A simple matching algorithm for fingerprint minutiae datasets in accordance with DIN V 66400

Lisa Thalheim
thalheim@informatik.hu-berlin.de

Abstract: This paper describes a simple matching algorithm for two sets of fingerprint minutiae given in a format as described in [1], along with a description of the development process, pseudocode, and considerations on possible flaws of the algorithm.

1 Introduction

The goal of this work was to develop a matching algorithm for sets of fingerprint minutiae. The sets were given in a format as specified in [1] [1]. Such a dataset according to [1] contains - besides some header information - a serialized set of minutia descriptions, each consisting of the minutia's type, cartesian x- and y-coordinates and its angle.
The task was to find an algorithm which solves the following problem in reasonable time: Do two given minutiae sets stem from the same finger?
In this work, the problem was treated as a mere decision problem; the algorithm was only required to output either 'true' or 'false'. In this work, two minutiae sets are considered to stem from the same finger if there are at least 12 matching minutiae.[2]
The scenario for that algorithm to be used in is one of a biometric fingerprint verification system. A user would place her finger on a sensor; the system would then decide by a previously recorded sample whether or not the fingerprint matches the originally recorded one.
Additionally, the background was a system that combines chipcard and fingerprint verification using on-card-matching. This introduces the aspect that a single matching algorithm has to handle the characteristics and peculiarities of many sensors which are unknown during the development of the algorithm. Most fingerprint verification systems come as a bundle of sensor device and software, which enables the developers of that system to optimize their algorithm for that specific sensor device. This was not possible for the algorithm described here; it was known beforehand that the algorithm might be used with just any sensor which emits datasets according to [1].
The rest of this document is divided into four sections. Section 2 informally develops the algorithm, section 3 describes it in pseudocode. Section 4 analyzes some characteristics of the algorithm. Section 5 gives a summary and objections for possible future work.

[1] For information on how a minutia's type, x/y-coordinates and angle are defined here, see Section 5, "Minutia description"in [1].

[2] This number is controversial, however it is recommended as the valid minimum number of matching minutiae in [1] and will be used in this paper. Consider this under appropriate reserve.

2 Development of the algorithm

In this section, the development process of the algorithm will be described. Why is that? Because there is still a need for public fingerprint matching algorithms. I am sure there are excellent algorithms for that purpose - unfortunately, they are part of the most valuable property of biometric systems' manufacturers. Thus, the need for such algorithms lies in the need for non-proprietary, open biometric systems (which are easier to evaluate and hence potentially more trustworthy) and in the need for means to compare and evaluate proprietary biometric systems. This section is intended to give other developers ideas for where to start when trying to develop another, better fingerprint matching algorithm.
Besides, the information contained in this section is helpful for understanding how the algorithm works und why it works the way it does.
Inherent to the nature of the given problem is that such an algorithm would not necessarily need to be correct: Since no input to the sensor is exactly the same as any previous one (with the exception of inputs during a latent fingerprint attack or replay attack), we are confronted with fuzziness.
The first step in the development process was to analyze the data the algorithm was meant to operate on. Since there was no database of catalogized fingerprint-images accessible to the developers , the first step was to systematically generate 124 fingerprint-images using my own fingers. Three different sensors were used, one of which used a capacitive approach, the other two an optical one. The fingerimages were of different characteristics: "normal"(means: in a manner the manufacturers probably meant the user to put her finger on the sensor), rotated about 30 $^\circ$ versus "normal", rotated about -30 $^\circ$ versus "normal", using greater pressure, shifted downwards, shifted left. [3]
The next step was to mark the minutiae in the finger-images, which had to be done by hand, since there was no software handy for that at the time of development. It would have been too much work to manually process all of the 124 fingerimages, so in the first hand some images were selected which were considered characteristic.
In the end, there were 42 datasets of minutiae available in a suitable format.
The next step was to analyze the data visually: A program was written to visualize the sets of minutiae by displaying their x/y-position, type and angle graphically. The program allows to load up to 9 minutiae datasets and display them. Besides, it allows to manipulate one dataset at a time: shift it around pixelwise, rotate it, shrink and stretch it.
Having constructed this tool, it was possible to "play around"with the data the algorithm was targeted on. The basic idea was then to imitate the human approach to matching two minutiae sets. Note that matching two datasets given in the above (very abstract) form is usually also not an easy task for a human. Nevertheless, while experimenting with different datasets and the tool, one soon develops some kind of methodics to match two datasets. [4] Figure 1 shows the output of the program with two datasets loaded. Dataset 1 is depicted with white circles, dataset 2 with grey ones. The circle's tails depict the minutia's orientation.

