Masaryk University
Faculty of Informatics

Metric Sketches for Similarity Searching

Master's Thesis

Matej Kvaššay

Brno, Fall 2017

Masaryk University
Faculty of Informatics

Metric Sketches for Similarity Searching

Master's Thesis

Matej Kvaššay

Brno, Fall 2017

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Matej Kvaššay

Advisor: Pavel Zezula

Acknowledgement

I would like to express my gratitude to my advisor prof. Ing. Pavel Zezula, CSc. and RNDr. Vladimír Míč of the DISA laboratory, Faculty of Informatics, Masaryk University for their helpful guidance, patience and passionate participation during whole process making this diploma thesis. Furthermore I would like to thank other members of DISA laboratory and participants of DISA seminar for their helpful remarks and advice. I am also very grateful for the motivation and support that was provided to me by my mother and grandparents.

Abstract

Searching for objects in typically large collections of unstructured data which is compared by notion of similarity is one of fundamental tasks researched in the field of information retrieval. Metric space approach is addressing some challenges of similarity searching by projecting data into a metric space and exploiting it's properties to reduce number of object comparisons required in order to retrieve search results.

Metric sketches are compact in-memory fixed-length binary data representations of objects that are being utilized to speed up similarity searching in general metric spaces by applying filter & refine paradigm. Search is first conducted in the sketch space to identify set of result candidates which are then examined in the original space where comparison of objects is significantly more computationally demanding. To further speed up searching in sketch space we organize sketches in hash-based data structure called Multi-hash index.

In this thesis we define notion of metric sketches and discuss their properties and aspects of their creation, optimization and application for the task of similarity searching. Furthermore we explain core ideas behind the Multi-hash index and it's usage for searching among sketch representations. Finally, we evaluate contribution of sketches and Multi-hash index on real-life dataset consisting of feature vectors of images.

Keywords

similarity searching, metric sketches, multi hash index, metric distance, indexing, filtering, refinement

Contents

List of Tables

List of Figures

1 Introduction

Organizations all around the world are continuously gathering huge amounts of data from various sources. Weather sensors measurements, telecommunication data, on-line videos, insurance health records, videos, social media posts are only a small sample of types of large data collections that are being stored and utilized. Making value of all this stored information has created need for further advancements in research of organizing and processing large data collections effectively in respect to some intended practical tasks. Search is one such task, with goal to retrieve subset of database objects matching some specified criteria usually with emphasis on minimizing look-up time.

In traditional relational database systems containing structured records we usually define criteria explicitly by setting constraint on attribute values. That enables us for example to query for all rows from *user* table where value of attribute *height* is less than *180*.

However in current large data collections we often to deal with unstructured data types e.g. images, videos, sound, etc., making explicit criteria specification inconvenient. In that case common approach to is to present the system some reference object from the same domain and search the data collection for similar objects. Since notion of similarity can be vague, *Metric Space Approach* [1] provides mathematical foundations to quantify similarity between objects and speed up results retrieval by projecting objects into a *metric space* and exploiting it's properties.

Metric Sketches are novel approach used to speed up searching in metric spaces by transforming objects from the database to their compact in-memory binary representations, which can be used to quickly identify some subset of promising candidates for search query result before examining the original objects stored in the database. To improve search speed even further we'll examine *Multi-hash index*, a hashing-based data structure used for organizing binary strings for fast search query result retrieval.

In this thesis we'll describe basic concepts of Metric Space Approach, Metric Sketches and Multi-hash Index and evaluate them in experimental setting on real-life dataset consisting of high-dimensional image descriptors.

2 Similarity Searching in Metric Spaces

In this chapter we'll define core principles of Metric Space Approach.

2.1 Metric Space

Metric space $M = (D, d)$ consists of objects domain D and distance function

$$d\colon D \times D \to R$$

with following properties [1]:

$$\begin{aligned}
\textit{non-negativity:} \quad & \forall x, y \in D\colon & d(x,y) \geq 0 & \qquad (2.1)\\
\textit{symmetry:} \quad & \forall x, y \in D\colon & d(x,y) = d(y,x) & \qquad (2.2)\\
\textit{reflexivity:} \quad & \forall x \in D\colon & d(x,x) = 0 & \qquad (2.3)\\
\textit{positiveness:} \quad & \forall x, y \in D\colon & x \neq y \Rightarrow d(x,y) > 0 & \qquad (2.4)\\
\textit{triangle inequality:} \quad & \forall x, y, z \in D\colon & d(x,z) \leq d(x,y) + d(y,z) & \qquad (2.5)
\end{aligned}$$

Domain D specifies the class of objects distance function might be defined upon e.g. univariate vectors, integer matrices, continuous signals, etc. Distance function d, also referred to as *metric function* or *metric* is measure of dissimilarity between any couple of objects from domain D.

In distance computations are often computationally difficult operations and exploiting properties of metric space, most importantly triangle inequality (2.5) enables us to avoid them.

2.2 Similarity query

Goal of similarity query is for given query object $q \in D$ to retrieve result subset of objects $R \subseteq X$ of our data collection[1] $X \subseteq D$ closest to q.

1. Definition of similarity query can be furthermore extended for multiple data collections allowing for other elementary similarity query types such as *similarity joins* [1].

$$\begin{aligned} &\forall o \in R, \forall z \in X \setminus R: \\ &d(o,q) \leq d(z,q) \end{aligned} \tag{2.6}$$

Additionally to (2.6) we specify some constraint on R. Based on differences in this constraint we distinguish two most common types of similarity queries, namely *Range query* and *Nearest neighbor query* [2].

Range query Input to range query consists of query object q and distance range r. Our task is to retrieve all objects from the data collection within this range. Formally we define range query as:

$$RQ(q,r) = \{o \in X \mid d(o,q) \leq r\} \tag{2.7}$$

Constraint (2.7) restricts objects in the result set by their distance to q, meaning distance of objects in R from q is limited but cardinality of R varies depending on r and q.

Range query would for example be useful if we wanted to search a city map for all restaurants within some distance radius from given GPS position.

Nearest neighbor query Input to k Nearest neighbor query [3] commonly referred to as *k-NN* query is query object q and number of objects to retrieve k. Result set of this query contains k closest objects to q, if $|X| \geq k$.

$$kNN(q,k) = \{o \in R \mid R \subseteq X, |R| \leq k, \forall z \in X \setminus R, d(o,q) \leq d(z,q)\} \tag{2.8}$$

Note that in constraint (2.8) we restrict cardinality of R rather than distance of objects in R from q.

Nearest Neighbor query are utilized for instance in recommender systems [4], to classify objects based on patterns in data [5], etc.

2.3 Partitioning

Central idea of Metric Similarity Searching is avoiding unnecessary distance computations in order to reduce query evaluation time. Common way to do so is to split metric space in several sub-regions referred to as partitions. Conducting a search we avoid searching those partitions which are unlikely or impossible to contain objects from desired query result.

In pivot-based partitioning methods we first select some set of reference objects called *pivots* and then identify regions of the metric space object belongs to based on distances to these pivots.

The two most common partitioning methods are *Ball Partitioning* and *Generalized hyperplane partitioning*. Both of them split objects from the domain into two distinct partitions however it's possible to apply them several times in a row to achieve more granular partitioning of the space.

Ball Partitioning In ball partitioning we select pivot object p from the domain D and estimate median distance d_m from this pivot to other objects from the domain. Then for each object we compute distance $d(o, p)$ and assign either partition $P1$ or $P2$ based on following rule:

$$P1 = \{\forall o \in D \mid d(o, p) \leq d_m\} \tag{2.9}$$
$$P2 = \{\forall o \in D \mid d(o, p) \geq d_m\} \tag{2.10}$$

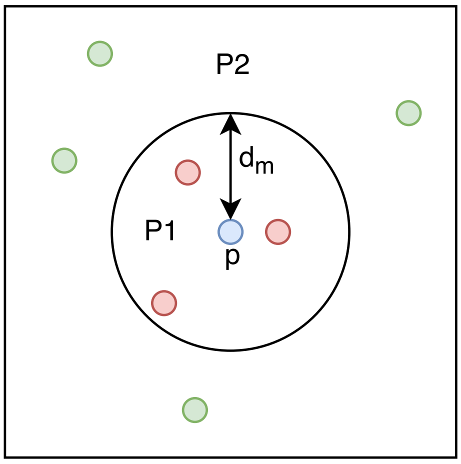

Figure 2.1: Ball partitioning.

The redundant $\leq, \geq$ in (2.9) and (2.10) ensures balanced splits in case multiple objects o_m fall at dividing distance $d(o,p) = d_m$ [6].

Generalized Hyperplane Partitioning Partition in generalized hyperplane partitioning (*GHP*) is determined by object's position in respect to couple of pivots $p_1, p_2, p_1 \neq p_2$ selected from the domain D. Hyperplane separates objects from the domain into two partitions:

$$P1 = \{\forall o \in D \mid d(o,p_1) \leq d(o,p_2)\} \tag{2.11}$$
$$P2 = \{\forall o \in D \mid d(o,p_2) \leq d(o,p_1)\} \tag{2.12}$$

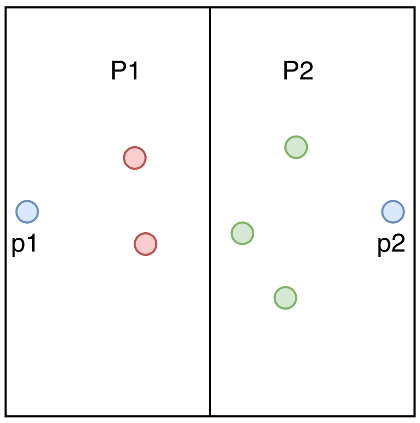

Figure 2.2: Generalized hyperplane partitioning.

As with Ball partitioning, for balancing purposes objects lying on diving hyperplane can belong to either partition.

Pivot selection Selection of pivots affects suitability of the partitioning for application in search algorithms. However for GHP, simple approach which often works reasonably well is to randomly select set of pivot candidates from our data collection and afterwards eliminate least promising ones based on some criteria such as partitioning split balance or distance of objects to dividing hyperplane.

2.4 Search index

Search index is a data structure that is used to organize the data collection in a way that enables efficient retrieval of results for similarity query. Metric search index is class of search indexes that can be utilized for any metric space.

Example of an index is *Vantage Point Tree*[7] which stores objects in leaves of a binary tree and applies ball partitioning in order to prune branches during search and avoid expensive distance computations. Containers of objects in indexes are often referred to as *buckets* and task of search index is to organize objects in these buckets in a way that will enable avoid examining buckets with objects that are not included in the query result.

2.5 Metric space transformations

In case computing distances is too computationally demanding we might transform original metric space to another more compact metric space, in order to speed up query evaluation at cost of some approximation error [8].

We define *metric space transformation* as a function $f\colon D_1 \to D_2$ which transforms original metric space $M_1 = (D_1, d_1)$ to another metric space $M_2 = (D_2, d_2)$. We compare objects in transformed metric space by distance function d_2 that approximates distances from original metric space:

$$\forall o_a, o_b \in D_1\colon d_1(o_a, o_b) \approx d_2(f(o_a), f(o_b)) \tag{2.13}$$

Special case of transformation functions are lower-bounding functions where distance function d_1 is lower-bound of function d_2:

$$\forall o_a, o_b \in D_1\colon d_1(o_a, o_b) \leq d_2(f(o_a), f(o_b)) \tag{2.14}$$

With lower-bounding property satisfied (2.14) range query result in transformed space is guaranteed to contain all objects within specified range from the original space.

