# An Alternating Optimization Scheme for Binary Sketches for Cosine Similarity Search

Erik Thordsen(✉) and Erich Schubert

TU Dortmund University, Otto-Hahn-Straße 14, 44227 Dortmund, Germany
erik.thordsen@tu-dortmund.de

**Abstract.** Searching for similar objects in intrinsically high-dimensional data sets is a challenging task. Sketches have been proposed for faster similarity search using linear scans. Binary sketches are one such approach to find a good mapping from the original data space to bit strings of a fixed length. These bit strings can be compared efficiently using only few XOR and bit count operations, replacing costly similarity computations with an inexpensive approximation. We propose a new scheme to initialize and improve binary sketches for similarity search on the unit sphere, i.e., for cosine similarity. Our optimization iteratively improves the quality of the sketches with a form of orthogonalization. We provide empirical evidence that the quality of the sketches has a peak beyond which it is not correlated to neither bit independence nor bit balance, which contradicts a previous hypothesis in the literature. Regularization in the form of noise added to the training data can turn the peak into a plateau and applying the optimization in a stochastic fashion, i.e., training on smaller subsets of the data, allows for rapid initialization.

## 1 Introduction

Similarity search in large and high-dimensional data sets poses two major problems. Firstly, due to the high intrinsic dimensionality, indexing approaches are difficult and for too many dimensions degenerate to the case of a linear search in the Euclidean case as per the Nearest Neighbor Indexing Theorem [17]. Secondly, linear search is almost infeasible for many applications on very large data sets, where many such searches are necessary. In dense Euclidean space, to further complicate the issue, every distance or similarity computation is increasingly costly as the number of representation dimensions grows. Yet, when allowing for *approximate* results, i.e., not always returning correct neighbors, some of these problems can be lessened. Locality-sensitive hashing (LSH) [5] and sketching are two techniques for approximate search. LSH puts similar items into the same "hash buckets" with high probability, allowing to filter out vast parts of the data set based on comparably cheap hash functions. Alternatively, the bucket assignments can be used as features in a thus dimensionally reduced data set.


O. Pedreira and V. Estivill-Castro (Eds.): SISAP 2023, LNCS 14289, pp. 41–55, 2023.
https://doi.org/10.1007/978-3-031-46994-7_4

Sketching is a general technique for compressing data by mapping each sample onto a smaller representation like bit strings while ideally approximating certain properties by proxy, such as similarity or distance. These compressed representations can be used in a classical index or to reduce the linear search cost.

In this paper we consider similarity search using cosine similarity, although the algorithm potentially extends to inner product similarity, and we propose methods for binary sketching. Binary sketching can be seen as a special case of both sketching and LSH where each "hash function" has only two buckets. The resulting representation is a bit string for each sample. This representation is interesting because distances of bit strings can be computed efficiently using fast CPU instructions like `popcount`. To obtain such binary sketches, simple geometric expressions have been used in the literature where a 1 is assigned if a sample is inside a specific volume and a 0 otherwise (occasionally also $-1$ [1]). The volumes used as these "hash functions" can be hyperballs, hypercubes, half-spaces, or simple combinations (intersection or difference) thereof [11]. When these volumes are scaled or located such that approximately half of the data set is assigned a 1, the volumes and corresponding bits are called "balanced", otherwise "unbalanced". Balanced bits have been assumed to produce bit strings providing better recall values for approximate search whilst also leading to bit-strings of higher intrinsical dimension [12]. In the literature, the typical approach to balance bit assignments generated from half-spaces/hyperplanes is to add an affine bias such that exactly half the samples are assigned a 1. Yet, in the context of cosine similarity search, using half-spaces induced by non-affine hyperplanes are a natural choice and we limit this paper on non-affine hyperplanes.

Segmenting the unit sphere into cells with non-affine hyperplanes has a long history in the literature. Already in the 19th century, Schläfli derived the precise number of cells on the unit sphere $C(n, d)$ induced by $n$ random hyperplanes in $d$ dimensions in general position, i.e., oriented such that every intersection $k$ hyperplanes is $k$-codimensional. The formula with proof and further details is given in [15, p. 299] as

$$C(n,d) = 2\sum_{k=0}^{d-1} \binom{n-1}{k} \tag{1}$$

Whenever $d \geq n$ the number of cells equals $2^n$ and when $2d \geq n$ then there are at least $2^{n-1}$ cells. Since the number of cells only increases for larger $n$, we can in any other case give a lower bound of $2^{2d-1}$ cells. In any case, the number of cells is likely much larger than the number of samples in high dimensional data sets and with a number of hyperplanes proportional to $d$.

