IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 29, 2019, accepted February 16, 2019, date of publication February 27, 2019, date of current version April 3, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2900730

A New Cell-Level Search Based Non-Exhaustive Approximate Nearest Neighbor (ANN) Search Algorithm in the Framework of Product Quantization

YANG WANG[1,2], ZHIBIN PAN[1,3], AND RUI LI[1]

[1]School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China
[2]Research Institute of Xi'an Jiaotong University, Zhejiang, Hangzhou 311215, China
[3]Key Laboratory of Spectral Imaging Technology, Chinese Academy of Sciences, Xi'an 710119, China

Corresponding author: Zhibin Pan (zbpan@xjtu.edu.cn)

This work was supported in part by the Open Project Program of the National Laboratory of Pattern Recognition (NLPR) under Grant 201800030, in part by the Open Research Fund of the Key Laboratory of Spectral Imaging Technology, Chinese Academy of Sciences, under Grant LSIT201606D, and in part by the Major Science Program of Xiaoshan District, Hangzhou, Zhejiang, under Grant 2018225.

ABSTRACT Non-exhaustive search is widely used in the approximate nearest neighbor (ANN) search. In this paper, we propose a new cell-level search-based non-exhaustive ANN search algorithm in the framework of product quantization (PQ) called cell-level PQ to speed up the ANN search. The cell-level search is introduced by searching all the PQ cells of the query vector on the cell level, and the length of the candidate list can be significantly reduced with negligible computational costs. Instead of using the high time-consuming coarse quantizers, which are necessary in all of the existing non-exhaustive ANN search algorithms such as inverted files (IVFs) and inverted multi-index (IMI), our proposed cell-level PQ reuses the PQ cells of query vector to reject database vectors so that the ANN search in the framework of PQ can be efficiently speeded up. In addition, because our proposed cell-level PQ does not need to store the auxiliary indexes of coarse quantizers for each database vector, no extra memory consumption is required. The experimental results on different databases demonstrate that our proposed cell-level PQ can significantly speed up the ANN search in the framework of PQ, and meanwhile, the search accuracy is almost the same as that of the standard PQ.

INDEX TERMS Approximate nearest neighbor search, vector-level search, cell-level search, product quantization, partial distance search.

I. INTRODUCTION

With the development of multimedia technology, content-based image retrieval technique has been widely used in many applications, such as face retrieval [1], object recognition [2], image matching [3] and image classification [4]. In the state-of-the-art image retrieval systems, the feature vectors are extracted at first as a pre-processing step from both query and reference images. Then, the feature vectors of query images are on-line searched in the database. A fundamental task in the on-line searching is the nearest neighbor (NN) search, which is defined as follows:

The associate editor coordinating the review of this manuscript and approving it for publication was Wen Chen.

For a query vector and a corresponding database, the task of NN search is to return the nearest vector which has the smallest Euclidean distance from the current query vector to the database. For a small and low-dimensional database, exhaustively searching the database vector-by-vector to exactly find the nearest vector of the query vector is feasible. In this paper, we call the technique of vector-by-vector searching as the vector-level search.

However, exhaustive search on vector-level requires the calculations of real Euclidean distances between a high-dimensional query vector and all candidate vectors in a high-dimensional database, which is extremely time-consuming. To solve this problem, many fast approximate nearest neighbor (ANN) search algorithms for high-dimensional

database, such as tree-based algorithms [5], [6], Hash-based algorithms [7], [8] and vector quantization (VQ)-based algorithms [9], [10] are proposed. ANN algorithms relax the condition of the exact NN search and try to quickly find a small set of several approximate nearest database vectors instead of only one exact nearest database vector. For this reason, ANN is a trade-off between search accuracy and computational costs.

Vector quantization (VQ) [11] is a compact and popular encoding technique for ANN search applications. Vector quantizer employs a pre-trained codebook and divides the database into a number of Voronoi cells. Then, each longer database vector can be efficiently encoded by using the shorter index of its nearest codeword in the pre-trained codebook.

Because VQ only needs to store the shorter indexes of nearest codewords instead of the original database vectors, and each index of VQ usually has only a few bits, the memory consumption of storing the encoded database can also be efficiently reduced.

However, VQ is ineffective for high-dimensional databases. Product quantization (PQ) [9] is a widely used ANN search algorithm, which can provide efficient quantization performance on high-dimensional databases. The core idea of PQ is to encode the high-dimensional vector space by the Cartesian product of several low-dimensional sub-spaces, then PQ independently quantizes them by using the corresponding low-dimensional VQ quantizers.

To further improve the search speed of PQ, several non-exhaustive algorithms, such as the inverted index (also known as inverted files, IVF) [12] and the inverted multi-index (IMI) [13] are proposed.

IVF returns a short list of database vectors which are close to the query vector x. This short list is called as a candidate list and used to search the approximate nearest neighbors of the query vector x. In fact, IVF is a coarse VQ quantizer which firstly quantizes the database into k' cells. When inputting a query vector, IVF selects the database vectors in the first w ($w \ll k'$) nearest cells as the candidate list. Instead of exhaustively searching the database vector-by-vector (i.e., vector-level search), IVF only searches the database vectors in the candidate list so that only a small amount of Euclidean distances between query vector and all database vectors in the candidate list need to be calculated. IVF extracts the candidate list by searching the coarsely quantized database cell-by-cell. In this paper, we call the technique of searching cell-by-cell as the cell-level search. Because cell-level search only requires to calculate k' ($k' \ll n$, n is the number of database vectors) real Euclidean distances between each query vector and the centroids of cells, the cell-level search must be more efficient than vector-level search. By using cell-level search in IVF, the ANN search can be significantly speeded up.

On the other hand, IMI is a typical inverted index-based non-exhaustive search algorithm which uses two sub-space PQ quantizers instead of one VQ quantizer as the coarse quantizer. Because IMI can provide much more cells than IVF, IMI is able to use a much shorter candidate list than that of IVF to achieve the same level of search accuracy. Obviously, IMI is also a kind of cell-level search algorithm. However, because IMI needs to store two indexes based on using two sub-quantizers for each database vector, the memory consumption of IMI is far more than that of IVF.

Inverted index-based non-exhaustive search algorithms focus on solving how to generate an efficient inverted file system so as to extract a high-quality and short candidate list of database vectors. By using cell-level search instead of vector-level search, a majority of real Euclidean distance calculations between query vector and all database vectors can be avoided. However, inverted index-based algorithms need to store the auxiliary indexes of coarse quantizer, which is usually a non-negligible large memory consumption. In addition, the computational costs of inverted index-based non-exhaustive search algorithms cannot be neglected as well in many cases.

In this paper, we propose a new cell-level search based non-exhaustive ANN search algorithm in the framework of PQ called Cell-level PQ to speed up the ANN search. Different from the inverted index-based algorithms, our proposed Cell-level PQ completely avoids using the coarse quantizer. And it generates the candidate list by reusing the cells of PQ quantizer to find the cells which are far enough from the query vector so that all the database vectors in these cells can be rejected.

Compared with inverted index-based algorithms, there are two advantages in our proposed Cell-level PQ. Firstly, our proposed Cell-level PQ can efficiently speed up the ANN search in the framework of PQ meanwhile keeping almost the same search accuracy of PQ. Without using high time-consuming coarse quantizers, Cell-level PQ achieves better search performance than the inverted index-based algorithms. Secondly, Cell-level PQ requires no extra memory consumption. In contrast, inverted index-based algorithms need a large amount of extra memory to store the auxiliary indexes of coarse quantizers.