3 Metric Sketches

In this chapter we introduce fundamental concepts of Metric Sketches.

3.1 Motivation

In many similarity searching applications data objects are represented as high-dimensional univariate vectors. Distance computation on such vectors involve many operations which significantly increases query execution run-time. Additionally high volume data collections cannot be stored in-memory, thus HDD or SSD drive read speed becomes the bottleneck. These problems motivated research in compact object representations including so called *sketches*, which can be used to decrease number of performed distance computations during evaluation of similarity search query. These compact representations are used in place of original objects to identify set promising query results usually however at cost of some approximation error. Even though sketches have been studied and researched mainly in context of vector space approximation [9], concept of *Metric sketch* is advantageously general for any metric space satisfying properties (2.1) - (2.5).

3.2 Definition

Metric sketch Metric sketches are compact binary representations of original database objects [10]. Each bit of these fixed-length binary strings is created by binary partitioning in the original metric space. Primary database objects are stored on disk while their sketches are being held in-memory and compared efficiently using hamming distance. To search for query results in the original space we employ *filter & refinement* [11] strategy. First in *filtering* phase we use sketches to restrict the search space by preselecting subset of candidate objects from the database. Afterwards during *refinement* procedure we conduct exact similarity search in the original space but only considering preselected objects. Applying this technique sketches were shown to significantly reduce number of objects to refine in the original space [12].

Formal definition of sketch Let $M = (D, d)$ be the original metric space, $p \in \mathbb{Z}^+$ desired bit-string length and $B_i\colon o \rightarrow \{0,1\}, \forall o \in D, \forall i = 1 \dots p$ binary partitioning functions defined upon M. Then we define *sketch metric space* $S_p = (D_s, d_s)$ as transformation of M using function:

$$t_p\colon o \rightarrow b^p, \forall o \in D,$$
$$b^p = [B_1(o), B_2(o), \dots, B_p(o)] \tag{3.1}$$

Sketch domain $D_s \subseteq \mathbb{Z}_2^p$ consists of binary strings b^p of length p which are compared by metric distance function d_s. S_p is approximation of M but lower-bounding property does not apply.

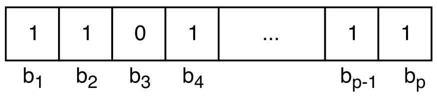

Figure 3.1: Sketch of length p.

3.3 Sketch properties

In this section we'll define 4 properties of sketches that we expect to affect on quality of approximation of the original space by sketches, namely partitioning balance, bit clarity, bit independence and sketch length. Our goal is to establish some notion of what values of these properties make a *"good"* sketch which will result in *low approximation error* during similarity searching. We'll refer to inverted value of approximation error as to **approximation quality**.

3.3.1 Partitioning balance

Each bit of sketch is determined by some binary partitioning in the original space. Single sketch is likely to represent multiple objects. In ideal case partitioning for each bit divides objects from the distribution uniformly, resulting to each sketch representing equal number of original objects from the database. In such perfect case filtering becomes perfectly stable - with each random query with same number of retrieved sketches the same amount of original objects will be retrieved for refinement procedure. In opposite extreme case all objects from the database might be represented by single sketch eliminating any filtering effect making sketch search useless. Thus for each bit of sketch we want the probability of object falling to partition 0 and 1 to be equal.

After we apply binary partitioning on all objects from non-empty data collection X we can use function b to evaluate partitioning balance property of corresponding bit:

$$b(c_1, |X|) = 1 - 2\left|0.5 - \frac{c_1}{|X|}\right| \tag{3.2}$$

where c_1 is the number of objects assigned to partition 1. Function b (3.2) assigns value 1 if partitioning is perfectly balanced and 0 otherwise. Further in this thesis we'll refer to function b (Figure 3.2) as to **balance score** of sketch bit. Balancing property of sketch can be evaluated as mean of balance scores of all it's corresponding bits.

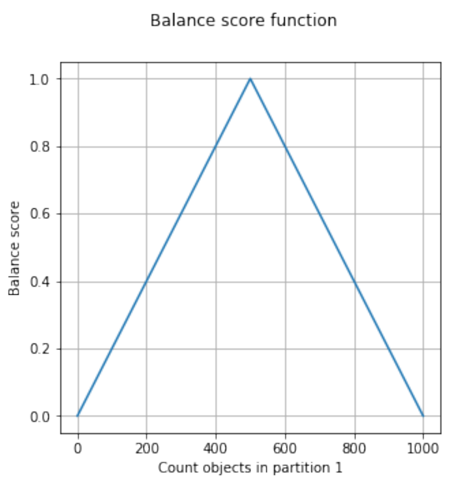

Figure 3.2: Balance score function.

Note that balancing property does not imply that bits are uncorrelated, hence it's not sufficient condition for claiming quality of sketches. We could represent whole large data collection with only 2 sketches of arbitrary length with all bits either 0 or 1 making all except one bit useless for filtering but still perfectly satisfy the balancing property.

3.3.2 Bit dependence

We can perceive each bit of sketch as an encoding of some useful information about object's position in the original space. If two bits of sketches are mutually dependent it means that they share some redundant information. To encode as much information as possible we need to ensure that bits are mutually independent. Linear dependence of bits can be evaluated as *Pearson correlation coefficient* [13]:

$$\rho(X,Y) = \frac{cov(X,Y)}{\sigma_X \sigma_Y} \tag{3.3}$$

In our case X and Y are binary vectors of length n, where each bit represents partition of database object $i, i = 1 \ldots n$. Note that Pearson correlation coefficient is undefined in case standard deviation of one of vectors is equal to 0, which would occur if balance score of bit was also equal to 0. However totally unbalanced bits don't encode any information, therefore we have no reason to even consider them. We will compute absolute values of pair-wise correlations because amount of redundancy of encoded information of two mutually dependent bits is the same whether the dependence is positive or negative.

To compute absolute pair-wise correlations matrix M for each possible bit couple $i, j, i \neq j$ we'll use binary matrix of sketches S, to which we'll further refer as **sketch matrix**. This matrix is created by transforming n database objects to sketches of length p and stacking them row-wise.

$$S = \begin{bmatrix} s_{(1,1)} & s_{(1,2)} & \cdots & s_{(1,p)} \\ s_{(2,1)} & s_{(2,2)} & \cdots & s_{(2,p)} \\ \vdots & \vdots & \vdots & \vdots \\ s_{(n,1)} & s_{(n,2)} & \cdots & s_{(n,p)} \end{bmatrix}$$

Let's denote i^{th} column of matrix S as z_i. We define **sketch correlation matrix** M as matrix containing absolute values of all pair-wise Pearson correlations of columns from S:

$$M = \begin{bmatrix} |\rho(z_1, z_1)| & |\rho(z_1, z_2)| & \dots & |\rho(z_1, z_p)| \\ |\rho(z_2, z_1)| & |\rho(z_2, z_2)| & \dots & |\rho(z_2, z_p)| \\ \vdots & \vdots & \vdots & \vdots \\ |\rho(z_p, z_1)| & |\rho(z_n, z_2)| & \dots & |\rho(z_p, z_p)| \end{bmatrix}$$

Elements $m_{i,j} \in [0, 1]$ of this matrix represent *absolute pair-wise bit correlations* of sketches - 0.0 meaning perfect independence and 1.0 maximal dependence.

Overall linear dependence of sketches can be interpreted as mean of values of elements from upper-triangle of matrix M above diagonal and we will further refer to this value as **mean bit correlation** of sketches.

3.3.3 Bit clarity

For special case of sketches created by generalized hyperplane partitioning the **bit clarity** is defined by distance of database objects from dividing hyperplane between two pivots. Let distance from object o to pivot A, d_A be greater or equal than distance to pivot B, d_B. Distance to dividing is then computed as:

$$\frac{d_A + d_B}{2} - d_A \tag{3.4}$$

If objects tend to fall further from dividing hyperplane, due to triangle inequality it's more likely that distances between objects from different partitions are larger, hence space division is more robust against noise in data and shifts in it's distribution. However it was shown in [14] that satisfying bit clarity unfortunately results in high bit dependence.

3.3.4 Sketch length

Choice of sketch length impacts both effectiveness and efficiency of applying metric sketch approach.

For each additional bit of sketch one more costly binary partitioning in the original space is required to compute compact representation of every object involved in the search process (see Section 3.4). Shorter sketches will furthermore result in lower memory consumption and distance computation time during the filtering phase and might effect efficiency of indexing structure used for reducing search time in sketch space (see Chapter 4).

Shorter sketch intuitively results in less information about position of object in original space being encoded, decreasing the approximation quality of sketches. To maximize approximation quality we construct sketches that satisfy bit balance and independence properties. However the longer the sketch, the more difficult it is to find suitable pivots for partitioning that would produce new bit balanced and independent of others.

To determine suitable sketch length V. Míč suggests in [15] formula (3.5) based on the *Intrinsic dimensionality*[15], which is the number of dimensions in which original objects can be embedded in the vector space while preserving distances among them [16].

$$iDim = \frac{p}{2(1 + (p-1)c^2)} \tag{3.5}$$

Choice of sketch length might be seen as bi-objective optimization problem where our goal is to minimize sketch length p and simultaneously maximize sketch intrinsic dimensionality *iDim*. If we started with $p = 1$ and added new bits one by one, at some point increase in sketch length wouldn't provide reasonable increase in *iDim* which is useful to determine when to stop adding new bits.

3.4 Sketch creation

We can expect 3 usual cases when object from original domain has to be transformed into it's in-memory compact sketch representation. Initially we need to construct object's sketch for every object in our database. Secondly we have to construct new sketch with each database object insert or update to keep sketch representation consistent with current state of stored data collection. Third case sketch is created is for every query object to search for it's nearest sketches to further refine in original metric space.

Each Sketch construction involves assignment of binary value to p bits, each computed using some partitioning of the original space in respect to transformed object. Assigning single bit will include computing distance to one or more pivots in the original space which can be significantly time consuming. However this process can take advantage of modern distributed computing systems by running multiple sketch creation procedures in parallel resulting in linear speed up.

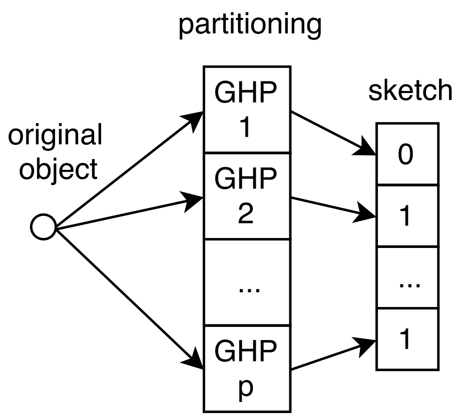

Figure 3.3: Object transformed to it's sketch representation using generalized hyperplane partitioning.

3.5 Sketch optimization

Sketch optimization is task where we are given a large data collection of original objects X along with metric distance function to compare them and desired sketch length p and our goal is to choose partitioning method and pivots, which will produce sketches that satisfy balance and independence properties for any objects from given data distribution. We will first draw subset of objects $L, L \subset X$ to find pivots for selected partitioning producing sketches with desired properties and then we will transform the whole data collection using selected subset of pivots.

