Modifying Hamming Spaces for Efficient Search

Vladimir Mic
Faculty of Informatics
Masaryk university
Brno, Czech republic
xmic@fi.muni.cz

David Novak
Faculty of Informatics
Masaryk university
Brno, Czech republic
david.novak@fi.muni.cz

Pavel Zezula
Faculty of Informatics
Masaryk university
Brno, Czech republic
zezula@fi.muni.cz

***Abstract*—We focus on the efficient search for the most similar bit strings to a given query in the Hamming space. The Hamming distance can be lower-bounded by the difference of the “number of ones” in the compared strings, i.e. of their *weights*. Recently, such property has been successfully used by the *Hamming Weight Tree* (HWT) indexing structure. We propose modifications of the bit strings that preserve pairwise Hamming distances but improve the tightness of these lower bounds, so the query evaluation with the HWT is several times faster. We also show that the unbalanced bit strings, recently reported to provide similar quality of search as the traditionally used balanced bit strings, can be more efficiently indexed with the HWT. Combined with the distance preserving modifications, the HWT query evaluation can be more than one order of magnitude faster than the HWT baseline.**

***Keywords*-Similarity search, Hamming space, Hamming Weight Tree, Lower bound in the Hamming space**

I. Introduction

The goal of the similarity search is to find objects that are close to a given query object considering a specific similarity function. Many applications require efficient similarity search and volumes of current data make this task difficult to solve. Recently proposed approaches include those that substitute original data objects with bit strings [1], [2], [3], [4], [5], [6], [7]. The (dis)similarity of a pair of these bit strings is usually expressed by the Hamming distance h [8], which evaluates the number of different bits.

We consider a query bit string q and dataset X. The goal of the k *nearest neighbour* query (kNN) is to find the k bit strings in X that are the closest to q. Despite the efficiency of Hamming distance evaluation, sequential evaluation of all distances $h(q, o), o \in X$ on big datasets can take even minutes [9], which is not acceptable for many applications. Therefore, there is a need for well-scaling indexes.

Recently, authors of the *Hamming weight tree* (*HWT*) proposed to lower bound the Hamming distance using the number of 1s in the bit strings and to use this lower bound to prune the Hamming search space during evaluation of similarity queries [10]. The efficiency of query evaluations depends on the tightness of the provided lower bounds. The main contribution of this paper is a proposal of bit string modifications that preserve pairwise Hamming distances and tighten these lower bounds that are crucial to HWT efficiency.

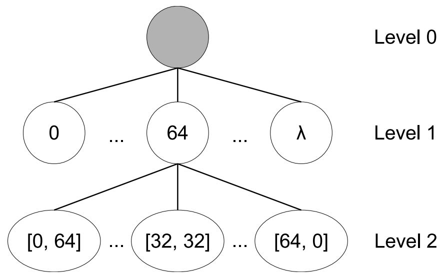


Figure 1: Structure of the Hamming Weight Tree (HWT)

The HWT suffers from the phenomenon called the *curse of dimensionality* [11], [12], [13]: the efficiency of query evaluation degrades rapidly with the increasing distance $h(q, o_k)$ between query bit string q and its kth nearest neighbour o_k. We show that the HWT is up to an order of magnitude slower than the sequential evaluation of the Hamming distances using our real-life data to illustrate this feature. Beside the bit string modifications that preserve Hamming distances, we propose to further speed up the HWT by combining these modifications with *unbalanced bit strings*[1]. We conduct experiments with real-life datasets of image visual descriptors, and we show that query evaluations with the HWT are speeded up to 6 times when we apply the distance preserving modifications on balanced bit strings, and up to 30 times with unbalanced bit strings. The notation used throughout this paper is provided by table I.

The rest of the paper is organised as follows. Section II contains a description of the HWT, Section III contains a study of data modifications to make the HWT lower bounds tighter while preserving pairwise Hamming distances between bit strings and investigation of bit strings with unbalanced bits. Section IV provides experiments to show the influence of the proposed data modifications on the tightness of the lower bounds, Section V contains measurements of the HWT efficiency, and Section VI provides conclusions and description of the future work.

[1]i.e. bit strings with bits set to 1 in a different ratio then a half of bit strings $o \in X$

A. Related work

Embedding to the Hamming space is often used in similarity search to reduce the volume of processed data and to speed up query processing [14], [1], [6]. The speed up is also enabled by the efficiency of Hamming distance evaluation, but the sequential evaluation of all distances $h(q, o), o \in X$ is not efficient enough for many applications [9], [10]. Indexing of the Hamming space is difficult due to the dimensionality curse which says that efficiency of indexes degrades towards sequential scan with increasing data complexity [11], [12].

For instance, Norouzi et al. [15] propose to split indexed bit strings to sub-parts of the same length. Then limits for Hamming distances between substrings of q and its nearest neighbours are derived, considering a given query radius $h(q, o_k)$. These limits are utilised to efficiently identify candidate bit strings o for which the precise Hamming distance $h(q, o)$ is evaluated. Then the kNN queries are processed using an incremental search strategy, so the efficiency of this approach decreases with increasing query radius.

Pagh et al. [16] focus on the *Jaccard similarity* and improve a concept of *min-hashing* [17], [18]. They transform the original vectors to short bit strings called the *odd sketches*. These bit strings are compared by the Hamming distance, and they are proved to approximate the Jaccard similarity of original sets.

Besides direct indexing of bit strings, some authors propose techniques that combine more approaches to boost the efficiency of the similarity search. For instance, Jegou et al. [19] speed up the similarity search based on local descriptors [20]. They provide binary signatures to refine the matching based on the visual words [19]. A similar paper, but focused on a search based on global descriptors exists as well [21]. This paper contains a proposal to combine bit strings with an arbitrary metric index. Seidl and Kriegel [22] have analysed lower bounding, and they have discussed optimality of searching algorithms. The tightness of lower bounds determines their usefulness, that is the key motivation behind this paper.