II. RELATE WORK

The fast ANN search is an important task in content-based image retrieval and many other fields of information sciences. There are many works which have been proposed in the past decades. In this section, we briefly describe several related algorithms and analyze their performances.

A. VECTOR QUANTIZATION (VQ)

VQ [11] is a widely used low-bit-rate data compression technique, which can be regarded as a mapping from a D-dimensional space to a finite set, which is called a codebook. The codebook $C = \{CW_1, CW_2, \ldots, CW_k\}$ is pre-trained by the well-known LBG algorithm [11] and the centroids of trained cells are called codewords, where CW_i is the i-th codeword in codebook C and k is the size of C.

In the encoding procedure, VQ finds the nearest codeword of an input vector x in the pre-trained codebook C as below.

$$d(x, CW_{bm}) = \min_{CW_i \in C} d(x, CW_i), \tag{1}$$

where CW_{bm} is the best match codeword of x in the pre-trained codebook C and its index is bm. $d(x, CW_i)$ outputs the Euclidean distance between two vectors x and CW_i. Then, the input vector x is encoded by using the index bm of its nearest codeword CW_{bm}.

VQ has also be used in ANN search. For a database $Y = \{y_1, y_2, \ldots, y_i, \ldots, y_n\}$, where $y_i = (y_i^1, y_i^2, \ldots, y_i^D)$, $y_i \in R^D$ is the i-th D-dimensional feature vector and n is the size of the database, VQ encodes the database vectors and produces a n-size index table for database Y. Each element of this index table is the codeword index of the corresponding database vector y_i. Because the memory bits for storing the index table are far less than the bits used to straightforwardly store the original database vectors, the database Y can be efficiently compressed.

In the decoding procedure, a simple table look-up technique is used and a VQ compressed database can be easily reconstructed by using the index table and the codebook. The flowchart of VQ is shown in Fig. 1.

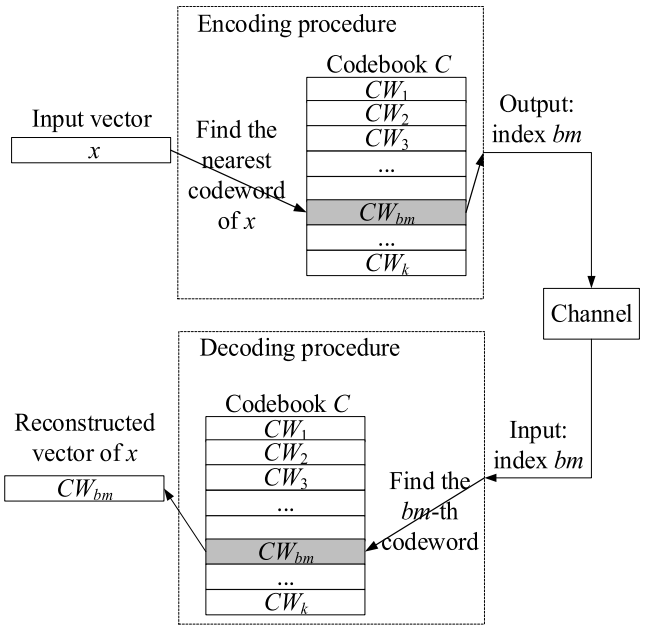


FIGURE 1. The flowchart of VQ.

B. PRODUCT QUANTIZATION (PQ)

Although VQ is popularly and successfully used in ANN search applications, VQ is inefficient for high-dimensional and large-scale databases. To overcome this drawback of VQ, PQ is proposed in [9] and has been proved to be an efficient solution for encoding high-dimensional vectors. In PQ algorithm, the high-dimensional space is firstly decomposed into m low-dimensional subspaces, then PQ applies VQ in each subspace independently. It is easy to see that VQ is a special case of PQ when the number m of sub-spaces is taken as $m = 1$. Assuming that the quantizers of all m sub-spaces have the same size of k^*, the main codebook C of PQ can be described as the Cartesian product of m sub-codebooks of m low-dimensional sub-spaces:

$$C = C^1 \times C^2 \times \ldots C^j \times \ldots \times C^m, \tag{2}$$

where C^j is the pre-trained sub-codebook of j-th sub-space. Obviously, the size of C is $k = (k^*)^m$ and PQ can easily generate a tremendous number of cells to quantize the database vectors even though k^* is a small value. In the framework of PQ, a database vector y_i is divided into m sub-vectors $y_i = [u_1(y_i), u_2(y_i), \ldots, u_j(y_i), \ldots, u_m(y_i)]$, where the function of $u_j(y_i)$ outputs the j-th sub-vector of a vector y_i and $u_j(y_i) \in R^{D/m}$. PQ quantizes each sub-vector $u_j(y_i)$ by the nearest codeword in the corresponding j-th sub-codebook C^j.

$$d(u_j(y_i), CW^j_{bm^j}) = \min_{CW_i^j \in C^j} d(u_j(y_i), CW_i^j), \tag{3}$$

where $CW^j_{bm^j}$ is the best match codeword of $u_j(y_i)$ in the pre-trained sub-codebook C^j and its index is bm^j.

In this paper, we define each PQ cell $Cell_i^j$ as the set of database vectors which are quantized by CW_i^j in j-th sub-quantizer. Similar to VQ, PQ quantizes a database into mn-size index tables. The cost of storing each database vector is $m \log_2 k^*$ bits. The flowchart of PQ is shown in Fig. 2.

In the pre-processing, PQ off-line quantizes the database and each database vector y_i is encoded as m indexes of $\log_2 k^*$ bits by using m corresponding sub-codebooks of size k^*. When inputting a query vector x, PQ firstly on-line computes the squared Euclidean distances of $d^2(u_j(x), CW_i^j)$ for $i = 1, 2, \ldots, k^*$ and $j = 1, 2, \ldots, m$ between the j-th sub-vector of x and all k^* codewords in the corresponding j-th sub-codebook, and stores these squared Euclidean distances as a $m \times k^*$ look-up table. Then, the asymmetric distance computation (ADC) is used to approximate the real squared Euclidean distance between x and a database vector y_i. The ADC distance between query x and a database vector y_i is defined as the squared Euclidean distance between x and PQ quantized y_i, which can be computed as

$$\tilde{d}^2(x, y_i) \triangleq \sum_{j=1}^{m} d^2(u_j(x), CW^j_{bm^j}), \tag{4}$$

where $CW^j_{bm^j}$ is the nearest codeword of $u_j(y_i)$ in the j-th sub-quantizer and its index is bm^j. For calculating the squared ADC distance of each database vector, PQ only needs to load m squared distances $d^2(u_j(x), CW_i^j)$ for $j = 1, 2, \ldots, m$ from the look-up table and adds them together. The computational costs in this step are only $n(m-1)$ additions for a n-size database.

