Signature authentication using graph edge labeling

Karl-Sverre Knutsen

2021

Abstract

Handwritten signatures are still widely used for authentication in today's society. Compared with other biometric features like fingerprint and face recognition, handwritten signatures are easier to falsify. It is especially difficult to defend against scenarios where criminals know the victims signature shapes (called skilled forgeries). Therefore, we need secure system implementations that is capable of distinguishing between forgeries and legitimate users.

This project aims to investigate a new signature verification method where the shape is converted into a labeled graph and we attempt to utilize information about the edges. There are defined two types of nodes in the graph; endpoints and crossing lines. The edges are labeled with different distance estimations related to the connected nodes. Three edge labeling methods are tested; pixel distance, euclidean distance and efficiency. Efficiency is a ratio between the other two methods. In order to test this system implementation, we use the online dataset MCYT-100 and focus on the signature shape only. This study is a first contribution for the utilization of labeling graph edges for signature authentication, and may create foundations for further investigations in this area of signature analyses.

Sammendrag

Håndskrevne signaturer er fortsatt utbredt i dagens samfunn for å gjøre autentisering. Sammenlignet med andre biometriske kjennetegn som fingeravtrykk og ansiktsgjenkjenning, er signaturer lettere å forfalske. Det er spesielt vanskelig å forsvare seg mot hendelser hvor kriminelle kjenner til hvordan håndskriften ser ut på forhånd (skilled forgeries). På grunn av dette trenger vi sikre systemimplementasjoner som klarer å skille mellom forfalskede signaturer og legitime brukere.

Dette prosjektet undersøker en ny metode for å verifisere signaturer, der selve formen blir omgjort til en graf og vi forsøker å utnytte informasjonen om kantene. Det blir definert to typer noder i grafen; endepunkter og kryssede linjer. Kantene er merket med forskjellige distanseutregninger basert på form og avstand mellom tilhørende noder. Det er gjennomført tre typer utregninger; piksel-distanse, euclidean-distanse og effektivitet. Effektivitet er forholdet mellom de andre to metodene. Vi benytter et online-datasett kalt MCYT-100 for å teste implementeringen av metoden, men utnytter bare informasjonen om selve signaturformen. Denne oppgaven er et første bidrag for å se på utnyttelse av kantene til grafen for autentisering av signaturer, og kan legge grunnlaget for videre forskning innenfor feltet.

Preface

This thesis is the final part of a Master's degree within Information security at the Norwegian University of Science and Technology (NTNU). The school attendance has been completed as a part-time study spread over three and a half years from 2018 to 2021. The last year and a half has been aimed towards this thesis, which has given me great learning benefits in an exciting and growing field.

I would like to thank my supervisor, Patrick Bours at NTNU Department of Information Security and Communication Technology. His continuous support, guidance and ideas throughout the whole period, from the early stage of research project planning until the finished Master's thesis delivery, has been very valuable and motivating.

My education at NTNU is now finished. Nevertheless, I'm motivated to continue expanding my knowledge within the field of information security in order to keep up with growing cyber threats.

Lillehammer, Monday 6th December 2021
Karl-Sverre Knutsen

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Topic covered by the project

Handwritten signatures have been a natural method of authentication for centuries. This method is still widely used on systems, with different technical approaches of implementation.

There exist two types of signature verification approaches, *Online* and *Offline*. *Offline* signature verification considers the shape of the signature only, while *Online* also considers information from the signing process. Examples of additional signature data from online capturing are signing speed and pen pressure. This project will use *Offline* signature information, and look closer into characteristics related to the length of drawn lines. We are going to extract the shapes into graphs, and label all lines in between each junction points and connected endpoints. The equality between signatures will be defined by the line distance differences, referred to as the edge label.

Even though graphs give high representational power, graph based approaches are not widely used for signature authentication systems. The reason for this might be the high mathematical complexity for system implementation. There are some studies using *Graph Edit Distance (GED)* for this purpose. GED finds the lowest possible distance for converting one graph into another. The approach presented in this thesis is slightly different, but due to the lack of specifically related studies we will introduce GED in order to explain important concepts of graph based approaches. Previous studies focus on node labeling, while we are going to investigate the performance of edge labeling.

1.2 Keywords

Authentication, Signature biometrics, Backtracking, Graph edge labeling

1.3 Problem description

Many companies are using handwritten signatures as proof of identity for contract signing and other costumer-oriented services. Within the area of authentication, there will always exist criminals willing to use a person's identity for own personal interest. A signature may seem easier to falsify than for instance a fingerprint. There are mainly three different signature forgeries; random, simple and skilled. Random forgeries don't even bother writing the victim's actual name, while simple forgeries uses the name of the victim but is unaware of what the original signature looks like. Skilled forgeries attempts to imitate the original signature in order to make it look as close as possible to the original. Due to the increasing incidents of forgeries, there is a definite need of strict signature authentication systems that filters out criminals. If there cannot be developed improvements, handwritten signatures may become disapproved as a method of verification in the future.

We know from previous studies [1] that the use of GED for signature authentication increases the representational power and system accuracy. By extending this work, we may invent even better systems for protecting peoples identity. Therefore, we will introduce other attributes from the signature and try a new method for comparing graph equality. The implementation compares edge distances between two signatures, and afterwards creates a matrix based on all possible edge transformation costs. A backtracking algorithm will find the lowest possible combination for converting all edges from one signature graph into another. Node neighbor equality between signatures will benefit the cost during calculation. We will then see if this implementation increases or decreases the system performance compared to existing techniques.

1.4 Justification, motivation and benefits

There will always be a race in information security between criminals and security developers. Every system has weaknesses, and new methods must be developed in order to deal with constant threats. Since signature is such a common way of doing authentication, it is important to develop better techniques for today's systems.

This project will test new data from the signature in order to see if the technique gives better results than the already existing systems in literature.

1.5 Research questions

- Can a graph based approach with labeled edges be used for signature authentication?

Sub-questions:

1. What distance-related edge labels could be used?
   Short explanation: One could use the number of pixels on the signature curve between two nodes as an edge label, or the Euclidean distance between the two nodes. Alternatively one could have the edge label "efficiency" of the edge. This could be the ratio between the direct distance and the number of pixels on the signature curve.
2. How should the specific characteristics of a signature-based graph influence the implementation of the graph comparison algorithm?
   Short explanation: The signature graphs have specific characteristics, e.g. nodes have only degree 1 (endpoints), 2 (change of direction) or 4 (crossing of lines).

1.6 Planned contributions

This thesis will attempt to create a more secure method for signature authentication. The planned contribution is an experiment with the signature verification technique and a program that is capable of comparing signature patterns with this method.

Chapter 2

Related work

This chapter introduces different topics that are relevant for solving the research question. We will at first take a look at fundamental biometric theory, give a general introduction into graphs, explain backtracking and illustrate the concept of Graph Edit Distance (GED). Afterwards, this chapter will focus on State of the Art theory where signature authentication, general theory about GED and the combination of these two approaches are in focus.

2.1 Background

This chapter introduces the fundamental knowledge in order to understand the research question. We will first talk about biometrics in general and how a biometric system is constructed. The chapter will also include general graph theory, the concepts of backtracking and an introduction into GED as a concept.

2.1.1 Authentication

"*Authentication is the process of validating the identity of someone or something*" [2] (also refereed to as verification). This area have massively increased focus for the last decades, and people rely on authentication during many daily activities. Access into phones, computers, physical areas and payments with bank cards are some examples of daily authentication processes. All these examples may differ in their authentication method, but the concept of approving that you are the person you claim to be is still fundamental. We will now look closer into three different classes of authentication [3].

Something you know

This is something you (and you only) know of. It is typically a secret that is not to be shared with others. A very common example is passwords. The benefit of using passwords is that it is administratively inexpensive, and you don't need any physical equipment that could be lost or broken. A strong password demands a certain complexity of characteristics and length. Nevertheless, people tend not to use strong passwords for different reasons (like fear of forgetting, laziness etc.). This makes it easier for an attacker to guess the password, which will also give the attacker access. Common passwords are often including date of births, pet names and other dictionary words that relate to that person. If the person creates a really strong password, they tend to use the same password on many different accounts and even write the password down.

Something you have

This class uses physical equipment for authentication. Examples are ID-cards, keys, SIM cards etc. Unlike passwords, this form of authentication is slightly safer because it is harder to steal physical items than guessing passwords. People are also less likely to lose an item than to forget a password. The downside is the administration required and expenses for the production of new, lost or broken physical authentication devices. When someone loses their authenticator, the item must also be deactivated immediately in order to maintain security.

