Indexed Signature Slice Lists for Fast Nearest-Neighbour Search

ABSTRACT

Signature Files in the Information Retrieval context are a technique used to represent documents, images and other abstract objects with binary signatures of fixed length. The signatures have the property that they preserve in signature space the mutual topological relationships that exist in the original representation of objects. The original representation is typically some form of very high-dimensional, often highly sparse, feature vector derived with some probabilistic, language model, or other suitable feature extraction/definition approach. Binary signatures offer a compression of the original representation onto a lower dimensional and dense representation. It facilitates efficient similarity computations using the Hamming distance[4] measure. This efficiency motivates most applications of signatures. Locality Sensitive Hashing is a method of dimensionality reduction that uses binary signatures often used in near-duplicate detection in image or text document collections. There are numerous publications about successful applications of Signature files. While signatures offer an acceptable measure of similarity, performing a search with a given search argument (a signature) requires a Hamming distance calculation against every signature in a collection. This quickly becomes excessive when dealing with web-sized collections, and there lies the problem of scalability. This paper addresses the scalability problem in signature search. We describe a new method of indexing and searching large binary signature collections to efficiently find similar signatures in very large collections. Experimental results demonstrate that our approach is effective in finding a set of nearest signatures to a given search argument (signature) with high efficiency and high fidelity.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Clustering, Retrieval Models, Search Process

General Terms

Algorithms, Experimentation, Performance, Theory

Keywords

Document Signatures, Near-Duplicate Detection

1. DOCUMENT SIGNATURES

Topological signatures, as described in Geva and De Vries[3] are a refinement on Locality Sensitive hashing. It is related to the approach originally explored by Faloutsos and Christodoulakis[2] of using file signatures to represent documents. ThesFaloutsos approach was shown to be inferior to inverted file approaches by Zobel et al.[6] Topology preserving signatures in the work reported in this paper are derived from a document-term matrix, randomly-projected onto a compact N dimensional binary $\{\pm 1\}^N$ subspace to produce a new, lower-order matrix (that can be efficiently stored) while preserving the topological relationships of documents in the original document-term matrix. The geometric interpretation of this projection is that the document collection is mapped from its original vector-space representation as a term-document weight matrix, onto the $\{\pm 1\}^N$ hypercube. Each document is mapped to a vertex of the hypercube, and the property of the random projection is that it largely preserves the relative pairwise distances between points in input space, on the hypercube. Furthermore, in [3] an effective approach to performing ad-hoc keyword based search with signatures is described. It overcomes the difficulties identified in the earlier models. A complete description of this approach is given in Geva and De Vries[3], where it is demonstrated that for early precision the use of signatures for ad-hoc retrieval leads to search performance that is not statistically different from state of the art models such as BM25, Language Models and Divergence from Randomness models.

For the sake of completeness, we briefly describe some details of that file signature approach. The original Faloustos et al approach is essentially a bloom filter; although not explicitly described as such in the original paper[2]. Terms in a document are hashed to create pseudorandom sparse bit strings that are combined together (through a bitwise OR operator) to create a document signature. Queries are then tested by comparing the signatures of terms in the query (derived in the same manner) with the document signatures. If all the set bits in the query signature are also set in a document signature, there is a possibility that the terms appear in the document. Since it is possible for these term bit

strings to combine to create a false positive for a term not present in the document, the document must be scanned to ensure that the terms in the query are actually present. This is characteristic of bloom filters, which are highly efficient but do result in false positives. The rate of false positives can be reduced by increasing the size of the bit strings at the expense of a greater storage requirement.

Signatures generated in this method can be used for information retrieval but have significant drawbacks, including the necessity of storing the terms present in the document to avoid false positives, and the cost of eliminating those become highly excessive. They also require extensive amounts of storage space to ensure signatures are long enough to reduce the occurrence of these false positives.

Locality Sensitive Hashing, as implemented and refined in the TopSig approach[3] fixes most of the problems identified with document signatures in terms of retrieval performance. However, one problem remained unsolved in that paper- the scalability of the approach is still limited while it relies on sequential exhaustive Hamming distance calculations. The approach required performing a Hamming distance calculation against every signature in the collection and it becomes prohibitive when collection sizes become very large. An approach is therefore needed to avoid performing this calculation against every signature, while still retrieving the nearest signatures.

2. NEAR-DUPLICATE DETECTION

While we have approached the use of document-derived signatures from the perspective of information retrieval, the use of signatures for nearest-neighbour search is not new as many approaches already exist that involve using signatures for near-duplicate detection.

Manku et al[5] describe their approach to the problem of trying to find close document-derived signatures for the purpose of finding duplicate documents. Their approach involves using 64-bit locality-sensitive signatures with the specific aim of finding all signatures that are at a Hamming distance of no more than 3 bits from the source signature. While the parameters of the search are completely different, this is a similar problem to the problem this paper is attempting to find a solution for. Ultimately their approach comes down to storing the signatures in a number of sorted lists and using binary searches to find runs that may contain signatures that match the properties described.

This approach is not feasible for large signatures (1024-bit is the typical signature size for a TopSig signature) due to the potential maximum distance and number of sorted lists required; however, one of the other possible approaches described in the paper and discarded for being inappropriate for 64-bit signatures is used here to some extent; namely, using probes into all possible signatures n bits away from the search signature. This does not work for 64-bit signatures, but when the signatures are longer the approach gains validity. It should be noted that 64 bit signatures are often used because it require less storage space and maps nicely to processor architecture. However, this choice is has a serious flaw - with 64 bits, and assuming a binomial distribution of signatures, there are effectively very few possible distances to work with in a similarity search. Most signatures will fall at a hamming distance of about 32 bits from the search argument. This leaves very little granularity (i.e. <32) in distances if millions of objects were to be distinguished by their distance from a search argumnet. Our approach works with signatures having thousands of bits as is necessary when working with document collections of realistic sizes (e.g the Wikipedia, or larger).