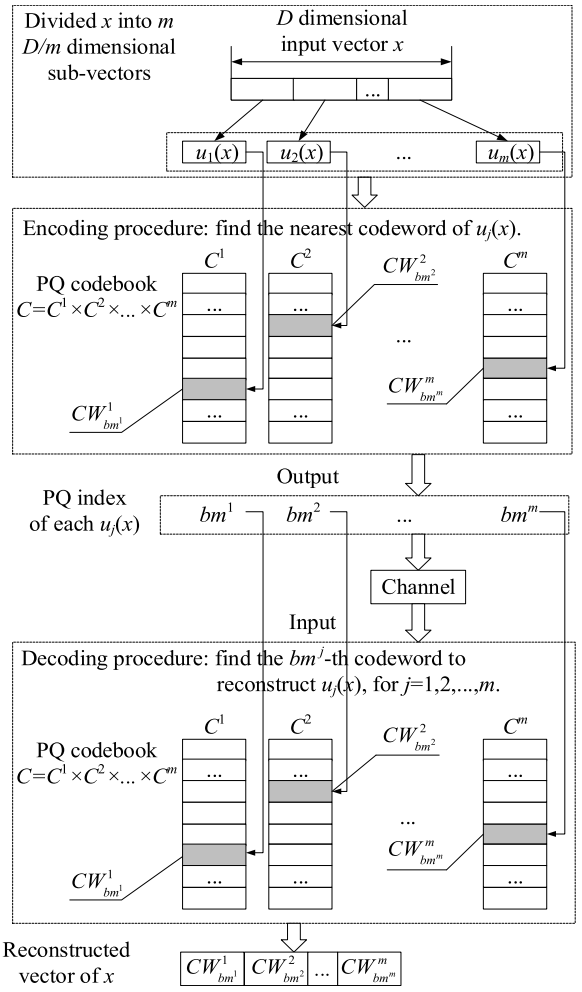


FIGURE 2. The flowchart of PQ.

C. INVERTED INDEXING AND ITS FURTHER IMPROVEMENTS

Inverted index (IVF) [12] is a widely-used non-exhaustive search algorithm, which can be used in PQ algorithm to avoid vector-by-vector search. An auxiliary coarse VQ quantizer of k'-size is introduced and the database is firstly divided into k' cells. IVF searches each of k' cells and chooses the database vectors in the first w nearest cells of x as the candidate list, which are a very small portion of (w/k') $(w \ll k')$ of the database vectors. An overwhelming majority of database vectors can be rejected in this efficient cell-level search and only about $n(w/k')$ database vectors in the candidate list need to be exhaustively searched (here we use "about" because the cells usually contain different numbers of vectors). For the reasons above, PQ search can be efficiently accelerated by IVF algorithm.

A 2-D toy example of IVF is shown in Fig. 3.

The inverted multi-index (IMI) [13] can be considered as a generalized IVF algorithm. Different from conventional IVF, the database Y is divided into two sub-spaces and IMI employs two PQ coarse sub-quantizers instead of one VQ coarse quantizer like IVF for quantization.

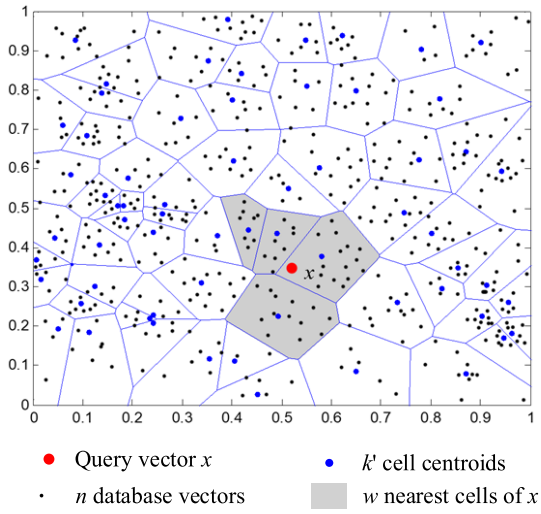


FIGURE 3. A 2D toy example of IVF. The number of IVF cells is $k' = 64$ and the candidate list returned by IVF contains $w = 4$ cells which are the nearest 4 cells to the query vector x.

Because IMI uses far more cells than IVF to quantize the database, the candidate list of IMI can achieve much higher search accuracy than that of IVF when the two candidate lists have the same length.

A 2-D toy example of IMI is shown in Fig. 4.

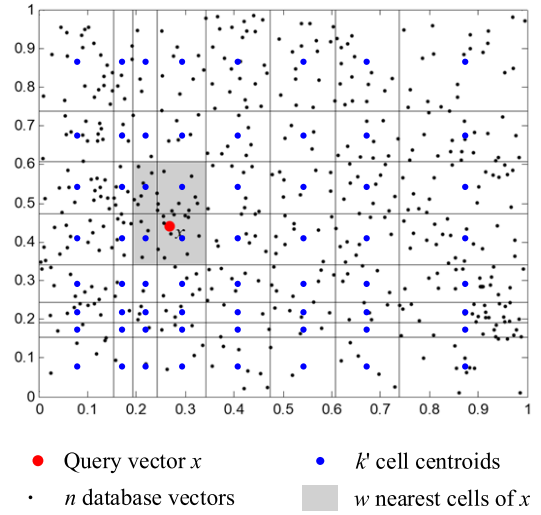


FIGURE 4. A 2D toy example of IMI. The number of IMI cells is $k' = 64$ and the candidate list returned by IMI also contains $w = 4$ cells which are the nearest 4 cells to the query vector x.

The key idea of inverted index and inverted index-based non-exhaustive search algorithms is that for a query vector x and a database Y of size n, efficient cell-level search is conducted on coarsely quantized database Y to firstly generate a short candidate list and then only the database vectors in the candidate list are exhaustively searched. Because the size of candidate list is usually far smaller than the database size of n, a majority of real Euclidean distance calculations among x

and all database vectors can be avoided to speed up the ANN search.

However, we find that in many cases, the computational costs of coarse quantization in IVF and IMI cannot be neglected and need to be re-evaluated. The coarse quantization of IVF and IMI in fact require a lot of extra additions and multiplications, and these extra computational costs depend on the dimensionalities of databases and the sizes of coarse codebooks. For convenience, we assume the total numbers of cells are $k' = k_{IVF}$ for IVF and $k' = k_{IMI}^2$ for IMI. Table 1 shows the computational costs of coarse quantization of IVF and IMI.

TABLE 1. Computational costs of coarse quantization of IVF and IMI.

Algorithm	Additions	Multiplications
IVF	$(2D-1)k_{IVF}$	Dk_{IVF}
IMI	$(2D-1)k_{IMI}$	$2Dk_{IMI}$

Note that exhaustive search in PQ on vector-level requires $n(m-1)$ additions. If $Dk_{IVF} \ll nm$ or $2Dk_{IMI} \ll nm$ is not satisfied, the computational costs for coarse quantization should not be neglected. For example, when $k_{IVF} = 4096$, $n = 1000000$, $D = 960$ and $m = 8$, to find the first w nearest cells, IVF needs to calculate the real squared ADC distance between an input query vector x and each codeword y_i in the coarse codebook. exhaustive search of PQ on vector-level requires $n(m-1) = 7000000$ additions to calculate the real squared ADC distances between x and y_i, $i = 1, 2, \ldots, n$. In comparison, IVF takes $(2D-1)k_{IVF} = 7860224$ additions and $Dk_{IVF} = 3932160$ multiplications for coarse quantization. Obviously, IVF is less efficient than exhaustive search of PQ in this case. Generally, inverted index-based algorithms are not efficient when the size of coarse codebook is large, the dimensionality of database is high and the size of database is not extremely large.

