Improved Hamming Distance Search using Variable Length Hashing

Eng-Jon Ong and Miroslaw Bober
Centre for Vision, Speech and Signal Processing
University of Surrey, Guildford, UK
e.ong,m.bober@surrey.ac.uk

Abstract

This paper addresses the problem of ultra-large-scale search in Hamming spaces. There has been considerable research on generating compact binary codes in vision, for example for visual search tasks. However the issue of efficient searching through huge sets of binary codes remains largely unsolved. To this end, we propose a novel, unsupervised approach to thresholded search in Hamming space, supporting long codes (e.g. 512-bits) with a wide-range of Hamming distance radii. Our method is capable of working efficiently with billions of codes delivering between one to three orders of magnitude acceleration, as compared to prior art. This is achieved by relaxing the equal-size constraint in the Multi-Index Hashing approach, leading to multiple hash-tables with variable length hash-keys. Based on the theoretical analysis of the retrieval probabilities of multiple hash-tables we propose a novel search algorithm for obtaining a suitable set of hash-key lengths. The resulting retrieval mechanism is shown empirically to improve the efficiency over the state-of-the-art, across a range of datasets, bit-depths and retrieval thresholds.

1. Introduction

At present, there is a need for efficient searching in image datasets that are increasingly larger in size, ranging from millions (e.g.Flickr 1M[5], ImageNet[1]) to a billion images (e.g. ANN1B [7]). Tackling this problem requires two important mechanisms: 1) efficient image representation and 2) the ability to quickly search inside the representation space.

This paper addresses the second part of efficient searching. For our purposes, we tackle the task of fast large-scale retrieval of compact binary vectors, posed as a thresholded Hamming distance search. To achieve this, we propose a novel unsupervised approach for performing thresholded search in Hamming space. Our approach is able to cope efficiently with binary code dimensionalities that are large (e.g. 512-bits) and a wide range of Hamming distance radii.

1.1. Background Review

There exists a large body of work on generating binary codes for the purpose of large scale image retrieval, in particular the method of hashing [11, 10, 14]. They aim to extract hashing functions that binarise high dimensional feature vectors into compact binary codes.

Nonetheless, the problem of efficiently searching through a large dataset of binary vectors remains. A linear scan is usually used at this stage, which can be accelerated by the build-in CPU hardware instructions. However, for large datasets (hundreds of millions, billions), the linear search time for a single query is still in the order of minutes. One solution is the hierarchical decomposition of the search space using multiple trees [8]. Hashing using binary codes has also been used for fast approximate nearest neighbour search in image retrieval [13], where, the binary code is the hashtable key. Retrieval of related examples to a query example are the colliding hashtable entries. However, this approach is only applicable when the dimensionality of the binary code is small (i.e. less than 32), otherwise, the memory footprint of the hashtable itself becomes prohibitively large. Our work differs from this in that we can still cope with large dimensionalities.

Another approach is to divide the binary code into smaller segments and build multiple hashtables, leading to the Multi-Index Hashing (MIH) approach [3] and its use for large scale search by Norouzi et al.[9]. Here, a binary code is divided into equally sized substrings and separate hashtables are built from them. The configuration of substring lengths and their number is selected such that a superset of relevant examples (i.e. within some r-neighbourhood in Hamming space) are returned. Examples that are above distance r are then removed using linear scan. The resulting search speeds were significantly faster than linear scan. However, this speedup is possible only for small Hamming thresholds r. When r increases, the time spent on removing inaccurate retrievals increases very quickly and eventually, becomes very similar to exhaustive linear scans. This is due to the constraint of equal length strings.

Our work removes this constraint and we show how this improves in the retrieval time compared to the MIH approach, across the a large range of Hamming thresholds. Simultaneously, our approach also provides the option for a faster approximate search.

1.2. Contributions and Overview

In this paper, we propose a novel approach which delivers significant efficiency improvements over the existing multi-index hashing methods, used for large scale retrieval of nearest neighbours binary vectors in Hamming metric spaces. This is achieved through the following contributions:

- Extension of the MIH method to support variable length hashkeys for the different hashtables.
- Theoretical analysis of using variable length hashkeys.
- Novel tree-based search method for locating near-optimal set of substring lengths.

To our knowledge, we are not aware of any other work that uses multiple hashtables with variable length hashkeys for efficient retrieval. We will show that this gives a retrieval mechanism that is significantly more efficient than the state-of-the-art multi-index hashing method with no loss of accuracy.

In the rest of the paper, Section 2 details the problem statement that we tackle in this paper. The hashtable-based retrieval mechanism is described in Section 3, it also derives detailed retrieval probabilities from a combinatorial perspective. Section 4 derives upper and lower bounds for the retrieval probabilities. This leads to an efficient tree-based method for finding suitable hashkey lengths, where the theoretical bounds provide crucial pruning criteria for efficient search. We then provide experimental results, evaluating the performance of the proposed method in comparison to state-of-the-art approaches in Section 5. Finally, we conclude in Section 6. To aid clarity, most of the proofs for lemmas and theorems are given in the appendix.

2. Problem Statement

In this section, we provide a formal statement to the problem of *thresholded* Hamming distance search. Let us be given a dataset $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$ of N number of D-dimensional binary vectors. The distance metric is assumed to be Hamming distance, denoted as $||_H$. Let $\mathbf{q}$ be the D-dimensional query vector. We are also given two parameters known as the Hamming threshold, θ and minimum required recall rate β.

