Efficient Fingercode Classification

Hong-Wei SUN†, Kwok-Yan LAM‡, Dieter GOLLMANN††, Nonmembers, Siu-Leung CHUNG††‖‖, Member, Jian-Bin LI††††, and Jia-Guang SUN†, Nonmembers

SUMMARY In this paper, we present an efficient fingerprint classification algorithm which is an essential component in many critical security application systems such as in homeland security systems and e-commerce. Fingerprint identification is one of the most important security requirements in homeland security systems such as personnel screening and anti-money laundering. The problem of fingerprint identification involves searching (matching) the fingerprint of a person against each of the fingerprints of all registered persons. To enhance performance and reliability, a common approach is to reduce the search space by firstly classifying the fingerprints and then performing the search in the respective class. Jain et al. proposed a fingerprint classification algorithm based on a two-stage classifier, which uses a K-nearest neighbor classifier in its first stage. The fingerprint classification algorithm is based on the fingerprint representation which is an encoding of fingerprints that has been demonstrated to be an effective fingerprint biometric scheme because of its ability to capture both local and global details in a fingerprint image. We enhance this approach by improving the efficiency of the K-nearest neighbor classifier for fingercode-based fingerprint classification. Our research first studies the various fast search algorithms in vector quantization (VQ) and then proposes two efficient algorithms based on the pyramid-based search algorithms in VQ. Experimental results on DB1 of FVC 2004 demonstrate that our algorithms can outperform the full search algorithm and the original pyramid-based search algorithms in terms of computational efficiency without sacrificing accuracy.

key words: vector quantization, fingercode, fingerprint classification, system software, homeland security

1. Introduction

In the face of strong needs arising from counter-terrorism and anti-money laundering requirements, new technologies and standards have been developed recently to address key issues related to the identification needs of financial and government systems [1], [2]. With the emerging trend of incorporating biometrics information in e-commerce and e-government systems arising from international efforts against terrorist financing and for effective border control, biometric identification is gaining increasing importance as a component in information security applications.

Among all the biometric characteristics, fingerprints have been shown to have one of the highest levels of reliability [3]. Fingerprint identification systems have become an essential component in security-sensitive application systems and are used in a wide range of civilian and commercial applications for user authentication purposes (e.g. biometric passport, electronic fund transfer, door access to critical facilities). A typical fingerprint identification system has to match a given fingerprint with each of the templates of all registered people [4]. Since each match is computationally intensive, the maximum size of the database has a critical impact on the efficiency of the identification system [5]. To enhance the efficiency of fingerprint identification, a common approach is to firstly classify the fingerprints, and then perform matching only within the respective class, as shown in Fig. 1. Hence, fingerprint classification is able to reduce the number of templates that has to be matched, thus greatly reducing fingerprint identification time for a large database.

Various schemes have been developed for automatic fingerprint classification, which can be broadly classified into two categories: structure-based schemes and model-based schemes. Structure-based schemes, such as [6], use the estimated orientation field of fingerprint images to classify fingerprints. The weakness of this scheme is that it is very difficult to reliably obtain the estimated orientation field in low quality fingerprint images. With model-based schemes, such as [7], the locations of singular points (core and delta) are used to classify fingerprints. The problem of these schemes is that in real situations singular points could be difficult to detect in noisy fingerprint images; in addition, some new fingerprint scanning devices may capture only a part of the fingerprint, which often makes the delta points missing.

[8] proposed a fingerprint classification algorithm that aimed to overcome the problems of the existing schemes. This algorithm is based on fingercode, which is directly de-
Fig. 2 Classification scheme by Jain et al.

Fig. 3 Fingerprint classes. (a) Whorl (W), (b) Left loop (L), (c) Right loop (R), (d) Arch (A), (e) Tented arch (T).

rived from the local ridge structures and does not use the core points and estimated orientation field explicitly. [8] firstly uses a bank of Gabor filters [9] to encode both local and global details in a fingerprint as a fingercode, which is represented by a fixed-length vector. The classification algorithm is divided into the pre-computing phase and online computing phase (as shown in Fig. 2), which are described as follows:

**Pre-computing phase:** Suppose there are some templates (fingercodes) in the database which are manually labelled with one of the five classes, namely, whorl, right loop, left loop, arch and tented arch (as shown in Fig. 3). Select the templates of any pair of two classes, and then train a multilayer feed-forward neural network [10] to distinguish between each pair of classes using the templates from only the two corresponding classes in the database. The input of each neural network is a fingercode, and the output is the determined class from the pair of classes. Note that these neural networks have one hidden layer with 20–40 neurons, and two output neurons [8]. The number of the input neurons is equal to the dimension of the fingercode. There will be \(C_5^2 = 10\) neural networks since there are totally five classes.

