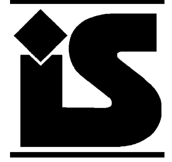
Available online at www.sciencedirect.com

SCIENCE DIRECT

Information Systems 31 (2006) 595–609



www.elsevier.com/locate/infosys

Accurate discovery of co-derivative documents via duplicate text detection ☆

Yaniv Bernstein*, Justin Zobel

School of Computer Science and Information Technology, RMIT University, Melbourne, Australia

Abstract

Documents are co-derivative if they share content: for two documents to be co-derived, some portion of one must be derived from the other, or some portion of both must be derived from a third document. An existing technique for concurrently detecting all co-derivatives in a collection is document fingerprinting, which matches documents based on the hash values of selected document subsequences, or chunks. Fingerprinting is hampered by an inability to accurately isolate information that is useful in identifying co-derivatives. In this paper we present SPEX, a novel hash-based algorithm for extracting duplicated chunks from a document collection. We discuss how information about shared chunks can be used for efficiently and reliably identifying co-derivative clusters, and describe DECO, a prototype package that combines the SPEX algorithm with other optimisations and compressed indexing to produce a flexible and scalable co-derivative discovery system. Our experiments with multi-gigabyte document collections demonstrate the effectiveness of the approach.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Fingerprinting; Duplicate detection; Hashing

1. Introduction

Many document collections contain sets of documents that are *co-derived*. Examples of co-derived documents include plagiarised documents, document revisions, and documents written by amending a template. Knowledge of co-derivative document relationships in a collection can be used for returning more informative results from search engines, detection of plagiarism, and management of document versioning in an enterprise.

Depending upon the application, we may wish to identify all pairs of co-derived documents in a given collection (the $n \times n$ or *discovery* problem) or only those documents that are co-derived with a specified query document (the $1 \times n$ or *search* problem). We focus in this research on the more difficult discovery problem. While it is possible to naïvely solve the discovery problem by repeated application of an algorithm for solving the search problem, such an application becomes too time-consuming for practical use.

Though the task of detecting co-derivative documents is superficially similar to that of document search or categorisation, there are marked differences. Ranking and categorisation are concerned

☆This article is based on a conference presentation: Y. Bernstein, J. Zobel, A scalable system for identifying co-derivative documents, Proceedings of the String Processing and Information Retrieval Symposium, October 2004, Padua, Italy, pp. 55–67.

*Corresponding author.

E-mail addresses: ybernste@cs.rmit.edu.au (Y. Bernstein), jz@cs.rmit.edu.au (J. Zobel).

0306-4379/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.is.2005.11.006

with the semantics of documents, while co-derivative detection is concerned with a document's syntactic structure. While independently authored documents can have similar semantics (student essays on the same topic are an example), it is exceedingly unlikely for documents from different sources to have the same syntactic structure.

Existing feasible techniques for solving the discovery problem are based on document fingerprinting, in which a compact representation of a selected subset of contiguous text *chunks* occurring in each document—its *fingerprint*—is stored. Pairs of documents are identified as possibly co-derived if enough of the chunks in their respective fingerprints match. Fingerprinting schemes differ primarily in the way in which chunks to be stored are selected.

In this paper we introduce SPEX, a novel and efficient algorithm for identifying those chunks that occur more than once within a collection. We present the DECO package, which uses the shared-chunk indexes generated by SPEX as the basis for accurate and efficient identification of co-derivative documents in a collection. We show that DECO effectively addresses some of the deficiencies of existing approaches to this problem. Using several collections, we experimentally demonstrate that DECO is able to reliably and accurately identify co-derivative documents within a collection while using fewer resources than previous techniques of similar capability. Our results also suggest that DECO scales well to large collections.

2. What is co-derivation?

We consider two documents to be co-derived if some portion of one document is derived from the other, or some portion that is present in both documents is derived from a third. The notion of co-derivation is in many ways analogous to the idea of a genetic or 'blood' relationship in a human family.

While the above is an intuitive and appealing definition, it is purely qualitative. It tells us nothing of how to detect co-derivation, or even what characteristics we expect a pair of co-derived documents to have. Formulating such a quantitative definition is not straightforward.

Broder [1] defines two measures of co-derivation—*resemblance* and *containment*—in terms of the number of *shingles* (we shall use the term *chunks*) a pair of documents have in common. A chunk is defined by Broder as 'a contiguous subsequence'; that is, each chunk represents a contiguous set of words or characters within the document. An example chunk of length six taken from this paper would be 'each chunk represents a contiguous set'. The intuition is that, if a pair of documents share a number of such chunks, then they are unlikely to have been created independently. Such an intuition is what underlies fingerprinting-based approaches, described later.

Two difficult issues in defining co-derivation are boilerplate and template text. Boilerplate text appears in many documents that are otherwise of separate heritage. One example of such text is the GNU Public License[1] (GPL), which is a tract of licensing text that is included in many items of Linux documentation that were otherwise written separately. While the GPL is clearly the work of a single group of authors, does its inclusion in two works mean that they are co-derived? Template text is used as the basis for the creation of a certain class of document: Sanderson [2] cites an example of financial reports in a collection of Reuters newswire documents that seemed nearly identical but in fact referred to different events. Should the presence of boilerplate or template text in two documents mean they are co-derived? The answer to this and other such questions is ultimately application-dependent.

3. The relationship graph

We introduce the concept of a *relationship graph* for representing and analysing co-derivation relationships within a collection. In a relationship graph for a given collection, each document is represented by a vertex. A co-derivation relationship between a pair of documents is indicated by the presence of an edge between the vertices representing these documents. The relationship graph emphasises the essentially pairwise nature of the co-derivation relationship, and allows for easy visualisation and analysis of text reuse patterns within a collection.

In this context, it becomes clear that the task of the discovery problem is to identify the structure of this graph, whereas the task of the search problem is to identify the set of edges incident upon a particular vertex.

Fig. 1 shows a hypothetical relationship graph for a collection of eight documents. It shows a co-derivation relationship between documents 3 and 4, and a triangular co-derivation relationship between documents 2, 5, and 7. Document 6 is co-derived

[1]See www.gnu.org/copyleft/gpl.html

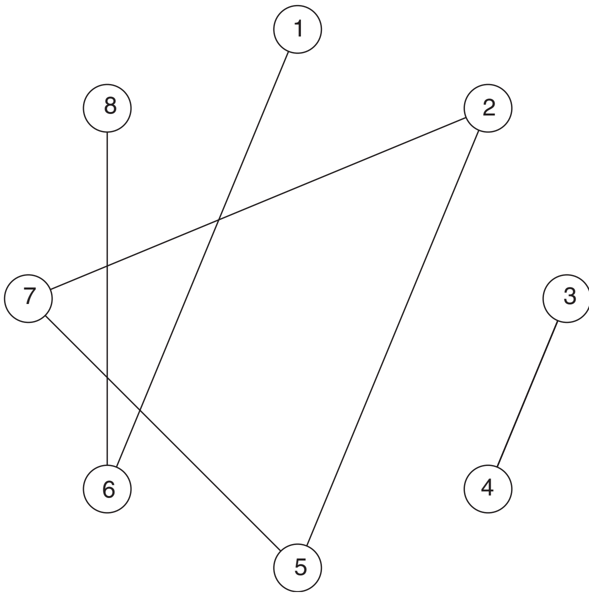


Fig. 1. An example of a relationship graph for a collection of eight documents.

with documents 1 and 8, but they are not co-derived with each other. This means that documents 1 and 8 have commonality with *different* parts of document 6; perhaps each contains an excerpt from document 6, or maybe document 6 is an aggregation of a number of different documents. Note that this graph is for illustration only: in most practical cases we would expect the relationship graph to be far sparser than this, with a significant number of vertices having degree zero; that is, not being co-derived with any other document in the collection.