3. INDEXED SIGNATURE SLICE LISTS

Indexed Signature Slice Lists (ISSL) is a hybrid approach that combines the inverted file approach with the signature file approach. The purpose of an ISSL table is to provide an index into a signature file to allow searching for signatures close (in terms of Hamming distance) to a given search signature. In a normal text inverted file scheme the postings correspond to terms in the document collection and the search process combines those postings for a given set of query terms to produce a ranked list of results. File Signatures are just long binary vectors and do not have a vocabulary as such. If we are to invert a signature file we need to have a representation analogous to terms in a text document. Signatures are *sliced* into 16-bit segments. Each segment can take on any of the 64K possible binary values; these values become the vocabulary that we use to invert the respective slice (over the entire collection.) In a somewhat analogous manner to term inverted lists, ISSLs are posting lists of signature slice values. For instance, consider the first slice (bit positions 1 to 16) in all the signatures in the collection; we keep a posting list of all signature IDs that have a particular binary value in this slice position. Similarly we keep a posting list for each of the 64K possible values in every other slice position, corresponding to bit positions 17 to 32, positions 33 to 48, and so on for all slice positions. We now have an inverted-by-slice representation of the collection of signatures. Each slice is separately inverted with a vocabulary that is the set of all possible 16-bit binary values. Here the similarity with conventional inverted files ends since the processing of slice inverted lists during search is very different as we describe next.

The creation of an ISSL table proceeds as follows: We start from a set of binary signatures. We will focus the discussion on 1024-bit signatures, but other signature sizes are possible and indeed used. We treat each signature as a multiple set of 16 bit slices and view the signature as a set of 64 16-bit integers. For efficiency reasons related to processor and software architecture considerations, 16-bit slices work well; but signature slices of other sizes are possible. We find that this choice is adequate even as we work with web-scale collections. The ISSL index maintains posting lists for each slice position and each of the 65,536 possible 16-bit integer values. Hence there are $2^{22} = 4,194,304$ posting lists. When indexing a signature, each of the 64 bit slices in the signature appears in one of 64 posting lists. The specific lists are identified by the combination of the slice position and the binary value of the signature slice. The posting lists contain the signature IDs of the represented signatures. To identify all signatures that have a aparticluar binary value in a particular slice position we only need access the inverted list identified by the slice position and the binary value. That inverted list consists of the corresponding signature-ids.

When searching an ISSL file for a search signature, the search signature is divided into slices in the same manner. Each slice can be used to look up an ISSL, and all the signature IDs listed in the retrieved ISSLs can be consolidated. Trivially, when searching for a specific signature that was previously indexed (i.e it exists in the collection), we would

find the signature ID in each of the posting lists corresponding to its 64 slices. However, this is not satisfactory for our nearest neighbour search. In general, we are looking for nearest signatures which are highly unlikely to be identical. This means that we need to be able to compute the Hamming distance of signatures from the search signatures by using the ISSL postings - but these may not necessarily be exact matches. For instance, a signature may not appear in any of the 64 postings that correspond to the query signature, but may be exactly one bit away in each of those slices. That would amount to a Hamming distance of 64, and it may well be the case that this is the nearest signature. So a direct lookup of the search argument slice values does not produce the result we seek. In order to find the approximate nearest neighbours of the search argument we need to process the postings in a different manner to the way we do with conventional term-based search. The conventional inverted file approach only requires access to the postings of terms in the query. ISSL based search requires access to not only the postings of matching signature slices, but also to their neighbouring slices' postings. We next describe the search process, and in the experimental section discuss the efficiency of this approach.

3.1 Searching the ISSL table

Given a search argument (a signature) we are searching for neighbouring signatures using the ISSL table. An ISSL in the following is a reference to a posting list for a given slice position and a given 16-bit binary pattern. With signatures 1024 bits wide, we need to consider the neighbourhood in each of the 64 slices of the search argument. We consider not only the exact-matching ISSL, but also its neighbourhood. Given a 16-bit pattern in a signature slice, we have 1 ISSL that matches the pattern, 16 ISSLs that have a value that is 1 bit away, 120 ISSLs that have a value that is 2 bits away, and so on. The search is therefore expanded with the set of 16-bit patterns that covers a wider Hamming distance neighbourhood.

The determination of the Hamming distance of signatures to the search argument proceeds by consulting the ISSL table and keeping track of the distance estimate of signatures in the collection. Each time a signature ID is observed in an ISSL more information is revealed about its distance from the search argument. Clearly, if a signature ID is observed in all 64 slices we can compute its distance to the search argument with complete accuracy. However, if the signature is only observed in N of the 64 slices, the distance is known accurately only in those N from 64 slices, and in $64-N$ slices it is unknown. We can however compute a worst-case estimate of the distance by assuming that the distance is maximal in all unobserved slices. We can also compute a best-case distance by assuming that the unobserved slices for a given signature will all be seen at the next Hamming distance we process. For example, if we completed consulting the 3-bit neighbourhood, and observed a particular signature-id in N slices, the best-case scenario assumes we shall find the signature in the 4-bit neighbourhood in all the missing slices; we then add $4 * (64 - N)$ to the known distance from the N seen slices and this is the most optimistic estimate of the distance. For the pessimistic estimate of the distance we add $16 * (64 - N)$ to the known distance from the N seen slices (i.e. we pessimistically assume it will only be seen at the 16-bit neighbourhood in remaining slices.)