**Online-computing phase:** Each query fingercode is fed into a two-stage classifier, which classifies the fingerprint into one of the five classes. In the first stage, the squared Euclidean distance between the query fingercode and each tem-plate in the database is computed in order to find the \(K\) nearest templates. Since each template is manually labelled with the right class, two most possible classes can be found which are most frequently represented among the \(K\) nearest templates. The top two categories can be retrieved from the \(K\)-nearest neighbor classifier corresponding to the classes which have the highest and the second highest count among the \(K\) nearest neighbors. In the second stage, firstly select the corresponding neural network using the two most possible classes, and then send the query fingercode to the selected neural network for further classification in order to make the final decision between these two classes.

[8] reported that this scheme achieved a classification accuracy of 90 percent. In a typical fingerprint identification system, once a query fingerprint was misclassified, it would not be matched with any template of the misclassified category, and then it will be matched with all templates in the database to improve the matching accuracy.

Since the first stage needs more computation time compared with the second stage in the online-computation phase of fingercode-based fingerprint classification [8], it is more important to improve the performance of the \(K\)-nearest neighbor classifier to enhance the original approach. Our research focuses on the first stage of the classifier for the fingercode-based fingerprint classification. To find the \(K\) nearest templates of the given fingerprint, the fingerprint classification system needs to compute and compare the squared Euclidean distance between the user’s fingercode and all stored templates, which is a computation-intensive process.

To enhance the performance of fingercode-based classification, we formulate the \(K\)-nearest neighbor classifier as a VQ encoding problem, which is an efficient technique for low-bit-rate image compression. We then investigate the various fast search algorithms in VQ [11], [12] and their potential application in improving the efficiency of the \(K\)-nearest neighbor classifier in fingercode-based classification system.

However, the approach of applying VQ techniques to the \(K\)-nearest neighbor classifier is not straightforward. Though many efficient algorithms have been developed to enhance the VQ encoding process [13]–[26], not all VQ schemes can be applied directly to fingercode-based classification. The VQ encoding schemes can be roughly classified into two types according to the coding quality. The first type guarantees that the closest codeword is found [13]–[22] while the second type does not provide such guarantee [23]–[26]. VQ techniques of the first type typically use the statistical features of a vector or pyramid data structures to estimate the Euclidean distance and reject most of the unlikely codewords without computing the actual Euclidean distances. Recently, various efficient pyramid data structures, such as the mean-variance pyramids [15], \(L_2\)-norm pyramid [18] and 2-pixel merging sum pyramid [20], have been proposed for achieving fast codeword search in VQ. Techniques of the second type usually shorten the encoding time at the expense of the coding quality, as it does not
guarantee that the nearest codeword is found. Only the first type can be applied to fingercode classification because the classifier requires that the closest codewords are found.

Besides, existing VQ techniques cannot be readily applied to solve the fingercode classifier problem because the software parameter of the problem is significantly different from conventional image compression. This is because, for the latter problems, the dimension of vectors representing image blocks is small, typically 16 dimensions; whereas, for the former, fingercode tends to have a dimensionality of several hundreds.

In this paper, we propose two efficient algorithms for fingercode-based fingerprint classification. The organization of this paper is as follows. The fingercode biometric scheme is introduced in Sect. 2. In Sect. 3, we investigate how the various fast search algorithms in VQ can improve the efficiency of the K-nearest neighbor classifier. The new fingercode search algorithms will be described in detail in Sect. 4 which is followed by a presentation of the experimental results in Sect. 5. The discussion of the paper is concluded in Sect. 6.

2. Fingercode Biometric Scheme

Fingercode has been demonstrated to be an effective fingerprint biometric scheme, which can capture both local and global details in a fingerprint [8]. The fingercode biometric scheme uses a bank of Gabor filters [9] to capture both local and global details in a fingerprint as a fingercode, which is represented by a fixed-length vector. The fingercode scheme divides the fingerprint image of a user into a number of sectors (see Fig. 4), applies Gabor filters to transform the image in each sector; then obtains a real value from each sector by computing the standard deviation for each sector. This results in a k-dimensional real vector where k is the number of sectors on the fingerprint.

The fingercode generation process [8] can be summarized by the following steps:

Step 1: Locate the core and registration points and determine the region of interest for the fingerprint image. The core point of a fingerprint is defined as the point of maximum curvature of the concave ridges in the fingerprint image [27]. For fingercode encoding, [8] located the core point based on multi-resolution analysis of the orientation fields of the fingerprint image. Since most of the category information in a fingerprint lies in the lower part of the fingerprint, the core point is shifted 40 pixels down to get the registration point.

Step 2: Tessellate the region of interest around the registration point. The region of interest is divided into a series of B concentric bands and each band is subdivided into k sectors. The inner most band is not used because it contains very few pixels. In our experiments, we use four bands (B = 4). Each band is 20-pixel wide and segmented into sixteen sectors (k = 16) (see Fig. 4), thus resulting in a total of 16 x 4 = 64 sectors.