Note that, as the possible number of edges in a graph is quadratic in the number of nodes, the task of discovering the structure of the relationship graph is a formidable one: for example, a collection of 100,000 documents contains nearly 5 billion unique document pairings. Clearly, it is infeasible to individually investigate this number of relationships. We must rely on intelligent processing and the relative sparsity of the relationship graph in most situations[2] in order to attempt the discovery problem.

[2] In most cases there will be many documents in a collection that are not co-derived at all. However, even if this is not so, it is reasonable to assume that most documents are co-derived with at most a few others in the collection. The low order of vertices in the graph will therefore ensure that the graph is increasingly sparse.

4. Existing work: strategies for co-derivative discovery

There are several approaches to solving the search problem, most of which can be categorised as being either relative-frequency or fingerprinting methods:

4.1. Relative-frequency techniques

Relative-frequency techniques such as relative frequency matching [3] and the identity measure [4] are based on the intuition that co-derived documents are likely to contain the same words with similar frequencies. In particular, identical documents will contain an identical word-frequency profile, and insertions and deletions will only gradually degrade this correlation. Thus, relative word frequencies should constitute a robust signifier of the co-derivation of documents.

In comparisons between relative-frequency methods and fingerprinting [3,4], the relative-frequency methods tended to more reliably identify co-derived documents, though there was also a higher rate of false positives. It is worth noting that the comparisons were carried out by the proponents of these systems.

These results suggest that relative frequency approaches are a good choice for the search problem. However, the only computationally feasible algorithms for the discovery problem to date have used the process of document fingerprinting.

4.2. Fingerprinting

The key observation underlying document fingerprinting [5–8,4] mirrors that behind the definitions of Broder [1]: if documents are broken into small contiguous chunks, then co-derivative documents are likely to have a large number of these chunks in common, whereas independently derived documents with overwhelming probability will not. Fingerprinting algorithms store a selection of chunks from each document in a compact form and flag documents as potentially co-derived if the number of chunks their fingerprints have in common exceeds a specified threshold.

While fingerprinting algorithms vary in many details, their basic process is as follows: documents in a collection are parsed into units (typically either characters or individual words); representative chunks of contiguous units are selected through

the use of a heuristic; the selected chunks are then hashed for efficient retrieval and compact storage; the hash-keys, and possibly also the chunks themselves, are then stored, often in an inverted index structure [9]. The index of hash-keys contains all the fingerprints for a document collection and can be used for the detection of co-derivatives.

The principal way in which document fingerprinting algorithms differentiate themselves is in the choice of selection heuristic, that is, the method of determining which chunks should be selected for storage in each document's fingerprint. The range of such heuristics is diverse, as reviewed by Hoad and Zobel [4]. The simplest strategies are full selection, in which every chunk is selected, and random selection, where a given proportion or number of chunks is selected at random from each document to act as a fingerprint. Other strategies pick every nth chunk, or only pick chunks that are rare across the collection [7]. Taking a different approach is the *anchor* strategy [5], in which chunks are only selected if they begin with certain pre-specified combinations of letters. Simpler but arguably as effective is the *modulo* heuristic, in which a chunk is only selected if its hash-key *modulo* a parameter k is equal to zero. The *winnowing* algorithm of Schleimer [10] passes a window over the collection and selects the chunk with the lowest hash-key in each window. Both the *anchor* and *modulo* heuristics ensure a level of synchronisation between fingerprints in different documents, in that if a particular chunk is selected in one document, it will be selected in all documents.

4.2.1. Fingerprinting for the search problem

The task of the search problem is to identify the documents in the collection that are co-derived with a given query document. When the system is presented with a candidate or query document, it once again uses a selection heuristic to select chunks from this document. This can be done either by using the same selection heuristic as is used for archiving documents, or a separate—typically more exhaustive—selection heuristic. Each of the chunks selected from the query document is then hashed, and the corresponding postings lists retrieved from the fingerprint index. All the retrieved postings lists will be merged, sorted and aggregated, thus determining the number of common hash-keys the fingerprint of each document has with the query document. Any document containing more than a user-specified number of hash-keys in common (the threshold) with the query document will be considered by the algorithm to be co-derived.

For example, if the phrases selected from the query document were *fo*, *ob*, and *ar*, then each of those phrases would be hashed and the postings list at those locations in the hashtable retrieved. Let us say that the postings list for the three phrases above were `<fo: 3, 6, 9>`, `<ob: 2, 4, 6>` and `<ar: 6, 9, 11>`, respectively. After aggregating the number of fingerprints that the fingerprint of each document shares with the query document, we get the following list of tuples: { (2: 1) (3: 1) (4 : 1)(6 : 3)(9 : 2)(11 : 1)}. If the user were to specify a cutoff of two matches, then the system would flag the documents 6 and 9 as co-derived with the query document.

In their comparative experiments, Hoad and Zobel [4] found that few of the fingerprinting strategies tested could reliably identify co-derivative documents in a collection. Of those that could, Manber's *anchor* heuristic was the most effective, but its performance was inferior to their relative-frequency based identity measure system. Similarly, Shivakumar and García-Molina [3] found that the COPS fingerprinting system [6] was far more likely than their SCAM ranking-based system to fail to identify co-derivative documents.

4.2.2. Fingerprinting for the discovery problem

There have been several previous explorations of using document fingerprinting for the discovery problem:

Manber [5] counts the number of identical postings lists in the chunk index, arguing this can be used to identify clusters of co-derived documents in the collection. However, as Manber points out, there are many cases in which the results produced by his method can be extremely difficult to interpret.

Broder et al. [8] describe an approach in which each postings list is broken down to a set of document-pair tokens, one for each possible pairing in the list. For example, a postings list

`<quick brown fox : 3,7,9,13>`

would be expanded to

`<3,7> <3,9> <3,13> <7,9> <7,13> <9,13>`.