[3] One of the optical sensors turned out to deliver images that were turned by 180 $^\circ$ compared to the other images. They could have "corrected"by hand, but they were used the way they were as a nice challenge.

[4] In this case, the human user was supported in developing this methodics by the original images, which where still available, so one could always have a look at it and "cheat"a bit.

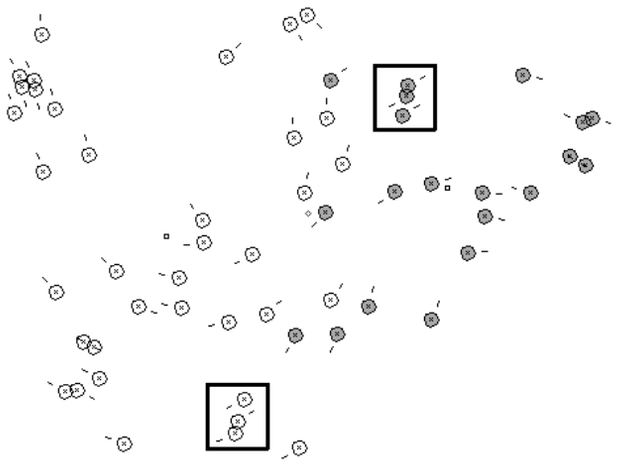

Figure 1: Two rather characteristic minutiae formations

Enclosed in the black boxes, you see two quite characteristic constellations of minutiae. They seem to be correspondences in the two datasets. The intuitive thing to do now: bring these ”twins”to match each other. In our example, this is achieved by turning one dataset by about 180 ° and some additional shifting and stretching. Figure 2 shows the result:
The black box contains the characteristic formations from Figure 2.
It turns out that there is a procedure which allows a human to systematically analyze two visually presented minutiae datasets:

- Find characteristic formations of minutiae which are present in both datasets.
- Bring these to match.
- See whether there are sufficient accordances visible between the datasets.
- If not: Continue if possible.

The above given pattern can be formalized and transformed into an algorithm.
Note that it will be frustrating for a human to check two datasets for accordances which do not stem from the same finger because her pattern matching abilites will not allow any clear answer as the datasets get larger than just a few minutiae. A human has to operate much like a computer here. So why not let the computer do it?

3 The algorithm itself

This section formalizes and implements the thoughts from the previous section, thus building the actual algorithm.

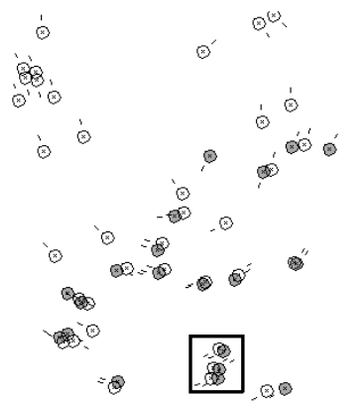

Figure 2: The two datasets, brought into accordance

3.1 Pseudocode and explanation

For the sake of clarity, the pseudocode is divided into the main routine and various helper routines.
The rough functionality should become clear from the main routine, MATCH-SETS; the helper functions are only presented here to clarify the juicy details.

```
function MATCH-SETS(source-minutiae-list, target-minutiae-list)
        returns success or failure
    input:
        source-minutiae-list, a list of minutiae
        target-minutiae-list, a list of minutiae
    source-pairs ← GENERATE-PAIRS(source-minutiae-list)
    target-pairs ← GENERATE-PAIRS(target-minutiae-list)
    SORT(source-pairs)      ; sort by distance ascending
    SORT(target-pairs)      ;
    next-source-pair:
    for each x ∈ source-pairs
        next-target-pair:
        if exists y ∈ target-pairs and SIMILAR-PAIRS(x,y)
            if tparams ← EXTRACT-TRANSFORMATION-PARAMS(x,y) succeeds
                DO-ROTATION-ON-SOURCE-DATA(tparams.rotation)
                DO-TRANSLATION-ON-SOURCE-DATA(tparams.translation)
                if EXIST-SUFFICIENT-MATCHES(source-minutiae-list, target-minutiae-list)
                    return(success)
                else
                    RESTORE-ORIGINAL-SOURCE-DATA()
                    goto next-target-pair
```

