Generation and Storage of Large Synthetic Fingerprint Database

A Project Report

Submitted in partial fulfilment of the

requirements for the Degree of

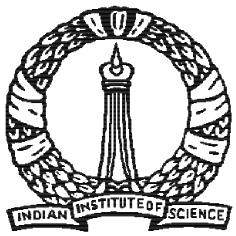
Master of Engineering

in

Computer Science and Engineering

by

Afzalul Haque Ansari



Department of Computer Science

Indian Institute of Science

Bangalore – 560 012

July 2011

TO

My parents

Acknowledgements

I would like to express my sincere gratitude to my advisor, Prof. Jayant Haritsa. I want to thank him for his patient guidance, encouragement and advice which made this thesis possible.

Abstract

Estimated size of database for UIDAI's project[9] is 1.2 billion rows, larger than the current largest fingerprint based biometric database. To simulate and analyze different algorithm and methods that can be used to establish a complete UID system, a database of similar size is required. Our motive is to generate large database of synthetic fingerprints using parametric modeling of fingerprints [10][1][2] and querying this database using genuine and impostor query-fingerprints with fast and efficient near-neighbors search methods in order to estimate query-time and accuracy of such system.

Like other biometric databases our synthetic fingerprint database also suffers curse of dimensionality. Several different features associated with a fingerprint are represented as vectors in high dimensional spaces. Since fingerprint databases are massive in size and high-dimensional in nature, a really fast near-neighbors algorithm is required to find most similar fingerprints. Approximation based near-neighbors search methods provide time-accuracy trade-off. Storage and retrieval strategy adapted in this work is based on such a well known method, *Locality Sensitive Hashing*[11].

Contents

List of Figures

Chapter 1

Introduction

Fingerprints are most widely used biometric feature for identification and authentication. There are several efficient methods for fingerprint recognition and continuous research is going on to make it more accurate. All such methods are trained and tested with relatively small databases. Using small databases for training makes the accuracy statistics valid only for databases of similar size. So they can not be generalized for a large fingerprint database. Large fingerprint databases are not easily available due to public-security issues. As alternative to original large fingerprint databases, synthetic large fingerprint databases can be created using *SFinGe*[2] approach.

1.1 Synthetic Fingerprint Database

In our implementation of *SFinGe* algorithm we fixed several parameter for fingerprint modeling after inspecting several real fingerprints (details in Section 2). There are only few small databases of real fingerprints are available. So in-spite of Fingerprints generated by *SFinGe* approach being very realistic, an in-depth analysis is necessary to understand if it can be a valid substitute for real fingerprints for analyzing fingerprint recognition algorithms. For validation check, we have simulated fingerprint matching algorithms over synthetic database and compare statistics of results, such as distributions of matching scores and similarity-measures /distances, *genuine/impostor-distributions*,

FAR/FRR curves with those of simulation over real databases. These statistics were used to *'tune'* our synthetic database to ensure its validity for analysis, training and testing of matching algorithms.

We have generated large fingerprint database having 10 million fingerprints within couple of weeks. For our work we have used database of 1 million fingerprints.

1.2 Fingerprint Recognition

Fingerprint recognition algorithms rely on specific features extracted from fingerprint image. Few most frequently used features in fingerprint recognition systems are *FingerCode*[6] and *Minutiae*. There are several types of *FingerCode* features based on their dimensionality. *FingerCode* feature, we are using in our work, is a 192 dimensional vector. We have referenced these as **FC192**. *FingerCode* is based on global characteristics of fingerprints as texture and ridge curvature[3]. So this feature can be used to narrow down the search space according to global characteristics of query fingerprint.

Fingerprint matching for the purpose of identification involves searching most similar fingerprint in the database. It is a near-neighbors search problem which involves searching the data-object nearest to query object, where all objects are represented as points in high-dimensional feature space using Euclidean distance function. For our Fingerprint database high dimensionality of feature space makes it difficult to locate nearest fingerprint. Several indexing methods as R-tree and KD-tree also do not perform well for high dimension and they also suffer from high maintenance overhead [11].

It has been observed that for several cases getting the exact near neighbors is not necessary[11]. Even an approximately chosen near neighbors will be sufficient to solve the problem. In fact in high dimensional space the target point is much closer to the query point than other points. In such a scenario an approximately chosen near neighbors can work well with an accuracy similar to exact near neighbor.

Searching exact near neighbors requires more time than searching approximated near neighbors because many of such algorithms include calculating distance of each point

from query fingerprint. On the other hand hashing based approximate near neighbors searching method, LSH, pre-processes all data points in a way such that when a query point is given it returns the similar fingerprints quickly[12] (details in Section 4.1).

Chapter 2

Generation of Synthetic Fingerprint Database

2.1 Knowing Fingerprints

A fingerprint impression represents outer layer of skin of a finger. At macroscopic level fingerprint appears to be composed of several curved lines known as **ridge lines**. The region between two adjacent ridges is called **valley**.

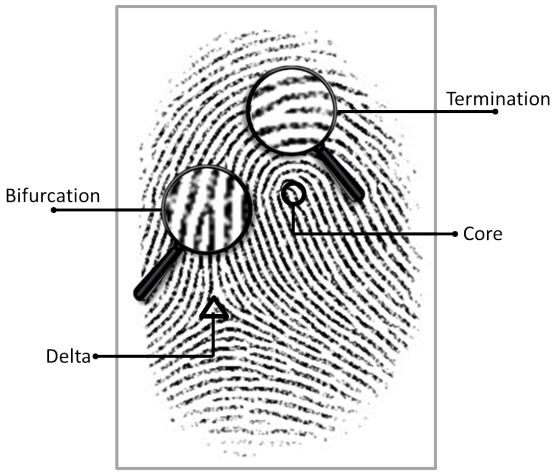


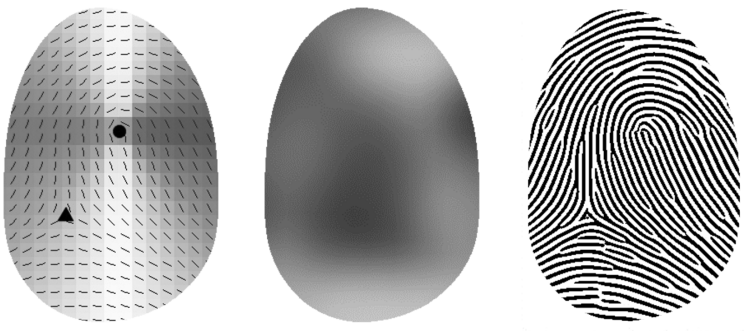
Figure 2.1: Macro-singularities and micro-singularities of a fingerprint

Ridges lines generally run smoothly in parallel but at one or more areas they exhibit special patterns such as high curvature and merging of ridge-flows from three directions. Points where ridges have highest curvature are known as **core points**. Core point is the point of maximum curvature on the inner-most curved ridge in the area of high curvature. The points with confluent ridge-flows known as **delta points** [3]. These points, cores and deltas, are called **macro-singularities** (Figure 2.1).

2.1.1 Global Features

Macro-singularities are important to identify global features, such as features based on ridge-flow pattern of fingerprints. The ridge-line flow can be effectively described by a structure called **directional map**. It is a matrix (Figure 2.2(a)) whose elements denote the orientation of the tangent to the ridge lines at corresponding points of fingerprint image .

The ridge line density can be represented using a **density map** (Figure 2.2(b)). The local ridge frequency or density $f_{x,y}$ at point [x, y] is the number of ridges per unit length along a hypothetical segment centered at [x, y] and orthogonal to the local ridge orientation $\theta_{x,y}$ [3].



(a) Orientation map (b) Density map (c) Ridge pattern

Figure 2.2: Global features of fingerprint

2.1.2 Local Features

Minutiae points are the irregularities and discontinuities in the ridge flow pattern. The two most important types of irregularities are **ridge endings** or **terminations** and **ridge division** or **bifurcations** (Figure 2.1). Ridge endings are the points where the ridge curve terminates, and bifurcations are where a ridge splits from a ridge to two ridges and creates a *Y-junction*. Minutiae points are also called **micro-singularities** and these singularity points are local feature of fingerprints.

2.2 Fingerprints Classification

Fingerprints are usually partitioned into five main classes (Figure 2.3) according to the presence and position of their macro-singularities [3].

Arch in this type fingerprints do not have any macro singularity point.

Tented arch contains one pair of core and delta points. The axis of symmetry passes through the delta point. Delta point may be replaced by a delta region in order to handle error in calculating symmetry axis.

Left loop contains one pair of core and delta points. The delta point is on right side of the axis of symmetry.

Right loop contains one pair of core and delta points. The delta point is on left side of the axis of symmetry.

Whorl contains two pairs of core and delta points.

(a) Arch (b) Tented arch (c) Left loop

(d) Right loop (e) Whorl

Figure 2.3: Generation of fingerprint impression

2.3 Fingerprints Generation

SFinGe is a method for generating synthetic fingerprints on the basis of mathematical models that describe the main features of real fingerprints. It is only available application for synthetic fingerprint generation and it is very popular in biometric community.

We have implemented *SFinGe* method with an aim to generate large fingerprint database. The synthetic images are randomly generated according to given parameters based on fingerprint features. The approach is able to generate very realistic fingerprints, which can be useful for performance evaluation and testing of fingerprint-based systems.

The basic approach of *SFinGe* (Figure 2.4) method includes:

1 Generation of a directional map and density map separately. These features are combined to obtain a fingerprint pattern using a specific filtering procedure.

2 Generation of fingerprint shape (mask). Master-fingerprint is finally made more realistic by masking it with fingerprint-shape and adding fingerprint specific noises.

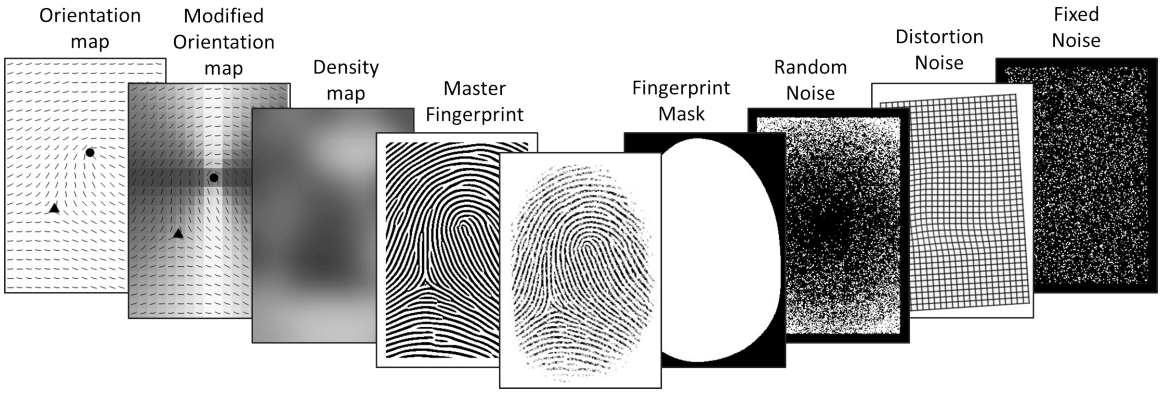


Figure 2.4: The *SFinGe* approach

Step 2 can be applied several times separately on the fingerprint generated in first step to generate different impression of same fingerprint.