All these document-pair tokens are then sorted and aggregated so that each pair of documents has associated with it the number of chunks the two documents share. This count is then used as the basis for a set of discovery results. Broder et al. [8]

show that the ratio of shared chunks to the total stored chunks for the two documents provides an unbiased estimator for the resemblance measure. However, the variance of this estimator is not discussed.

While this approach usually yields far more informative results than that of Manber [5], taking the Cartesian product of each postings list means that the number of tokens generated is quadratic in the length of the list; this can easily cause resource blowouts and introduces serious scalability problems for the algorithm.

The *probabilistic counting* technique was used for iceberg database queries [11] and was later applied by the same team to the task of finding co-derivatives [12]. The process is very simple: for each document pair, instead of storing a token, the pair is hashed and a counter at the relevant field in a hashtable is incremented. A second pass generates a list of candidate pairs by discarding any pair that hashes to a counter that recorded insufficient hits. Assuming the hashtable is of sufficient size, this pruning significantly reduces the number of tokens that must be generated for the exact counting phase.

4.3. Redundancy and lossiness in fingerprinting algorithms

A fundamental weakness of fingerprinting strategies is that they cannot identify and discard chunks that do not contribute towards the identification of any co-derivative pairs. Unique chunks form the vast majority in most collections, yet do not contribute toward solving the discovery problem. We analysed the *LATimes* newswire collection (see Section 7) and found that out of a total of 67,808,917 chunks of length eight, only 2,816,822 were in fact instances of duplicate chunks: less than 4.5% of the overall collection. The number of distinct duplicated chunks is 907,981, or less than 1.5% of the collection total.

The inability to discard unused data makes full fingerprinting too expensive for most practical purposes. Thus, it becomes necessary to use chunk-selection heuristics to keep storage requirements at a reasonable level. However, this introduces *lossiness* to the algorithm: current selection heuristics are unable to discriminate between chunks that suggest co-derivation between documents in the collection and those that do not. There is a significant possibility that two documents sharing a large portion of text are passed over entirely.

For example, Manber [5] uses character-level granularity and the *modulo* selection heuristic with $k = 256$. Thus, any chunk has an unbiased one-in-256 chance of being stored. Consider a pair of documents that share an identical 1 KB (1024 byte) portion of text. On average, four of the chunks shared by these documents will be selected. Using the Poisson distribution with $\lambda = 4$, we can estimate the likelihood that C chunks are selected as $P(C = 0) = e^{-4} \cdot 4^0/0! = 1.8\%$ and $P(C = 1) = e^{-4} \cdot 4^1/1! = 7.3\%$. This means that a pair of documents containing a full kilobyte of identical text have nearly a 2% chance of not having a single hash-key in common in their fingerprints, and a greater than 7% chance of only one hash key in common. The same results obtain for an identical 100-word sequence with a word-level chunking technique and $k = 25$, as used by Broder et al. [8]. Such lossiness is unacceptable in many applications.

Schleimer et al. [10] make the observation that the *modulo* heuristic provides no guarantee of storing a shared chunk no matter how long the match. Whatever the match length, there is a nonzero probability that it will be overlooked. Their winnowing selection heuristic is able to guarantee that any contiguous run of shared text greater than a user-specifiable size w will register at least one identical hash-key in the fingerprints of the documents in question. However, a document that contains fragmented duplication below the level of w can still escape detection by this scheme: it is still fundamentally a lossy algorithm.

4.3.1. Algorithms for lossless fingerprinting

We make the observation that, as only chunks that occur in more than one document contribute towards identifying co-derivation, a selection strategy that selected all such chunks would provide functional equivalence to full fingerprinting, but at a fraction of the storage cost for most collections. The challenge is to find a way of efficiently and scalably discriminating between duplicate and unique chunks.

The simplest way to eliminate the redundant unique phrases in a fingerprint index is as a postprocessing step; perform full fingerprinting, then scan through the resulting index to eliminate all entries that contain just one reference. The problem with this technique is that, although the resulting index is minimal, the peak resource usage

is as high as if the technique were not applied. This means that we need as much disk space for this method as for full fingerprinting. Clearly, we need a method that is able to identify duplicate chunks at an earlier stage in the fingerprinting process.

Hierarchical dictionary-based compression techniques such as SEQUITUR [13] and RE-PAIR [14] are primarily designed to eliminate redundancy by replacing strings that occur more than once in the data with a reference to an entry in a ruleset. Thus, passages of text that occur multiple times in the collection are identified as part of the compression process. This has been used as the basis for phrase-based collection browsing tools such as PHIND [15] and RE-STORE [16]. However, the use of these techniques in most situations is ruled out by their high memory requirements: the PHIND technique needs about twice the memory of the total size of the collection being browsed [15]. To keep memory use at reasonable levels, the input data is generally segmented and compressed block by block; however, this negates the ability of the algorithm to identify globally duplicated passages. Thus, such algorithms are not useful for large collections.

Suffix trees are another potential technique for duplicate-chunk identification, and are used in this way in computational biology [17]. However, the suffix tree is an in-memory data structure that consumes a quantity of memory equal to several times the size of the entire collection. Thus, this technique is also only suitable for small collections.

5. The SPEX algorithm

Our contribution in this work is the SPEX algorithm, a resource-efficient technique for lossless chunk selection. The SPEX algorithm is a novel hash-based method for duplicate-chunk extraction and has far more modest and flexible memory requirements than the algorithms discussed in Section 4.3 and is thus the first selection algorithm that is able to provide *lossless* chunk selection within large collections. In the case of large collections, the memory needs of SPEX are in most cases many times smaller than the size of the collection. SPEX identifies repetition by use of repeated passes over the data, and thus is slower than some of the alternatives, but is, as we show, more accurate for a given space overhead.

The fundamental observation behind the operation of SPEX is that it is only possible for a chunk to be a duplicate if all subchunks of that chunk are also duplicates. Thus, we only need to demonstrate the uniqueness of one subchunk in order to discount the possibility that a chunk as a whole is non-unique. For example, if the chunk 'quick brown' occurs only once in the collection, there is no possibility that the chunk 'quick brown fox' is repeated. SPEX uses an iterated hashing approach to discard unique chunks and leave only those that are likely to be duplicates. This iterative approach has shared facets with data mining algorithm such as a priori [18] that find various data relationships by iteratively refining from more general relationships.

Our iterated approach begins with subchunks of length one—that is, individual words. The collection is linearly parsed and each word is added to a large hash-based accumulator. The hash-based accumulator (or hashcounter) is simply a large array of counters. When an item is added to the hashcounter, the location in the table corresponding to the hash of the item is incremented. Two particulars are noteworthy: first, collisions are not resolved, guaranteeing constant-time performance at the expense of the possibility of false positives; second, the counters need take on only three distinct values (zero, one, and 'greater than one'), thus allowing each field to occupy only two bits of memory. This means that a large hashcounter can fit into a relatively modest quantity of memory: 64 MB of memory is sufficient for over 256 million fields.