3.5.1 Partitioning method

Important choice we need to make is choice of a binary partitioning method. Two most commonly used methods for partitioning Metric Spaces are *ball partitioning (BP)* and *generalized hyperplane partitioning (GHP)*. Both of them have been experimentally evaluated in context of sketches in [14] with results showing GHP to be more suitable for producing of sketch bits than BP with conclusion that ball overlapping causes increased bit dependency. On the other hand partition in BP is determined by computing distance to only one pivot instead of two pivots in case of GHP. For creation of sketch of length p we'll need to compute p distances in case of BP and $2p$ distances in case of GHP which results in doubling the sketch creation time with GHP possibly being significant difference when run-time available for single transformation operation is very limited. For clarity and simplicity further in this thesis we won't consider any other pivot selection method except GHP.

3.5.2 Pivot couple selection

Now we have to look for pivot couples for GHP that would produce sketches with balanced and mutually independent bits. We can first draw $t, t \gg p$ pivot couples from L either randomly or using some pivot selection method such as *Sparse Spatial Selection*[17]. Then we'll use these pivots to transform all objects from L to their sketch representations of length t and stack them row-wise creating $|L| \times t$ binary sketch matrix S_t.

Each pivot couple was used to produce single column in S_t. By finding p-subset of columns from S_t with maximized mean balance score and minimized mean bit correlation we'll select suitable pivot couples for constructing sketches of the whole data collection X, newly inserted/updated objects and query objects. Therefore, the task is to reduce matrix S_t to $|L| \times p$ sub-matrix S_p while optimizing for maximal mean balance score and minimal mean bit correlation. To find an exact solution one would have to examine all $\binom{t}{p}$ possible subsets of columns and evaluate their properties. Additionally these two objectives can be contrary and it's not guaranteed that subset of bits with best balance score will also be the most independent.

3.5.3 Computational complexity of bit selection

Balance score of all bits can be computed using formula (3.2) during single iteration over rows of S_t, resulting in $O(t|L|)$ computational complexity [18]. Since balance score computed on one column of S_t is independent of other we can compute the score only once and then evaluate mean balance score of some subset of columns from S_t simple by computing mean value of precomputed balance scores for included columns.

To evaluate mean bit correlation we first need to compute $t \times t$ sketch correlation matrix M and then mean of values in it's upper triangle above diagonal. In order to construct M, for each couple of columns of S_t we evaluate Pearson correlation coefficient on vectors of length $|L|$ and take absolute values. Computation of absolute Pearson correlation coefficient is linear to size of compared vectors which leaves us with complexity of sketch correlation matrix creation equal to $O(t^2|L|)$. Same as balance scores absolute pair-wise correlations from M can be memorized.

With $O(1)$ access to precomputed balance scores and pair-wise absolute correlations properties evaluation of properties of any selected p-bit subset will require only computation of mean balance ($O(p)$) and mean of absolute bit couple Pearson correlations ($O(p^2)$). Thus complexity of finding optimal p-subset of t pivot couples which will using data sample of size $|L|$ is

$$O(t^2|L|)$$

for initial procedure and

$$O(\binom{t}{p}p^2)$$

for examining all possible solutions using memorized precomputed properties of bits.

3.5.4 Heuristics for bit selection

Description Depending on number of pivot couples t, desired Sketch length p and data sample size $|L|$ brute force approach for finding of optimal p-subset of pivot tuples might be impractical. Moreover, it might not be clear which solution is the optimal one and when prioritize correlation and when balance since these two properties can be contrary. Instead of solving bi-objective optimization problem we can relax it by setting a lower-bound constraint on balance score of individual columns of S_t and optimize only for the bit independence criteria.

Bit balance property of each bit is independent of other bits but how some selected bit contributes to mean bit correlation depends on all other bits in the selected set, therefore finding a large set of well balanced bits is easier problem to solve than finding independent bits. To select independent subset of bits we present two heuristic algorithms - **Fast minimal correlation columns algorithm (FMCC)** and **Greedy minimal correlation columns algorithm (GMCC)**. Input to both algorithms is vector of pivot couple candidates P, $|L| \times t$ sketch matrix S, desired Sketch length p and lower bound for bit balance l. Both algorithms first compute balance scores, remove pivot couples that produced bits with balance score lower than l.

FMCC then computes sketch correlation matrix M on remaining columns of S and selects p pivot couples with lowest mean of values in related column of M.

GMCC on the other hand starts with all t pivot couples and removes them along with columns of S iteratively one by one. In each iteration for each remaining pivot couple we compute value of mean sketch correlation after removal of bit produced by this couple from sketches. At the end of each iteration we remove the pivot couple which one's removal will result in largest decrease of mean bit correlation.

Algorithms Pesudocode for algorithms *FMCC*, *GMCC* and their initial procedures *RemoveUnbalanced* for removing bits with balance score below lower bound and *CorrMtx* for computing sketch correlation matrix are attached in *appendix A*.

Algorithms - computational complexity In this paragraph we'll analyze computational complexity of FMCC and GMCC algorithms and their initial procedures from *appendix A*.

- **RemoveUnbalaced** Procedure performs only one linear scan through S which includes examining $t \times |L|$ items and then examining t results:

$$\Rightarrow O(t|L|)$$

- **CorrMtx** During construction of correlation matrix we compute Pearson coefficient with linear complexity $O(|L|)$ for each couple of t columns of S

  $$\Rightarrow O(t^2|L|)$$

  . Since complexity of removing unbalanced bits is $O(t|L|) < O(t^2|L|)$ the complexity of initial procedure including both removing unbalanced bits and computing correlation matrix is also $O(t^2|L|)$.

- **FMCC** Algorithm starts with $O(t^2|L|)$ initial procedure and then computes one column-wise sum of $t \times t$ sketch correlation matrix M which includes examining t^2 values

  $$\Rightarrow O(t^2|L|)$$

  .

- **GMCC** After $O(t^2|L|)$ initial procedure there is $t - p$ iterations in which we remove one column of S and pivot tuple. In each i-th iteration we compute mean of upper triangle of M excluding removed column which we can for simplicity count as examining $\frac{1}{2}(t-i)^2$ items and this is done for each of $(t-i)$ remaining columns of S. Thus we can compute number of examined values as

  $$\frac{1}{2}\sum_{i=0}^{t-p}(t-i)^3 < \sum_{i=0}^{t}(t)^3$$

  $\Rightarrow O(t^4)$. Complexity of $GMCC$ algorithm is then

  $$O(t^4 + t^2|L|)$$

  .

3.6 Application of Sketches

After we found eligible pivot couples and transformed our data collection X to it's in-memory sketch representation K we can utilize these sketches to speed up $k - NN$ similarity queries on original data. We're looking some portion of nearest neighbors of query object's sketch q_s to further refine in original metric space.

3.6.1 Distance computation in sketch space

In order to utilize sketches for similarity searching we need to define a distance function for comparing any arbitrary couple of objects from sketch space.

The efficient way to compare two sketches is to compute their *hamming distance* [19], which is defined as sum of bit-wise differences of two binary strings g and h:

$$d_H(h,g) = \sum_{i=1}^{p} |h_i - g_i|$$

For fast implementation we can use efficient *population count* (*POPCNT*) [20] hardware instruction which is included in x64 instruction set available on most current AMD and Intel CPU's.

Another proposed way of comparing sketches is to use *asymmetric comparison* [21] which improves quality of approximation however at cost of increased distance computation time, thus in this thesis we will employ only more efficient hamming distance.

3.6.2 Filter & refine

Filter & refinement procedure aims to reduce portion of database to be inspected in order to retrieve similarity query result. It starts with *filtering* phase which takes place in the sketch metric space. We identify $k - NN$ candidate subset C, $C \subset K$, $k \ll |C| \ll |K|$ containing objects which are the closest to sketch q_s of the query object q. Then during the *refinement* phase we will examine all objects from C in original metric space and identify k closest neighbors of q. Important parameter of this 2-phase process is choice of candidate set size $|C|$. Smaller size results in less objects needed to be examined in the original space where comparison of 2 objects is computationally expensive. However as a trade-off smaller candidate set size also decreases the approximation quality.

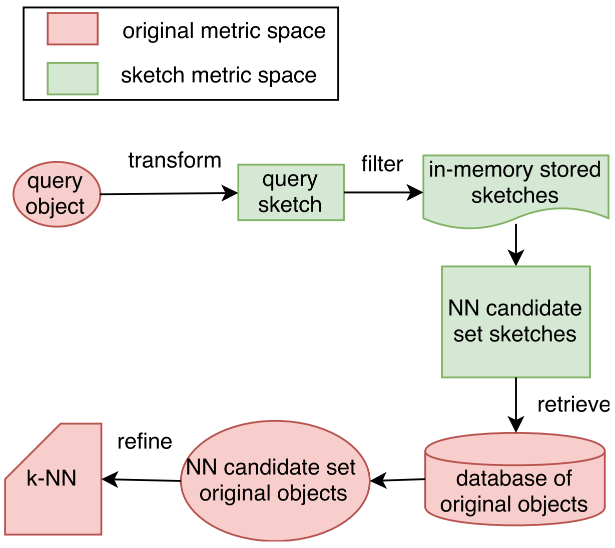
original metric space
sketch metric space
query object
transform
query sketch
filter
in-memory stored sketches
NN candidate set sketches
retrieve
database of original objects
NN candidate set original objects
refine
k-NN

Figure 3.4: Applying filter & refine procedure for *k-NN* search using sketches.

3.6.3 Objective & evaluation

We can say that good quality sketches preserve distance ordering of objects from original space in respect to any arbitrary query object. Therefore, some measure of distance ordering preservation might be general indicator of quality of sketches. However in practical scenario where we need to retrieve k nearest neighbors we might not care about ordering of results as strictly. The criteria of good approximation can be specified as measure what portion of true k nearest neighbors are actually contained in retrieved candidate set C results after filtering procedure. Let $R*$ be a set true k nearest neighbors of query object q in data collection X and C be the candidate set retrieved for refinement after filtering phase in sketch space. To measure of approximation quality the we will use *recall*[1] from information theory:

$$recall(C, R^*) = \frac{|C \cap R^*|}{|R^*|} \tag{3.6}$$

Since $|R^*|$ is in case of *k-NN* query always equal to *k*:

$$recall(C, R^*) = \frac{|C \cap R^*|}{k} \tag{3.7}$$

Values of $recall(C, R^*) \in [0, 1]$, thus 1.0 means no approximation error and 0.0 otherwise. For better interpretation of results we will furthermore denote $100 \times recall(C, R^*)$ as $\%recall$, the portion of true nearest neighbors included in the candidate set C.

We will further denote (3.7) as *recall*, $C \cap R^*$ as *true positives* and $C - R^*$ as *false positives*.

4 Multi-hash Index

In this chapter we'll introduce indexing structure for evaluating similarity queries in sketch metric space called *Multi-hash index*.

4.1 Motivation

After we design sketches optimized for approximating distances between objects in our data collection we'll store them in-memory. With each nearest neighbor query we first initiate a look up in memory among sketches and identify portion of closest objects which will be then examined in the original metric space. One possibility to search the sketch space is to perform a sequential scan which will iterate over all sketches. In order to speed up the search we may organize sketches in a search index in order to reduce number of hamming distance computations needed. Since sketch space is a valid metric space indexes designed for general metric spaces such as tree based *M-Tree* [22] or multi-tier *D-Index* [23] can be used for sketches as well. However restricting the domain to fixed-length binary string encodings enables us to step out from general metric space and focus on some specifics of searching among sketches which could speed up the searching process even more. One such index developed for searching specifically in hamming spaces is *Multi-hash index* [24].