2.3.1 Generation of master fingerprint

A master fingerprint is a synthetic fingerprint with unique ridge-flow pattern without any noise. Different parameters used in generation process are described in following subsections.

Orientation field

Sherlock and Monroe [4] proposed a Zero-pole Model for orientation field estimation that allows a directional map (Figure 2.5) to be calculated from the position of the core and delta points only.

In this model, core is considered as zero, and delta is considered as pole in the complex plane and orientation at any point z in complex plane is given by:

$$o(z) = \left[o_0 + \frac{1}{2} \left(\sum_{j=1}^{n} arg(z - z_{dj}) - \sum_{j=1}^{m} arg(z - z_{cj}) \right) \right]$$

where z_{cj} is j^{th} core-singularity and z_{dj} is j^{th} delta-singularity in complex plane and o_0 is a constant.

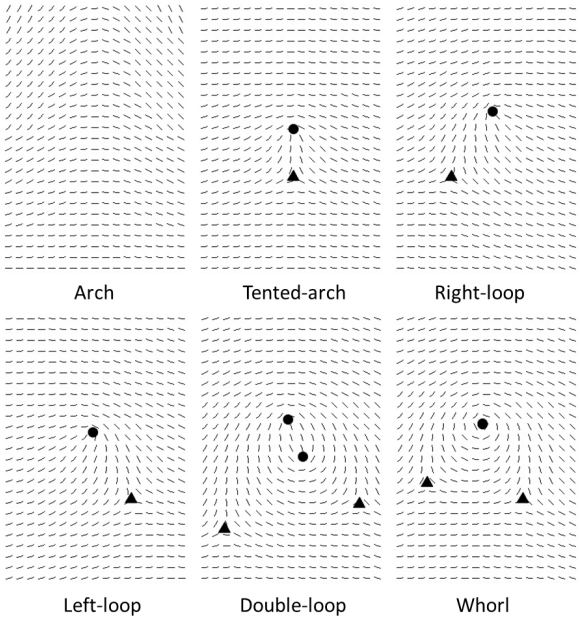


Figure 2.5: Orientation field using Sherlock and Monroe model

Vizcaya and Gerhardt [5] proposed a variant of the Sherlock and Monro model that introduces modification in orientation field and adds more degrees of freedom to cope with the variations in orientation field for different fingerprint. The orientation θ at each point z is calculated as

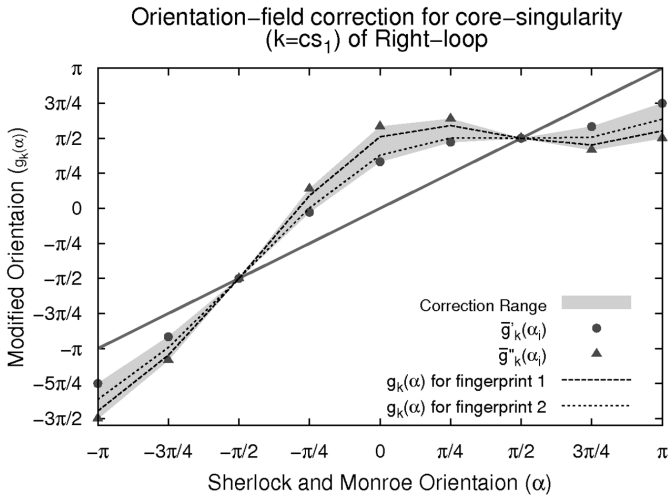
$$\theta = \frac{1}{2}\left[\sum_{i=1}^{n_d} g_{ds_i}(arg(z - ds_i)) - \sum_{i=1}^{n_l} g_{cs_i}(arg(z - cs_i))\right]$$

where functions $g_k(\alpha)$, defined for different delta and core singularity $k \in \{ds_1, ds_2, ..., ds_{n_d}, cs_1, cs_2, ...$ are piecewise linear functions for local correction of the orientation field with respect to the value given by the Sherlock and Monroe model.

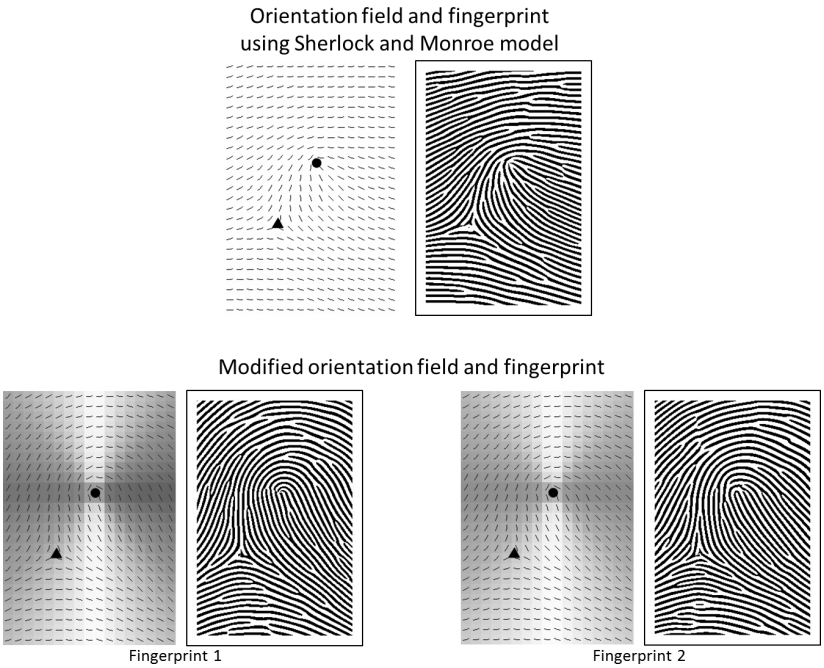
Each function $g_k(\alpha)$ is defined by the set of values $\{\overline{g}_k(\alpha_i)|i = 0, 1, 2, .., L-1\}$ where each value is the amount of correction of the orientation image at a given angle.

$$g_k(\alpha) = \overline{g}_k(\alpha_i) + \frac{\alpha - \alpha_i}{2\pi/L}(\overline{g}_k(\alpha_{i+1}) - \overline{g}_k(\alpha_i))$$

$$\text{for } \alpha_i \leq \alpha \leq \alpha_{i+1}, \alpha_i = -\pi + \tfrac{2\pi i}{L}$$



(a) Orientation field correction



(b) Different fingerprints with different modification in Sherlock and Monroe orientation field

Figure 2.6: Orientation field modification

Mapping from interval $[\alpha_i\ ,\ \alpha_{i+1}]$ to $[\overline{g}_k(\alpha_i)\ ,\ \overline{g}_k(\alpha_{i+1})]$ using a linear interpolation function entirely depends on k, where k is a specific singularity present in fingerprint. So modification for any interval must be defined separately for different singularity points. Since fingerprint classes are defined on the basis of position and number of singularities, we will have to define separate piecewise linear functions for every class depending upon singularity present in that class.

From the analysis of real fingerprints it is found that $L = 8$ is a reasonable value[5]. Problem is to find out $\overline{g}_k(\alpha_i)$ for α_i ($i \in \{0, 1, ...7\}$) for each fingerprint class and for each singularity k present in that class. Values $\overline{g}_k(\alpha_i)$ are also called control points, as these values guide the whole modification.

In our generation process in place of using fixed value $\overline{g}_k(\alpha_i)$ we have fixed a range of $[\overline{g'}_k(\alpha_i)\ ,\ \overline{g''}_k(\alpha_i)]$ for every α_i in order to keep orientation more randomized for every fingerprint class. We have fixed these ranges after analysis of real fingerprints of different fingerprint classes. An example of correction functions for core-point singularity of right loop is given in Figure 2.6 shows the modification in orientation field using correction parameters.

Density map

The local ridge density $f_{x,y}$ at point (x, y) is the number of ridges per unit length along a hypothetical line segment centered at (x, y) and orthogonal to the local ridge orientation $\theta_{x,y}$ [3].

Analyzing real fingerprint images, it is observed [2] that quite often, in the areas above the upper most loop and below the bottom most delta, the ridge density is lower and the ridges are thicker in these areas.

We are using density maps which are based on the thickness of the ridges. Generating a density map of size 400x275 requires 600 milliseconds. It is one of the costlier parts in whole generation process . To speed up our generation process we have generated 2000 random density maps (Figure 2.7) before-hand. During generation process we have used a random combination of two or three density maps. This way we are able to generate

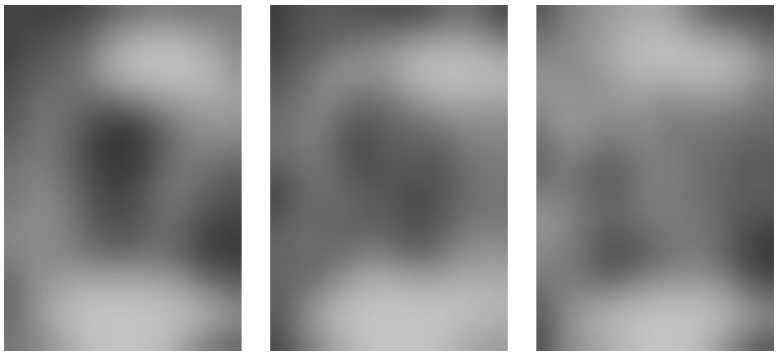


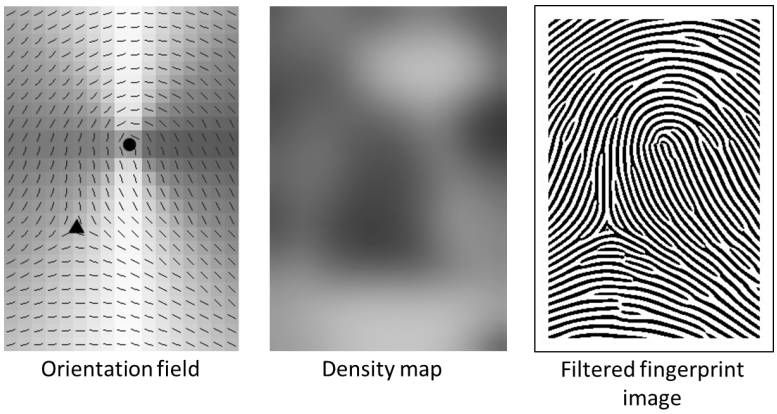
Figure 2.7: Density maps

total ${}^{2000}C_2 + {}^{2000}C_3$ different density-maps.

