Efficient Top-K Retrieval with Signatures

Timothy Chappell
Faculty of Science and Technology
Queensland University of Technology
Brisbane, Australia
t.chappell@connect.qut.edu.au

Shlomo Geva
Faculty of Science and Technology
Queensland University of Technology
Brisbane, Australia
s.geva@qut.edu.au

Anthony Nguyen
Australian e-Health Research Centre
CSIRO
Brisbane, Australia
Anthony.Nguyen@csiro.au

Guido Zuccon
Australian e-Health Research Centre
CSIRO
Brisbane, Australia
Guido.Zuccon@csiro.au

ABSTRACT

This paper describes a new method of indexing and searching large binary signature collections to efficiently find similar signatures, addressing the scalability problem in signature search. Signatures offer efficient computation with acceptable measure of similarity in numerous applications. However, performing a complete search with a given search argument (a signature) requires a Hamming distance calculation against every signature in the collection. This quickly becomes excessive when dealing with large collections, presenting issues of scalability that limit their applicability.

Our method efficiently finds similar signatures in very large collections, trading memory use and precision for greatly improved search speed. Experimental results demonstrate that our approach is capable of finding a set of nearest signatures to a given search argument with a high degree of speed and 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, Hamming Distance, Locality-Sensitive Hashing, Nearest Neighbour, Top-K

1. DOCUMENT SIGNATURES

Document signature approaches to information retrieval involve representing documents, images and other searchable abstract objects as binary strings of fixed length, called signatures. The signatures are derived in such a way that they preserve in signature space the topological relationships that exist in the original representation of objects. Locality preserving hashing means that these representations are locality-sensitive - similar documents or objects in the original representation will also have similar signatures, and the converse is true as well. This is in contrast to conventional hashing approaches that are designed to produce different hashes for any pair of documents (even if the documents are very similar).

Locality-sensitive hashing (LSH)[8] is a method of dimensionality reduction using binary signatures. The original representation of an information object 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. The binary signatures used in LSH methods offer a compression of the original representation onto a dense, fixed, low-dimensional representation. LSH allows for comparing two different segments of source text for similarity far more efficiently than traditional methods, such as cosine similarity, especially if the source texts are large. In fact, a relatively expensive cosine similarity computation between two documents can be replaced with a Hamming distance [4] calculation (counting the bits that differ between the two signatures). This efficiency motivates most applications of LSH and signatures, such as in information retrieval [3] and near-duplicate detection [6]. There are a number of different document signature models in use, including Minhash [1], Simhash [7], Topsig [3] and Reflexive Random Indexing [2], all of which use some variation of LSH.

While there are many aspects of document signatures that make them attractive for a variety of purposes, performing a search with a given search argument (a signature) requires a Hamming distance calculation against every signature in a collection. Even though the individual comparisons can be performed using a small number of efficient machine instructions, this approach scales poorly to increasingly large collections.

This paper is not concerned with the relative merits of different approaches for signature generation; instead, we focus on the common problem on how to search collection of signatures efficiently.

To address this problem we present an inverted list ap-

ADCS '13, December 05 - 06 2013, Brisbane, QLD, Australia

ACM 978-1-4503-2524-0/13/12 ...$15.00
http://dx.doi.org/10.1145/2537734.2537742

proach that utilises additional initial processing time and memory to subsequently reduce the processing time required to locate the closest document signature(s) to a given search signature. This approach was motivated by the fact that any approach that needs to consider all signatures will eventually be too slow to handle larger collections. This is the same motivation behind inverted file approaches commonly used in information retrieval that allow all documents that contain a particular term to be identified; we chose a similar approach, but inverting the signatures file by signature-substrings instead of terms. This allows us to continue using the signature representation of documents while gaining some of the search efficiency benefits of inverted file approaches. Experimental results suggest that, while the memory usage can be considerable, search performance of at least 25× that of exhaustive approaches is achievable.