The number of words in a collection of documents is in most cases relatively modest relative to the overall size of the collection: the highly skew distribution of word use is a well-studied phenomenon [19]. This, combined with the large size of the hashcounter, means that the number of collisions at this stage will be minimal. After the hashcounter has been fully populated, querying it on a particular word in the collection will return either a one or a 'greater than one'. If the returned count is one, then that word is unique in the collection. If it is greater than one, then it is assumed to occur multiple times, though false positives are possible due to hashing collisions.

Once the initial hashcounter for chunks of length one has been fully populated from the collection, a new hashcounter is initialised. A second pass of the collection begins, this time sequentially extracting all two-word chunks. Rather than simply inserting

each chunk into the new hashcounter, it is first broken down into two subchunks of size one. Each of these subchunks is used as a query against the first hashcounter; if either of the words is marked as unique in the collection, then the chunk is not inserted into the second hashcounter: as one of its subcomponents is unique, it too must be unique.

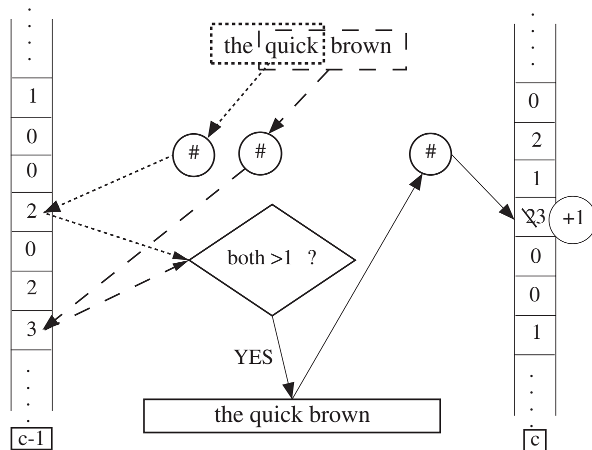


Fig. 2. The process for inserting a new chunk into the hashcounter in SPEX. The chunk 'the quick brown' is divided into two sub-chunks 'the quick' and 'quick brown'. They are each hashed into the old hash table. If the count for both sub-chunks is greater than one, the full chunk is hashed and the counter at that location in the new hashcounter is incremented.

After the second pass, the process is repeated for chunks of length three: each of these chunks is broken into two subchunks of length two, which are queried against the hashcounter for chunks of this length. Note that the memory for the initial hashcounter can now be reclaimed, as the information it contains is no longer needed. At any stage there need be no more than two hashcounters in memory. Fig. 2 provides an illustration of one iteration of the SPEX process.

The process is iterated until we reach the desired chunk length; we use a length of eight words. When the desired length is reached, an inverted index of shared chunks can be created by only indexing chunks whose subchunks all hash to fields that indicate multiple hits. The entire SPEX process is described more formally in Algorithm 1.

In each iteration of the SPEX algorithm, the chunks become longer, and thus the cost of hashing each chunk grows. However, a larger factor in determining execution time is the number of chunks that must be hashed. We have empirically verified that the execution time of each iteration closely mirrored the number of hashcounter insertions made.

Algorithm 1 The SPEX algorithm

```
1:  // C: Collection of chunks
2:  // l: Target chunk length
3:  // c_n: chunk of length n
4:  // c_n{p...q}: The subchunk composed of words p through q of chunk c_n
5:  // #(c): The hash value of chunk c
6:  // h_n: Hashcounter for chunks of length n
7:
8:  for all c_1 ∈ C do
9:    h_1[#(c_1)] ← h_1[#(c_1)] + 1
10:   end for
11:   for n ∈ [2, l] do
12:   for all c_n ∈ C do
13:     if h_{n-1}[#(c_n{1...n-1})] > 1 and h_{n-1}[#(c_n{2...n})] > 1 then
14:       h_n[#(c_n)] ← h_n[#(c_n)] + 1
15:     end if
16:   end for
17:   end for
```

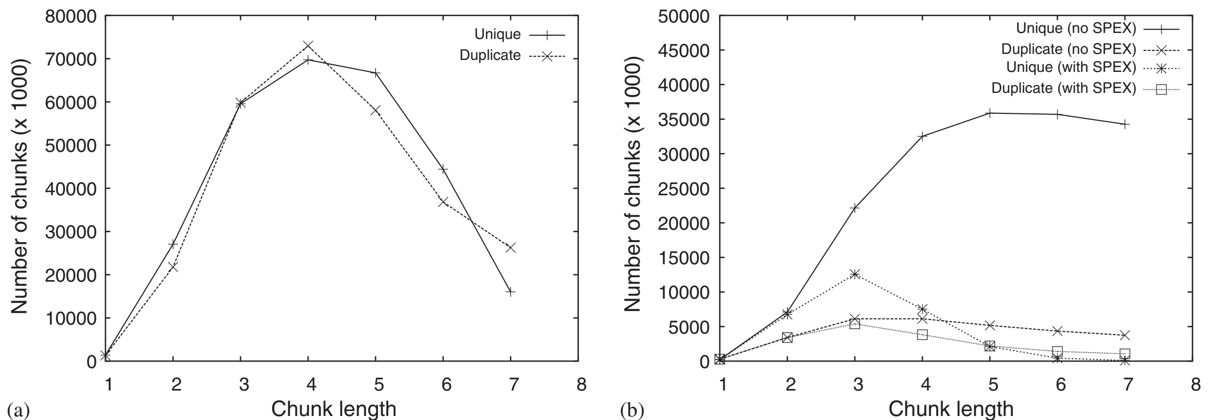


Fig. 3. Number of hashcounter elements containing single and multiple hits for: (a) the *all-newswire* collection (2^{26} elements total); (b) the *LATimes* collection with and without SPEX iterative refinement (2^{23} elements total).

Fig. 3 illustrates how SPEX works. Part (a) tracks the occupancy of the hashcounter after each pass of the algorithm for the *all-newswire* (see Section 7) collection. The solid line shows the number of fields in the hashcounter that recorded just a single hit (corresponding to unique chunks), whereas the dashed line shows the number of fields in the hashcounter that recorded multiple hits (non-unique chunks or hash collisions). As discussed earlier, the number of single-word chunks (both unique and duplicated) is low. This means that SPEX does not yet have much discriminative power: only chunks that contain a unique subchunk are discarded by the algorithm and there are not that many unique subchunks at this stage. However, as the number of unique chunks grows in subsequent passes, more can be discounted. The compounding effect of discounting chunks at each pass eventually means that the overall number of occupied hashcounter entries begins to drop dramatically, keeping the hashcounter from becoming flooded.