Accordingly, when intending to use non-affine hyperplane tessellation (occasionally denoted as conical tessellation [15]) for binary sketching, we can expect (almost) no identical bit vectors in practical settings. The goal, thus, is rather to find hyperplanes, such that the Hamming distance on bit strings is correlated to the distance on the hypersphere – at least for "small" distances as we are only interested in finding the nearest neighbors. Entirely random hyperplanes – originally proposed for cosine similarity search by Charikar [3] – are (almost) optimal if the data is uniformly distributed [13]. But on non-uniformly distributed, i.e.,

clustered or otherwise structured data, they can lead to a suboptimal distribution of cells. The resulting high pairwise Pearson correlation of bits (bit correlation) has been observed to lead to decreased filtering quality when selecting candidates for nearest-neighbor search [11], where bit correlation, bit balance, etc., are evaluated on the columns of the $|X| \times B$ matrix of bit strings of length $B$. Yet, only little research has been done on how to find a good set of hyperplanes, such that the bit correlation is minimal – at least for the case of cosine similarity search. Balu et al. [1] propose the use of geometrically orthogonal hyperplanes for which bits are greedily flipped to improve the correlation of samples with their "reconstruction" (sum of normal vectors where bit is 1 minus sum of normal vectors where bit is 0). This slightly accounts for the data distribution but is neither a proper optimization, nor does it work when all bit strings are approximately equal. It is also constrained to the bit string length equaling the data dimension. Mic et al. [11] introduced an algorithm that oversamples the number of required hyperplanes and discards all but a well-performing subset using a clique-based approximation algorithm.

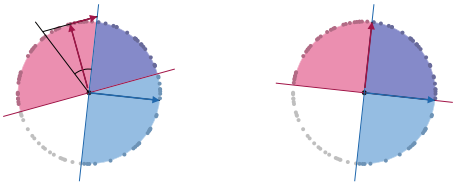
In this paper, we intend to fill this gap by providing a fast initialization algorithm that iteratively adds the best hyperplane from a set of candidate hyperplanes and an alternating optimization algorithm that iteratively minimizes the maximum (or mean) pairwise bit similarity – a different yet similar measure to bit correlation. The algorithm does not immediately enforce bit balance yet empirically increases it. Not using an affine bias eliminates a degree of freedom for the hyperplanes, whereby it suffices to rotate the hyperplanes around the origin. The iterative process allows for budgeted training, a dynamic data set, and varying bit string lengths. The update step can be used in multiple "flavors", since either a single or multiple hyperplanes can be updated in each iteration based on the most similar or all other hyperplanes. The sole drawback is that by orthogonalizing all bits (in bit assignments, not geometrically), we increase the intrinsic dimensionality of the representation, making them more difficult to index, yet, the lower length necessary for a similar recall alleviates this.

In Sect. 2 we introduce the alternating optimization algorithm. Section 3 then focuses on our initialization, which is optional to the algorithm but allows for shorter training times. This section also includes a visual example of the initialization and the optimization algorithm. In Sect. 4 we provide empirical results based on a large real world data set. Lastly we close in Sect. 5 with a summary of the paper and an outlook on potential expansions of this approach.

## 2 Alternating Optimization of Binary Sketches

The alternating optimization algorithm introduced here is called "Hyperplane-based Iteratively Optimized Binarization" (HIOB). In each iteration, we update one or multiple hyperplanes – represented by their normal vectors – by rotating them such that their bit similarity with other hyperplanes decreases. For a pair of hyperplanes, we aim to equalize the number of samples assigned 00, 01, 10, and 11. By computing the number of samples having the same or opposite bits, we obtain an approximation of the "sample density per radian" and

can compute an ideal angle to rotate one of the two hyperplanes. For that, we assume that the distribution in each of these four segments is approximately uniform. Instead of immediately rotating one of the normal vectors, we compute the tangent of the rotation angle, which yields an additive displacement vector in the tangent hyperplane. Using displacement vectors in the tangent hyperplane allows to aggregate multiple updates in one step, i.e., to rotate a single hyperplane towards or away from multiple other hyperplanes at once, similar to the mean gradient in a gradient-descent algorithm. The idea of this process is displayed in Fig. 1.



(a) Before update (b) After update

**Fig. 1.** An example to motivate the update step. On uniform data, each of the differently colored areas is of approximately the same size. From the difference in area (obtained from the angle between the normal vectors), we can derive the optimal angular change. By calculating the angular change as an additive vector in the tangent space, as displayed in (a), we allow aggregating multiple changes in one step. Afterwards the bit assignments corresponding to the hyperplanes are independent, i.e., equal in size, as displayed in (b).

Let $p$ and $q$ be two normal vectors and let $X \subset \mathbb{R}^d$ be a set of samples with $|X| = N$. We denote the number of samples assigned 1 by both hyperplanes as

$$\mathbf{11}_{p,q} := |\{x \in X \mid \langle x, p\rangle \geq 0 \wedge \langle x, q\rangle \geq 0\}| \tag{2}$$

We similarly denote the numbers of samples for the other three possible assignments with $\mathbf{01}_{p,q}, \mathbf{10}_{p,q}$, and $\mathbf{00}_{p,q}$. We obtain the fractions of samples with equal and different bits for $p$ and $q$ with

$$f_{p=q} := \frac{\mathbf{00}_{p,q} + \mathbf{11}_{p,q}}{N} \text{ and } f_{p\neq q} := 1 - f_{p=q} \tag{3}$$

with which we can compute the angular change

$$\alpha_{p,q} := \frac{(f_{p\neq q} - f_{p=q})\pi}{2} \tag{4}$$

The displacement vector for $p$ using $q$ is then defined as

$$\delta_p(q, s) := \tan(s \cdot \alpha_{p,q}) \frac{q - p\langle p, q\rangle}{\|q - p\langle p, q\rangle\|} \tag{5}$$

where $s \in (0, 1]$ is a scaling factor to control the speed of this process, resembling a learning rate. On uniformly distributed infinite data, the definition of $\delta_p(q, 1)$ is optimal, yet for non-uniformly distributed data, most of the samples may be clustered in a narrow cone. Were we to use $\delta_p(q, 1)$, we would likely skip over all samples in each update step and simply flip the bits on the assignments for $p$. To accommodate for that case and help convergence, the scaling factor can be reduced to a smaller value like 0.1. Choosing a smaller $s$ monotonously improves the optimization quality, i.e., pairwise bit independence, but increases the computational cost, since the optimum is only achieved asymptotically. If the hyperplane for $p$ should be updated using $m$ other hyperplanes, we advise to change the scale to $s/m$ and sum the displacement vectors, which results in the average displacement for the set of hyperplanes.