2. INVERTED SIGNATURE SEARCHING

Any approach that requires Hamming distances to be computed between the search signature and every signature in a collection is inherently of linear complexity. Regardless of how efficient the Hamming distance computation is, increasing the size of the collection means increasing the processing time required to search it by the same factor. This can quickly become computationally infeasible; an approach is therefore required that ignores a large portion of the document collection and calculates Hamming distances to the remaining signatures, knowing the signatures that have been ignored are unlikely to be close to our search signature.

Substring indexing

Each signature, identified by a signature-id (a number), is a binary bit string comprised of a sequence of binary substrings of fixed sizes. For simplicity of explanation we shall assume that a signature length is some power of 2 in length, and is evenly divided into a set of k substrings of equal length, also some power of 2 in length; this is not a constraint and indeed later in this paper we describe experiments where this is not the case. A signature is thus regarded as a sequence of s substrings, $N_1, N_2, ..., N_s$. Each of the substrings is n bits long, and the signature length is $s \times n$.

We now invert the entire collection of signatures, storing them by substring position, and by the respective n-bit substring value. This means that for each of the k substring positions we have 2^n posting lists. Each posting list consists of all signature-ids where the signatures have the corresponding n-bit substring in the corresponding position.

For example, consider a signature size of 32 bits and a substring size of 8 bits. Given the substring 01110101 in position 2, we can find in the corresponding posting list (position 2, value 117) the signature ids of all signatures in the collection that have that value in that position. We note that there are $2^8 = 256$ posting lists for each position, and with 4 substring positions we have 1024 posting lists in total. This is independent of the collection size. With smaller collections or greater substring sizes it is possible for some posting lists to be empty, corresponding to the situation that some signature substrings do not occur in some positions anywhere in the collection of signatures. In general we choose the length of the substrings (n) such that the posting lists will be relatively short while most posting lists are non-empty. This depends trivially on the size of the collection - the larger the collection the larger we choose n.

Consider a collection of N documents and a choice of n bits per substring. If we double the collection size and increase n by 1, we expect the average posting list length to be about the same. This is due to the number of lists (2^n) doubling with each increment and hence providing twice as many locations for a given signature to be placed. It is immediately clear that for small values of n we can scale collections to astronomical proportions and still keep substring size small. It is important, for reasons that will become clearer later in this paper, to choose n so as to optimise the length of the posting lists, trading off processor memory requirements for processing speed. The data structure for holding the inverted lists can be as simple as an array of pointers whereby each pointer position in the array is computed directly from the position and value of a corresponding signature substring. Each pointer then leads to an array of corresponding signature-ids. This facilitates direct access to posting lists in memory and we shall refer to it as a lookup table. This lookup table can be created in a preprocessing stage and it can be stored on disk and read into memory at system start-up (a once-off small cost) to support subsequent searching operations.

Substring searching

With the lookup table available, finding an exact match to a given search signature becomes a very efficient task; for a 64-bit signature, and choosing $n = 8$, eight lists would need to be consulted to find a signature that matches; finding a matching signature is simply a matter of finding the signature-id that has a match in all 8 positions by consulting the respective posting lists for these positions and values. While each of these lists would need to be scanned sequentially, only a fraction ($\frac{1}{2^8}$) of the substring signatures in the full collection will be accessed, and the average length of a posting list is $\frac{N}{2^8}$ each. This results in a substantial reduction in the amount of work required to perform the search. For suitable choices of n this is computationally much less expensive than comparing every signature to the search argument.

It is possible to trade progressively larger amounts of memory for increased performance by working with larger choices of n, at the cost of requiring more (albeit shorter) posting lists to be stored.