Fig. 3(b) shows what happens if the SPEX process of stepwise refinement is not used and all chunks are inserted into the hashcounter at each stage. At the first couple of passes, the lines track quite closely. However, at chunk size three, they begin to diverge dramatically. Where SPEX is not used, the unique chunks are not weeded out. They therefore begin to occupy more and more space in the hashcounter, thus increasing the possibility of collisions. At chunk size of seven, the hashcounter where SPEX was not used is showing a significantly higher number of fields that had recorded more than one hit. As the number of true duplicate chunks remains the same, these extra fields are false positives created by hash collisions. This directly leads to the creation of a larger index.

6. The DECO package

Our DECO system for co-derivative detection is a software package that combines the SPEX algorithm with advanced indexing techniques, sophisticated scoring functions, and other previous innovations in the field.

DECO operates in two phases: index construction and relationship graph generation.

6.1. Index construction

DECO uses a multi-stage 'selection pipeline' during index construction in order to minimise index size and allow the user to adjust the tradeoffs between build time, size, and reliability. The first stage is to eliminate exact duplicates. These are identified by taking one hash of each entire document and then sorting these hashes. Documents with the same hash are considered to be identical, and only one of these documents is indexed. All documents that are co-derived with the 'representative' document are also flagged as co-derived with its duplicates. This exact technique was used by Broder et al. [8]. Though they did not quantify the savings made by eliminating duplicates, they can be quite significant. This is even more true in the case of SPEX, as every chunk in a pair of identical documents will be indexed, meaning that identical documents take up a disproportionate amount of space within the index. In order to minimise the possibility of collision—a collision

with one document could be a way for a document to avoid being detected as co-derived with another—we have chosen to use the reasonably secure MD5 message digest algorithm [20], which produces hash-keys 160 bits in length, for this phase.

DECO currently supports the SPEX and *modulo* selection schemes, which can be used individually or together in the selection pipeline. By default, SPEX is used. However, it can be combined with the *modulo* heuristic so that a chunk is only indexed if it is selected by both algorithms. This allows for the creation of compact indexes that still support identification of co-derivatives. The selection pipeline code was written in such a way as to facilitate the creation of additional 'plug-ins' for the pipeline. For example, one could easily write a new selection filter for the *anchor* heuristic, or a filter that selected chunks only if they contained certain words.

The low-level index operations in DECO are based on code from the Zettair[3] search engine being developed at RMIT University. Zettair implements flexible, optimised, compressed indexes using variable-byte encoding and various other techniques.

6.2. Relationship graph generation

DECO uses the *probabilistic counting* technique of Shivakumar and García-Molina [12] to efficiently keep scores for all the document pairs in the relationship graph generation phase.

Several parameters must be specified to guide this process: the most important of these are the *scoring function* and the *inclusion threshold*. DECO calculates the co-derivation score for a pair of documents u and v using one of the following formulae:

$$S_1(u,v) = \sum_{c \in u \wedge c \in v} 1, \quad S_2(u,v) = \sum_{c \in u \wedge c \in v} 1/\min \bar{u}, \bar{v},$$

$$S_3(u,v) = \sum_{c \in u \wedge c \in v} 1/\text{mean}\, \bar{u}, \bar{v},$$

$$S_4(u,v) = \sum_{c \in u \wedge c \in v} \frac{1/f_c}{\text{mean}\, \bar{u}, \bar{v}},$$

where $\bar{u}$ is the length (in words) of a document u, and f_c is the number of collection documents a given chunk c appears in. Function S_1 above simply counts the number of chunks common to the two documents; this elementary scoring method is how fingerprinting algorithms have worked up to now. Functions S_2 and S_3 attempt to normalise the score relative to the size of the documents, so that larger documents do not dominate smaller ones in the results. They are similar to the resemblance measure of Broder [1] but are modified for more efficient computation. Function S_4 gives greater weight to phrases that are rare across the collection. These scoring functions are all simple heuristics; further refinement of these functions and the possible use of statistical models is a topic for future research.

The inclusion threshold is the minimum value of $S(u,v)$ for which an edge between u and v will be included in the relationship graph. We wish to set the threshold to be such that pairs of co-derived documents score above the threshold while pairs that are not co-derived score below the threshold.

7. Experimental methodology

We seek to experimentally investigate two facets of the DECO package: the accuracy and reliability of the package in identifying co-derivative document pairs, and the scaling characteristics of the system.

Document collections: We make use of six document collections for our experiments. The *webdata + xml* and *linuxdocs* collections were accumulated by Hoad and Zobel [4]. The *webdata + xml* collection consists of 3307 web documents totalling approximately 35 MB, into which have been seeded 9 documents (the *XML documents*), each of which is a substantial edit by a different author of a single original report discussing XML technology. Each of these nine documents shares a co-derivation relationship with each of the other eight documents, though in some cases they only have a relatively small quantity of text in common. The *linuxdocs* collection consists of 78,577 documents (720 MB) drawn from the documentation included with a number of distributions of RedHat Linux. While the *webdata + xml* collection serves as an artificial but easily analysed testbed for co-derivative identification algorithms, the *linuxdocs* collection, rich in duplicate and near-duplicate documents, is a larger and more challenging real-world collection.

The *all-newswire* collection is an aggregation of the newswire collections gathered for the TREC project [21]. This collection includes a large number of articles from newswires originating from sources such as the Wall Street Journal, the Los Angeles Times, Associated Press and the Financial Times.

[3] http://www.seg.rmit.edu.au/zettair/

This collection totals to about 5.3 GB in size. This relatively large collection will be used to investigate the scaling properties of the DECO package.

The *GOV1* and *GOV2* collections were also created for various tracks of the TREC project. Both are crawls of the .gov domain, the former consisting of 18.1 GB of data, while *GOV2* contains 426 GB of data, making it the largest publicly available collection of web documents.

The *LATimes* collection is a subset of the *all-newswire* collection, consisting of 476 MB of articles from the Los Angeles Times. This is used to investigate the index growth we may expect from a typical collection of documents.[4]

Metrics for evaluation: We define a collection's *reference graph* as the relationship graph that would be generated by a human judge for the collection.[5] The *coverage* of a given computer-generated relationship graph is the proportion of edges in the reference graph that are also contained in that graph, and the *density* of a relationship graph is the proportion of edges in that graph that also appear in the reference graph.

While coverage and density are in many ways analogous to the traditional recall and precision metrics used in query-based information retrieval [22], we choose the new terminology to emphasise that the task is quite different to querying: we are not trying to meet an explicitly defined information need, but are rather attempting to accurately identify existing information relationships within the collection.

Fig. 4 shows a hypothetical example of a computer-generated relationship graph for the same collection as Fig. 1. The dashed line shows a spurious co-derivation judgement between documents 3 and 5. The dotted lines show the omission of co-derivation relationships between documents 2 and 7, and between documents 5 and 7. Given that the algorithm identified four of the six co-derivation relationships in the reference graph, it has a coverage of 67%. As four of the five co-derivation relationships identified existed in the reference graph, it has a density of 80%.

