MQH: Locality Sensitive Hashing on Multi-level Quantization Errors for Point-to-Hyperplane Distances

Kejing Lu
Nagoya University, Japan
lu@db.is.i.nagoya-u.ac.jp

Yoshiharu Ishikawa
Nagoya University, Japan
ishikawa@i.nagoya-u.ac.jp

Chuan Xiao
Osaka University, Japan
chuanx@ist.osaka-u.ac.jp

ABSTRACT

Point-to-hyperplane nearest neighbor search (P2HNNS) is a fundamental problem which has many applications in data mining and machine learning. In this paper, we propose a provable Locality-Sensitive-Hashing (LSH) scheme based on multi-level quantization errors to solve this problem. In the indexing phase, for each data point, we compute the hash values of its residual vectors generated by a stepwise quantization process. In the query phase, for each processed point, we first determine its suitable level for hashing and then determine the size of hash bucket based on its quantization error in that level. We theoretically show that this treatment not only yields a probability guarantee on query results, but also makes the generated hash functions much more efficient to prune those false points. Experimental results on five real datasets show that the proposed approach generally runs 2X-10X faster than the state-of-the-art LSH-based approaches.

PVLDB Reference Format:
Kejing Lu, Yoshiharu Ishikawa, and Chuan Xiao. MQH: Locality Sensitive Hashing on Multi-level Quantization Errors for Point-to-Hyperplane Distances. PVLDB, 16(4): 864 - 876, 2022.
doi:10.14778/3574245.3574269

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/LUKEJING/MQH.

1 INTRODUCTION

In recent years, *Point-to-hyperplane Nearest Neighbor Search* (P2HNNS) problem has attracted much attention due to its wide applications in pool-based active learning [27, 30, 32], dimension reduction [5, 29] and maximum margin clustering [6, 26]. This problem can be described as follows: given a dataset $\mathcal{D}$ in a Euclidean space $\mathbb{R}^d$ and a query hyperplane H with dimension $d-1$ in the same space, how do we efficiently find the data point closest to H? When d is low, we can solve this problem by existing tree structures like KD-Tree [4] and Cone-Tree [25], etc. However, as d goes larger, these tree structures lose effectiveness due to *the curse of dimensionality* [16]. Currently, for the search in high-dimensional spaces, *Locality Sensitive Hashing* (LSH) has been regarded as an efficient technique due to its robust theoretical guarantees and sub-linear query overheads [1, 10, 18, 20]. The basic idea of LSH is that, by designing a hash function family, the approximate nearest neighbors of queries can be found with probability guarantees by comparing the hash values of queries and those of data points. Note that, except for the theoretical implications, the guarantees owned by LSH are also important for the determination of parameters. Therefore, LSH-based approaches still play both theoretically and practically important roles especially in the database field.

Currently, although most of LSH-based schemes are designed to solve the point-to-point nearest neighbor search problem, by some transformations on the hyperplane H, it is feasible to design effective LSH families, such as BH, MH, NH and FH, to solve the P2HNNS problem [15, 33, 34]. Nevertheless, in this paper, we will take a totally different way to design the LSH function family. Our motivation is based on the following fact. Since LSH is essentially a random-projection technique, it generally performs worse than those heuristic learning-based approaches, such vector quantization, in capturing important features on real datasets. Therefore. it is natural to raise the following question: is it possible to design a provable LSH-based scheme which could achieve the competitive performance of quantization-based approaches. In this paper, our main goal is to devise such structure. In addition, compared with BH and MH [33, 34], our approach can apply to non-normalized datasets, while compared with NH and FH [15], our approach does not need any transformation on the query hyperplane and data points. Actually, the dimension of transformed data is much higher than the original one, which may weaken the efficiency of random projection in the transformed space.

It is notable that, except for LSH-based approaches mentioned above, there also exist some heuristic approaches (for the point-to-point search) like vector quantization [2, 3, 13, 17, 23] and similarity graph [11, 19, 22], which generally own higher search efficiencies than LSH-based ones. However, we argue here that a theoretical guarantee owned by LSH is sill important for some applications. For instance, when we deal with the top-k nearest neighbor search problem which requires high query accuracies, it is very hard to determine the size of the candidate set containing those points whose exact distances to the query need to be computed if no theoretical guarantee exists. Thus, in this paper, we not only hope to greatly improve the search efficiency of existing LSH-based approaches but also need to ensure that the proposed approach owns guarantees on query results. Actually, we can even adjust their relationships in a reasonable manner, as will be shown in Sec. 4.

Before proceeding into the details, we first present the definition of P2HNNS problem. For a given d-dimensional query hyperplane H, we can express it as $q_{d+1} + \sum_{i=1}^{d} y_i q_i = 0$, where $\{q_i\}$'s are hyperplane parameters and $\{y_i\}$'s represent the coordinates of

Proceedings of the VLDB Endowment, Vol. 16, No. 4 ISSN 2150-8097.
doi:10.14778/3574245.3574269

points on H. Then, the P2HNNS problem can be written in a formal way as follows:

$$x_{\min} = \arg\min\{s_H(x) = |\langle \boldsymbol{x}, \boldsymbol{q} \rangle - b|\} \tag{1}$$

, where $x_{\min}$ is the true nearest neighbor; q denotes the normalized vector of $[q_1, \ldots, q_d]^\top$ and $b = -q_{d+1}/\sqrt{\sum_{i=1}^{d} {q_i}^2}$. For each x, we first quantize it to $\tilde{x}$ via some quantization technique and decompose x as $x = \tilde{x} + r(x)$, where $r(x)$ denotes the residual vector of x. We observe that, to obtain a strict probability guarantee on query results, we only need to consider residual vectors instead of original data points tackled by existing LSH-based schemes, that is, we only compute the hash values of $r(x)$. On the other hand, since for existing quantization techniques, the quantization error $\|r(x)\|$ is generally not taken into account in the query phase, the treatment of $r(x)$ can help improving the query accuracy further. Note that, our work goes far beyond a straightforward combination of a quantization process and a hashing process. In fact, our focus in this paper is to show that there are some intrinsic connections between these two processes, which not only yields the theoretical guarantees on our final results, but also implies when to combine these two processes from the perspective of efficiency. Both of these two aspects lay the foundation for the proposed structure. Specifically, the connections mentioned above can be shown roughly as follows. (1) For every data point, the size of hash bucket depends on the quantization error. Generally, the smaller the quantization error is, the smaller the required size of hash bucket is. Note that, as the size decreases, the probability that the processed point falls inside the hash bucket increases, which means that we can prune the false point (if it is) more easily. (2) For those points which are far away from H, we can accordingly tolerate larger quantization errors. In fact, for every data point, by checking the relationship between its distance to the hyperplane and its quantization error, we can automatically determine if the current quantization error is tolerable for the examination of current hash values (More details can be found in Fig. 1.).

Based on the observations above, we summarize our contributions as follows. Note that, the proposed framework is essentially an LSH-based scheme and is compatible with any *Multiple Codebook Quantization* (MCQ) technique.

(1) In Sec. 3, we derive some results regarding the random projection of residual vectors. We show that, under some assumptions, if we set the size of hash bucket to a reasonable value, the nearest point to the hyperplane is guaranteed to be found with a user-specified success probability. This not only yields a theoretical guarantee of our final algorithm, but also quantitatively shows how the size of quantization error affects the effectiveness of random projection.

(2) In Sec. 4, based on the results obtained in Sec. 3, we propose a structure called MQH (**H**ashing on **M**ulti-level **Q**uantization Errors). Specifically, we generate several groups of hash functions, each of which corresponds to a level of quantization error. In the query phase, we can automatically determine a level for each data point in an online fashion and examine its hash values in that level. If the processed data point can be pruned with the probability guarantee determined beforehand, we skip the examination of its remaining sub-codewords to gain the efficiency.

Table 1: Some notations

Notation	Explanation
$\mathcal{D}$	The dataset
n	The size of dataset
H	The query hyperplane
q	An arbitrary normalized vector of H
L	The number of levels (L_0 level is excluded)
$\mathcal{M}$	The number of sub-codebooks in each level
$\hat{x}$	The quantized vector of x
$r(x)$	The residual vector of x
$s_H(x)$	The distance of point x to hyperplane H
c	The approximation ratio ($0 < c < 1$)
w	The half-width of hash bucket
m	The number of hash functions in each level
l_0	The parameter regarding the collision threshold
δ	The parameter controlling search performance
ϵ	The error rate ($0 < \epsilon < 1$)
$h_\alpha(x)$	The hash value of x regarding vector α
#Col(,)	The collision number of two points
α, β	Two coefficients regarding the time complexity

(3) In experiments, we show that, with the guarantee for approximate P2HNNS solutions, MQH runs 2X-10X faster than existing P2HNNS solvers for each target recall rate. In addition, MQH with the strict guarantee of recall rates also generally performs better than the competitive solver FH, which shows that MQH could achieve a practical tradeoff between search performance and theoretical bound.

The rest of this paper is organized as follows. In Sec. 2, we introduce some relevant definitions of P2HNNS and some LSH-based P2HNNS solvers. In Sec. 3 and Sec. 4, we introduce the details of MQH. In Sec. 5, we present some analysis of the proposed algorithm. In Sec. 6, we verify the effectiveness of MQH by experiments.