Our optimization aims at decreasing the maximum and average bit similarity which we define as

$$\mathcal{S}_{p,q} := 2 \left| f_{p=q} - \tfrac{1}{2} \right| \tag{6}$$

While $f_{p=q}$ is essentially the Simple Matching Coefficient [18], $\mathcal{S}_{p,q}$ is a sort of maximum over $f_{p=q}$ and $f_{p\neq q}$. For equal and inverted bit strings, the bit similarity is 1 and for independent balanced bit vectors it is 0. It is a fast approximation of the absolute Pearson correlation that only requires a single Hamming distance on the bit strings. The more balanced the bits are, the closer the two definitions align and the target of having 50% equal bits in each pair of bit strings is intended to push the bit strings towards balanced without forcing it.

The resulting algorithm is rather simple. In each iteration, we decide which hyperplanes to update and what hyperplanes to use for the update. We then compute all necessary displacement vectors, add them to the normal vectors and normalize to unit length. We then have to recompute the bit assignments for the updated vectors. To improve the run time, we propose to use a stochastic approach by only considering a subsample of the full data set when comparing bit strings and exchanging that subsample at fixed intervals similar to stochastic gradient descent. We observed that the overall quality does not suffer from this approach, and it makes the run time of the optimization algorithm independent of the data set size. To create the random subsets, we used permutations generated from multiple random chained modulo cycles on $\{0, \ldots, 2^n - 1\}$ with cycle-walking [2]. In that way, all samples are used for training once before reusing some. Additionally, storing the pairwise bit similarities in a matrix and only updating values affected by an update further speeds up the algorithm.

In our experiments, two "modes of operation" proved to be very useful: The first mode is to find the two most similar hyperplanes (the "worst offenders") and update one of them – chosen at random – using the other with a scale appropriate for the dataset (0.1 turned out to be very useful in all cases, but lower values should be used when the average bit similarity does not decrease). The random choice between the two hyperplanes avoids oscillating between two states if they are selected repeatedly. The second mode is to update all hyperplanes using all other hyperplanes at once. This mode requires to compute two displacement vectors for each pair of hyperplanes, which is why this method is rather costly.

Yet, even a small number of iterations (<10) produced a good starting point for further optimization even when initializing with uniformly random normal vectors. This second mode of operation, however, struggles to converge on a good final result, since at some point single bit assignments are changed. The discrete steps tend to be comparatively large and affect other bit correlations too much. We, hence, propose to use the first mode of operations until a desired result is achieved. Due to the problems with discrete steps in the pairwise bit similarity, other modes of operations investigated (e.g., always updating the "worst" hyperplane using all other hyperplanes) performed worse than the first mode. Pseudocodes for the two proposed modes are provided in Algorithm 1 and Algorithm 2.

**Algorithm 1** Updates one of the two normal vectors with the "worst" bit similarity by adding the displacement vector induced by the other normal vector.

**function** UpdateArgmax($\mathcal{N} \subset \mathbb{R}^b, s \in \mathbb{R}$)
  $p, q \leftarrow \arg\max_{p,q \in \mathcal{N}} \mathcal{S}_{p,q}$ ▷ *Choose "worst" pair of hyperplanes*
  $p', q' \leftarrow$ either $p, q$ or $q, p$ uniform at random ▷ *Shuffle $p$ and $q$*
  $p'' \leftarrow p' + \delta_{p'}(q', s)$ ▷ *Add displacement by $q'$*
  $\mathcal{N} \leftarrow \mathcal{N} \setminus \{p'\} \cup \{p''/\|p''\|\}$ ▷ *Add new normalized vector to result*
  **return** $\mathcal{N}$

**Algorithm 2** Updates all normal vectors at once by using the displacement vectors induced by all other normal vectors.

**function** UpdateBroad($\mathcal{N} \subset \mathbb{R}^b, s \in \mathbb{R}$)
  $s' \leftarrow s/|\mathcal{N}|$ ▷ *Compute effective scale*
  $\mathcal{N}' \leftarrow \{\}$
  **for** $p \in \mathcal{N}$ **do**
    $p' \leftarrow p + \sum_{q \in \mathcal{N}} \delta_p(q, s')$ ▷ *Add displacement by all other normals*
    $\mathcal{N}' \leftarrow \mathcal{N}' \cup \{p'/\|p'\|\}$ ▷ *Add new normalized vector to result*
  **return** $\mathcal{N}'$

This optimization process iteratively but not necessarily monotonously reduces pairwise bit similarity, bit correlation and balances most bit assignments, i.e., results in approximately 50% samples assigned 1 for most hyperplanes. Yet, the quality in terms of indexing as, e.g., measured by the $k@n$-recall using the Hamming distance as a proxy for the cosine similarity, is only improved up to some point, after which further iterations begin to reduce the recall again. Empirical evidence for these claims are discussed in Sect. 4.