[4]Though in one sense there is no such thing as a 'typical' collection of documents, *LATimes* is neither cleaned of co-derivatives nor deliberately contrived to contain them, in contrast to the *webdata + xml* and *linuxdocs* collections.

[5]Although the concept of an 'ideal' underlying relationship graph is a useful artifice, the usual caveats of subjectivity and relativity must be borne in mind.

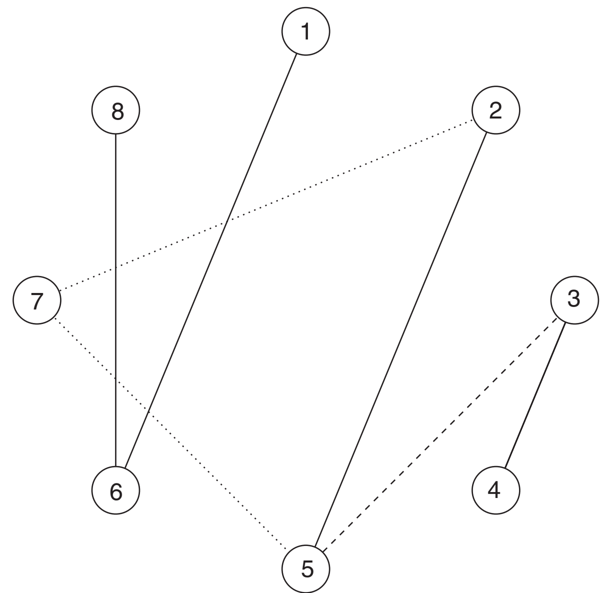


Fig. 4. An example of a computer-generated relationship graph. The dashed line indicates a spurious inclusion; the two dotted lines show co-derivation relationships that were not detected by the algorithm.

While the notion of a human-generated reference graph is a useful one, it is infeasible to construct one for any but the smallest of document collections. Doing so would require for side-by-side judgements of every document pair in the collection to be made. For a relatively modest collection of 10,000 documents this amounts to nearly 50 million judgements. Given the impracticality of generating a full reference graph, it is not possible to directly determine the coverage and density of the relationship graph generated by an algorithm. We must therefore resort to estimates.

To estimate the density of a relationship graph, we take a random selection of edges from the graph and judge whether the documents they connect are in fact co-derived. To estimate the coverage of a relationship graph, we select a number of representative documents and manually determine a list of documents with which they are co-derived. The coverage estimate is then the proportion of the manually determined pairings that are identified in the relationship graph. A third metric, average precision, is simply the average proportion of co-derivative edges to total edges for the documents selected to estimate coverage. While it is an inferior measure to density, it plays a role in experimentation because it is far less time-consuming to calculate.

8. Testing and discussion

8.1. Index growth rate

In order to investigate the growth trend of the shared-chunk index as the source collection grows, we extracted subcollections of various sizes from the *LATimes* collection and the *linuxdocs* collection, and observed the number of duplicate chunks extracted as the size of the collection was increased.

This growth trend is important for the scalability of SPEX and by extension the DECO package: if the growth trend were quadratic, for example, this would set a practical upper bound on the size of the collection that could be submitted to the algorithm, whereas if the trend were linear or $n \log(n)$ then far larger collections would become practical.

We found that, for the tested collections at least, the growth rate follows a reasonably precise linear trend, as illustrated in Fig. 5. While further testing is warranted, a linear growth trend suggests that the algorithm has potential to scale extremely well.

8.2. Scalability to larger collections

In order to test the ability of SPEX and DECO to scale to larger collections, we ran DECO on the *all-newswire* collection, which is over 5 GB in size. This took a little over 4 h on a lightly-loaded machine with dual Intel Pentium III processors and 768 MB of RAM. We consider an indexing speed in excess of 1 GB/h on an ageing machine to be acceptable for this application.

The index generated by DECO was approximately 2.3 GB in size. This index included the full text of each shared chunk (not just the hash-key), but was nonetheless less than 50% of the size of the source collection. This is not an unreasonable proportion, and can be further reduced by omitting the full chunk text from the index; doing so would result in a significant space saving at the expense of not being able to explicitly list the chunks shared between two documents. Furthermore, there would be a slight risk of hashing collisions if full text is not stored.

More recent experiments have had DECO successfully indexing the 18.1 GB GOV1 collection, with a total index size of 7.6 GB, much in line with the ratio above. Early experiments on the 426 GB GOV2 collection have also indicated that DECO is able to scale to collections of this size on a standard production server.

The precise relationship between collection size and memory requirements is ultimately dependent upon the level of duplication inherent in the particular collection being indexed. However, our results show that the algorithm is certainly able to scale to quite large collections without running up against resource limitations.

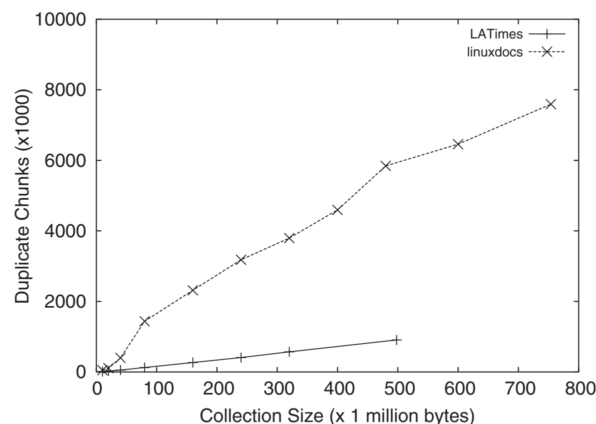


Fig. 5. The total number of duplicate chunks found as the size of the *LATimes* collection is increased.

8.3. Effect of identifying and excluding exact duplicates

As discussed in Section 6.1, DECO optionally includes a preprocessing stage in which exact duplicate documents are omitted from the indexing process. The justification for this is that it is relatively easy to identify such exact duplicates and that they consume a disproportionate amount of space in the shared chunk index.

Turning on this preprocessing step for the *all-newswire* collection reduced the size of the resultant shared-chunk index from 2.3 to 2.1 GB, a reduction in size of approximately 10%. While this is not a spectacular result—somewhat less so than anticipated—it is nonetheless a worthwhile saving.

When investigating the reason for the modesty of the saving, we found that a large number of documents were nearly identical, but differed in just a few words. In many newswires, for example, the same article was released multiple times for different editions. As the edition heading differed, the documents were not considered identical and thus no exclusions resulted. While this problem might be especially characteristic of newswire collections, there are many cases where one can imagine this such trivial differences thwarting this simple preprocessing step.