II. Hamming Weight Tree

The *Hamming Weight Tree* (HWT) [10] have been proposed to efficiently evaluate similarity queries in the Hamming space. It utilises the L_1 norm of bit string o, i.e. the number of bits set to 1 in o, to provide a lower bound on the Hamming distance $h(o_1, o_2)$. We denote this L_1 norm of bit string o the *weight* $w(o)$. The lower bound is defined as:

$$h(o_1, o_2) \geq |w(o_1) - w(o_2)|. \tag{1}$$

The HWT built on bit strings of length λ is a tree with a root and up to $\lambda + 1$ nodes in level 1 (see Figure 1[2]).

[2]Figure adopted from [10]

(S_λ, h)	Hamming space (domain and Hamming distance)
λ	length of bit strings
$X, X' \subseteq S_\lambda$	dataset and its transformation
q, k	query bit strings, number of objects to be found
$o = (0, 1)$	bit string o with two bits 0 and 1
$w(o)$	weight of a bit string o (number of 1s)
$o[a..b]$	the bits in bit string o from position a to position b (including)
l	level of the HWT
$\pi = 2^{l-1}$	number of parts into which bit strings o are split at level l of the HWT
$\beta = \lambda/2^{l-1}$	number of bits in a part of o
[a, b, c, d]	weights in quarters of bit strings
lb_l	lower bound on the Hamming distance, as defined by Equation 3
$\mathcal{F}$	indexes of bits to flip
flipD$(X, \mathcal{F})$	function that flips bits $\mathcal{F}$ in dataset X
perm	permutation of indexes $\{0, .., \lambda - 1\}$
$corr(2, 4)$	Pearson correlation coefficient of the second and fourth bit (numbered from 0)
M, Ξ	correlation matrices
ϱ	ratio of 1s in particular bits of $o \in X$ (for unbalanced bits)

Table I: Notation used throughout this paper

Let us denote them $N_0..N_\lambda$. Each node N_i covers the bit strings $o \in X$ such that $w(o) = i$, and only non-empty nodes are stored to reduce the memory occupation. Since the pruning capabilities based on Equation 1 are rather small, authors [10] propose to split nodes of the HWT that contain more than t bit strings for some selected threshold t. In general, nodes in level l consider bit strings split into 2^{l-1} parts and weights are then evaluated on these disjunctive parts. Let us denote $o[a..b]$ the bits in bit string o from position a to position b (including), and a size of block β:

$$\beta = \frac{\lambda}{2^{l-1}}. \tag{2}$$

Then the lower bound lb_l provided by level l of the HWT is defined as [10]:

$$lb_l(o_1, o_2) = \sum_{i=0}^{2^{l-1}-1} |w(o_1[i \cdot \beta..(i+1)\beta - 1]) - w(o_2[i \cdot \beta..(i+1)\beta - 1])| \tag{3}$$

This lower bound has a potential to be tighter in deep levels of the HWT. For instance, having a query bit string q with weights [32, 32] in its halves, it cannot be in the distance lower than 2 from bit strings in node [31, 33]. However, when the HWT just with level 1 is used, no lower bound on the Hamming distance can be set on these bit strings.

KNN queries with query bit string q are processed using an incremental search strategy. It tries to find the k bit strings within distance 0 to q, and then increments the searching radius rad until the lower bounds lb_l ensures that unattended nodes cannot contain more similar bit strings.

The algorithm is formally described on-line[3]. Please notice that this approach provides the precise query evaluation.

The efficiency of the HWT is mainly influenced by (1) the tightness of the provided lower bounds, and (2) the *query radius*, defined as the distance $h(q, o_k)$ to the kth nearest neighbour of q. Query evaluation time increases rapidly with this radius, so we focus on both these features in the following sections to speed up the query evaluation with HWT.

III. Data Modifications for Tighter Lower Bounds

We consider the Hamming space (S_λ, h) with the domain S_λ of bit strings of length λ and the Hamming distance h. Having a dataset $X \subseteq S_\lambda$ and a set of indexes $\mathcal{F} \subseteq \{0, .., \lambda - 1\}$ we introduce a transformation *flipD* of the dataset X to create another dataset $X' \subseteq S_\lambda$:

$$\mathit{flipD}(X, \mathcal{F}) = X'. \tag{4}$$

Function *flipD* perform XOR of bits indexed in $\mathcal{F}$ on each bit string $o \in X$. Formally, it apply a function *flipS*:

$$\mathit{flipS}(o, \mathcal{F}) = o',$$

on each $o \in X$ and $o' = o'[0, .., \lambda - 1]$ such that:

$$o'[i] = \begin{cases} \neg o[i] & \text{if } i \in \mathcal{F} \\ o[i] & \textit{otherwise}. \end{cases}$$

The key property of function *flipD* is described by the following Lemma:

Lemma 1 (Preserving of the Hamming distances): Using an arbitrary set $\mathcal{F}$, function $\mathit{flipD}(X, \mathcal{F})$ preserves the pairwise Hamming distances of bit strings. Formally:

$$\forall \mathcal{F} \subseteq \{0, .., \lambda - 1\} : \forall o_1, o_2 \in X : \\ h(o_1, o_2) = h(\mathit{flipS}(o_1, \mathcal{F}), \mathit{flipS}(o_2, \mathcal{F}))$$

Proof: Lemma holds trivially for $\mathcal{F} = \emptyset$. If $|\mathcal{F}| = 1$ and arbitrary bit strings o_1, o_2 have the same value in bit $i \in \mathcal{F}$, i.e. $o_1[i] = o_2[i]$, the equality is not influenced by the flipping, and if originally $o_1[i] \neq o_2[i]$ then the flipping does not change this inequality as well. Therefore $h(o_1, o_2) = h(o'_1, o'_2)$. The induction step is trivially done for $|\mathcal{F}| > 1$, so the Hamming distances are not influenced by the flipping *flipD*. ∎

We use terms *flip* and *flipping* further in this paper to refer to the dataset X modifications by function *flipD* defined by Equation 4.