The aim of the paper is then to find a retrieval function $R : \{0,1\}^D \rightarrow 2^N$ that quickly returns a set ($\mathbf{R}$) of database example indices given some example query $\mathbf{q}$, which we denote as: $\mathbf{R} = R(\mathbf{q})$. We require that the Hamming distance between $\mathbf{q}$ and all the examples indexed in $\mathbf{R}$ be less than or equal to θ. Now, suppose that the true number of examples in $\mathbf{X}$ with Hamming distance less or equal to $\mathbf{q}$ is N_+. We also allow for an approximate retrieval, where: $1 \geq |\mathbf{R}|/N_+ \geq \beta$.

3. Multiple Hashtables Retrieval

This section describes a multiple hashtable approach that allows us to efficiently retrieve examples that are within a predefined Hamming distance, θ, to a given query example. We find that the efficiency of retrieval can be further increased if we allow a small factor of false negative retrieval error to occur, denoted by the factor, $\epsilon \in [0,1]$. That is, suppose the number of examples within θ Hamming distance to the query is N_θ, then we allow ϵN_θ examples to be rejected. The factor ϵ is directly related to β as: $\epsilon = 1 - \beta$.

3.1. HL-Sets

In order to perform efficient retrieval from a large number of examples, multiple hashtables are used. The keys for the hashtables are obtained by dividing the D-dimensional binary feature vector into a number of M mutually exclusive substrings, each acting as a hashkey. The set of M hashkey lengths is denoted as $\mathbf{M} = (m_i)_{i=1}^M$. Note that in this work, $\mathbf{M}$ is a *multiset*, allowing us to have multiple elements with the same value. For the rest of the paper, we will denote a set of Hashkey Lengths $\mathbf{M}$ as a *HL-set*.

Previous work [3] required that all the hashkey lengths be equal, with the lengths summing to the feature vector dimension, D. Here, we allow the hashkey lenghts m_i to be different and do not require their sum to equal D. We show experimental evidence (Section 5) how both of the above improve the retrieval efficiency and accuracy, compared to existing work.

3.2. Retrieval Mechanism using Hashtable Sets

The retrieval mechanism consists of M separate hashtables, which we denote as a *hashtable set*. Each hashtable takes a substring as the hashkey and returns a set of example indices with a similar key (i.e. colliding examples). The full set of retrieved examples is the union of the retrieved examples across all the M hashtables.

In this work, we aim to configure the keys such that there is high probability of collisions in the hashtable for examples that have Hamming distances less than θ, whilst minimising the collisions for examples with distances greater than θ. Formally, we can think of each hashtable as a function, $H_i : \{0,1\}^{m_i} \rightarrow 2^N$. Associated with the i^{th} hashtable H_i, is a set of key vector dimensions $\mathbf{D}_i = \{d_{i,j}\}_{j=1}^{m_i}$. The "dimension index set" $\mathbf{D}_i$ can then be used to extract the hashtable key from the query binary vector.

Then, suppose we are given an input query $\mathbf{q} \in \{0,1\}^D$. We first extract the substring keys (of length m_i) for each

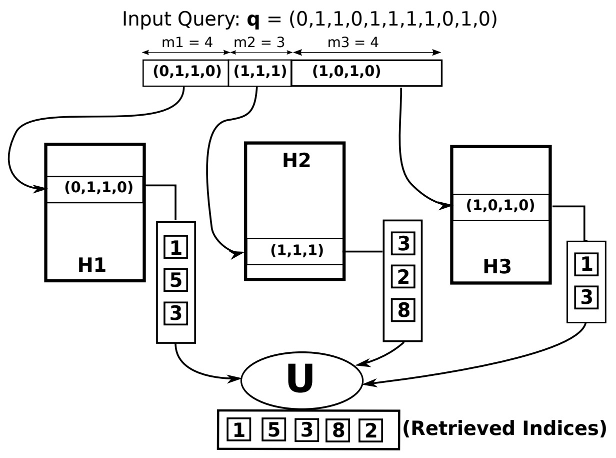


Figure 1. An illustration of the hashtable based retrieval mechanism given in Eq. 1. Here, the feature vector is split into 3 hashkeys of lengths $m1, m2, m3$ respectively. Each substring is a hashkey to the corresponding hashtables $H1, H2, H3$ respectively. Given a query vector $\mathbf{q}$, its substrings are used to retrieve relevant example indices before a final union to give the final retrieved indices.

of the hashtables using their respective dimension index set $\mathbf{D}_i$ as follows: $\mathbf{q}^i = (q_{d_{i,j}})_{j=1}^{m_i}$. The final set of retrieved examples, $\mathbf{R}$, is then given as:

$$\mathbf{R} = \bigcup_{i=1}^{M} H_i(\mathbf{q}^i) \tag{1}$$

An illustration of the multiple hashtable based retrieval in Eq. 1 can be seen in Fig. 1. Eq. 1 returns a superset of examples that have distances less than θ, thus, may return some with distances that are greater than θ, an equivalent of "false positives". Subsequently, the retrieved false positives are filtered by explicitly computing and thresholding based on their Hamming distances to the query example.