2 RELATED WORK

2.1 P2HNNS and LSH Function Families

Firstly, we introduce some background and definitions with respect to LSH. Given a dataset $\mathcal{D}$ of n d-dimensional data points, P2HNNS needs to find the point $x_{\min}$ in $\mathcal{D}$ with the minimum distance to query hyperplane H. However, since finding such exact solution may be exhaustive, especially in high-dimensional spaces, some researchers turn to find approximate solutions and introduce approximation ratio c to measure the difference between the true nearest neighbor and the returned approximate nearest neighbor. Specifically, for *c-approximate nearest neighbor* (c-ANN) search, only an approximate nearest neighbor x_* needs to be returned, that is, $s_H(x_*) \leq c s_H(x_{\min})$, where $s_H(x)$ denotes the distance between x and H. Similarly, kNN search returns k results $x_{\min}^{(i)}$ $(1 \leq i \leq k)$, where $x_{\min}^{(i)}$ is the i-th true nearest neighbor of H. Its c-approximate version, c-k-ANN, returns a set of k objects $x_*^{(i)}$ $(1 \leq i \leq k)$ satisfying $s_H(x_*^{(i)}) \leq c s_H(x_{\min}^{(i)})$.

Next, let us turn to the point-to-point search in ℓ_2 space to introduce the classical definition of LSH. We use q to denote the

query point and x to denote any data point. That is, the distance function $s_H(x)$ should be replaced by $\|x - q\|$ in this case. Given a hash function family $\mathcal{F}$ and a hash function h drawn randomly from $\mathcal{F}$, if the following two conditions are satisfied, where r denotes a search radius and $0 \leq p_1 < p_0 \leq 1$, we say that $\mathcal{F}$ is a (r, cr, p_0, p_1)-sensitive hash function family.

(1) $\Pr[h(q) = h(x)] \geq p_0$ if $\|q - x\| \leq r$.

(2) $\Pr[h(q) = h(x)] \leq p_1$ if $\|q - x\| \geq cr$.

It is easy to see that, according to the properties above, we can use $\mathcal{F}$ to distinguish the nearest neighbor from other false points. We will show that, the hash function family designed in this paper also satisfies these two conditions (See Lemma 1 and Lemma 2 below). Actually, in our proposal, the normalized vector of query hyperplane plays a similar role of the query point in the point-to-point search problem.

2.2 Existing P2HNNS Solutions

To solve P2HNNS with probability guarantees, some researchers proposed hyperplane hashing schemes to deal with the hyperplane. The first two solutions in this category are AH and EH [24, 28] whose basic ideas are to design hash function families sensitive to the angle between every data vector and the hyperplane. Later, to make better use of the LSH property, authors in [33] and [34] proposed BH and MH with bilinear and multi-linear hash functions, respectively. Recently, authors in [15] proposed NH and FH. The basic ideas of NH and FH are to embed data points into another Euclidean space with higher dimensions, and then transform P2HNNS to *Nearest Neighbor Search* (NNS) and *Furthest Neighbor Search* (FNS) in the new space.

As mentioned earlier, except for LSH-based approaches, there also exist some heuristic approaches for the point-to-point search like vector quantization [2, 3, 13, 17, 23] and similarity graph [11, 19, 22]. For the completeness, we would also like to discuss if graphs are suited to P2HNNS. The effectiveness of similarity graph depends on the Voronoi Diagram with all data points as seed points. If the query is a point, it lies in one and only one Voronoi cell and we may reach the desired cell along a searching path. However, if the query is a hyperplane, it can intersect with many Voronoi cells. Since graph-based search is essentially a greedy algorithm, it can not reach these cells simultaneously in a single-round search. Therefore, such strategy is not applicable to P2HNNS.

3 RANDOM PROJECTION OF RESIDUAL VECTORS

In this section, we present some theoretical results such that the introduction of MQH in the next section becomes clear and natural. According to (1), for each x, we need to compute or estimate $|\langle \hat{\boldsymbol{x}}, \boldsymbol{q} \rangle - b + \langle \boldsymbol{r}(\boldsymbol{x}), \boldsymbol{q} \rangle|$ to find x_*, which is always interpreted as the true nearest neighbor in this section. Since it is always efficient to compute the exact value of $\langle \hat{\boldsymbol{x}}, \boldsymbol{q} \rangle$ via quantization, if we can estimate $\langle \boldsymbol{r}(\boldsymbol{x}), \boldsymbol{q} \rangle$ precisely and determine if this value is smaller or larger than $b - \langle \hat{\boldsymbol{x}}, \boldsymbol{q} \rangle$, we can also estimate the point-to-hyperplane distance precisely since b is always known and fixed. The precision mentioned here is measured by a criterion used to judge if the value $\langle \hat{\boldsymbol{x}}, \boldsymbol{q} \rangle$ is smaller or larger than some bound determined beforehand with a user-specified success probability. One of main goals in this section is to determine such criterion. Specifically, let us consider the following concrete problem: given a residual vector $r(x) = x - \hat{x}$, an interval $[b - w, b + w]$ and a value $I(\hat{x}) = \langle \hat{\boldsymbol{x}}, \boldsymbol{q} \rangle < b - w$, how do we get a lower bound of the probability that the value of $I(\hat{x}) + \langle \hat{\boldsymbol{x}}, \boldsymbol{q} \rangle$ lies in the interval $[b - w, b + w]$, where w denotes the exact minimum point-to-hyperplane distance (The discussion for $I(\hat{x}) > b + w$ is similar, as will be shown later.). Although w is unknown in practice, we assume that we know its value here for the simplicity and will get rid of this assumption in our main theoretical result (Th. 1). Clearly, according to the definition in (1), if we can answer this question, we actually obtain a criterion to determine if an arbitrary data point is the nearest one with probability guarantees.

To solve the problem mentioned above, let us introduce some notations and build their relationships. Let $n(x) = \|r(x)\|$ and $\tilde{x} = r(x)/n(x)$. We introduce two normalized vectors q^+ and q^-, where $q^+ = q/\|q\|$ and $q^- = -q/\|q\|$. If $b - w \leq I(\hat{x}) \leq b + w$, we compute the exact distance of x to H since x is a promising candidate of the nearest neighbor. Thus, in the following discussion, we only

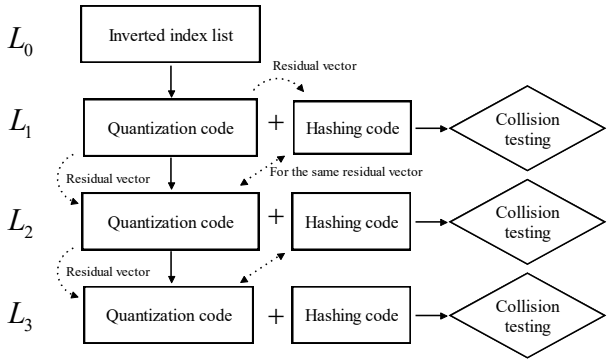


Figure 1: A working mechanism of MQH. We can see that MQH has a hierarchical structure. Specifically, in each level, we need to make a decision, that is, we choose to check the hashing code in this level or to step to the next level to decrease the quantization error further, which depends on if the processed point is highly likely to be a false point. From the L_1 level, every data point is represented by a combination of a hashing code and a quantization code. Particularly, the hashing code in the L_i level ($i \geq 1$) and the quantization code in the L_{i+1} level are the approximate representations of the same residual vector although they are different. In the query phase, for every data point which could enter the L_1 level, we need to check if it is the false point. Specifically, in each level, we need to make a decision, that is, we choose to check the hashing code in this level or to step to the next level to decrease the quantization error further, which depends on if the processed point is highly likely to be a false point, since, if this point cannot pass the collision testing in the current level, we prune it instantly. Clearly, this structure can take the different advantages of hashing codes and quantization codes simultaneously, that is, the hashing code helps us pruning false points with deterministic probability guarantees while quantization codes can approximate residual vectors in a more accurate way to strengthen the collision testing.

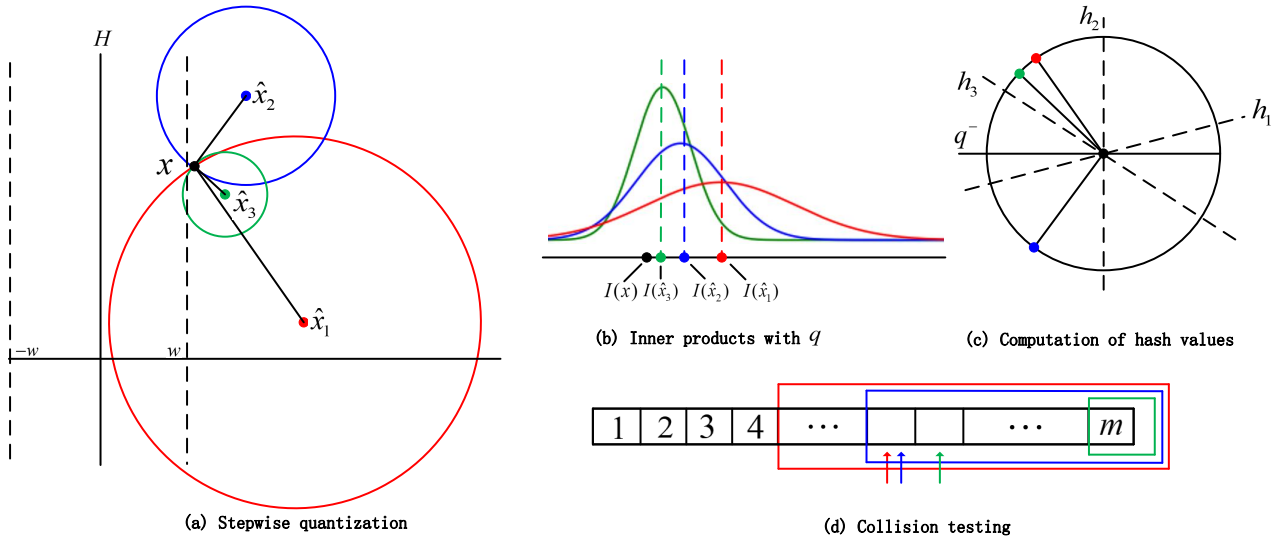


Figure 2: A geometrical illustration. In this example, we show how to prune the false point x (in black) by MQH. By a stepwise quantization, we approach x in the order $\hat{x}_1$, $\hat{x}_2$, $\hat{x}_3$ (red-blue-green).