4.2 Hash-Index

Multi-hash index is based on idea of exploiting hash tables for indexing. We will refer to hash table used for indexing as to *Hash index*. In Hash index we use binary strings as addresses for single hash table, thus any possible sketch of length p is used as an address to access single bucket in the index. If objects in data collection are identified by some unique reference ID or pointer to the original object we put it in the relevant bucket. This is very convenient for exact matching and searching for range query results with very small range (i.e. exact match or $r = 1$) which is feasible for some search applications. When given query sketch we can directly access bucket with references to identical sketches. However if we conducted a range search with range $r > 0$ we would have to generate all possible binary codes in hamming distance up to r and examine $\sum_{k=0}^{r} \binom{p}{k}$ of all 2^p possible buckets from which most of are likely to be empty ($2^p \gg n$, where n is number of objects in our data collection). Therefore, with increasing sketch length p and search range r usage of Hash index becomes impractical.

4.3 The one substring rule

Basic idea exploited in Multi-hash index is to first split all sketches in our database and the query sketch into m equal [1] substrings $h^{(1)} \dots h^{(m)}$ (Figure **??**), each of size $\frac{b}{m}$. If two binary strings h and g differ in at most r bits then there must exist at least one substring $i = 1 \dots m$ in which they differ in at most $\left\lfloor \frac{r}{m} \right\rfloor$ bits:

$$\sum_{j=1}^{p} |h_j^{(i)} - g_j^{(i)}| \leq \left\lfloor \frac{r}{m} \right\rfloor \tag{4.1}$$

If hamming distance between each of their substrings was $\left\lfloor \frac{r}{m} \right\rfloor + 1$, then hamming distance of both strings would be $m(\left\lfloor \frac{r}{m} \right\rfloor + 1) > r$ which is a contradiction, hence the proposition must hold.

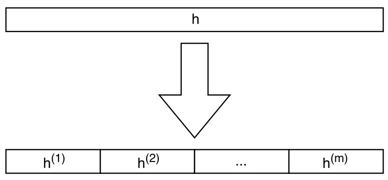

Figure 4.1: Splitting sketch h into m equally long substrings.

1. For the sake of simplicity let's consider only cases when split to equally long substrings is possible, however these principles apply in any case.

4.4 Multi-hash index

Note that in eq. (4.1) if $r < m$ then two binary strings must be equal in at least one of their substrings in order for them to be at most r far away. We can exploit this property to utilize $O(1)$ bucket access of the Hash index. We'll create m Hash indexes - each for one of m substrings. We split each sketch in our database into m equal substrings and use every substring $h^{(i)}$ as an address to a bucket in Hash index I_i. Buckets of these indexes will contain references to original sketches. Reference to a single sketch will be then present once in each of m different Hash indexes. Further in this thesis we will refer to this indexing structure as to **Multi-hash index** abbr. *MHI*. Buckets of this index placed in any of m hash indexes are sets with union and intersection operations defined upon them. We will furthermore call intersection of buckets *bucket overlap*.

4.5 Range search in MHI

Buckets of MHI can be looked upon as sets of sketch references. Search upon indexed data collection is conducted in following way (Figure 4.2):

1. Split query sketch q into m equal substrings $q^{(1)} \dots q^{(m)}$.

2. Look up buckets $C_1, C_2 \dots C_m$, where C_i is bucket in Hash index I_i under address of substring $q^{(i)}$.

3. Make union $C = C_1 \cup C_2 \cup \dots \cup C_m$ of these buckets. We'll further denote C as the *candidate set*.

Because of the *one substring rule* we are guaranteed that the candidate set contains references to all sketches within hamming distance of $m - 1$ from q, i.e. all true positives of range query with range $r = m - 1$ along with some false positives. We then use sequential scan on original sketch objects which were referred to by objects from C to sort out false positives and retrieve $m - 1$ range query result sketches.

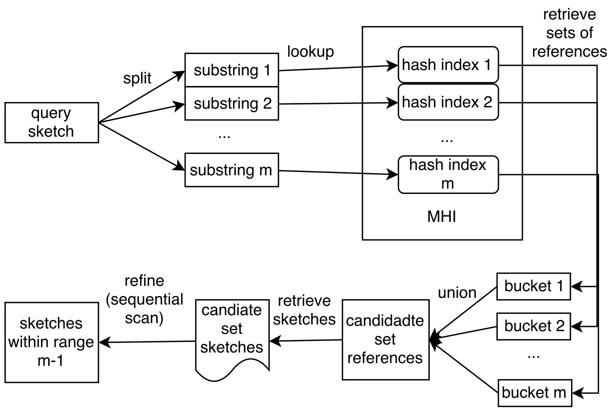

Figure 4.2: Searching for range query results in Multi-hash index.

4.6 Speeding up sketch searching with MHI

We want to conduct a nearest neighbor query on sketches to retrieve some closest portion of the sketch database. MHI will always return all sketches within range $r = m - 1$ from the query sketch, hence we are not guaranteed the number of objects in the result. By analyzing the sketch distance distribution we can find suitable range which will on average result in desired number of true positives contained in the candidate set. However candidate set also contains false positives which will have to be sorted out by sequential scan. Therefore, we need to pay attention to two important factors:

- How many true positives are within selected range $r = m - 1$. We need to set the range high enough that it's likely for the candidate set to contain desired number of true positives. Lowering this requirement will result in less closest sketches identified which will negatively affect recall after refinement in the original metric space.
- What is the cardinality of the candidate set. Candidate set is searched by sequential scan to sort out false positives. Contribution of MHI to speeding up similarity search can be viewed as a difference between number of all sketches in database (which we would searched through in simple sequential scan) versus cardi-

nality of the candidate set (which will be searched by sequential scan in MHI).

Number of objects in candidate set is affected by data distribution, number of sketch references in particular buckets and by how much do these buckets overlap. These aspects can be controlled by changing two parameters:

- Number of splits m
- Sketch length p

Smaller m will result in following:

- Substrings are longer.
- Probability of references to two random Sketches having the same substring (falling into the same bucket) is smaller.
- There are more possible bucket addresses for single *Hash index*.
- Search range is smaller.
- Number of visited buckets while conducting search is smaller.
- Mean number of sketch references placed in single bucket is smaller.
- Candidate set C is contains less sketches.
- Search time is shorter.

It's clear that we would like to set m as small as possible so we visit less buckets with smaller overlap and smaller number of sketch references stored which will result in decreased size of the candidate set to sort out. However it depends on the sketch distance distribution whether there is likely to be sufficient amount of object within range that is limited by small m. Longer sketches will require larger range to be searched which will result in need to set m higher which will result to larger overlap among buckets. Therefore, MHI is more suitable for searching among smaller sketches where distance distribution is focused around small distance.

5 Evaluation

Since notion of general metric space is very broad and and often there are no general mathematical formulas and proofs on how to optimize parameters of our search techniques and indexes for efficient similarity search and also desirable values of these parameters often heavily depend on distance and data distribution, it is very common to evaluate hypotheses on experimental datasets.

In this chapter we will design, implement and discuss series of experiments to verify our claims and expectations on real-life dataset with intention to better understand relationships, causes and effects regarding sketch creation, approximation quality of sketches for *k-NN* search and contribution of organizing sketches in Multi-hash index.

5.1 Experiments implementation

Experiments are implemented in *Python 3*[1] which is popular high-level interpreted general-purpose programming language. Main reasons for usage of Python in comparison to two most popular compiled languages *Java* and *C++* are it's interactivity, succinct syntax and good availability of wide range of popular scientific libraries also referred to as packages. The main disadvantage of Python as of other interpreted languages is it's inferior performance however it's often being boosted up to some level by implementation of some underlying logic in *C* encapsulated in Python interface. Conventions for Python programming are covered in detail in so called *PEP-8* guide. Python applications and libraries are installed with a package manager such as *PIP*.

5.1.1 Python packages used for experiments

In this section we'll describe some of Python tools used in our experiments.

1. Please note that code valid in Python 3.x.x is not necessarily compatible with Python 2.x.x. Our code is implemented and tested only in Python 3.5.2 and is guaranteed to be compatible with all Python 3 versions.

NumPy NumPy is scientific Python package that implements operations on data of homogeneous type organized in multi-dimension arrays called *ndarray*. This extension of Python allows it to handle numerical data in a way similar to *MATLAB*, *IDL*, *Octave* etc. For instance in our experiments we make use of random numbers generation, array and matrix manipulation operations and mathematical functions such as mean and standard deviation. Important argument for choosing to operate upon NumPy arrays over working with standard Python lists is that these arrays and operations are implemented at core in efficient *C* language which brings significant performance boost.

SciPy This scientific Python package is build upon NumPy and implements functionality for solving problems from fields like numerical analysis, statistics, optimization, linear algebra and many more. In our experiments we mostly make use of statistical and distance computation functions. NumPy and SciPy are two most fundamental components of Python's scientific stack.

Jupyter Jupyter allows users to create interactive documents called *notebooks* which contain code, visualizations and text and can be shared in multiple formats including *.html*. Code is executed in separated blocks rather than scripts which allows to change parameters and implementation of some parts of the program and execute them real-time while rest of the script is still unchanged in memory with all variables and environment preserved. Jupyter notebooks are commonly used for data exploration, analysis and visualization, data cleaning, numerical simulations, testing statistical hypotheses and other tasks requiring interactive data manipulation.

Pandas Pandas is Python package used for manipulating and organizing structured data into tables called *DataFrames* consisting of columns called *Series* with useful operations defined upon these two data types. It's the most popular package used for data exploration and analysis with API covering functionality such as data selection, computing statistics and visualization. This package is built to be effectively used in interactive scenarios with Jupyter notebooks.

Matplotlib Matplotlib is a visualization library implementing 2-D plotting functionality. It enables creating plots, scatter plots, histograms, bar charts and other common types of visualization and to make them available in multiple formats including embedded into a Jupyter notebooks. Interface is similar to the *MATLAB* visualization API.

Reference links

- NumPy: https://docs.scipy.org/doc/numpy-1.13.0/reference/
- SciPy: https://docs.scipy.org/doc/scipy/reference/index.html
- Jupyter: http://jupyter.org/documentation.html
- Pandas: https://pandas.pydata.org/
- Matplotlib: https://matplotlib.org/

5.1.2 Contents of the electronic attachment

This subsection describes software that is included in form of electronic attachment with this thesis in the *information system* of Masaryk University. Implementation of experiments consists of Python package and Jupyter notebooks.

Python package Implemented Python package *similarity-searching-sketches* contains used classes and methods are available in uploaded attachment and on GitHub[2] repository. All implementations are placed in folder *similarity-searching-sketches/similarity_searching_sketches/* in scripts with *.py* suffix. Package is contains valid *setup.py* script and is installable by *PIP* package manager.