Something you are

This is a naturally acquired physical or behavioural characteristic which is unique for each individual. During the last decades, many IT companies have implemented this class into daily authentication use cases. Smartphones may be the leading area, where both fingerprint and face recognition are widely used for both access into main systems, but also implemented for third party applications like bank authentication.

Combining classes

In order to make authentication systems even more secure, it is very common to combine these classes. For instance, in order to pay in the grocery store where transactions exceed a specific value, you need both the card itself (something you have) and a PIN code (something you know). Nowadays, it is also common to use smartphones as the physical object. In these scenarios, biometric authentication (something you are) is commonly used for accessing the virtual bank card.

2.1.2 Biometrics

A biometric system measures one or more physical or behavioral characteristics [4]. These characteristics are unique for each individual and separates people from one another.

Figure 2.1 illustrates different human biometrics. Examples of biological biometrics are fingerprint, vein, face and iris. Other, more environmentally developed characteristics called behavioural biometrics are for instance signature, gait and keystroke dynamics.

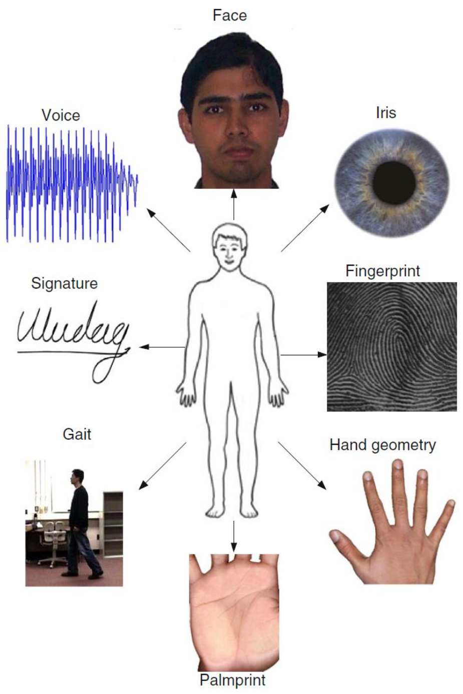


Figure 2.1: Examples of body traits that have been used for biometric authentication [4]

A biometric system will never result in a 100 percent perfect match between two samples, even from the same individual. For fingerprint capturing, there are many different sources of modifications; level of pressure on the finger while the sample was taken, direction of the finger, where on the finger the pressure was focused and so forth. We differentiate between two types of samples. The *probe* is the unidentified sample to be compared against the system database, while the *reference* is the system's trusted samples linked up with individuals [4]. Comparing two biometric samples results in a *comparison score*. *Similarity scores* are comparison scores that increases with similarity, while *dissimilarity scores* decreases with similarity.

Figure 2.2 illustrates a similarity matrix with N number of instances. Probe samples are compared with references, resulting in either a genuine score (green) or imposter score (red). The genuine scores represent instances where the probe sample and reference sample originates from the same biometric feature of an individual. Imposter scores are instances of probe samples and references that do not originates from the same individual. A *threshold* is set in order to find a value low enough to allow as many genuine scores as possible, but high enough to deny as many imposter scores as possible. The value is defined in order to conclude whether the instance is equal enough to originate from the same person [4].

The accuracy measurement of biometric systems are referred to as *False-Non-Match-Rate* (FNMR) and *False-Match-Rate* (FMR). FNMR refers to the expected

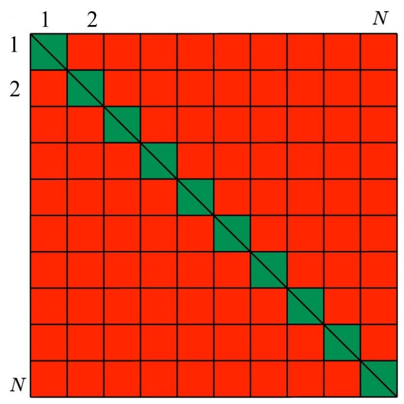


Figure 2.2: Similarity matrix [5]

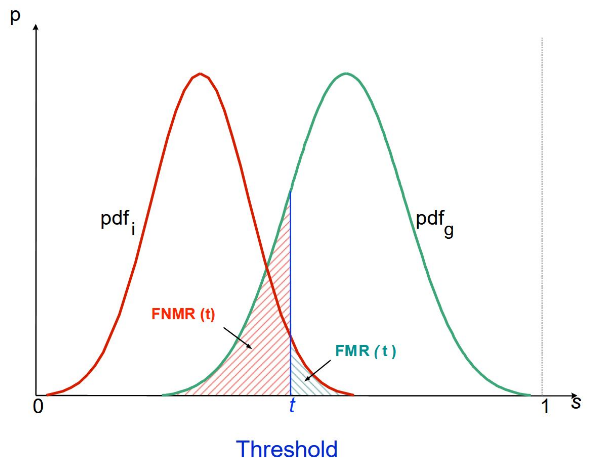


Figure 2.3: False-Match-Rate vs False-Non-Match-Rate [5]

propability that a probe and a reference sample obtained from the same individual feature will be falsely declared as a non-match. FMR is the expected propability that two non-mate samples will be incorrectly recognized as a match [4]. Figure 2.3 illustrates imposter scores (red) and genuine scores (green). It is impossible for this system to prevent all imposter attempts and simultaneously accept all genuine attempts. The threshold decides level of FNMR and FMR for the system. Systems of different purposes must adjust the threshold in order to fit their use cases. For instance, a system that controls access into a fitness gym may need a lower FNMR than other systems. In order to maintain reputation, it is important that all costumers get access into the gym facilities without any problems. Because of that, they will accept some level FMR instead of possible complains and unsubscriptions. On the other hand, companies in possession of classified information available for authorized personnel only, need a very accurate authentication system with low FMR. In biometric performance testing we are interested in a threshold that can optimally separate the two different classes; genuine samples and imposter samples. The two most common methods are:

- *Total Error Rate (TER)*
  - Where the sum of FMR and FNMR is lowest
- *Equal Error Rate (EER)*
  - Where FMR and FNMR are closest to each other.

2.1.3 Graph theory

A graph is a set containing a finite number of points, called nodes. These nodes are connected by lines called edges [6]. Let L be a finite set of labels for nodes and edges. A labeled graph g is defined as a four-tuple [7]

$$g = (V, E, \mu, \nu)$$

- V is the finite set of nodes.
- $E \subseteq V \times V$ is the set of edges.
- $\mu : V \rightarrow L_V$ is the node labeling function.
- $\nu : E \rightarrow L_E$ is the edge labeling function.

Figure 2.4 illustrates a basic graph represented by the definition above. We can see that functions μ and ν are defining the labels of nodes and edges.

By converting human written signatures into graphs, they can be labeled based on predefined criteria. Nodes are representing characteristic keypoints in the signature, and edges connect these nodes [1]. Figure 2.5 illustrates an example of how nodes can be defined in a signature. In this scenario, we have defined three types of characteristics in the graph that is converted into nodes:

- Order 1: Endpoints
- Order 2: Change of direction
- Order 4: Crossing lines

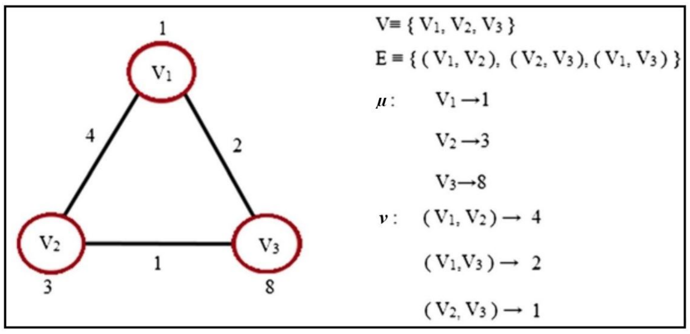


Figure 2.4: Illustration of the graph definition [6]

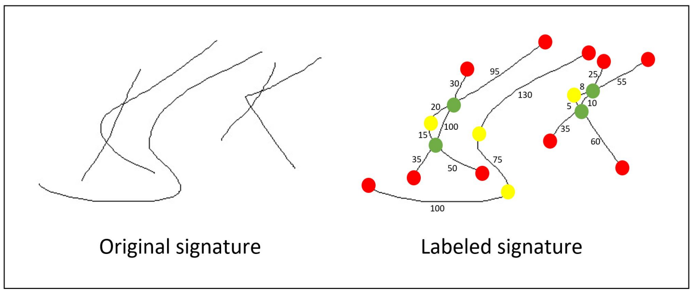
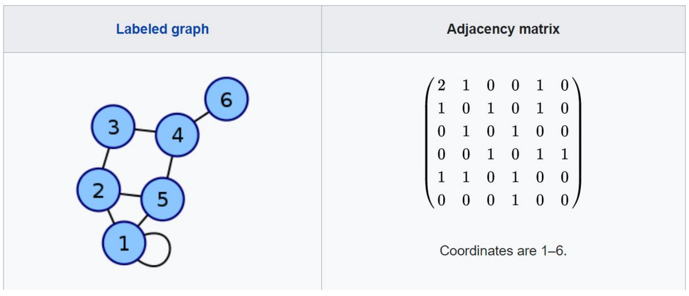


Figure 2.5: Example of a labeled signature

Red nodes are endpoints with the order of 1 because one edge is connected to those nodes. *Yellow* is order 2 with change of direction and two edge connections, while *green* is order 4 with crossing lines and four edges connected to the node. The edges are labeled with a distance calculation between the connected nodes. These calculations could be for instance number of edge pixels or straight line distance between the connected nodes.