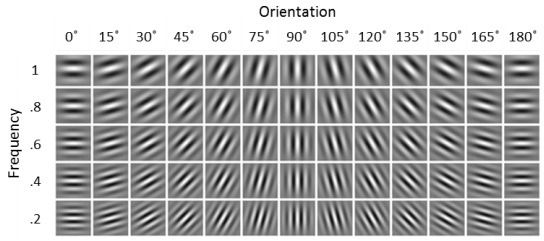
Ridge generation : Filtering

Initial image is created by randomly placing few black points into a white image as noise. By iteratively enhancing this initial image through Gabour filters [1][4], ridge pattern are created. For every filtering pass each point is filtered using a different Gabour filter based on density and orientation values at that point (Figure 2.8(b)). Iteratively applying *striped* filters to random images will produce striped images. This method generates very realistic minutiae at random positions because of different densities at different point. Due to change in orientation of filter, real-like ridge patterns are generated (Figure 2.8(a)).

For filtering an image with Gabour filter, filter has to be calculated on each point of image. At any point, Gabour filter used to filter that point depends on orientation and density value at that point. Calculating filter for every pixel is costly. We have used fixed number of discrete values for orientation and density map to reduce total number of different Gabour filter and a matrix for Gabour filters can be calculated before-hand. We have used a set of 18000 Gabour filters in our generation process for 100 different density values and 180 different orientations.



(a) Master fingerprint with orientation field and density map



(b) A sample Gabour Filter-bank

Figure 2.8: Generation of master fingerprint

2.3.2 Generating realistic fingerprints impression using fingerprint-specific noise

For each fingerprint impression (Figure 2.9(e)) to be generated from a given master fingerprint, following steps are performed sequentially[4][7]:

Distortion

Due to different placement and non-orthogonal pressure of finger on sensor surface, different skin deformations (Figure 2.9(b)) are introduced in different impressions of same finger. A mathematical model was given in[1][8] to generate this deformation.

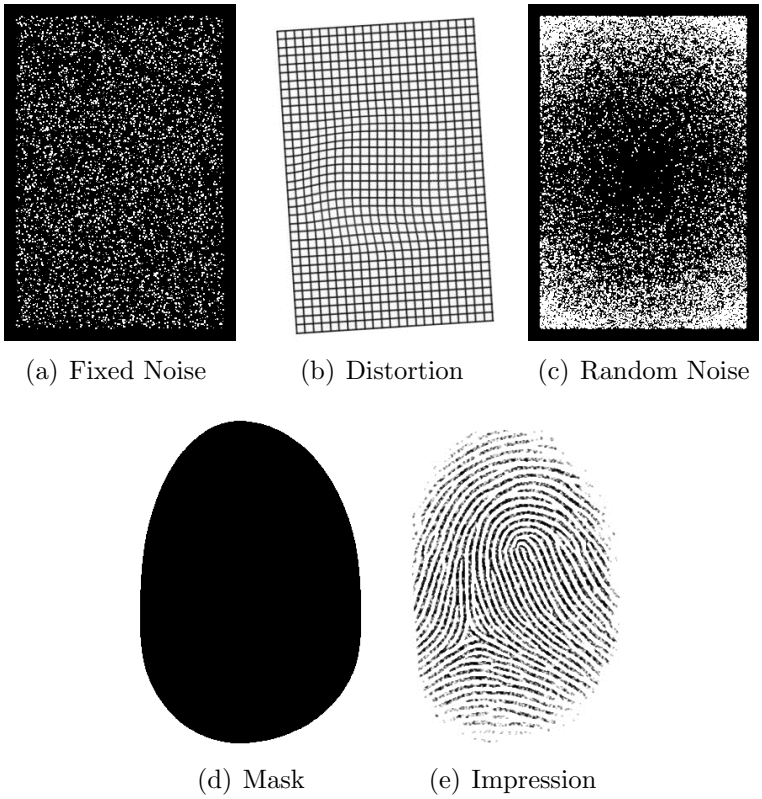
Noise generation

We have divided fingerprint noise in two parts:

- **Fixed noise** : It is the noise (Figure 2.9(a)) that we have associated with irregularity of the ridges, sweat pores and rough texture of fingerprint surface, which are fixed for a fingerprint.
- **Random noise** : This noise (Figure 2.9(c)) depends on environmental factor such as non uniform contact and pressure of fingerprint on the surface of capturing device.

Fingerprint mask

Depending on the finger size, position, and pressure against the fingerprint capturing device, acquired fingerprint images have different shapes (Figure 2.9(d)). A simple method for the fingerprint area has been introduced in [5]. It defines a model to define external shape or fingerprint-mask, based on four elliptical arcs and a rectangle.



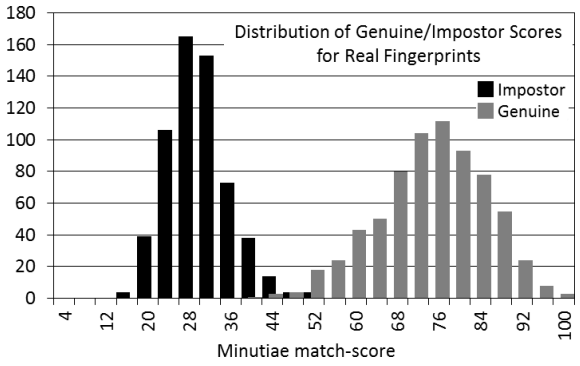
(a) Fixed Noise (b) Distortion (c) Random Noise

(d) Mask (e) Impression

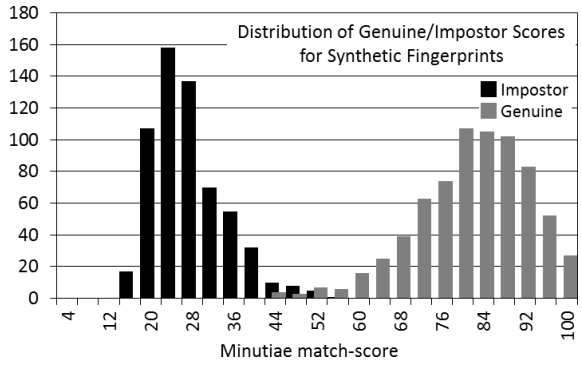
Figure 2.9: Generation of fingerprint impression

2.3.3 Validation of synthetic fingerprints

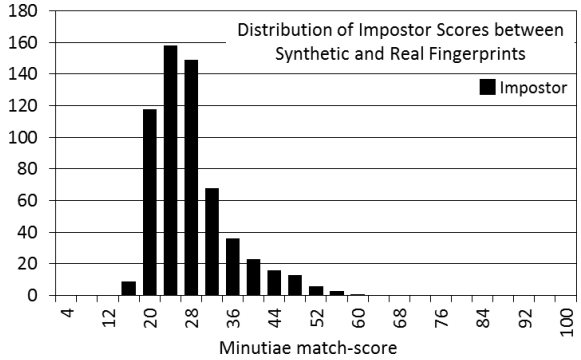
We have simulated minutiae based matching algorithms over real fingerprint database and synthetically generated fingerprint database. After analyzing the *impostor-distribution* for real (Figure 2.10(a)) and synthetic (Figure 2.10(b)) fingerprint databases it was found that distribution of synthetic fingerprints and real fingerprints are same in feature space. We have also analyzed the distribution of distances among synthetic and real fingerprints (Figure 2.10(c)). We found it similar to the *impostor-distribution* of real database and synthetic database. It means not only their distributions are same but they are also homogeneously located in feature space.



(a) Distributions of scores for real fingerprints



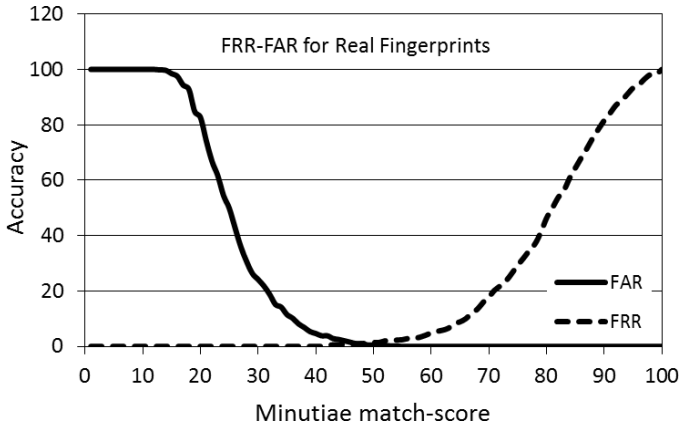
(b) Distributions of scores for synthetic fingerprints



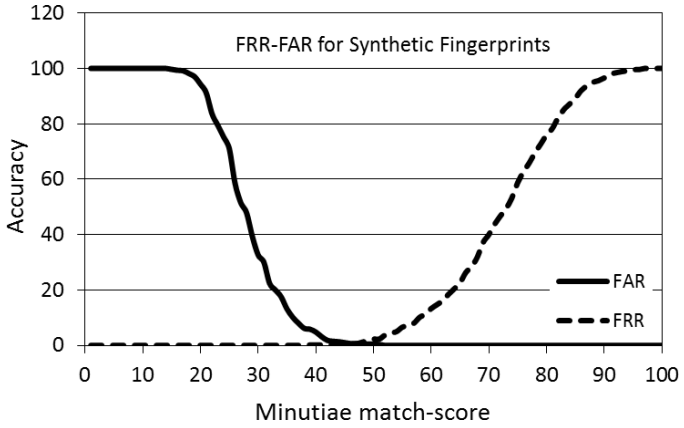
(c) Distributions of scores between real and synthetic fingerprints

Figure 2.10: Distributions of genuine and impostor scores for different databases

We have also studied *FAR* and *FAR* curves for synthetic fingerprints databases and real fingerprint databases. Their accuracy indicators as *CER (Common Error Rate)* , *zeroFAR (Zero False Acceptance Rate)*, *zeroFRR (Zero False Rejection Rate)* are almost the same (Figures 2.11(a) anf 2.11(b)). Performance of our synthetic database is similar to real fingerprint databases and it can be used to calculate accuracy estimates.



(a) FAR curves for different databases



(b) FRR curves for different databases

Figure 2.11: FAR and FRR curves for different databases

2.3.4 Generation Speed

Apart from generating very realistic fingerprint, using parameters determined by analyzing real fingerprints, our main effort invested in synthetic fingerprint generation was to speed up the generation process. The implementation of actual *SFinGe* algorithm by its authors is available for trail. The fastest generated fingerprint using trail version of *SFinGe* takes more than 3 seconds to be generated. Our implementation on average takes 400 millisecond to generate a fingerprint impression and 300 milliseconds to add distortion-noise to it. It was claimed in [3] that *full version of SFinGe can generate a database of 100,000 fingerprints (10,000 fingers, 10 impressions per finger), using 10 3GHz PCs in a network, in less than 2 Hr.* Estimated time for generating same amount of fingerprint with similar hardware scenario using our implementation is 1 Hr.

Chapter 3

Fingerprint Feature

3.1 FingerCode

We have used FingerCode[6] to measure similarity between any two fingerprints. In this approach a circular area from fingerprint is taken as region of interest centered at core point. This circular disk is divided in to several tracks and tracks are divided in to several sectors. The ROI is filtered with eight different Gabour filters. Standard deviation of gray values in all sectors for each of eight filtered images are computed. Final feature vector is consist of standard deviation in all sectors of each of the filtered image. Matching-score is calculated using Euclidean distance between the corresponding FingerCode.