While the flipping of bits does not influence pairwise Hamming distances, it may change weights $w(o), o \in X$ and a tightness of the lower bounds defined by Equation 3. A trivial example is given by a dataset $|X| = 2$ with bit strings $o_1 = (0, 1), o_2 = (1, 0)$. The Hamming distance $h(o_1, o_2)$ is 2, and $lb_l(o_1, o_2)$ for level $l = 1$ of the HWT is 0, since $w(o_1) = w(o_2) = 1$. However, after flipping the second bit of these bit strings, we get $o_1 = (0, 0), o_2 = (1, 1)$ and thus the lower bound $lb_l(o_1, o_2)$ on this level is 2, i.e. tight.

A. Selecting Bits to Flip

Having a dataset $X \subseteq S_\lambda$, the task is to select the set $\mathcal{F}$ to utilize lower bounds lb_l at maximum, i.e. to make them as tight as possible.

Intuitively, lower bound defined in Equation 1 results in higher value if weight w(o) have extreme values across $o \in X$. Having more extreme values in w(o) (either close to 0 or λ) for bit strings $o \in X$ means, that more bits of $o \in X$ have the same values. This trend is exactly expressed by bit correlations over bit strings in X. In this paper, we use the Pearson correlation coefficient $corr(i, j)$ to describe correlation of bits i, j. Please notice, that this is well illustrated by the above mentioned example as well: Having the bit strings $o_1 = (0, 1)$ and $o_2 = (1, 0)$, the correlation $corr(0, 1)$ of their bits is -1. After the flipping of their one bit, this correlation is $+1$. The following lemma allows to better understand the influence of the flipping on pairwise bit correlations:

Lemma 2: Having two arbitrary lists of binary values I and J, e.g. values in the ith and jth bit of bit strings $o \in X$, and list $\neg I$ of the negated values from I, the following holds[4] for the Pearson correlation coefficient $corr$:

$$corr(I, J) = -corr(\neg I, J).$$

Lemma 2 is used in the following to formalise the problem of the selection of the set $\mathcal{F}$.

We transform the challenge to select the set $\mathcal{F}$ to another problem defined on a general correlation matrix:

Challenge 1: Having a correlation matrix $M : \lambda \times \lambda$ (calculated for dataset X) and operation $\mathit{flipD}(X, \mathcal{F})$ which for each $i \in \mathcal{F}$ changes the sign of all values in ith column and row of the matrix M (thus value $M(i, i)$ is changed twice, i.e. remains equal to $+1$). The goal is to select the set $\mathcal{F}$ to maximize sum C of all values in M:

$$C = \sum_{0 \leq i < j < \lambda} M(i, j). \tag{5}$$

For now, we postpone a solution of this challenge, as we further enhance it and solve it in the following sections.

B. Selecting Bits to Flip For HWT Levels

Next, we focus on a pruning ability of the HWT in its particular levels. As described above, the maximisation of the sum of correlations over the whole correlation matrix M is motivated by pushing weights $w(o)$ of bit strings $o \in X$ to extreme values, i.e. towards 0 and λ. Such an approach

[3]https://fi.muni.cz/~xmic/sketches/HWT-knn.pdf

[4]Proof of this lemma is provided at http://fi.muni.cz/~xmic/sketches/Corr_flipped_bit.pdf

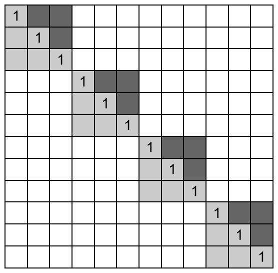


Figure 2: Correlations summed in Equation 6 for $\lambda = 12$ and $\pi = 4$

leads to an efficient space pruning at level $l = 1$ of the HWT, in which it compares the weights of the whole bit strings o – see Equation 3.

However, pruning ability of the HWT at level 1 is rather weak, since the weights $w(o), o \in X$ tends to cluster around their mean value and the nodes with most of the bit strings o are usually accessed.

Therefore, we propose to focus on the other levels of the HWT, i.e. we consider bit strings split into $\pi = 2^{l-1}$ parts where $l \geq 2$. In this scenario, we want to maximise the sum of correlations in subparts of the correlation matrix $M : \lambda \times \lambda$, which corresponds to pairwise bit correlations in particular substrings of $o \in X$. Visualisation of these correlations within matrix M is provided in Figure 2 for $\lambda = 12$ and $\pi = 4$. Pairwise correlations within 4 blocks of bits are shown in grey colours. When summing them, we focus just on those in dark grey, since each correlation matrix have 1s in the main diagonal, and it is symmetric. Let us formalize the whole problem in the following challenge:

Challenge 2: Having a value $\pi = 2^{l-1}$, the goal is to employ the flipping of bits as defined by Equation 4 and Challenge 1 to maximize value:

$$C = \sum_{p=0}^{\pi-1} \left(\sum_{i=p\cdot\frac{\lambda}{\pi}}^{(p+1)\cdot\frac{\lambda}{\pi}-2} \left(\sum_{j=i+1}^{(p+1)\cdot\frac{\lambda}{\pi}-1} M(i,j) \right) \right) \tag{6}$$

We point out, that for $\pi = 1$, i.e. in case of no splitting of bit strings $o \in X$, the Equation 6 degrades to Equation 5, as was defined in Challenge 1. The solution to this challenge is postponed again, as we enhance and solve it in the following section.

C. Flipping and Permuting Bits

The Challenge 2 tries to maximise correlations in subparts of the bit strings by flipping individual bits. There is another way to increase these correlations: we may permute bits of bit strings $o \in X$ to put highly correlated bits to the same block. Permuting bits of $o \in X$ does not influence the pairwise Hamming distances as well. Let us formulate the third and final challenge to transform dataset X to dataset X' in a way that preserves the pairwise Hamming distances and it increases the pruning ability of the HWT:

Challenge 3: Given a dataset X, the goal is to find:

- set $\mathcal{F} \subseteq \{0, .., \lambda - 1\}$, and
- permutation *perm* of bits

in a way, that flipping bits as defined by Equation 4 and then their permuting according to *perm* maximize value C for a given value π as defined by Equation 6.