consider the case $I(\hat{x}) < b - w$. If $I(\hat{x}) > b + w$, the discussion is completely similar if we replace q^+ by q^- (see Fig. 2). Clearly, when $I(\hat{x}) < b - w$, we need to check the relationship between $I^+(\tilde{x}) = \langle \boldsymbol{q}^+, \tilde{\boldsymbol{x}} \rangle$ and $[t_0(x), t_1(x)]$, where $t_0(x) = (b - w - I(\hat{x}))/n(x) \geq 0$ and $t_1(x) = (b + w - I(\hat{x}))/n(x)] \geq 0$. Specifically, there are the following three additional cases. Case 1: $1 < t_0(x)$. In this case, we prune x since its distance to the hyperplane is definitely greater than w. Case 2: $1 < t_1(x)$. We set $t_1(x)$ to 1 such that this case can be included in the following case. Case 3: $0 \leq t_0(x) \leq t_1(x) \leq 1$. We first introduce some notations. Let $h_\alpha(\tilde{x})$ be the hash value of point $\tilde{x}$ regarding hash function h_α. That is, $h_\alpha(\tilde{x}) = 1$ if $\langle \tilde{\boldsymbol{x}}, \boldsymbol{\alpha} \rangle \geq 0$ and $h_\alpha(\tilde{x}) = 0$ otherwise, where α is drawn randomly from any isotropic distribution U [7]. Then, we have the following result.

Lemma 1. *Suppose that $w > 0$. If $-w \leq s_H(x) \leq w$, $\Pr_{\alpha \in U}[h_\alpha(\tilde{x}) = h_\alpha(q)]$ lies in the interval $[P_0, P_1]$, where $P_0 = 1 - [\arccos(t_0)/\pi]$ and $P_1 = 1 - [\arccos(t_1)/\pi]$.*

Proof. Since $\left\|q^+\right\| = \|\tilde{x}\| = 1$, by the definitions of t_0 and t_1, it is easy to see that $-w \leq s_H(x) \leq w$ is equivalent to $t_0 \leq \langle \boldsymbol{q}^+, \tilde{\boldsymbol{x}} \rangle \leq t_1$, and is also equivalent to $\arccos(t_1) \leq \theta_x \leq \arccos(t_0)$, where θ_x is the angle between q^+ and $\tilde{x}$. On the other hand, it is well known that [7], for two normalized data points x_1, x_2, and a hash function $h_\alpha(x)$ defined above, we have the following collision probability:

$$\Pr[h_\alpha(x_1) = h_\alpha(x_2)] = 1 - \theta/\pi \tag{2}$$

, where θ is the angle between x_1 and x_2. Then, we obtain the result in Lemma 1. □

For the nearest neighbor x_*, Lemma 1 provides us with a characteristic of its hash value. Next, similar to traditional LSH-based schemes, we introduce an approximation ratio $c > 1$ and consider those false data points x's with $s_H(x) < -cw$ or $s_H(x) > cw$. Then, we have the following result.

Lemma 2. *Suppose that $c > 1$ and $\eta(x) = (c-1)w/n(x)$. If $s_H(x) < -cw$ and $\eta(x) \leq t_0$, $\Pr_{\alpha \in U}[h_\alpha(\tilde{x}) = h_\alpha(q)] \leq P_0'$, where $P_0' = 1 - [\arccos(t_0 - \eta(x))/\pi]$. If $s_H(x) > cw$ and $\eta(x) \leq 1 - t_1$, $\Pr_{\alpha \in U}[h_\alpha(\tilde{x}) = h_\alpha(q)] \geq P_1'$, where $P_1' = 1 - [\arccos(t_1 + \eta(x))/\pi]$.*

Proof. since $s_H(x) \leq -cw$ is equivalent to $\theta_x \geq \arccos(t_0 - \eta(x))$ and $s_H(x) \geq cw$ is equivalent to $\theta_x \leq \arccos(t_1 + \eta(x))$, we obtain the result in Lemma 2 by Lemma 1. □

Note that we do not need to consider the other cases of $\eta(x)$, that is, $\eta(x) > t_0$ or $\eta(x) > 1 - t_1$ since they actually have been tackled in the preceding discussion. On the other hand, it is easy to see that $P_0' < P_0$ and $P_1' > P_1$. Moreover, the differences $P_0 - P_0'$ and $P_1' - P_1$ increase as c increases, which implies the LSH property. That is, we can use hash function h_α to distinguish the nearest neighbor x_* from other false points. Since a single hash function can only be regarded as a weak classifier, by combing m hash functions generated independently, we can actually get a stronger one [12]. We introduce the notation $\#Col(\tilde{x}, q^+)$ to denote the collision number between $\tilde{x}$ and q^+, which indicates how many times $h_{\alpha_i}(\tilde{x})$ equals $h_{\alpha_i}(q)$, where $i = 1, \ldots, m$ and $\alpha_i \sim U$, i.i.d. Let $g(\epsilon) = \sqrt{\frac{m}{2} \log \frac{1}{\epsilon}}$. Then, by the Hoeffding's inequality, we have the following result.

Lemma 3. *Given an error $\epsilon > 0$. Let $l_0 = g(\epsilon)$ and $l_1 = g(\epsilon/2)$. If $-w \leq s_H(x) \leq w$, $\Pr[\#Col(\tilde{x}, q^+) \geq mP_0 - l_0] \geq 1 - \epsilon$ and $\Pr[mP_0 - l_1 \leq \#Col(\tilde{x}, q^+) \leq mP_1 + l_1] \geq 1 - \epsilon$.*

Proof. We focus on the second inequality in the statement of Lemma 3, while the first inequality can be proved in a similar way.

Based on the relationships between x and P_0, P_1, we have the following two Hoeffding's inequalities:

$$P(\#\mathrm{Col}(\tilde{x}, q) \leq mP_0 - l_1) \leq e^{-2m(l_1/m)^2} = \epsilon/2. \tag{3}$$

$$P(\#\text{Col}(\tilde{x}, q) \geq mP_1 + l_1) \leq e^{-2m(l_1/m)^2} = \epsilon/2. \quad (4)$$

Thus, l_1 should be set to $\sqrt{\frac{m}{2}\log\frac{2}{\epsilon}}$, which is exactly the setting in Lemma 3. □

By Lemma 3, we can check if the collision number of each point lies in either of these two buckets to determine the candidate set. In this way, the probability that x_* does not pass the collision testing is bounded by ϵ. In practice, we mainly use $[mP_0 - l_0, m]$, where $l_0 = g(\epsilon/2)$, since most of false points fall outside this bucket. If x falls inside this bucket, we further check $[0, mP_1 + l_0]$.

Next, let us take hash bucket $[mP_0 - l_0, m]$ as an example and simply discuss why such hashing scheme potentially performs better than previous ones. From the definitions of t_0 and t_1, it is easy to see that, the size of hash buckets highly depends on the quantization error, that is, $n(x)$. Roughly speaking, as $n(x)$ decreases, the size of hash bucket, that is, $m - mP_0$ is very likely to decrease accordingly, which strengthens the collision testing and makes false points more difficult to be added into the candidate set. Later, we will use this property to design our index structure (See Fig. 1 and Fig. 2 for more details.).

4 HASHING ON MULTI-LEVEL QUANTIZATION ERRORS

We have shown the impact of $n(x) = \|r(x)\|$ and the fact that we can compute hash values of $\tilde{x} = r(x)/n(x)$ to find the nearest neighbor. It is natural to raise the following question: how do we determine a suitable value of $n(x)$ for an efficient collision testing of $\tilde{x}$. If we answer this question, we can automatically and reasonably determine the number of required sub-codebooks for each point and the size of candidate set, while both of these two parameters are hard to be tuned for pure quantization approaches. Generally, we cannot control $n(x)$ since quantization errors depend on the used quantization approach. Nevertheless, we can alternatively use a stepwise quantization process to achieve our goal. Specifically, we generate a codebook C_1 by any existing quantization approach. For every data point x, we first use C_1 to quantize $r_0(x)$ to $\hat{x}_1$ and compute the residual vector $r_1(x) = r_0(x) - \hat{x}_1$ ($r_0(x)$ is the residual vector of x in the L_0 level.). Then, we generate codebook C_2 based on the residual vectors $r_1(x)$'s and quantize every $r_1(x)$ to $\hat{x}_2$ with residual vector $r_2(x) = r_1(x) - \hat{x}_2$. We repeat this process and generate L codebooks $C_1, C_2, \ldots, C_L$. Clearly, x can be represented in L ways, that is, $x = \sum_{i=1}^{\ell} \hat{x}_i + r_\ell(x)$ $(1 \leq \ell \leq L)$, where ℓ actually denotes a level of quantization error. Note that this treatment is essentially different from *Residual Quantization* (RQ) [8] since RQ focuses on the minimization of $\|r_L(x)\| = \left\|x - \sum_{i=1}^{L} \hat{x}_i\right\|$ while in our proposal, the focus is how to choose a reasonable value of ℓ such that the residual vector $r(x)$ in that level is suitable for hashing. Clearly, we can always ensure that $\|r_\ell(x)\| \leq \|r_{\ell-1}(x)\|$ if the codebook in each level contains the zero codeword. On the other hand, we can give the condition under which the size of hash bucket decreases in the next level. We extend some notations for x as follows. $n_\ell(x) = \|r_\ell(x)\|$; $\tilde{x}_\ell = r_\ell(x)/\|r_\ell(x)\|$; $t_0^{(\ell)}$, $P_0^{(\ell)}$ are the values of t_0 and P_0 in the ℓth level, respectively; θ_ℓ is the angle between $\tilde{x}_\ell$ and q^+; $W_\ell = [mP_0^{(\ell)} - l_0, m]$ is the hash bucket in the ℓth level. Then, we have the following result, where the condition $\cos\theta_{\ell+1} \leq t_0^{(\ell)}$ is necessary since otherwise, we should compute the exact distance of x to the hyperplane (If $I(\hat{x}_\ell) > b + w$, we have a similar result.).