## 3 RANSAC-Style Initialization

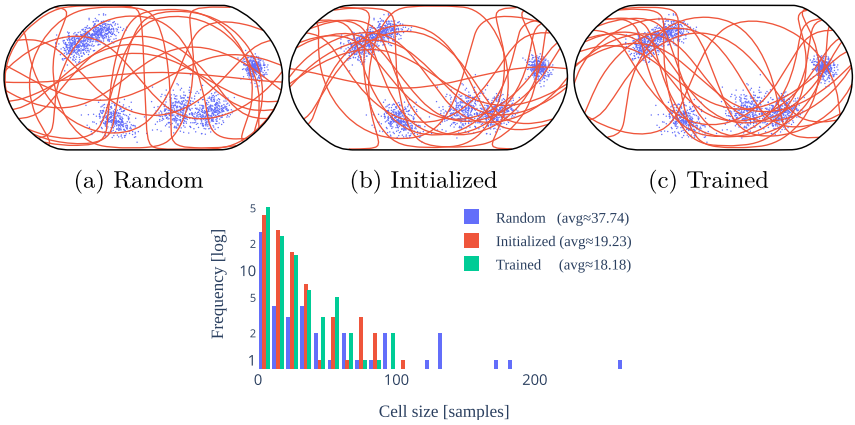
As discussed in Sect. 1, hyperplanes generated from normal vectors sampled uniformly at random from the unit hypersphere do not account for the data distribution. Whilst we could run our optimization algorithm starting with entirely

random normal vectors, that unnecessarily increases the number of iterations. We instead propose an initialization that is inspired by the random sample consensus, RANSAC [4]. We iteratively choose the next normal vector from a set of $P$ random pairs sampled uniformly at random from the data. For each of these pairs we take the normalized difference as normal vector $p$ and compute the corresponding bit assignments. Afterwards we consider the bit similarity $\mathcal{S}_{p,q}$ to all previously chosen normal vectors $q$. From the normal vectors corresponding to all of these pairs, we choose the one with the smallest maximum bit similarity to any previous hyperplane. To further speed up this process, we do not compute the initialization on all samples but rather on a subsample of much smaller size. A pseudocode of this algorithm is displayed in Algorithm 3. Even for large data sets ($|X| > 1$M), parameters such as $M = 2000$ and $P = 200$ sufficed to get decent initializations in our experiments, rendering the initialization mostly invariant of data set size as well.

**Algorithm 3** RANSAC-style initialization given a data set $X$, a number of hyperplanes $b$ to produce, a subsample size $M$, and a number of pairs $P$ to select candidates from. Returns a set of hyperplanes as normal vectors $\mathcal{N}$.

**function** RansacInitialization($X \subset \mathbb{R}^d, b \in \mathbb{N}, M \in \mathbb{N}, P \in \mathbb{N}$)
  $X' \leftarrow M$ samples from $X$ uniform at random
  $\mathcal{N} \leftarrow \left\{ \frac{x-y}{\|x-y\|} \right\}$ where $x, y \in X'$ ▷ *First normal vector*
  **for** $i \leftarrow 2, \ldots, b$ **do**
    $\mathcal{C} \leftarrow \left\{ \frac{x_i - y_i}{\|x_i - y_i\|} \mid i \in \{1, \ldots, P\}, x_i, y_i \in X' \right\}$ ▷ *Candidate normal vectors*
    $p^* \leftarrow \arg\min_{p \in \mathcal{C}} \max_{q \in \mathcal{N}} \mathcal{S}_{p,q}$ ▷ *Choose best normal vector*
    $\mathcal{N} \leftarrow \mathcal{N} \cup \{p^*\}$ ▷ *Add to final normal vectors*
  **return** $\mathcal{N}$

Figure 2 shows examples of the resulting hyperplanes of an entirely random state, of the initialization proposed here, and of an optimized state. The planes are represented by the projected great circles that are obtained by intersecting the planes with the unit sphere. These great circles are the dividing line between the subspaces assigned 0 and 1. The entirely random planes as proposed by Charikar [3] do not consider the data distribution and, hence, do not focus on splitting the clusters apart and often partition empty space. The initialized planes clearly better divide each of the clusters, yet, e.g., also by chance have one of the planes mimic an "equator" which is mostly meaningless. The optimized state rotated this plane to help divide the lower left cluster and the division of the clusters appears more homogeneous. To support this intuition, Fig. 2 shows the distribution of cell sizes and also the mean cell size for each of these plane sets. The optimized state has on average the smallest number of samples per cell on the unit sphere which should lead to a better quality in terms of indexing.




(a) Random (b) Initialized (c) Trained

(d) Distribution of cell sizes

**Fig. 2.** Cells created by 16 (a) random, (b) RANSAC-style initialized, and (c) alternating optimization trained hyperplanes for the displayed data set of 2000 samples on the three-dimensional unit sphere projected with Natural Earth projection [7]. The distribution of samples per cell is displayed in (d).