There are different methods for representing a graph without the graphical expression. One method is an Adjacency matrix [8]. The elements of a squared matrix indicates whether nodes in the graph have edges connected to one another or not.



$$\begin{pmatrix} 2 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Figure 2.6: Adjacency matrix [9]

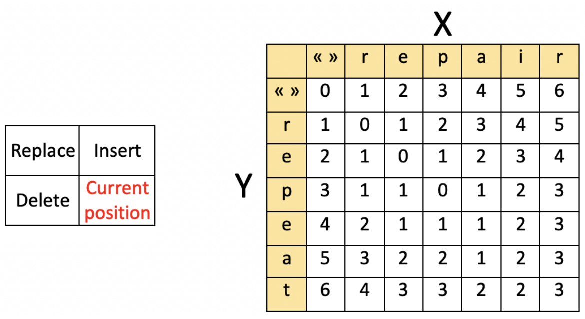
From figure 2.6 we see a graph with nodes numbered from one to six. These numbers corresponds to the rows and columns in the matrix. If we want to see relations to node 5 for instance, we either check row number five or column number five. Values not equal to zero determines edges between nodes. Value 2 in figure 2.6 indicates an edge loop.

2.1.4 Graph Edit Distance

Graph Edit Distance (GED) operates in a two-dimensional space. In order to get an understanding of Edit Distance as a concept, we are going to start with explaining the one-dimensional String Edit Distance.

String Edit Distance

Given two strings X and Y over a finite alphabet, String Edit Distance (SED) between X and Y can be defined as the minimum weight of transforming X into Y through a sequence of weighted edit operations. These operations are usually defined in terms of *insertion* of a symbol, *deletion* of a symbol, and *substitution* of one symbol for another [10]. Scientist V. I. Levenshtein introduced the distance back in 1966 [11]. It is commonly known as *Levenshtein distance.*


Replace
Insert
Delete
Current position
X
Y

	« »	r	e	p	a	i	r
« »	0	1	2	3	4	5	6
r	1	0	1	2	3	4	5
e	2	1	0	1	2	3	4
p	3	1	1	0	1	2	3
e	4	2	1	1	1	2	3
a	5	3	2	2	1	2	3
t	6	4	3	3	2	2	3

Figure 2.7: String Edit Distance calculation

Figure 2.7 illustrates the SED between *repair* and *repeat*. Each of the cells in the table represent a sub-problem. By calculating each one of them we get the global optimal edit distance. First row and column represent an empty string, and the edit distance is represented with the number of insertions to get from empty string into each words. The three operations are illustrated to the left in Figure 2.7. By placing *current position* inside a sub-problem cell, the cost of insertion is the value above, deletion to the left etc. You always chose the lowest cost and add by one for each operation. The SED between repair and repeat is *3*.

Instead of computing the number of operations, different weights for different operations could be used. For example, it is possible to weigh the cost for deletion of a character higher than replacing it with another character [12]. Text editing programs with an implemented error correction function verifies words up against the dictionary. For keyboard typing, the distance between *e* and *r* should be lower than *e* and *m* because the physical distance on the keyboard is shorter and causes a higher chance of miss-typing.

Graph Edit Distance

GED measures the amount of distortion needed to transform one graph into another graph [13]. There is a specific set of possible operations in order to perform the transformation:

- *Node insertion*: Introduces a single new labeled node to a graph.
- *Edge insertion*: Introduces a new edge between a pair of nodes
- *Node deletion*: Removes a single node from a graph.
- *Edge deletion*: Removes a single edge between a pair of nodes.
- *Node substitution*: Changes the label of the node.
- *Edge substitution*: Changes the label of the edge.

GED attempts to find the lowest possible cost by using a combination of the different operations. Instead of only having to deal with characters (SED), there are two possible labels of error; nodes and edges. This results in a higher level of complexity for the Edit Distance calculation.

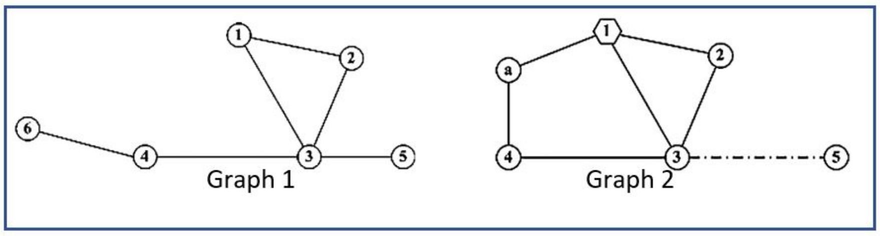


Figure 2.8: GED between two graphs [14]

Figure 2.8 illustrates two graphs that are not identical. In order to transform graph 1 into graph 2, there must be a number of edit operations. One possible solution is:

- Edge substitution between node 3 and 5
- Node 6 deletion
- Node a insertion
- Edge substitution for node 4, redirecting edge to node a
- Node 1 substitution
- Edge insertion between node a and 1

There is a defined cost function for each operation, and the entire edit operation cost value is the sum of all operations in the sequence. Important definitions are how to determine similarity of components and cost of edit operations [14]. GED separates two different types of graphs: *attributed graphs* and *non-attributed graphs*. The difference is that attributed graphs labels nodes and/or edges, while non-attributed graphs does not. We will now give a short explanation of two different approaches for GED calculation. One is based on machine learning and the other is graph-based.

Self-Organizing Maps (SOM) based algorithm [15] networks consists of two layers: input layer and competitive layer. The input layer places nodes in a space based on their features. These nodes are connected to all neurons in the competitive layer. Neurons compete in order to be the closest one to the inputs. Figure 2.9 illustrates the structure of SOM, and how an input (hollow node) is connected to each neuron (red nodes) in the competitive layer. Neurons are also connected to each others neighbors. There is a specific process in order to create the competitive layer of neurons [14].

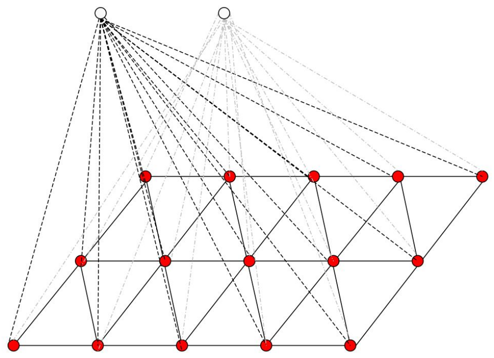


Figure 2.9: SOM structure [14]

In the context of GED, SOM is created for use on attributed graphs and is based on the already mentioned definition $g = (V, E, \mu, v)$ for objects to be processed. Every node and edge labels are m-dimensional and n-dimensional vectors, respectively. The labeled nodes and edges of a graph is placed into the input layer of a SOM network, and the competitive layer will afterward become trained [15]. The actual edit costs are derived from a distance measure for labels that is defined with the distribution encoded in SOM [14].