Lemma 4. *Suppose that $I(\hat{x}_\ell) < b - w$ and $\cos\theta_{\ell+1} \leq t_0^{(\ell)}$ $(1 \leq \ell \leq L-1)$, $W_{\ell+1} \subseteq W_\ell$ if and only if $\cos\theta_{\ell+1} \leq \frac{n_\ell(x) - n_{\ell+1}(x)}{n_\ell(x)} t_0^{(\ell)}$.*

Proof. By the definitions of notations, $W_{\ell+1} \subseteq W_\ell$ is equivalent to the following inequality:

$$\frac{b - w - I(\hat{x}_\ell)}{n_\ell(x)} \leq \frac{b - w - I(\hat{x}_{\ell+1})}{n_{\ell+1}(x)}. \quad (5)$$

On the other hand, we have $I(\hat{x}_\ell) + n_\ell(x)\cos\theta_{\ell+1} = I(\hat{x}_{\ell+1})$. Actually, we assume that $n_\ell(x) = \|\hat{x}_{\ell+1}\|$ here, which can always hold if we use the norm-explicit quantization technique [9] in each level. Then, by some elementary transformations and the definition of $t_0^{(\ell)}$, we have the result in Lemma 4. □

Lemma 4 describes the relationship between the quantization error and the size of hash bucket. The condition making the bucket size decrease in the next level depends on two factors: the decreasing rate of the quantization error and the value of $t_0^{(\ell)}$. The first factor explains the motivation of using a stepwise quantization. As for $t_0^{(\ell)}$, we would like to show that, even if its value is much smaller than 1 (For example, $t_0 = 0.5$.), this condition is still highly likely to be satisfied. To show this, we make some assumptions on the distribution of residual vectors. For every $r(x)$, we suppose that it is drawn randomly from the sphere $S^d(n(x))$, which is generally close to the practical error distribution. Under this assumption, we can get a closed form of the probability that x passes the collision testing as d goes infinity. Let $F(\tau; m, p)$ be the probability that the variable x of Binomial distribution $B(m, p)$ is not less than τ and $G(d, m, \tau) = \int_0^\pi [e^{-d\cos^2\theta/2}\sqrt{\frac{d}{2\pi}}F(\tau; m, 1 - \frac{\theta}{\pi})|\sin\theta|]d\theta$. Then, we have the following asymptotic result.

Lemma 5. $\Pr_{\tilde{x}\sim U(S^d(1))}[\#Col(\tilde{x}, q) \geq \tau] \to G(d, m, \tau)$ *as d goes infinity.*

Proof. Let v_1 and v_2 be two vectors drawn randomly from the unit sphere $S^d(1)$. We denote the angle between v_1 and v_2 by θ, and denote the probability density function of θ by $p^d(\theta)$. By some elementary transformations, we can easily derive the closed form of $p^d(\theta)$ as follows:

$$p^d(\theta) = \frac{\Gamma(d/2)}{\Gamma((d-1)/2)} \cdot \frac{\sin^{d-2}(\theta)}{\sqrt{\pi}} \quad (6)$$

, where $\Gamma()$ denotes the Gamma function. According to the proof of Theorem 1 in [21], we have

$$\lim_{d\to\infty} \frac{1}{\sqrt{d - z^2}} \times p^d(\arccos\frac{z}{\sqrt{d}}) = f(z) \quad (7)$$

, where $f(z) = \frac{1}{\sqrt{2\pi}}e^{-z^2/2}$. Then, based on the threshold in collision testing and the collision probability of generated hash functions, we have the following equation:

$$\Pr_{\tilde{x}\sim U(S^d(1))}[\#Col(\tilde{x}, q) \geq \tau] = \int_0^\pi p^d(\theta)F(\tau; m, 1 - \frac{\theta}{\pi})d\theta. \quad (8)$$

Algorithm 1: The Indexing of MQH

Input: $\mathcal{D}$ is the dataset ($x \in \mathcal{D}$); L is the maximum level; $\mathcal{M}$ is the number of sub-codebooks in each level; m is the number of hash functions generated independently in each level;

Generate coarse quantizers in the L_0 level;
for ℓ from 1 to L **do**
    Train the codebook C_ℓ containing $\mathcal{M}$ sub-codebooks in the ℓ-th level;
    Quantize every $r_{\ell-1}(x)$ to $\hat{x}_\ell$ by C_ℓ and compute residual vector $r_\ell(x)$;
    Compute m hash values for every $r_\ell(x)$ and obtain the binary hashing code with length m;
Store quantization codes and hashing codes in all levels;

By (7) and (8), we conclude. □

In practice [21], the difference between the probability on the left-hand side and $G(d, m, \tau)$ is small enough when d exceeds 100. We plotted the curves showing the values of $G(d, m, mP_0 - l_0)$ for different values of d (See Fig. 6(c)(d)). On one hand, the value of function G is predictable by t_0 since, in our proposal, the threshold $mP_0 - l_0$ of the bucket depends on t_0. On the other hand, for high-dimensional datasets, the collision numbers of most of vectors on sphere S^d are around $m/2$, which means that a small t_0 may be enough to ensure a high rejection rate of the collision testing. Both of these two observations imply that, for every data point x, t_0 is a critical parameter controlling the possibility that x passes the collision testing. In the query phase, we will use a coefficient δ to indicate the upper bound of t_0 and thus control the tradeoff between search efficiency and query accuracy. In experiments, we will compare the values of predicted δ and experimentally optimal δ (See Fig. 6).

A toy example: for each point in a given data set, MQH needs to determine if it can be regarded as a candidate. Based on the discussion above, we use a top example to show how MQH works (See Figure 2). Let the number of levels be 3 and m (the number of hash functions) be 16. For a false point x, let $\hat{x}_1$, $\hat{x}_2$ and $\hat{x}_3$ be three quantized vectors of x with quantization error $n_1(x) = 4$, $n_2(x) = 2$ and $n_3(x) = 1$, respectively. In the first level, based on $\hat{x}_1$ and $n_1(x)$, we compute the bucket regarding x as [5, 16] and the collision number of x as 8. Clearly, we cannot prune x since x does not fall outside the bucket. Then, we move to the next level. In the second level, we decrease the quantization error to $n_2(x)$ and consider $\hat{x}_2$. For a similar reason, x is still not rejected by the collision testing in this level. In the third level, the bucket shrinks to [16,16] and the collision number of x is 9, which means that we can now safely prune x with some probability guarantee. We treat every data point in this way. If some point cannot be pruned even in the last level, we regard this point as a promising candidate of P2HNNS.

Algorithm 2: The Query Phase of MQH

Input: q and b are hyperplane parameters; k is the number of returned points; δ is the parameter for the tradeoff of efficiency and accuracy; $\mathcal{D}$ is the dataset ($x \in \mathcal{D}$); L is the maximum level; m is the number of hash functions in each level; C is the set containing k nearest neighbors among current candidates; $Flag$ is an indicator, where $Flag = 1$ for guarantees on recall rates and $Flag = 0$ for guarantees of finding approximate nearest neighbors.

Build multi-level tables for the fast processing of quantization codes;
Find some initial candidates in the L_1 level;
Update $C = \{x_*^{(1)}, \ldots, x_*^{(k)}\}$ and $w_* = s_H(x_*^{(k)})$;
for $x \in \mathcal{D}$ **do**
    for ℓ from 1 to L **do**
        Compute $w_\ell(x) = b - w_* - I(\hat{x}_\ell)$; % $I(\hat{x}_\ell) = I(\hat{x}_{\ell-1}) + \langle \hat{\boldsymbol{x}}_{\boldsymbol{\ell}}, \boldsymbol{q} \rangle$;
        if $w_\ell(x) \le 0$ **then**
            Compute the exact distance of x to H; % *x is a promising candidate in this case*;
            Update C and w_* if necessary;
        if $w_\ell(x) > r_\ell(x)$ **then**
            Turn to process the next data point; % *x is not a top-k point*;
        if *$Flag = 0$ and $w_\ell(x)/r_\ell(x) > \delta$* **then**
            Turn to process the next data point; % *Improve the efficiency by δ*;
        if *($Flag = 1$ and $w_\ell(x)/r_\ell(x) > \delta$) or ($Flag = 0$ and $\ell = L$ and $w_\ell(x)/r_\ell(x) \le \delta$)* **then**
            Compute the collision number of x and q^+ or q^-; % *left for q^+ and right for q^-*;
            if *x passes the collision testing* **then**
                Compute the exact distance of x to H; % *judge if x falls in the bucket*;
                Update C and w_* if necessary;
            Turn to process the next data point; % *skip checking remaining levels*;
return *k points in C*

5 DISCUSSION

5.1 Implementation and Algorithm

Now, we are in a position to give the algorithm of MQH (Alg. 1 and Alg. 2). In the indexing phase, we just take a stepwise quantization process to generate the quantization codes and hashing codes of every data point, as discussed earlier. In the query phase, the choice of w needs additional explanations. In the preceding discussion, we suppose that we know the value of w. Actually, to ensure the correctness of our probability guarantee, we can replace w by any $w_* \ge w$ without making the earlier analysis essentially different, where w_* denotes the distance of the kth nearest neighbor which

we have found to H (Of course, from the perspective of efficiency, we still hope that w_* is as close to w as possible.). Thus, our query phase consists of two sub-phases. In the first sub-phase, we get an approximate value of w, i.e., w_*, by checking a small number of promising candidates, as existing quantization approaches do. In the second sub-phase, we use MQH to deal with the remaining points, that is, to determine if every processed point can be regarded as a candidate. If we find a closer point, we need to update w_* to strengthen the collision testing.