In addition, because inverted index-based algorithms need to store the index of each database vector due to introducing auxiliary coarse codebook, the inevitable extra memory consumption of them for each database vector is considerably large. IVF need $\log_2 k_{IVF}$ bits for storing k' auxiliary indexes of IVF coarse codebook. IMI cells are generated by using two PQ sub-quantizers. Because each of the two PQ sub-quantizers has k_{IMI} codewords, the coarse quantization of IMI requires $2\log_2 k_{IMI}$ bits to store the auxiliary indexes of two sub-quantizers for each database vector. Table 2 shows the extra memory consumptions of coarse quantization of IVF and IMI.

Because the extra memory consumptions of coarse quantization in IVF and IMI depend on the database size n, the extra memory consumptions are considerably large and should not be neglected, especially when the database size n is a large value. For example, considering the case of $k_{IVF} = 1024$, the extra memory consumption of coarse quantization for a n-size database is $10 \times n$ bits for IVF-accelerated PQ, which is considerably large and cannot be neglected.

TABLE 2. Memory consumption of IVF and IMI for coarse quantization.

Algorithm	Storing indexes of coarse codebook (coarse sub-codebooks)	Storing coarse codebook (coarse sub-codebooks)
IVF	$n\log_2 k_{IVF}$	$D\log_2 k_{IVF}$
IMI	$2n\log_2 k_{IMI}$	$2D\log_2 k_{IMI}$

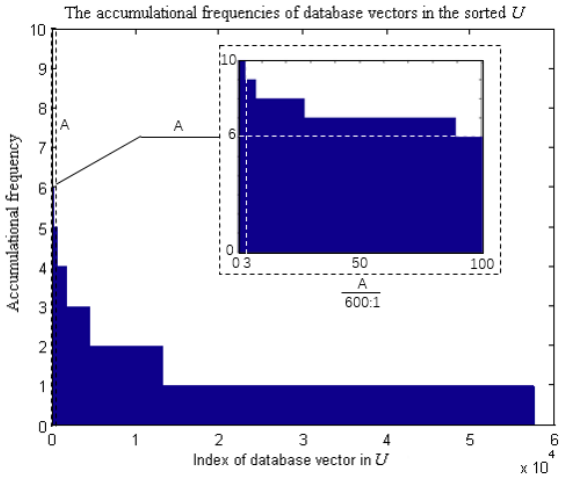


FIGURE 5. The accumulational frequency of each database vector in the sorted *U*.

III. THE PROPOSED CELL-LEVEL SEARCH BASED NON-EXHAUSTIVE ANN SEARCH ALGORITHM

Inverted index-based non-exhaustive search algorithms coarsely quantize database at first to generate k' cells and then use cell-level search to extract a short candidate list from the k' cells efficiently. However, the coarse quantization is computationally expensive. For this reason, computation speed of inverted index-based algorithms is limited. In addition, inverted index-based algorithms need a large amount of extra memory consumption to store the auxiliary index of each database vector in the coarsely cells. These drawbacks of inverted index-based algorithms show that the coarse quantizers used in inverted index-based algorithms are inefficient. In this section, we propose a new cell-level PQ search based non-exhaustive ANN search algorithm called Cell-level PQ, which applies cell-level search by only reusing the PQ cells without using any other coarse quantizer.

In our proposed Cell-level PQ, we assume that the sub-codebook size of PQ is k^*, the size of database is n, the dimensionality of each database vector is D and PQ uses m sub-quantizers for quantization.

Our proposed Cell-level PQ algorithm consists of three parts as follows: (1) Initializing the "so far" smallest squared Euclidean distance. (2) Extracting the candidate list by cell-level searching the PQ cells. (3) Accelerating the search by partial distance search (PDS) [14]. The overview of our proposed Cell-level PQ is shown in Fig. 6. We now explain how our proposed Cell-level PQ runs.

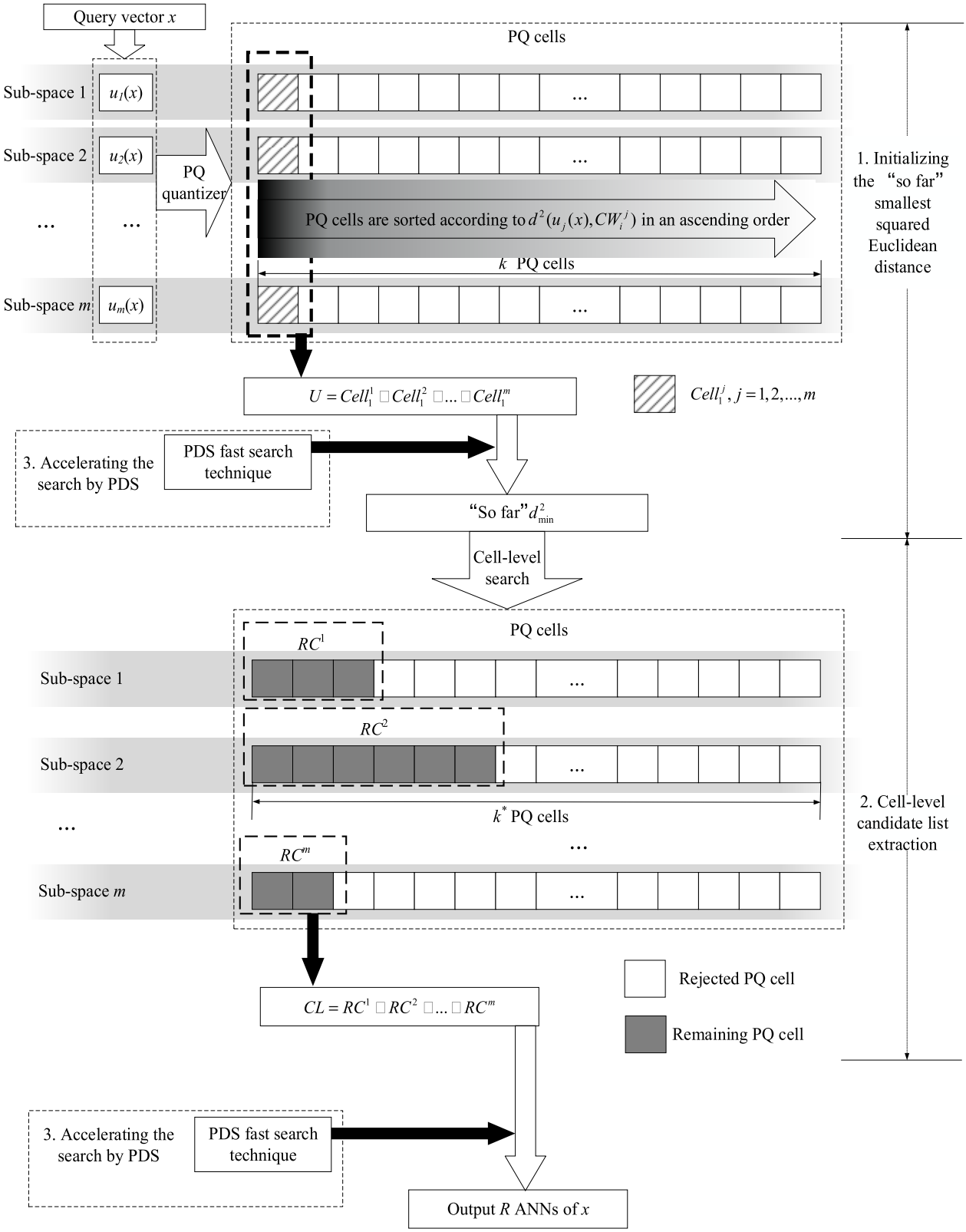


FIGURE 6. Framework of cell-level PQ.