Thus, this paper proposes the retrieval function R described in the problem statement (Section 2) to be as follows:

$$R(\mathbf{q}) = \{i \in \mathbf{R} : |\mathbf{q} - \mathbf{x}_i| \leq \theta\}$$

where $\mathbf{R}$ is defined in Eq. 1, and $\mathbf{x}_i$ is the i^{th} example in the dataset described in Section 2.

3.3. Probability of Collisions: Combinatorial Perspective

In this section, we will consider the probability of retrieving examples within θ Hamming distance to a query using Eq. 1. More specifically, we seek to determine the probability of retrieval of an example with Hamming distance r from the query example, using M hashtables. The lengths of the hashkeys of these hashtables are given in the set $\mathbf{M}$ and corresponding dimension indices $\mathbf{Q}$.

Considered from a combinatorial perspective, when the Hamming distance between an database example and query is r-bits, the number of valid "different bit" configurations is the binomial coefficient: ${}^{D}C_r$. Now, suppose we have a collision with the i^{th} hashtable, this implies that at least m_i bits between the query and the dataset example are *the same*. Thus, a collision in hashtable H_i implies that there is only ${}^{D-m_i}C_r$ valid configurations left. Hence, the probability of a query having collisions with entries in hashtable H_i is:

$$\begin{aligned} P_{m_i}(r) &= \frac{{}^{D-m_i}C_r}{{}^{D}C_r} \\ &= \frac{(D-m_i)^{\underline{r}}}{D^{\underline{r}}} \end{aligned} \tag{2}$$

where, for conciseness, we write the falling power as: $A^{\underline{r}} = A(A-1)(A-2)...(A-r+1)$.

Eq. 2 can be given a more convenient form by moving the variable r from the reducing power into multiplying factors (derivation details in Appendix C):

$$P_{m_i}(r) = \frac{(D-r)^{\underline{m_i}}}{D^{\underline{m_i}}} \tag{3}$$

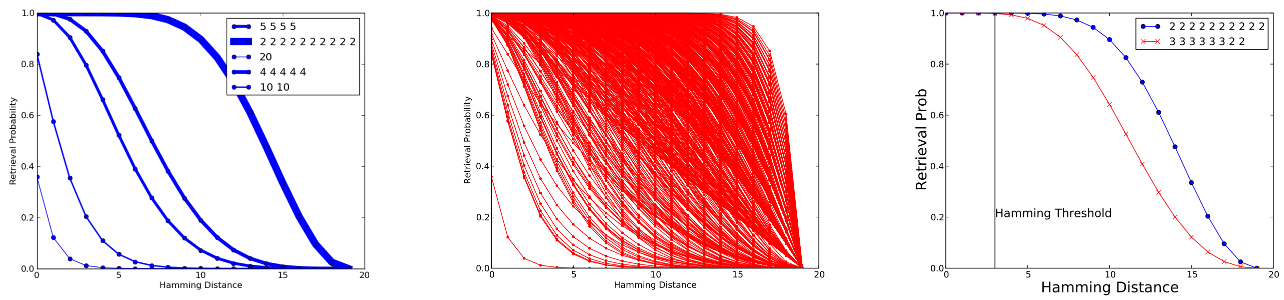
In order to obtain the probability of a query containing a substring that collided with at least one entry in one hashtable, we use the assumption that all the substrings associated with the hashtables are independent and therefore the product rule can be used, resulting in the following probability retrieval curve P for different Hamming distances r:

$$P(r) = 1 - \prod_{i=1}^{M}(1 - P_{m_i}(r)) \tag{4}$$

3.4. Fixed Length vs Variable Length Hashkeys

Given a configuration of hashkey lengths, $\mathbf{M}$, $P(r)$ in Eq. 4 allows us to theoretically predict the probability of retrieval of an example with respect to their distances from a query example. This in turn allows one to predict the percentage of retrieved examples within distance θ from a query example. Importantly, $P(r)$ (Eq. 4) predicts the proportion of examples that requires filtering using explicit Hamming distance computation. Consequently, we seek to minimise the probability of retrieval for Hamming distances above θ whilst maximimising the retrieval probabilities for Hamming distances below θ.

When the hashkey lengths are restricted to be equal and sum to D, the possible curves $P(r)$ are limited. As an example, when $D = 20$ bits, only 5 such curves are possible, as shown in Fig. 2a. As a result, we find that there are only a small set of threshold values that will not result in the retrieval of a large number of irrelevant examples (i.e. distance greather than θ) or the rejection of a large number of relevant examples.



(a) (b) (c)

Figure 2. Retrieval probability curves with fixed length hashkeys (a) and with variable hashkey lengths (b). (c) shows the advantage of fixed vs variable length hashkeys, where we require retrieval of examples less than distance 3. It can be seen that the variable length hashkeys (red curve) has lower probability of retrieving examples greater than distance 3 compared to fixed length hashkeys (blue curve).

In contrast, when variable hashkey lengths are used, the number of possible $P(r)$ curves are increased greatly, allowing us more efficiently handle a much larger range of required threshold values. This can be seen in Fig. 2b, showing all possible retrieval probability curves generated by different HL-sets (i.e. hashkey length configurations). The increase in the number of retrieval probability curves is crucial in allowing us to effectively deal with different Hamming distance thresholds and minimum recall rates. This is illustrated in Fig. 2c, showing a retrieval probability curve that has lower probability of retrieving examples of distance greater than a threshold (5 in the figure) when variable length hashkeys are used. In particular, the retrieval probability when variable length hashkeys (red curve) is consistently lower than when fixed lenght hashkeys (blue curve) are used for Hamming distances over threshold.