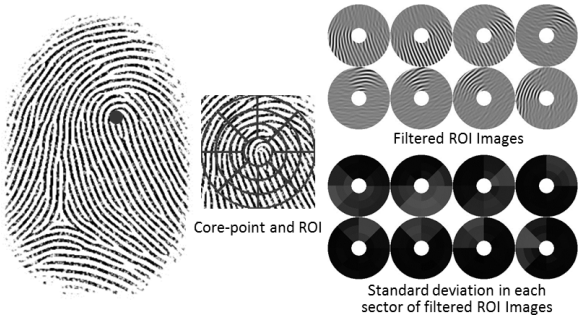


Figure 3.1: FingerCode

We are using 3 tracks, each having 8 sectors, totaling 24 sectors for each of the filtered image; it results in 192 dimensional feature vector i.e. FC192 (Figure 3.1).

For retrieval of target fingerprint from databases corresponding to any query fingerprint we have used LSH technique. For storing fingerprints in database, their FC192 features are preprocessed with hashing technique used in LSH and these hash values are stored along with every fingerprint in database.

Chapter 4

Near-Neighbors Search and Storage Strategy

4.1 Near-Neighbors Search Strategy

The *nearest neighbor problem* includes, given a point set P of n objects search the nearest point in P for given query point q. This problem generalizes to *K-near neighbors*, if k points in P which are nearest to q are required as result and generalizes to *R-near neighbors*, if all points within distance R from point q are required as result.

In approximation based near neighbors algorithm, a point is returned as result if its distance from the query is at most c times the distance from the query to its nearest neighbor, c is the *approximation factor*. It is known as *c-approximate near neighbors problem*. This can also be generalized to *c-approximate k-near neighbor problem* and *c-approximate R-near neighbor problem*.

An efficient approximation algorithm can be used to get exact near neighbors by listing all *approximate k-near neighbors* or *R-near neighbors* for appropriate k or R and selecting the nearest k neighbors or neighbors within radius R respectively [12]. The same idea is being used in E2LSH, an application based on Locality Sensitive Hashing.

The original locality sensitive hashing scheme basically solves *c-approximate R-near neighbors problem*. E2LSH uses basic LSH scheme to get all approximated near neighbors and then drops the near neighbors whose distances from query points is more than R, in order to solve randomize version of *R-near neighbors problem*. It is called $(R, 1-\delta)$-*near neighbors problem*. In this case each point p satisfying $||p-q|| \leq R$ has to be reported with a probability at least $(1-\delta)$. δ is the probability that a *R-near neighbor* is not reported.

4.1.1 Locality sensitive hashing

Locality-Sensitive Hashing[11] is main memory based algorithms for nearest neighbor search. The main idea is to hash the points using several hashing functions to ensure that for each hash function the probability of collision is much higher for points which are close to each other than for those points which are far away[12][13]. The neighbors of query points can be easily retrieved by determining the bucket containing query points.

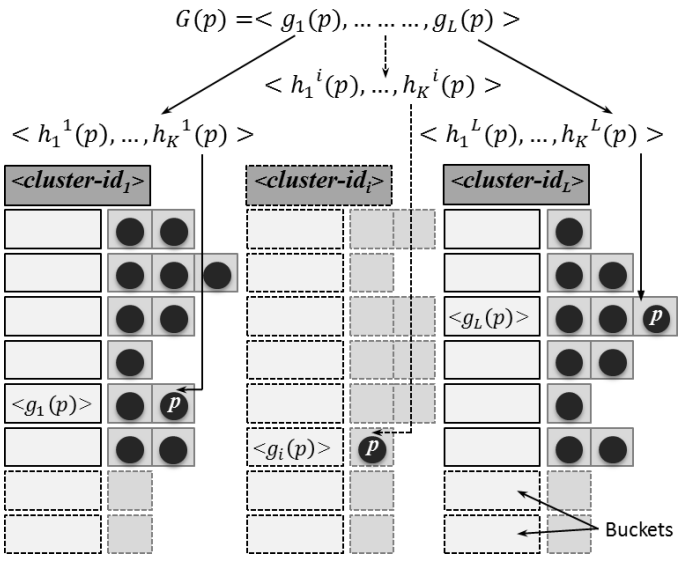


Figure 4.1: Clustering using $g_i, i \in \{1, 2, ..., L\}$ as cluster identifiers

The LSH relies on the locality sensitive hash functions[12]. A family $\mathcal{H} = \{h : S \rightarrow U\}$ is called locality sensitive if for any q, the function $p(t) = Pr_{\mathcal{H}}[h(q) = h(v) : ||q-v|| = t]$ is strictly decreasing in t. That is, probability of collision of points q and v is decreasing with distance between them.

The LSH scheme uses Cauchy or Gaussian distribution in order to define Locality sensitive hash function[13][14]. Each function $h(v) = \mathbb{R}^d \rightarrow \mathbb{Z}$ maps a $d - dimensional$ vector v in to integer. Every hash function in this family is defined by

$$h = \lfloor \frac{a.v + b}{w} \rfloor$$

. Where a is a *d-dimensional* vector with entries chosen independently from a Cauchy or Gaussian distribution and b is a real number chosen uniformly from the range $[0, w]$. It was suggested in [13] that $w = 4$ value provides good result.

Probability of collision for given hash functions, for points which are nearer in *d-dimensional* space is higher than the probability of collision among points which are far apart. To amplify this effect several such projections functions are used in E2LSH to increase the gap between probabilities of collision for nearer points and distantly located points.

A function g is defined[12] comprising of K different locality sensitive hash function as

$$g =< h_1, h_2, ...h_K >$$

$$g : \mathbb{R}^d \rightarrow \mathbb{Z}^k$$

All points for which vector $< h_1, h_2, ...h_K >$ is similar are in neighborhood in *d-dimensional* space with a high probability and they define a cluster which is identified by its *cluster-id* given by vector $< h_1(v), h_2(v), ...h_K(v) >$ for any point v in that cluster.

In order to ensure retrieval of all required near neighbors, L such functions are used.

$$G =< g_1, g_2, ...g_L >$$

For every $g_i, i \in \{1, 2, ..., L\}$, all points are clustered according to projection function $g_i =< h_1^i, h_2^i, ..h_K^i >$ separately (Figure 4.1).

Every data-point is associated with L different clusters. The near neighbor output is the union of all L clusters associated with query point.

4.2 Storage

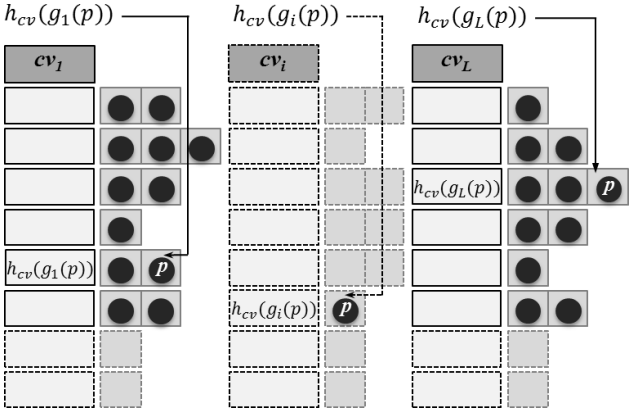
For each random projection function $g_i : \mathbb{R}^d \rightarrow \mathbb{Z}^K$, clusters are created and maintained separately. Clusters are identified by a *K-dimensional* vector, $cluster - id_i$ given by

$$< h_1^i, h_2^i, ..h_K^i >$$

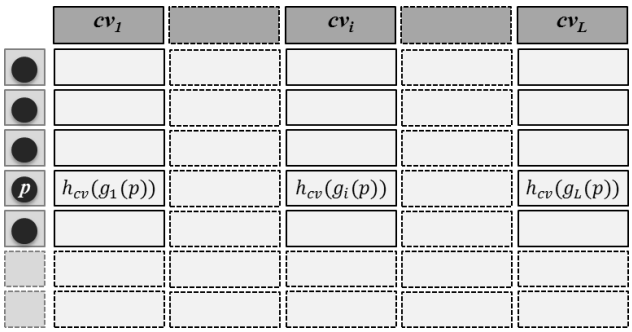
For the sake of simplicity and to speed up the retrieval of near neighbors, this *K-dimensional* id corresponding to each cluster is converted in to a single control-value cv using a separate hash function h_{cv}. For this such a hash function is used which ensures that two different *K-dimensional* ids corresponding to different clusters do not get hashed to same control value.

This way each cluster associated with projection function g_i is identified by single number cv_i in place of a *K-dimensional* vector, and every data point has $1D$ control-value cv_i in place of *K-dimensional* id $cluster - id_i$ associated with each g_i, $i \in \{1, 2, ..., L\}$ (Figure 4.2(a)).

In our storage scheme these L control-values $cv_1, cv_2, ..., cv_L$ corresponding to each point are stored in a table in database (Figure 4.2(b)) and separate B-Tree index is created on every control-value column.



(a) Clustering using $h_{cv}(g_i), i \in \{1, 2, ..., L\}$ as cluster identifiers



(b) Database storage scheme using $h_{cv}(g_i), i \in \{1, 2, ..., L\}$ as columns

Figure 4.2: Database storage scheme using LSH

Table schema:

```
uid(id : INTEGER, cv1 : NUMERIC,
                  cv2 : NUMERIC,
                  ............,
                  cvL : NUMERIC )
```

For any query point q SQL query will be

```
SELECT id FROM uid WHERE cv1 = h_cv(g_1(q)) OR
                         cv2 = h_cv(g_2(q)) OR
                         ..... OR
                         cvL = h_cv(g_L(q))  )
```

4.3 Selection of K and L

To use LSH scheme parameters K and L have to be specified. Given search radius R and success probability $(1 - \delta)$, E2LSH empirically calculates values of K and L.

Parameter L depends on the accuracy required i.e. $(1 - \delta)$. For higher accuracy larger value of L is required which means more no of buckets are searched in order to get near-neighbors. Higher value of K will take more time to compute random projection of a given point. A larger value of K also means data points are being embedded in higher dimension $(K < d)$ for each random projection function $g_i : \mathbb{R}^d \to \mathbb{Z}^K$, $i \in \{1, 2, ..., L\}$. It will result in larger number of clusters with lesser points in each cluster. To ensure given accuracy, larger value of L is required because of smaller size of clusters.

E2LSH is a main memory based solution to *R-near neighbor* problem. For each of $g_1, g_2, ...g_L$ all points are hashed separately. E2LSH maintains separate hash-data structure for each of g_i. It has to save L such data structures in memory. So apart from accuracy, main criteria to decide K and L are memory and time.

