A Scalable System
for Identifying Co-derivative Documents

Yaniv Bernstein and Justin Zobel

School of Computer Science and Information Technology
RMIT University, Melbourne, Australia
{ybernste,jz}@cs.rmit.edu.au

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. The current 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 currently 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 system that makes use of SPEX. Our experiments with several document collections demonstrate the effectiveness of the approach.

1   Introduction

Many document collections contain sets of documents that are *co-derived*. Examples of co-derived documents include plagiarised documents, document revisions, and digests or abstracts. Knowledge of co-derivative document relationships in a collection can be used for returning more informative results from search engines, detecting plagiarism, and managing document versioning in an enterprise.

Depending on 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 designed for solving the search problem, this quickly becomes far too time-consuming for practical use.

All current 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 differentiate themselves largely on 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 phrase indexes generated by SPEX as the basis for accurate and efficient identification of co-derivative documents in a collection. We believe 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. We also have data to suggest that DECO should scale well to very large collections.

2   Co-derivatives and the Discovery Problem

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 document. Broder (1997) 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 document 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 drives fingerprinting-based approaches, described later.

We can conceptualise the co-derivation relationships within a collection as a graph, with each node representing a single document and the presence or absence of an edge between two nodes representing the presence or absence of a co-derivation relationship between the documents represented by those nodes. We call this the *relationship graph* of the collection. The task of the discovery problem is to discover the structure of this graph. Note that, as the 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.

3   Strategies for Co-derivative Discovery

There are several approaches to solving the search problem, in particular fingerprinting systems and ranking-based systems. Ranking-based systems such as relative frequency matching (Shivakumar & García-Molina 1995) and the identity measure (Hoad & Zobel 2003) make use of document statistics such as the relative frequency of words between documents to give a score for how likely a pair of documents is to be co-derived. In comparisons between such methods and fingerprinting, the ranking-based methods tended to perform better, though it is worth noting that the comparisons were carried out by the proponents of these systems. However, the only computationally feasible algorithms for the discovery problem to date have used the process of document fingerprinting.

3.1 Fingerprinting

The key observation underlying document fingerprinting (Manber 1994, Brin et al. 1995, Heintze 1996, Broder et al. 1997, Hoad & Zobel 2003) mirrors that behind the definitions of Broder (1997): if documents are broken down 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 they have some common chunks in their fingerprints.

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/or compact storage; the hash-keys, and possibly also the chunks themselves, are then stored, often in an inverted index structure (Witten et al. 1999). 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 & Zobel (2003). 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 (Heintze 1996). Taking a different approach is the *anchor* strategy (Manber 1994), 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 et al. (2003) 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.

In their comparative experiments, Hoad & Zobel (2003) 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 ranking-based identity measure system. Similarly, Shivakumar & García-Molina (1995) found that the COPS fingerprinting system (Brin et al. 1995) was far more likely than their SCAM ranking-based system to fail to identify co-derivative documents.

Several techniques use fingerprinting for the discovery problem:

Manber (1994) 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. (1997) 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. The number of tokens for each pair of documents is counted and used as the basis for a set of discovery results. While this approach can yield far more informative results, taking the Cartesian product of each postings list means that the number of tokens generated is quadratic in the length of the postings list; this can easily cause resource blowouts and introduces serious scalability problems for the algorithm.

Shivakumar & García-Molina (1999) addressed the scalability problems of the previous algorithm by introducing a hash-based *probabilistic counting* technique. For each document pair, instead of storing a token, a counter 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.

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 6) 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 (1994), 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. (1997). Such lossiness is unacceptable in many applications.

Schleimer et al. (2003) 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 over-

looked. 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.

3.2   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.

Hierarchical dictionary-based compression techniques like SEQUITUR (Nevill-Manning & Witten 1997) and RE-PAIR (Larsson & Moffat 2000) 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 (Nevill-Manning et al. 1997) and RE-STORE (Moffat & Wan 2001). 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 (Nevill-Manning et al. 1997). 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 collections of significant size.

Suffix trees are another potential technique for duplicate-chunk identification, and are used in this way in computational biology (Gusfield 1997). 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.