3.5. Approximation of Retrieval Probability

In the previous section, we have shown how the probability of a query feature vector colliding with an entry in at least one hashtable can be computed using Eq. 4. In practice, this equation has an inconvinient form involving falling factorials. Instead, we use an approximation to Eq. 4:

$$P_{\mathbf{M}}(r) = 1 - \prod_{i=1}^{M} \left(1 - \left(1 - \frac{r}{D}\right)^{m_i}\right) \qquad (5)$$

Empirical analysis on the probabilities calculated from Eq.5 show minimal deviation from Eq. 4. An example of this ($D = 128$) can be seen in Fig. 3a, with the difference between the 2 curves across all valid Hamming distances is shown in Fig.3b). The histogram of maximum differences between the approximate and exact curves is shown in Fig.3c). We can see there that the majority of differences is less than 0.05. When all the possible retrieval curves given $D = 128$ are considered, the mean maximum difference is 0.01. In fact, it is possible to show that both curves converge as D increases. For the remainder of the paper, we shall denote $P_{\mathbf{M}}(r)$ as the *retrieval probability curve*.

4. HL-Set Tree-based Searching

The use of variable length hashkeys allow us to generate a large number of possible probability retrieval curves. However, the total number of possible HL-sets for dimension D is the partition number of the integer D. We find that the partition number ($p(n)$) for an integer n increases exponentially and can be approximated as:

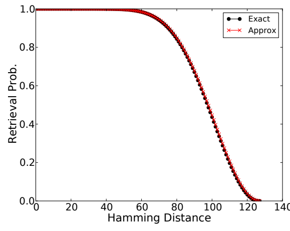
$$p(n) \approx \frac{1}{4n\sqrt{3}} \exp\left(\pi\sqrt{\frac{2n}{3}}\right)$$

As an example, when $n = 1024$, the partition number is on the order of 10^{31} possible HL-sets. Thus, exhaustively searching for the optimial hashkey length set would be impossible, except for very small bit-depths. Unfortunately, many configurations do not lead to efficient retrieval.

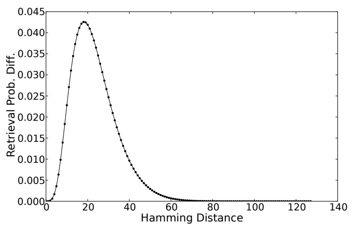
To address this issue, in this section, we propose a novel efficient tree-based search algorithm for finding efficient HL-sets given a Hamming distance threshold $\theta \in \{1, ..., D\}$ whilst resulting in retrievals that meet the minimum recall rate $\beta \in [0, 1]$. In order to make the tree-search efficient, pruning based on the lower bounds of retrieval probabilities will be employed.

4.1. Retrieval Probability Lower Bounds

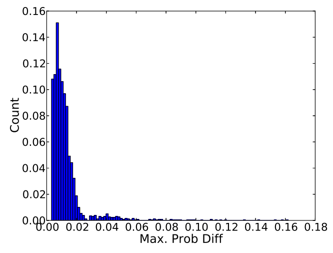
One important property is that by adding a new hashkey to an existing hashkey set, the original probability retrieval curve will be "lifted", in that the new retrieval probability values will be raised for all Hamming distances. This property establishes a lower-bound of the retrieval performance of a hashkey set and all its supersets. This in turn is crucial for the pruning criteria HL-set search algorithm proposed in the next section.



(a)



(b)



(c)

Figure 3. An example of the retrieval probability curves for a hashkey length set ($D = 128$) using the approximation and exact calculation (a). The difference between the computed exact and approximation retrieval probability curves (b). The histogram of *maximum* difference between exact and approximate values over all valid retrieval curves when $D = 128$-bits.

Lemma 1. *Let $(m_1, m_2, ..., m_i, m_{i+1})$ be a HL-set, $A_j = 1-(1-r/D)^{m_j}$ and $P_i(r) = 1-\prod_{j=1}^{i} A_j$. Then, $P_i(r) \leq P_{i+1}(r)$ for all $0 \leq r \leq D$. (Proof in Appendix A)*

We find that adding a shorter hashkey to an existing hashkey set will raise the retrieval probability *more* than adding a longer hashkey over all Hamming distances:

Lemma 2. *Let $\mathbf{M} = (m_1, m_2, ..., m_n)$ be a HL-set. Let $m'_1 < m'_2$ and $\mathbf{M}_1 = \mathbf{M} \cup \{m'_1\}$ and $\mathbf{M}_2 = \mathbf{M} \cup \{m'_2\}$. Let $P_{\mathbf{M}}(r) = 1 - \prod_{m \in \mathbf{M}}(1-(1-r/D)^m)$. Then, $P_{\mathbf{M}_2}(r) < P_{\mathbf{M}_1}(r)$ for all $0 \leq r \leq D$. (Proof in Appendix B.)*

4.2. Search Tree Pruning Criteria

In order to increase the efficiency of the search process, a pruning criteria based on the lower bounds of extending HL-sets is proposed. Firstly, we define the *cost* of the HL-set, $\mathbf{M} = (m_1, m_2, ..., m_n)$ using the following function S:

$$S(\mathbf{M}, \theta) = \sum_{r=\theta+1}^{D} P_{\mathbf{M}}(r) \tag{6}$$

The value of the function S represents the sum of example proportions after the threshold distance θ. A larger value of S would require us to filter more examples by means of explicit computation of the Hamming distance. Consequently, the aim is to minimise S whilst ensuring that the minimum recall requirement β is still met.