else
goto `next-target-pair`
else
goto `next-source-pair`
return failure

The following are the helper functions, GENERATE-PAIRS, SIMILAR-PAIRS, EXTRACT-TRANSFORMATION-PARAMS, DO-ROTATION-ON-SOURCE-DATA, DO-TRANSLATION-ON-SOURCE-DATA and EXIST-SUFFICIENT-MATCHES.

GENERATE-PAIRS takes a list of minutiae and creates a list of pairs. The result is a list which contains all possible pairs of minutiae from the list which have a distance less than the threshold MAX-PAIR-DISTANCE.

function GENERATE-PAIRS(*minutiae-list*) **returns** list of pairs
input:
minutiae-list, a list of minutiae
for i $\leftarrow$ 1 **to** size of minutiae-list
for j $\leftarrow$ i + 1 **to** size of minutiae-list
if EUCLIDIAN-DISTANCE(minutiae-list[i], minutiae-list[j]) $\leq$
MAX-PAIR-DISTANCE
add pair *(minutiae-list[i], minutiae-list[j])* to *pair-list*
return *pair-list*

SIMILAR-PAIRS takes two minutiae pairs and returns *true* if the pairs are *similar* (as determined by the parameter TOLERANCE-FOR-SIMILARITY) and *false* if not.

function SIMILAR-PAIRS(*minutiae-pair-1*, *minutiae-pair-2*) **returns** true or false
input:
minutae-pair-1, a minutiae pair
minutae-pair-2, a minutiae pair
if EUCLIDIAN-DISTANCE(*minutiae-pair-1*) $\in$
] EUCLIDIAN-DISTANCE(*minutiae-pair-2*) $-$
TOLERANCE-FOR-SIMILARITY,
EUCLIDIAN-DISTANCE(*minutiae-pair-2*) $+$
TOLERANCE-FOR-SIMILARITY [
return true
else
return false

EXTRACT-TRANSFORMATION-PARAMS extracts the parameters necessary for the translation and rotation of the source set from the two given pairs. It returns a failure if this was not possible.

```
function EXTRACT-TRANSFORMATION-PARAMS(minutiae-pair-1, minutiae-pair-2)
        returns params or failure
    input:
        minutiae-pair-1, a minutiae pair
        minutiae-pair-2, a minutiae pair
    angle-diff-1 ← minutiae-pair-1.minutia-a.angle −
    minutiae-pair-2.minutia-a.angle
    angle-diff-2 ← minutiae-pair-1.minutia-b.angle −
    minutiae-pair-2.minutia-b.angle
    if SIMILAR-ANGLES(angle-diff-1, angle-diff-2)
        params.rotation ← ((NORMALIZED(angle-diff-1) +
            NORMALIZED(angle-diff-2) / 2)
    else
        angle-diff-1 ← minutiae-pair-1.minutia-b.angle −
        minutiae-pair-2.minutia-a.angle
        angle-diff-2 ← minutiae-pair-1.minutia-a.angle −
        minutiae-pair-2.minutia-b.angle
        if SIMILAR-ANGLES(angle-diff-1, angle-diff-2)
            params.rotation ← ((NORMALIZED(angle-diff-1) +
                NORMALIZED(angle-diff-2)) / 2
        else
            return failure
    x-diff-1 ← minutiae-pair-1.minutia-a.x − minutiae-pair-2.minutia-a.x
    x-diff-2 ← minutiae-pair-1.minutia-b.x − minutiae-pair-2.minutia-b.x
    y-diff-1 ← minutiae-pair-1.minutia-a.x − minutiae-pair-2.minutia-a.x
    y-diff-2 ← minutiae-pair-1.minutia-b.x − minutiae-pair-2.minutia-b.x
    if SIMILAR-DIFFS(x-diff-1, x-diff-2) and SIMILAR-DIFFS(y-diff-1, y-diff-2)
        params.translation.x ← (x-diff-1 + x-diff-2) / 2
        params.translation.y ← (y-diff-1 + y-diff-2) / 2
    else
        x-diff-1 ← minutiae-pair-1.minutia-b.x − minutiae-pair-2.minutia-a.x
        x-diff-2 ← minutiae-pair-1.minutia-a.x − minutiae-pair-2.minutia-b.x
        y-diff-1 ← minutiae-pair-1.minutia-b.x − minutiae-pair-2.minutia-a.x
        y-diff-2 ← minutiae-pair-1.minutia-a.x − minutiae-pair-2.minutia-b.x
        if SIMILAR-DIFFS(x-diff-1, x-diff-2) and SIMILAR-DIFFS(y-diff-1, y-diff-2)
            params.translation.x ← (x-diff-1 + x-diff-2) / 2
            params.translation.y ← (y-diff-1 + y-diff-2) / 2
        else
            return failure
    return params
```