Experiment Jupyter notebooks Implementations from Python package are imported and used in 5 Jupyter notebooks with *.ipynb* suffix

2. GitHub repository can be found at https://github.com/matejkvassay/similarity-searching-sketches

placed in *similarity-searching-sketches/experiments* directory. These notebooks are numbered from 1 to 5 and contain all experiments we conducted along with results and output of run of the program. These experiments will be discussed in the rest of this chapter. All notebooks are also exported to *.html* format also containing all code and results, placed in directory *experiments_export*. Additionally notebooks can be viewed in web browser on GitHub repository[3].

List of Jupyter notebooks :

1. *1_dataset_analysis*: Data analysis and train/test split.
2. *2_pivot_selection_balance_score*: Pivot selection by balance score.
3. *3_pivot_selection_bit_correlation*: Pivot selection by mean absolute Pearson correlation.
4. *4_approximation_evaluation_and_train_vs_test_set*: Evaluation of approximation quality and comparison of train and test set.
5. *5_multi_hash_index*: Evaluation of Multi-hash index.

5.2 Dataset

This section describes dataset we used for our experiments.

5.2.1 Profiset collection

In many typical similarity searching applications database objects are represented by univariate feature vectors. In case of images, features extracted from hidden layer of *Convolutional Neural Networks (CNN)* [25] were shown to achieve state-of-the-art results on multiple tasks including image similarity search capturing both visual and semantic properties of images.

In our experiments evaluated our hypotheses on image vectors extracted from so called *Profiset collection* using *DeCaf CNN* [26]. This

3. Notebooks containing experiments can be found at the GitHub repository at URL https://github.com/matejkvassay/similarity-searching-sketches/blob/master/experiments/

data collection provided by *Laboratory of Data Intensive Systems and Applications (DISA)* contains over 20 million 4,096-dimensional feature vectors, each representing one of original images. For better idea of what kind of images are present in the Profiset collection, DISA laboratory provides publicly available demo[4] of their similarity search application.

5.2.2 Used data sample

For purposes our experiments we chose to compare vectors by Manhattan distance and to draw a random sample of size 10,000 vectors from Profiset collection DeCaf feature vectors. We also split this sample into 8000 vectors we refer to as the *training set* and 2,000 vectors we refer to as the *test set*.

Train/test split Splitting data into training and test set is common practice used in field of statistical modeling. We optimize parameters of some mathematical model in respect to some optimization objective using only the train set keeping the test set aside. After the optimization is finished in order to assess ability of this model to generalize we evaluate quality of the model in respect to the objective on remaining test set.

In our scenario we aim to optimize the sketch creation model by choosing feasible pivot couple subsets from superset of all possible pivot couples from the data sample. Our first optimization objective is to achieve desirable values of sketch properties, namely to maximize the mean balance score and minimize the mean bit correlation. Satisfying this first optimization objective we expect will lead to satisfying the main objective we set for ourselves which is to maximize approximation quality evaluated by recall on results of *k-NN* queries on our data sample.

To find out whether our sketch creation model optimized for the train set also generalizes well for other data from the same distribution we evaluated these two optimization criteria on the separate 2,000 vectors test set.

4. The DISA Similarity Search application demo can be found at http://disa.fi.muni.cz/demos/profiset-decaf/random

Choice of distance function Feature vectors are often being compared by *Minkowski* [27] distances, most commonly Manhattan (L1 distance) and Euclidean (L2 distance) [28]. To get a better idea of distance distribution in our datasets we made it's estimation (Figure 5.1) for both L1 and L2 distances on the train set by drawing random 100 vectors as reference objects and computing distances from them to all 7,900 remaining vectors.

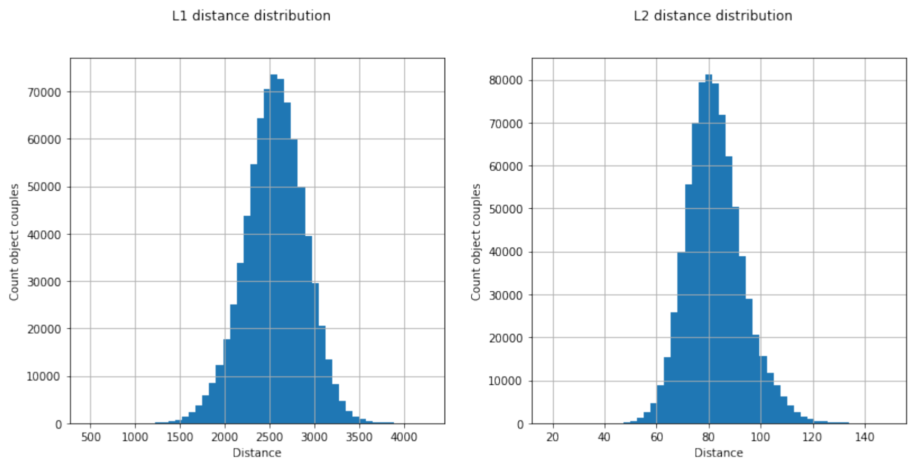

Figure 5.1: L1 and L2 distance distribution estimation

It was shown that due to the so called *distance concatenation* [29] (also referred to as the *curse of dimensionality*[30]) phenomena in high-dimensional vector spaces L1 distance is consistently more preferable for Nearest Neighbor search than higher order Minkowski distances [31]. Although we don't provide valid proof of this claim applying specifically to our high dimensional dataset we had no reason to not follow this suggestion and we have chosen Manhattan distance as a measure to compare DeCaf vectors.

Justification for the sample size Data sample of 10,000 vectors randomly drawn of 20,000,000 might not be a sufficient portion to well represent variety of the original data distribution, however we didn't aim to optimize for the entire data collection but rather to evaluate our claims and hypotheses regarding sketches and Multi-hash index on some real-life data sample. We didn't observe any irregularities in distance distribution and as we'll show later in this chapter properties on sketches optimized on the train set hold also for the test set which is unlikely to happen if distributions of train and test set were too different.

5.3 Sketch creation

While designing sketches optimized for our train set we were looking for set of 128 pivot couples that would be used for generalized hyperplane partitioning to produce sketches with maximized mean balance score and minimized mean bit correlation. First we have chosen 1,000 random pivot couples and used them to create a sketch matrix S, where one row contains 1 sketch of length 1,000 representing one of objects from the train set.

5.3.1 Optimizing sketches for maximal mean balance score

First step in designing sketches was to choose pivots with best *balance score*.

Balance score analysis For each column of S we computed the balance score (3.2) ranging from zero to one, one being the best balanced and zero otherwise. Then we ordered these scores in descending order and visualized them along with their histogram. We can observe on Figure 5.2 that decrease in the balance score in order is linear except for very low balance scored bits which have slightly higher probability to be generated by random pivot couple.

Best 200 balanced bits had balance score ≥ 0.8 which means that in the worst case 40% of all objects fell into one partition and 60% to the other.

We conclude from the visualization that if we set balance score lower bound at some level l and then increased number of randomly drawn pivot couples linearly we would then expect the number of pivot couples generating bits with balance score $\geq l$ to also increase linearly. We claim that difficulty of randomly finding new pivot couples producing sketch bits with balance score above some specific

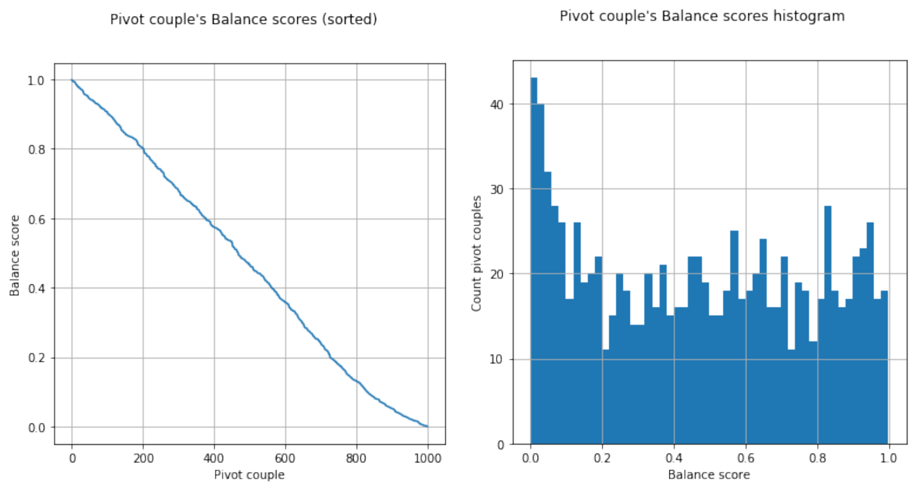

Figure 5.2: Balance Score of all 1,000 bits (columns of S) in descending order & histogram.

lower bound is linear in respect to number of pivot couples already picked if the data sample is sufficiently large [5].

Dependence of balance score on bit clarity For each of 1,000 pivot couples we computed mean distance from all objects from train set to the dividing hyperplane (3.4) to find out whether there's a relationship between the balance score and the bit clarity. On Figure 5.3 we visualized mean distances to the dividing hyperplane sorted by the balance score. We also computed non-parametric *Spearman* correlation coefficient with result -0.59 and p-value smaller than 0.05 which on confidence interval 95% proves moderate negative linear dependence of mean distance of objects to the dividing hyperplane on the balance score. However visualization suggests there's also a non-linear negative dependence with apparent exponential growth of the distance to the dividing hyperplane with higher slope in cases when bits are highly unbalanced.

5. Probability of picking a pivot couple is also affected by number of objects in remaining sample set. If the sample set is large enough removing two new objects from the pool for random selection results in only some minor and insignificant change in probability of drawing any specific sample

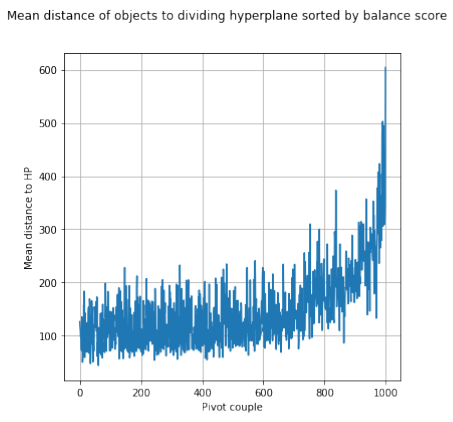

Figure 5.3: Mean distance from dividing hyperplane sorted by balance score in descending order.

5.3.2 Optimizing sketches for minimal mean bit correlation

After we chose 1,000 random pivot and evaluated their *balance scores* we removed 500 most unbalanced pivot couples and associated columns and rows in the sketch matrix S, leaving us with other 500 pivot couples having balance score lower-bound close to 0.5, meaning that bit balance for each column is in worst case 25% in one partition and 75% in the other.

In the second step of designing sketches we analyzed absolute pair-wise Pearson correlations of bits and used heuristic algorithms to choose subset of 128 bits with the lowest mean bit correlation.

Pairwise absolute Pearson correlations We computed sketch correlation matrix containing all absolute pair-wise Pearson correlations and took values from upper triangle of this matrix above the diagonal to get a view on values of correlation between bits themselves (Figure 5.4). We can see that distribution for smaller correlations appears similar to uniform however close to the median distribution changes to rather steep linear decrease. Mean value of correlations from the upper triangle is 0.3.