Let $x_{\min}^{(1)}, x_{\min}^{(2)}, \ldots, x_{\min}^{(k)}$ be the true top-k nearest neighbors. By the analysis in the preceding sections, we have the following main result for the guarantees on the returned points, where $w_\ell(x)$ is defined in step 6 in Alg. 2.

Theorem 1. *Given an error rate $\epsilon > 0$, a coefficient $0 < \delta < 1$ and $Flag = 0$. If $(1/\delta - 1)w_\ell(x_{\min}^{(i)}) < w_*$ holds for each level ℓ, the probability that Algorithm 2 finds a point x such that $s_H(x) \leq \frac{w_*}{w_* - (1/\delta - 1)w_\ell(x_{\min}^{(i)})} s_H(x_{\min}^{(i)})$ is at least $1 - \epsilon$ $(1 \leq i \leq k)$. If $Flag = 1$, the probability that $x_{\min}^{(i)}$ is found is at least $1 - \epsilon$ for each i $(1 \leq i \leq k)$.*

Proof. We first consider the case $Flag = 1$. since $w_* \geq w$, by the definitions of t_0, t_1, we know that the bucket size determined by w_* is not smaller than that determined by w. If the condition in step 7 in Algorithm 2 is satisfied, the exact distance of x to H will be computed. If not, according to the observation above and Lemma 3, we know the probability that $x_{\min}^{(i)}$ falls inside the bucket is at least $1 - \epsilon$ $(1 \leq i \leq k)$. Note that, the discussion above applies to any level and the value of δ in this case only determines the terminal level for each point.

Next, we consider the case $Flag = 0$. Since $(1/\delta - 1)w_\ell(x_{\min}^{(i)}) < w_*$, we define $c = \frac{w_*}{w_* - (1/\delta - 1)w_\ell(x_{\min}^{(i)})} > 1$. Then, we consider the following two additional cases when we are processing $x_{\min}^{(i)}$. Case 1: we have found a point x such that $s_H(x) \leq c s_H(x_{\min}^{(i)})$. In this case, whether $x_{\min}^{(i)}$ passes the collision testing or not does not affect our conclusion. Case 2: we have not found a c-approximate nearest neighbor of $x_{\min}^{(i)}$. According to the definition of w_*, this implies that, $c w_{\min}^{(i)} \leq w_*$, where $w_{\min}^{(i)}$ is the distance of $x_{\min}^{(i)}$ to the hyperplane. By the definition of $r_\ell(x_{\min}^{(i)})$ and the definition of c, we have the following relationships:

$$w_{\min}^{(i)} \leq w_* - (1/\delta - 1)w_\ell(x_{\min}^{(i)}) \Rightarrow w_\ell(x_{\min}^{(i)})/r_\ell(x_{\min}^{(i)}) \leq \delta. \quad (9)$$

Therefore, we know that $x_{\min}^{(i)}$ could not be pruned by the condition in step 12 in Algorithm 2. On the other hand, according to the preceding discussion for $Flag = 1$, $x_{\min}^{(i)}$ can pass the collision testing with probability at least $1 - \epsilon$. Therefore, no matter in which case, we can ensure that a c-approximate nearest neighbor of $x_{\min}^{(i)}$ is found with probability at least $1 - \epsilon$. □

By this theorem, we can see that, for $Flag = 0$, δ plays a similar role of approximation ratio c. In addition, by the definitions and the high precision of quantization, $w_\ell(x_{\min}^{(i)})$ is generally very small, which means that the condition stated in the theorem is highly likely to be satisfied even with a small value of δ (For example, $0.3 \leq \delta \leq 0.5$.).

5.2 Complexity Analysis

Next, we analyze the space complexity and the time complexity of MQH. Let n be the data size. In the indexing phase, since we need to execute the stepwise quantization process, the space complexity is $O(nd + \mathcal{M}n + nm/8)$, where in each level, the complexity for storing quantization codes is $O(\mathcal{M}n)$ and the complexity for storing hashing codes is $O(nm/8)$. For the training complexity, since it highly depends on the used quantization approach in each level, we skip its estimation. Nevertheless, we will show that, for large-scale datasets, the training time is actually much smaller than the indexing time by choosing a training-efficient quantization approach. On the other hand, the time complexity for indexing is $O(ndLT + ndmL)$, where $T = 256$ is the number of sub-codewords in each sub-codebook; the complexity for the computation of quantization codes is $O(ndLT)$ and the complexity for the random projection is $O(ndmL)$. Note that, the indexing time can be greatly shorten in a multi-core environment since all data points are encoded independently.

In the query phase, the space complexity is $O(nd + L\mathcal{M}n + Lnm/8)$ since we need to store the quantization and hashing codes in all levels for the level determination. On the other hand, we also need to store the original dataset since the exact distance computation is indispensable for the top-k search. The time complexity for searching is $O(\alpha L\mathcal{M}n + nm + \beta nd)$. Here $O(\alpha L\mathcal{M}n)$ denotes the time for the examination of quantization codes, where αL $(0 < \alpha \leq 1)$ is the average terminal level; $O(nm)$ denotes the time for the counting of 1's of hashing codes, which can be executed efficiently by using suitable instructions; βn denotes the number of candidates passing the collision testing. In the scenario of isotropic distributions, $\beta \doteq G(d, m, mP_0 - l_0)$ if the terminal levels of all data points can be bounded by $L - 1$. Note that both α and β are controllable. In fact, with the existence of L_0 level, we can feed only a small subset of the original dataset into our model such that α can be small enough, although in this case, our guarantee only holds for the input subset. As for β, it implicitly depends on the user-specified parameter δ. Theoretically, β can be arbitrarily small if δ was set to be close to 0 enough. On the other hand, if we normally run MQH with the guarantees for approximate solutions (That is, we do not use the heuristic strategies above.), α is generally smaller than $0.3/L$ and β is generally smaller than 0.01 for a practical value of δ on real datasets (See Table 5). Since $\mathcal{M} \ll d$ on high-dimensional datasets , the complexity of our algorithm is much lower than that of the linear scan.

5.3 Handling Updates

By the discussion above, it is easy to see that, after determining the codebook in each level, the binary codes of all data points are generated independently with each other. If the domain of data points does not change significantly, which occurs rarely in practice, we do not need to retrain the codebooks and can thus easily handle the updates of data points. Specifically, for a data point to be inserted or to be modified, the time for computing its quantization codes is $O(TLd)$ and the time for computing its hash values is $O(mLd)$,

Table 2: The statistics of five used datasets.

Dataset	#Points	Dimension	Data size	Type
Music	1,000,000	100	0.38 GB	Rating
Glove	1,183,514	100	0.45 GB	Text
Tiny1M	1,000,000	384	1.43 GB	Image
Deep10M	10,000,000	96	3.88 GB	Image
Deep100M	100,000,000	96	38.8 GB	Image

Table 3: Indexing time (including the training time) and Index size. BH and MH have been outperformed by NH and FH in [15]. In addition, all approaches except for MQH crashed on Deep100M due to overlarge memory costs in the indexing phase on our PC.

Dataset	Index Size (GB)			Indexing Time (s)		
	NH	FH	MQH	NH	FH	MQH
Music	2.9	2.2	0.53	285	114	789
Glove	3.5	2.6	0.73	342	130	846
Tiny1M	3.0	5.2	1.67	1069	931	3702
Deep10M	29.3	21.4	5.1	1123	1129	2740
Deep100M	\	\	51.5	\	\	22324

where $T = 256$. For the deletion, the time is $O(1)$ because of the independency of codes.

6 EXPERIMENTS

All experiments were performed on a PC with Intel(R) Xeon(R) Gold 6285V CPU@2.70Ghz with 157GB memory, running in Ubuntu 18.04. Our code and all used datasets are available on the Github[1].

6.1 Experimental Setup

We chose five real datasets: Music (dim: 100; size: 1M), Glove (dim: 100, size: 1.2M), Tiny1M (dim: 384; size: 1M), Deep10M (dim: 96; size:10M) and Deep100M (dim: 96; size:100M), where Tiny1M was extracted from Tiny80M [31]; Deep10M and Deep100M were extracted from Deep1B (Note that, the norms of data points in Deep are all 1.). The details of used datasets can be found in Table 2. For each dataset, the number of queries was fixed to 100, as in [15].