Performing an exact signature lookup in this fashion is essentially a specific case of the general problem of performing an exact Hamming distance search. This is, naturally, a very limited special case and is of limited usefulness, but we can do much better with the data structure as we now explain. The more general problem is to find signatures in the nearest Hamming-distance neighbourhood of a search argument. We present a solution to the Hamming distance search problem which is based on these lookup tables and can be performed so efficiently that the overall search time will still be much smaller than what would be required for an exhaustive search.

Extending the search neighbourhood

Consider the previous example, but with the Hamming distance neighbourhood threshold set to 1 instead of 0. The increase in threshold values means that in addition to any signatures that precisely match the search signature, we also want to identify any signatures that differs from the search

signature by 1 bit.

A simple approach is to extend the search to lookup not only the exact-match 8-bit substrings, but also lookup all substrings that are one bit away. This is often referred to as *query expansion*. For each 8-bit substring in the search argument we need to allow for 8 near patterns that are only 1-bit away. In this case, in addition to searching the 8 position lists having the same substrings as our search argument, we also need to consult additional $8 \times 8 = 64$ lists. These lists consist of the 8 possible permutations of the bit substring for each of the 8 positions, requiring a total of $8 + 64 = 72$ lookups instead of 8. In this example, a signature has a Hamming distance of no more than 1 providing it is a precise match in all positions, or it appears in 7 of the 8 positions with a precise match, and in the unmatched position it appears in one of the 8 lists that are one bit away from the search argument in this position.

This approach is far more efficient than linear searching, but it is still very limited. For instance, all 2-bit permutations of the 8-bit search patterns need to be consulted if this approach is extended to a Hamming threshold of 2; this would in turn require to check 8×56 additional lists on top of the 72 as previously described. If the desired Hamming threshold is 8, every list needs to be consulted and no gain would be made compared to an exhaustive search. To ensure that every signature in this collection within a Hamming distance of 8 from our search signature is located, the exhaustive approach is necessary. However, under most conditions it is not necessary to locate every signature that is within a certain distance of the search query. A more common type of search is finding the top-k nearest neighbours, where the k closest signatures should be returned, regardless of their actual distance. We can take advantage of this by limiting the number of permutations of each substring we consult. As we gradually extend the neighbourhood search, the Hamming distance of signatures grows, and it becomes less likely for a signature not yet seen in some positions to be one of the k closest signatures. This is essentially the basis for the approach we present in this paper. We explain it in more detail in the next section.

2.1 Top-K signature searching

Given a search argument (i.e., a signature) we aim to find neighbouring signatures using the lookup table. A *signature list* in the following description is a reference to a posting list for a given substring position and a given n-bit substring. Henceforth we will be describing a more realistic case of using 1024-bit signatures (a signature width shown to be sufficient for information retrieval [3]) and 16-bit binary patterns. With signatures 1024 bits wide, we need to consider the neighbourhood in each of the 64 positions of the search argument. We consider not only the exact-matching signature list, but also its Hamming neighbourhood. Given a specific signature substring, for each position we have one signature list that matches the search pattern, 16 signature lists that have a value that is 1 bit away, 120 signature lists that have a value that is 2 bits away, and so on (these are the Newton binomial coefficients). The search is therefore expanded with the set of 16-bit substrings that covers a wider Hamming distance neighbourhood.

Hamming distance approximation

The determination of the Hamming distance of signatures to the search argument proceeds by consulting the table and keeping track of the distance estimate of signatures in the collection. Each time a signature ID is observed in a list, more information is revealed about its distance from the search argument. Clearly, if a signature ID is observed in all 64 positions, its distance to the search argument can be computed with complete accuracy. However, if the signature is only observed in N of the 64 positions, the distance is known accurately only in the subspace of those N from 64 positions, and in $64 - N$ positions it is unknown. We can however compute a worst-case estimate of the distance by assuming that the distance is maximal in all unobserved positions. We can also compute a best-case distance by assuming that the unobserved positions 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 positions, in the best-case scenario we shall find the signature in the 4-bit neighbourhood in all the remaining positions. Then, the most optimistic estimate of the distance is obtained by adding $4 * (64 - N)$ to the known distance from the N seen positions. For the pessimistic estimate of the distance, $16 * (64 - N)$ is added to the known distance from the N seen substrings (i.e. we pessimistically assume it will only be seen at the 16-bit neighbourhood in remaining substrings).