Step 3: Normalize the region of interest in each sector to a constant mean and variance. Let I(x, y) denotes the gray value at pixel (x, y), M_i and V_i, the mean and variance of the gray values in sector S_i, and N_f(x, y), the normalized gray value at pixel (x, y). For all the pixels in sector S_i, the normalized image is defined as:

\[ N_f(x, y) = \begin{cases} M_0 + \frac{\sqrt{V_0(x, y) - M_0^2}}{V_i} & I(x, y) > M_i \\ M_0 - \frac{\sqrt{V_0(x, y) - M_0^2}}{V_i} & \text{otherwise} \end{cases} \]

where M_0 and V_0 are the desired mean and variance values, respectively. The values of M_0 and V_0 should be the same for all the test and template sets. In our experiments, we set the values of both M_0 and V_0 to 100 like [8]. The normalization process removes the effects of sensor noise and gray level deformation due to the finger pressure differences;

Step 4: Filter the region of interest in four different directions using a bank of Gabor filters to produce a set of four filtered images. Properly tuned Gabor filters can remove noise, preserve the true ridge and valley structures, and provide information contained in a particular orientation in the image. The typically used even symmetric Gabor filter [8], [9] has the following general form in the spatial domain:

\[ G(x, y; f, \theta) = \exp \left(-\frac{1}{2} \left[ \frac{x'^2}{\sigma_x^2} + \frac{y'^2}{\sigma_y^2} \right] \right) \cos(2\pi f x') \]

\[ x' = x \sin \theta + y \cos \theta \]
\[ y' = x \cos \theta - y \sin \theta \]

where f is the frequency of the sinusoidal plane wave along the direction \( \theta \) from x-axis, and \( \sigma_x \) and \( \sigma_y \) are the space constants of the Gaussian envelope along x and y axes, respectively.

Step 5: For each filtered output, compute the standard deviation in individual sectors in filtered images to form the fingercode. Let C_\theta(x, y) denotes the value of the filtered image corresponding to \( \theta \) at pixel (x, y) for sector S_i, the standard deviation is defined as:

\[ F_\theta = \sqrt{\sum_k (C_\theta(x, y) - M_\theta)^2} \]
where \( k_i \) is the number of pixels in \( S_j \) and \( M_{ij} \) is the mean of \( C_{ij}(x, y) \) in \( S_j \). For the fingerprint image, sixty-four features of each of the four filtered images provide a total of 256 (64 \( \times \) 4) features. Hence the fingerprint of the user is represented by a collection of four (4 filters) 64-dimensional real vectors.

To perform fingercode-based fingerprint classification, a user’s fingercode is generated from his fingerprint image. A K-nearest neighbor classifier is firstly used to find the K nearest templates of this fingercode and assign two most possible classes which are most frequently represented among the K nearest templates. Then a set of 10 trained neural networks are used to solve the 10 (C_5^2) different two-class problems, which selects the final class from the two classes determined in the first stage. This research focuses on the first stage of the fingercode-based fingerprint classification.

### 3. Fast VQ Search Methods

In the first stage of fingercode-base fingerprint classification, the classification system uses a K-nearest neighbor classifier to find the K nearest templates of the given fingerprint and assign two most possible classes which are most frequently represented among the K nearest templates. If the full search (FS) algorithm [11] is used, this process needs to compute the squared Euclidean distance between the fingercode of the given fingerprint and each of the templates in the database, thus it is computation-intensive.

To enhance the efficiency of fingercode-based fingerprint classification, we investigate how the various fast search algorithms in VQ [13] can improve the efficiency of the K-nearest neighbor classifier. In essence, the classification system needs to search through the fingercode database for the fingercodes which are the nearest (in terms of Euclidean distance) to the given fingerprint. This search problem is similar to the encoding process in VQ.

VQ is a mapping \( Q \) of a k-dimensional Euclidean space \( \mathbb{R}^k \) into a certain finite subset \( C \) of \( \mathbb{R}^k \), where \( C \) is the codebook with size \( N \) and each codeword \( c_i = \{c_{i1}, c_{i2}, \ldots, c_{ik}\} \) in \( C \) is \( k \)-dimensional. The codeword searching problem in VQ is to assign one codeword to the input test vector such that the distance between this codeword and the test vector is the smallest among all codewords. Given one codeword \( c_i = \{c_{i1}, c_{i2}, \ldots, c_{ik}\} \) and the test vector \( x = \{x_1, x_2, \ldots, x_k\} \), the squared Euclidean distance can be expressed as follows:

\[
d^2(x, c_i) = \sum_{j=1}^{k} (x_j - c_{ij})^2.
\]

In the fingerprint classification system, since a fingercode typically has several hundred dimensions, and there are thousands of templates in the database, the searching process is computation-intensive if the FS algorithm is used. From the above equation, each distance calculation needs \( k \) multiplications and \( 2k - 1 \) additions. In the first stage, the FS method [11] computes the squared Euclidean distances between the given fingercode and each template and determines the K nearest ones, which have the minimal squared Euclidean distances. To find the K nearest templates, the FS method needs \( N \) distance computations and at least \( N - 1 \) comparisons. In other words, it must perform \( kN \) multiplications, \((2k - 1)N\) additions and at least \( N - 1 \) comparisons, which is time-consuming.

