat. 1999. Managing Gigabytes: Compressing and Indexing Documents and Images.