With the above, we are now in a position to progressively expand the search neighbourhood, and as we do so we progressively improve our estimate of the nearest-neighbourhood of a search argument. If we proceed with the neighbourhood expansion to completion we have exact distance calculation. Of course, this would be a very expensive process and we are seeking to drastically improve performance. If we stop the neighbourhood expansion early we have only partial information. We note, however, that sometimes even partial information will suffice to completely accurately identify the top-k nearest neighbours. We are not seeking a complete ordering of the collection of signatures with respect to the search argument. We are only searching for the nearest k neighbours. It is possible to stop the search as soon as we have observed k distinct signatures in all their 64 slices and have the exact top-k signatures. For instance, if we are looking for $k = 10$ and there exist 10 signatures that are exactly a Hamming distance of 1 from the search argument, then after consulting the 0-bit neighbourhood and the 1-bit neighbourhood (17 ISSL per slice, $64 * 17 = 1088$ ISSLs in total), we have already collected the top-10 neighbours with complete accuracy and we can stop. Seeing all top-k signatures in all slices with short distances is a very strict requirement though; it turn out to be too strict to be of any practical use. We conducted experiments with 1 million random 1024-bit signatures. We search for the top-16 neighbours of one of the signatures. We stopped consulting the ISSL when the distances of the top 16 signatures are completely determined from the ISSL. What we found is that we can only stop the search after consulting the 11- to 13-bit neighbourhood. This is far too expensive and so this approach is not viable.

We can stop earlier though; by keeping track of the best-case and worst-case distances, we can stop the search when the best-case distance of the k+1 nearest signature is larger than the worst-case distance of the k nearest signature. At this point the top-k can no longer change. We do not know the exact distances of the top-k yet, but we can easily compute them directly since we only need to perform k distance calculations. This approach may seem promising at first, but as it turns out, proceeding to the point of certainty in set of top-k (rather than their accurate distances) is still too expensive. With same set of 1 million random signatures, we find that a search for the nearest 16 signatures still typically terminates after consulting the 11-bit to 13-bit neighbourhood. That is not a viable approach either and requires processing the vast majority of posting lists.

Fortunately, it is not necessary to progress to the point of complete certainty. As the Hamming distance neighbourhood is expanded in the search, the probability that the top-k signatures will change diminishes rapidly. For instance, it is possible but highly unlikely that a signature that was never observed at a neighbourhood of 3-bits distance in any of the 64 slices will subsequently be found at the 4-bit distance in all of the 64 slices. Conversely, it is possible but highly unlikely that a signature that was observed at a distance of 0-bits in 60 slices, will be found at the 16-bit distance in the remaining 4 unseen slices.

This brings us to the key idea in this paper: we conjecture that in order to determine with high accuracy the neighbourhood of a search signature, it is not necessary to proceed with the calculation until complete information is available. We can stop the calculation early and have high confidence

Bits changed	Lists per slice	% of signatures
0	1	0.62%
1	17	7.13%
2	137	34.76%
3	697	79.62%
4	2517	98.93%
5	6885	100.00%
6	14893	100.00%
7	26333	100.00%
8	39203	100.00%
9	50643	100.00%
10	58651	100.00%
11	63019	100.00%
12	64839	100.00%
13	65399	100.00%
14	65519	100.00%
15	65535	100.00%
16	65536	100.00%

Table 1: Number of posting lists checked per slice based on the number of bits changed in the search slice. The third column shows the average portion of the collection covered by this neighbourhood in a typical document collection. Also see figure 1.

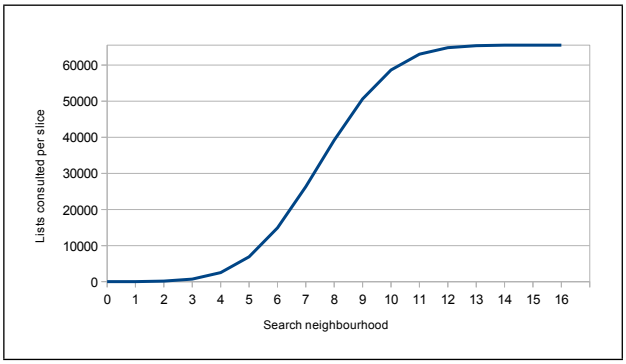

Figure 1: The relationship between the size of the search neighbourhood and the number of posting lists checked per slice. Derived from table 1.

that a relatively small set of the nearest signatures, say N, will contain the top-k signatures with high probability. Furthermore, we conjecture that when it does not contain all of the top-k nearest, the error will be relatively small and other almost as near signatures will be identified. So the search is stopped after consulting ISSL that correspond to a small hamming distance neighbourhood, a set of N nearest signatures is selected, having $N >> k$, and an exhaustive distance calculation over these N signatures is used to identify the top-k signatures. The value of k is more often than not relatively small. It does not usually depend on the collection size, but rather on the users' reluctance to review long result lists; this is typically determined by user time constraints and limited patience rather than collection size. Since k is small, we can choose $N >> k$ for accurate selection of the final top-k after early stopping. This approach allows us to more accurately estimate the top-k without excessive distance calculations over the entire collection by choosing $k << N << M$ where M is the collection size.