DO-ROTATION-ON-SOURCE-DATA rotates the source minutiae by the degrees given in *params*.

function DO-ROTATION-ON-SOURCE-DATA(*params*) **returns** nothing
  input:
    params, datastructure containing parameters for rotation
  for each x $\in$ source-minutiae-list
    ROTATE(*x*, *params*)
  return

DO-TRANSLATION-ON-SOURCE-DATA translates the source minutiae by (x,y)-values as given in *params*.

function DO-TRANSLATION-ON-SOURCE-DATA(*params*) **returns** nothing
  input:
    params, datastructure containing parameters for translation
  for each x $\in$ source-minutiae-list
    TRANSLATE(*x*, *params*)
  return

EXIST-SUFFICIENT-MATCHES determines whether there are enough matches between the target minutiae and the modified source minutiae.

function EXIST-SUFFICIENT-MATCHES(*source-minutiae-list*, *target-minutiae-list*)
    returns true or false
  input:
    source-minutiae-list, a minutiae list
    target-minutiae-list, a minutiae list
  matched-minutiae $\leftarrow$ 0
  for each *x* $\in$ *source-minutiae-list*
    for each *y* $\in$ *target-minutiae-list*
      if MATCHES(*x*,*y*))
        matched-minutiae $\leftarrow$ *matched-minutiae* + 1
        if *matched-minutiae* $\geq$ 12
          return true
  return false

MATCHES takes two minutiae and returns *true* if they *match* (as determined by diverse parameters to the algorithms) and *false* if they don't.

function MATCHES(*minutia-1*, *mintuia-2*) **returns** true or false
input:
minutia-a, a minutia
minutia-b, a minutia
if ABS(*minutia-a.x* − *minutia-b.x*) ≤ *X-TOLERANCE* and
if ABS(*minutia-a.y* − *minutia-b.y*) ≤ *Y-TOLERANCE* and
if ABS(NORMALIZE(*minutia-a.angle*) − NORMALIZE(*minutia-b.angle*)) ≤
ANGLE-TOLERANCE and
if COMPATIBLE-TYPES(minutia-a, minutia-b)
return true
else
return false

3.2 Parameters

All names in the pseudocode denoted in uppercase italic font are parameters. These parameters influence greatly the behaviour and performance of the algorithm.
One might assign the following values for acceptable results:

MAX-PAIR-DISTANCE := 1.5 (mm)
TOLERANCE-FOR-SIMILARITY := 0.3 (mm)
X-TOLERANCE := 0.7 (mm)
Y-TOLERANCE := 0.7 (mm)
ANGLE-TOLERANCE := 10 (degrees)

4 Characteristics of the algorithm

The straight-forward approach to matching two sets of minutiae occurs in this algorithm in the sub-function FIND-MATCHES. So actually, this algorithm is just a refinement of this straight-forward method: compare each minutia from one set to each minutia from the other set. The main advantage is that the algorithm is very robust, compared to the ordinary method: It is quite tolerant towards any rotation or shifting of the two datasets against each other, as long as they contain enough corresponding minutiae. This meets the requirements of being interoperable with any sensor device that might come along.