With the above procedure, 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. Continuing to expand the search neighbourhood will eventually result in perfect determination of the nearest neighbours; however, at a severe cost to performance.

Early stopping

While stopping the neighbourhood expansion early will provide a neighbourhood estimation based on only an incomplete picture of the collection, we note that sometimes even partial information will suffice to precisely identify the top-k nearest neighbours. It is possible to stop the search as soon as we have observed k distinct signatures in all their 64 positions and hence have the exact top-k signatures. We note that distances of remaining signatures can only increase as we expand the search. 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 lists per substring, $64 * 17 = 1088$ lists in total), we have already collected the top-10 neighbours with complete accuracy and we can stop.

Seeing all top-k signatures in all positions with short distances is a very strict requirement though; it turns out to be too strict to be of any practical use. We conducted experiments with 1 million random 1024-bit signatures, searching for the top-16 neighbours of one of the signatures. We stopped consulting the table when the distances of the top 16 signatures are completely determined from the signature lists. The findings of this experiment suggest that the search can be stopped only after consulting the 14-bit to 15-bit neighbourhood: this is far too expensive, effectively consulting all but 64 inverted lists from 1088, and thus this

approach is not viable.

However, by keeping track of the best-case and worst-case distances, we can stop the search when the best-case distance of the $k + 1^{th}$ nearest signature is larger than the worst-case distance of the k^{th} nearest signature. At this point the top-k can no longer change. Note that we do not know the exact distances of the top-k yet, but we can easily compute that directly since we now only need to perform k distance calculations and this is a negligible cost (a small fraction of a millisecond). 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 the 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 still not a viable approach either and requires processing the vast majority of posting lists.

However, it is not necessary to progress to the point of complete certainty either. 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 positions will subsequently be found at the 4-bit distance in all of the 64 positions. Conversely, it is possible but highly unlikely that a signature that was observed at a distance of 0-bits in 60 substrings will not be found until the 16-bit distance in the remaining 4 unseen positions; it is highly likely to be observed sooner rather than later as we expand the neighbourhood in these positions. This is due to the nature of random indexing; any given set of bits is far more likely to be spread throughout the signature and cover multiple substrings than appear in the same substring.

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 that a relatively small set of the nearest signatures, say M, will contain the top-k signatures with high probability. Furthermore, we conjecture that when this set 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 the lists that correspond to a small Hamming distance neighbourhood, a set of M nearest signatures is selected, having $M >> k$, and an exhaustive distance calculation over these M signatures is used to identify the top-k signatures. It is often the case that k is relatively small. This is because k 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. If k is small, we can choose $M >> k$ 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 $k << M << N$ where N 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 positions inspected maintain a worst case Hamming distance of 1024. On the other hand, a signature that is observed in a particular position immediately receives a more optimistic estimate of its distance. For instance, if a signature is observed in a list corresponding to a distance of 3 bits from the search argument substring 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 substring positions and their distances become known with high accuracy. Of course, if the process is carried through until all lists are consulted then the distance of all signatures from the search argument is precisely known.

Bits changed	Lists per substring	% 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 substring based on the number of bits changed in the search substring. The third column shows the average portion of the collection covered by this neighbourhood in the Wall Street Journal collection.

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 signature lists, per position, 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. The percentage figure corresponds to the fraction of document IDs seen as the signature lists are processed. For instance, when the signature lists are scanned up to a 3-bit Hamming distance from the search argument, 697 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 postings 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 shall demonstrate, in practice the approach is highly accurate and the tradeoff is very attractive.