The estimation of the signature distance proceeds as follows: Initially all signatures are assumed to be at a pessimistic Hamming distance of 1024. As search progresses, all signatures that have not yet been encountered in any of the ISSL postings inspected maintain a worst case Hamming distance of 1024. On the other hand, a signature that is observed in a particular posting immediately receives a more optimistic estimate of its distance. For instance, if a signature is observed in an ISSL corresponding to a distance of 3 bits from the search argument slice then its worst case Hamming distance is reduced by 13. Most nearby signatures to the search argument are observed sooner rather than later in most or all slice positions and their distances becomes known with high acuracy. Of course, if the process is carried through until all ISSL postings are consulted then the distance of all signatures from the search argument is precisely known.

It is necessary to decide on the early stopping criterion. The number of posting lists consulted is a function of the search-breadth in bits. Table 1 provides the number of ISSL postings, per slice, that have to be consulted as function of search-breadth. The third column contains the percentage of distinct signature IDs encountered while processing the list. This column corresponds to signatures derived from the TREC Wall Street Journal collection of news articles[1]. The percentage figure corresponds to the fraction of document-ids seen as the ISSL posting are processed. For instance, when the ISSLs are scanned up to a 3-bit Hamming distance from the search argument, 697 ISSL posting lists are processed, and 79.62% of the document-ids in the collection are encountered (at least once) in the process.

When the maximum search-breadth is lower, the amount of processing time required is reduced as fewer posting need to be considered. Assuming an equal distribution of signatures per list[2] allowing a maximum search-breadth of 3 bits requires only 1.06% of the computational effort that would be required for a full search up to the maximum distance of 16 bits. The tradeoff is in accuracy; but as our experiments demonstrate, in practice the approach is highly accurate and the tradeoff is quite attractive.

After the distances for all signatures have been determined, the nearest signatures can be reranked using a full Hamming distance calculation to ensure that the ranking of these signatures is precise. At higher levels of search-breadth, more processing time is required but the likelihood of documents that should appear in the top-k being omitted is reduced. There are other speed-accuracy tradeoffs available; increasing the number of signatures that are reranked using Hamming distance may result in fewer top documents being omitted at the cost of a linear increase in processing time.

[1]Tested on the Wall Street Journal collection indexed by TopSig, averaged over 10 searches.

[2]Not necessarily a valid assumption for real data. For example, TopSig signatures typically contain more off-bits than on-bits, skewing the signature distribution towards the lists that cover slices with fewer bits set. This is due to off-bits being the default state of bits in a TopSig signature; any bits unaffected by any of the terms in a document will assume this state. Bits that are set have an equal chance of being either on-bits or off-bits so the presence of this default state results in more off-bits than on-bits.

4. IMPLEMENTATION

The ISSL search platform was developed on top of the open source signature search tool TopSig and is now a part of that tool. The tool was developed in C using pthreads for multithreading support.

The platform was developed with a focus on 16-bit ISSL slices. As the main justification for ISSL is performance-related, slices widths that weren't powers of 2 were not considered and both 8-bit and 32-bit slices were deemed inappropriate as 8-bit slices would not be able to substantially reduce the amount of processing work required and 32-bit slices would impose too much overhead, making the approach worthless for all but the largest of collections.

Due to the performance focus of ISSL, the approach described was designed to work entirely within memory while building the ISSL table and while searching it. While it would be possible to create an implementation that works from an ISSL table stored on disk without reading it into memory first, this approach would take careful consideration as many random accesses of the table are required.

Building the ISSL table was designed as a two-pass process; the first pass is to determine how large each of the posting lists are for the purpose of memory allocation and the second pass is to fill them. Each list is stored as an array of 32-bit integers that uniquely identify the signature within a signature file. The $s \times 2^w$ lists[3], along with an integer giving the number of signatures in this list are written to a file for reading by the ISSL search tool.

The ISSL table can be read quickly if the on-disk representation and in-memory representation of the arrays of integers is the same. This will almost always be the case when the ISSL table is written and read on the same machine but may not be the case if the ISSL table is moved between machines of different endianness. The tool was designed to recognise this and fall back on a slower approach if necessary.

The amount of space taken up by the ISSL table scales linearly with both signature width and collection size. While a collection of 1,000,000 1024-bit signatures will take up $\frac{1000000 \times 1024}{8}$ bytes or $\sim$ 122 megabytes of space[4], the ISSL table for this same collection will require $4(1000000 \times 64 + 2^{16} \times 64)$ bytes, or $\sim$ 260 megabytes.

The potentially large sizes of the ISSL table and collection can impose limitations on the collections that can be used with ISSL depending on the hardware available as both signature and table are stored in memory[5].

The third data structure that uses a considerable amount of memory is the score table, which needs one element for each signature. For simplicity 32-bit integers were used for this implementation but 16-bit integers are fine if the signature width is lower than 2^{16}. This structure still ends up being much smaller than the other two, at only $\sim$ 4 megabytes for our example 1,000,000 signature collection. The score table is necessary as the score for any signature may be increased by any of the posting lists; it is not feasible to keep a top-k list or similar structure that only contains the highest scoring signatures. As table 1 shows, most of the signatures are touched even at relatively low search-breadth thresholds.

The ISSL search tool begins the process to search for a particular signature by resetting the score table to 0 and iterating through the search signature, one 16-bit slice at a time. For each slice, the Hamming distance neighbourhood array[6] is iterated through until the allowable search-breadth threshold is met. For example, if the allowable search-breadth threshold is set at 4, the first 2517 values (see table 1) of the Hamming neighbourhood array will iterated through in the process of scoring documents. These 2517 values will contain all possible 16-bit values with between 0 and 4 on-bits. The current value is combined with the search slice by exclusive or and the result, combined with the position of the slice within the signature, identifies one list in the ISSL table. This list is then iterated through and each signature that appears on the list gets its score incremented by $16 - n$ where n is the number of on-bits in the current value in the Hamming neighbourhood array.