4.1 Complexity

It is obvious that the algorithm's has no optimal time complexity[5]. Anyway, this will usually not have a great effect, since the size of minutiae datasets has a recommended upper bound of 60, and will probably be even smaller in practice.
Nevertheless, if implemented in a real production environment, additional countermeasures would need to be taken in order to eliminate the possibility that the algorithm receives very large datasets, be it accidently or as part of some attack.
The meaning of "very large"depends heavily on the structure of the data and the computing resources, which are usually very limited on a smartcard.

4.2 Optimization

One can take some simple measures to enhance the algorithms performance. This first one is to make sure the smaller one of the two datasets is passed to the algorithm as the source minutiae list, since the transformations are only applied to the source data.
Furthermore, it would be worth initially sorting the target minutiae list by x, y and angle ascending and modify the function FIND-MATCHES to take advantage of this. This would have the effect that the algorithm most often would not have to touch all elements of the target minutiae list when trying to find a match for a particular minutia from the source minutiae list.

4.3 Flaws and potential problems

This algorithm is not too sophisticated. Hence there are some potential problems. The possible problem of performance has already been addressed, but there are still some flaws concerning the reliability and the fraud resistance.
The algorithm relies on the presence of characteristic minutiae constellations in both datasets. Whether two minutiae form a characteristic constellation depends mainly on their distance. A threshold is defined for the maximum distance two minutiae can have to be considered for a characteristic constellation. Any minutiae pairs with a distance greater than this threshold are not considered. Since one wants to keep the threshold low for performance reasons, it is not unlikely that there will be datasets in which the algorithm cannot recognize any characteristic constellations at all and this reject the datasets as "not matching", even if only a simple matching test as executed in FIND-MATCHES would have given a positive result.
The next issue is one of fraud resistance. The algorithm has no counter-measures whatsoever to prevent replay-attacks. If one leaves aside countermeasures which should be taken externally anyway, this still leaves the fact, that the algorithm cannot even recognize if it

[5] A rough estimation suggests a complexity not much better than $O(n^6 * m^2)$, where n is the number of minutiae in the source set and m the number of minutiae in the target set

is given the same dataset two times, as source minutiae list and as target minutiae list.
The last issue to be addressed here can be called 'minutiae flood'. The basic idea is to artificially construct minutiae datasets with a very large number of minutiae. In the extreme case, this would mean one minutia for each combination of x/y-coordinate, type and angle. The algorithm as it is described above would happily match such a dataset to its reference dataset [6]. Since in this extreme case there would be multiple minutiae at one x/y-coordinate, which is highly unlikely, this would be relatively easy to detect. Alas, it is not done up to now. Besides that, the issue remains and necessitates some artificial upper bound for the number of minutiae a dataset may contain, because one could still generate a dataset with not more than one minutia per x/y-coordinate and a ridiculously large number of minutiae, some of which will probably match.

5 Conclusion

This work described a simple matching algorithm for sets of fingerprint minutiae. The purpose of this algorithm was a rather academic one: Provide a simple algorithm for comparison with and black-box analysis of other, typically commercial, undisclosed fingerprint matching algorithms. Another objective was to take one more step towards fully disclosed, public algorithms.
However, there are still some issues left for possible future work: It might be desirable to have the algorithm output a degree of likeliness that the two given minutiae sets stem from the same finger instead of just a plain positive/negative output, although the utility of this functionality probably depends on the application. A smartcard-scenario as described in the introduction could happily work with just a true/false-matching scheme.
Another point to be clarified is the number of required matching minutiae. Although in the German law enforcement it is still common to require only 12 minutiae to match, this practice is controversial, especially in the field of electronic biometrics.

6 Annotations

A reference implementation in C is available.
I would like to thank the ES22 of T-Systems Nova GmbH for supporting this work, furthermore Jan Krissler, Frank Rieger and Susanne Schmidt for review and help with this paper.

References

[1] DIN, Finger Minutiae Encoding Format and Parameters for On-Card Matching, Standard proposition 66400, June, 2002

[6] ...although this would take quite a while.