[2]Not necessarily a valid assumption for real data. Locality-

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; for instance, increasing M, the number of signatures that are reranked using exact Hamming distance calculation, may result in fewer top documents being omitted at the cost of a linear increase in processing time.

3. IMPLEMENTATION DETAILS

3.1 Data structures

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

The amount of space required to store the table scales linearly with both signature width and collection size. While a collection of 2^{20} 1024-bit signatures will take up $\frac{2^{20} \times 1024}{8}$ bytes or 128 megabytes of space, the inverted table for this same collection will require $4(2^{20} \times 64 + 2^{16} \times 64)$ bytes, or 272 megabytes. In this case, this means an overhead of 272 bytes per document; typically a small fraction of the total size of the original collection.

The potentially large sizes of the inverted signature table and signature file can impose limitations on the collections that can be used with this approach depending on the hardware available as, for performance reasons, both the table and the signature files may be stored in memory. This is to allow the exact Hamming distance of a given signature from the search signature to be efficiently obtained for reranking purposes.

The third data structure that uses a non-trivial amount of memory is the score table, which needs one element for each signature. We used 32-bit integers but smaller or larger integers can be used depending on the collection size. This structure still ends up being much smaller than the other two previously described, at only ~ 4 megabytes for our example 2^{20} 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.

Note that, for explanatory purposes, the following implementation description assumes a signature width of 1024 bits and a signature position substring width of 16 bits. These are not implementation limits and tests have been performed with a variety of substring widths, including those that are not factors of the signature width.

3.2 The search process

The 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 position at a time. For each position, the Hamming distance neighbourhood array is iterated through until the allowable search breadth threshold is met.

To simplify determining the possible permutations of bits that may be toggled in expanding the search neighbourhood, the search tool precomputes a Hamming neighbourhood array of all possible n-bit signatures and sorts the array by the number of on-bits in each value. This is a one off cost at system start up, not a query time cost. A substring can be combined with these values with a XOR operation to produce substrings with the required number of flipped bits. As an example, if the allowable search breadth threshold is set at 4, the first 2517 values (see Table 1) of the Hamming neighbourhood array will be 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. Each value in turn is combined with the search substring by bitwise XOR and the resulting value, combined with the position of the substring within the signature, identifies one posting list in the inverted signature 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 corresponding entry 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[4]. The top-k signatures then have their scores recomputed with a full Hamming distance calculation and are sorted. The resorted list then becomes the final result.

4. EXPERIMENTAL RESULTS

The effectiveness of inverted table searching is measured in comparison to an exhaustive Hamming distance search on the same data set. It should be noted that we are not evaluating the utility of the signatures *per se*; we only evaluate our ability to identify the Hamming neighbourhood of a search argument accurately. The utility of signatures in signature search is instead related to the application that uses signature approaches (e.g. duplicate detection, image clustering, etc.): this evaluation is left for future work. As the purpose of the inverted table 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 quality evaluation we take the true Hamming neighbourhood as the quality ground truth. If we are able to compute it precisely then we have perfect performance. Furthermore, the degradation in performance can be simply measured by the Hamming distance discrepancy observed between the ground truth top-k signatures, and the top-k signatures that our search identifies.

sensitive hashing will naturally result in similar documents producing similar signatures - natural clustering - and some spatial Hamming neighbourhoods will have more signatures than others.

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

[4] In our 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.

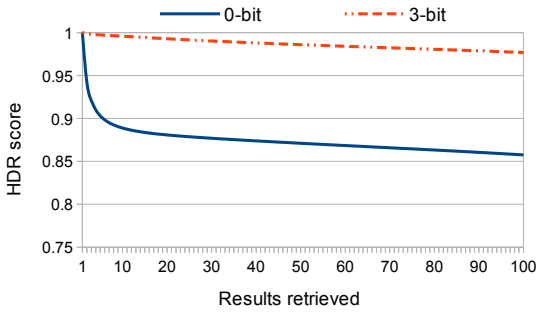

Figure 1: How the HDR score of a search (with a 0- or 3-bit Hamming neighbourhood) evolves as more of the results are compared. Errors found early have the largest effect on the final score. Based on 50 searches of the Wall Street Journal collection.