Next we discuss the choice of quantization approach in MQH. Based on the analysis in Sec. 4, we know that any MCQ approach can apply to the proposed framework. In this paper, we chose NE-RQ [9] for the following reason. According to Definition (1), the selected quantization approach should be suited to the estimation of inner product, and NEQ [9] has been proven to be a competitive approach for this purpose. Among various NEQ variants, NE-RQ can achieve the best performance with a comparatively small training time. Therefore, in the following experiments, we incorporate NE-RQ into the stepwise quantization process in MQH. It is notable that, some quantization approaches proposed for *Maximum Inner Product Search* (MIPS), such as ScaNN [14], are not suited to P2HNNS due to essentially different objective functions.

We first consider existing P2HNNS solutions and choose the following five LSH-based approaches.

[1]https://github.com/LUKEJING/MQH

- **BH [33].** K was set to {2, 4, 6, 8, 10} and L was set to {8, 16, 32, 64, 128, 256}. We chose the best search results under all the combinations of parameters.
- **MH [34].** We set the number of random projections to {4,8,16} and set other (common) parameters to the same values in BH.
- **NH [15].** The number of hash functions was set to {8, 16, 32, 64, 128, 256} and λ was set to $d, 2d, 4d, 8d$, as in its paper. The other parameters were set to the default values in its original code.
- **FH [15].** The setting of λ and the number of hash functions were the same with those of NH. Parameter t was set to {2, 4, 6, 8, 10} as suggested in its paper.
- **MQH.** We generated four levels ($L = 4$). Actually, we will show that only a very small portion of data points could reach the fourth level (See Table 5). Thus, $L = 4$ is enough for most of real datasets. Besides, in each level, the number of sub-codebooks was set to 16 ($\mathcal{M} = 16$) which is a standard setting in many quantization-based approaches. Actually, the concrete value of $\mathcal{M}$ is not very important since our structure can automatically determine the total number of sub-codebooks which need to be processed. The number of hash functions was set to 64 ($m = 64$) since the binary code of such length (64 bits) can be processed efficiently on our PC. l_0 was recommended to set to 5 on Tiny1M and set to 3 on the other datasets (ϵ was thus fixed.). δ was generally taken in [0,2, 0.5] for achieving different tradeoffs between efficiency and accuracy (for approximate solutions). The performance of MQH under different settings of l_0 and δ will be discussed in Sec. 6.5. The number of training samples was fixed to 100K.

6.2 Index Size and Indexing Time

Since the number of hash tables for BH, MH, and the number of hash functions for NH, FH were chosen in a large range, the indexing times and index sizes of compared methods were not fixed on each dataset. On the other hand, BH and MH have been outperformed by NH and FH in these two performance metrics, as reported in [15]. Thus, we focus on the comparison among MQH, NH and FH with the number of hash functions fixed, that is, $m = 256$. Actually, this is the recommended setting for NH, and FH under $m = 256$ can perform slightly better than it under other values of m, as shown in [15]. Note that the total number of hash functions for MQH is also 256 since there are 4 levels in each of which 64 hash functions are built.

Table 3 shows the index sizes and the indexing times of compared approaches. Since BH, MH, FH, NH crashed on Deep100M in the indexing phase even with the smallest feasible value of each parameter, we focus on the comparison results on the other four datasets. We can see that, for $m = 256$, MQH has the smallest index size among all LSH-based approaches. This is because, compared with FH and NH, every data point in MQH is represented by a binary code rather than a group of integers. This might be the reason why only MQH could work on Deep100M on our PC.

On the other hand, MQH generally requires more indexing time than the other compared approaches. This is because MQH needs additional training time for the stepwise quantization. Note that, although the indexing time of MQH is obviously larger than those of NH and FH on 1M-scale datasets, the difference decreases as the

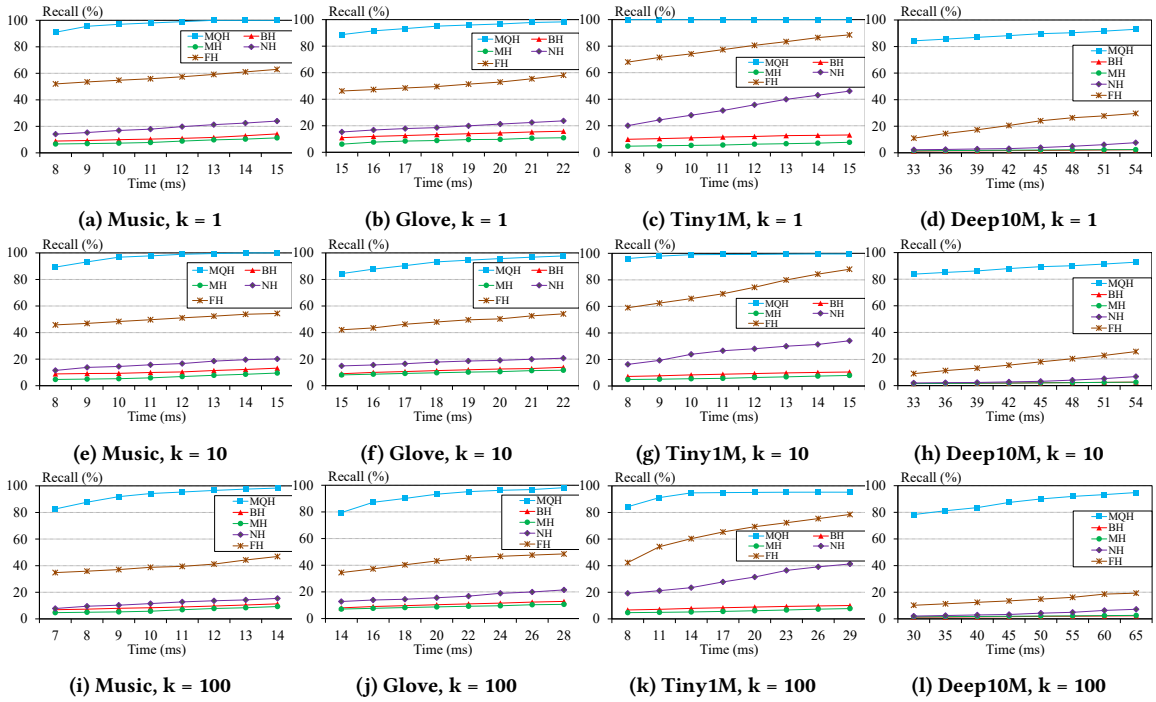


Figure 3: The performances of compared approaches on four real datasets (Normalized case).

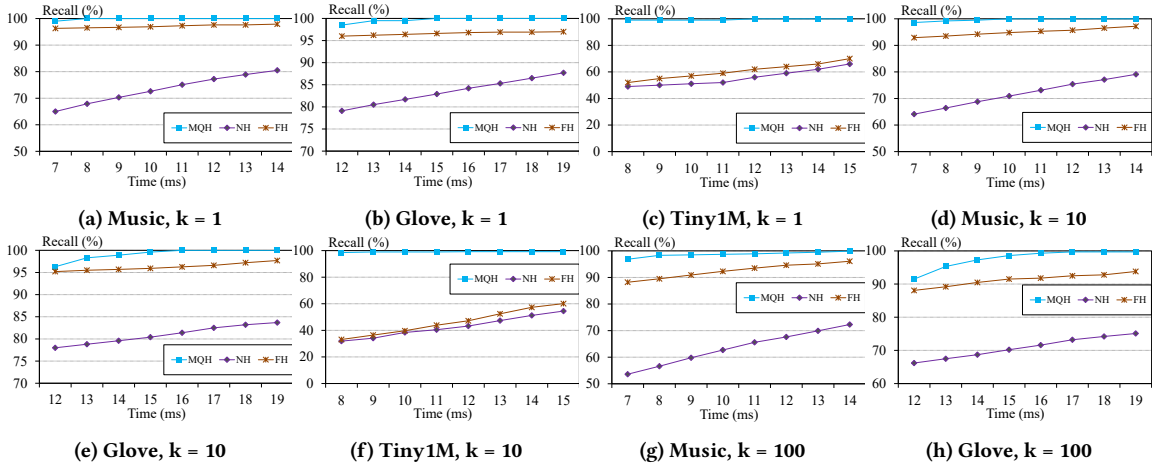


Figure 4: The results on original datasets (Non-normalized case).

data size grows (on Deep10M). This implies that the training time does not dominate in the total indexing time on large-scale datasets. On the other hand, since the data points are indexed independently in MQH, we can easily accelerate the indexing process in a multi-thread regime.

6.3 Comparison on Search Performances

6.3.1 Results for Approximate Solutions on Normalized Datasets. We set *Flag* to 0 in MQH and show the search performances of compared approaches on normalized datasets in Fig. 3 (Specifically, we normalize Music, Glove and Tiny1M such that the norm of every data point is fixed to 1). Actually, original BH and MH can only apply to the dataset whose data points locate on a sphere, which occur frequently in the domain of active learning. From the results, we can see that, MQH performs much better than the other approaches. Specifically, for each target running time, the recall rate of MQH is 20%-70% larger than that of the state-of-the-art LSH-based approach FH, which shows the superiority of the hashing scheme built on quantization errors.

Table 4: MQH($Flag = 1$) vs. FH in various cases. Here Deep denotes Deep10M.