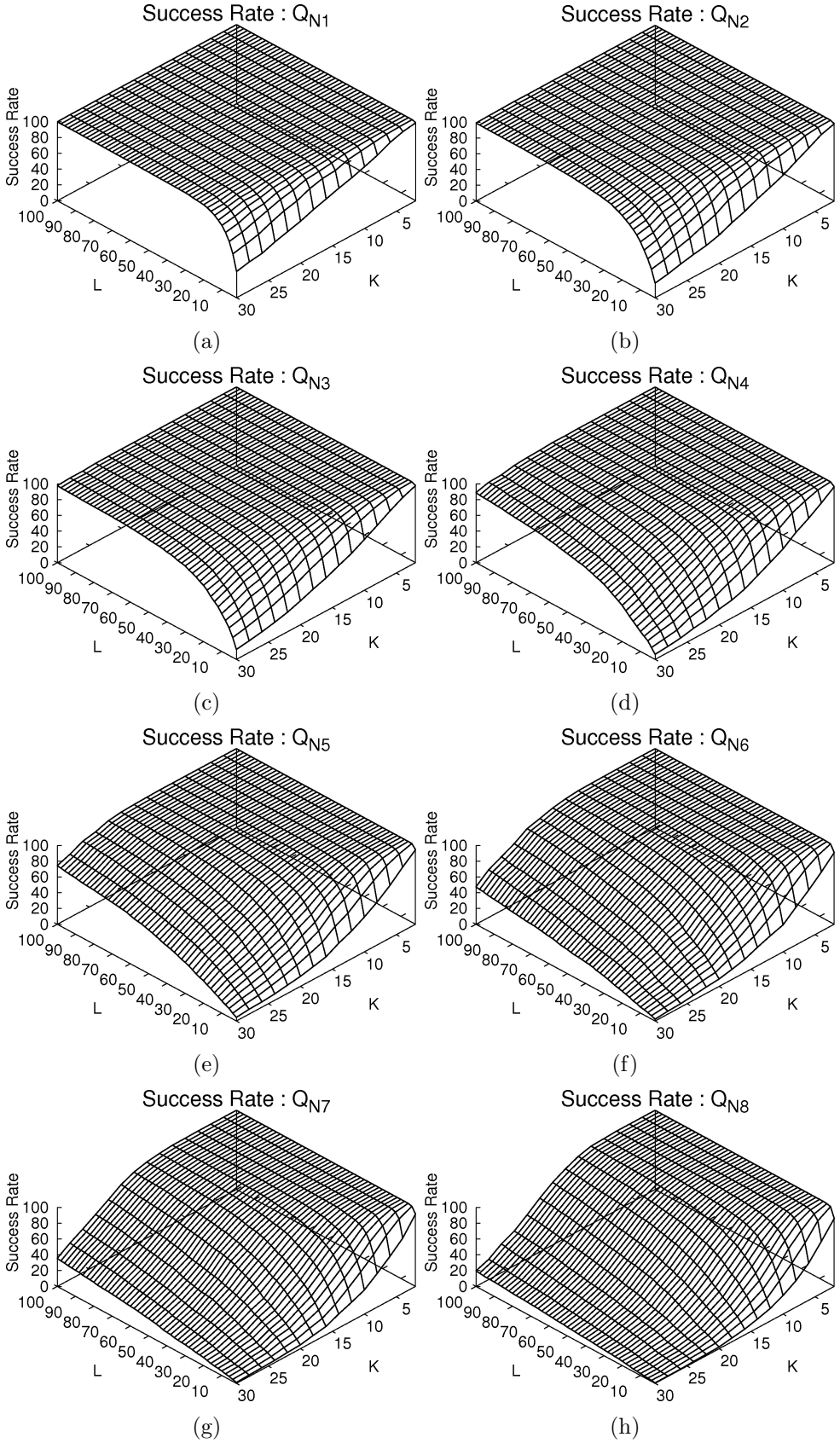


Figure 4.3: Success-rates for different query-sets

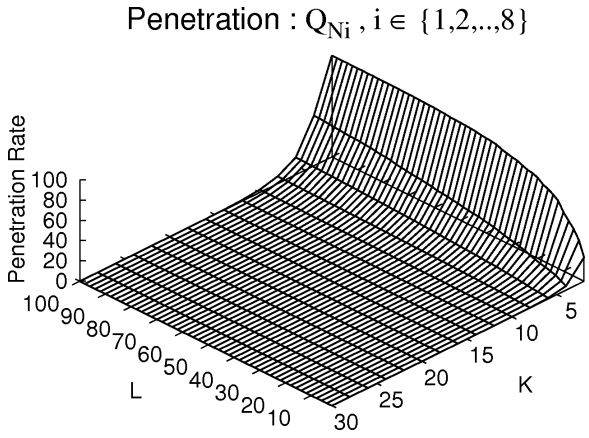


Figure 4.4: Penetration-rates for different values of K and L

NOISE LEVEL (Ni)	K_{Ni}
$N1$	20
$N2$	18
$N3$	16
$N4$	14
$N5$	10
$N6$	10
$N7$	8
$N8$	8

Figure 4.5: Values of K for different noise-levels

We are using projection scheme (hashing scheme) of E2LSH package, but with a different storage scheme. We are storing control-values associated with all data points in to database. So main memory isn't a restriction for us to decide values of K and L. In fact we can go for any large value of L because L is major factor in deciding accuracy. Since each of the control-value column can be searched independently, we can select an appropriate $L' < L$ at query-time. This is more important because for different level of noises we need different values of L' to ensure required accuracy.

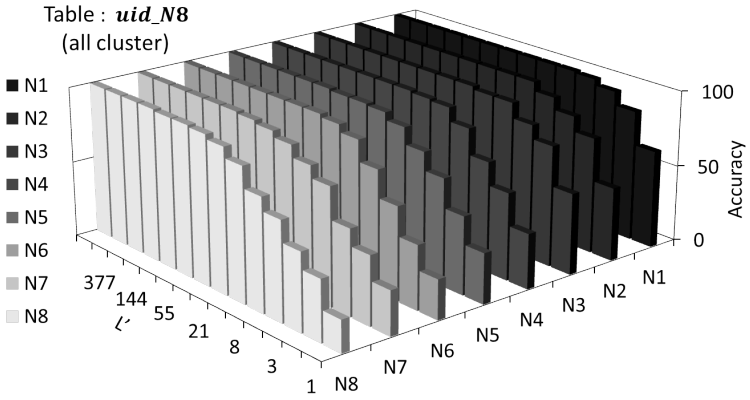
We are first calculating value of K empirically. Initially L is taken as a large value.

We looked for such a value of K for which we get maximum accuracy with minimum size of the near-neighbors result (penetration) for the given maximum acceptable level of noise.

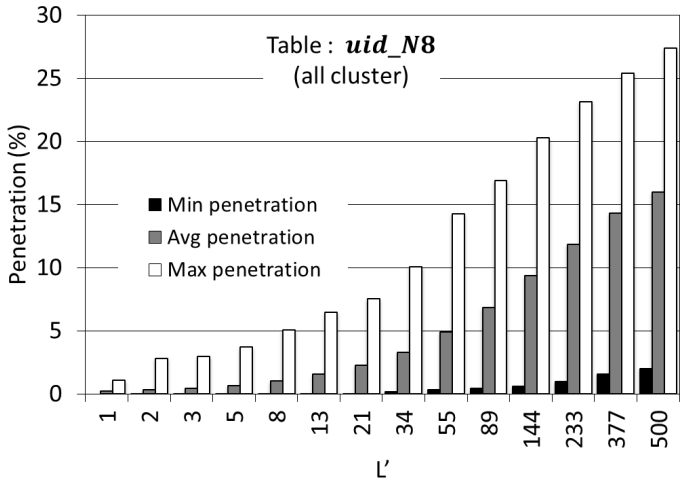
For this experiment we used a database of 10,000 synthetic fingerprints and 8 query-sets, $Q_{Ni},\ i \in \{1, 2, ..., 8\}$, each having 1000 genuine query-fingerprints of different pre-specified noise levels, $Ni,\ i \in \{1, 2, ..., 8\}$. For each query-set (Figure 4.3) we selected highest value of K in maximum success rate region as K_{Ni} to ensure least possible penetration because penetration rate reduces for higher value of K (Figure 4.4). For each query-set $Q_{Ni},\ i \in \{1, 2, ..., 8\}$, we decided value of K_{Ni}, which represent optimal value of K for a system where noise level Ni is considered to be maximum acceptable noise (Figure 4.5).

Next we have created a large database uid_N8 with one million fingerprints with $K = K_{N8}$ decided using query-set Q_{N8} considering N8 as maximum acceptable noise level. We fixed large value of L to be 500 due to the limit over the length of row in *postgres*. But this is not a restriction over L as we can store each column separately to store a really large L. We used same eight query-sets to calculate values of L' for different noise levels. Even though maximum acceptable noise level is fixed, we tried to fix L' for different noise level (noise levels less than $N8$) in order to save the efforts of scanning irrelevant points. To decide L' for a noise level we look for number of control-value columns which are sufficient to scan in order to give maximum accuracy for that particular noise level (Figure 4.6). Even this test can be run for higher noise level to see how much more noise the system can handle easily, given it was specified to accept a maximum noise of level $N8$.

Similar analysis were done for noise level N4 (Figure 4.7) and N2 (Figure 4.8) to determine L'.



(a)



(b)

$K = K_{N8}$

NOISE LEVEL	L'	Penetration (%)
N1	8	0.99
N2	8	0.99
N3	13	1.56
N4	21	2.23
N5	55	4.85
N6	55	4.85
N7	55	4.85
N8	144	9.32

(c)

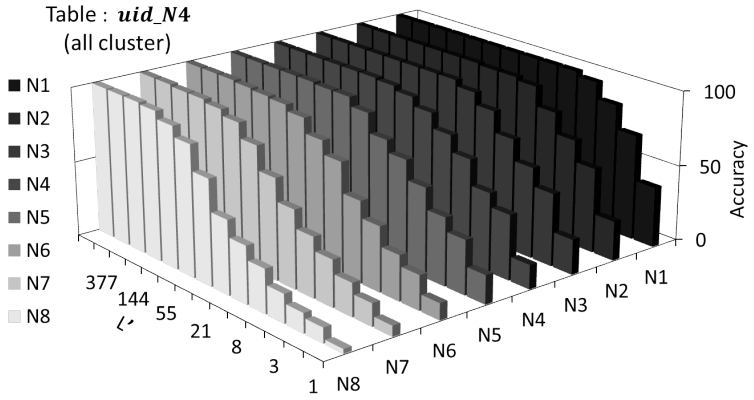
$K = K_{N8}$, $L = 144$

Average time in scanning all columns
57883 ms

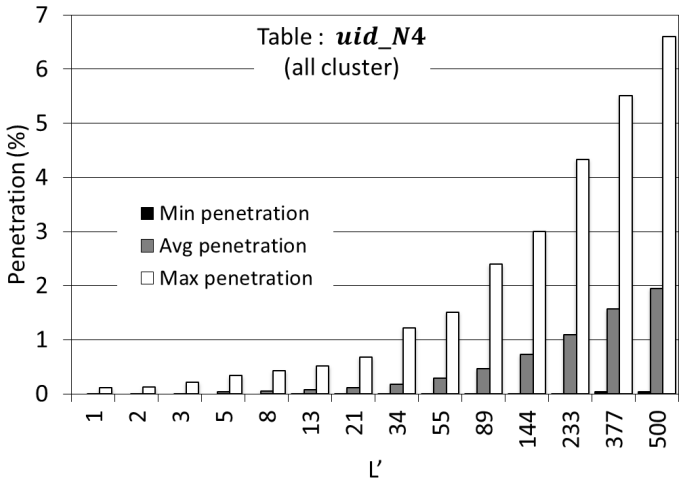
Average time taken in scanning a single column cv_i
517 ms