A standard metric used in evaluating the performance of search engines is Average Precision [9] or Normalised Cumulative Gain [5]. To get an analogous measure that works in the Hamming neighbourhood we introduce a similarly behaving metric called Hamming Distance Ratio (HDR). The metric is based on the calculation of the cumulative Hamming distance from the search argument of the Top-k signatures, having the cumulative distance of the true Top-k serve as the baseline for normalisation. Consider A - the list of ranked true Top-k signatures, and B - the list of ranked approximate Top-k signatures. We calculate the cumulative Hamming distance ratio (HDR) of the lists at successive rank positions as follows:

$$HDR(a,b) = \frac{1}{k} \cdot \sum_{i=1}^{k} \frac{\sum_{j=1}^{i} A_j}{\sum_{j=1}^{i} B_j} \qquad (1)$$

If A and B are identical, 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. As Figure 1 shows, errors have their largest effect on the Hamming distance ratio early on as omissions closer to the top of the result list penalise the final score more than omissions further down, working in much the same way as Average Precision. An exhaustive search would obtain a HDR score of 1 and would be represented by a horizontal line at 1 (the equivalent of a *line-at-one* in a recall-precision plot, for a perfect retrieval result).

The quality of a given inverted table 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 searches retrieving the top 100 results on two collections of signatures; one filled with randomly-generated data, the other created using the TREC Wall Street Journal collection as a source. We ran 60 searches on each collection, choosing a random source signature from the same collection to search against and repeating this search 17 times, once for each search neighbourhood (0-bit to 16-bit). The HDR ratios of each search were then averaged to produce the tabulated results.

Figure 3 shows that with a set of signatures derived from a real document collection, using sematic hashing, far su-

Breadth	Random (HDR)	Document (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: The effect of early stopping on performance – the HDR of searches on random signatures and signatures derived from text documents. Also see Figure 2.

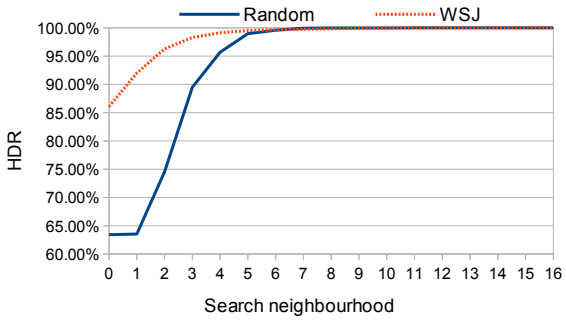

Figure 2: The effect of early stopping on performance. Based on data from Table 2.

Breadth	Query time (ms)	Lists per substring
0	4.17	1
1	4.8	17
2	8.36	137
3	23.2	697
4	66.91	2517
5	162.71	6885
6	322.48	14893
7	538.09	26333
8	766.45	39203
9	955.05	50643
10	1088.4	58651
11	1157.07	63019
12	1183.4	64839
13	1190.97	65399
14	1187.46	65519
15	1188.1	65535
16	1189.94	65536

Table 3: The relationship between early stopping and search time. Searching more posting lists requires spending proportionately more time. Also see Figure 3.

perior perfromace is acheivable with a smaller search neighbourhood. The greater HDR scores of document-derived signatures are due to the natural clustering of documents. Documents signatures cluster by topics much more tightly than random signatures do (which, it should be noted, also cluster, albeit to a much lesser extent - that's a well understood property of random uniform data points) . Topical clustering results in the closest signatures in the document collection being much closer on average than those in the random collection.