The selection of bits $\mathcal{F}$ and permutation *perm* should be probably solved together, as one task influences the second one and vice versa. It constitutes a complicated problem from the linear algebra, that deserves to be solved independently of this paper. We verify our findings using an efficient greedy algorithm to find the set $\mathcal{F}$ and permutation *perm* (see Algorithm 1). Our approach is proposed to utilise the maximum absolute values of correlations. It flips particular bits to make these correlations positive, and it puts corresponding bits into the same blocks.

Algorithm 1 Selection of bits $\mathcal{F}$ and permutation *perm*

```
Input: number of parts $\pi$ into which bit strings $o$ are split
Input: correlation matrix $M$
Output: $\mathcal{F}$: set of bits to flip
Output: perm: the permutation of bits
  $\beta \leftarrow \lambda/\pi$                                  ▷ size of block
  idxToProc $\leftarrow \{0, .., \lambda\}$
  $\mathcal{F} \leftarrow \emptyset$
  perm = array()
  while idxToProc $\neq \emptyset$ do
      currBlock $\leftarrow$ array()              ▷ permutation for 1 block
      select $i, j$ such that $i, j \in$ idxToProc to maximize value $|M(i,j)|$
      currBlock.add($i, j$)
      idxToProc.remove($i, j$)
      if $M(i,j) < 0$ then
          $\mathcal{F}$.add($j$)
          switch sign of values in $j$th column and row of $M$
      while currBlock.size $< \beta$ do
          find $j$ to maximize value $|\sum_{i \in currBlock} M(i,j)|$
          if $\sum_{i \in currBlock} M(i,j) < 0$ then
              $\mathcal{F}$.add($j$)
              switch sign of values in $j$th column and row of $M$
          currBlock.add($j$)
      perm.addAll(currBlock)
```

In particular, this algorithm keeps an array of indexes to process *idxToProc*, and it processes them per blocks of size $\beta = \lambda/\pi$. At the beginning of a block processing, it finds indexes i, j such that $i, j \in$ *idxToProc* and the absolute value of correlation $|M(i,j)|$ is the largest. If $M(i,j) < 0$

then the jth bit is flipped[5]. Then it repeatedly finds index j which maximizes value $|\sum_{i \in currBlock} M(i,j)|$, and if this is negative, the bit j is flipped. Such indexes j are being added until the current block of bits has the desired size. Then the rest of the blocks is found in the same way.

D. Bit Strings with Unbalanced Bits

The bit strings are often made as sketches of complex data objects utilised to speed up similarity search (see Section I-A for details). The transformation techniques typically produce sketches with *balanced bits*, i.e. each bit i is set to 1 in half of the bit string sketches $o \in X$. At the same time, sketches o usually have low correlated bits, as low pairwise correlations lead to good compression ratio [4]. However, these two properties cause a poor indexability [5], as the balanced bit strings have a maximum mean Hamming distance $h(o_1, o_2)$ [12] and if their bits are uncorrelated, they have the smallest variance [4]. Therefore, they have maximum distances to nearest neighbours [5].

This reasoning resulted in a recent proposal to use sketches with *unbalanced bits* [5]. In this proposal each bit of $o \in X$ contains a fixed ratio $\varrho \neq 0.5$ of 1s. These bit strings have lower distances to their nearest neighbours thanks to lower mean Hamming distance [5][6]:

$$mean = 2\lambda \cdot \varrho \cdot (1 - \varrho) \tag{7}$$

Authors of paper [5] experimentally verify that if the sketches with unbalanced bits have sufficient length λ, they have a similar ability to describe similarity relationships as the sketches with balanced bits. Quite a lot of transformation techniques are tunable to produce bit strings with unbalanced bits. However, according to our best knowledge, there is no paper that experimentally verifies a hypothesis from [5] that such bit strings are easier indexable. Therefore, we conduct experiments with the HWT to compare indexability of balanced and unbalanced bit strings in this paper.

IV. Experiments – Tightness of the Lower Bounds

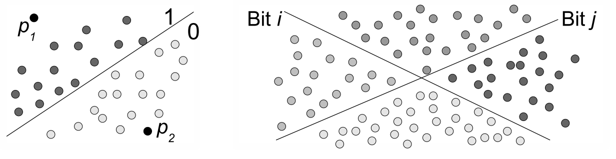
We verify theoretical findings about the permuting and flipping bits. At first, we describe testing data in Section IV-A, then we present results in Section IV-B.

A. Testing data

We use two datasets of visual descriptors extracted from images, and we further transform them into bit strings. The first dataset consists of a combination of five MPEG-7

[5] It can be seen further, that the selection of bit to flip here (i or j) does not play a role, since if i is flipped instead of j, complementary set of $\mathcal{F}$ is selected at a given block. Please notice, that if a complementary bits are flipped, it does not influence pairwise bit correlations at all.

[6] More precisely, lower distances to nearest neighbours are a consequence of a lower mean value and lower bounded variance of the Hamming distance [5].



(a) GHP to set values in one bit (b) Two GHPs to set two bits of bit strings

Figure 3: Generalized hyperplane partitioning to binarize space

visual descriptors [23] that were provided by the CoPhIR[7] data collection [24]. Each descriptor is accompanied with a suitable similarity function [23], and all five descriptor spaces are combined into a single space by a weighted sum of particular distances. In total, this representation can be considered as a 280-dimensional vector. We use the 10-million and 100-million collections of these MPEG7 descriptors.

The second dataset is formed by 20 million *DECAF* [25], [26] descriptors from the *Profiset collection*[8]. These descriptors are 4,096-dimensional vectors of float numbers taken as an output from the last hidden layer of a deep convolutional neural net [27]. These descriptors are compared with the Euclidean distance L_2.