In order to accelerate the VQ encoding process, many fast methods [13], [14], [16]-[26] have been proposed. As analyzed in Sect. 1, not all VQ schemes can be applied directly to fingercode-based classification. We only investigate those fast VQ encoding methods [13]-[22] suitable for fingercode-based classification. These methods use an “estimate” of the Euclidean distance to quickly determine whether a codeword can be eliminated thus the actual Euclidean distance need not be computed. Suppose the running minimum for the input vector \( x \) is \( d_{\text{min}} \), if the estimation for the Euclidean distance between \( x \) and current codeword \( c_i \) is larger than \( d_{\text{min}} \) and the corresponding real Euclidean distance is larger than the estimation, then we can safely reject \( c_i \) and avoid computing the actual Euclidean distance.

For example, the partial distance search (PDS) algorithm [13], [16] uses the following rejection test:

\[
\sum_{j=1}^{t} (x_j - c_{ij})^2 \geq d_{\text{min}}^2, \text{ for any } t < k.
\]

If the partially calculated squared Euclidean distance from dimensions 1 to \( t \) is greater than the running minimum \( d_{\text{min}} \), this codeword can be rejected without calculating the actual one in \( k \) dimensions.

As another example, the subvector (SV) method [17], [19] divides \( x \) and \( c_i \) into the first and second subvectors as \( x_f = \{x_1, x_2, \ldots, x_{k/2}\}, x_s = \{x_{k/2+1}, \ldots, x_k\}, c_{if} = \{c_{i1}, c_{i2}, \ldots, c_{ik/2}\}, c_{is} = \{c_{ik/2+1}, \ldots, c_{ik}\} \), respectively. The sums, means and the variances of \( x \) and \( c_i \) are defined as \( S_x = \sum_{j=1}^{k} x_j, S_{ci} = \sum_{j=1}^{k} c_{ij}, M_x = S_x/k, M_{ci} = S_{ci}/k, V_x = \sqrt{\sum_{j=1}^{k}(x_j - M_x)^2}, V_{ci} = \sqrt{\sum_{j=1}^{k}(c_{ij} - M_{ci})^2} \). Similarly, the partial sums of each subvector are defined as \( S_{xf} = \sum_{j=1}^{k/2} x_j, S_{xs} = \sum_{j=k/2+1}^{k} x_j, S_{cif} = \sum_{j=1}^{k/2} c_{ij}, S_{cis} = \sum_{j=k/2+1}^{k} c_{ij} \).

The SV method uses the following 3-step rejection test flow. If any inequality holds, it rejects \( c_i \) as the “nearest” codeword.

**Step 1.** \((S_x - S_{ci})^2 \geq kd^2_{\text{min}}\).

**Step 2.** \((S_x - S_{ci})^2 + k(V_x - V_{ci})^2 \geq kd^2_{\text{min}}\).

**Step 3.** \((S_x - S_{ci})^2 - 2(S_x - S_{ci}) \times ((S_x - S_{ci}) - (S_x - S_{cis})) \geq (k/2)d^2_{\text{min}}\).

In order to realize recursive computation in a memory efficient way, a 2-pixel-merging sum pyramid (SP) as shown in Fig. 5 was proposed in [20] for codeword search in VQ. As interpreted in [20], this SP needs no extra memory, in other words, it only needs \( k \) memories for a \( k \)-dimensional vector. A hierarchical rejection rule is set up as

\[
d^2(x, c_i) = d^2_m(x, c_i) \geq \cdots \geq 2^{-((u-1)d^2_m(x, c_i))}.
\]
The squared Euclidean distance at the \(v\)th level of the hierarchy for \(v \in [0, u_1]\) is \(d_2^v(x, c_i) = d_2^{u_1}(x, c_i) - 2^{-v} \sum_{m=1}^{u_1} (S_{x,v,m} - S_{c_i,v,m})^2\) where \(S_{x,v,m}\) is the \(m\)th pixel at the \(v\)th level for \(x\) and \(S_{c_i,v,m}\) similarly defined for \(c_i\). For a \(k\)-dimensional vector, \(u_1 = \log_2 k\).