A. INITIALIZING THE "SO FAR" SMALLEST SQUARED ADC DISTANCE

Let us consider the j-th sub-space. At the beginning, all squared ADC distances of $d^2(u_j(x), CW_i^j)$, $i = 1, 2, \ldots, k^*$ are calculated and the PQ cells are sorted according to $d^2(u_j(x), CW_i^j)$ in an ascending order. We find the first PQ cell $Cell_1^j$ of $u_j(x)$, which with CW_1^j as its centroid is the nearest codeword of $u_j(x)$ in the sorted C^j. The database

vectors in the PQ cell of $Cell_1^j$ quantized by CW_1^j are highly possible to be the nearest database vectors of $u_j(x)$ because at least these database vectors are best match vectors of x in the j-th sub-space. The squared ADC distance between $u_j(x)$ and CW_1^j is defined as d_{bmj}^2 and stored. We also store the value of $\sum_{j=1}^{m} d_{bmj}^2$ for the cell-level candidate list extraction in Section III.B. Then, we calculate the union set $U = Cell_1^1 \cup Cell_1^2 \cup ... \cup Cell_1^m$, meanwhile the accumulational frequency of each database vector appearing in $Cell_1^j, j = 1, 2, \ldots, m$ is counted. Finally, U is sorted according to their accumulational frequencies in a descending order. An example of sorted U is shown in Fig.6.

In this example, PQ uses $m = 16$ sub-quantizers to quantize the SIFT 1M database [15], which contains 10^6 128-dimensional vectors. U contains 57385 database vectors, and there are 3 database vectors with the highest accumulational frequency of 10. The sorted U is used in Section III. C for fast PDS search.

After obtaining sorted U, we calculate the squared ADC distance between x and each database vector in U to find the nearest database vector of x in U. By using the stored squared ADC distances in the look-up table, the squared ADC distance between x and a database vector y_i can be calculated by using (4). Next, we define y_{BM} the nearest database vector of x in U, and the "so far" smallest squared distance $d_{\min}^2$ is the squared ADC distance between x and y_{BM}.

Moreover, the calculation of finding the nearest database vector of a query vector x in U can be further accelerated by using partial distance search (PDS). The detailed accelerating procedures are described in Section III.C.

B. CELL-LEVEL CANDIDATE LIST EXTRACTION

In this section, we will explain how to use the "so far" smallest squared ADC distance $d_{\min}^2$ to search the PQ cells. Let us consider an arbitrary $Cell_i^j$ in j-th sub-space, where $i = 1, 2, \ldots, k^*, j = 1, 2, \ldots, m$. The squared ADC distance between x and the database vectors in $Cell_i^j$ satisfies

$$\tilde{d}^2(x, y) \geq d^2(u_j(x), CW_i^j) + ((\sum_{j=1}^{m} d_{bmj}^2) - d_{bmj}^2), \qquad (5)$$

where d_{bmj}^2 is the squared ADC distance between $u_j(x)$ and its nearest codeword in C^j, which has been calculated and stored in Section III.A. We do not need to calculate the value of $\sum_{j=1}^{m} d_{bmj}^2$ because this value is fixed and is stored in Section III.A for fast calculation so that it only needs on on-line substitution to obtain $((\sum_{j=1}^{m} d_{bmj}^2) - d_{bmj}^2)$.

Obviously, if $Cell_i^j$ with CW_i^j as its centroid satisfies,

$$d^2(u_j(x), CW_i^j) + ((\sum_{j=1}^{m} d_{bmj}^2) - d_{bmj}^2) \geq d_{\min}^2, \qquad (6)$$

all database vectors in this $Cell_i^j$ are not nearer than y_{BM}. Because in Section III.A the PQ cells in each j-th sub-space have been sorted according to $d^2(u_j(x), CW_i^j)$ in an ascending order, once (6) is satisfied, all database vectors in the PQ cells $Cell_i^j, Cell_{i+1}^j, \ldots, Cell_{k^*}^j$ can be safely rejected. The set of remaining database vectors in the remaining PQ cells of $Cell_1^j, Cell_2^j, \ldots, Cell_{i-1}^j$ in each j-th sub-space is obtained by $RC^j = Cell_1^j \cup Cell_2^j \cup ... \cup Cell_{i-1}^j$. Then, we defined CL as the candidate list. When all PQ cells have been searched, CL can be constructed as $CL = RC^1 \cap RC^2 \cap ... \cap RC^m$. The other database vectors which are not included in U or CL are represented as O. The framework of extracting the candidate listCL is summarized in Algorithm 1.

Algorithm 1 Extracting the Candidate List CL

INPUT: The database $Y = [y_1, y_2, \ldots, y_n]$, a query vector x, the sorted PQ cells according to $d^2(u_j(x), CW_i^j)$, $i = 1, 2, \ldots, k^*, j = 1, 2, \ldots, m$ in an ascending order, the "so far" smallest squared distance $d_{\min}^2$.

OUTPUT: The candidate list CL.

1. **for** each sub-vector $u_j(x)$ of $x, j = 1, 2, \ldots, m$ **do**
2. $i = 1$
3. $c = (\sum_{j=1}^{m} d_{bmj}^2) - d_{bmj}^2$
4. **while** $i \leq k^* \&\& \; d^2(u_j(x), CW_i^j) + c < d_{\min}^2$ **do**
5. $i = i + 1$
6. **end while**
7. $RC^j = Cell_1^j \cup Cell_2^j \cup ... \cup Cell_{i-1}^j$
8. **end for**
9. $CL = RC^1 \cap RC^2 \cap ... \cap RC^m$
10. **Return** CL.

C. ACCELERATING SEARCH BY PARTIAL DISTANCE SEARCH

In the framework of PQ, $d^2(u_j(x), CW_i^j)$ in each j-th sub-space is firstly computed and stored in a look-up table. When computing $\tilde{d}(x, y_i)$, PQ simply needs to load m squared ADC distances in the look-up table and calculate the real squared ADC distance by using Eq. (4). Because the look-up table is already computed and saved for a query vector x, the only computational cost of PQ is $(m - 1)$ times additions for computing $\tilde{d}(x, y_i)$.

Partial distance search (PDS) technique [14] is employed in our proposed Cell-level PQ to avoid completely calculating $(m-1)$ times additions to obtain real squared ADC distance of the database vectors in U and CL. PDS is a popular fast search algorithm, which uses the partial squared ADC distance to reject database vectors in U and CL. In Cell-level PQ, a two-stage PDS is introduced for fast search.