## 4 Evaluation

In our evaluation, we inspect how the optimization algorithm (HIOB) affects the bit correlation/similarity, the bit balance, and the quality in terms of the $k@n$-recall when using the Hamming distance on the bit strings as a stand-in for the cosine similarity. For that, we implemented the algorithm in Rust and added Python-bindings which can be accessed on GitHub.[1] The implementation uses arrays to store the pairwise bit similarity and current bit assignments, and is otherwise a straightforward implementation of Algorithms 1, 2 and 3. Aside from HIOB, we added functions to query objects from a data set using the binarization and a brute-force approach. The general idea of the query functions is to select a set of candidates using the nearest neighbors as per Hamming distance on the bit strings and refine the candidates to the number of required neighbors by evaluating the cosine similarity on all candidates. We further evaluated how filtering the candidates with a different set of longer bit strings affects the query time. We evaluated our implementation on subsets of the LAION5B data set [16] provided by the SISAP 2023 LAION2B challenge [19].[2] The data sets contain normalized vectors with 768 dimensions – embeddings from a deep neural network for images and texts – with varying sample sizes of 100K, 300K, 10M, 30M, and 100M together with a query set of 10K vectors with precomputed 100-nearest neighbors for validation.

[1] https://github.com/eth42/hiob.
[2] https://sisap-challenges.github.io/datasets/.

Running HIOB with RANSAC-style initialization and scale $s = 0.1$ for up to 50K iterations of Algorithm 1 (our empirical optimum in terms of $k@n$-recall) on a 32-core machine took less than 10 min even for the largest subset of 100M vectors and bit string lengths of up to 2048. We used the stochastic approach with 10K samples and 128 iterations per batch. These values were not tuned and are not sensitive based on our experience.

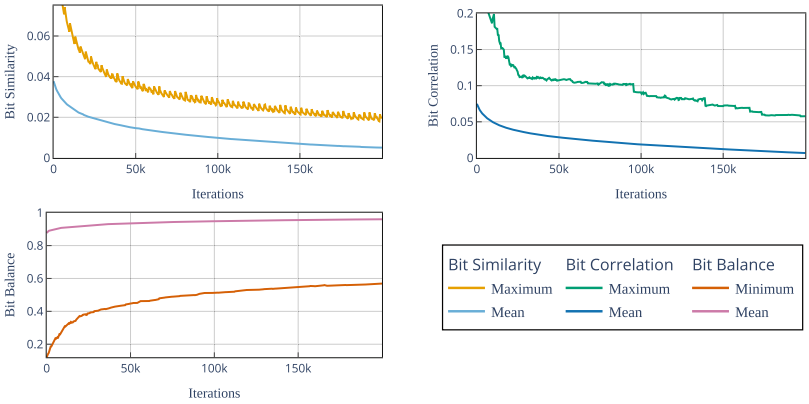
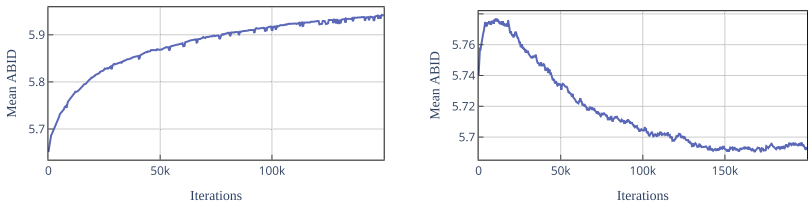



**Fig. 3.** Bit similarity, correlation and balance over HIOB iterations starting from a RANSAC-style initialization on the 10M subset and for 256 bit sketches.

Figure 3 displays how the bit similarity, bit correlation, and bit balance (ranging from 0 for pure to 1 for half-0-half-1) change over the number of iterations. As can be seen from the plot, HIOB decreased bit similarity and correlation and increased bit balance up to some local optimum, at which small discrete steps produce a bit of noise. The resulting increased independence of bits does not only lead to more homogeneous cells on the hypersphere, w.r.t. the data distribution, but should in theory also increase the intrinsic dimensionality of the bit strings. We evaluated the local intrinsic dimensionality (LID) using the ABID estimator [20,21] on the 200-nearest neighbor bit strings of the bit strings generated from the query set. The resulting mean LID-estimates are displayed in Fig. 4. It is unclear why the mean ABID estimates drop for the 10M subset, yet the change in absolute value is miniscule compared to the increase on the 100K subset. We interpret these results as the indexability of bit strings on the 100K subset decreasing while remaining almost unchanged for the 10M subset, perhaps due to an already high-LID initialization. Different tested indexes like those implemented in Hnswlib [9] and FAISS [8] did not execute the queries faster for comparable recall than brute-force on the bit strings, which would suggest a large intrinsic dimensionality.




(a) 100K subset (b) 10M subset

**Fig. 4.** Development of the local intrinsic dimensionality around the bit strings corresponding to the queries over iterations of HIOB. The low value of $\leq 6$ compared to the length of the bit strings is due to the sparsity of the space and the discrete distribution of bit strings. It is an artefact of the estimator. The relative size of estimates should be indicative of spatial complexity nonetheless.