With respect to an existing "minimal" cost S_{min}, we find that if adding a hashkey of length m' to an existing set $\mathbf{M}$ results in a higher cost than S_{min}, then adding *any shorter* hashkey will also result in a higher cost than S_{min}. Additionally, any superset of $\mathbf{M} \cup \{m'\}$ will have a higher cost than S_{min}. More formally:

Theorem 1. *Let $\mathbf{M} = (m_1, m_2, ..., m_n)$ be a set of hashkey lengths, with $P_{\mathbf{M}(r)}$ its retrieval probability curve. Let θ be a given Hamming distance threshold and S_{min} be a minimum hashkey set cost, and $m' < D$ be some integer. Then, if $S(\mathbf{M} \cup \{m'\}, \theta) \geq S_{min}$, then for all $1 \leq l \leq m'$, $S(\mathbf{M} \cup \{l\}, \theta) \geq S_{min}$. Also, for any $\mathbf{M}' \supset \mathbf{M} \cup \{l\}$, we have $S(\mathbf{M} \cup \{l\}, \theta) \geq S_{min}$.*

Proof. Since $S(\mathbf{M} \cup \{m'\}, \theta) \geq S_{min}$, we have $\sum_{r=\theta}^{D} P_{\mathbf{M} \cup \{m'\}}(r) \geq S_{min}$ (Eq. 6). From Lemma 2, $P_{\mathbf{M} \cup \{l\}}(r) \geq P_{\mathbf{M} \cup \{m'\}}(r)$ for all $0 \leq l \leq m'$. Thus, it follows that $S(\mathbf{M} \cup \{l\}, \theta) = \sum_{r=\theta}^{D} P_{\mathbf{M} \cup \{l\}}(r) \geq \sum_{r=\theta}^{D} P_{\mathbf{M} \cup \{m'\}}(r) \geq S_{min}$. Hence, we have shown that for all $1 \leq l \leq m'$, $S(\mathbf{M} \cup \{l\}, \theta) \geq S_{min}$.

To show that for any $\mathbf{M}' \supset \mathbf{M} \cup \{l\}$, then $S(\mathbf{M} \cup \{l\}, \theta) \geq S_{min}$, we follow similar lines. Since $\mathbf{M}' \supset \mathbf{M} \cup \{l\}$, then from Lemma 1, $P_{\mathbf{M}'}(r) \geq P_{\mathbf{M} \cup \{l\}}(r)$, and so, $S(\mathbf{M}', \theta) = \sum_{r=\theta}^{D} P_{\mathbf{M}'}(r) \geq \sum_{r=\theta}^{D} P_{\mathbf{M} \cup \{l\}}(r) \geq S_{min}$. Hence, we have shown that for any $\mathbf{M}' \supset \mathbf{M} \cup \{l\}$, we have $S(\mathbf{M} \cup \{l\}, \theta) \geq S_{min}$. $\square$

This means, we can stop considering any branches, breadth or depth-wise when adding a new hashkey results in a cost greater than the current optimal cost S_{min}.

4.3. Search Algorithm

The proposed algorithm for obtaining a HL-set is given in Algorithm 1 (procedure SUBLENSEARCH). This algorithm aims to efficiently find a HL-set, $\mathbf{M} = (m_1, m_2, ..., m_n)$ with a minimal cost, $S(\mathbf{M}, \theta)$ and have $P_{\mathbf{M}}(\theta) \geq \beta$. The search problem is tackled as an integer partitioning task, where the valid range of integers are considered in a tree-manner, implemented recursively. To this end, it considers every multiset of integers that will sum up to D. The requirement that the integers must sum to at most D is checked in line 12. The breadth search is done as a for loop in line 10, whilst depth searching performed recursively in line 24. To allow the algorithm to consider all HL-sets in a reasonable time, Theorem 1 provides a simultaneous breadth and depth pruning of irrelevant branches in Line 17 by breaking out of the for loop (line 10) and stopping the recursive call (line 24) from being called.

5. Experiments

This section describes the experiments conducted to evaluate the performance of the proposed method against

Algorithm 1 Hashkey Length Set Search

```
1:  $S_{min} = D$
2:  $\mathbf{M}_{opt} = \emptyset$
3:  procedure SUBLENSEARCH($D, \theta, \beta, \mathbf{M}$)
4:      if $\mathbf{M} = \emptyset$ then
5:          $m_{max} = D$
6:      else
7:          $m_{max} = \min(\mathbf{M})$
8:      end if
9:      $S_{\mathbf{M}} = \sum_{m \in \mathbf{M}} m$
10:     for $i \in m_{max}, m_{max} - 1, ..., 1$ do
11:         $\mathbf{M}' = \mathbf{M} \cup \{i\}$
12:         if $S_{\mathbf{M}} + i > D$ then
13:             continue;
14:         end if
15:         $\beta_{cur}, A_{cur} = FindTPandArea(D, \mathbf{M}', \theta)$
16:         if $A_{cur} \geq S_{min}$ then
17:             break; (Theorem 1)
18:         end if
19:         if $\beta_{cur} \geq \beta$ then
20:             $S_{min} = A_{cur}$
21:             $\mathbf{M}_{opt} = \mathbf{M}'$
22:         else
23:             if $A_{cur} < S_{min}$ then
24:                 SubLenSearch( $D, \theta, \beta, \mathbf{M}'$)
25:             end if
26:         end if
27:     end for
28: end procedure
29: function FINDTPANDAREA($D, \mathbf{M}, r$)
30:     $S_r = \sum_{i=r}^{D} P_{\mathbf{M}}(r)$
31:     return $P_{\mathbf{M}}(r), S_r$.
32: end function
```