After this process is complete, the score table will contain scores for each signature. The top-k scores can now be extracted from the array using a heap or similar structure[7]. The top-k signatures then have their scores recomputed with a full Hamming distance calculation and are sorted to produce the final results.

5. PERFORMANCE CONSIDERATIONS AND SCALABILITY

ISSL searches demonstrate an improvement in computational performance over the approach of calculating the Hamming distance with every signature in a collection; however, computational time and memory use still scale linearly with collection size, causing problems when dealing with large collections. However, like basic signature searching, this approach is also inherently conducive to parallel processing.

[3] s = number of slices in a signature. w = slice width (in bits). For example, a 1024-bit signature with 16-bit slices will have 1024/16 or 64 slices per signature. ($s = 64$, $w = 16$)

[4] Plus overhead; TopSig imposes a mandatory overhead of $33 + w$ bytes per signature, where w is the maximum length of any document identifier. This overhead can potentially be considerable.

[5] This implementation of ISSL stores both in memory. This is not necessarily required; for example, if a limited number of searches are going to be performed on a particular collection the scoring phase and the reranking phase can be separated with the intermediate results stored in memory or on disk. This implementation of ISSL was designed to create quick responses to individual queries while keeping all information in memory, therefore necessitating the storage of both signatures and ISSL in memory.

[6] To simplify determining the possible permutations of bits that may be toggled in expanding the search neighbourhood, the ISSL search tool precomputes a Hamming neighbourhood array of all possible 16-bit signatures and sorts the array by the number of on-bits in each value. A slice can be combined with these values with an exclusive or operation to produce slices with the required number of bits; combining the search slice with the first element will produce the original slice, while using last element will produce the inverse of the original slice. Intermediate values will produce all possible versions of the slice with all possible permutations of flipped bits, allowing all permutations of a slice with a certain number of bits changed to be accessed with little further computation.

[7] In this implementation the top-k scores are extracted from the array using a K-sized array that holds the K-highest scoring signatures seen so far and replaces the signature with the lowest score when a signature with a higher score is seen.

The TopSig implementation of ISSL developed for this study implements basic support for parallel processing using the pthreads library. The user can specify via configuration file or command line the number of threads that should be used for searching. The threads are created for each search and each is assigned an approximately equal portion of the posting lists in the ISSL table. Each thread processes its assigned lists, adding scores to the score table with atomic operations[8]. When all threads have terminated the main thread continues as normal, performing the top-k extraction and Hamming distance calculations without using extra threads. This operation is efficient as it can be implemented without locks.

The process of extracting the top-k elements from a list can also be parallelised and this may be beneficial for searching larger collections where this operation may consume a greater portion of the total processing time. Each thread can perform top-k extraction on separate portions of the list and the top-k lists can be merged at the end.

These approaches allow for performance improvements across a system with shared memory. For larger processing tasks, it may be useful to split the workload across multiple systems with independent memory. Signatures are also conducive to this task, as a signature file can be split into multiple parts and a separate ISSL created for each to divide the memory and processor burden across many machines. The final results can then be merged together providing the scores are preserved.

Performance approaches may vary depending on whether focus is on improving the performance of individual queries as would be desirable in an interactive system or on reducing the amount of time required to process a large batch of queries as may be desirable for cluster generation. While the previously discussed techniques are geared towards optimising the former, for the latter dividing up the queries among threads may be more effective than dividing up the collection. The tradeoff made involves increasing the latency between submitting the query and retrieving results in exchange for reducing the amount of overall processing time required.

6. EXPERIMENTAL RESULTS

The effectiveness of ISSL searching is measured in comparison to an exhaustive Hamming distance search on the same data set. As the purpose of the ISSL approach is to provide a more efficient way of finding the signatures with the lowest Hamming distance from the search signature, for the purpose of evaluation the fact that the Hamming distance inversely correlates with document similarity is considered a ground truth.

The standard metric to evaluating the performance of search engines is Mean Average Precision. (MAP) To get an analogue to MAP that works in the Hamming neighbourhood (where results are not marked as relevant/non-relevant, but we nonetheless know that lower distances are better,) we use a similar metric called Hamming Distance Ratio. (HDR) The HDR between the lists of top-k Hamming distances A and B is calculated as $\frac{1}{k} \cdot \sum_{i=1}^{k} \frac{\sum_{j=1}^{i} A_j}{\sum_{j=1}^{i} B_j}$. If A and B are the same, the ratio is 1; if B is based on an incomplete picture of the collection and hence shows larger Hamming distances in the top-k the ratio will drop proportionately; such that omissions closer to the top of the result list will exact a harsher penalty on the final score than omissions further down, much in the same way as MAP.

[8] __sync_fetch_and_add(), which is portable across Intel architectures. The C11 and C++11 language standards include support for atomic primitives; however, compiler support for these is still limited.

Breadth	Random (HDR)	WSJ (HDR)
0	63.44%	86.09%
1	63.56%	92.00%
2	74.55%	96.28%
3	89.48%	98.29%
4	95.69%	99.14%
5	98.97%	99.51%
6	99.59%	99.66%
7	99.94%	99.76%
8	99.98%	99.83%
9	99.99%	99.92%
10	99.99%	99.98%
11	100.00%	100.00%
12	100.00%	100.00%
13	100.00%	100.00%
14	100.00%	100.00%
15	100.00%	100.00%
16	100.00%	100.00%

Table 2: How early stopping affects performance-the HDR of searches on random signatures and signatures derived from the Wall Street Journal. Also see figure 2.