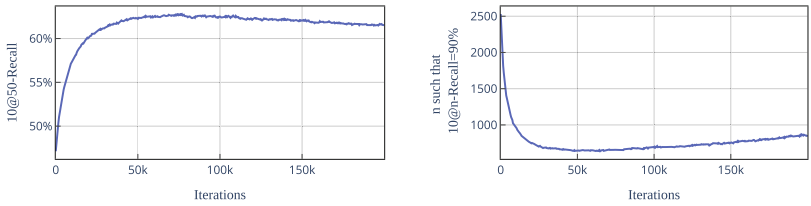



**Fig. 5.** The 10@50-recall and the number of candidates required for a 10@$n$-recall of 90% over iterations of HIOB on the 10M subset. The best results are achieved around 50K iterations (approximately 30K for the 100K subset not visualized here), which contradicts the hypothesis that better bit balance and bit independence correlate with filtering quality of binary sketches.

Whilst the results so far were within the expected spectrum, we observed that the $k$@$n$-recall decreases when running HIOB too long. As shown in Fig. 5, the $k$@$n$-recall decreased beyond roughly 50K iterations (30K iterations for 100K subset, plot omitted) even though bit balance increases and bit correlation decreases. This result contradicts the previous hypothesis in the literature, that the achievable recall during indexing with binary sketches is positively affected by larger bit balance and lower bit correlation [12]. We assume that the shape of the cells on the hypersphere is an important and so far neglected factor, although we do not have an efficient test to verify or contradict that hypothesis. In an attempt to regularize HIOB on the shape of cells, we added normally distributed random noise to each subsample drawn in the stochastic approach. We hoped it would force the hyperplanes to be less coaligned and consequentially produce more "compact" cells. Although the von Mises-Fisher distribution would be a better choice for spherical data, the normal distribution with subsequent normalization is isotropic on the sphere as well and much faster to compute. Adding full-dimensional univariate normal noise with a varying standard deviation $\sigma$ led to the average angle between hyperplanes approaching $\pi/2$ for increased $\sigma$. In principle this should provide more homogeneous cells, yet, the data distribution

is less well represented. Figure 6 displays how the added noise affects the recall on the produced bit strings. For a small amount of noise, the recall is improved and the peak in recall is "stabilized" such that it does not decrease after additional iterations. Adding too much noise rapidly decreases the recall since the data distribution is not represented by the hyperplanes well enough. The optimal choice of standard deviation in terms of recall depends on both the number of samples and the number of hyperplanes as can be seen in Fig. 7. We could, however, not find any tangible relation between the data and the optimal amount of noise, yet. The optimal amount of noise must for now be evaluated experimentally. Further insights in the topic or a better regularization are required.

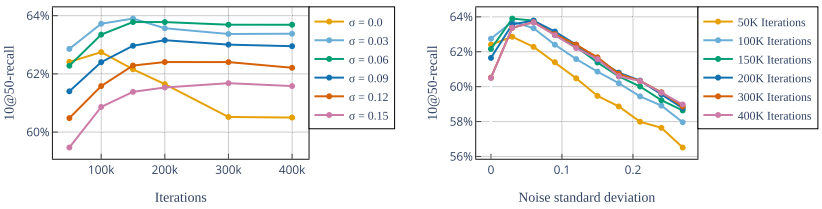
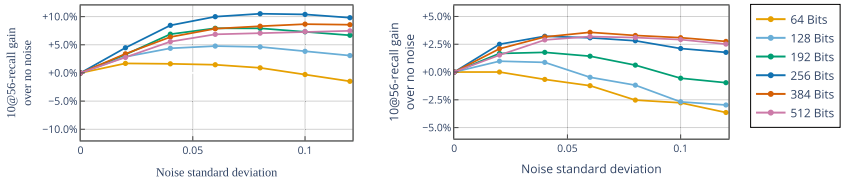



**Fig. 6.** Change in recall over varying iterations and standard deviation of the noise added to stochastic HIOB subsamples for 256 bits. A small amount of noise improves the recall and stabilizes the peak in recall.




(a) 100K subset (b) 10M subset

**Fig. 7.** Change in recall when using varying amounts of noise in the stochastic HIOB approach. All values were computed after 300K iterations. The ideal noise depends on both numbers of bits and samples.

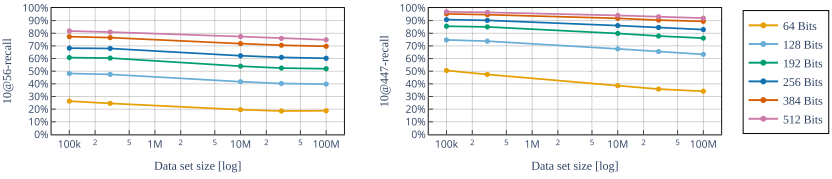



**Fig. 8.** Dependence of the 10@56- and 10@447-recall over varying data set size. Above $10M$ samples, the recall is almost unaffected by additional data.

