# Modifying Hamming Spaces for Efficient Search

Vladimir Mic, David Novak, Pavel Zezula

Masaryk University
Brno, Czech Republic

17th November 2018

# Similarity Search on Bit Strings – Motivation

- Searching for similar objects

- Wide range of applications
  - recommender systems, searching in biometrics, event detection, ...

# Similarity Search on Bit Strings – Motivation

- Searching for similar objects

- Wide range of applications
  - recommender systems, searching in biometrics, event detection, ...
- Original complex objects are often described by bit strings
  - We assume mapping 1 to 1 between bit strings and objects
- Similarity of objects ≈ similarity of bit strings
  - *Hamming distance* $h$:
    having two bit strings $o_1, o_2$, it evaluates number of different bits

# Problem: Efficiency of the Similarity Search

- Use case: *Query by example*
  - Search for the most similar bit strings to a given query bit string
- Problem: time needed for a query execution
- Evaluation of the Hamming distance $h$ is very efficient
  - $\approx 10^7$ Hamming distances are evaluated per second on an ordinary computer
- Problem: big datasets
- Solution: indexes

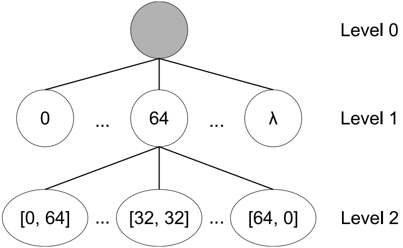
- *The Hamming Weight Tree* (*HWT*):
  indexing structure based on *weights* $w$ of bit strings
  - Sepehr Eghbali et al.: *Online Nearest Neighbor Search in Hamming Space*, ICDM 2017[1]
  - Weight $w(o)$ of a bit string $o$ is a number of bits in $o$ set to 1
- Observation: lower bound on the Hamming distance $h$:

$$h(o_1, o_2) \geq |w(o_1) - w(o_2)|$$

[1] www.cas.mcmaster.ca/ashtiani/papers/online-nearest-neighbor.pdf

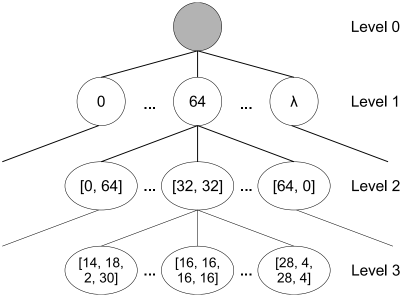
# The Hamming Weight Tree (paper from ICDM 2017) (2/5)

- Pruning ability of the weights of whole bit strings is weak
  - Lower bounds can be defined on a subparts of bit strings
- HWT exploits these lower bounds in a tree-like structure:
  - Artificial root
  - Level 1: up to $\lambda + 1$ nodes
    - Node labelled $i$ covers bit strings $o$ with weight $w(o) = i$
    - $\lambda$ is maximum length of bit strings



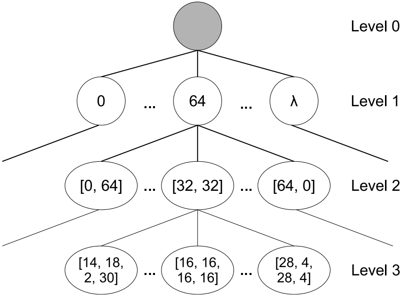
# The Hamming Weight Tree (paper from ICDM 2017) (3/5)

- Level 2: Nodes labelled by $[a, b]$
  - $a$ and $b$ are weights of first and second half of bit strings




- Level n: weights of $2^{n-1}$ parts of bit strings
  - Stored are just non-empty nodes
  - Dynamic depth of the HWT – maximum capacity of nodes, splitting
  - HWT is usually very unbalanced

# The Hamming Weight Tree (paper from ICDM 2017) (4/5)




- Overall lower bound on Hamming distance of two bit strings: sum of partial lower bounds
- Example:

| | | | | |
|---|---|---|---|---|
| partial weights of $o_1$ | 10 | 20 | 15 | 12 |
| partial weights of $o_2$ | 10 | 15 | 5 | 20 |
| partial lower bounds | 0 | 5 | 10 | 8 |

Lower bound on $h(o_1, o_2)$ is: $0 + 5 + 10 + 8 = 23$

# The Hamming Weight Tree (paper from ICDM 2017) (5/5)

- Search for $k$ most similar bit strings to bit string $q$
  - *Incremental search strategy*: search for bit strings $o$ in distance $h(q, o)$ equal to 0, then 1, 2 ...