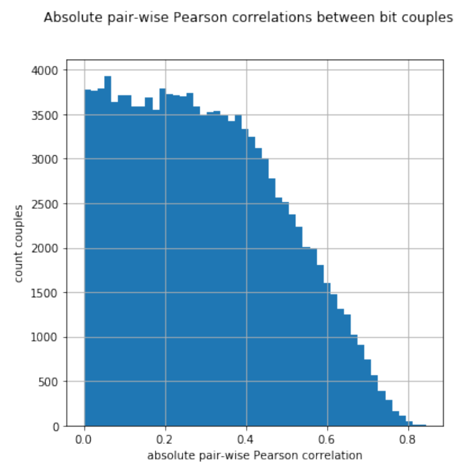

Figure 5.4: Histogram of pair-wise absolute Pearson bit correlations.

One vs. all correlations Mean absolute Pearson correlation of one bit against all remaining 499 bits carries useful information about how dependent on other bits this specific bit is overall and provides some sort of ranking and evaluation of bits even without evaluating correlations for all possible p-bit subsets. We computed these one-vs-all correlations and analyzed their dependence on the balance score and on distance of objects do dividing hyperplane (Figure 5.5). We sorted one-vs-all correlations by both balance score and distances and we observed slightly noticeable dependence on the balance score and evident dependence on mean distance from the dividing hyperplane.

To properly evaluate these dependencies we conducted statistical test computing Spearman correlation with both results having p-value below 0.05, thus on 95% confidence interval we proved following linear dependencies:

- Correlation 0.24 between one-vs-all absolute Pearson correlation and the balance score.
- Correlation 0.79 between one-vs-all absolute Pearson correlation and the mean distance of objects from dividing hyperplane.

This confirms hypothesis from [14] that optimizing Sketches creation model for small correlations is contrary to optimizing for larger dis-

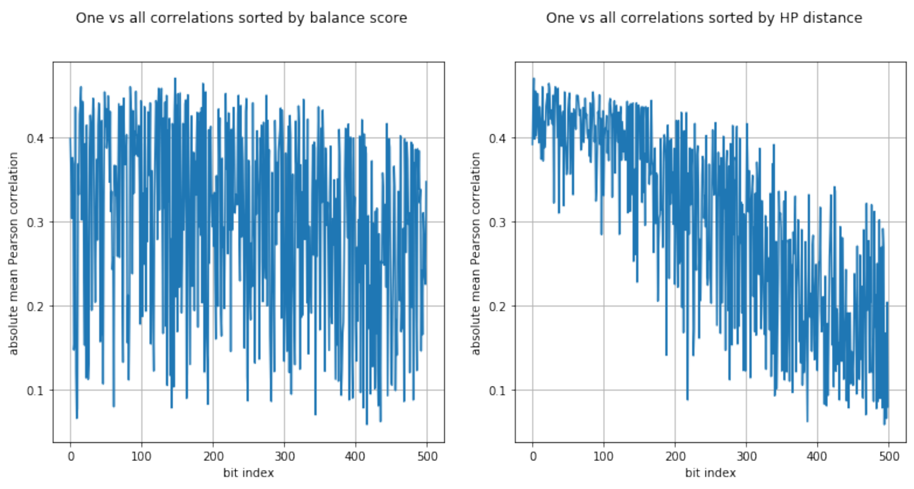

Figure 5.5: One-vs-all correlations of bits sorted by balance score and distance to dividing hyperplane.

tances of objects from dividing hyperplane. There is also small positive linear dependence of one-vs-all bit correlations on the balance score.

Uncorrelated bit subset selection algorithms evaluation Along with FMCC and GMCC algorithms we implemented one additional *baseline* algorithm which simply selects 100 p-bit subsets randomly, computes their mean bit correlation and chooses the least correlated subset. We applied these algorithms on remaining 500 columns of the sketch matrix for sketch lengths p between 2 and 128 and visualized resulting mean bit correlations (Figure 5.6).

On the visualization we can observe that GMCC provides consistent results for increased sketch length. Discrepancies in multiple runs of FMCC confirms that adding bit with the lowest sum of one-vs-all correlation doesn't imply whether will mean bit correlation increases or decreases. We conclude that advantageously each new bit added by GMCC results in subset with equal or greater mean bit correlation hence bits are ordered by correlations and shortening of sketches is guaranteed not to result in decline of the bit independence property.

Resulting correlations after we selected 128-bit subset can be found in Table 5.1. Note that mean correlation in case we would select all

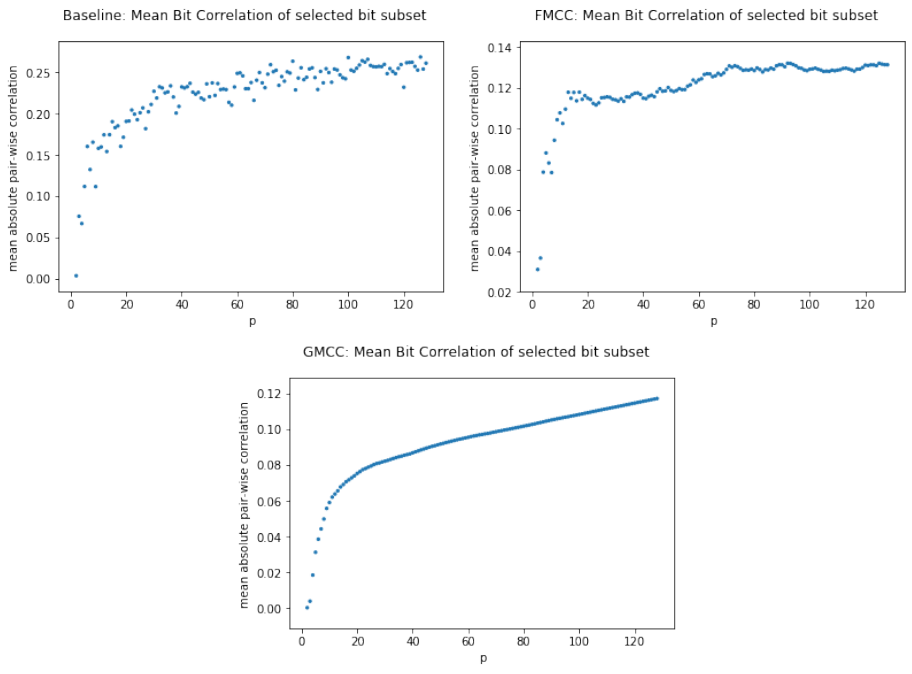

Figure 5.6: Mean bit correlations of selected p-bit subsets for baseline algorithm, GMCC and FMCC.

Baseline	FMCC	GMCC
0.261	0.131	0.117

Table 5.1: Mean bit correlations for selected 128-bit subsets.

500 bits is 0.3. Compared to the baseline approach FMCC performed approximately 2 times better while GMCC outperformed FMCC by a small margin. We selected the subset of 128 pivot couples returned by GMCC algorithm which resulted in generating Sketches from the train set with mean bit correlation 0.117 and mean balance score 0.69.

5.3.3 Sketch properties train vs. test set

We used chosen 128 pivot couples to transform both train and test sets to sketch metric space. We then measured how much these sets differ in mean balance score and mean bit correlation. Table 5.2 shows that differences between both sets are very minor. We conclude that sketches optimized for our train set are also well optimized for the test set in respect to mean balance score and mean bit correlation properties.

p	Balance score train	Corr train	Balance score test	Corr test	Balance score diff	Corr diff
2	0.714875	0.000094	0.705500	0.035173	-0.009375	0.035080
4	0.613812	0.018349	0.613250	0.025895	-0.000562	0.007546
8	0.662219	0.049696	0.654750	0.054369	-0.007469	0.004673
16	0.644188	0.070557	0.642062	0.075108	-0.002125	0.004551
32	0.645672	0.083050	0.644313	0.085052	-0.001359	0.002002
64	0.666715	0.096758	0.664281	0.098420	-0.002434	0.001662
128	0.688744	0.117108	0.687281	0.118643	-0.001463	0.001535

Table 5.2: Mean balance score and mean bit correlation differences train vs. test set.

5.4 Nearest neighbor approximation evaluation

To evaluate sketch approximation quality we have picked 100 random query objects and identified their 100 nearest neighbors in the original DeCaf vector space as our set of true positives. Then we sorted all objects in the sketch space according to their distances from sketches of query objects and selected specific percentage of the closest objects to include in the result set. Result set and true positives were afterwards used to compute % recall (aggregated by mean over all 100 query objects). This was done for multiple settings of sketch length p, percentage of the dataset to return and both for train and test sets separately.

Results for train set are showed in Table 5.3, for test set in Table 5.4 and visualized for both data sets on Figure 5.7. We can see that for instance in train set for sketches of length 128 we can find 87% of all 100 true nearest neighbors among the closest 10% objects in the sketch space. The approximation quality of sketch as we expected increases with sketch length since sketch encodes more information about position of the object in the original space.

Table 5.5 and figure 5.8 show differences between approximation quality on train and test set. Although two optimized sketch properties

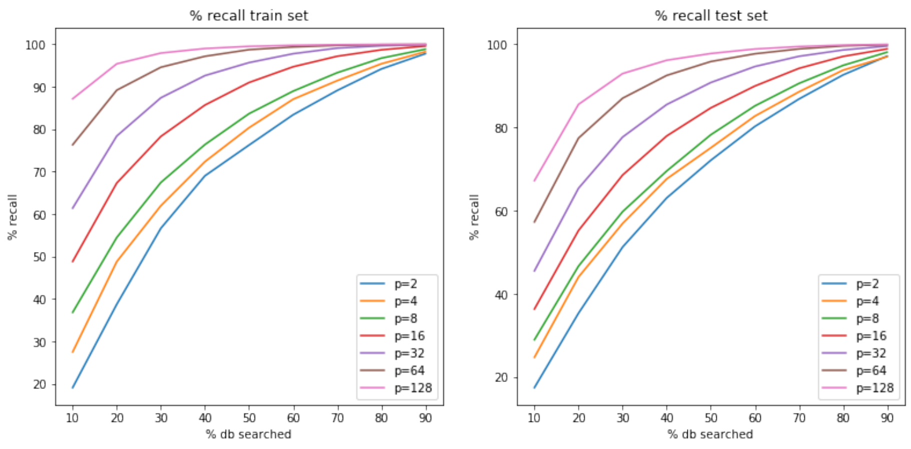

Figure 5.7: Mean % recall train (left) and test set.

% db searched	p=2	p=4	p=8	p=16	p=32	p=64	p=128
10	19.14	27.55	36.87	48.86	61.42	76.35	87.18
20	38.77	48.82	54.51	67.33	78.41	89.21	95.41
30	56.67	61.99	67.46	78.31	87.41	94.61	97.95
40	69.03	72.37	76.39	85.70	92.64	97.19	99.02
50	76.27	80.36	83.67	91.01	95.70	98.75	99.54
60	83.41	87.07	88.98	94.72	97.80	99.39	99.79
70	89.14	91.45	93.32	97.20	99.12	99.76	99.93
80	94.20	95.42	96.75	98.71	99.67	99.94	99.99
90	97.81	98.26	98.86	99.60	99.93	100.00	100.00

Table 5.3: Mean % recall on the train set.