The quality of a given ISSL search is dependant on the search neighbourhood (the degree to which early stopping is performed.) As explained earlier, the search breadth also has implications on the processing time required to run individual queries.

Table 2 shows the results of $K = 100$ searches on two collections of identical size. (222,922 signatures.) The first collection consists of signatures created from random noise (created with a pseudorandom number generator) while the second collection consists of signatures generated from the Wall Street Journal (WSJ) collection with TopSig (using splitting, a process by which longer documents are split into multiple signatures.) We ran 60 searches on each collection, for each search choosing a random source signature from the same collection to search against and repeating this search 17 times for each search neighbourhood. The HDR ratios of each search were then averaged to produce the tabulated results.

The WSJ signature file was generated from the TREC Ad-Hoc Wall Street Journal collection (173,252 documents) using TopSig for indexing with the following settings:

```
SIGNATURE-WIDTH = 1024
#Size of signature (in bits)

SPLIT-TYPE = sentence
SPLIT-MAX = 512
SPLIT-MIN = 256
#Method of splitting long documents when creating
signatures. 'sentence' means to split between
```

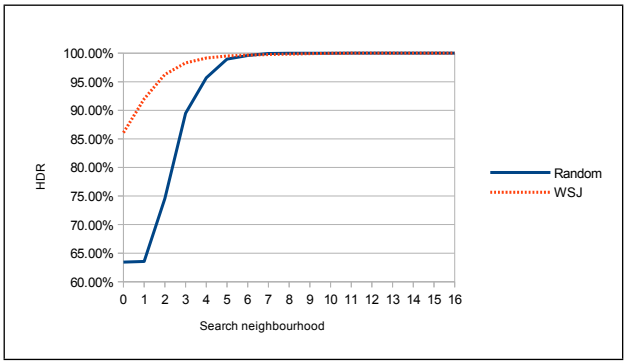

Figure 2: How stopping early affects random signatures and document signatures. Based on data from table 2.

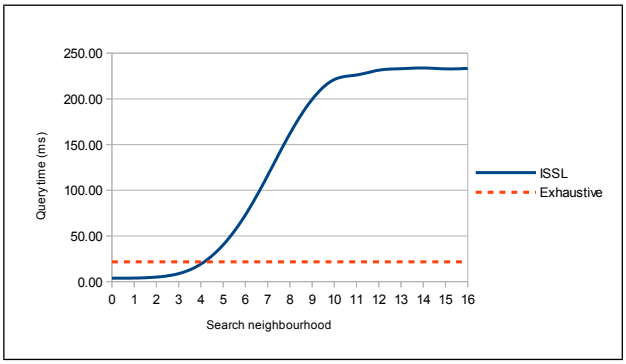

Figure 3: The relationship between early stopping and search time. Based on data from table 3.

```
'split-min' and 'split-max' terms if a full stop
appears; otherwise, split at 'split-max'.
#This generally increases search quality because
#long documents can result in signatures that are
#too noisy.

SIGNATURE-DENSITY = 21
SIGNATURE-SEED = 0
SIGNATURE-METHOD = SKIP
#Signature-density refers to the proportion of bits
#in the signature to set to either +1 or -1. SKIP
#is a method of signature generation and 0 is the
#random seed to start with for generating terms.

CHARMASK = alpha
#Characters in the file to treat as words for the
#purpose of indexing. Other characters are treated
#as whitespace.

STEMMER = porter
#Stemming method to use; this refers to the
#Porter stemming algorithm.

STOPLIST = data/stopwords.long.txt
#Words that appear in the stoplist are discarded.
#This stoplist is provided with TopSig.
```

The splitting configuration options result in a file containing 222,922 signatures, an expansion of $\sim 128.67\%$.

The random signature file was generated from the WSJ signature file using the create-random-sigfile tool packaged with TopSig, which generates a copy of an existing signature file with the actual signature data overwritten with repeated calls to the C standard library function rand().

These performance tests were run on an i7 950 4-core processor running at 3.06GHz.

The ISSL table generated for the each signature file is 70.4mb. This table was generated from the signature file in 7 seconds on the testing platform.

The greater HDR scores of document-derived signatures when early stopping is applied is due to the natural clusters that documents form, resulting in the closest documents in the WSJ collection being closer on average than those in the random collection. Lower signature distances are more effective in conjunction with ISSL because they have a higher likelihood of appearing in the early posting lists. A signature with a Hamming distance of 200 (assuming 1,024 bit signatures and 16-bit slices) will have an average of $\frac{200}{64} = 3.125$ bits changed per slice, meaning a search with a 3 or 4-bit breadth will likely find this signature in many posting lists. With a distance of 400, the average per-slice distance of 6.25 will require a larger search breadth to be effective.

Widening the Hamming neighbourhood by increasing the search breadth improves the HDR scores by bringing in more documents, but this improvement comes with a cost to search speed. Table 3 shows how searching the WSJ collection takes longer per query as the search breadth is increased. As figure 3 shows, the time spent on each search correlates with the number of posting lists (compare with figure 1) that need to be considered for that search breadth.