Bipartite approximation [7] reduces the problem of graph edit distance to an instance of a linear sum assignment problem (LSAP). Two graphs are proposed with each of its node set. These node sets are calculated from the graph normalization and is defined as $V_1 = \{u_1,...,u_n\}$ and $V_2 = \{v_1,...,v_m\}$.

$$\mathbf{C} = \left[\begin{array}{cccc|cccc}
c_{11} & c_{12} & \cdots & c_{1m} & c_{1\varepsilon} & \infty & \cdots & \infty \\
c_{21} & c_{22} & \cdots & c_{2m} & \infty & c_{2\varepsilon} & \ddots & \vdots \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \ddots & \infty \\
c_{n1} & c_{n2} & \cdots & c_{nm} & \infty & \cdots & \infty & c_{n\varepsilon} \\
\hline
c_{\varepsilon 1} & \infty & \cdots & \infty & 0 & 0 & \cdots & 0 \\
\infty & c_{\varepsilon 2} & \ddots & \vdots & 0 & 0 & \ddots & \vdots \\
\vdots & \ddots & \ddots & \infty & \vdots & \ddots & \ddots & 0 \\
\infty & \cdots & \infty & c_{\varepsilon m} & 0 & \cdots & 0 & 0
\end{array}\right]$$

Matrix C calculates the cost of each operation. Variable c_{ij} denotes the cost of a node substitution ($u_i \rightarrow v_j$), $c_{i\epsilon}$ denotes the cost of a node deletion ($u_i \rightarrow \epsilon$), and $c_{\epsilon j}$ denotes the cost of a node insertion ($\epsilon \rightarrow v_j$) [7]. The operations that results in the lowest sum are used.

2.1.5 Backtracking

In order to understand backtracking, the fundamental concept will be briefly explained first. Afterwards, these concepts will be set into context by solving a typical backtracking calculation approach; the N-Queens problem.

A backtracking algorithm attempts to find all, or some, or the most optimal solution(s) to computational problems by systematically searching through paths and continuously checking if conditions are valid before either exploring further, choosing another direction or going backwards [16]. All solutions that does not satisfy the given conditions will be removed. The algorithm is based on a trial and error method, which means that many error attempts causes higher inefficiency. Compared with brute-force approach that attempts all possible paths, backtracking will be much more efficient due to its validation of conditions at every phase.

The N-Queens problem [17][18]: Given an N x N square chess board, how can N queens in the respective rows and coloums be placed while not threatening or attacking each other? In chess, queens can attack vertically, horizontally and diagonally.

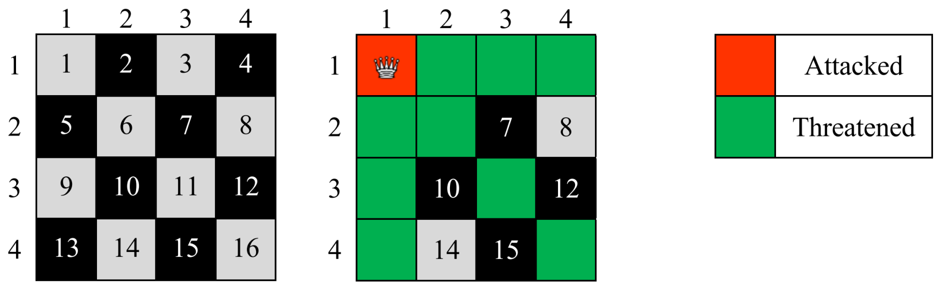


Figure 2.10: N-Queens problem illustration [16]

Figure 2.10 illustrates a 4 x 4 board with 16 different fields. If the first queen is placed in field (1,1), the remaining non-threatened fields are 7, 8, 10, 12, 14 and 15. There are many decision points throughout this problem solving, which benefits the backtracking approach. In order to program backtracking, there are three important aspects of the code; the choice that has to be made at each stage, indications of when the program must stop that path or not even considering it, and defining the target goal of the operations. For this scenario, the backtracking algorithm must decide which path to do next.

The decision tree (figure 2.11) shows how the algorithm must run through the different possibilities until it finds a valid path that meets the required conditions. At each stage, it systematically determines next path and checks if that choice

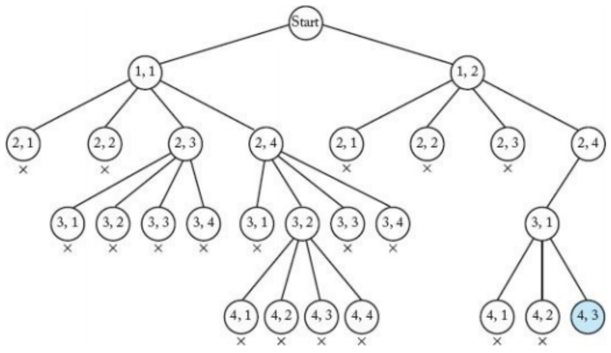


Figure 2.11: N-Queens backtracking illustration [17]

leads to any conflicts with other queens on the table. If so, no further investigations are done within that path. If the choice does not lead to any conflicts, the program will investigate that path further. We see from figure 2.11 that no valid paths were found during the investigation from starting-point (1,1) on the table. Starting-point (1,2) on the other hand discovered one valid set of queen placements for a possible solution of 4-Queens problem. In comparison, brute force attempt tries all different paths from top to bottom and does not check the queen conditions until after all stages. This may not be too comprehensive for this scenario with 4 Queen, but with 8 x 8 8-Queens problem there will be a total of $64 \times 63 \times 62 \times ... \times 57 = 178,462,987,637,760$ different cases [18]. Backtracking is an efficient way of finding valid paths.

2.2 State of Art

This chapter illustrates detailed knowledge related to the project. The first and second part go in-depth on offline signature authentication and GED separately, while the last section combines these two subjects.

2.2.1 Signature Authentication

There are two main implementation methods for signature-based biometric systems. *Online* signature authentication uses dynamic features from the signing process in combination with the actual signature. These features are collected with a digital capturing device, measuring pen pressure, tilt of pen, writing speed along with others [19]. *Offline* considers the shape of the signature only [20]. There are some fundamental pre-processing steps in order to prepare the signature data, and they are briefly illustrated below. [21].

Data Acquisition

Since offline signatures are written with pen and paper, the data must be converted into a digital format for further analysis. Common techniques are scanning or digital images of the signature. If the main interest is the signature pattern, an online signature electronic device could be used for shape capturing.

Signature Pre-processing

The complexity of the pre-processing part depends majorly on whether to use online or offline samples. Image processing is necessary for all offline samples because the signature shape must be extracted from surrounding noise in the photo. This process is called binarization and gives each pixel two possible values; zero and one. For optimal binarization, there should be a major color difference between surrounding background and the signature text (e.g black and white). This is to avoid any misclassifications from the pixel leveled threshold. For even more precise representation of the signature, the binary sample is converted into a skeleton. This process makes the signature lines one pixel wide. Figure 2.12 illustrates an example of binary and skeleton image pre-processing from an original offline signature.

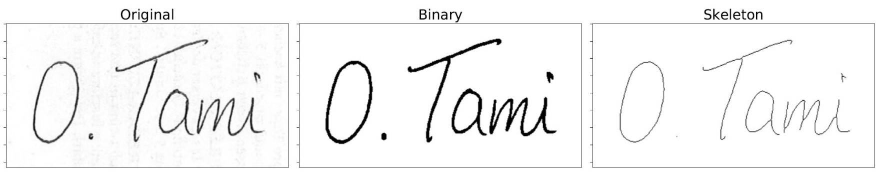


Figure 2.12: Signature Pre-processing [1]

Feature Extraction

There exist many different feature extraction algorithms used for gathering information from signature images. One approach called Local Binary Pattern (LBP) will be explained in more detail. LBP is a general and widely used method for describing certain parts of an image. The approach labels pixels of an image as a bit string [22]. The binary string is based on one pixel's surrounding neighbor pixels. Subtracting this center pixel by each neighbor gives an either positive or negative encoding. Positive values are encoded as one, while negatives as zero. These steps are illustrated in figure 2.13. Descriptors like LBP are capable of recognising different features of a signature, e.g. number of holes, distributions, projections, number of branches in the skeleton, tortuosities and many others [1].

Below are some specific and common features extracted from the signature shape:

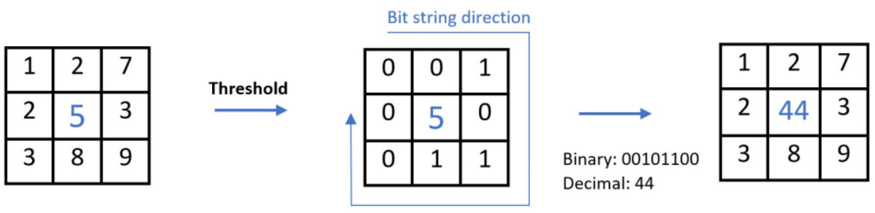


Figure 2.13: Local Binary Pattern

- *Dominant angle* [21]
  - Evaluates the angle of the written signature. Some people tend to not write in a straight line, and valuable information are possibly extracted from this analysis.
- *Signature triSurface feature* [23]
  - Evaluates the surface area of the signature. The signature is separated into three equal parts, vertically. The surface area is the surface covered by the signature, including the holes. All black pixels in each surface are counted and compared with all background pixels from the entire signature image, resulting in a percentage score between the three surfaces.
- *Length feature* [23]
  - Represents the signature length after scaling all signatures from the database into the same height. In order to normalize the length into values between 0 and 1, the shortest signature in the database are valued 0, while the longest are valued 1. All other signatures are converted into values somewhere in between.
- *Texture feature* [24]
  - Analyses the pixel pattern. In order to extract the texture feature group, the co-occurrence matrices of the signature image is used. For a binary signature image, the co-occurrence matrix pd[i,j] is defined by first specifying the displacement vector d = (dx, dy) [21]. This vector could for instance go through the middle of the signature horizontally and counting all pairs of pixels. The co-occurrence matrix is 2 × 2 and describes the transition between black and white pixels and is defined as

$$Pd[i,j] = \begin{pmatrix} p00 & p01 \\ p10 & p11 \end{pmatrix}$$

    Where p00 is the number of instances with two white pixels connected, p01 is from white to black, p10 is black to white and p11 is two black pixels.