4   The SPEX Algorithm

Our novel hash-based SPEX algorithm for duplicate-chunk extraction has much more modest and flexible memory requirements than the above and is thus the first selection algorithm that is able to provide *lossless* chunk selection within large collections. The fundamental observation behind the operation of SPEX is that if any subchunk of a given chunk can be shown to be unique, then the chunk in its entirety must be 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 very likely to be duplicates.

The basic mechanics of the algorithm are shown in Algorithm 1. At the core of SPEX is a pair of hashcounters – hashtable accumulator arrays – designed to

---

Algorithm 1 The SPEX algorithm.

1: // **C**: Collection of chunks
2: // l: Target chunk length
3: // c_n: chunk of length n
4: // $c_n\{p \dots q\}$: The chunk 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** $c_1 \in \mathbf{C}$ **do**
9:     $h_1[\#(c_1)] \leftarrow h_1[\#(c_1)] + 1$
10: **end for**
11: **for** $n \in [2, l]$ **do**
12:     **for all** $c_n \in \mathbf{C}$ **do**
13:         **if** $h_{n-1}[\#(c_n\{1 \dots n-1\})] > 1$ **and** $h_{n-1}[\#(c_n\{2 \dots n\})] > 1$ **then**
14:             $h_n[\#(c_n)] \leftarrow h_n[\#(c_n)] + 1$
15:         **end if**
16:     **end for**
17: **end for**

---

count string occurrences. Each time a string is inserted into a hashcounter, it is hashed and a counter at that location is incremented. Collisions are not resolved. For the purposes of the SPEX algorithm, we care about only three counter values: 0, 1 and '2 or more'. As such, each field in the hashcounter need be only two bits wide. If the same string is inserted into a hashcounter more than once, the hashcounter will indicate this. The hashcounter can also return false positives, indicating a string occurs multiple times when it in fact does not. A small number of such false positives can be tolerated by SPEX; the number can be kept small because the two-bit wide fields allow for extremely large hashcounters to reside in a relatively modest amount of memory

When a document collection is presented to SPEX, the first step is to sequentially scan the collection and insert each word encountered into a hashcounter. This hashcounter thus indicates (with the possibility of false positives) whether a word occurs multiple times in the collection. Following this, we pass a sliding window of size two words over the collection. Each two-word chunk is broken down into two single word subchunks and compared against the hashcounter. If the hashcounter indicates that both subchunks occur multiple times then the chunk is inserted into the second hashcounter. Otherwise, the chunk is rejected. After this process is complete, the second hashcounter indicates whether a particular chunk of size two is a possible duplicate chunk. For chunks of size three, we pass a sliding window of length three over the collection and decompose the candidate chunks into two subchunks of length two. We similarly accept a chunk only if it is indicated by the hashcounter that both subchunks occur multiple times within the collection. Figure 1 illustrates this process.

The algorithm can be extended to any desired chunk size l by iteration, at each phase incrementing the chunk size by one. We only ever require two hashcounters because the hashcounter for chunks of size $n - 2$ is no longer re-
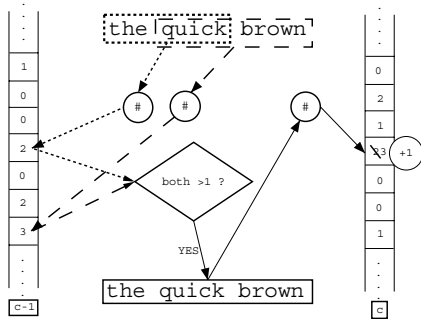
Fig. 1. The process for inserting a new chunk into the hashcounter in SPEX. The chunk "the quick brown" is divided into two sub-chunks "the quick" and "quick brown" They are each hashed into the old hash table. If the count for both sub-chunks is greater than one, the full chunk is hashed and the counter at that location in the new hashcounter is incremented.

quired when searching for chunks of size n and may be reused. We are not overly concerned about false positives, because subsequent iterations tend to have a dampening rather than an amplifying effect on their presence. SPEX is thus able to provide an accurate representation of duplicate chunks of length u in a time proportional to $O(uv)$, where v is the length of the document collection.