Thus, at any \(v\)th level for \(v \in [0, u_1]\), if the inequality \(d_2^v(x, c_i) \geq d_2^{\text{min}}\) holds, then \(c_i\) can safely be rejected at the \(v\)th level.

As another example of pyramid data structures for codeword search in VQ, a L2-norm pyramid (NP) [18] was proposed as shown in Fig. 6. As interpreted in [18], this NP needs \((k-1)/3\) extra memories for a \(k\)-dimensional vector.

As mentioned, existing VQ techniques cannot be readily applied to solve the fingerprint classifier problem because the software parameter of this problem is significantly different from conventional image compression. For the latter, the dimension of vectors representing image blocks is small, typically 16 dimensions; whereas, for the former, fingerprint tends to have a dimensionality of several hundreds. In this respect, we firstly investigate some suitable fast VQ encoding methods and evaluate their performance for fingerprint identification.

In this study, we use benchmarking data set DB1 from FVC 2004 [28] for our experiments. There are a total of 100 fingers and 8 impressions per finger (800 impressions) in this database. We use the first impression of each finger to test the performance of the PDS method [16], the SV method [19], the SP method [20] and the NP method [18]. We rotate all the 800 impressions by an angle of 11.25°, which together with the other 700 original impressions are used to generate the templates, so there are totally 1500 templates in the database. All methods are implemented in Matlab and executed on a Pentium III PC.

As shown in Fig. 7, it is clear that the SP and NP methods are more efficient and scalable for fingerprint classification. Although the NP method is more efficient than the SP method, it needs \((k - 1)/3\) extra memories for a \(k\)-dimensional vector. Besides, since there will be more levels for fingerprints than the vectors in VQ and using the first several levels can hardly reject enough fingerprints, we propose two new schemes, namely the truncated SP (TSP) and the truncated NP (TNP) based on SP and NP respectively, for the fingerprint classification system. To achieve the improvement, the TSP and TNP schemes begin the computation of the squared Euclidean distance at the \(l\)th level instead of the 0th level. The TSP scheme uses the following hierarchical rejection rule,

\[
d^2(x, c_i) = d^2_{u_1}(x, c_i) - 2^{-v} \sum_{m=1}^{u_1} (L_{x,v,m} - L_{c_i,v,m})^2 - ||c||^2
\]

where \(L_{x,v,m}\) is the \(m\)th pixel at the \(v\)th level for \(x\) and \(L_{c_i,v,m}\) similarly defined for \(c_i\). For a \(k\)-dimensional vector, \(u_2 = \log_4 k\). Thus, at any \(v\)th level for \(v \in [0, u_2]\), if the inequality \(d^2_v(x, c_i) \geq d^2_{\text{min}}\) holds, then \(c_i\) can safely be rejected at the \(v\)th level.

4. Fast Fingercode Search Algorithms Based on VQ
These two inequalities guarantee that the proposed TSP and TNP algorithms get the same K-nearest neighbors as the FS algorithm.

Since the TSP and TNP schemes are similar, we describe them together as follows:

Step 1: Convert each template \( c_i \) in the database to a \( n \times n \) matrix as shown in Fig. 8 \((n = 16 \text{ in our experiment})\).

Step 2: Construct TSP or TNP by computing the pyramid from the bottom level to the \( l \)-th level for each matrix computed in Step 1. The selection of start levels \( l \) for TSP and TNP are done by experiments (\( l = 4 \) for TSP, and \( l = 2 \) for TNP). For TNP, use one extra memory to store \( a_{c_i}^2 \), which is needed in the computation of the new distortion. Compared with NP, TNP can reduce 4 extra memories if \( l = 2 \). Each template in the database is associated with one TSP or TNP.

Step 3: For the given fingercode \( x \), TSP or TNP is constructed. Select the first \( K \) templates \( c_1, c_2, \ldots, c_K \) in the database to be the current \( K \) nearest templates to the given fingercode \( x \). Compute the squared Euclidean distances for TSP or the new distortion for TNP, sort them in the ascending order to get the running minimums \( d_{\min,1}, d_{\min,2}, \ldots, d_{\min,K} \) where \( d_{\min,i} \leq d_{\min,j} \) if \( i \leq j \), and temporarily store them. For TSP, further compute \( d_{\min,K}^2 = 2^{(u-v)}d_{\min,K}^2 \) for \( v \in [l, u_1] \) using the new \( d_{\min,K}^2 \).