Problems such as this are what motivates systems such as I-Match [23] and parts of the Talent system [24]. I-Match is an algorithm that is designed to detect near-duplicates by selectively extracting words from each document using statistical techniques (for example, extracting only the least frequently occurring words) and hashing the aggregation of these words. Documents with the same hash-value are then flagged as duplicates or near-duplicates. This is nearly as fast and efficient a process as our current naïve preprocessing method, and we are looking at extending our preprocessing algorithm in a similar manner in order to improve its effectiveness.

8.4. Effectiveness in identifying co-derivatives

Because the *webdata + xml* collection contains the nine seed documents for which we have exact knowledge of co-derivation relationships, it makes a convenient collection for proving the effectiveness of the DECO package and determining good parameter settings. Using DECO to create a shared-chunk index with a chunk size of eight took under 1 min on an Intel Pentium 4 PC with 512 MB of RAM. For this collection, we tested DECO using the four scoring functions described in Section 6. We tested a range of five inclusion thresholds for each scoring function, making for a total of 20 combinations. These thresholds are named—in order of increasing value—T_1–T_5; the values vary between the scoring functions and were chosen based on preliminary experiments. Exact values are listed in Table 1. We use the notation $S_x/T_y/K_z$ to denote a run with scoring function x, threshold y and modulo factor z. For example, run $S_3/T_2/K_{16}$ indicates the run using scoring function S_3, threshold T_2 and a modulo parameter k equal to 16.

Each of the 20 generated relationship graphs were then tested for the presence of the 36 edges connecting the XML documents to each other.

Table 1
The values of thresholds T_1–T_5 for the various scoring functions

	S_1	S_2	S_3	S_4
T_1	20	0.03	0.02	0.02
T_2	100	0.05	0.05	0.05
T_3	200	0.10	0.10	0.20
T_4	500	0.15	0.15	0.50
T_5	1000	0.30	0.30	1.0

Table 2
Coverage estimates, as percentages, for the *webdata + xml* collection calculated on the percentage of XML document pairings identified

	T_1	T_2	T_3	T_4	T_5
S_1	100.0	97.2	36.1	8.3	0.0
S_2	100.0	100.0	83.3	58.3	25.0
S_3	100.0	91.7	72.2	52.8	16.7
S_4	100.0	97.2	91.7	58.3	22.2

The average precision was 100% in all cases.

As can be seen in Table 2, the estimated coverage values strongly favour the lower inclusion thresholds. Indeed, for all scoring functions using the inclusion threshold T_1, 100% of the pairings between the XML documents were included in the relationship graph. In all cases the average precision was also 100%. These values—100% coverage and 100% average precision—suggest a perfect result, but are certainly overestimates. The nature of the test collection—nine co-derived documents seeded into an entirely unrelated background collection—made it extremely unlikely that spurious edges would be identified. This not only introduced an artificially high average precision estimate but also strongly biased the experiments in favour of the lower inclusion thresholds, because they allowed all the correct edges to be included with very little risk that incorrect edges would likewise be admitted.

In order to test DECO in a less artificial environment, we repeated our experiments on the *linuxdocs* collection. We again used DECO to create a shared-chunk index with a chunk size of eight, taking approximately 30 min on an Intel Pentium 4 PC with 512 MB of RAM. For generation of relationship graphs we used the same range of scoring functions and inclusion thresholds as in the previous section.

To estimate the coverage of the relationship graphs, we selected 10 documents from the collection representing a variety of different sizes and types, and manually collated a list of co-derivatives for each of these documents. This was done by searching for other documentation within the collection that referred to the same program or concept; thus, the lists may not be entirely comprehensive. Estimated coverage and average precision results for this set of experiments are given in Table 3.

Overall, the results are very good. In general, scoring functions S_2, S_3, and S_4 were more effective

Table 3
Coverage and average precision estimates, as a pair X/Y of percentages, for DECO applied to the *linuxdocs* collection, using a full shared-chunk index

	T_1	T_2	T_3	T_4	T_5
S_1	100/70	89/71	56/93	36/95	34/100
S_2	100/57	100/75	100/92	89/94	57/100
S_3	98/75	96/84	94/100	84/100	47/100
S_4	99/83	96/91	94/100	78/100	30/100

than the simple chunk-counting S_1 scoring function. This is as expected and demonstrates the importance of normalising results to the size of the documents in question.

A number of combinations yielded excellent results. In particular, it is worth noting the extremely high reliability of the system. A number of combinations were able to identify all or nearly all pairs of co-derivative documents in our test set with very few false positives, or none at all. This is a highly desirable characteristic for such an algorithm.

We had insufficient human resources to complete an estimate of density for all of the relationship graphs generated. Instead, we selected a range of configurations that seemed to work well and estimated the density for these configurations. This was done by picking 30 random edges from the relationship graph and manually assessing whether the two documents in question were co-derived. The results corresponded very closely with the average precision for the same runs: $S_2/T_3/K_1$, $S_3/T_2/K_{256}$, and $S_4/T_3/K_{16}$ all scored a density of 93.3% (28 out of 30) while $S_4/T_3/K_1$ and $S_2/T_1/K_{16}$ both returned an estimated density of 100%. This suggests that the average precision is a good predictor of the true density value.

8.5. Effects of introducing lossy selection

In order to test the relationship between the lossiness of a selection heuristic and the degradation in the reliability of co-derivative identification, we performed the same experiments on the *linuxdocs* collection as above, this time with the *modulo* heuristic added to the selection pipeline. We experimented with the k operator set to 16 and 256. The only change to the experimental parameters (apart from the inclusion of the *modulo* heuristic) was that the inclusion thresholds for these experiments were adjusted downward commensurately with the *modulo* operator so as not to prejudice the results. In other words, the thresholds were divided by 16 and 256, respectively, as this is the expected decrease in the number of shared hash-keys between a given pair of documents.

Table 4
Coverage and average precision estimates, as a pair X/Y of percentages, for DECO applied to the *linuxdocs* collection for indexes that store chunks only if their hash-key equals zero *modulo* 16 and 256

	T_1	T_2	T_3	T_4	T_5
Fingerprinting *modulo* 16					
S_1	90/72	88/76	56/94	36/96	34/100
S_2	90/75	90/75	80/94	78/100	57/100
S_3	88/82	86/91	74/100	74/100	47/100
S_4	88/85	86/93	86/93	69/100	60/100
Fingerprinting *modulo* 256					
S_1	54/95	54/95	54/95	54/95	34/97
S_2	54/97	54/97	54/97	54/97	44/97
S_3	54/97	54/100	54/100	51/100	42/100
S_4	54/97	54/100	54/100	44/100	31/100

The results for these experiments are presented in Table 4. When the *modulo* heuristic is used with $k = 16$, the results are noticeably inferior to those using the full shared-chunk index generated by SPEX. Nonetheless, they are still reasonably good. In an application where finding every single co-derived document pair is not critical, the loss of a few percentage points in reliability in return for a 16-fold reduction in the number of chunks indexed might be a very attractive trade-off.