Matching

After the extraction of the signature features, the data is ready to be compared with other signatures in the database. A matching-technique is required to perform this comparison calculation. One approach is to use distance metrics. This could for instance be *Euclidean distance* or *Manhattan distance* [25]. Other, more advanced approaches are classification algorithms such as the binary classifier *Support Vector Machine* or the multi-class *k-Nearest Neighbors*.

Biometric datasets come with a given number of samples from each individual. The online signature dataset MCYT-100 [26] has samples from 100 different individuals. Each individual has 25 genuine and 25 skilled forgery samples each. MCYT-75 [27] is an offline dataset containing 75 users with 15 genuine samples and 15 skilled forgeries each.

A good matching algorithm can reliably distinguish between original and forgery samples. The database must be split up into probes and references. The probe samples should include the majority, but the defined references must contain enough samples in order to represent the user in a satisfactory manner. When a probe signature is provided, it is compared with each of the signature samples in the reference. Each comparison will result in a separate score. There are different approaches for selecting the final comparison score between one probe and the references. Examples are the lowest, the highest or the average score.

2.2.2 Signature authentication using GED

This sub-chapter goes in depth on specific published literature related to the combination of signature authentication with the use of graph edit distance. The literature is mainly based on two papers; [1] and [7].

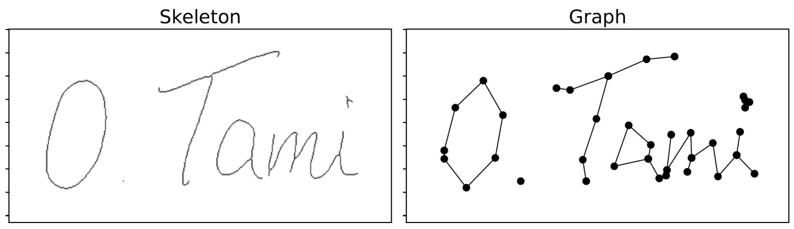


Figure 2.14: Keypoint graph [1]

As mentioned in chapter 2.2.1, the first step of signature pre-processing for offline signatures is skeletonization. By using the definition of labeled graphs from chapter 2.1.3, $g = (V, E, \mu, \nu)$, a signature can be labeled based on certain criteria. [1] defines *keypoint* graphs as labeled graphs, where the labels are based on characteristics extracted from an image of handwriting. Figure 2.14 illustrates an example of keypoint graph from a skeleton image. Two types of points are converted into nodes: characteristic keypoints in the signature pattern and consistent points between those characteristics. The characteristic keypoints are carried out by looking at each black pixel in the skeleton image. Black pixels with exactly one neighbored pixel are *end points*. If the pixel has more than two neighbored pixels, it is defined as *possible junction points*. Incidents of multiple possible junction points next to each other confirms a *junction point*. The average of these pixels are chosen as the exact position. When the signature is a whole circle without any end points or junction points, the left outer pixel is chosen as keypoint. A distance parameter is chosen in order to fill the rest of the signature with additional nodes between the keypoints.

[1] defines different forms of graph normalization. One approach is to exclude normalization and use the raw node labels only. Another approach is to move all nodes such that the center of the signature becomes the point of origin. It is also possible to extend the center normalization by scaling the signature such that the standard deviation is 1 in both x - and y - direction.

Both [1] and [7] uses bipartite approximation as introduced on chapter 2.1.4. The *cost function* is an important decision during the calculation of GED. For [1], in case of substitution the cost model is based on the coordinate labels of the nodes. The Euclidean distance is used between two node labels:

$$c(u \rightarrow v) = \sqrt{(x_u - x_v)^2 + (y_u - y_v)^2}$$

In case of insertion or deletion, the cost model uses average edge length. This is the weighted average length of all edges in the current reference graph, defined as $m(g_1)$. The calculation is as follows:

$$c(u \rightarrow \epsilon) = c(\epsilon \rightarrow v) = \gamma_{node} \cdot m(g_1)$$

where $\gamma_{node} \in \mathbb{R}^+$

The cost of edge insertion and deletion is defined as follows:

$$c(e_1 \rightarrow \epsilon) = c(\epsilon \rightarrow e_2) = \gamma_{edge} \cdot m(g_2)$$

where $\gamma_{edge} \in \mathbb{R}^+$. The cost of edge substitution is set to zero, defined $c(e_1 \rightarrow e_2) = 0$.

Chapter 3

Signature processing

3.1 Data

As already mentioned in chapter 2.2.1, MCYT-100 [26] is an online signature dataset with signatures from 100 different individuals. Each one of the 100 individuals had provided 25 samples, and 25 samples were skilled forgeries trying to simulate the original signatures. The signatures were stored as fpg-files. In order to extract the file information into a visible drawing, the dataset included an attached script. This script extracted and visualized information related to online signature values as well. Since we were interested in the shape only, the extended online data was excluded from our calculations.

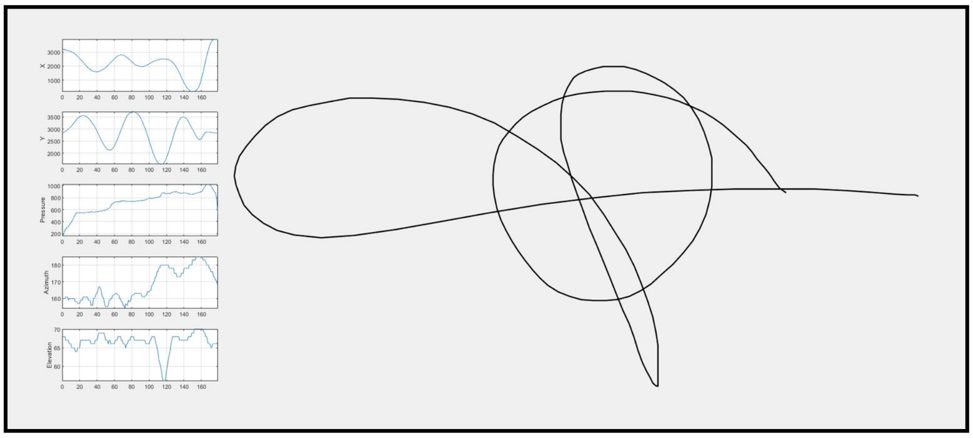


Figure 3.1: Illustration of the MCYT-100 dataset with signature *0021f03*

Figure 3.1 is an example from the MCYT-100 dataset. From a close look it is possible to see that the signature is constructed from multiple straight lines. The graphs to the left are the environmental data. We are going to use this particular signature (*0021f03*) for explaining the processing steps throughout this chapter.

In comparison, the offline dataset MCYT-75 [27] uses photos of the signatures. An example is illustrated in figure 3.2.

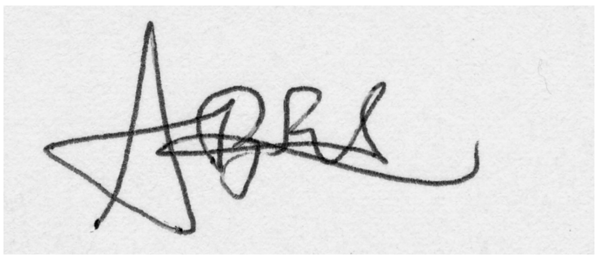


Figure 3.2: Example signature from the MCYT-75 dataset [27]

3.2 Signature pre-processing

This section will go through every step in the pre-processing part, from the raw fpg-file until we have captured all valuable information about the signature.

3.2.1 Extract coordinates

We collected all valuable information about the signature by using a Matlab-script attached to the MCYT-100 dataset called *`FPG_Signature_Read.m`*. This function returned x and y coordinates captured with a distance interval, and the Matlab-function *plot* drew straight lines in between for signature visualization. We used Bresenham's line algorithm [28] for constructing the remaining in-between coordinates. The new set of coordinates gave a complete representation of the signature, and could simply be converted into an image for offline signature calculation or be used for pixel distance estimations between nodes. At this point, we have extracted the signature coordinates into an unlabeled graph without any defined characteristics.