5   The DECO Package

Our DECO system for co-derivative detection presents a number of innovations. The most significant of these is the use of SPEX for creating shared-chunk indexes. Another addition is the inclusion of more sophisticated scoring functions for determining whether documents are co-derived. DECO operates in two phases: index building and relationship graph generation. In the index building phase, SPEX is used as described earlier. At the final iteration of the algorithm, the chunks that are identified as occurring more than once are stored in an inverted index structure (Witten et al. 1999). This index contains an entry for each duplicate chunk and a list of each document where it occurs. We call this index the *shared-chunk index*.

In the relationship graph generation phase, DECO uses the shared-chunk index and an approximate counting technique similar to that proposed by Shivakumar & García-Molina (1999) in order to identify co-derived document pairs. Several parameters must be specified to guide this process: the most important of these are the *scoring function* and the *inclusion threshold*. Given documents u and v, the scoring function may at present be one of the following:

$$S_1(u,v) = \sum_{c \in u \wedge c \in v} 1 \qquad S_2(u,v) = \sum_{c \in u \wedge c \in v} 1/\min \bar{u}, \bar{v}$$
$$S_3(u,v) = \sum_{c \in u \wedge c \in v} 1/\operatorname{mean} \bar{u}, \bar{v} \qquad S_4(u,v) = \sum_{c \in u \wedge c \in v} \frac{1/f_c}{\operatorname{mean} \bar{u}, \bar{v}}$$

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 don't dominate smaller ones in the results. They are very similar to the resemblance measure of Broder (1997) 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 desirable and a topic of 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.

6   Experimental Methodology

We use three document collections in our experiments. The *webdata+xml* and *linuxdocs* collections were accumulated by Hoad & Zobel (2003). The *webdata+xml* collection consists of 3,307 web documents totalling approximately 35 megabytes, into which have been seeded nine 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 *LATimes* collection is a 476 megabyte collection of newswire articles from the Los Angeles Times, one of the newswire collections created for the TREC conference (Harman 1995). This collection is used as an example of a typical document collection and is used to investigate the index growth we may expect from such a typical collection.

We define a collection's *reference graph* as the relationship graph that would be generated by a human judge for the collection[1]. 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 these two concepts are in many ways analogous to the traditional recall and precision metrics used in query-based information retrieval (Baeza-Yates & Ribeiro-Neto 1999), we choose the new terminology to emphasise that the task is quite different to querying: we are not trying to meet an explicit

---

[1] 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.

information need, but are rather attempting to accurately identify existing information relationships within the collection.

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 average density, it plays a role in experimentation because it is far less time-consuming to calculate.

7   Testing and Discussion

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 grew.

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 which 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 this collection at least, the growth rate follows a reasonably precise linear trend. For the *LATimes* collection, 40 MB of data yielded 54,243 duplicate chunks; 80 MB yielded 126,542; 160 MB 268,128; and 320 MB 570,580 duplicate chunks. While further testing is warranted, a linear growth trend suggests that the algorithm has potential to scale extremely well.

Webdata+XML Experiments. 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 one minute 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 5. For each scoring function, we tested a range of five inclusion thresholds, named – in order of increasing value – T_1 to T_5; the values vary between the scoring functions and were chosen based on preliminary experiments. Each of the 20 generated relationship graphs were then tested for the presence of the 36 edges connecting the XML documents to each other.

As can be seen in Table 1, 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

Table 1. Coverage estimates, as percentages, for the *webdata+xml* collection calculated on the percentage of XML document pairings identified. The average precision was 100% in all cases.

	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

Table 2. 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 and 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
Full chunk indexing					
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
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

included in the relationship graph. In all cases the average precision was also 100%. These values – 100% coverage and 100% density – 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 density 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.

Experiments on the Linux Documentation Collection. For the *linuxdocs* collection, we used DECO to create a shared-chunk index with a chunk size of eight, taking approximately 30 minutes 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. We wished also to investigate the level of deterioration witnessed in a fingerprinting strategy

as the selectivity of the fingerprint increased; to this end, we experimented with relationship graphs generated from indexes generated using the *modulo* heuristic with $k = 16$ and $k = 256$. The inclusion threshold for these experiments were adjusted downward commensurately.