For the *modulo* 256 index, no configuration was able to find more than 54% of the relevant edges. This is almost certainly because the other 46% of document pairs simply do not have any chunks in common that evaluate to 0 *modulo* 256 when hashed using our particular hash function. This risk exists to a certain degree with any lossy selection scheme, but especially so when the lossiness is too aggressive. Two documents with a significant number of shared chunks could be overlooked even at the lowest thresholds because not one of their shared chunks is included in the index. This is not acceptable in many situations.

9. Future work & conclusions

There are many reasons why one may wish to discover co-derivation relationships amongst the documents in a collection. Previous feasible solutions to this task have been based on fingerprinting

algorithms that used heuristic chunk selection techniques. We have argued that, with these techniques, one can have either reliability or acceptable resource usage, but not both at once.

We have introduced the SPEX algorithm for efficiently identifying non-unique chunks in a collection. Unique chunks represent a large proportion of all chunks in the collection—over 98% in one of the collections tested—but play no part in discovery of co-derivatives. Identifying and discarding these chunks means that document fingerprints only contain data that are relevant to the co-derivative discovery process. In the case of the *LATimes* collection, this allows us to create an index that is functionally equivalent to full fingerprinting but is one-fiftieth of the size of a full chunk index. Such savings allow us to implement a system that is effective and reliable yet requires only modest resources.

Tests of our DECO system, which used the SPEX algorithm, on two test collections demonstrated that the package is capable of reliably discovering co-derivation relationships within a collection, and that introducing heuristic chunk-selection strategies degraded reliability.

There is significant scope for further work and experimentation with DECO. Although we have demonstrated that the system is able to scale to quite substantial collection sizes, it is important to continue to investigate additional optimisations to the algorithms and systems; this is both to increase the speed of the system and to allow it to scale to the extremely large multi-terabyte collections that are being managed in many domains.

A number of potential applications for systems such as DECO were mentioned in the paper; we are currently investigating the effect of text reuse on search effectiveness. Our results thus far indicate that document redundancy is a significant problem for search engines and that using DECO we are able to substantially improve search effectiveness.

One current area of difficulty is that SPEX is a one-shot algorithm: it must have access to the entire collection in order to build a shared-chunk index for it. It is not possible to later add additional documents to the collection without rebuilding the entire index. The difficulty of extending the index is the one major defect of SPEX compared to many other fingerprinting selection heuristics. We believe that it is possible to write an adjunct to SPEX that, while it would carry some overhead, would allow for incremental building of the shared-chunk index as new documents are added to the collection.

However, the sensitivity, reliability and efficiency of SPEX make it already a valuable tool for analysis of document collections.

Acknowledgements

This research was supported by the Australian Research Council.

References

[1] A.Z. Broder, On the resemblance and containment of documents, in: Compression and Complexity of Sequences (SEQUENCES'97)', 1997, pp. 21–29.
[2] M. Sanderson, Duplicate detection in the Reuters collection, Technical Report TR-1997-5, University of Glasgow, 1997.
[3] N. Shivakumar, H. García-Molina, SCAM: a copy detection mechanism for digital documents, in: Proceedings of the Second Annual Conference on the Theory and Practice of Digital Libraries, 1995.
[4] T.C. Hoad, J. Zobel, Methods for identifying versioned and plagiarised documents, Journal of the American Society for Information Science and Technology 54 (3) (2003) 203–215.
[5] U. Manber, Finding similar files in a large file system, in: Proceedings of the USENIX Winter 1994 Technical Conference, 1994, pp. 1–10.
[6] S. Brin, J. Davis, H. García-Molina, Copy detection mechanisms for digital documents, in: Proceedings of the ACM SIGMOD Annual Conference, 1995, pp. 398–409.
[7] N. Heintze, Scalable document fingerprinting, in: 1996 USENIX Workshop on Electronic Commerce, 1996.
[8] A.Z. Broder, S.C. Glassman, M.S. Manasse, G. Zweig, Syntactic clustering of the Web, Computer Networks and ISDN Systems 29 (8–13) (1997) 1157–1166.
[9] I.H. Witten, A. Moffat, T.C. Bell, Managing Gigabytes: Compressing and Indexing Documents and Images, Morgan Kauffman, Los Altos, CA, 1999.
[10] S. Schleimer, D.S. Wilkerson, A. Aiken, Winnowing: local algorithms for document fingerprinting, in: Proceedings of ACM SIGMOD conference, ACM Press, 2003, pp. 76–85.
[11] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, J.D. Ullman, Computing iceberg queries efficiently, in: Proceedings of the 24th International Conference on Very Large Data Bases, Morgan Kaufmann, Los Altos, CA, 1998, pp. 299–310.
[12] N. Shivakumar, H. García-Molina, Finding near-replicas of documents on the web, in: WEBDB, International Workshop on the World Wide Web and Databases, WebDB, Springer, Berlin, 1999.
[13] C.G. Nevill-Manning, I.H. Witten, Compression and explanation using hierarchical grammars, The Computer Journal 40 (2/3) (1997) 103–116.
[14] N.J. Larsson, A. Moffat, Offline dictionary-based compression, Proc. IEEE 88 (11) (2000) 1722–1732.
[15] C.G. Nevill-Manning, I.H. Witten, G.W. Paynter, Browsing in digital libraries: a phrase-based approach, in: Proceedings

of the Second ACM International Conference on Digital Libraries, ACM Press, 1997, pp. 230–236.

[16] A. Moffat, R. Wan, Re-Store: a system for compressing, browsing, and searching large documents, in: Proceedings of the International Symposium on String Processing and Information Retrieval, IEEE Computer Society, 2001, pp. 162–174.

[17] D. Gusfield, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Cambridge University Press, Cambridge, 1997.

[18] R. Agrawal, R. Srikant, Fast algorithms for mining association rules in large databases, in: Proceedings of the 20th Conference on Very Large Data Bases, Morgan Kaufmann, Los Altos, CA, 1994, pp. 487–499.

[19] I.H. Witten, T.C. Bell, Source models for natural language text, Int. J. Man Machine Studies 32 (1990) 545–579.

[20] R. Rivest, The MD5 Message-Digest Algorithm, RFC 1321, 1992.

[21] D. Harman, Overview of the second text retrieval conference (TREC-2), Information Processing and Management 31 (3) (1995) 271–289.

[22] R. Baeza-Yates, B. Ribeiro-Neto, Modern Information Retrieval, Addison-Wesley, Longman, Reading, MA, New York, 1999.

[23] A. Chowdhury, O. Frieder, D. Grossman, M.C. McCabe, Collection statistics for fast duplicate document detection, ACM Transactions on Information Systems (TOIS) 20 (2) (2002) 171–191.

[24] J.W. Cooper, A.R. Coden, E.W. Brown, Detecting similar documents using salient terms, in: Proceedings of the Eleventh International Conference on Information and Knowledge Management, ACM Press, 2002, pp. 245–251.