For database vectors in sorted U, the "so far" nearest database vector is initialized as the first database vector y_i in the sorted U, and the "so far" $d_{\min}^2$ is initialized as the squared ADC distance between x and the first database vector in U. For database vectors in CL, the "so far" nearest

database vector is initialized as y_{BM}, and the "so far" $d^2_{\min}$ is initialized as $\tilde{d}(x, y_{BM})$. For a database vector y_i in U or CL, the first-stage partial squared ADC distance between x and y_i is defined as,

$$d^2_{p1}(x, y_i) = \sum_{j=1}^{m/4} d^2(u_j(x), CW^j_{bm^j}), \tag{7}$$

where $CW^j_{bm^j}$ is the nearest codeword of $u_j(y_i)$ in the j-th sub-quantizer and its index is bm^j. $d^2_{\min}$ is the squared ADC distance between x and the "so far" nearest database vector. If $d^2_{p1}(x, y_i) \geq d^2_{\min}$, then y_i cannot be nearer than the "so far" nearest database vector y_{BM} and can be safely rejected. Otherwise, we check the second-stage partial squared ADC distance between x and y_i,

$$d^2_{p2}(x, y_i) = d^2_{p1}(x, y_i) + \sum_{j=m/4+1}^{m/2} d^2(u_j(x), CW^j_{bm^j}). \tag{8}$$

If $d^2_{p2}(x, y_i) \geq d^2_{\min}$, y_i also cannot be the nearest neighbor of x and can be safely rejected. Otherwise, we completely calculate the squared ADC distance between x and y_i by,

$$\tilde{d}^2(x, y_i) = d^2_{p2}(x, y_i) + \sum_{j=m/2+1}^{m} d^2(u_j(x), CW^j_{bm^j}). \tag{9}$$

If $\tilde{d}^2(x, y_i) < d^2_{\min}$, y_i is the nearer database vector to x than the "so far" nearest database vector y_{BM} so that the "so far" $d^2_{\min}$is updated and replaced by $\tilde{d}^2(x, y_i)$, meanwhile the "so far" nearest database vector y_{BM} is updated and replaced by y_i.

By rejecting the database vectors in U or CL using Eq. (7) and Eq. (8), a majority of squared ADC distance calculations of PQ can be avoided. The flowchart of the PDS checking procedures is shown in Fig. 7.

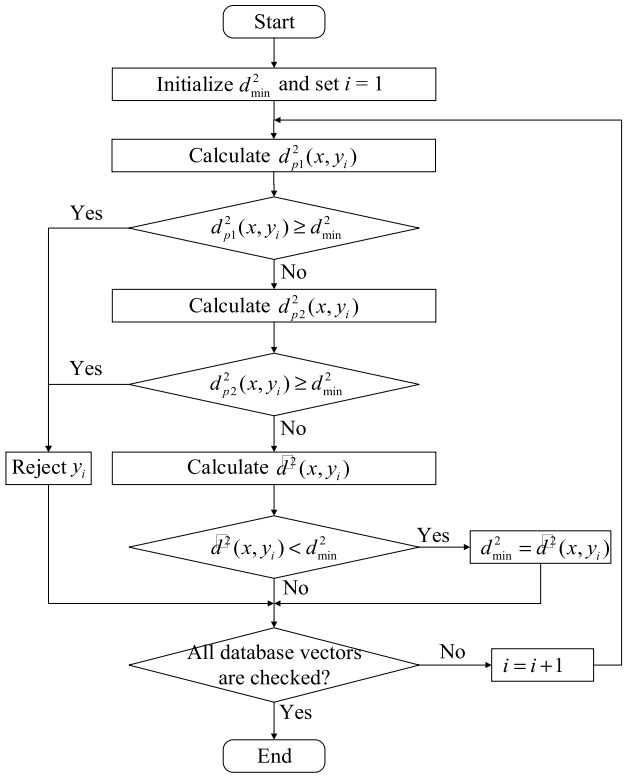


FIGURE 7. The flowchart of PDS search.

IV. EXPERIMENTAL RESULTS

In this section, we firstly introduce the databases used in our experiments. Then, the computational cost of our proposed Cell-level PQ is evaluated and compared with two existing non-exhaustive algorithms, which are inverted index (IVF) and inverted multi index (IMI). Finally, the performances of search accuracy of these three algorithms are compared and analyzed.

A. DATA SETS

To evaluate the performance of our proposed Cell-level PQ, the experiments are performed on two well-known and widely used databases:

1. SIFT 1M [15] database. SIFT database consists of a gallery set which contains one million (10^6) 128-dimensional SIFT descriptors, a learning set which contains 100k (10^5) SIFT descriptors, a query set with 10k (10^4) SIFT descriptors and the precomputed groundtruth (the Euclidean nearest neighbors) of the query set.
2. GIST 1M [16] database. GIST database consists of a gallery set which contains one million (10^6) 960-dimensional GIST descriptors, a learning set which contains 10k (10^4) GIST descriptors, a query set with 10k (10^4) GIST descriptors and the precomputed groundtruth of the query set.

Codebooks with different sizes are generated off-line by the popular LBG algorithm [11] using the learning sets of different databases. Following the settings of IVF, the coarse codebook sizes of IVF are set to $k' = 1024$ and $k' = 4096$. In order to making a fair comparison, the coarse codebook sizes of IMI for each sub-quantizer are set to $\sqrt{1024} = 32$ and $\sqrt{4096} = 64$, respectively. In these two cases, the numbers of IMI cells are the same as the numbers of IVF cells. In each of these two cases, the number of IMI cells is the same as the number of IVF cells. Following recent related PQ-based fast ANN works, the number of PQ sub-spaces is typically set to $m = 8$ or $m = 16$ and the size of PQ sub-codebooks is set to $k^* = 256$. All of our experiments are implemented by using Visual Studio 2017, which works on a PC with Intel Core i5-6500 3.2GHz processor and 8GB RAM using single thread.

B. COMPUTATIONAL COSTS EVALUATION

Our proposed Cell-level PQ is proposed to avoid the squared distance calculations in standard PQ. In Cell-level PQ, the database can be divided into three parts as below:

Part 1. The database vectors in U which are used to initialize the "so far" smallest squared distance $d^2_{\min}$.

Part 2. The database vectors in candidate list CL which are used to find approximate nearest neighbors.

Part 3. The database vectors in O which can be safely rejected by using cell-level search.

The percentages of these three parts of database vectors are represented as P_U, P_{CL} and P_O, respectively. Because U and CL are partially overlapped, the percentage $P_{U\cup CL}$ of the union set of U andCL is a little smaller than $P_U + P_{CL}$. The percentages of three different parts of database vectors are shown in Table 3.

TABLE 3. The average vector numbers in different parts of database.

Database	m	P_U	P_O	P_{CL}	$P_{U\cup CL}$
SIFT 1M database	8	3.85%	94.08%	2.34%	5.93%
	16	4.20%	72.46%	24.91%	27.53%
GIST 1M database	8	3.70%	95.86%	0.54%	4.14%
	16	7.32%	68.81%	26.83%	31.19%

Squared ADC distance computations between x and PQ quantized database vectors y_i in these three parts of database are different. Database vectors in O can be rejected safely by using the cell-level search and all of squared ADC distance computations can be avoided.

For the database vectors in $U \cup CL$, PDS technique can be applied and a part of complete squared ADC distance computations can be saved. If a database vector y_i can be rejected by using PDS at the first stage check by Eq.(7), the computational costs are only $(m/4 - 1)$ additions. Otherwise, if y_i can be rejected by using PDS at the second stage check by Eq.(8), only $(m/2 - 1)$ additions are required. For other database vectors in $U \cup CL$, complete real squared ADC distance computations require $(m-1)$ additions. For the database vectors in $U \cup CL$, the percentage of remaining real squared ADC distance computations can be calculated as:

$$R_{U\cup CL} = \frac{(\frac{m}{4} - 1)P_1 + (\frac{m}{2} - 1)P_2 + (m-1)P_3}{(m-1)P_{U\cup CL}} \times 100\%, \tag{10}$$

where P_1 and P_2 are the percentage of database vectors in $U \cup CL$ which can be rejected by the first stage rejection of PDS and the second stage rejection of PDS, respectively. $P_3 = P_{U\cup CL} - P_1 - P_2$ is the percentage of remaining database vectors in $U \cup CL$. The percentage of the saved computational costs of complete real ADC squared distances can be evaluated as:

$$R_{total} = 1 - R_{U\cup CL} \times P_{U\cup CL}. \tag{11}$$

The computational costs reduction performance of our proposed Cell-level PQ is shown in Table 4. Table 4 shows that for SIFT database, a majority of complete real squared ADC distance computations for database vectors in $U \cup CL$ can be saved by using the PDS technique. For GIST database, PDS technique also works, but the computational costs reduction is not very significant. We can also learn from Table 4 that because the database vectors in O do not require any real squared ADC distance computations, PQ can be efficiently accelerated. Taking the SIFT database for example, when

TABLE 4. Computational costs reduction performance of cell-level PQ.