Multithreading was used with 4 threads and 10,000 queries used to amortise the average query execution time and reduce the influence of the overhead time required to read the signature file and ISSL into memory on the first run. The appropriate TopSig configuration information for each test was as follows:

```
SEARCH-DOC-FIRST = 0
SEARCH-DOC-LAST = 9999
#First and last signatures to search for. This is
#inclusive so these parameters will search for the
#first 10,000 signatures that appear in the
#signature file.

SEARCH-DOC-THREADS = 4
#Number of threads to use when searching. The main
#thread will spawn these threads and resume once
#these threads have completed their tasks.

SEARCH-DOC-TOPK = 100
#Number of documents to retrieve.
```

The time required to search this collection exhaustively using TopSig query search is 21.58ms, putting it approximately on par with using a 4-bit search breadth. It should be noted, however, that TopSig's exhaustive search is more mature and hence more optimised than the ISSL search process. There may be greater potential to increase search performance in this area.

Of course, as mentioned earlier, the search breadth is not the only way to improve search quality. The number of

Breadth	Query time (ms)	Lists per slice
0	3.74	1
1	3.88	17
2	4.98	137
3	8.74	697
4	19.24	2517
5	40.23	6885
6	72.95	14893
7	116.05	26333
8	161.84	39203
9	199.34	50643
10	221.15	58651
11	226.14	63019
12	231.37	64839
13	233.00	65399
14	233.78	65519
15	232.81	65535
16	233.32	65536

Table 3: The relationship between early stopping and search time. Searching more posting lists requires spending proportionately more time. Based on searches of the WSJ collection. Also see figure 3.

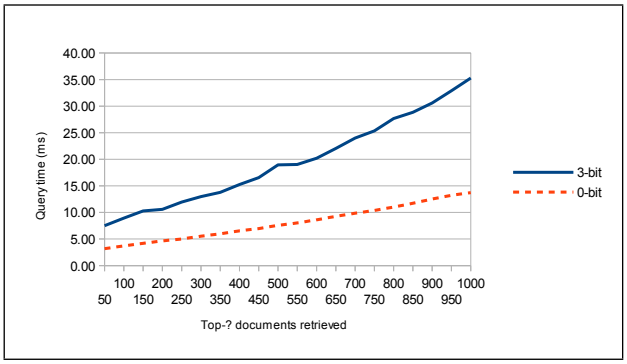

Figure 4: The relationship between top-k and search time. Based on data from table 4.

documents returned also matters as, when early stopping is invoked, the top-k documents are the ones reranked using the full Hamming distance. As the initial ranking that determines the top-k is based on incomplete information when early stopping is invoked returning more documents can return higher-quality results.

Table 4 shows how increasing the value of k increases both search time and HDR. The improvement from increasing the number of results returned is much greater for the 0-bit search breadth than the 3-bit search breadth; however, as the same table shows, the increases in HDR from increasing the search breadth are more efficient. For example, using a 0-bit search breadth and returning the top 500 results takes 7.57ms/query and gives a HDR of 92.04%, while using a 3-bit search breadth and returning the top 50 results takes 7.51ms/query and gives a 98.34% HDR. Therefore, while increasing the value of k can be situationally useful, greater benefits come from increasing the search breadth.

The Wikipedia XML Corpus[1] is a larger collection, consisting of 2,666,190 documents. When indexed with de-

blah	0-bit		3-bit	
Top-k	Time (ms)	HDR	Time (ms)	HDR
50	3.18	84.15%	7.51	98.34%
100	3.71	87.06%	8.94	99.12%
150	4.19	88.52%	10.29	99.35%
200	4.66	89.49%	10.59	99.48%
250	5	90.22%	11.96	99.56%
300	5.53	90.77%	12.97	99.61%
350	5.98	91.16%	13.77	99.65%
400	6.54	91.50%	15.27	99.69%
450	6.97	91.80%	16.55	99.71%
500	7.57	92.04%	18.94	99.73%
550	8.03	92.26%	19.03	99.75%
600	8.61	92.43%	20.21	99.77%
650	9.27	92.58%	22.05	99.78%
700	9.84	92.70%	23.99	99.79%
750	10.37	92.81%	25.34	99.80%
800	11.01	92.92%	27.68	99.81%
850	11.72	93.00%	28.84	99.82%
900	12.53	93.08%	30.6	99.83%
950	13.22	93.15%	32.9	99.84%
1000	13.74	93.21%	35.29	99.84%

Table 4: The relationship between top-k, search time and HDR. Increasing the number of documents results in a linear increase in search time. Based on searches of the WSJ collection. Also see figures 4 and 5.

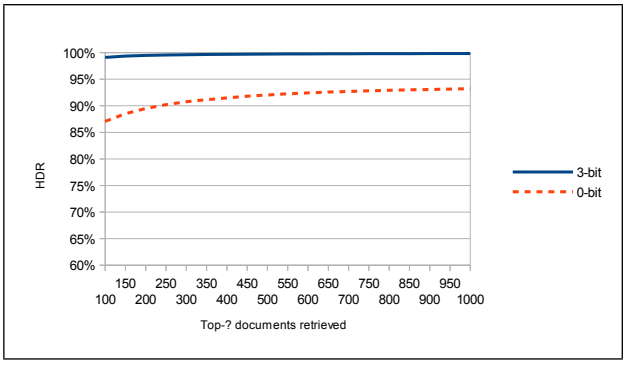

Figure 5: The relationship between top-k and HDR with 0-bit and 3-bit search breadths. Based on data from table 4.