related state-of-the-art methods. Specifically, we have performed experiments on thresholded Hamming distance retrieval using a number of large scale datasets. In particular, we tackle the problem of a retrieval of a percentage (β) of examples below a given Hamming distance (θ) to an example query.

5.1. Datasets and experimental setup

The datasets considered in this section are as follows: 1 Billion SIFT 128D dataset [7], 1 Million SIFT dataset using 128-bits [6] and Flickr 1 Million [4] dataset using 512 bits. For the ANN datasets, the provided 128D SIFT features were binarised using Gaussian random projection followed by binarisation as detailed in [2] and as is consistent with the method used in [9]. The Flickr 1 Million dataset used a 512D binary feature vector extracted using the Robust Visual Descriptor method[12].

For all the experiments, we will compare the performance of the proposed variable length hashkey method against the multi-index hashing (MIH) method using fixed length hashkeys and the linear scan method proposed in [3]. For each dataset, 10000 examples were reserved for the test set. The remaining were used to build the search database.

In all experiments, the Hamming distance thresholds (Θ) considered lie in the range of $[1, D/3]$ at increments of 16. We consider a set of required minimum recall rates: $\mathbf{B} = \{0.999, 0.9, 0.8, 0.7\}$. Then, for each given threshold $\theta \in \Theta$, and required minimum recall rate $\beta \in \mathbf{B}$, the appropriate HL-set is obtained using the proposed search method (Section 4.3).

Next, the required hashtables, as dictated by the HL-set are built. The feature dimensions for the hashkeys in each hashtable were randomly chosen, but ensured to only be used by a single hashtable. To compare against the MIH method, fixed hashkey lengths based on the length $\log_2(N)$, where N is the dataset size was used, as described in [3].

Following this, each example in the test set is used as a query $\mathbf{q} \in \{0,1\}^D$. Using the retrieval mechanism from Section 3.2, with $\mathbf{q}$ as input, a set $\mathbf{R}$, of $|\mathbf{R}|$ number of retrieved examples is obtained. The Hamming distance of members in $\mathbf{R}$ to $\mathbf{q}$ is then computed, allowing the extraction of the set of retrieved examples with Hamming distance equal or less than θ, $\mathbf{R}^+$:

$$\mathbf{R}^+ \quad = \quad \{\mathbf{r} \in \mathbf{R} : |\mathbf{r} - \mathbf{q}|_{\mathcal{H}} \leq \theta\}$$

To compute the recall rate to query $\mathbf{q}$, the subset of examples $\mathbf{N}^+$ with distance less or equal to θ from $\mathbf{q}$ in dataset $\mathbf{D}$ is obtained using linear scan. The recall rate is then: $MRR = |\mathbf{R}^+|/|\mathbf{N}^+|$, and is used to verify whether the minimum recall rate β is met. The retrieval size $|\mathbf{R}|$ is used to evaluate the improvement of the proposed method over linear scan and MIH.

5.2. Results and Analysis

For all the experiments, in terms of search time for the hashkey lengths, the proposed method took less than 10 seconds on a single threaded Intel Processor (2.2GHz) for most cases. However, for a small minority of cases, the search took up to 6 minutes for 512D vectors. However, the current implementation is in Python, and this hashkey search time can be significantly improved by implementation in C++. The largest memory footprint was 200GB for the 128D 1Billion ANN dataset.

The results of the experiments can be seen in Fig. 4. Firstly, as can be seen in Fig. 4a),b),c), MIH will only satisfy a limited range of minimum recal rates. For example, for the 1Billion ANN dataset, with $\beta = 0.999$, the MIH method recovers at the required minimum recall rate uptill $\theta = 12$. From then onwards, the required dataset is too small and rejects too many examples below the required threshold. In contrast, the proposed method always finds a

β	0.999	0.9	0.8	0.7
ANN 1B	12	12	16	16
1 Million SIFT 128	6	12	12	16
1 Million Flickr 512	97	113	129	129

Table 1. This table shows the maximum Hamming thresholds where the MIH method met the required minimum recall rate, shown in the first row.

Min. Rec. Rate	0.999	0.9	0.8	0.7
1 Billion ANN	30	875	784	785
1 Million 128	13	818	1359	1064
1 Million 512	18	86	98	103

Table 2. Average speedup factor over MIH using the proposed method for different minimum recall rate values.

suitable set of hashkey lengths that result in the β satisfied for any Hamming distance threshold θ. Table 1 shows the other values for other minimum recal rates and datasets.

