Indexed Signature Slice Lists for Nearest-Neighbour Determination

ABSTRACT

File signatures allow a document to be represented with a signature of fixed length that shares the topological properties of the original document and can be used for efficient document similarity computations using the Hamming distance[4] between the two signatures. While this approach produces an acceptable measure of similarity, performing searches requires performing a Hamming distance calculation against every signature in a collection, which quickly becomes excessive when dealing with web-sized collections. This paper describes a method of indexing slices of signatures to efficiently find similar documents in a collection. Experimental results suggest that indexed signatures are a competitive approach to the problem of finding the signatures with the smallest Hamming distance from the search signature with high efficiency and a low error rate.

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 by Geva et al.[3] are a refinement on the approach of using bit signatures to represent documents explored by Faloutsos and Christodoulakis[2] and shown to be inferior to inverted file approaches by Zobel et al.[5]. Topological signatures consist of a document-term matrix projected into a randomised subspace to produce a new, lower-order matrix (that can be efficiently stored) while preserving the statistical properties of the original document-term matrix. The new matrix is then flattened into a bit string with positive and negative values being reduced to 1 and 0 bits respectively. (with zero values also reduced to bits with a value of 0.) Queries can then be projected into the same space, flattened and the Hamming distance[4] between the query and document signatures can be used to approximate the sum of the cosine similarities between the term vectors present in both the original document and query.

This approach to signatures is a refinement on the approach discredited by Zobel et al.[5], which is essentially a bloom filter, although not explicitly described as such in the original paper[2]. Terms are hashed to create pseudorandom sparse bit strings that are combined together (through a bitwise OR filter) to create a signature. Queries are then tested by comparing the bit strings the terms in the query hash to with the document signature. If all the bits appear, there is a possibility that the term is in the document. As 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. They also require extensive amounts of storage space to ensure signatures are long enough to reduce the occurrence of these false positives.

While topological signatures fix some of the criticisms levelled at document signatures in terms of retrieval performance

[3], the approach required to search a collection (performing a Hamming distance calculation against every signature) quickly becomes prohibitive when collection sizes become large enough. An approach is therefore needed to avoid performing this calculation against every signature, while still retrieving most of the nearest signatures.

2. INDEXED SIGNATURE SLICE LISTS

Indexed Signature Slice Lists (ISSLs) are an approach similar to both inverted files and clustering. The purpose of an ISSL file 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.

Max error	Lists checked	% 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 slice lists checked based on the maximum error allowed. The third column shows the average portion of the collection covered by these lists[1]

To store a signature in an ISSL file, the signature is first divided up into slices. For efficiency, 16 bits works well, but signature slices of other sizes are possible. The combination of the slice position and the value of the signature slice is then used to select an ISSL to list the signature in. A single signature will be listed in one ISSL for each slice the signature is divided into. In the case of a 1024-bit signature and 16-bit slices, each signature will be divided into 64 slices and therefore be placed in 64 lists out of a total of 64×2^{16} lists.

When searching an ISSL file for signatures close to a search signature, the search signature is divided into slices in the same way and each slice is used to look up an ISSL. All the signatures listed in the retrieved ISSLs are recorded. This means that a signature will be a possible candidate if any of the slices match, bit for bit, the associated slice of the search signature.

While this step has a good chance of retrieving some of the nearest signatures, it is not sufficient. The search is therefore repeated with the search slices slightly altered, with each possible 1-bit error applied to the search slice. For 16-bit slices, this means an extra 16 searches per slice. This can then be repeated for every 2-bit error, every 3-bit error and so forth up to the slice length if desired; with $\sum_{i=0}^{k} {}^{n}C_i$ lists checked when allowing for k bits of error in a search consisting of n-bit slices.

Each signature is assigned a score for the purpose of this search. Each score starts at 0 and is incremented by the difference between the slice width and the Hamming distance between the search slice and the slice associated with the list in which this signature appeared. When the maximum error allowed is equal to the slice width, the score for each signature will be equal to the difference between the signature width and the Hamming distance. When the maximum error is lower, the amount of processing time required is reduced as fewer lists need to be considered. Assuming an equal distribution of signatures per list[2] allowing a maximum error of 3 bits while using 16-bit slices will result in 1.06% of the work being done than would be required for a maximum distance of 16, but with potentially less accurate scores.

After the scores for all signatures have been determined, the highest scoring signatures can be reranked using a full Hamming distance calculation to ensure that the ranking of these signatures is correct. At higher levels of maximum error, 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 results from document similarity searches to 10 documents chosen with the first 10 5-digit numbers from A Million Random Digits with 100,000 Normal Deviates.

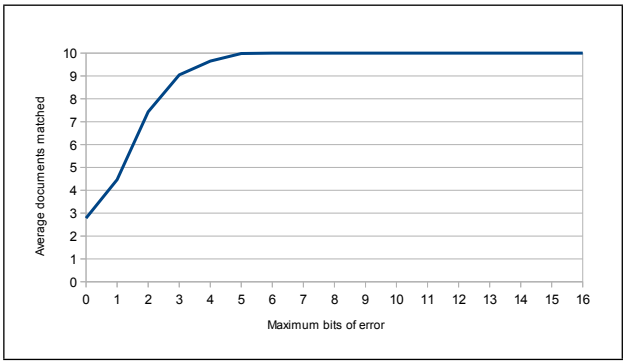

Figure 1: Out of the top 10 results returned, the number of results that match the actual top 10 results (ordered by Hamming distance)

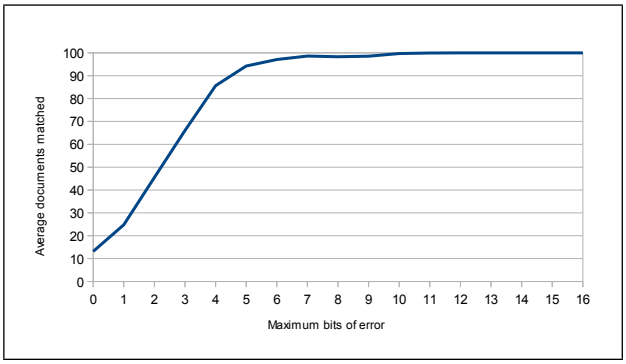

Figure 2: Out of the top 10 results returned, the number of results that match the actual top 100 results (ordered by Hamming distance)

3. IMPLEMENTATION

[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.

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 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 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 scores are touched even at relatively low error 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 error array[6] is iterated through until the allowable error threshold is met. For example, if the allowable error threshold is set at 4, the first 2517 values (see table 1) of the error 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 error value.

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.

4. 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.

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 lists in the ISSL table. Each thread processes its assigned lists, adding scores to the score table with atomic opera-

[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 ISSLs in memory.

[6] To simplify determining the possible permutations of 1-bit, 2-bit, 3-bit errors and so on, the ISSL search tool precomputes an 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 the errors required; the first element will produce no error, while the last element will flip all of the bits of the search slice.

[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.

tions[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.

5. EXPERIMENTAL RESULTS

6. CONCLUSION

7. REFERENCES

[1] J. Callan, M. Hoy, C. Yoo, and L. Zhao. Clueweb09 data set. *boston. lti. cs. cmu. edu, Jan*, 2009.

[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] 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.

[8] __sync_fetch_and_add(), which is portable across Intel architectures.