Breadth	Query time (ms)	Ratio to WSJ
0	46.37	12.4:1
1	49.5	12.8:1
2	64.22	12.9:1
3	128.73	14.7:1
4	311.44	16.2:1
5	720.54	17.9:1
6	1406.73	19.3:1
7	2377.4	20.5:1
8	3339.28	20.6:1
9	4199.53	21.1:1
10	4762.97	21.5:1
11	4981.49	22.0:1
12	5135.17	22.2:1
13	5118.8	22.0:1
14	5096.39	21.8:1
15	5157.97	22.2:1
16	5142.36	22.0:1

Table 5: Time required to search the Wikipedia XML corpus. The ratio column shows how it scales in time compared to the search of the same breadth in table 3, a collection 1/16th of the size. Also see figure 6.

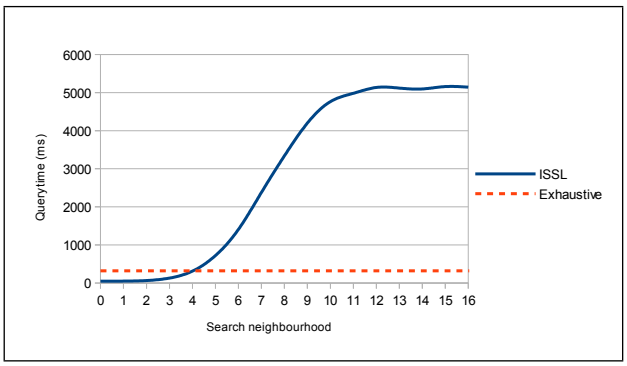

Figure 6: Time required to search the Wikipedia XML corpus. Based on data from table 5.

fault settings in TopSig the resulting signature file contains 3,606,901 signatures. Using 1024-bit signatures it results in an ISSL table 896mb large. It takes 1 minute and 30 seconds to generate the ISSL table on the testing platform.

This is a collection $\sim$ 16$\times$ the size of the Wall Street Journal collection and, as shown in table 5, takes correspondingly longer to query. This makes sense, as the two time-consuming portions of the ISSL task (adding scores from the posting lists and extracting the top-k) both scale in complexity linearly with the size of the collection.

The time required to search this collection exhaustively using TopSig query search is 320.4ms, once again putting it approximately on par with using a 4-bit search breadth.

So far we have not tested ISSL searching on any larger data sets, such as ClueWeb09B. Extrapolating from the data we have already, assuming the linear increase in query time we saw going from WSJ to Wikipedia holds we can estimate that it will take 1.6 seconds per query to search the 50 million document collection on the testing platform.

7. LIMITATIONS

The nature of the ISSL approach results in certain limitations that do not apply to an exhaustive Hamming search.

First of all, the requirement that the ISSL table remain in memory during processing (as the search process involves highly random access) limits the use of ISSL for collections that expand to an ISSL table that is too large to fit in memory. For clustered approaches the collection can be split over multiple machines with a separate ISSL table created for each, simultaneously improving search time and reducing the memory requirement for the individual machines; however, when only one machine is available it may be faster to use on-disk exhaustive search as opposed to on-disk ISSL search.

Second, the ISSL approach is limited to cases where searching with the entire search signature is desirable. While the individual slices can be turned on and off as required (to implement field-based indexing and search, for example) there is no feasible way to facilitate disabling the individual bits of a signature. This is a serious drawback for query-based searching, which relies on masking off bits that are unset in the query signature (which, by nature will be very sparse due to containing far fewer terms than a document) as both off- and on-bits in a signature indicate the possible presence of certain terms. Performing a dense search with a query signature will result in false matches against documents with more off-bits.

Finally, ISSL searching loses most of its advantages when dealing with collections where typical searches result in large Hamming distances to the closest signatures. Unfortunately, this limits the use of ISSL in situations other than near-duplicate detection. This is due to the number of differing bits per slice frequently being too high to be captured in the search breadths (3 or small) that provide the greatest performance improvements over exhaustive searching.

8. REPRODUCIBILITY

The ISSL search tool was built into TopSig, which is available at `http://www.topsig.org` in the SVN repository. It is open source and available under the GPLv3 license. Except where otherwise declared, all configuration settings remain at their defaults. The Wall Street Journal corpus is part of the TREC Research Collection volumes 1 and 2. The Wikipedia XML Corpus is available from `http://www.mpi-inf.mpg.de/departments/d5/software/inex/`.

9. CONCLUSION

We have presented the Indexed Signature Slice List approach to improving the speed of signature searching without a considerable loss to search fidelity. While the effective use of ISSLs is limited to certain situations (such as near-duplicate detection and some clustering approaches) in those situations they can provide great increases in speed over traditional exhaustive approaches. As the cost of searching the ISSL table grows linearly with the size of the collection it cannot be said that the scalability problem of signature-based searching has been solved; however, it can be said that searching is a bit faster with this approach.

10. REFERENCES

[1] Ludovic Denoyer and Patrick Gallinari. The Wikipedia XML Corpus. *SIGIR Forum*, 2006.

[2] C. Faloutsos and S. Christodoulakis. Signature files: An access method for documents and its analytical performance evaluation. *ACM Transactions on Information Systems (TOIS)*, 2(4):267–288, 1984.

[3] S. Geva and C.M. De Vries. Topsig: Topology preserving document signatures. 2011.

[4] R.W. Hamming. Error detecting and error correcting codes. *Bell System technical journal*, 29(2):147–160, 1950.

[5] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. Detecting near-duplicates for web crawling. In *Proceedings of the 16th international conference on World Wide Web*, pages 141–150. ACM, 2007.

[6] J. Zobel, A. Moffat, and K. Ramamohanarao. Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems (TODS)*, 23(4):453–490, 1998.