We further explore the scalability of the approach. When considering different subsets of the LAION5B data set, the $k@n$-recall of optimized hyperplanes changed quite little over varying data set sizes as displayed in Fig. 8. This reinforces the claim that HIOB properly fits the hyperplanes to the data distribution. Aside from that, we evaluated how many candidates are necessary to achieve a certain recall for $k$-neighbor search and observed, that the recall-over-$n$ curves approximately followed the model $r(n) = a/\left(1 + \left(\frac{n}{b}\right)^c\right)$ with the inverse $n(r) = b\sqrt[c]{\frac{a}{r} - 1}$. Using least-squares to fit the model to observed values from a grid search over comparably small $n$ values, good values of $n$ can be extrapolated. Figure 9 displays the extrapolation and the required $n$ for a $10@n$-recall of 90%. The approximation error of the recall over $n$ curves were negligible (RMSE below $10^{-3}$). The number of bits $B$ affected the required number of candidates $n$ somewhere between $n \propto 1/\log(B)$ and $n \propto 1/B$ and the data set size did not affect the required number of samples much beyond $10M$ samples. Even though the $k@n$-recall values for a fixed number of candidates over varying data set sizes were not too affected, not loosing any recall at all can nonetheless require substantially larger candidate sets. The large values of $n$ in the $n$-over-$|X|$ plot for 64 bits at $|X| \in \{10\text{M}, 30\text{M}\}$ are likely due to errors in the extrapolation. In terms of $k@k$-recall, HIOB significantly outperformed the baseline (and only other entry) in the SISAP challenge [19] task B by Santoyo et al. [14], by achieving comparable recall while using only 192 bits where the baseline used 1024 bits.

The computational cost of the linear search on the bit strings is linear in the size of the data set and can not compete with more involved indexes like HNSW [9]. Figure 10 displays the queries per second over recall on the 100K, 300K, and 10M subsets of LAION5B. The HNSW implementation in Hnswlib [9] requires – with default hyperparameters – only less than a second (Queries per

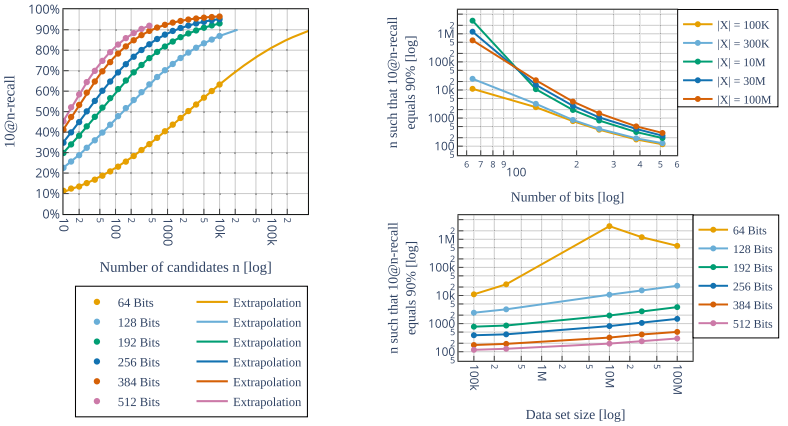



**Fig. 9.** Example of the extrapolated $10@n$-recall over $n$ for the $100M$ subset and plots generated from the extrapolated values for $n$ over number of bits and data set size such that a constant $10@n$-recall of 90% is achieved.

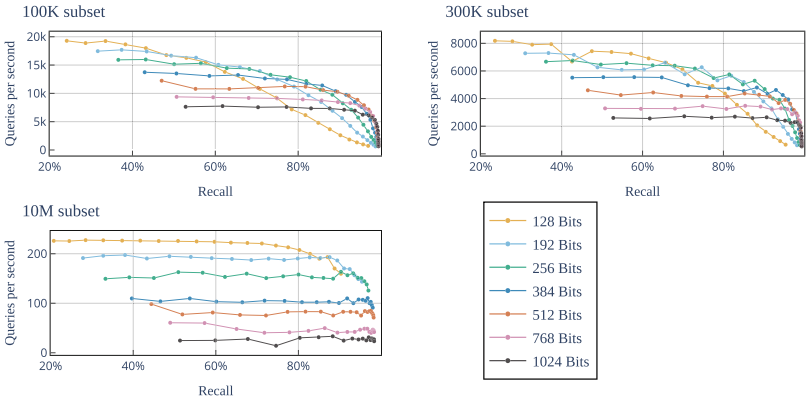



**Fig. 10.** Queries per second using brute-force search on the HIOB optimized bit strings over 10@$n$-recall where $n$ is the varied variable for each trace. 10 000 queries were performed, so, e.g., 200 queries per second correspond to 50 s.

second $>10^4$) for a recall beyond 90%, which is at best comparable to the brute-force performance on up to 100K samples. Yet, the build times of HNSW can be in the hours and increase with data set size. Considering the results published by the SISAP challenge [19], stochastic HIOB sketches can be optimized and up to millions of linear search queries can be evaluated before the fastest approaches (in terms of query time only) have been built. The linear search query time on HIOB can further outperform some of the submitted indices, although that is likely in part due to a better optimized implementation. If a decently fast index during querying with a rapid build time and comparably low memory footprint is desired, stochastic HIOB can be preferred.

## 5 Conclusion

In this paper we introduced an optimization algorithm to improve bit independence and bit balance for binary sketches on the unit sphere. We gave empirical evidence that the algorithm improves both of these values and that the algorithm is able to improve the filtering quality of binary sketches in the context of similarity search. Our experiments further highlight that bit independence and balance are not entirely correlated to filtering quality, which diminishes if the optimization iterates too long. Adding small amounts of noise allows to avoid decreasing quality when training for longer, yet, the optimal parameterization of the noise remains an open problem. This observation contradicts a previous hypothesis in the literature [12]. Using a stochastic approach to the optimization and a newly introduced initialization allows for a fast optimization of the binary sketches, with running time invariant to data set size. Yet, brute-force querying with these optimized sketches is not capable of outperforming state-of-the-art approaches like HNSW [9] and the increased bit independence makes the binarized data difficult to index.