We binarize these datasets by the technique adopted from paper [4], which is based on *generalized hyperplane partitioning* (GHP) (see Figure 3). A pair of pivoting descriptors (*pivots*) p_{i1}, p_{i2} is selected for each bit $i \in \{0, .., \lambda - 1\}$, and value of bit $o[i]$ expresses which of these two pivots is closer to o. Therefore, one instance of GHP determines one bit of all bit strings $o \in X$. The pivot pairs are selected [4] to produce low correlated and *balanced* bits, i.e. each bit i of bit strings $o \in X$ contains a half of ones and a half of zeros. In particular: (1) an initial set of pivots P_{sup} is selected at random, (2) balance of GHP is evaluated using a sample set of descriptors for all pivot pairs $(p_1, p_2), p_1, p_2 \in P_{sup}$, (3) set P_{bal} is formed by all pivot pairs that divide the sample set into two parts balanced with tolerance 0.05 (at least 45 % to 55 %) and corresponding bit strings o_{bal} with balanced bits are created. (4) The absolute value of the Pearson correlation coefficient is evaluated for all pairs of bits of bit strings o_{bal} to form correlation matrix Ξ, and (5) a heuristic[9] is applied to select rows and columns of Ξ, which form its sub-matrix of size $\lambda \times \lambda$ with low values. Finally, the pivot pairs which produce the corresponding low correlated bits define bit strings o. We denote the datasets produced by this technique *MPEG7_10M*, *MPEG7_100M* and *DECAF_20M*.

Besides this technique, we employ the technique proposed in [5], which is similar to the previous one, but the pivots

[7] http://cophir.isti.cnr.it/

[8] http://disa.fi.muni.cz/profiset/

[9] available at https://www.fi.muni.cz/~xmic/sketches/SelLowCorBits.pdf

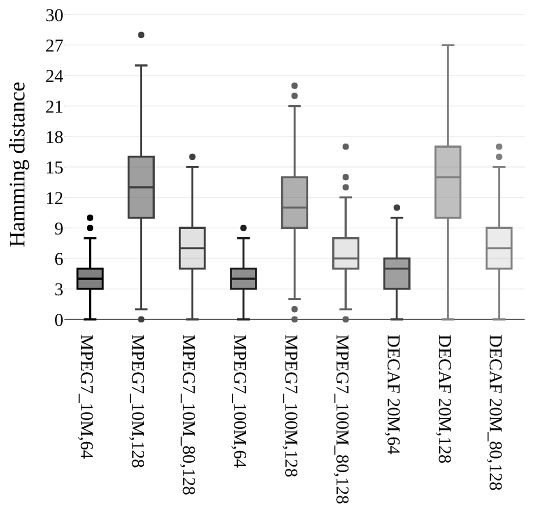


Figure 4: Distances to 1 nearest neighbour on particular datasets

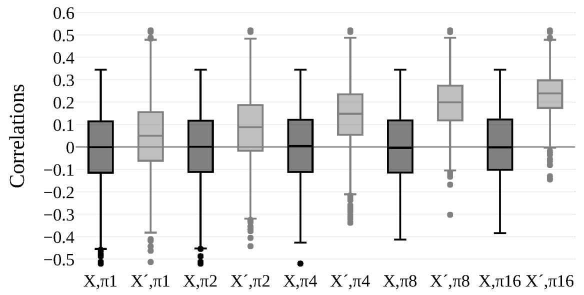


Figure 5: Pairwise bit correlations in blocks of bit strings with balanced bits

P_{bal} are selected to produce bits containing 80 % of ones and 20 % of zeros. It has been discussed [5] that when sufficient length λ of bit strings is used, these unbalanced bit strings are of similar quality, i.e. they approximate similarity relationships of the original descriptors at the similar level as bit strings with balanced bits. However, these bit strings should be indexable more easily due to their lower *intrinsic dimensionality* [5]. The mean Hamming distance on these bit strings is $0.32 \cdot \lambda$ according to Equation 7. We denote the datasets produced by this technique *MPEG7_10M_80*, *MPEG7_100M_80* and *DECAF_20M_80*.

We are using bit strings of lengths 64 and 128 bits. Since the distance between the query bit string q and its nearest neighbour (query radius) strongly influences the efficiency of the HWT, we depict these values for 1,000 query objects in Figure 4. We use standard *box plots* to show distribution of measured values in this paper.

B. Correlations & Lower Bounds: Balanced Bit Strings

We conduct experiments on the dataset *MPEG7_10M* with balanced bit strings of length λ = 128 to show the influence of Algorithm 1 on pairwise bit correlations. We focus on correlations $M(i,j)$ summed in Equation 6 that are key to tighten lower bounds lb_l. We consider bit strings split into $\pi \in \{1,2,4,8,16\}$ parts[10], and we depict distribution of these correlations in a box plot in Figure 5. Correlations in the original dataset X are in dark grey, and correlations in the dataset X' which is made from X by the Algorithm 1 are in light grey. This experiment illustrates that Algorithm 1 rise correlations $M(i,j)$ summed in Equation 6. Let us remind, that positive pairwise bit correlations within the blocks of bit strings split into 2^{l-1} parts tighten lower bounds lb_l on the

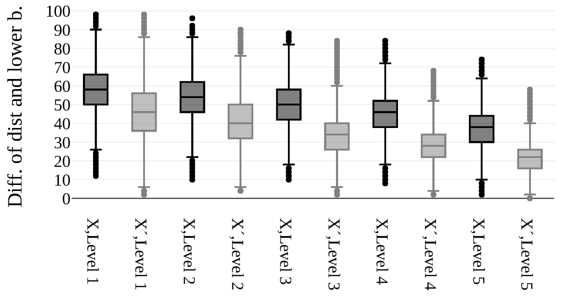


Figure 6: Difference between the Hamming distance and lower bound $h(o_1,o_2)-lb_l(o_1,o_2)$, bit strings with balanced bits

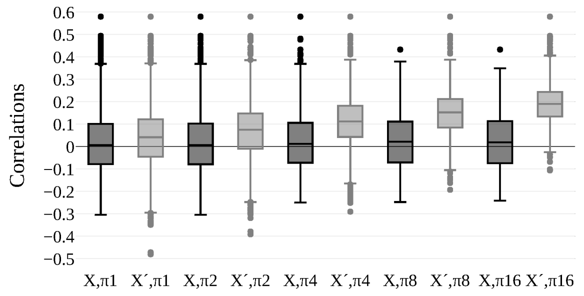


Figure 7: Pairwise bit correlations in blocks of bit strings with unbalanced bits