To estimate the coverage of the relationship graphs, we selected ten 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 2. Several trends are observable in the results. The first of these is that in general, scoring functions S_2, S_3, and S_4 were more effective than the simple chunk-counting S_1 scoring function Another trend is that performance is noticeably superior with the full shared-chunk index than with the selective shared-chunk indexes. Note in particular that, 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 do not have any chunks in common that evaluate to 0 *modulo* 256 when hashed. This illustrates the dangers of using lossy selection schemes when a high degree of reliability is desired.

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 were pleasingly high: $S_2/T_3/1$, $S_3/T_2/256$, and $S_4/T_3/16$ all scored a density of 93.3% (28 out of 30) while $S_4/T_3/1$ and $S_2/T_1/16$ both returned an estimated density of 100%. Other combinations were not tested.

8   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 shared 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 is 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. One area of particular importance is the scalability of the algorithm. We have demonstrated that the system performs capably when presented with a highly redundant 700 MB collection and are confident that it can handle much larger collections, but this needs to be experimentally demonstrated. Another important further development is the design of an adjunct to the SPEX algorithm that would make it possible to add new documents to a collection without rebuilding the entire shared-chunk index. The difficulty of extending the index is the one major defect of SPEX compared to many other fingerprinting selection heuristics. However, the sensitivity, reliability and efficiency of SPEX make it already a valuable tool for analysis of document collections.

Acknowledgements

References

Baeza-Yates, R. & Ribeiro-Neto, B. (1999), *Modern Information Retrieval*, Addison-Wesley Longman.

Brin, S., Davis, J. & García-Molina, H. (1995), Copy detection mechanisms for digital documents, *in* 'Proceedings of the ACM SIGMOD Annual Conference', pp. 398–409.

Broder, A. Z. (1997), On the Resemblance and Containment of Documents, *in* 'Compression and Complexity of Sequences (SEQUENCES'97)', pp. 21–29.

Broder, A. Z., Glassman, S. C., Manasse, M. S. & Zweig, G. (1997), 'Syntactic clustering of the Web', *Computer Networks and ISDN Systems* **29**(8-13), 1157–1166.

Gusfield, D. (1997), *Algorithms on strings, trees, and sequences: computer science and computational biology*, Cambridge University Press.

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

Heintze, N. (1996), Scalable Document Fingerprinting, *in* '1996 USENIX Workshop on Electronic Commerce'.

Hoad, T. C. & Zobel, J. (2003), 'Methods for Identifying Versioned and Plagiarised Documents', *Journal of the American Society for Information Science and Technology* **54**(3), 203–215.

Larsson, N. J. & Moffat, A. (2000), 'Offline Dictionary-Based Compression', **88**(11), 1722–1732.

Manber, U. (1994), Finding Similar Files in a Large File System, *in* 'Proceedings of the USENIX Winter 1994 Technical Conference', San Fransisco, CA, USA, pp. 1–10.

Moffat, A. & Wan, R. (2001), 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, pp. 162–174.

Nevill-Manning, C. G. & Witten, I. H. (1997), 'Compression and Explanation Using Hierarchical Grammars', *The Computer Journal* **40**(2/3), 103–116.

Nevill-Manning, C. G., Witten, I. H. & Paynter, G. W. (1997), Browsing in digital libraries: a phrase-based approach, *in* 'Proceedings of the second ACM international conference on Digital libraries', ACM Press, pp. 230–236.

Schleimer, S., Wilkerson, D. S. & Aiken, A. (2003), Winnowing: local algorithms for document fingerprinting, *in* 'Proceedings of the 2003 ACM SIGMOD international conference on on Management of data', ACM Press, pp. 76–85.

Shivakumar, N. & García-Molina, H. (1995), SCAM: A Copy Detection Mechanism for Digital Documents, *in* 'Proceedings of the Second Annual Conference on the Theory and Practice of Digital Libraries'.

Shivakumar, N. & García-Molina, H. (1999), Finding Near-Replicas of Documents on the Web, *in* 'WEBDB: International Workshop on the World Wide Web and Databases, WebDB', Springer-Verlag.

Witten, I. H., Moffat, A. & Bell, T. C. (1999), *Managing Gigabytes: Compressing and Indexing Documents and Images*, Morgan Kauffman.