Indexed Signature Slice Lists for Fast Nearest-Neighbours 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[3] 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 by Geva et al[2] are a refinement on Locality Sensitive hashing and the approach of using file signatures to represent documents, originally explored by Faloutsos and Christodoulakis[1]. These initial approaches were shown to be inferior to inverted file approaches by Zobel et al.[4] Topology preserving signatures in our work are derived from a document-term matrix, randomly-projected onto a compact binary (+1,-1) 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 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 projection is that it largely preserves the relative pairwise distances between points in input space, on the hypercube. Furthermore, Geva et al describe an effective approach to performing ad-hoc keyword based search with signatures, that overcomes the difficulties identified in earlier models. A complete description of this approach is given in [2], 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 DFR.

For the sake of completeness, we briefly describe some details about 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[1]. 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 of [2], 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 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 the nearest signatures.

2. INDEXED SIGNATURE SLICE LISTS

Indexed Signature Slice Lists (ISSLs) is a hybrid approach that combines the inverted file approach with the signature file approach. 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. In a normal 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. In an analogous manner, ISSL are posting lists, where signature slice values correspond to terms, and the postings correspond to signatures in the collection. The signature slices of a query signature are used to combined the respective ISSL postings during search to provide a ranked list of signatures.

The creation of ISSLs 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 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 integer value of the signature slice. The posting lists contain the signature IDs of the represented signatures.

When searching an ISSL file for 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 signature IDs listed in the retrieved ISSLs are consolidated. Trivially, when searching for a specific signature that was previously indexed, we would find the signature ID in each of the posting lists corresponding to its 64 slices. In general however 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. 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 we need to process the postings in a different manner to the way we do with conventional term-based search.

We need to consider not only the matching ISSL postings, but also their neighbourhood. With 16-bit slices, we have 1 matching slice having the exact value, 16 slices that have a value that is 1 bit away, 120 slices that have a value that is 2 bits away, and so on. The search is therefore repeated with the search slices progressively altered to cover a wider Hamming neighbourhood. As the lookup of postings progresses, we tally the worst-case Hamming distance of signatures encountered in postings. Initially all signatures are assumed to be at a distance of 1024. As search progresses, all signature IDs that have not yet been encountered in any of the postings have a worst case Hamming distance of 1024. On the other hand, a signature ID that is observed in a particular posting immediately receives a more optimistic estimate of its distance. For instance, if a signature is observed in a slice with a distance of 3 bits away from a search 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 all slice positions and their distances becomes known with full precision. 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.

This brings us to the key idea in this paper: we conjecture, and test the conjecture, that it is not necessary to proceed with the calculation until complete information is available, in order to determine with high accuracy the neighbourhood of a search signature. That is; we can stop the calculation early and use the worst-case distance estimation to choose the candidate list of nearest neighbours. The desired length of the list of nearest neighbours is more often than not very short (it does not usually depend on the collection size, but rather on the users' capacity to review long result lists.) Therefore, once we identify the candidate results list we can afford an accurate Hamming distance calculation and a final sort to produce our final ranked result list.

The number of posting lists consulted is of course a function of the search-breadth in bits. Table 1 provides the number of ISSL postings that have to be consulted as function of search-breadth. It also contains the percentage of distinct signature IDs encountered while processing the list. This corresponds to the Wall Street Journal collection of document signatures.

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 search-breadth (the number of bits that can differ between the search slice and the signature slice) is equal to

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

Bits changed	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 number of bits toggled in the search slice. The third column shows the average portion of the collection covered by these lists[1].

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 search-breadth 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 search-breadth 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 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.

3. 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 ~ 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

[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.

[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.

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 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 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.

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 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.

5. EXPERIMENTAL RESULTS

6. CONCLUSION

7. REFERENCES

[1] 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.

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

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

[4] 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.

[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.

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