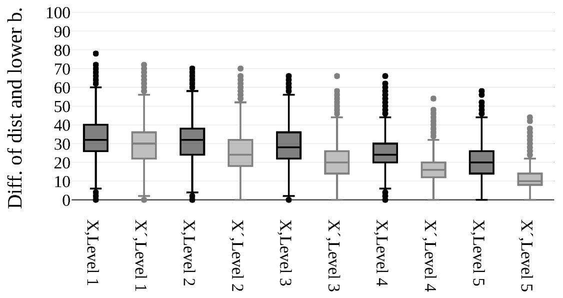


Figure 8: Difference between the Hamming distance and lower bound $h(o_1,o_2)-lb_l(o_1,o_2)$, bit strings with unbalanced bits

[10] If π = 1 then permuting bits does not influence the result and just the flipping of bits matters.

Hamming distance $h(q, o), o \in X$ as provided by the level l of the HWT. We verify this in the following experiment.

Figure 6 depicts differences $h(o_1, o_2) - lb_l(o_1, o_2)$. We depict results for the HWT with unmodified bit strings $o \in X$ in dark grey. In light grey are shown results for dataset X transformed to X' by Algorithm 1 with the π value corresponding to the proper level l. Please notice, that for the final experiments we are going to select just one[11] value π. The median of this difference $h(o_1, o_2) - lb_l(o_1, o_2)$ is decreased by 21 % (from 58 to 46) at the level 1, and up to 42 % (from 32 to 22) at level 5. The final measurements of efficiency of the HWT are presented in Section V.

C. *Correlations & Lower Bounds: Unbalanced Bit Strings*

We employ dataset *MPEG7_10M_80* in the following experiments, and we show the influence of Algorithm 1 on pairwise bit correlations summed in Equation 6 in Figure 7. Therefore Figure 7 is analogue to Figure 5. Beside an increase of summed correlations, we can observe a property of bit strings with bits balanced to ratio[12] ϱ: The correlations of the unbalanced bits are strictly lower bounded [5], in case of $\varrho = 0.8$ by -0.25 (for our data, it is $-1/3$ because of a tolerance 0.05 on $\varrho = 0.8$). Since the correlations of unbalanced bits are upper bounded by $+1$, as usual, maximum correlations have usually significantly higher absolute values than negative ones (see dark grey box plots in Figure 7). As a consequence, average pairwise correlation of unbalanced bits tends to be slightly positive. This is another property of unbalanced bits favourable for the HWT, according to our previous investigation of the lower bounds lb_l.

Let us focus on tightness of the lower bounds $lb_l(o_1, o_2)$. We conduct a similar experiment as in Section IV-B to show differences between the actual Hamming distance $h(o_1, o_2)$ and lower bounds $lb_l(o_1, o_2)$ provided by different levels l of the HWT. Results in Figure 8 confirm that the approach proposed in this paper makes lower bounds lb_l tighter. Let us emphasize a 50% decrease of median of difference $h(o_1, o_2) - lb_l(o_1, o_2)$ for level $l = 5$ (from 20 to 10). Moreover, depicted differences are significantly smaller than in case of balanced bits (compare Figures 8 and 6), and therefore these results comply with the hypothesis, that bit strings with unbalanced bits form an easier indexable search space.

V. Experiments – Efficiency of the HWT

We measure the efficiency of the HWT precise similarity search regarding search time (in seconds). Our implementation is based on the C++ code used in [10]. We evaluate 1,000 1NN queries on each configuration, and we present average values. We split the nodes of the HWT when they contain $t = 1000$ bit strings, and the max depth of HWT is $l = 5$.

A. *Results on datasets MPEG7_10M and MPEG7_10M_80*

Table II contains search times for the datasets *MPEG7_10M* (balanced bits) and *MPEG7_10M_80* (unbalanced bits). We present result using two lengths of bit strings $\lambda \in \{64, 128\}$ bits. Let us remind, that bit strings with unbalanced bits must be of a sufficient length λ to provide similar quality[13] as bit strings with balanced bits [5]. For this reason, we do not show results for $\varrho = 0.8, \lambda = 64$: these bit strings represent a too strong simplification of the search space. As a consequence, the search times with the HWT are drastically lower using these bit strings (usually approximately 0.003 s – 0.005 s).

The first two lines of Table II contain average times of sequential evaluation of all distances $h(q, o), o \in X$. The next two lines show results on queries with the HWT on the original (not modified) datasets X. Evaluation with the HWT on bit strings $\lambda = 64, \varrho = 0.5$ is 2.4 times more efficient than the sequential evaluation, but it is about 8 times slower using bit strings $\lambda = 128, \varrho = 0.5$. For unbalanced bits ($\varrho = 0.8$), the HWT is about 1.5 times slower than the sequential evaluation using length λ = 128 bits.

The rest of the Table II contains results measured with the same implementation of the HWT, but on datasets modified by Algorithm 1. We examine values $\pi \in \{1, 2, 4, 8, 16\}$. Evaluations with the HWT built on short and balanced bit strings $\lambda = 64, \varrho = 0.5$ are up to 1.6 times more efficient when the proposed modifications are employed – (0.026 s versus 0.042 s), and it is 3.9 times faster then with the sequential evaluation.

Evaluations on longer and balanced bit strings $\lambda = 128, \varrho = 0.5$ are sped up 6.1 times (0.196 s vs. 1.196 s), but even this is insufficient to outperform the efficiency of the sequential evaluation (0.146 s). When the HWT is employed with unbalanced and modified bit strings $\lambda = 128, \varrho = 0.8$, query evaluations are up to 1.8 times more efficient than the sequential scan (0.083 s vs. 0.148 s) and they are 14.5 times faster then the HWT on unmodified bit strings with balanced bits (1.196 s).