% db searched	p=2	p=4	p=8	p=16	p=32	p=64	p=128
10	17.45	24.73	28.95	36.34	45.52	57.29	67.21
20	35.34	44.04	46.66	55.22	65.38	77.44	85.55
30	51.24	56.91	59.75	68.55	77.69	87.03	92.95
40	63.05	67.63	69.53	77.95	85.51	92.51	96.16
50	72.10	75.10	78.24	84.70	90.80	95.85	97.78
60	80.24	82.73	85.18	89.96	94.65	97.72	98.86
70	86.84	88.57	90.59	94.21	97.11	98.88	99.46
80	92.63	93.78	94.92	97.08	98.63	99.61	99.81
90	97.16	97.02	98.10	98.89	99.58	99.90	99.98

Table 5.4: Mean % recall on the test set.

changed on the test set sketches only slightly, differences between approximation quality are more significant - in the worst case almost 20% decrease of mean % recall on test set compared to the train set. Since both sets of 100 query objects were drawn from train and test set separately, it's possible that this discrepancy in mean % recall is caused by differences in selected query objects. Another possible cause is also the disproportion of size of train and test sets.

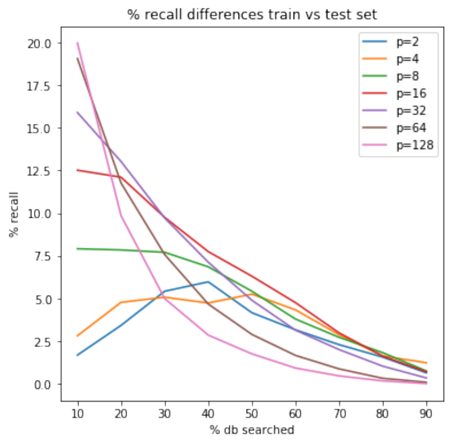

Figure 5.8: Difference in mean % recall train vs test set.

% db searched	p=2	p=4	p=8	p=16	p=32	p=64	p=128
10	1.69	2.82	7.92	12.52	15.90	19.06	19.97
20	3.43	4.78	7.85	12.11	13.03	11.77	9.86
30	5.43	5.08	7.71	9.76	9.72	7.58	5.00
40	5.98	4.74	6.86	7.75	7.13	4.68	2.86
50	4.17	5.26	5.43	6.31	4.90	2.90	1.76
60	3.17	4.34	3.80	4.76	3.15	1.67	0.93
70	2.30	2.88	2.73	2.99	2.01	0.88	0.47
80	1.57	1.64	1.83	1.63	1.04	0.33	0.18
90	0.65	1.24	0.76	0.71	0.35	0.10	0.02

Table 5.5: Mean % recall difference between train and test set.

During the experiment we showed that optimized sketches can achieve good results in terms of *k-NN* query recall. We designed sketches that even on separate test set achieved $> 85\%$ mean recall while during filtering procedure we excluded 80% of candidates for *k-NN* results.

5.5 Multi-hash index analysis

To evaluate MHI we joined train and test set sketches together and since MHI is efficient only for small search radii we took only first 16 bits. We drew 100 sketches as query objects from all 10,000 and inserted remaining 9,900 objects into 8 indexes with number of splits m ranging from 1 to 8.

5.5.1 MHI bucket statistics

We computed several statistics on buckets of all indexes to get a better idea of what effect has different setting of number of splits m on attributes of the index. Results are shown in Table 5.6.

m	r	Bucket count	Bucket size mean	Bucket size sum	m x Bucket size mean
1	0	5250	1.885714	9900	1.885714
2	1	484	40.909091	19800	81.818182
3	2	128	232.031250	29700	696.093750
4	3	64	618.750000	39600	2475.000000
5	4	48	1031.250000	49500	5156.250000
6	5	40	1485.000000	59400	8910.000000
7	6	36	1925.000000	69300	13475.000000
8	7	32	2475.000000	79200	19800.000000

Table 5.6: Multi-hash index statistics

Setting the number of splits to m will enable the index to retrieve all objects within range $r = m - 1$. Number of possible buckets is $2^{\frac{16}{m}}$. Results confirm that linear increase in m results in:

- Exponential decrease of count of non-empty buckets (fig. 5.9).
- Exponential increase of mean number of sketch references in single bucket.

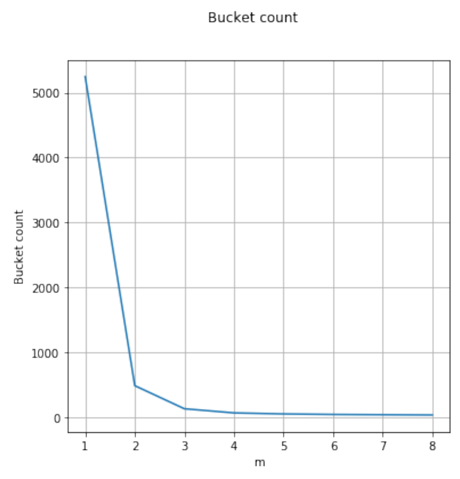

Figure 5.9: Number of non-empty buckets in MHI.

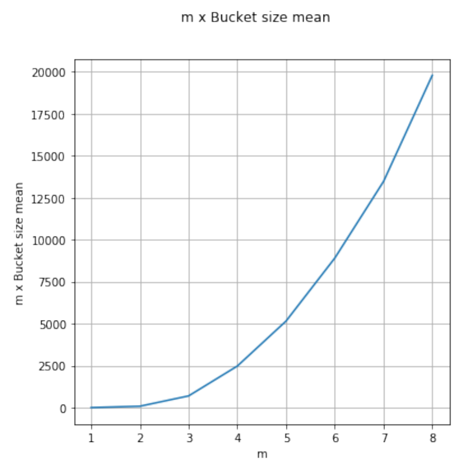

Figure 5.10: $m\times$ mean bucket size.

- Linear increase of total number of sketch references stored in the index.

In case when $m = 8$ number of buckets decreases to only 32 and mean count of objects in a bucket is $2,475$ which is $\approx 25\%$ of all objects in our dataset. In this case it's very likely for buckets to overlap and index might perform even worse than sequential scan considering it might examine all objects with additional computational complexity for looking up buckets and computing their union.

Mean bucket size $\times$ m If bits of both stored and query sketches were distributed uniformly $m \times$ *mean bucket size* would be the size of candidate set before union we would receive for sequential scan during query evaluation on average. As we can see on fig. 5.10 it grows exponentially in respect to number of splits m.

5.5.2 MHI range query evaluation

We conducted a 100 range queries on each of 8 indexes and examined retrieved candidate sets. Results are shown in Table 5.7.

m	r	Mean TP count	m x Bucket size mean	Mean |C_1|+...+|C_m|	Mean |C|	Mean % bucket overlap	Mean % db filtered
1	0	4.09	1.885714	4.09	4.09	0.000000	99.958687
2	1	32.40	81.818182	221.68	217.59	1.845002	97.802121
3	2	150.92	696.093750	1561.34	1462.64	6.321493	85.225859
4	3	495.62	2475.000000	4101.93	3424.64	16.511496	65.407677
5	4	1257.68	5156.250000	8292.10	5854.75	29.393640	40.861111
6	5	2561.43	8910.000000	13864.14	7826.59	43.547959	20.943535
7	6	4326.62	13475.000000	19794.12	8921.41	54.928989	9.884747
8	7	6207.82	19800.000000	25735.23	9417.68	63.405495	4.871919

Table 5.7: MHI range query evaluation results.

To evaluate how efficient and effective the index is with some specific setting of m we need to pay attention to two important aspects:

1. How many true positive range query results are present in the candidate set.
2. What is the size of the candidate set.

Number of true positive sketches within some radius in the sketch space is affected by distance distribution of sketches which is determined by distribution of original metric space and by sketch length. For instance according to results $\approx 12\%$ of stored objects are within range 5 from query object on average.

Size of candidate set determines how many objects will we examine using sequential scan and is equal to number of hamming distance computations that have to be done in order to retrieve set containing only true positives. Difference between candidate set size (column *Mean* |*C*|) and sum of sizes of examined buckets before union (column *Mean* |*C_1*|+...+|*C_m*|) depends on number of buckets and their cardinality. Union of buckets require iteration over all these sketch references which are of size $16/m$. Since sizes of references decrease linearly but sizes of buckets increase exponentially with increasing m for greater values of m this process can be also computationally demanding.

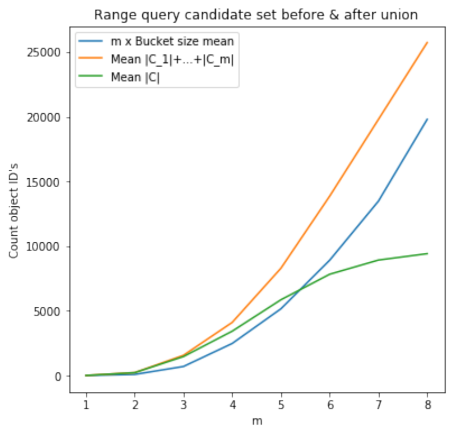

Figure 5.11: Comparison of candidate set sizes before and after union and $m\times$ bucket size.

On fig. 5.11 we visualized sum of cardinalities of candidate sets before union and cardinality of candidate set after union of buckets. Mean bucket size $\times m$ grows slower than candidate set meaning that probability of examining more populated bucket is higher than it would be if we operated on perfectly uniform distribution of bits. Some buckets have greater probability of objects falling into them and also since query objects are from the same distribution these buckets have also greater probability of being examined during query evaluation.

5.5.3 Filter & refine with sketches and MHI

Let's look back to the approximation quality experiment on test set (Table 5.4). We showed that for 100 nearest neighbor queries on sketches with length 16 bits in order to achieve recall around 36% one needs to examine 10% of objects from the original space. In other words because of approximation by sketches we filtered out 90% of original database and then achieved 36% recall on *k-NN* query after refinement of the result in the original metric space.

Searching for 10% of closest sketches is done in-memory and can be performed by sequential scan. However in order to avoid some of distance computations we organized sketches in multi-hash index (Table 5.7). We found out that due to the sketch distance distribution in order to find closest 10% of sketches we would need to search within range $r = 4$ which includes $\approx$ 12% of all sketches in our dataset. Candidate set size was $\approx$ 40% smaller than size of the dataset, therefore number of hamming distance computations was reduced to $\approx$ 60% of count of hamming distance computation that would be conducted during linear scan among sketches.

6 Conclusion

In this thesis we researched and discussed basic concepts of Metric sketches and Multi-hash index and evaluated them on real-life dataset sample consisting of high-dimensional image descriptors.

We formally defined notion of *metric sketches* and specified properties and criteria which are substantial for application of sketches to similarity searching tasks in metric spaces, most importantly high bit balance and low bit dependence. We analyzed ways of optimizing sketches to a given data distribution by pivot selection to improve these properties. We designed *FMCC* and *GMCC* heuristic algorithms for selecting pivot subset that produce independent bits. In experiments GMCC outperformed FMCC by very small margin , however computational complexity of FMCC heuristic is significantly smaller which makes it more reasonable and easy to implement heuristic for solving the problem.

In experiments we demonstrated usefulness of sketches on evaluation of 100-NN queries. With sketches of length 128 we were able to filter out 90% of the dataset objects and still achieve high recall of 0.87 and therefore concluded, that sketches can result in significant reduce of size of the candidate set searched in the original metric space while preserving high recall on results of k-NN queries. Longer sketches perform better in terms of recall, however it becomes difficult to find new pivot couples for generating balanced and independent bits when sketch size grows larger.