Next, we find that the hashtable retrieval is consistently more efficient than linear scan over all values of θ and β. This can be seen in Fig. 4d,e,f) showing the speedup factors of both the proposed method and MIH over linear scan. Since the MIH method uses only a constant hashkey length ($\log_2(N)$) for building the hashtables, it has a constant retrieval size, regardless of the required threshold θ or minimum recall rate, β. It is therefore shown as a constant horizontal line in Fig. 4a,b,c. Here, it can be seen that the proposed method retrieves sets that can be significantly smaller, ranging from 10^{-7} the size of the dataset for 1 Billion examples to 10^{-6} for 1 million examples. When the minimum recall rate is met, as detailed in Table 2, we find that the proposed method on average returns retrieval sets that are smaller than those of MIH. However, we have found that this is not always the case. For the Flickr 1 Million dataset, we find that the MIH method returns retrieval sets that are smaller for Hamming thresholds between 100 and 150. This limitation is due to the assumption in the search method that examples are equally distributed across all Hamming distances. In the future we will aim to incorporate information on the Hamming distance distribution into the search method.

6. Conclusions and Future Work

In this paper, we have proposed a novel, unsupervised approach to thresholded search in Hamming space, supporting long codes (e.g. 512-bits) with a wide-range of Hamming distance radii. Based on the theoretical analysis of the retrieval probabilities of multiple hash-tables we have proposed a novel tree-based search algorithm for obtaining a suitable set of hash-key lengths that guarantees a minimum required recall rate for retreival of examples below a given Hamming distance threshold. We have shown empirically that our method is capable of handling bit depths up to 512 bits and working efficiently up to a billion codes delivering resulting one to three orders of magnitude acceleration, as compared to the MIH method.

For future work, we aim to extend the variable length hashkey method for weighted Hamming distances. Additionally, more theoretical analysis is required for the accuracy bounds of the approximation to the retrieval probability used here. Finally, we have also experimentally observed that during the search process, hashkeys that are long in length or too short are always pruned in the search tree. It would be beneficial to obtain a better bounds for the range of hashkey lengths that will eventually be useful.

Acknowledgements

This work was supported by the European Commission under Grant 610691 (project BRIDGET) and TSB-funded CODAM project.

Appendix

A. Proof of Lemma 1

Proof. We have $P_i(r) = 1 - A_1A_2...A_i$ and $P_{i+1}(r) = 1 - A_1A_2...A_iA_{i+1}$. Then,

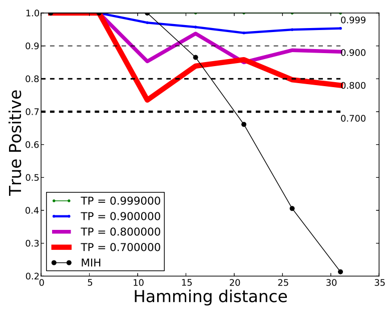
$$\begin{aligned} P_i(r) - P_{i+1}(r) &= 1 - A_1A_2...A_i - (1 - A_1...A_iA_{i+1}) \\ &= (A_{i+1} - 1)\prod_{j=1}^{i} A_j \\ &\leq 0 \text{ (since } A_{i+1} \leq 1) \end{aligned}$$

Hence, since $P_i(r) - P_{i+1}(r) \leq 0$, we have shown that $P_i(r) \leq P_{i+1}(r)$. □

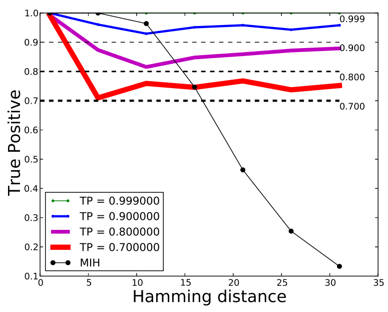
B. Proof of Lemma 2

Proof. To prove that $P_{\mathbf{M}_2}(r) < P_{\mathbf{M}_1}(r)$, we consider the difference between $P_{\mathbf{M}_1}(r)$ and $P_{\mathbf{M}_2}(r)$:

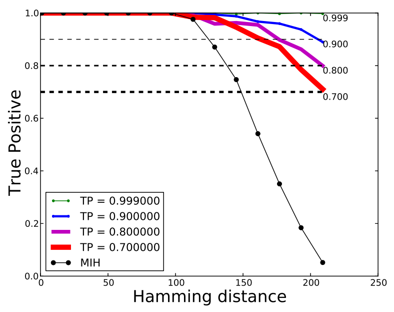
$$\begin{aligned} P_{\mathbf{M}_1}(r) - P_{\mathbf{M}_2}(r) &= 1 - \prod_{m\in\mathbf{M}_1}\left(1 - \left(1 - \frac{r}{D}\right)^m\right) \\ &\quad -1 + \prod_{m\in\mathbf{M}_2}\left(1 - \left(1 - \frac{r}{D}\right)^m\right) \\ &= -\prod_{m\in\mathbf{M}_1}\left(1 - \left(1 - \frac{r}{D}\right)^m\right) + \\ &\quad \prod_{m\in\mathbf{M}_2}\left(1 - \left(1 - \frac{r}{D}\right)^m\right) \\ &= \left[\prod_{m\in\mathbf{M}}\left(1 - \left(1 - \frac{r}{D}\right)^m\right)\right] \times \\ &\quad \left[\left(1 - \frac{r}{D}\right)^{m_1'} - \left(1 - \frac{r}{D}\right)^{m_2'}\right] \\ &> 0 \end{aligned}$$