Step 4: For any other template \( c_i \) in the database, execute the rejection tests from the \( l \)-th level to the bottom level. At the \( l \)-th level, if the squared Euclidean distance for TSP is greater than \( d_{\min,K}^2 \), this template can be rejected. If \( c_i \) goes through all the rejection tests and arrives the bottom level, then compute the squared Euclidean distances for TSP or the new distortion for TNP between the given fingercode and this template, and insert it into \( d_{\min,1}, d_{\min,2}, \ldots, d_{\min,K} \) to guarantee the ascending order and replace the old \( d_{\min,1}, d_{\min,2}, \ldots, d_{\min,K} \). For TSP, further compute \( d_{\min,K}^2 = 2^{(u-v)}d_{\min,K}^2 \) for \( v \in [l, u_1] \) using the new \( d_{\min,K}^2 \).

Step 5: If there is no more candidate template, the current \( K \) nearest templates are the real \( K \) nearest ones, and then assign two most possible classes which are most frequently represented among these \( K \) nearest templates.

Because of the test structure of the TSP and TNP algorithms, our algorithms are obviously more efficient than the original SP and NP algorithms, respectively. The efficiency of our algorithms is illustrated through experimental results to be presented in the next section.

5. Experimental Results

Experiments were performed to compare the efficiency of our algorithms with that of the FS algorithm, the original SP and NP algorithms respectively. In our experiments, the same parameters as in Sect. 4 were used. In the first experiment, we compare the average online execution time of the 100 tested fingercodes required for different start levels using 200 templates and 1500 templates respectively. Table 1 illustrates the respective performance comparisons for TSP where the time unit is denoted by ‘s’. As can be seen from Table 1, the appropriate truncation level of TSP is 4 \((l = 4)\). The TSP algorithm degrades to the FS algorithm if we start the computation of the squared Euclidean distance at the bottom level. Table 2 illustrates the respective performance comparisons for TNP. As can be seen from Table 2, the appropriate truncation level of TNP is 2 \((l = 2)\).

In the second experiment, the improvement ratio in terms of execution time required in the proposed TSP algorithm over the FS and original SP algorithms are denoted by

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Improvement Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>FS</td>
<td>( R_1 = \frac{T_{FS} - T_{TSP}}{T_{FS}} \times 100% )</td>
</tr>
<tr>
<td>SP</td>
<td>( R_2 = \frac{T_{SP} - T_{TSP}}{T_{SP}} \times 100% )</td>
</tr>
</tbody>
</table>

respectively. Here \( T_{FS}, T_{SP} \) and \( T_{TSP} \) denote the average online execution time of the 100 tested fingercodes required in the FS algorithm, the SP algorithm and the proposed TSP algorithm, respectively. Table 3 illustrates the respective performance comparisons. As can be seen from Table 3, the proposed TSP algorithm performs more efficiently than the original SP algorithm. More importantly, experimental results also illustrate the scalability of our algorithm in the sense that the larger the number of the template is, the better...
Table 1 Average computation time for TSP with different start levels.

<table>
<thead>
<tr>
<th>(\ell)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>0.34</td>
<td>0.33</td>
<td>0.32</td>
<td>0.30</td>
<td>0.29</td>
<td>0.31</td>
<td>0.34</td>
<td>0.39</td>
<td>0.56</td>
</tr>
<tr>
<td>1500</td>
<td>1.30</td>
<td>1.21</td>
<td>1.11</td>
<td>1.02</td>
<td>0.99</td>
<td>1.20</td>
<td>1.67</td>
<td>2.70</td>
<td>4.41</td>
</tr>
</tbody>
</table>

Table 2 Average computation time for TNP with different start levels.

<table>
<thead>
<tr>
<th>(\ell)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>0.30</td>
<td>0.29</td>
<td>0.28</td>
<td>0.33</td>
<td>0.43</td>
</tr>
<tr>
<td>1500</td>
<td>1.02</td>
<td>0.97</td>
<td>0.94</td>
<td>1.64</td>
<td>3.35</td>
</tr>
</tbody>
</table>

Table 3 Comparison of average online execution time per query fingercode for SP.