In regards to future work, further insight on how to regularize hyperplane tessellation for similarity search is required and resulting regularizations could be added to HIOB. Further, HIOB can most likely be sped-up by using a heuristic to estimate whether or not bits can be flipped. The angle to each hyperplane can be stored and decreased by the total change in angle after each update to maintain bounds, similar to the heuristic developed by Hamerly for the $k$-Means algorithm [6]. The optimization could automatically be stopped when a peak in some quality measure, like those proposed by Mic et al. [10], is achieved. Lastly, to extend the approach towards Euclidean distances, affine hyperplanes and an appropriately modified update routine could be introduced like rotating around the intersection point of hyperplanes in the plane spun by their normal vectors.

## References

1. Balu, R., Furon, T., Jégou, H.: Beyond "project and sign" for cosine estimation with binary codes. In: IEEE International Conference Acoustics, Speech and Signal Processing, ICASSP, pp. 6884–6888 (2014). https://doi.org/10.1109/ICASSP.2014.6854934
2. Black, J., Rogaway, P.: Ciphers with arbitrary finite domains. In: Topics in Cryptology, CT-RSA, pp. 114–130 (2002). https://doi.org/10.1007/3-540-45760-7_9
3. Charikar, M.: Similarity estimation techniques from rounding algorithms. In: Symposium Theory of Computing, pp. 380–388 (2002). https://doi.org/10.1145/509907.509965
4. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM **24**(6), 381–395 (1981). https://doi.org/10.1145/358669.358692
5. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: Very Large Data Bases, VLDB, pp. 518–529 (1999). https://doi.org/10.5555/645925.671516
6. Hamerly, G.: Making k-means even faster. In: Proceedings of SIAM Data Mining, SDM, pp. 130–140 (2010). https://doi.org/10.1137/1.9781611972801.12
7. Jenny, B., Patterson, T., Hurni, L.: Flex projector-interactive software for designing world map projections. Cartographic Perspect. **59**, 12–27 (2008). https://doi.org/10.14714/CP59.245
8. Johnson, J., Douze, M., Jégou, H.: Billion-scale similarity search with GPUs. IEEE Trans. Big Data **7**(3), 535–547 (2019). https://doi.org/10.1109/TBDATA.2019.2921572
9. Malkov, Y.A., Yashunin, D.A.: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Trans. Pattern Anal. Mach. Intell. **42**(4), 824–836 (2018). https://doi.org/10.1109/TPAMI.2018.2889473
10. Mic, V., Novak, D., Vadicamo, L., Zezula, P.: Selecting sketches for similarity search. In: Advance Databases and Information Systems, ADBIS, pp. 127–141 (2018). https://doi.org/10.1007/978-3-319-98398-1_9
11. Mic, V., Novak, D., Zezula, P.: Improving sketches for similarity search. In: Proceedings of MEMICS, pp. 130–140 (2015)
12. Mic, V., Novak, D., Zezula, P.: Sketches with unbalanced bits for similarity search. In: Similarity Search and Applications, SISAP, pp. 53–63 (2017). https://doi.org/10.1007/978-3-319-68474-1_4

13. Plan, Y., Vershynin, R.: Dimension reduction by random hyperplane tessellations. Discret. Comput. Geom. **51**(2), 438–461 (2014). https://doi.org/10.1007/s00454-013-9561-6
14. Santoyo, F., Chávez, E., Tellez, E.S.: A compressed index for hamming distances. In: Similarity Search and Applications, SISAP, pp. 113–126 (2014). https://doi.org/10.1007/978-3-319-11988-5_11
15. Schneider, R., Weil, W.: Stochastic and integral geometry (2008). https://doi.org/10.1007/978-3-540-78859-1
16. Schuhmann, C., et al.: LAION-5B: an open large-scale dataset for training next generation image-text models. In: NeurIPS (2022)
17. Shaft, U., Ramakrishnan, R.: Theory of nearest neighbors indexability. ACM Trans. Database Syst. **31**(3), 814–838 (2006). https://doi.org/10.1145/1166074.1166077
18. Sokal, R.R., Michener, C.D.: A statiscal method for evaluating systematic relationships. Univ. Kansas Sci. Bull. **38**(22), 1409–1438 (1958)
19. Tellez, E.S., Aumüller, M., Chavez, E.: Overview of the SISAP 2023 indexing challenges. In: Pedreira, O., Estivill-Castro, V. (eds.) SISAP 2023, LNCS, vol. 14289, pp. 255–264. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-46994-7_21
20. Thordsen, E., Schubert, E.: ABID: angle based intrinsic dimensionality. In: Similarity Search and Applications, SISAP, pp. 218–232 (2020). https://doi.org/10.1007/978-3-030-60936-8_17
21. Thordsen, E., Schubert, E.: ABID: angle based intrinsic dimensionality - theory and analysis. Inf. Syst. **108**, 101989 (2022). https://doi.org/10.1016/j.is.2022.101989