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 a document-derived collection of 2^{20} signatures (using TopSig [3] to generate signatures) 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 that need to be considered for that search breadth (Table 1 shows the relationship between search breadth and posting lists).

The time required to search this collection exhaustively is 195.07 milliseconds, making it a bit slower than using a 5-bit search breadth. Note that all performance benchmarks were performed on an i7 950 (3.07GHz) 4-core CPU (although to-date we only used single threaded processing in our implementation).

As described previously, the nature of the tradeoff between memory and performance being made by this approach can be tweaked by altering the width of the substrings. Our experiments have largely covered the specific case of 16-bit wide substrings; however, working with other sizes can provide significant advantages.

The relationship between substring widths, memory usage and performance leans towards wider substrings requiring more memory but providing greater performance, as Figure 4 shows; however, the relationships between these variables is subtle. Very short substring widths can be less memory-efficient as the signature ID of every signature must

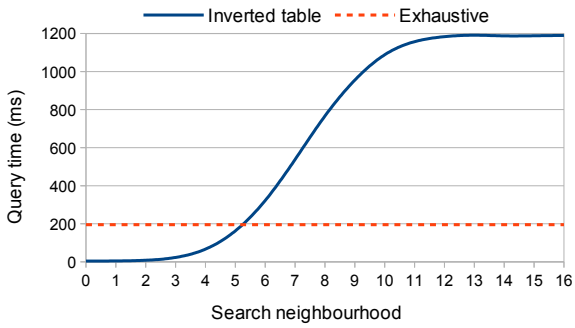

Figure 3: The relationship between early stopping and search time. Based on data from Table 3.

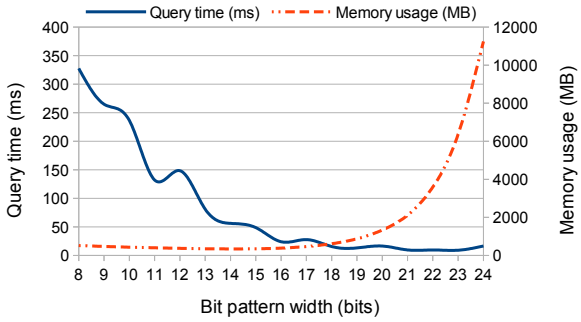

Figure 4: The relationship between substring width, memory usage and performance.

be stored once for each list and smaller substrings means more lists for each signature to go into. For example, with 64-bit signatures and 16-bit slice widths, each signature must be stored in 4 lists; with 8-bit slice widths, each signature must be stored in 8 lists. Thus, the best pattern width to use for maximising performance isn't necessarily the largest made available by the memory size. In this case, a 23-bit Hamming neighbourhood provides the most time-efficient search, at $\sim$ 9.12 milliseconds; however, at the cost of requiring $\sim$ 6365.88 megabytes of memory.

4.1 Scalability

Figure 5 shows how the query execution time increases with the size of the collection for both exhaustive and inverted table searches. A 200,000 signature collection takes 39.4 milliseconds to exhaustively search, and 3 milliseconds to search using the inverted signature table approach; an improvement in speed of $\sim$ 13.13×. A 2 million signature collection takes 382.4 milliseconds to exhaustively search, but only 15.2 milliseconds using the inverted signature table (a $\sim$ 25.16× improvement). As the curve suggests, search is relatively faster with a smaller collection because a larger portion of the signature lists referenced during neighbourhood expansion are empty and can therefore be immediately discarded without further processing. As the collection size increases more of the referenced lists are occupied and hence a higher processing cost is incurred. When the collection reaches the saturation point where all of the lists are used the time required to perform a search becomes linear. This can be resolved by increasing the substring width, which