<table>
<thead>
<tr>
<th>Number of templates</th>
<th>(T_{FS}) (s)</th>
<th>(T_{SP}) (s)</th>
<th>(T_{TSP}) (s)</th>
<th>(R_1) %</th>
<th>(R_2) %</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.28</td>
<td>0.21</td>
<td>0.19</td>
<td>33.9</td>
<td>11.4</td>
</tr>
<tr>
<td>200</td>
<td>0.56</td>
<td>0.34</td>
<td>0.29</td>
<td>49.0</td>
<td>14.8</td>
</tr>
<tr>
<td>300</td>
<td>0.85</td>
<td>0.43</td>
<td>0.36</td>
<td>57.4</td>
<td>17.1</td>
</tr>
<tr>
<td>400</td>
<td>1.13</td>
<td>0.52</td>
<td>0.43</td>
<td>62.1</td>
<td>18.4</td>
</tr>
<tr>
<td>500</td>
<td>1.41</td>
<td>0.60</td>
<td>0.48</td>
<td>65.7</td>
<td>19.8</td>
</tr>
<tr>
<td>600</td>
<td>1.69</td>
<td>0.69</td>
<td>0.55</td>
<td>67.6</td>
<td>20.5</td>
</tr>
<tr>
<td>700</td>
<td>1.97</td>
<td>0.77</td>
<td>0.61</td>
<td>69.1</td>
<td>21.1</td>
</tr>
<tr>
<td>800</td>
<td>2.25</td>
<td>0.85</td>
<td>0.66</td>
<td>70.5</td>
<td>21.7</td>
</tr>
<tr>
<td>900</td>
<td>2.53</td>
<td>0.92</td>
<td>0.72</td>
<td>71.7</td>
<td>22.0</td>
</tr>
<tr>
<td>1000</td>
<td>2.88</td>
<td>0.99</td>
<td>0.77</td>
<td>73.3</td>
<td>22.4</td>
</tr>
<tr>
<td>1100</td>
<td>3.29</td>
<td>1.05</td>
<td>0.81</td>
<td>75.2</td>
<td>22.8</td>
</tr>
<tr>
<td>1200</td>
<td>3.57</td>
<td>1.12</td>
<td>0.86</td>
<td>75.9</td>
<td>23.0</td>
</tr>
<tr>
<td>1300</td>
<td>3.85</td>
<td>1.17</td>
<td>0.90</td>
<td>76.6</td>
<td>23.3</td>
</tr>
<tr>
<td>1400</td>
<td>4.13</td>
<td>1.24</td>
<td>0.94</td>
<td>77.1</td>
<td>23.5</td>
</tr>
<tr>
<td>1500</td>
<td>4.41</td>
<td>1.30</td>
<td>0.99</td>
<td>77.5</td>
<td>23.6</td>
</tr>
</tbody>
</table>

Table 4 Comparison of average online execution time per query fingercode for NP.

<table>
<thead>
<tr>
<th>Number of templates</th>
<th>(T_{FS}) (s)</th>
<th>(T_{NP}) (s)</th>
<th>(T_{TNP}) (s)</th>
<th>(R_3) %</th>
<th>(R_4) %</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.28</td>
<td>0.19</td>
<td>0.18</td>
<td>36.7</td>
<td>7.5</td>
</tr>
<tr>
<td>200</td>
<td>0.56</td>
<td>0.30</td>
<td>0.28</td>
<td>50.7</td>
<td>7.5</td>
</tr>
<tr>
<td>300</td>
<td>0.85</td>
<td>0.37</td>
<td>0.35</td>
<td>59.0</td>
<td>7.2</td>
</tr>
<tr>
<td>400</td>
<td>1.13</td>
<td>0.44</td>
<td>0.41</td>
<td>63.5</td>
<td>7.2</td>
</tr>
<tr>
<td>500</td>
<td>1.41</td>
<td>0.50</td>
<td>0.46</td>
<td>67.0</td>
<td>7.0</td>
</tr>
<tr>
<td>600</td>
<td>1.69</td>
<td>0.57</td>
<td>0.53</td>
<td>68.9</td>
<td>7.1</td>
</tr>
<tr>
<td>700</td>
<td>1.97</td>
<td>0.63</td>
<td>0.58</td>
<td>70.4</td>
<td>7.0</td>
</tr>
<tr>
<td>800</td>
<td>2.25</td>
<td>0.69</td>
<td>0.64</td>
<td>71.8</td>
<td>7.2</td>
</tr>
<tr>
<td>900</td>
<td>2.53</td>
<td>0.74</td>
<td>0.69</td>
<td>72.9</td>
<td>7.3</td>
</tr>
<tr>
<td>1000</td>
<td>2.88</td>
<td>0.80</td>
<td>0.74</td>
<td>74.4</td>
<td>7.2</td>
</tr>
<tr>
<td>1100</td>
<td>3.29</td>
<td>0.84</td>
<td>0.78</td>
<td>76.3</td>
<td>7.3</td>
</tr>
<tr>
<td>1200</td>
<td>3.57</td>
<td>0.89</td>
<td>0.82</td>
<td>77.0</td>
<td>7.2</td>
</tr>
<tr>
<td>1300</td>
<td>3.85</td>
<td>0.92</td>
<td>0.86</td>
<td>77.7</td>
<td>7.1</td>
</tr>
<tr>
<td>1400</td>
<td>4.13</td>
<td>0.97</td>
<td>0.90</td>
<td>78.2</td>
<td>7.2</td>
</tr>
<tr>
<td>1500</td>
<td>4.41</td>
<td>1.02</td>
<td>0.94</td>
<td>78.6</td>
<td>7.2</td>
</tr>
</tbody>
</table>

In the third experiment, the improvement ratio in terms of execution time required in the proposed TNP algorithm over the FS and original NP algorithms are denoted by:

\[
R_3 = \frac{T_{FS} - T_{TNP}}{T_{FS}} \times 100\% \\
R_4 = \frac{T_{NP} - T_{TNP}}{T_{NP}} \times 100\%
\]

respectively. Here \(T_{FS}\), \(T_{NP}\) and \(T_{TNP}\) denote the average online execution time of the 100 tested fingercodes required in the FS algorithm, the NP algorithm and the proposed TNP algorithm, respectively. Table 4 illustrates the respective performance comparisons. As can be seen from Table 4, the proposed TNP algorithm can save nearly 7% computation compared with the original NP algorithm.