(d)

Figure 4.6: Success-rates, values of L' and penetration-rates for different noise levels and query-time for noise level $N8$ using K_{N8}



(a)



(b)

$K = K_{N4}$

NOISE LEVEL	L'	Penetration (%)
N1	8	0.0458
N2	21	0.1051
N3	21	0.1051
N4	55	0.2819
N5	89	0.4677
N6	233	1.0837
N7	377	1.5654
N8	377	1.5654

(c)

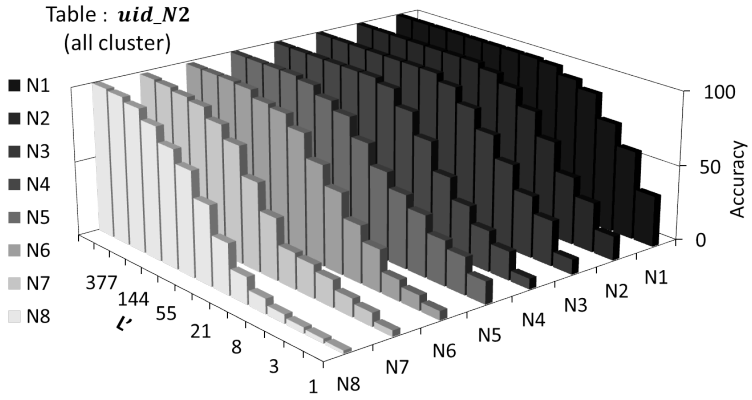
$K = K_{N4}, L = 55$

Average time in scanning all columns
8000 ms

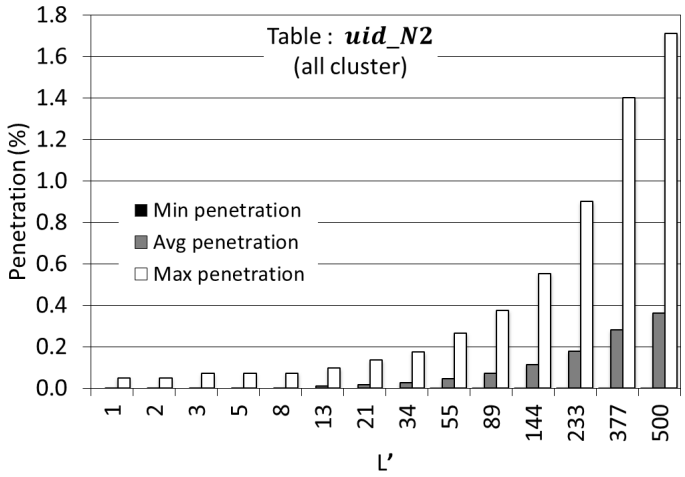
Average time taken in scanning a single column cv_i
207 ms

(d)

Figure 4.7: Success-rates, values of L' and penetration-rates for different noise levels and query-time for noise level $N4$ using K_{N4}



(a)



(b)

$K = K_{N2}$

NOISE LEVEL	L'	Penetration (%)
N1	21	0.0158
N2	34	0.0266
N3	34	0.0266
N4	55	0.0438
N5	144	0.1134
N6		
N7		
N8		

(c)

$K = K_{N2}$, $L = 34$

Average time in scanning all columns
1224 ms

Average time taken in scanning a single column cv_i
66 ms

(d)

Figure 4.8: Success-rates, values of L' and penetration-rates for different noise levels and query-time for noise level $N2$ using K_{N2}

4.4 Reducing Penetration Rate in LSH Scheme

Distribution of points across different buckets is not uniform. There are some very large clusters created by LSH scheme. Since actual target point is very near to query point compare to other irrelevant points, a large cluster will add more irrelevant points. To observe the importance of these bulky clusters we ran some queries to analyze size of different L clusters of a query.

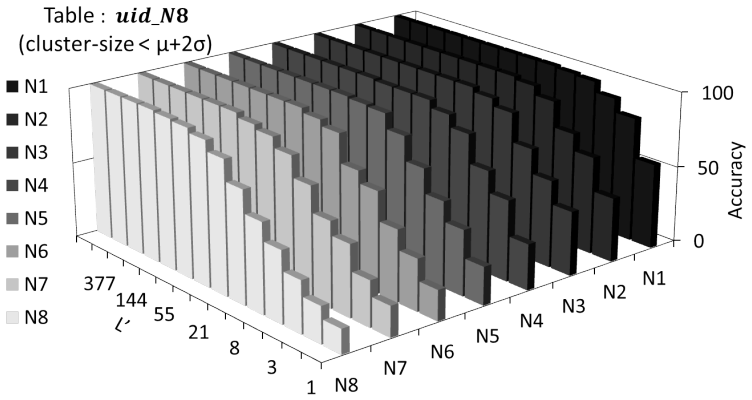
Bulky clusters are neither associated with any special category of fingerprints nor they are associated with any particular projection function i.e control-value column. Bulky cluster mark their presence equally for both kind of query fingerprint, those for which cluster-sizes are in general larger for different g_i, and those for which cluster sizes for different g_i is smaller. It can be concluded that these are not the bulky cluster only who play major role in bringing target fingerprint in to selection. So these clusters can be removed.

Probability of a cluster containing target point depends on size of the cluster, so removing bulky cluster means we are left with clusters with lower probability of point in it. It will increase the required number of columns L' to successfully search a query point.

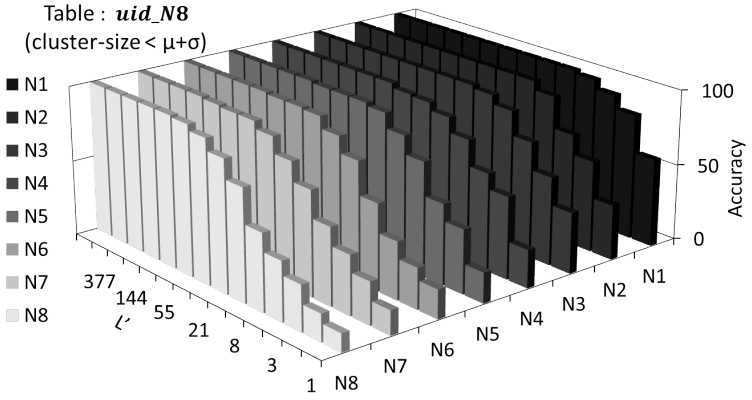
As we are using large value of L and practically used L' is lesser than L so larger cluster can be replaced by other cluster. This can be done by black-listing all bulky clusters and ignoring them at query time and using other columns in place of them as large choices for columns are available (L=500).

Removing bulky cluster will make penetration rate more predictable for a query fingerprint labeled with noise.

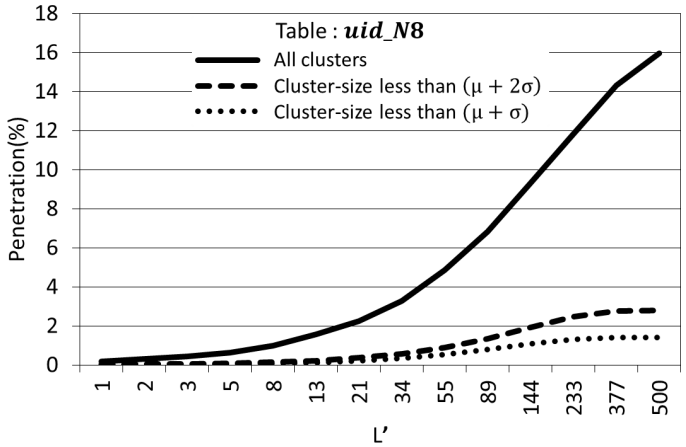
Removing these clusters also decreases penetration rate (Figure 4.9) as every cluster adds some irrelevant points and the size criteria restricts every cluster to add only limited number or irrelevant points . Reduced penetration rate will result in reduced query-time (Figure 4.10).



(a) Success-rates using cluster-size less than $\mu + 2\sigma$



(b) Success-rates using cluster-size less than $\mu + \sigma$



(c) Penetration-rates for different cluster-size constraints

Figure 4.9: Success-rates and penetration-rates for K_{N8}

$K = K_{N8}$

NOISE LEVEL	Cluster-size < μ+2σ		Cluster-size < μ+σ	
	L'	Penetration (%)	L'	Penetration (%)
N1	13	0.22	8	0.08
N2	13	0.22	8	0.08
N3	13	0.22	13	0.13
N4	34	0.57	34	0.34
N5	34	0.57	89	0.79
N6	34	0.57	89	0.79
N7	55	0.89	144	1.07
N8	144	1.93	144	1.07

(a) Values of L' and corresponding penetration-rates for different noise levels

$K = K_{N8}$, $L = 144$

Average time in scanning all columns

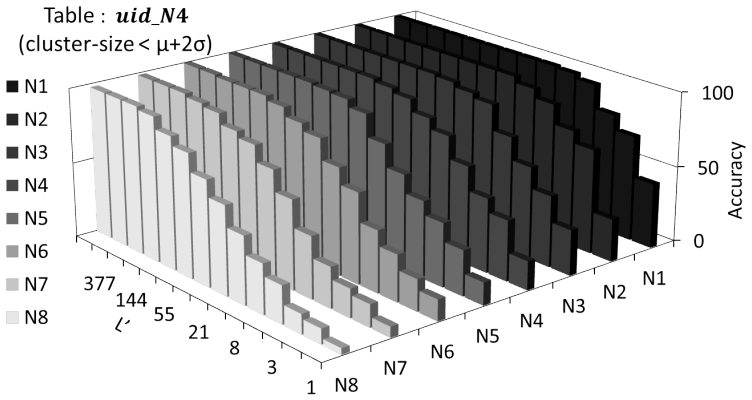
Cluster-size < μ+2σ	Cluster-size < μ+σ
21445 ms	10768 ms

Average time taken in scanning a single column cv_i

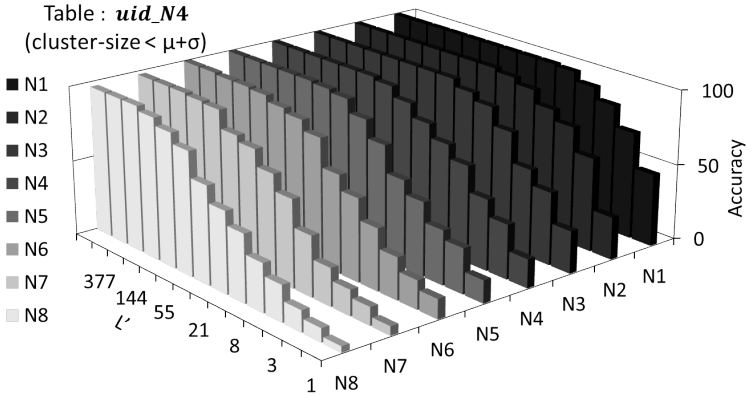
Cluster-size < μ+2σ	Cluster-size < μ+σ
389 ms	303 ms