B. *Results on datasets MPEG7_100M and MPEG7_100M_80*

Table III provides results measured on 100 million datasets *MPEG7_100M* and *MPEG7_100M_80*. Query evaluations with our modifications on short bit strings $\lambda = 64, \varrho = 0.5$ are up to 34 times faster than the sequential evaluation of all distances (0.03 s vs. 1.017 s). Our proposals speed up the HWT evaluation 6 times (0.03 s vs. 0.182 s).

[11]The reason is, that if the bit strings are modified for each tree level independently, the tree structure changes. Since the optimisation of such a tree is a different and complex problem, we consider it separately of this paper as future work.

[12]therefore it is valid just for dark grey box plots, since if a bit i is flipped, it is balanced to ratio $1 - \varrho$

[13]that is an ability to approximate similarity relationships of original data objects

	Balance of bits	λ = 64	λ = 128
Sequential evaluation	$\varrho = 0.5$	0.103	0.146
	$\varrho = 0.8$	–	0.148
HWT original	$\varrho = 0.5$	0.042	1.196
	$\varrho = 0.8$	–	0.219
HWT $\pi = 1$	$\varrho = 0.5$	0.036	0.335
	$\varrho = 0.8$	–	0.116
HWT $\pi = 2$	$\varrho = 0.5$	0.037	0.269
	$\varrho = 0.8$	–	0.116
HWT $\pi = 4$	$\varrho = 0.5$	0.036	0.196
	$\varrho = 0.8$	–	0.083
HWT $\pi = 8$	$\varrho = 0.5$	0.031	0.320
	$\varrho = 0.8$	–	0.129
HWT $\pi = 16$	$\varrho = 0.5$	0.026	0.585
	$\varrho = 0.8$	–	0.182

Table II: Datasets *MPEG7_10M* and *MPEG7_10M_80*, query evaluation times (in seconds)

	Balance of bits	λ = 64	λ = 128
Sequential evaluation	$\varrho = 0.5$	1.017	1.498
	$\varrho = 0.8$	–	1.503
HWT original	$\varrho = 0.5$	0.182	6.463
	$\varrho = 0.8$	–	0.886
HWT $\pi = 1$	$\varrho = 0.5$	0.064	2.704
	$\varrho = 0.8$	–	0.327
HWT $\pi = 2$	$\varrho = 0.5$	0.057	2.478
	$\varrho = 0.8$	–	0.291
HWT $\pi = 4$	$\varrho = 0.5$	0.040	1.845
	$\varrho = 0.8$	–	0.269
HWT $\pi = 8$	$\varrho = 0.5$	0.030	2.237
	$\varrho = 0.8$	–	0.214
HWT $\pi = 16$	$\varrho = 0.5$	0.059	2.540
	$\varrho = 0.8$	–	0.270

Table III: Datasets *MPEG7_100M* and *MPEG7_100M_80*, query evaluation times (in seconds)

Evaluations on longer and balanced bit strings $\lambda = 128, \varrho = 0.5$ remain slower than the sequential scan, but evaluations with unbalanced bits $\lambda = 128, \varrho = 0.8$ are up to 7 times faster then sequential processing (0.214 s vs. 1.503 s). We emphasize, that the HWT is up to 30 times faster using unbalanced bit strings $\lambda = 128, \varrho = 0.8$ modified by Algorithm 1 than the HWT build on unmodified balanced bit strings $\lambda = 128, \varrho = 0.5$ (0.214 s vs. 6.463 s), and and it is 4.1 times faster than the HWT on unmodified unbalanced bit strings (0.214 s vs. 0.886 s).

C. Results on datasets DECAF_20M and DECAF_20M_80

Last experiments are conducted on datasets *DECAF_20M* and *DECAF_20M_80*, which are hard to index due to big distances to nearest neighbours (see Figure 4). The times of sequential scan are linear with the size of the datasets and in compliance with the previous measurements. Search on short bit strings $\lambda = 64, \varrho = 0.5$ is sped up 3.8 times with respect to the sequential evaluation (0.054 s vs 0.204 s). Our modifications speeded up the HWT 2.3 times in this case.

	Balance of bits	λ = 64	λ = 128
Sequential evaluation	$\varrho = 0.5$	0.204	0.301
	$\varrho = 0.8$	–	0.301
HWT original	$\varrho = 0.5$	0.122	2.798
	$\varrho = 0.8$	–	0.427
HWT $\pi = 1$	$\varrho = 0.5$	0.065	1.111
	$\varrho = 0.8$	–	0.264
HWT $\pi = 2$	$\varrho = 0.5$	0.067	0.974
	$\varrho = 0.8$	–	0.249
HWT $\pi = 4$	$\varrho = 0.5$	0.061	0.817
	$\varrho = 0.8$	–	0.231
HWT $\pi = 8$	$\varrho = 0.5$	0.054	1.218
	$\varrho = 0.8$	–	0.229
HWT $\pi = 16$	$\varrho = 0.5$	0.062	1.643
	$\varrho = 0.8$	–	0.262

Table IV: Datasets *DECAF_20M* and *DECAF_20M_80*, query evaluation times (in seconds)

The HWT does not outperform the sequential evaluations using longer bit strings $\lambda = 128$ with balanced bits. However, searching with the HWT is about 1.3 times faster than the sequential evaluation using unbalanced bit strings $\lambda = 128, \varrho = 0.8$ (0.229 s vs. 0.301).

VI. Conclusions

We have focused on indexing in the Hamming space which is based on evaluating the number of ones in the indexed bit strings. In particular, we have employed the recently proposed *Hamming Weight Tree* (HWT) which uses this type of lower bounding to prune the search space, and we have introduced two distance-preserving bit string modifications which make these lower bounds tighter. Our experiments on several real-life datasets have shown a significant speed-up of the HWT evaluating (even 6-times) induced by the proposed modifications.