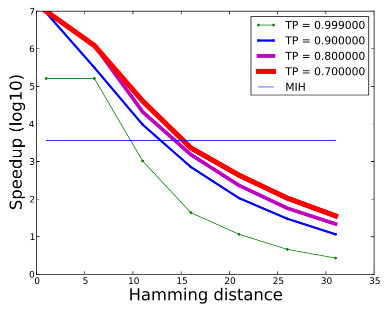
1 Billion ANN
(a)



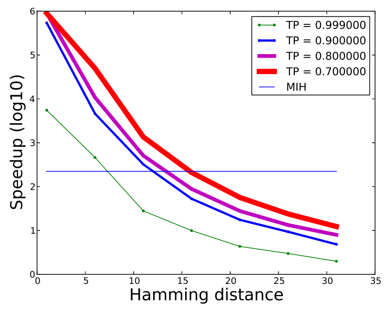
1 Million 128D SIFT
(b)



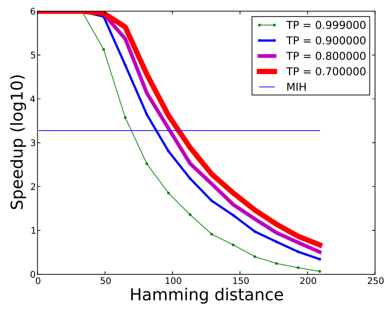
Flickr 1 Million (512-bit)
(c)



1 Billion ANN
(d)



1 Million 128D SIFT
(e)



Flickr 1 Million (512-bit)
(f)

Figure 4. Results for the ANN 1B SIFT dataset with 128-bit vectors, ANN 1M SIFT dataset and Flickr 1M Datasets. (a,b,c) show the achieved minimum recall rates (denoted as "True Positive Rate") for the variable length method and for the fixed length hashkey method. (d,e,f) show the speedup over linear scan for different minimum recall values across different Hamming thresholds.

Since $P_{\mathbf{M}_1}(r) - P_{\mathbf{M}_2}(r) > 0$, we have shown that $P_{\mathbf{M}_1}(r) > P_{\mathbf{M}_2}(r) > 0$ when $m'_1 < m'_2$. □

C. Proof of Eq. 3

To get Eq. 3 from Eq. 2, we first expand the RHS of Eq. 2 into its factorial factors (index i dropped for convinience):

$$\frac{(D-m)^{\underline{r}}}{D^{\underline{r}}} = \frac{(D-m)!}{(D-m-r)!} \times \frac{(D-r)!}{D!} \tag{7}$$

Next, we note that the first factor on the right hand side of Eq. 7 can be simplified as follows:

$$\begin{aligned} \frac{(D-r)!}{(D-m-r)!} &= \frac{(D-r)...(D-r-m+1)(D-m-r)!}{(D-m-r)!} \\ &= (D-r)(D-r-1)...(D-r-m+1) \\ &= (D-r)^{\underline{m}} \end{aligned}$$

The RHS second factor of Eq. 7 is similarly simplified:

$$\begin{aligned} \frac{(D-m)!}{D!} &= \frac{(D-m)!}{D(D-1)...(D-m+1)(D-m)!} \\ &= \frac{1}{D^{\underline{m}}} \end{aligned}$$

Substituting both the above formulas into Eq. 7 gives:

$$\frac{(D-m_i)^{\underline{r}}}{D^{\underline{r}}} = \frac{(D-r)^{\underline{m}}}{D^{\underline{m}}} \tag{8}$$

Hence, Eq. 2 can be rewritten as:

$$P_{m_i}(r) = \frac{(D-r)^{\underline{m_i}}}{D^{\underline{m_i}}}$$

References

[1] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *In Proc. of CVPR*, 2009.

[2] M. Diephuis, S. Voloshynovskiy, O. Koval, and F. Beekhof. Statistical analysis of binarized sift descriptors. In *In Proc. ISPA*, pages 460–465, 2011.

[3] D. Greene, M. Parnas, and F. Yao. Multi-index hsahing for information retrieval. In *In Procs of IEEE FOCS*, 1994.

[4] M. Huiskes, B. Thomee, and M. Lew. New trends and ideas in visual concept detection. In *Proc. of ACM MIR*, 2010.

[5] H. Jegou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large-scale image search. In *In Proc. of ECCV*, 2008.

[6] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Trans. PAMI*, 2011.

[7] H. Jegou, R. Tavenard, M. Douze, and L. Amsaleg. Searching in one billion vectors: re-rank with source coding. In *In Proc. of ICASSP*, 2011.

[8] M. Muja and D. Lowe. Fast matching of binary features. In *In Proc. of CRV*, pages 404–410, 2012.

[9] M. Norouzi, A. Punjanio, and D. Fleet. Fast exact serach in hamming space with multi-index hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(6):1107–1119, 2014.

[10] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *In Proc. of NIPS*, 2009.

[11] R. Salakhutdinov and G. Hinton. Semantic hashing. *Int. Journal of Approximate Reasoning*, 50(7), 2009.

[12] H. Syed and M. Bober. Robust and scalable aggregation of local features for ultra large scale retrieval. In *In Proc. of ICIP*, 2014.

[13] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. In *Proc. of IEEE CVPR*, 2008.

[14] J. Wang, S. Kumar, and S. Chang. Semi-supervised hashing for large-scale search. *IEEE PAMI*, 34(12), 2012.