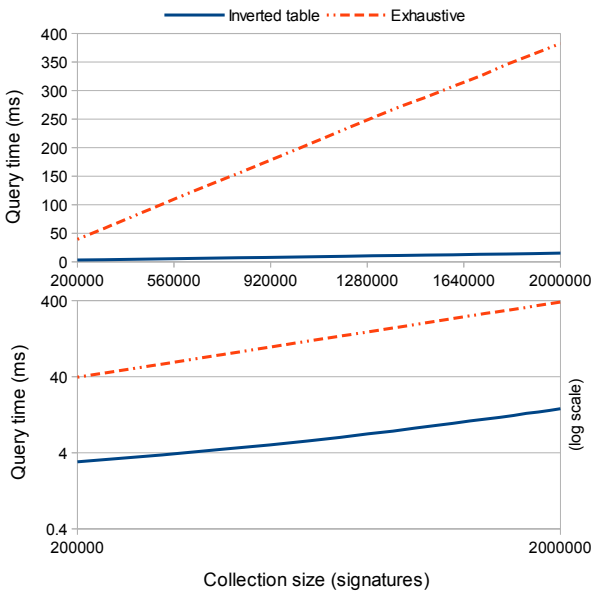

Figure 5: The relationship between collection size and query execution time for inverted table and exhaustive searches in linear (top) and log-log (bottom) scales. Using 23-bit substrings and a 3-bit search neighbourhood.

increases the number of lists and therefore the saturation point for the table.

It should be noted that we have used a state-of-the art Hamming Distance calculation algorithm. The Hamming distance of two words A and B can be calculated as the Hamming weight of A xor B. This is an elementary, well studied, and well understood problem and several simple solutions that preform similarly exist that have not been improved upon on standard hardware, including the use of the *popcnt* hardware instruction or resorting to assembly language. As a result we are absolutely confident that the baseline we use is reliable and our timing comparisons are sound.

5. LIMITATIONS

The requirement that the signature table remains in memory during processing (as the search process involves highly random access) limits the use of this approach for collections that expand to a signature table that is too large to fit in memory. This limitation is actually a fairly minimal one considering the low costs of RAM and the fact that signature representations of collections are typically much smaller than the collections themselves.

In practice, this is not a significant drawback as systems with sufficient memory to handle even very large collections, such as the 870-million document ClueWeb12[5], are available. In this case, the signature file would require (assuming 1024-bit signatures) 103.71 gigabytes of memory, while the inverted signature table would require (assuming 16-bit substrings) a further 207.44 gigabytes. (Increasing the substring width would not considerably increase this figure, as the overhead from longer substrings is independent of collection size.) While considerable, this is not beyond the reach of modern workstations.

[5] `http://lemurproject.org/clueweb12/`

6. CONCLUSION

We have presented an approach to improving the speed of signature searching without a considerable loss to search fidelity. While the effective use of inverted signature tables may be limited to certain applications (such as near-duplicate detection and some clustering approaches) in those situations they can provide great increases in performance over exhaustive approaches.

7. REFERENCES

[1] Andrei Z Broder. On the resemblance and containment of documents. In *Compression and Complexity of Sequences 1997. Proceedings*, pages 21–29. IEEE, 1997.

[2] Trevor Cohen, Roger Schvaneveldt, and Dominic Widdows. Reflective random indexing and indirect inference: A scalable method for discovery of implicit connections. *Journal of Biomedical Informatics*, 43(2):240–256, 2010.

[3] Shlomo Geva and Christopher M De Vries. Topsig: topology preserving document signatures. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, pages 333–338. ACM, 2011.

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

[5] Kalervo Järvelin and Jaana Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 41–48. ACM, 2000.

[6] 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.

[7] Caitlin Sadowski and Greg Levin. Simhash: Hash-based similarity detection. Technical report, Technical report, Google, 2007.

[8] Malcolm Slaney and Michael Casey. Locality-sensitive hashing for finding nearest neighbors [lecture notes]. *Signal Processing Magazine, IEEE*, 25(2):128–131, 2008.

[9] Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims. A support vector method for optimizing average precision. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 271–278. ACM, 2007.