3.2.2 Collection of nodes

One way to estimate where nodes of degree 1 and 4 were placed in the signature was to use the offline approach. The signature was then processed as a binary skeletonized image. Nodes of degree 4 were captured with the use of LBP [22] estimations. Since this method didn't result in 100 percent accurate calculations in our tests, we chose to instead use the online approximation. This method calculated the crossing lines based on plotted lines between the coordinates captured from *`FPG_Signature_Read.m`*. This gave endless scalability, so there was no doubt in whether or not two lines were crossing each other.

Figure 3.3 illustrates scenarios where offline pre-processing of node detection struggled. When two lines were near each other without touching, it was difficult for the offline pre-processing tools to not consider these as crossing lines. On the right scenario in figure 3.3, we experienced multiple false detections of degree 4 nodes with LBP.

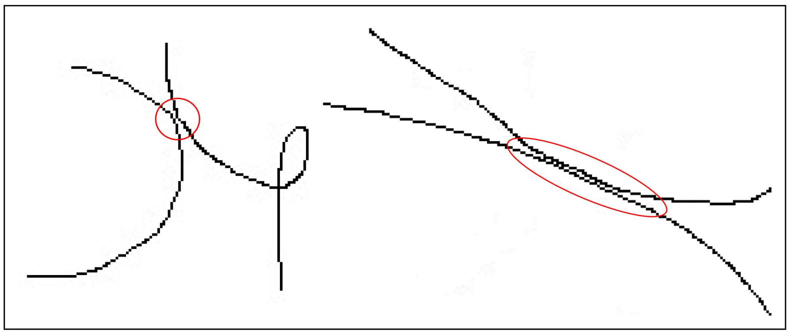


Figure 3.3: Problem with offline pre-processing

Degree 1 nodes

As already mentioned, signature pattern information from the MCYT-100 dataset were pure coordinates. Degree 1 nodes will always be endpoints of these coordinates. As you can see from table 3.1, when the pen was lifted and moved to another location, the coordinates were represented by Not a Number (NaN). This meant that endpoints were collected when:

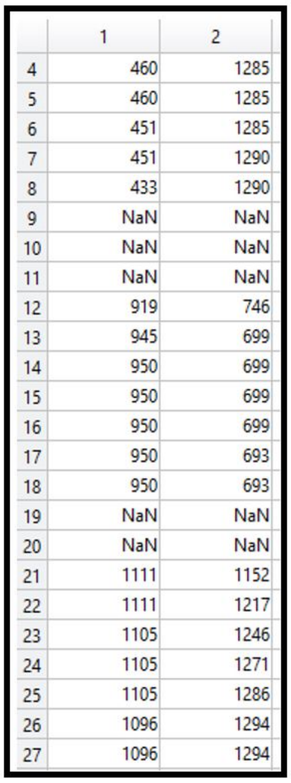
- It was the first coordinate in the table
- It was the last coordinate before NaN value
- It was the first value after NaN has occurred
- It was the last value in the table of coordinates

Degree 4 nodes

As mentioned earlier, by analyzing the signature from a graph plot instead of an offline image, the node calculation became 100 percent accurate. The Matlab-function *polyxpoly* returned coordinates of every intersection. This function also returned coordinates when the graph changed direction. Since we were interested in intersections of crossing lines only, we used the Matlab-function *setdiff* in order to exclude coordinates of the signature itself. The return value of this function gave all crossing lines in the signature graph.

Figure 3.4 shows an example of a signature graph with collected nodes of degree 4. The image to the right (inside figure 3.4) is an enlarged clip of the graph, where you can see that the online signature representation gave advantages when it came to accuracy since the sample did not rely on image resolution.

At this point, we have collected all nodes of degree 1 and 4 in the signature. Figure 3.5 also illustrates the corresponding numbers. The graph coordinates were systematically organized and ordered as the signatures were written, and our program numbered the nodes as they occurred in the analyzing process. This is due to the input coordinates as described in chapter 3.2.1. If we follow the numbering, we can see the user path of writing the signature.



	1	2
4	460	1285
5	460	1285
6	451	1285
7	451	1290
8	433	1290
9	NaN	NaN
10	NaN	NaN
11	NaN	NaN
12	919	746
13	945	699
14	950	699
15	950	699
16	950	699
17	950	693
18	950	693
19	NaN	NaN
20	NaN	NaN
21	1111	1152
22	1111	1217
23	1105	1246
24	1105	1271
25	1105	1286
26	1096	1294
27	1096	1294

Table 3.1: Example of x and y coordinates from MCYT-100 dataset

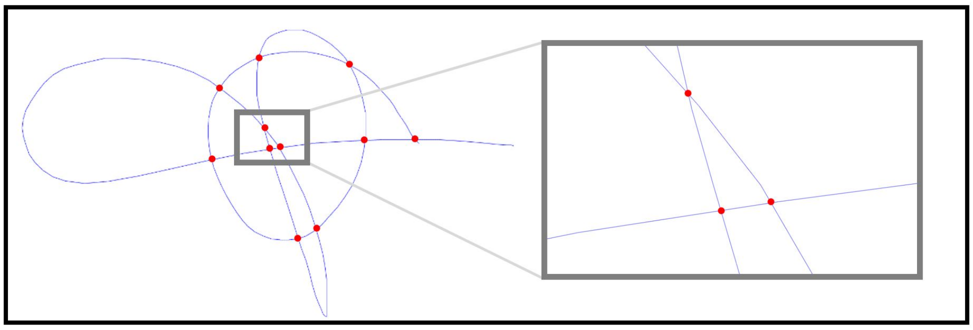


Figure 3.4: Signature with collected nodes of degree 4

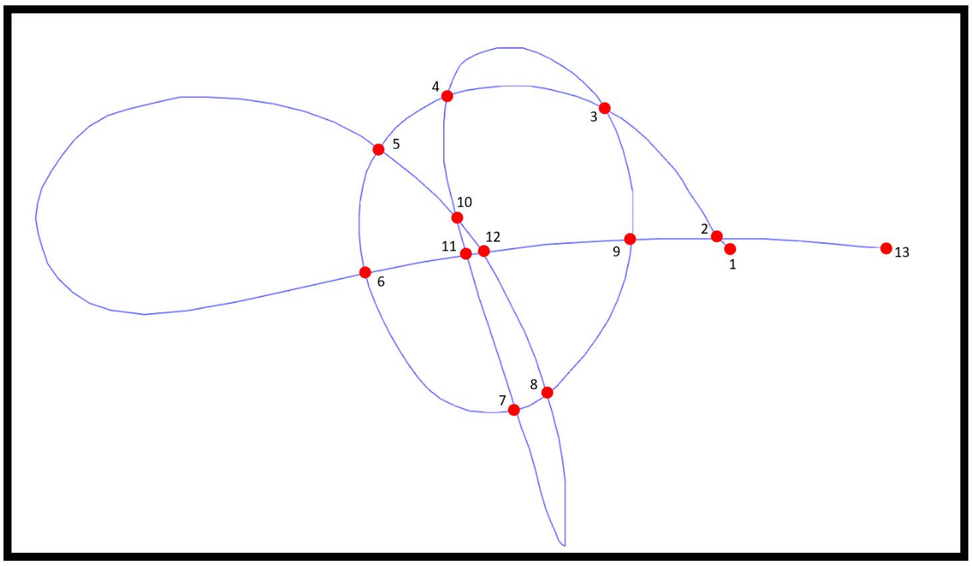


Figure 3.5: All captured nodes

3.2.3 Signature matrix

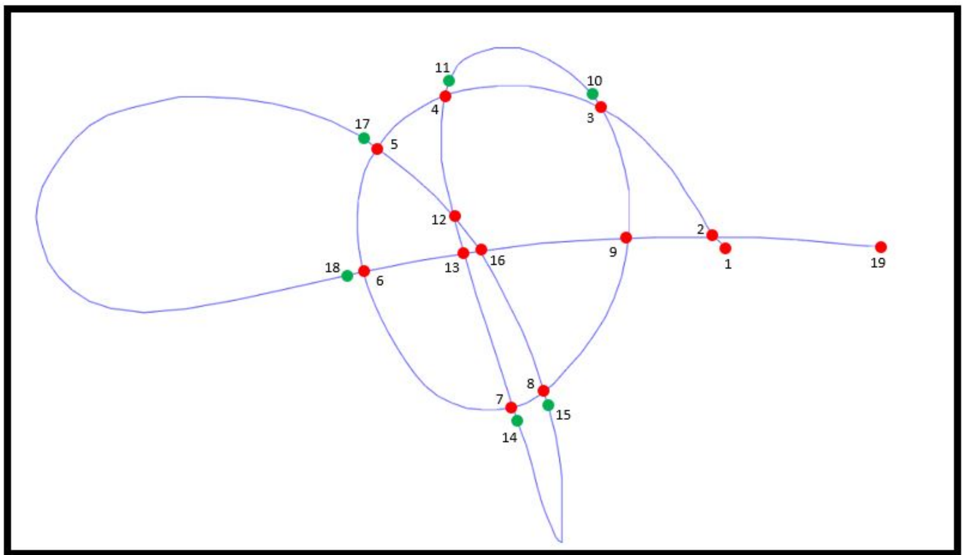
Information about the signature needed to be captured in a format that gave a clear understanding of the graph structure, as well as the edge labels. In section 2.1.3 we introduced Adjacency matrix [8]. We chose this method as the signature information storage format. Figure 3.6 illustrates how the signature graph was stored as a matrix.