Data	Normalized dataset								Original dataset							
	k = 10				k = 100				k = 10				k = 100			
	Recall(%)		Time(ms)		Recall(%)		Time(ms)		Recall(%)		Time(ms)		Recall(%)		Time(ms)	
	MQH	FH	MQH	FH	MQH	FH	MQH	FH	MQH	FH	MQH	FH	MQH	FH	MQH	FH
Music	99.3	73.1	36.6	41.6	99.7	66.3	40.6	41.6	99.5	99.4	30.9	35.0	99.1	99.1	30.4	42.3
Glove	98.5	80.2	58.7	90.3	99.7	80.3	63.4	90.4	100.0	99.6	52.1	44.1	99.8	99.8	52.0	90.8
Tiny	98.0	97.4	68.1	63.5	97.6	96.7	84.3	101	98.0	89.7	62.0	63.7	98.0	93.8	75.3	122
Deep	98.9	51.2	417	472	99.9	44.5	447	472	\	\	\	\	\	\	\	\

Table 5: The studies of α (Processing ratio) and β (Passing ratio), $k = 100$

Dataset	Processing ratio (‰)				Passing ratio (‰)				Recall rate (%)			
	$L = 1$	$L = 2$	$L = 3$	$L = 4$	$L = 1$	$L = 2$	$L = 3$	$L = 4$	$L = 1$	$L = 2$	$L = 3$	$L = 4$
Music	46.0	2.9	0.4	0.09	4.81	0.57	0.12	0.04	94.4	96.3	98.2	99.2
Glove	34.6	1.7	0.2	0.05	6.35	0.66	0.13	0.04	92.2	95.3	97.9	99.2
Tiny1M	167	167	81.9	39.6	80.3	45.4	25.6	14.1	92.1	93.2	93.7	95.3
Deep10M	10.0	0.31	0.02	0.004	1.17	0.07	0.01	0.003	93.1	95.8	97.9	98.8
Deep100M	1.7	1.7	0.12	0.001	0.35	0.02	0.002	0.001	89.7	92.8	95.7	97.1

6.3.2 Results for Approximate Solutions on Original Datasets. Although in many cases, P2HNNS is applied to normalized datasets, it also has some applications, such as dimension reduction, on non-normalized datasets. Therefore, we compared MQH with FH and NH on the original datasets of Music, Glove and Tiny1M (See Fig. 4 and Fig. 7-(a)). From the results, we can see that, MQH still performs better than FH and NH, especially on high-dimensional dataset Tiny1M. This shows that the performance of MQH is not sensitive to the norm distributions of datasets.

6.3.3 Results for Guarantees on Recall Rates. In the discussion above, we have shown the performances of MQH under $Flag = 0$. Now, we focus on the case $Flag = 1$, in which MQH owns a probability guarantee on recall rates. As for the choice of parameters, with the other parameters unchanged, we set δ to 0.5 for the reason which will be discussed in Sec. 6.5. Since the other compared methods do not have such strong guarantees on recall rates and FH has shown the best performance among four benchmark approaches in the experiments above, we focus on the comparison between MQH under $Flag = 1$ and FH under its common settings for approximate solutions. The results are shown in Table 4.

From the results, we have the following observations:

(1) In the case of normalized datasets, compared with FH, MQH required less running time to achieve higher recall rates on each dataset. This shows that, MQH could apply to those applications which need guarantees on recall rates with an acceptable efficiency.

(2) Still for the case of normalized dataset, compared with MQH ($Flag = 0$), MQH ($Flag = 1$) can achieve a slightly higher recall rate at the expense of much more running time. An important reason for such results is that, MQH ($Flag = 0$) determines those points which are highly unlikely to pass the collision testing beforehand by parameter δ and skips their collision testings to improve the efficiency. Thus, for those applications which focus on both of the efficiency and the accuracy, we recommend users to choose MQH with $Flag = 0$ and use δ to achieve a desired tradeoff.

Table 6: MQH vs. Vanilla NEQ ($k = 100$). X in V_X denotes the number of sub-codebooks in Vanilla NEQ.

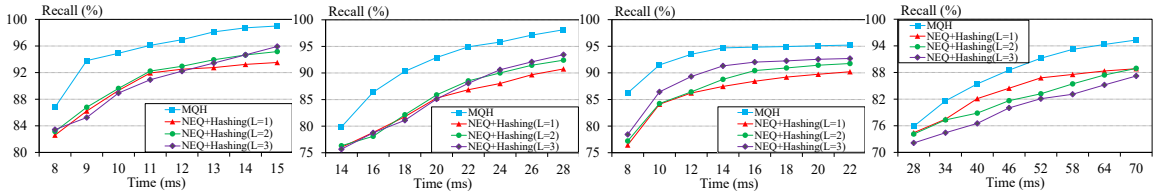
Dataset	Running time (ms)			Recall rate (%)		
	MQH	V_{16}	V_{32}	MQH	V_{16}	V_{32}
Music	13.4	24.9	29.9	98.7	78.6	98.3
Glove	31.5	36.0	42.5	98.9	76.9	98.4
Tiny1M	13.6	24.4	29.6	93.8	66.6	82.6
Deep10M	128.8	184.2	229.5	98.9	69.7	98.6
Deep100M	764.1	1274	1762	97.1	55.3	96.5

(3) In the case of original datasets, we can see that, on Glove ($k = 10$), the performances of FH and MQH are very close, while on the other datasets, MQH outperforms FH.

6.4 Ablation Study

Obviously, the discussion in Sec. 3 can lead to an approach with a straightforward combination of quantization and hashing, which also owns probability guarantees, as in the analysis of MQH. In order to demonstrate the necessity of stepwise quantization more clearly, we introduce an approach called NEQ+Hashing. Its information is shown below. In addition, since we incorporate NEQ in MQH, we also take NEQ (with some modification) as a baseline method for P2HNNS.

- **Vanilla NEQ.** Traditional NEQ was proposed to solve MIPS. Here, we adjust it for P2HNNS. Specifically, we follow NEQ to quantize all data points and estimate the inner product in (1). Then, we sort data points and compute the exact distances of some top points to the hyperplane. The number of sub-codebooks was set to {16, 32} and the size of candidate set was set to 1000.
- **NEQ+Hashing.** To show the necessity of dynamic level determination in MQH, we introduce NEQ+Hashing. It can be regarded as a straightforward combination of NEQ and LSH. Specifically, by choosing a value of L, we only compute the residual vector of



(a) Music, k = 100 (b) Glove, k = 100 (c) Tiny1M, k = 100 (d) Deep10M, k = 100

Figure 5: MQH vs. NEQ+Hashing, where L denotes the terminal level. Due to the large computational cost of quantization codes, NEQ+Hashing under $L \geq 4$ performed worse than that under $L = 3$ on each dataset.

each data point in the L-th level and then compute corresponding hash values. That is, compared with MQH, the terminal level for each data point in NEQ+Hashing was set to the same value. Thus, for each data point, we would not skip the examination of its remaining sub-codebooks until the terminal level is reached. Similar to MQH, we use δ to adjust its tradeoffs between efficiency and accuracy.

Table 6 shows the comparison results of MQH and Vanilla NEQ. We can see that, compared with NEQ, MQH can achieve higher recall rates with less running times. This is because MQH dynamically and automatically determines the required length of quantization code for each data point to gain the efficiency, while the length of code in NEQ should be fixed to a unified value beforehand.

Fig. 5 and Fig. 7-(b) show the comparison results of MQH and NEQ+Hashing. We can see that MQH outperforms NEQ+Hashing with a 5%-8% improvement on the recall rate for each target running time. This shows that, the optimal terminal levels of data points may be very different with each other, and the level determination in MQH is helpful for the improvement of query accuracy.

6.5 Coefficients and Parameters

There are two coefficients which indicate the performance of MQH: α regarding the total number of sub-codebooks which need to be processed in the query phase, and β regarding the size of candidate set. The details of both of these two coefficients can be found in Sec. 5. In addition, although several internal parameters, such as δ, l_0, have theoretical implications, their values may affect the performance of MQH. In Sec. 6.5, we also study their impact.

6.5.1 The Effect of α and β. In the analysis of time complexity, we introduced two parameters α and β, and claimed that both of them were much smaller than 1. In Table 5, we explicitly compute the values of α and β on each dataset. From the results, we can see that, thanks to the inverted index list in the L_0 level, only 5% - 16% data points could enter the L_1 level, which means that the other data points are pruned without affording any computational cost on their quantization or hashing codes. Besides, the number of processed data points decreases significantly as the level goes deeper. With these two observations, it is easy to see that the value of α could be generally smaller than 0.1.

As for β, which is indicated by the passing ratio, we can see that its trend is very similar to that of α. Even with L_1 level only, more than 95% data points cannot pass the collision testing on each dataset except for Tiny1M, which means that their exact distances to the query hyperplane do not need to be computed. When we use four levels, the passing ratio can be further reduced to 0.01.

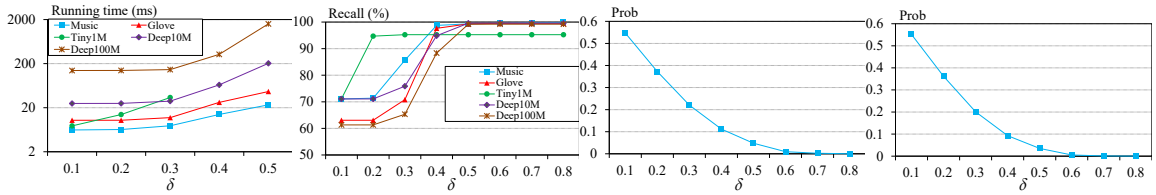
6.5.2 The Effect of δ. δ controls the tradeoff between efficiency and accuracy. We consider both cases of $Flag = 0$ and $Flag = 1$. In addition, we compute the predicted value of δ and compare it with the experimental value.

By Fig. 6, we can see that the practical effect of δ is almost consistent with the theoretical effect of δ, which is reflected in the following two aspects. (1) $\delta \leq 0.5$ is enough for achieving high recall rates since for $\delta > 0.5$, the theoretical probability of passing the collision testing is very close to 0. (2) δ is suggested to take value in $[0.2, 0.5]$ and the theoretical probability changes rapidly as δ varies in this interval.