Database	m	$P_{U\cup CL}$	P_1	P_2
SIFT 1M database	8	5.93%	1.35%	3.86%
	16	27.53%	9.26%	17.29%
GIST 1M database	8	4.14%	0.31%	0.75%
	16	31.19%	0.45%	10.35%
Database	m	$(P_1+P_2)/P_{U\cup CL}$	$R_{U\cup CL}$	R_{total}
SIFT 1M database	8	87.91%	43.24%	97.44%
	16	96.43%	39.60%	89.10%
GIST 1M database	8	25.44%	83.34%	96.55%
	16	34.61%	91.16%	71.63%

TABLE 5. Detailed search time (ms) of different non-exhaustive search algorithms.

Algorithm	$m=8$, SIFT 1M database				
	w	k'	t_1	t_2	Total
Standard PQ	N/A	N/A	N/A	49.42	49.42
IVF	8	2^{10}	5.64	0.46	6.09
	32	2^{12}	23.51	0.46	23.97
IMI	8	$2^5\times2^5$	0.22	0.46	0.69
	32	$2^6\times2^6$	0.58	0.46	1.04
Cell-level PQ	N/A	N/A	N/A	2.61	2.61
Algorithm	$m=16$, SIFT 1M database				
	w	k'	t_1	t_2	Total
Standard PQ	N/A	N/A	N/A	55.51	55.51
IVF	8	2^{10}	6.65	0.50	7.16
	32	2^{12}	24.36	0.50	24.86
IMI	8	$2^5\times2^5$	0.28	0.50	0.78
	32	$2^6\times2^6$	0.69	0.50	1.18
Cell-level PQ	N/A	N/A	N/A	6.26	6.26
Algorithm	$m=8$, GIST 1M database				
	w	k'	t_1	t_2	Total
Standard PQ	N/A	N/A	N/A	50.11	50.11
IVF	8	2^{10}	40.97	0.47	41.43
	32	2^{12}	153.82	0.47	154.29
IMI	8	$2^5\times2^5$	1.28	0.46	1.74
	32	$2^6\times2^6$	2.61	0.47	3.08
Cell-level PQ	N/A	N/A	N/A	2.13	2.13
Algorithm	$m=16$, GIST 1M database				
	w	k'	t_1	t_2	Total
Standard PQ	N/A	N/A	N/A	55.84	55.84
IVF	8	2^{10}	43.49	0.50	43.99
	32	2^{12}	157.28	0.53	157.81
IMI	8	$2^5\times2^5$	1.37	0.51	1.88
	32	$2^6\times2^6$	2.70	0.51	3.21
Cell-level PQ	N/A	N/A	N/A	16.01	16.01

TABLE 6. The overall performance of different non-exhaustive search algorithms for different databases.

Algorithm	w	k'	Search time (ms)	Q_{CL}	Recall@1	Recall@10	Recall@100
$m=8$, SIFT 1M database							
Standard PQ	N/A	N/A	49.42	N/A	31.8%	65.9%	92.1%
IVF	8	2^{10}	6.09	99.50%	31.8%	65.6%	91.9%
	32	2^{12}	23.97	99.90%	32.0%	65.8%	92.1%
IMI	8	$2^5\times2^5$	0.69	89.80%	31.3%	65.2%	91.7%
	32	$2^6\times2^6$	1.04	95.80%	31.7%	65.8%	91.9%
Cell-level PQ	N/A	N/A	2.61	100%	31.9%	65.9%	92.1%
$m=16$, SIFT 1M database							
Standard PQ	N/A	N/A	55.51	N/A	55.9%	89.6%	99.8%
IVF	8	2^{10}	7.16	99.50%	53.9%	85.9%	98.8%
	32	2^{12}	24.86	99.90%	55.6%	88.4%	99.1%
IMI	8	$2^5\times2^5$	0.78	89.80%	52.8%	85.5%	87.7%
	32	$2^6\times2^6$	1.18	95.80%	54.0%	87.5%	93.3%
Cell-level PQ	N/A	N/A	6.26	100%	55.9%	89.6%	99.8%
$m=8$, GIST 1M database							
Standard PQ	N/A	N/A	50.11	N/A	7.7%	13.9%	43.7%
IVF	8	2^{10}	41.43	97.40%	7.7%	13.6%	42.8%
	32	2^{12}	154.29	99.30%	7.8%	14.0%	44.3%
IMI	8	$2^5\times2^5$	1.74	71.90%	6.0%	13.1%	42.2%
	32	$2^6\times2^6$	3.08	82.70%	7.3%	13.3%	43.0%
Cell-level PQ	N/A	N/A	2.13	99.80%	7.7%	13.9%	43.7%
$m=16$, GIST 1M database							
Standard PQ	N/A	N/A	55.84	N/A	8.3%	34.4%	72.4%
IVF	8	2^{10}	43.99	97.40%	8.3%	33.8%	63.5%
	32	2^{12}	157.81	99.30%	8.4%	34.4%	71.6%
IMI	8	$2^5\times2^5$	1.88	71.90%	7.7%	31.8%	63.3%
	32	$2^6\times2^6$	3.21	82.70%	8.1%	33.1%	68.2%
Cell-level PQ	N/A	N/A	16.01	100%	8.4%	34.4%	72.4%

$m = 8$, 97.44% of real squared ADC distance computations between x and database vectors can be avoided.

The detailed search time of different algorithms is given in Table 5. In Table 5, t_1 is the search time of coarse quantization of IVF or IMI, and t_2 is the search time of vector-level search in different algorithms. We can see from t_2 in Table 5 that because the candidate list size of IVF and IMI is about $n(w/k') = n\times(8/2^{10}) = n\times(32/2^{12}) = n/128$, which is smaller than $n \times P_{U\cup CL}$ of our proposed cell-level PQ, IVF and IMI require shorter time for searching their candidate lists. Here $P_{U\cup CL}$ is the percentage of the vectors which require vector-level search in the database. However, t_1 in Table 5 demonstrated that the coarse quantization is very complex. Taking SIFT 1M database as an example, when $m = 8$, the coarse quantization of IVF costs 5.64ms and searching in the candidate list costs only 0.46ms. Obviously, the search time of IVF is mainly determined by the coarse quantization. For this reason, the search time of IVF is even longer than that of standard PQ when the dimensionality of database is high. By completely avoiding the coarse quantization, the search time of our proposed Cell-level PQ is much shorter than that of IVF and is comparable to IMI when $m = 8$. We can also see from Table 5 that IMI is the fastest algorithm. However, the search accuracy of IMI is not satisfactory, which will be detailly analyzed in Section IV.C.

C. SEARCH ACCURACY EVALUATION

A non-exhaustive ANN search algorithm generates a candidate list firstly, which is a small portion of database in order to avoid exhaustive search. Because only the database vectors included in the candidate list are searched afterwards, ANN search can be efficiently speeded up.