Further, we have discussed that the bit strings are often created as *sketches* of complex objects, e.g. multimedia descriptors, and that they usually have *balanced bits*. Conversely, bit strings with *unbalanced bits* have been proposed [5] as a better indexable alternative to balanced bit strings. It has been reported [5] that unbalanced bit strings with sufficient length have a similar ability to describe pairwise similarity of the original data objects as balanced bit strings of the same length.

Since the theoretical properties of the unbalanced bit strings are favourable for the efficiency of the HWT, we have investigated our distance preserving data modifications with these bit strings as well. In this way, we have speeded up the search with the HWT up to 30 times comparing to the HWT on unmodified bit strings with balanced bits.

We have considered bit string optimisations for one selected level of the HWT. Naturally, these modifications could be done for each level of the HWT independently which should further improve the pruning effect. However, our preliminary experiments indicate that this idea imply radical changes of the HWT structure and therefore we postpone it as future work.

VII. Acknowledgments

This research was supported by ERDF „CyberSecurity, CyberCrime and Critical Information Infrastructures Center of Excellence" (No. CZ.02.1.01/0.0/0.0/16_019/0000822).

References

[1] M. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proceedings on 34th Annual ACM Symposium on Theory of Computing, 2002, Montréal, Québec, Canada*, 2002, pp. 380–388.

[2] W. Dong, M. Charikar, and K. Li, "Asymmetric distance estimation with sketches for similarity search in high-dimensional spaces," in *Proceedings of the ACM SIGIR conf. on Research and development in information retrieval*. ACM, 2008.

[3] Q. Lv, M. Charikar, and K. Li, "Image similarity search with compact data structures," in *Proc. of the 2004 ACM CIKM Int. Conf. on Information and Knowledge Management, USA, 2004*, 2004, pp. 208–217.

[4] V. Mic, D. Novak, and P. Zezula, "Designing sketches for similarity filtering," in *IEEE International Conference on Data Mining Workshops, ICDMW 2016, December 12-15, 2016, Barcelona, Spain.*, 2016, pp. 655–662.

[5] ——, "Sketches with unbalanced bits for similarity search," in *Similarity Search and Applications - 10th International Conference, SISAP 2017, Munich, Germany, 2017, Proceedings*, 2017, pp. 53–63.

[6] A. J. Muller-Molina and T. Shinohara, "Efficient similarity search by reducing i/o with compressed sketches," in *Proceedings of the 2nd Int. Workshop on Similarity Search and Applications*, 2009, pp. 30–38.

[7] Z. Wang, W. Dong, W. Josephson, Q. Lv, M. Charikar, and K. Li, "Sizing sketches: a rank-based analysis for similarity search," in *Proceedings of the 2007 ACM SIGMETRICS Int. Conf. on Measurement and Modeling of Computer Systems, USA*, 2007, pp. 157–168.

[8] P. Zezula, G. Amato, V. Dohnal, and M. Batko, *Similarity search: the metric space approach*. Springer, 2006, vol. 32.

[9] E. Ong and M. Bober, "Improved hamming distance search using variable length hashing," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, USA, 2016*, 2016, pp. 2000–2008.

[10] S. Eghbali, M. H. Z. Ashtiani, and L. Tahvildari, "Online nearest neighbor search in binary space," in *2017 IEEE International Conference on Data Mining, ICDM*, 2017, pp. 853–858.

[11] M. Skala, "Measuring the difficulty of distance-based indexing," in *String Processing and Information Retrieval, 12th International Conference, SPIRE 2005, Buenos Aires, Argentina, 2005, Proceedings*, 2005, pp. 103–114.

[12] ——, "Aspects of metric spaces in computation," Ph.D. dissertation, University of Waterloo, Ontario, Canada, 2008.

[13] M. E. Houle, H.-P. Kriegel, P. Kröger, E. Schubert, and A. Zimek, "Can shared-neighbor distances defeat the curse of dimensionality?" in *Int. Conf. on Scientific and Statistical Database Management*. Springer, 2010, pp. 482–500.

[14] J. Wang, W. Liu, S. Kumar, and S. Chang, "Learning to hash for indexing big data - A survey," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 34–57, 2016.

[15] M. Norouzi, A. Punjani, and D. J. Fleet, "Fast exact search in hamming space with multi-index hashing," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 6, pp. 1107–1119, 2014.

[16] M. Mitzenmacher, R. Pagh, and N. Pham, "Efficient estimation for high similarities using odd sketches," in *Proceedings of the 23rd international conference on World wide web*. ACM, 2014, pp. 109–118.

[17] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of massive datasets*. Cambridge University Press, 2014.

[18] P. Li and A. C. König, "Theory and applications of b-bit minwise hashing," *Commun. ACM*, vol. 54, no. 8, pp. 101–109, Aug. 2011.

[19] H. Jegou, M. Douze, and C. Schmid, "Hamming embedding and weak geometric consistency for large scale image search," in *10th European Conference on Computer Vision (ECCV), France, 2008, Proceedings*, 2008, pp. 304–317.

[20] J. Sivic and A. Zisserman, "Video google: A text retrieval approach to object matching in videos," in *null*. IEEE, 2003, p. 1470.

[21] V. Mic, D. Novak, and P. Zezula, "Speeding up similarity search by sketches," in *Similarity Search and Applications: 9th International Conference, SISAP 2016, Proceedings*. Cham: Springer International Publishing, 2016, pp. 250–258.

[22] T. Seidl and H.-P. Kriegel, "Optimal multi-step k-nearest neighbor search," in *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '98. USA: ACM, 1998, pp. 154–165.

[23] MPEG7, *Multimedia content description interfaces. Part 3: Visual*, ISO/IEC ISO 15 938-3:2002, 2002.

[24] M. Batko, P. Kohoutkova, and D. Novak, "Cophir image collection under the microscope," in *Similarity Search and Applications, SISAP'09*. IEEE, 2009.

[25] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition." in *Icml*, vol. 32, 2014, pp. 647–655.

[26] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Proceedings of the IEEE CVPR conference*, 2014.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," vol. 60, no. 6, 2017, pp. 84–90.