In the theoretical analysis, we have explained how to improve the search efficiency of MQH by δ in the case $Flag = 0$. Here, we want to show that, δ is also an important parameter for $Flag = 1$. Note that, the implication of δ under $Flag = 1$ is slightly different from that under $Flag = 0$. Actually, as discussed earlier, the value of δ does not affect the probability guarantee on recall rates for $Flag = 1$. However, the earlier analysis still applies to this case. From the results in Table 7, we can see that, as δ increases from 0.5, the recall rate almost remains unchanged and the running time still increases, which implies that $\delta = 0.5$ is a reasonable upper threshold for δ, which is consistent with our theoretical prediction.

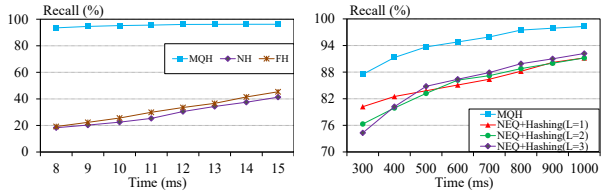
6.5.3 The Effects of l_0. Except for δ, another important parameter is l_0, which controls the size of error rate ϵ. Generally, the recall rate increases as ϵ decreases, that is, as l_0 increases. From the results in Table 8, we can see that $[3, 5]$ is a reasonable interval for l_0 since the recall rates can be high enough if l_0 takes a value in this interval.

6.5.4 The Effects of Initial Candidate Set. In Algorithm 2 of MQH, we have shown that MQH first needs to determine a small candidate set to get a value of w_*. Here we want to show that the performance of MQH is not sensitive to the size of such initial candidate set. In Table 9, we set the size to {1000, 2000, 5000}. We can see that, the performances of MQH under such parameters are generally close. This is because, we need to use MQH to deal with all remaining data points after this step and a roughly approximate value w_*, which is obtained by coarse quantizers in the L_0 level, can work well enough in our proposal.



(a) δ vs. Running time (b) δ vs. Recall rate (c) Probabilities of passing collision testing (dim=100) (d) Probabilities of passing collision testing (dim=384)

Figure 6: The effect of δ (the upper bound of t_0) under $Flag = 0$. The theoretical probabilities of passing the collision testing are obtained based on isotropic distributions.



(a) Results on original Tiny1M, k = 100 (b) Deep100M in Figure 5

Figure 7: Supplemental results to Figures 4 and 5

Table 7: The effect of δ, $k = 100$, $l_0 = 5$ and $Flag = 1$.

δ	Recall rate (%)			Running time (ms)		
	Glove	Tiny	Deep10M	Glove	Tiny	Deep10M
0.1	88.8	94.0	93.3	44.7	74.6	378.7
0.3	96.7	96.2	95.4	45.1	69.0	385.0
0.5	99.6	96.2	99.6	48.7	63.6	400.6
0.7	99.6	96.2	99.6	55.5	70.0	455.1
0.9	99.6	96.2	99.6	60.9	73.5	494.8

Table 8: The effect of l_0, $k = 100$, $\delta = 0.5$ and $Flag = 1$.

l_0	Recall rate (%)			Running time (ms)		
	Glove	Tiny	Deep10M	Glove	Tiny	Deep10M
1	99.2	93.7	99.2	47.4	58.8	345.3
3	99.6	96.2	99.6	48.7	63.5	343.1
5	99.8	98.0	99.9	50.2	75.3	399.3
7	99.9	99.2	100.0	58.5	95.3	497.8

Table 9: The effect of the size of initial candidate set, $k = 100$, $l_0 = 3$, $\delta = 0.4$ and $Flag = 0$.

Dataset	Recall rate (%)			Running time (ms)		
	1000	2000	5000	1000	2000	5000
Music	99.4	99.4	99.5	8.44	9.1	11.7
Glove	99.3	99.3	99.3	17.2	18.1	21.0
Tiny1M	95.1	96.2	97.5	41.8	42.7	44.9
Deep10M	94.6	94.9	95.1	68.1	70.9	77.5

7 CONCLUSION

In this paper, to solve P2HNNS, we build a hierarchical structure called MQH which can determine if the current residual vector is suitable for hashing. Theoretical analysis shows that MQH not only enjoys guarantees on query results, but also automatically determines the required length of code for each point to ensure the high efficiency. Experiments on real datasets confirm the superiority of MQH.

ACKNOWLEDGMENTS

This work was supported by KAKENHI (21H03555, 22H03594, 22H03903).

REFERENCES

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008.
[2] A. Babenko and V. S. Lempitsky. Additive quantization for extreme vector compression. In *CVPR*, pages 931–938, 2014.
[3] A. Babenko and V. S. Lempitsky. Efficient indexing of billion-scale datasets of deep descriptors. In *CVPR*, pages 2055–2063, 2016.
[4] J. L. Bentley. K-d trees for semidynamic point sets. In *SCG*, pages 187–197, 1990.
[5] C. Z. Bin Zhao, Fei Wang. Efficient maximum margin clustering via cutting plane algorithm. In *SDM*, pages 751–762, 2008.
[6] C. X. Y. R. Chang Xu, Dacheng Tao. Large-margin weakly supervised dimensionality reduction. In *ICML*, pages 865–873, 2014.
[7] M. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002.
[8] Y. Chen, T. Guan, and C. Wang. Approximate nearest neighbor search by residual vector quantization. *Sensors*, 10(12):11259–11273, 2010.
[9] X. Dai, X. Yan, K. K. W. Ng, J. Liu, and J. Cheng. Norm-explicit quantization: Improving vector quantization for maximum inner product search. In *AAAI*, 2020.
[10] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SoCG*, pages 253–262, 2004.
[11] C. Fu, C. Wang, and D. Cai. High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility. *IEEE Trans. Pattern Anal. Mach. Intell*, 2022.
[12] J. Gan, J. Feng, Q. Fang, and W. Ng. Locality-sensitive hashing scheme based on dynamic collision counting. In *SIGMOD*, pages 541–552, 2012.
[13] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization. *IEEE Trans. Pattern Anal. Mach. Intell*, 36(4):744–755, 2014.
[14] R. Guo, P. Sun, E. Lindgren, Q. Geng, D. Simcha, F. Chern, and S. Kumar. Accelerating large-scale inference with anisotropic vector quantization. In *ICML*, pages 3887–3896, 2020.
[15] Q. Huang, Y. Lei, and A. K. H. Tung. Point-to-hyperplane nearest neighbor search beyond the unit hypersphere. In *SIGMOD*, pages 777–789, 2021.
[16] P. Indyk and R. Motwani:. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *STOC*, pages 604–613, 1998.
[17] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Anal. Mach. Intell*, 33(1):117–128, 2011.
[18] K. Lu and M. Kudo. R2LSH: A nearest neighbor search scheme based on two-dimensional projected spaces. In *ICDE*, pages 1045–1056, 2020.
[19] K. Lu, M. Kudo, C. Xiao, and Y. Ishikawa. HVS: Hierarchical graph structure based on voronoi diagrams for solving approximate nearest neighbor search. *Proc. VLDB Endow.*, 15(2):246–258, 2021.

[20] K. Lu, H. Wang, W. Wang, and M. Kudo. VHP: Approximate nearest neighbor search via virtual hypersphere partitioning. *Proc. VLDB Endow.*, 13(9):1443–1455, 2020.

[21] K. Lu, H. Wang, Y. Xiao, and H. Song. Why locality sensitive hashing works: A practical perspective. *Inf. Process. Lett.*, 136:49–58, 2018.

[22] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans. Pattern Anal. Mach. Intell*, 42(4):824–836, 2020.

[23] J. Martinez, hobhit Zakhmi, H. H. Hoos, and J. J. Little. Lsq++: Lower running time and higher recall in multi-codebook quantization. In *ECCV*, pages 508–523, 2018.

[24] K. G. Prateek Jain, Sudheendra Vijayanarasimhan. Hashing hyperplane queries to near points with applications to large-scale active learning. In *NeurIPS*, pages 928–936, 2010.

[25] P. Ram and A. G. Gray. Maximum inner-product search using cone trees. In *KDD*, pages 931–939, 2012.

[26] M. J. Saberian, J. C. Pereira, N. Vasconcelos, and C. Xu. Large margin discriminant dimensionality reduction in prediction space. In *NeurIPS*, pages 1488–1496, 2016.

[27] G. Schohn and D. Cohn. Less is more: Active learning with support vector machines. In *ICML*, pages 839–846, 2000.

[28] K. G. Sudheendra Vijayanarasimhan, Prateek Jain. Hashing hyperplane queries to near points with applications to large-scale active learning. *IEEE Trans. Pattern Anal. Mach. Intell*, 36(2):276–288, 2014.

[29] Z.-H. Z. Teng Zhang. Optimal margin distribution clustering. In *AAAI*, pages 4474–4481, 2018.

[30] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *J. Mach. Learn. Res*, 2:45–66, 2001.

[31] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell*, 30(11):1958–1970, 2008.

[32] S. Vijayanarasimhan and K. Grauman. Large-scale live active learning: Training object detectors with crawled data and crowds. In *CVPR*, pages 1449–1456, 2011.

[33] J. W. Wei Liu, Y. Mu, S. Kumar, and S.-F. Chang. Compact hyperplane hashing with bilinear functions. In *ICML*, pages 467–474, 2012.

[34] X. F. Xianglong Liu, C. Deng, Z. Li, H. Su, and D. Tao. Multilinear hyperplane hashing. In *CVPR*, pages 5119–5127, 2016.