Furthermore, although the TNP algorithm is more efficient than the TSP algorithm, as shown in the analysis in Sect. 3 and Sect. 4, TSP needs no extra memory, but TNP needs \((k - 1)/3 - 4\) extra memories. Hence, the TNP algorithm is more suitable for the fingerprint classification systems with a small number of templates, but the TSP algorithm is more suitable for the ones with a large number of templates.

6. Conclusions

In this paper, we investigate how the various fast search algorithms in vector quantization (VQ) can improve the efficiency of the \(K\)-nearest neighbor classifier for fingercode-based fingerprint classification, and propose two efficient algorithms based on the pyramid-based search algorithms in VQ.

Experimental results on DB1 of FVC 2004 [28] demonstrated that our algorithms can outperform the full search algorithm and the original pyramid-based search algorithms in terms of computation efficiency without sacrificing accuracy. As a continuing effort to further enhance the fingercode classification system, future research in this direction may investigate the computation of representative templates using various codebook design algorithms in VQ.

References

SUN et al.: EFFICIENT FINGERCODE CLASSIFICATION


Hong-Wei Sun is a Ph. D. student in School of Software, Tsinghua University, P.R. China. He received his B.S. degree in civil engineering in 2002 and the M.S. degree in computer science and technology in 2004 both from Tsinghua University. His research interests include information security and biometric recognition.

Kwok-Yan Lam is the Director of Key Laboratory for Information System Security, Ministry of Education, P.R. China. He has been a Professor at the School of Software, Tsinghua University, P.R. China since 2002. Prior to joining the Tsinghua University, Professor Lam has been a faculty member of the National University of Singapore and the University of London since 1990. His main research interests include distributed systems, information security and tamper-resistant design. Professor Lam was a visiting scientist at the Isaac Newton Institute of the Cambridge University and a visiting professor at the European Institute for Systems Security. He has been a chief security architect for a number of electronic banking and electronic government systems in Singapore and Hong Kong. Professor Lam received his B.Sc. (First Class Honours) from the University of London in 1987 and his Ph.D. from the University of Cambridge in 1990.

Dieter Gollmann received his Dipl.-Ing. in Engineering Mathematics (1979) and Dr.tech. (1984) from the University of Linz, Austria. He was a Lecturer in Computer Science at Royal Holloway, University of London, and later a scientific assistant at the University of Karlsruhe, Germany. He was a Visiting Professor at the Technical University of Graz in 1991, an Adjunct Professor at the Information Security Research Centre, QUT, Brisbane, in 1995, and has acted as a consultant for HP Laboratories (Bristol). He joined Microsoft Research in Cambridge in 1998. In 2003, he took the chair for Security in Distributed Applications at the Technical University Hamburg-Harburg, Germany. He is a Visiting Professor with the School of Software at the Tsinghua University, as well as the Information Security Group at Royal Holloway. He is also an Adjunct Professor at the Technical University of Denmark. Dieter is one of the editors-in-chief of the International Journal of Information Security and an associate editor of the IEEE Security & Privacy Magazine. He has authored a textbook on ‘Computer Security’.

Siu-Leung Chung is an Associate Professor at the School of Business and Administration, The Open University of Hong Kong. Prior to joining The Open University, he has been a faculty member of the National University of Singapore and the University of Toledo, Ohio, USA since 1991. His main research interests are information and system security, e-commerce security and software project management modelling. Dr. Chung received his B.Sc. (Engineering) from the University of Hong Kong in 1977 and his Ph.D. in Computer Science from the University of Illinois in 1991.
Jian-Bin Li is a Senior Engineer of the Information Centre, State Administration of Taxation, P.R. China. He is a Visiting Professor of the School of Software, Tsinghua University; and a Visiting Researcher of the Software Institute, Academia Sinica. He specializes in design and planning of very large scale networks and information security management. His research interests include information security management, security risk analysis and disaster recovery planning.

Jia-Guang Sun is a Professor in School of Software, Tsinghua University, PR China. Professor Sun, widely regarded as a leader in CAD research in China, is a Fellow of the Chinese Academy of Engineering. Professor Sun is the Director of the National Engineering Research Center for CAD Supporting Software and is concurrently the Dean of the School of Software of Tsinghua University.