Furthermore we examined core ideas behind Multi-hash index and analyzed it's usage for speeding up similarity searching among sketches. We found out that index is suitable mostly for searching for objects within small distance range, which makes it perform well only for searching among short sketches. In experimental evaluation we were able to perform search in the sketch space with 40% reduction of number of performed hamming distance computations on 16-bit sketches, compared to the sequential scan. Since even short sketches perform reasonably well in terms of k-NN recall, especially with small k, Multi-hash index might be a viable choice for many similarity searching applications which would utilize sketches.

For future research we recommend to focus on several aspects that were not explored in this thesis. First is more advanced measurement of approximation quality than simple recall on nearest neighbor results and examination of false positive results. Even though we are trying to retrieve 100 nearest objects, for some applications there might be difference whether false positive objects from returned result set are still close objects, just not contained in top 100 closest or they are objects lying far from the query. Secondly, since performance of Multi-hash index apparently depend on properties of sketches such as their length, it might be interesting to find out how is the performance affected by other properties of sketches and whether we can optimize the index and sketches jointly to achieve best reduction in number of hamming distance computations. Lastly, during the experiments we found out that although sketch properties optimized for the train set were well preserved on the test set, recall on the test set significantly decreased. It might be reasonable to further analyze on discrepancy and determine the reason for it.

Sketches as binary strings are very compact enabling us to fit representation of whole database in memory and and compare objects efficiently using hamming distance computation, which can be computed by hardware instructions. Our experimental results confirmed that at cost of sometimes insignificant approximation error searching in sketch space can help avoid vast majority of distance computations needed to identify set of nearest neighbors among objects in the database. Since this approach can be used for any general metric space, range of possible applications of sketches is potentially very broad and further research of this topic might have positive impact on solving wide range of similarity searching problems.

Bibliography

1. ZEZULA, Pavel; AMATO, Giuseppe; DOHNAL, Vlastislav; BATKO, Michal. *Similarity Search: The Metric Space Approach*. 1st. Springer Publishing Company, Incorporated, 2010. ISBN 1441939725, 9781441939722.
2. BÖHM, Christian; BERCHTOLD, Stefan; KEIM, Daniel A. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Comput. Surv.* 2001, vol. 33, no. 3, pp. 322–373.
3. CLARKSON, Kenneth L. Nearest Neighbor Queries in Metric Spaces. In: *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*. El Paso, Texas, USA: ACM, 1997, pp. 609–617. STOC '97. ISBN 0-89791-888-6.
4. WANG, B.; LIAO, Q.; ZHANG, C. Weight Based KNN Recommender System. In: *2013 5th International Conference on Intelligent Human-Machine Systems and Cybernetics*. 2013, vol. 2, pp. 449–452.
5. MA, Hongxing; GOU, Jianping; WANG, Xili; KE, Jia; ZENG, Shaoning. Sparse Coefficient-Based k-Nearest Neighbor Classification. *IEEE Access*. 2017, vol. 5, pp. 16618–16634.
6. UHLMANN, Jeffrey K. Satisfying General Proximity / Similarity Queries with Metric Trees. *Inf. Process. Lett.* 1991, vol. 40, no. 4, pp. 175–179.
7. YIANILOS, Peter N. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In: *Proceedings of the Fourth Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 25-27 January 1993, Austin, Texas.* 1993, pp. 311–321.
8. FERHATOSMANOGLU, H.; TUNCEL, E.; AGRAWAL, D.; ABBADI, A. El. Approximate nearest neighbor searching in multimedia databases. In: *Proceedings 17th International Conference on Data Engineering*. 2001, pp. 503–511. ISSN 1063-6382.
9. LV, Qin; JOSEPHSON, William; WANG, Zhe; CHARIKAR, Moses; LI, Kai. Efficient filtering with sketches in the ferret toolkit. In: *Proceedings of the 8th ACM SIGMM International Workshop on Multimedia Information Retrieval, MIR 2006, October 26-27, 2006, Santa Barbara, California, USA*. 2006, pp. 279–288.

10. MOLINA, Arnoldo José Müller; SHINOHARA, Takeshi. Efficient Similarity Search by Reducing I/O with Compressed Sketches. In: *SISAP*. IEEE Computer Society, 2009, pp. 30–38.

11. CORRAL, Antonio; VASSILAKOPOULOS, Michael. Query Processing in Spatial Databases. In: *Encyclopedia of Database Technologies and Applications*. 2005, pp. 511–516.

12. MIC, Vladimir; NOVAK, David; ZEZULA, Pavel. Speeding up Similarity Search by Sketches. In: *SISAP*. 2016, vol. 9939, pp. 250–258. Lecture Notes in Computer Science.

13. BOSLAUGH, S.; WATTERS, P.A. *Statistics in a Nutshell: A Desktop Quick Reference*. O'Reilly Media, 2008. In a Nutshell (O'Reilly). ISBN 9781449397814.

14. MÍČ, Vladimír; NOVÁK, David; ZEZULA, Pavel. Improving Sketches for Similarity Search. In: KOFRON, Jan; VOJNAR, Tomas (eds.). *Tenth Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS'15)* [tištěná verze "print"]. Telc: LITERA, 2015, pp. 45–57. ISBN 978-80-214-5254-1.

15. MIC, Vladimir; NOVAK, David; ZEZULA, Pavel. Designing Sketches for Similarity Filtering. In: *ICDM Workshops*. IEEE, 2016, pp. 655–662.

16. CHAVEZ, Edgar; NAVARRO, Gonzalo; BAEZA-YATES, Ricardo A.; MARROQUIN, Jose L. *Searching in metric spaces*. ACM Comput. Surv. 2001, vol. 33, no. 3.

17. BUSTOS, Benjamin; PEDREIRA, Oscar; BRISABOA, Nieves R. A dynamic pivot selection technique for similarity search. In: *Proceedings of the 24th International Conference on Data Engineering Workshops, ICDE 2008, April 7-12, 2008, Cancún, México*. 2008, pp. 394–401.

18. ARORA, Sanjeev; BARAK, Boaz. *Computational Complexity: A Modern Approach*. 1st. New York, NY, USA: Cambridge University Press, 2009. ISBN 0521424267, 9780521424264.

19. HAMMING, Richard. Error Detecting and Error Correcting Codes. *Bell System Technical Journal*. 1950, vol. 26, no. 2, pp. 147–160.

20. WEGNER, Peter. A technique for counting ones in a binary computer. *Commun. ACM*. 1960, vol. 3, no. 5, pp. 322.

21. DONG, Wei; CHARIKAR, Moses; LI, Kai. Asymmetric distance estimation with sketches for similarity search in high-dimensional spaces. In: *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2008, Singapore, July 20-24, 2008*. 2008, pp. 123–130.

22. CIACCIA, Paolo; PATELLA, Marco; ZEZULA, Pavel. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In: *VLDB*. Morgan Kaufmann, 1997, pp. 426–435.

23. DOHNAL, Vlastislav; GENNARO, Claudio; SAVINO, Pasquale; ZEZULA, Pavel. D-Index: Distance Searching Index for Metric Data Sets. *Multimedia Tools Appl.* 2003, vol. 21, no. 1, pp. 9–33.

24. NOROUZI, Mohammad; PUNJANI, Ali; FLEET, David J. Fast Exact Search in Hamming Space With Multi-Index Hashing. *IEEE Trans. Pattern Anal. Mach. Intell.* 2014, vol. 36, no. 6, pp. 1107–1119.

25. KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. ImageNet Classification with Deep Convolutional Neural Networks. In: *NIPS*. 2012, pp. 1106–1114.

26. DONAHUE, Jeff; JIA, Yangqing; VINYALS, Oriol; HOFFMAN, Judy; ZHANG, Ning; TZENG, Eric; DARRELL, Trevor. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. In: *ICML*. JMLR.org, 2014, vol. 32, pp. 647–655. JMLR Workshop and Conference Proceedings.

27. WALTERS-WILLIAMS, Janett; LI, Yan. Comparative Study of Distance Functions for Nearest Neighbors. In: *Advanced Techniques in Computing Sciences and Software Engineering*. Ed. by ELLEITHY, Khaled. Dordrecht: Springer Netherlands, 2010, pp. 79–84. ISBN 978-90-481-3660-5.

28. DEZA, Michel Marie; DEZA, Elena. *Encyclopedia of Distances*. Springer Berlin Heidelberg, 2009.

29. DURRANT, Robert J.; KABÁN, Ata. When is 'nearest neighbour' meaningful: A converse theorem and implications. *J. Complexity*. 2009, vol. 25, no. 4, pp. 385–397.

30. BELLMAN, R.; KARUSH, R. *Dynamic Programming: A Bibliography of Theory and Application*. Rand Corporation, 1964. Memorandum (Rand Corporation).

31. AGGARWAL, Charu C.; HINNEBURG, Alexander; KEIM, Daniel A. On the Surprising Behavior of Distance Metrics in High Dimensional Spaces. In: *ICDT*. Springer, 2001, vol. 1973, pp. 420–434. Lecture Notes in Computer Science.

A FMCC and GMCC pseudocode

```
procedure REMOVEUNBALANCED($P, S, l$)
    $r \leftarrow S.nRows$
    $t \leftarrow S.nCols$
    $R \leftarrow 1 \times t$ vector, $C[i] \leftarrow 0, \forall i$
    $C \leftarrow 1 \times t$ vector, $C[i] \leftarrow 0, \forall i$
    for $i = 1 \ldots r$ do
        for $j = 1 \ldots t$ do
            if $S[i][j] = 1$ then
                $C[r] \leftarrow C[r] + 1$
    $S' \leftarrow$ copy of $S$
    $P' \leftarrow$ copy of $P$
    for $\forall i = 1 \ldots t$ do
        if $\leftarrow b(C[i], r) < l$ then
            remove $i$-th column from $S'$
            remove $i$
    return $P'$, $S'$
procedure CORRMTX($S$)
    $t \leftarrow S.nCols$
    $M \leftarrow t \times t$ matrix
    for $i = 1 \ldots t$ do
        for $j = i + 1 \ldots t$ do
            $M[i][j] \leftarrow |\rho(S.col[i], S.col[j])|$
    return $M$
```

```
procedure FMCC($P, S, p, l$)
    $t \leftarrow S.nCols$
    $P,S \leftarrow RemoveUnbalanced(P, S, l)$
    $M \leftarrow CorrMtx(S)$
    $Z \leftarrow 1 \times t$ vector
    for $i = 1...t$ do
        $Z[i] \leftarrow mean(M.col[i])$
    sort $P$ by values of $Z$ in increasing order
    return $P[: p]$
procedure GMCC($P, S, p, l$)
    $R \leftarrow \{\}$
    $P,S \leftarrow RemoveUnbalanced(P, S, l)$
    $M \leftarrow CorrMtx(S)$
    while $P.length > p$ do
        $bestCorr \leftarrow \infty$
        $r \leftarrow null$
        for $i = 1 \ldots S.nCols$ do
            $S' \leftarrow S$ without $i$-th column
            $M' \leftarrow M$ without $i$-th column and row
            $corr \leftarrow$ mean of values in upper triangle of $M'$
            if $corr < bestCorr$ then
                $bestCorr \leftarrow corr$
                $r \leftarrow i$
        remove $r$-th value from $P$
        remove $r$-th column from $S$
        remove $r$-th column and row from $M$
    return $P$
```

B Contents of the electronic attachment

Uploaded electronic attachment contains implementation of our experiments. For further information please refer to subsection 5.1.2.