(b) Query time for noise level $N8$

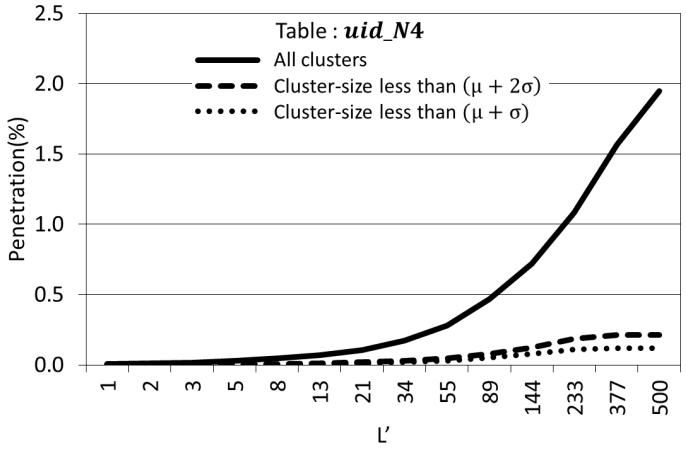
Figure 4.10: Values of L', penetration-rates and query-time for K_{N8}



(a) Success-rates using cluster-size less than $\mu + 2\sigma$



(b) Success-rates using cluster-size less than $\mu + \sigma$



(c) Penetration-rates for different cluster-size constraints

Figure 4.11: Success-rates and penetration-rates for K_{N4}

$K = K_{N4}$

NOISE LEVEL	Cluster-size < μ+2σ		Cluster-size < μ+σ	
	L'	Penetration (%)	L'	Penetration (%)
N1	13	0.0113	13	0.0076
N2	13	0.0113	21	0.0123
N3	34	0.0299	21	0.0123
N4	55	0.0479	55	0.0318
N5	89	0.0768	89	0.0508
N6	377	0.2128	377	0.1194
N7				
N8				

(a) Values of L' and corresponding penetration-rates for different noise levels

$K = K_{N4}$, $L = 55$

Average time in scanning all columns

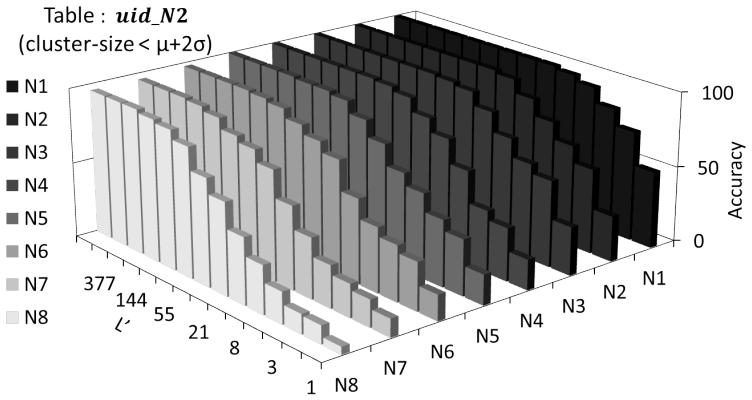
Cluster-size < μ+2σ	Cluster-size < μ+σ
1284 ms	1013 ms

Average time taken in scanning a single column cv_i

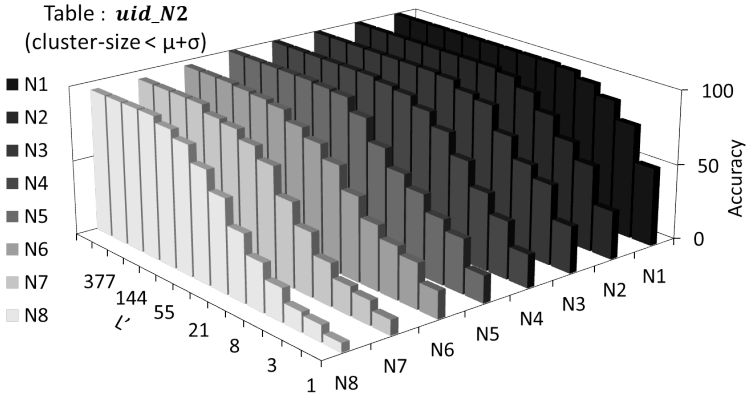
Cluster-size < μ+2σ	Cluster-size < μ+σ
64 ms	45 ms

(b) Query time for noise level $N4$

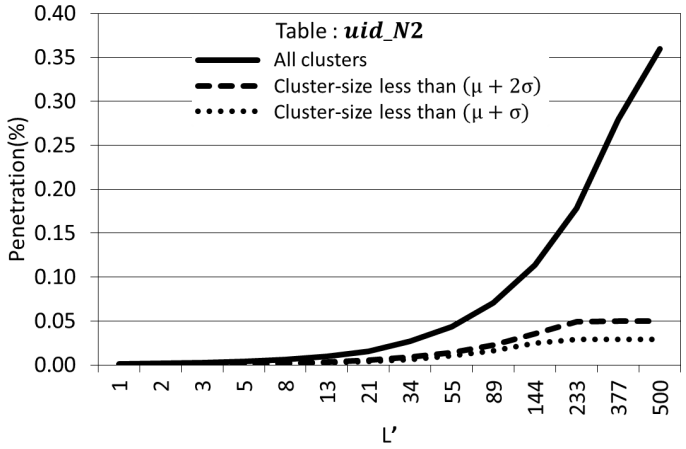
Figure 4.12: Values of L', penetration-rates and query-time for K_{N4}



(a) Success-rates using cluster-size less than $\mu + 2\sigma$



(b) Success-rates using cluster-size less than $\mu + \sigma$



(c) Penetration-rates for different cluster-size constraints

Figure 4.13: Success-rates and penetration-rates for K_{N2}

$K = K_{N2}$

NOISE LEVEL	Cluster-size < μ+2σ		Cluster-size < μ+σ	
	L'	Penetration (%)	L'	Penetration (%)
N1	13	0.0035	13	0.0025
N2	21	0.0056	21	0.0040
N3	34	0.0090	34	0.0063
N4	55	0.0143	55	0.0101
N5	144	0.0359	89	0.0161
N6				
N7				
N8				

(a) Values of L' and corresponding penetration-rates for different noise levels

$K = K_{N2}$, $L = 34$

Average time in scanning all columns

Cluster-size < μ+2σ	Cluster-size < μ+σ
460 ms	367 ms

Average time taken in scanning a single column cv_i

Cluster-size < μ+2σ	Cluster-size < μ+σ
27 ms	27 ms

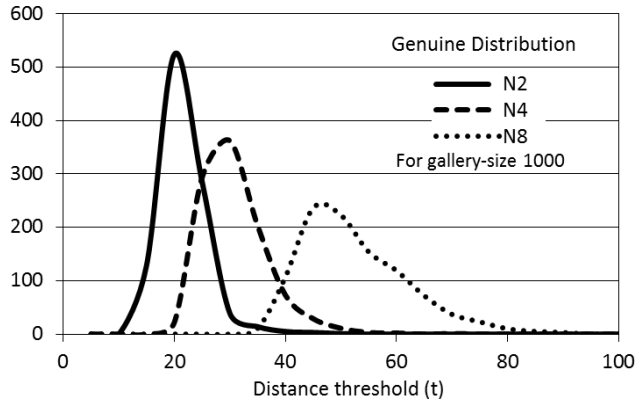
(b) Query time for noise level $N2$

Figure 4.14: Values of L', penetration-rates and query-time for K_{N2}

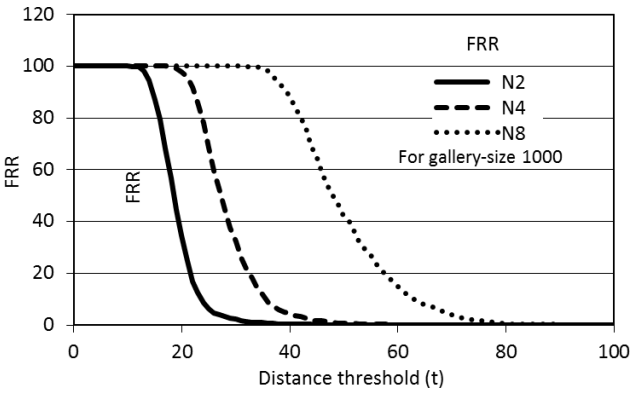
Same experiment was done with databases *uid_N4* and *uid_N2* using $K = K_{N4}$ and $K = K_{N2}$ respectively in order to look at the least possible penetration rate and query time. Although for K_{N4} and K_{N2} the cluster sizes are itself small, but removing bulky cluster can still decrease the penetration rates to great extent (Figures 4.11 and 4.13) and reduced query times (Figures 4.12 and 4.14).

4.5 Dropping Irrelevant Near-neighbors

LSH gives approximated near neighbors search result by projecting data-points and query point in to lower dimension space. This result includes many neighbors which were in fact far apart in original high dimensional space, but they were selected as near neighbors of query point on the basis of their projection in lower dimension space by LSH scheme. E2LSH package returns *R-near neighbors* from *approximated near neighbors* by removing all neighbors which are at a distance more than radius R.



(a) Distribution of genuine score for noise levels $N2$, $N4$, $N8$



(b) FRR curve for noise levels $N2$, $N4$, $N8$

Figure 4.15: Genuine distributions and FRR curves for different noise levels

In our near-neighbors search approach we have used $t_{zeroFRR}$ as search radius to drop irrelevant near-neighbors. For any sample query set of genuine fingerprints, $t_{zeroFRR}$ is distance threshold corresponding to $FRR = 0$. It ensures that no genuine fingerprint in

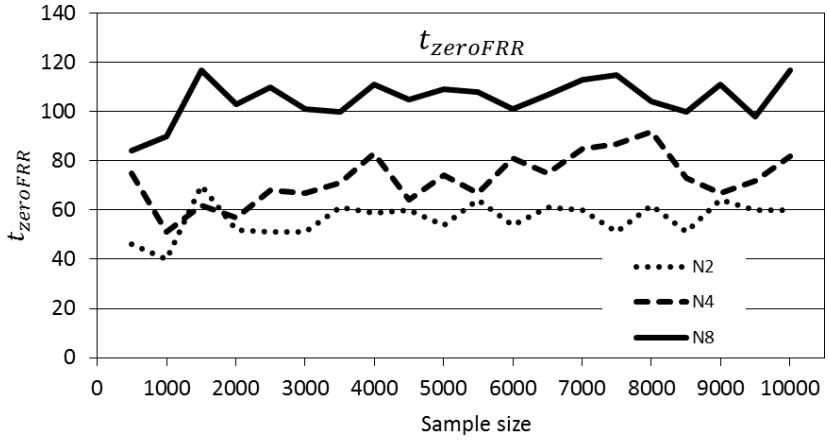


Figure 4.16: $t_{zeroFRR}$ for noise levels $N2$, $N4$ and $N8$

that sample will be rejected if we use $t_{zeroFRR}$ as search radius. Genuine distributions and FRR curves for different noise levels are shown in Figures 4.15(a) and 4.15(b) respectively. We observed that FRR curve changes with noise levels but for a fix noise level FRR curve does not depend on sample size. $t_{zeroFRR}$ found to be almost constant for different sample sizes.