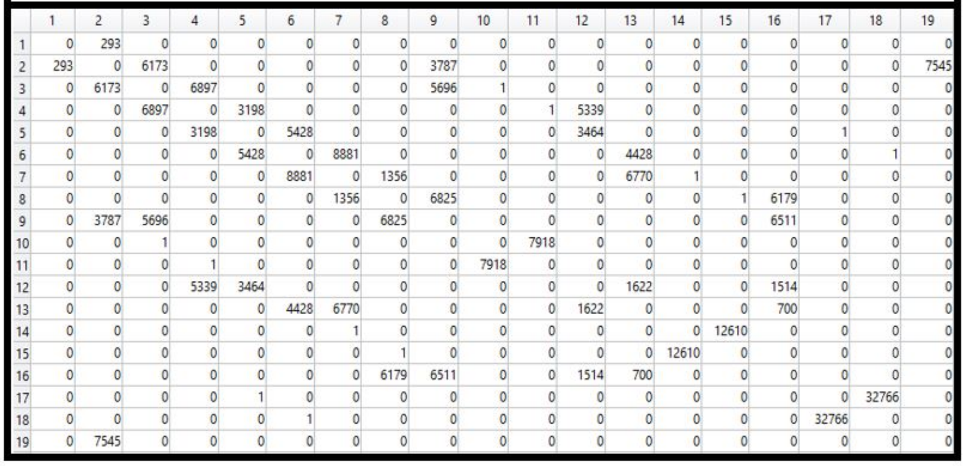
Construction

We wanted to store all information about the signature in the adjacency matrix. Normally, Adjacency matrices only collects information about whether or not nodes are connected. The matrix row/column has value 1 if there is a connection with another node and value 2 if its connected to itself or if there are two possible routes between nodes.

Instead of collecting information about connections only, we stored the edge labeling information inside the matrix. Figure 3.6 shows the signature and the corresponding adjacency matrix. If we wanted to find the distance between node 1 and 2, we checked at row 1 and column 2 (or row 2 and column 1). In this example the distance was 293 pixels.

Since we changed the normal adjacency matrix setup, we had to make some considerations. As mentioned above, when there are two possible routes between two nodes, the standard value inside an adjacency matrix is 2. This is not possible to represent in our matrix approach. Instead, we chose to split one of the two routes and created a new set of nodes. These nodes were all marked with green in figure 3.6 (for instance node 10 and 11). The distance between the original





	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	0	293	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	293	0	6173	0	0	0	0	0	3787	0	0	0	0	0	0	0	0	0	7545
3	0	6173	0	6897	0	0	0	0	5696	1	0	0	0	0	0	0	0	0	0
4	0	0	6897	0	3198	0	0	0	0	0	1	5339	0	0	0	0	0	0	0
5	0	0	0	3198	0	5428	0	0	0	0	0	3464	0	0	0	0	1	0	0
6	0	0	0	0	5428	0	8881	0	0	0	0	0	4428	0	0	0	0	1	0
7	0	0	0	0	0	8881	0	1356	0	0	0	0	6770	1	0	0	0	0	0
8	0	0	0	0	0	0	1356	0	6825	0	0	0	0	0	1	6179	0	0	0
9	0	3787	5696	0	0	0	0	6825	0	0	0	0	0	0	0	6511	0	0	0
10	0	0	1	0	0	0	0	0	0	0	7918	0	0	0	0	0	0	0	0
11	0	0	0	1	0	0	0	0	0	7918	0	0	0	0	0	0	0	0	0
12	0	0	0	5339	3464	0	0	0	0	0	0	0	1622	0	0	1514	0	0	0
13	0	0	0	0	0	4428	6770	0	0	0	0	1622	0	0	0	700	0	0	0
14	0	0	0	0	0	0	1	0	0	0	0	0	0	0	12610	0	0	0	0
15	0	0	0	0	0	0	0	1	0	0	0	0	0	12610	0	0	0	0	0
16	0	0	0	0	0	0	0	6179	6511	0	0	1514	700	0	0	0	0	0	0
17	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	32766	0
18	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	32766	0	0
19	0	7545	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3.6: Signature with the corresponding Adjacency matrix where edges are labeled by pixel distance

node and the new one is set to 1 pixel, which is visible in the adjacency matrix between node 3 and 10. The distance between node 10 and 11 is in this example represented by the other possible path between node 3 and 4.

We have focused on three different methods of labeling the edges in the signature; pixel distance between nodes, Euclidean distance and the efficiency ratio between these two.

Calculation of edge labels

The signature itself is not based on pixels due to the online signature implementation. We considered two possible options for finding the actual typing distance between two nodes. One way was the estimation of Euclidean distance between all coordinates from one node to another. Another approach was to calculate all coordinates in each line with the use of Bresenham's line algorithm [28] and then count the number of coordinates between the nodes. This method is transferable to pixel distance for offline signatures. We chose the pixel distance approach for our estimation, without arguments for or against one over the other. This method gave good performance when it came to execution speed. Euclidean distance was implemented with the use of the formula mentioned in section 2.2.2. The final edge labeling implementation we chose to investigate further was the efficiency ratio between Pixel distance and Euclidean distance. In this approach Euclidean distance was divided by Pixel distance.

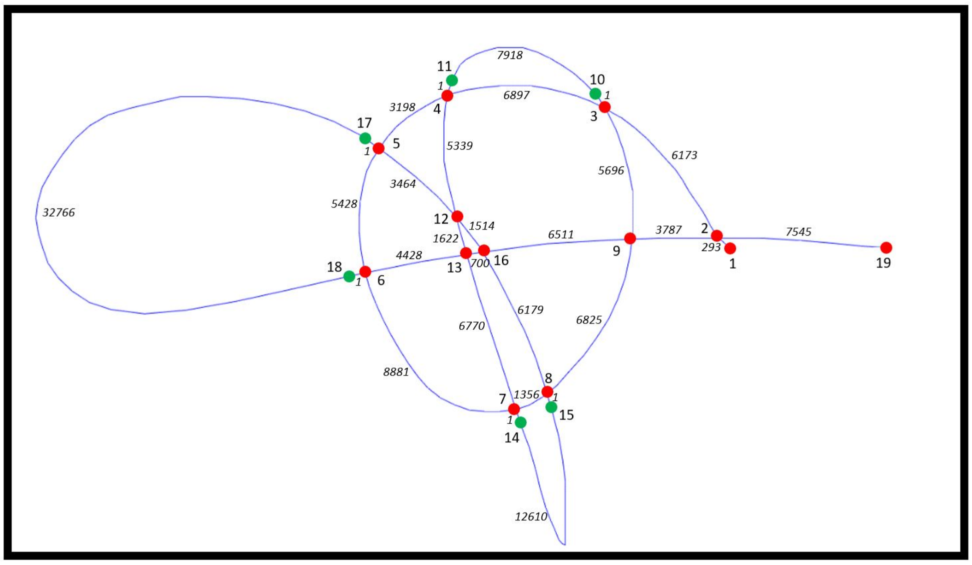
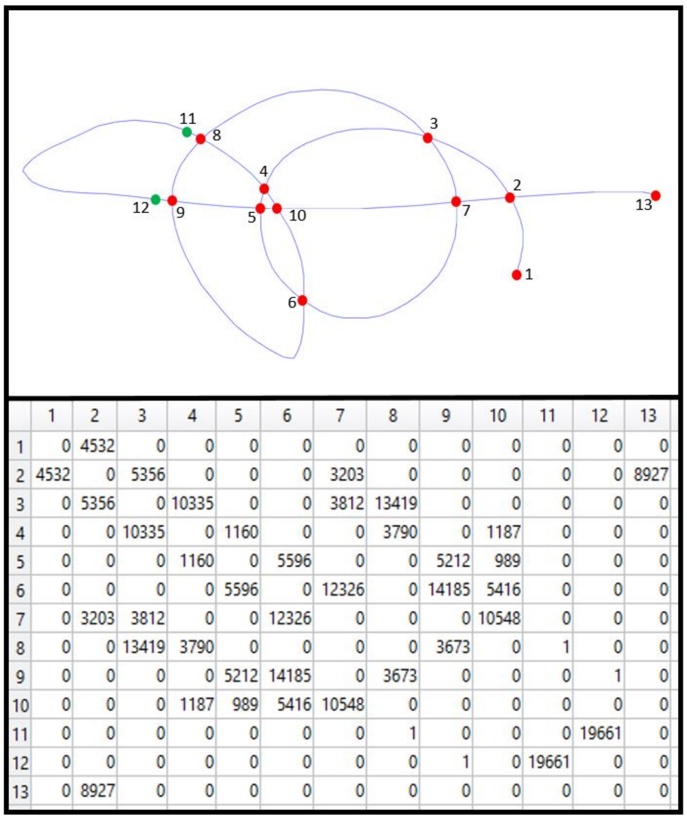