However, if the groundtruth vector is not included in the candidate list, non-exhaustive ANN search cannot return the

groundtruth anyway. We define the candidate lists which contain the groundtruth as the ''good'' candidate lists and the other candidate lists as the ''bad'' candidate lists. Assuming that the number of ''good'' candidate lists is N_1 and the number of ''bad'' candidate lists is N_2, Q_{CL} in our paper is then defined as $Q_{CL} = \frac{N_1}{N_1+N_2} \times 100\%$.

The search accuracy can also be evaluated in terms of Recall@R, which is calculated as the ratio of query vectors for whose groundtruth is presented in its nearest R quantized database vectors. Obviously, Q_{CL} is the upper bound of Recall@R, and a high Q_{CL} is a necessary condition for obtaining a high Recall@R. The overall search speed and search accuracy performance of our proposed Cell-level PQ is summarized in Table 6.

We can learn from Table 6 that Cell-level PQ can keep the highest Q_{CL}. We can also observe that the search accuracy is slightly affected by Cell-level PQ. Note that the experimental results are different from the result in [13], where the value of Recall@R of IMI is higher than that of standard PQ and IVF. But in our experiment, the value of Recall@R of IMI is worse. The reason is that in [13], IMI sets $k_{IMI} = 2^{14}$ and $k_{IVF} = 2^{13}$ or $k_{IVF} = 2^{16}$. In this case, IMI produces $k_{IMI}^2 = 2^{28}$ cells, which is far more than the cells of IVF. It means that IMI will use about twice memory of IVF for storing the indexes of two sub-codebooks. In our experiment, we set $k_{IVF} = k_{IMI}^2$ to make IVF and IMI have the same numbers of cells for a fair comparison. In this case, the candidate list of IMI achieves much worse search accuracy performance than IVF.

Taking SIFT 1M database as an example, when the number of cells is 2^{10} and we select the database vectors in the first 8 nearest cells of the current query vector as the candidate list, the Q_{CL} of IVF is 99.50%, whereas the Q_{CL} of IMI is 89.80%, which is quite smaller than 99.50% of IVF and 100% of our proposed Cell-level PQ. It means IVF can achieve better search performance than IMI when IVF has the same cells as IMI. Taking the search time into consideration, our proposed Cell-level PQ achieves the best trade-off between search speed and search accuracy.

V. CONCLUSION

In this paper, we proposed a new cell-level search based non-exhaustive ANN search algorithm named Cell-level PQ in the framework of PQ, which completely avoids using the coarse quantizer and reuses the PQ cells to extract a short and high-quality candidate list for fast ANN search. In our work, the inverted index-based algorithms are re-evaluated and we find that they are inefficient for their non-negligible extra computational costs and considerably large extra memory consumptions, especially when the database size is not extremely large and the dimensionality of database is high.

Compared with the inverted index-based non-exhaustive search algorithms, because our proposed Cell-level PQ only requires very little extra computational costs and no extra memory consumption, Cell-level PQ is more efficient than inverted index-based algorithms.

Experimental results on different databases demonstrate that PQ search can be significantly speeded up by using our proposed Cell-level PQ, and compared with the state-of-the-art algorithms, our proposed algorithm achieves the best trade-off between search speed and search accuracy.

REFERENCES

[1] J. Tang, Z. Li, and X. Zhu, ''Supervised deep hashing for scalable face image retrieval,'' *Pattern Recognit.*, vol. 75, pp. 25–32, Mar. 2017.

[2] Y. Chen, X. Li, A. Dick, and R. Hill, ''Ranking consistency for image matching and object retrieval,'' *Pattern Recognit.*, vol. 47, no. 3, pp. 1349–1360, Mar. 2014.

[3] J. Cheng, C. Leng, J. Wu, H. Cui, and H. Lu, ''Fast and accurate image matching with cascade hashing for 3D reconstruction,'' in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2014, pp. 1–8.

[4] C. Deng, X. Liu, Y. Mu, and J. Li, ''Large-scale multi-task image labeling with adaptive relevance discovery and feature hashing,'' *Signal Process.*, vol. 112, pp. 137–145, Jul. 2015.

[5] C. Silpa-Anan and R. Hartley, ''Optimised KD-trees for fast image descriptor matching,'' in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2008, pp. 1–8.

[6] J. Yuan and X. Liu, ''Product tree quantization for approximate nearest neighbor search,'' in *Proc. IEEE Conf. Image Process.*, Sep. 2015, pp. 2035–2039.

[7] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, ''Locality-sensitive hashing scheme based on p-stable distributions,'' in *Proc. 20th Ann. Symp. Comput. Geometry*, 2004, pp. 253–262.

[8] S. Ercoli, M. Bertini, and A. Del Bimbo, ''Compact hash codes for efficient visual descriptors retrieval in large scale databases,'' *IEEE Trans. Multimedia*, vol. 19, no. 11, pp. 2521–2532, Nov. 2016.

[9] H. Jegou, M. Douze, and C. Schmid, ''Product quantization for nearest neighbor search,'' *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–128, Jan. 2011.

[10] Q. Ning, J. Zhu, Z. Zhong, S. C. H. Hoi, and C. Chen, ''Scalable image retrieval by sparse product quantization,'' *IEEE Trans. Multimedia*, vol. 19, no. 3, pp. 586–597, Mar. 2017.

[11] Y. Linde, A. Buzo, and R. M. Gray, ''An algorithm for vector quantizer design,'' *IEEE Trans. Commun.*, vol. 28, no. 1, pp. 84–95, Jan. 1980.

[12] J. Sivic and A. Zisserman, ''Video Google: A text retrieval approach to object matching in videos,'' in *Proc. IEEE Conf. Comput. Vis.*, Oct. 2003, pp. 1470–1477.

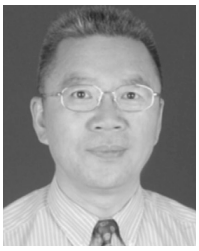
[13] A. Babenko and V. Lempitsky, ''The inverted multi-index,'' *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 6, pp. 1247–1260, Jun. 2015.

[14] W.-J. Hwang, ''Fast kNN classification algorithm based on partial distance search,'' *Electron. Lett.*, vol. 34, no. 21, pp. 2062–2063, Oct. 1998.

[15] D. G. Lowe, ''Distinctive image features from scale-invariant keypoints,'' *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[16] A. Oliva and A. Torralba, ''Modeling the shape of the scene: A holistic representation of the spatial envelope,'' *Int. J. Comput. Vis.*, vol. 42, no. 3, pp. 145–175, 2001.

YANG WANG received the B.S. degree from Xi'an Jiaotong University, Xi'an, China, in 2010, where he is currently pursuing the Ph.D. degree with the School of Electronic and Information Engineering. His research interests include image coding, image processing, and multimedia information retrieval.



ZHIBIN PAN received the B.S. degree in information and telecommunication engineering and the M.S. degree in automation science and technology from Xi'an Jiaotong University, China, in 1985 and 1988, respectively, and the Ph.D. degree in electrical engineering from Tohoku University, Japan, in 2000. He is currently a Professor with the School of Electronic and Information Engineering, Xi'an Jiaotong University. His current research interests include image compression, multimedia security, and object recognition.

RUI LI received the B.S. and master's degree from Xidian University, Xi'an, China, in 2012 and 2015, respectively. He is currently pursuing the Ph.D. degree with the School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an. His research interests include vector quantization and hyper-spectral image processing.