We fixed $t^{N2}_{zeroFRR}$, $t^{N4}_{zeroFRR}$ and $t^{N8}_{zeroFRR}$ as search radius for noise levels $N2$,$N4$ and $N8$ respectively (Figure 4.16). We dropped all neighbors outside search radius specified. It reduced numbers of near-neighbors for brute force search using more accurate fingerprint matching algorithms such as minutiae-matching.

Figure 4.17 shows reduced near-neighbors and time taken in calculating distances of all approximated near neighbors with query point in d-dimension for databases uid_N8, uid_N4 and uid_N2.

$$K = K_{N8}\,,\, L = 144\,,\quad t^{N8}_{zeroFRR} = 125$$

Percentage of exact NN in approximated NN		
All clusters	Cluster-size < μ+2σ	Cluster-size < μ+σ
51.60 %	65.76 %	66.77 %

Average time in distance calculation		
All clusters	Cluster-size < μ+2σ	Cluster-size < μ+σ
67 ms	16 ms	10 ms

(a) For database *uid_N8*

$$K = K_{N4}\,,\, L = 55\,,\quad t^{N4}_{zeroFRR} = 100$$

Percentage of exact NN in approximated NN		
All clusters	Cluster-size < μ+2σ	Cluster-size < μ+σ
58.06 %	64.80 %	66.03 %

Average time in distance calculation		
All clusters	Cluster-size < μ+2σ	Cluster-size < μ+σ
3 ms	385 μs	255 μs

(b) For database *uid_N4*

$$K = K_{N2}\,,\, L = 34\,,\quad t^{N2}_{zeroFRR} = 80$$

Percentage of exact NN in approximated NN		
All clusters	Cluster-size < μ+2σ	Cluster-size < μ+σ
16.85 %	21.63 %	22.74 %

Average time in distance calculation		
All clusters	Cluster-size < μ+2σ	Cluster-size < μ+σ
308 μs	52 μs	33 μs

(c) For database *uid_N2*

Figure 4.17: Reduced near-neighbors and time taken in distance calculation

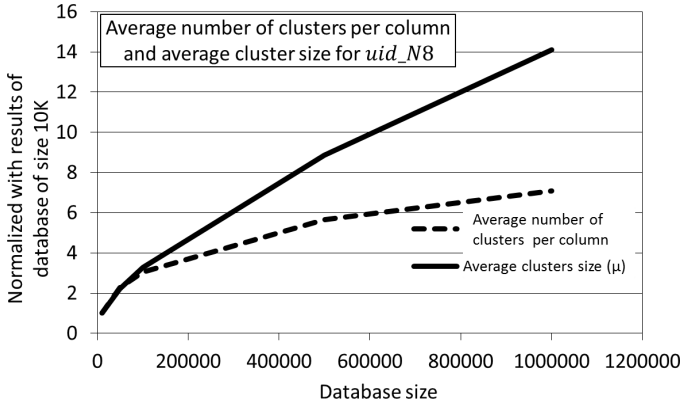
Chapter 5

Conclusion

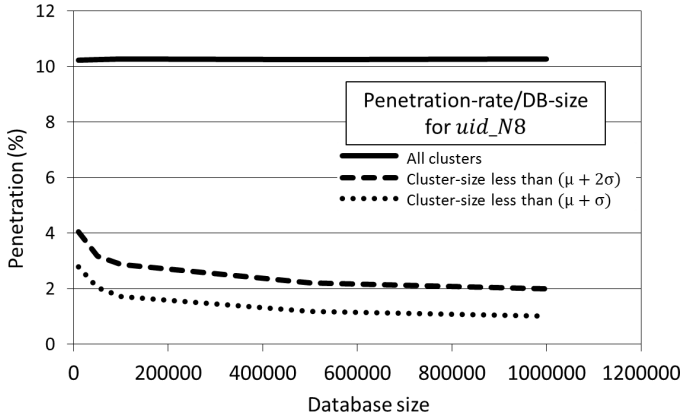
Although LSH is very efficient in searching *approximated near-neighbors* but there is no restriction over the size of result. We have reduced the penetration rates to a large extent by removing bulky cluster and providing large choices of clusters. It has reduced time taken in near-neighbors search.

We also experimented with several small databases to observe the behavior of LSH scheme with increase in size. It is observed that with increase in size of database, total number of clusters and average cluster-size both increase sub-linearly. Since penetration rate depends only on the sizes of individual clusters, approximated near-neighbor result size also increases sub-linearly with database size.

For scheme used for *N8*, cluster size increases faster than increase in number of average cluster (Figures 5.1(a)), while in lsh schemes used for *N4* and *N2*, cluster size increases slower than increase in number of average cluster (Figures 5.2(a)). So approximated near-neighbor result size increases faster in LSH scheme for *N8* than *N4* or *N2*. Which means penetration rates decreases faster for lsh scheme for *N4* and *N2* than in *N8* (Figures 5.1(b) and 5.2(b)).

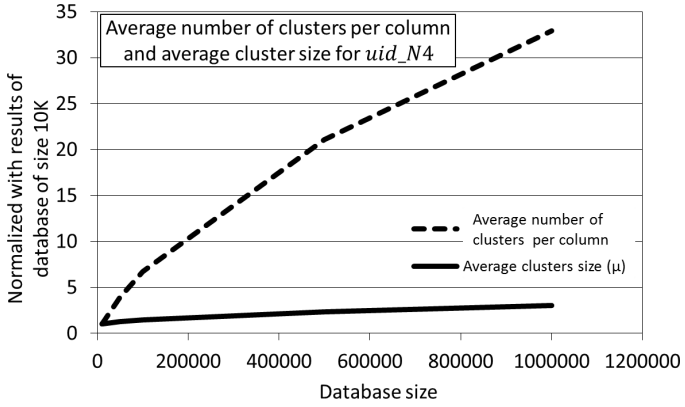


(a) Cluster-sizes for K_{N8}

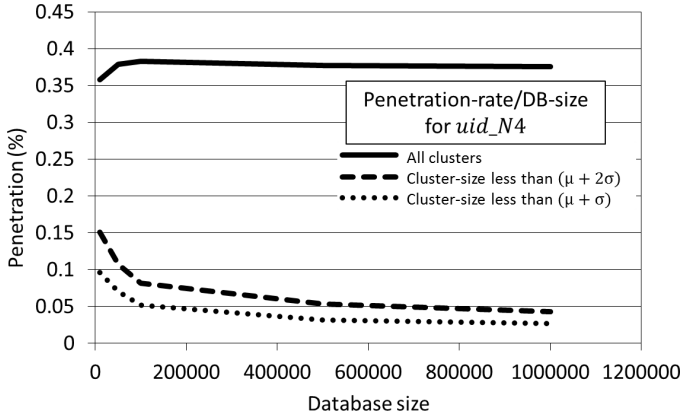


(b) Average number of clusters for K_{N8}

Figure 5.1: Growth in size and count of clusters for K_{N8}



(a) Cluster-sizes for K_{N4}



(b) Average number of clusters for K_{N4}

Figure 5.2: Growth in size and count of clusters for K_{N4}

Our fingerprint recognition based experiments were simulated on 1 million fingerprints. Using a single dedicated machine, approximated near-neighbors can be obtained in around *1 second* for noise level *N4* for database of size 1 million. So if we deploy 1000 such systems, we can perform a near-neighbor search for a database of size 1 billion within one second.

In our LSH based storage and querying strategy, query can by divided over several computing units. So search query can be divided both vertically and horizontally.

In vertical division of fingerprint query, each column cv_i can be searched individually.

In a single column there are duplicated values, fingerprints having duplicate values in any column are supposed to be in neighborhood. Each unique value in any column is independent of other unique values. Besides that, all the fingerprints in our database is properly labeled with Henry classes. So for horizontal division of fingerprint query, every column can be divided based on unique values in that columns and Henry classes.

Searching approximated near neighbors for a fingerprint with noise *N8* takes on average *10 seconds*. We ran vertically divided queries separately for each column. Searching for approximated near neighbors using a single column takes on average *300 milliseconds* for fingerprint of noise level *N8* (Figure 4.10). So even for higher noise levels approximated near-neighbors result can be given within *1 second* using more number of machines.

Bibliography

[1] R. Cappelli and D. Maio and D. Maltoni (2002). Synthetic Fingerprint-Database Generation, Proceedings of 16th International Conference on Pattern Recognition, vol 3, pages 744-747.

[2] Raffaele Cappelli (2004). SFinGe: an Approach to Synthetic Fingerprint Generation, Proceedings of International Workshop on Biometric Technologies, pages 147-154.

[3] D. Maltoni and D. Maio and A.K. Jain and S. Prabhakar (2003). Handbook of Fingerprint Recognition, Springer.

[4] B. Sherlock and D. Monroe (1993). A model for Interpreting Fingerprint Topology, Pattern Recognition, vol 26, pages 1047-1055.

[5] P. Vizcaya and L. Gerhardt (1996). A Nonlinear Orientation Model for Global Description of Fingerprints, Pattern Recognition, vol 29, pages 1221-1231.

[6] Anil K. Jain and Salil Prabhakar and Lin Hong and and Sharath Pankanti (1999). FingerCode: A Filterbank for Fingerprint Representation and Matching, IEEE Conf. on Computer Vision and Pattern Recognition, vol 2, pages 187-193.

[7] R. Cappelli and D. Maio and D. Maltoni (2004). An Improved Noise Model for the Generation of Synthetic Fingerprints, Control, Automation, Robotics and Vision Conference, Vol 2, pages 1250-1255.

[8] R. Cappelli and D. Maio and D. Maltoni (2001). Modelling Plastic Distortion in Fingerprint Images, Proceedings of 2nd International Conference on Advances in Pattern Recognition, vol 2013, pages 369-376.

[9] Biometrics Design Standards For UID Applications (2009). UIDAI Committee on Biometrics, Unique Identification Authority of India.

[10] R. Cappelli and A. Erol and D. Maltoni and D. Maio (2000). Synthetic Fingerprint-Image Generation, Proceedings of 16th International Conference on Pattern Recognition, vol 3, pages 475-478.

[11] Aristides Gionis, Piotr Indyk, Rajeev Motwani (1999). Similarity Search in High Dimensions via Hashing, Proceedings of 25th International Conference on Very Large Data Bases, vol 99, pages 518-529.

[12] Alexandr Andoni and Piotr Indyk (2008). Near-optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions, Commun. ACM, vol 51, pages 117-122.

[13] Mayur Datar, Piotr Indyk (2004). Locality-sensitive Hashing Scheme based on p-stable Distributions, Proceedings of 20th Annual Symposium on Computational Geometry, ACM Press, vol 4, pages 253-262.

[14] Piotr Indyk, Rajeev Motwani (1998). Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality, Proceedings of 30th Annual ACM Symposium on Theory of Computing, pages 604-613.