    ... until the lower bounds in the HWT ensures that the rest of bit strings is less similar to $q$ then those already found[2]
- A tightness of the lower bounds is crucial

[2] *Details and full algorithm are in our paper*

# Our Contribution

- We investigate two ways to tighten lower bounds exploited by the HWT
- … both preserves pairwise Hamming distances $h$ of bit strings

# Flipping bits

- Flipping bits
  - Having dataset $X$ of bit strings, *XORing* some bits of all $o \in X$ may improve the lower bounds
  - Example: dataset with just two bit strings of length 2:

| | Before flipping | After flipping |
|---|---|---|
| $o_1$: | 0 1 | 0 0 |
| $o_2$: | 1 0 | 1 1 |
| $h(o_1, o_2)$: | 2 | 2 |
| lower bound on $h(o_1, o_2)$: | $\lvert 1-1 \rvert = 0$ | $\lvert 0-2 \rvert = 2$ |

# Flipping bits – Results of Our Analysis

- Which bits should be flipped?
- Consider the level 1 of the HWT (weight of all bit strings is compared)
  - Weights of bit strings should be *extreme* (either close to 0 or to $\lambda$)

$$h(o_1, o_2) \geq |w(o_1) - w(o_1)|$$

  - ... i.e. pairwise bit correlations should be positive[3]
  - Lemma[4]: When $i$th bit of all $o \in X$ is flipped, just signs of all pairwise correlations $Corr(i, j), 0 \leq j < \lambda \land j \neq i$ is changed:

$$Corr(i, j) = -Corr(\neg i, j)$$

[3] We use *Pearson correlation coefficient*

[4] Proved in the paper

# Bit Correlations - Example

| Bit number | 0 1 | 0 1 |
|---|---|---|
| | Before flipping | After flipping |
| $o_1$: | 0 1 | 0 0 |
| $o_2$: | 1 1 | 1 0 |
| $o_3$: | 0 1 | 0 0 |
| $o_4$: | 1 0 | 1 1 |
| Corr(0, 1) | -0.577 | +0.577 |

# Flipping bits – Results of Our Analysis

- Extension for other levels of the HWT:
  - Weights of particular subparts of bit strings should be extreme
  - ... we need to maximise pairwise bit correlations of bits within the parts (*i.e. halves, quarters, ...* ) of bit strings
- Let us now focus on a second way to tighten lower bounds ...

# Permuting bits

- Focus on levels deeper then 1 of the HWT
  - weights of subparts of bit strings are compared
  - Permutation of bits may improve the tightness of the lower bound provided by particular levels of the HWT
- Example: lower bounds provided by weights of the halves of bit strings

| | Before permuting | After permuting |
|---|---|---|
| Bit index: | 0 1 2 3 | 0 3 2 1 |
| $o_1$: | 0 1 1 0 | 0 0 1 1 |
| $o_2$: | 1 0 0 0 | 1 0 0 0 |
| $h(o_1, o_2)$: | 3 | 3 |
| lower bound: on $h(o_1, o_2)$: | $\lvert 1-1\rvert + \lvert 1-0\rvert = 1$ | $\lvert 0-1\rvert + \lvert 2-0\rvert = 3$ |

# Flipping and Permuting Bits

- We propose a greedy algorithm to determine
  - bits of bit strings to flip and
  - permutation of bits

  at once to put correlated bit to the same blocks of bit strings
- … and therefore to tighten lower bounds exploited by the HWT

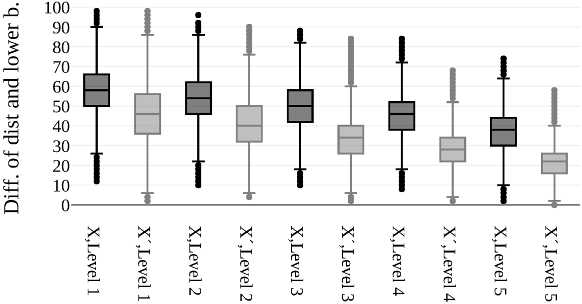



Figure: Differences of the Hamming distances $h$ and lower bounds provided by particular levels of the HWT
Dark: original bit strings, light: with proposed modifications

# Examples of results

| Dataset of 20 million bit strings (*DeCAF*) | $\lambda = 64$ |
|---|---|
| Sequential evaluation | 0.204 s |
| HWT original | 0.122 s |
| HWT with modified bit strings | 0.054 s |

| Dataset of 100 million bit strings (*MPEG7*) | $\lambda = 64$ |
|---|---|
| Sequential evaluation | 1.017 s |
| HWT original | 0.182 s |
| HWT with modified bit strings | 0.030 s |

Table: Times of the search for 1 most similar bit string to a query bit string $q$ (averages over 1,000 randomly selected $q$)

# Conclusions

- We are analysing weights of bit strings to exploit lower bounds on the Hamming distance
- We propose a heuristic that flips some bits of bit strings and permute them to tighten lower bounds exploited by the *Hamming Weight Tree* (HWT)
- Despite the progress in an efficiency of query evaluation, the HWT suffers from complex spaces