Figure 3.7: Signature with numbered nodes and labeled edges by pixel distance

We have now completed the pre-processing part, and the result is illustrated in figure 3.7. All nodes have been detected with either a degree of 1 or 4. Corresponding edges with labeled distance values are also defined and stored in the adjacency matrix.

3.3 Comparison algorithm details

This section talks about how the cost estimations were made, and goes through the calculations of the node mapping cost. We are going to explain the comparison techniques by comparing the already familiar signature (figure 3.6), now defined as signature 2, against signature in figure 3.8, now defined as signature 1.



	1	2	3	4	5	6	7	8	9	10	11	12	13
1	0	4532	0	0	0	0	0	0	0	0	0	0	0
2	4532	0	5356	0	0	0	3203	0	0	0	0	0	8927
3	0	5356	0	10335	0	0	3812	13419	0	0	0	0	0
4	0	0	10335	0	1160	0	0	3790	0	1187	0	0	0
5	0	0	0	1160	0	5596	0	0	5212	989	0	0	0
6	0	0	0	0	5596	0	12326	0	14185	5416	0	0	0
7	0	3203	3812	0	0	12326	0	0	0	10548	0	0	0
8	0	0	13419	3790	0	0	0	0	3673	0	1	0	0
9	0	0	0	0	5212	14185	0	3673	0	0	0	1	0
10	0	0	0	1187	989	5416	10548	0	0	0	0	0	0
11	0	0	0	0	0	0	0	1	0	0	0	19661	0
12	0	0	0	0	0	0	0	0	1	0	19661	0	0
13	0	8927	0	0	0	0	0	0	0	0	0	0	0

Figure 3.8: Signature 1 for comparison

3.3.1 Cost estimation

In order to calculate a similarity score between two signatures, we first had to look closer into node similarities and their connected edge labels. When we compared two nodes with a difference in number of connected edges, they were so dissimilar that a mapping would be inefficient. Therefore, a very high cost was given for each insertion/removal of edges. Finding a valid cost for changing the edge distance from one node to another was solved with an exponential approach. The higher distance, the higher the cost. For Pixel and Euclidean distance, the costs were calculated with the following formula:

$$COST = 2^{|E_a - E_b| \div 1000}$$

where E_a and E_b are the edge distances to be compared.

For Efficiency calculation, the values were somewhere in between zero and one. More similar Efficiency ratio resulted in lower transformation cost. We used the formula:

$$COST = 1.5^{|E_a - E_b| \times 100}$$

S2 N1 → S1 N1	
293	4532
0	0
0	0
0	0
COST = 18,9	

S2 N2 → S1 N2	
293	3203
3787	4532
6173	5356
7545	8927
COST = 13,6	

S2 N1 → S1 N2	
293	3203
0	4532
0	5356
0	8927
COST = 300 007,5	

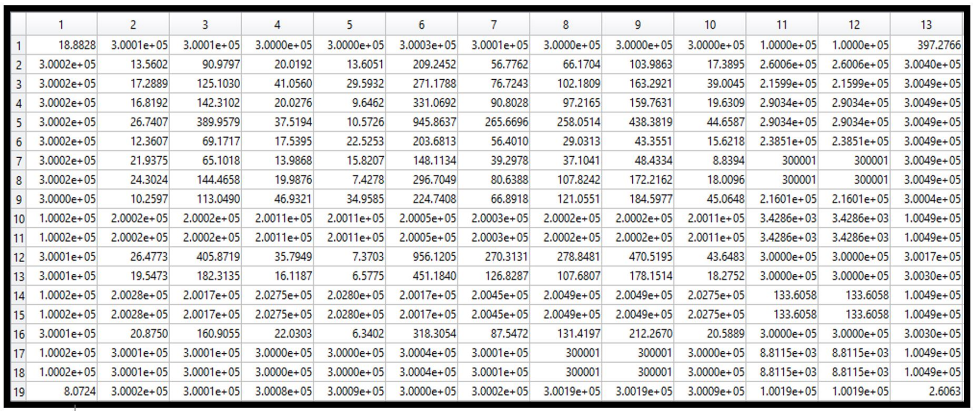
Table 3.2: Cost tables for node to node comparison

Table 3.2 illustrates three scenarios of node comparison between signature 1 and 2 with pixel distance as edge labeling. In the table headlines, *S1* stands for signature 1, *N1* stands for node 1 etc. The underlying columns are edge values for their corresponding node. These edge values can be found in table 3.6 and 3.8. Values of zero represent no edge. From the left table we see that the two nodes have one connected edge only. By using the formula above, we get a cost of 18,9 for transforming S2 N1 into S1 N1. In the middle table, we see a scenario where both nodes have four connected edges. We wanted to map edges that were most similar to each other in order to minimize the cost. One approach was to compare each and every edge mapping opportunity. That would require much processing and the execution time would be significantly longer. Instead, we sorted the edges in ascending order, which gave an estimation of lowest mapping cost. By adding the cost of all four edge transformations, we got a final cost of 13,6. The right table illustrates a comparison with a mismatch in the number of edges between the nodes. For each and every addition/removal, a cost of 100 000 was added. Since we needed to add three edges in this scenario, that cost became very high and would most likely not become a preferred option.

The entire comparison of all nodes between S1 and S2 resulted in the cost map illustrated in table 3.3. From this table, we collected each and every transformation cost between the two signatures. Node numbers from S1 are represented as columns, while node numbers of S2 are rows. The corresponding cell is the transformation cost. For instance, if we want to check the node comparison between S1 N5 and S2 N7, we check row 7 column 5. This cost was 15,8.

3.3.2 First estimation and Backtracking from cost map

The final step was to find the lowest possible cost for mapping all nodes from one signature into another. We could not reuse any nodes, so it was important



	1	2	3	4	5	6	7	8	9	10	11	12	13
1	18.8828	3.0001e+05	3.0001e+05	3.0000e+05	3.0000e+05	3.0003e+05	3.0001e+05	3.0000e+05	3.0000e+05	3.0000e+05	1.0000e+05	1.0000e+05	397.2766
2	3.0002e+05	13.5602	90.9797	20.0192	13.6051	209.2452	56.7762	66.1704	103.9863	17.3895	2.6006e+05	2.6006e+05	3.0040e+05
3	3.0002e+05	17.2889	125.1030	41.0560	29.5932	271.1788	76.7243	102.1809	163.2921	39.0045	2.1599e+05	2.1599e+05	3.0049e+05
4	3.0002e+05	16.8192	142.3102	20.0276	9.6462	331.0692	90.8028	97.2165	159.7631	19.6309	2.9034e+05	2.9034e+05	3.0049e+05
5	3.0002e+05	26.7407	389.9579	37.5194	10.5726	945.8637	265.6696	258.0514	438.3819	44.6587	2.9034e+05	2.9034e+05	3.0049e+05
6	3.0002e+05	12.3607	69.1717	17.5395	22.5253	203.6813	56.4010	29.0313	43.3551	15.6218	2.3851e+05	2.3851e+05	3.0049e+05
7	3.0002e+05	21.9375	65.1018	13.9868	15.8207	148.1134	39.2978	37.1041	48.4334	8.8394	300001	300001	3.0049e+05
8	3.0002e+05	24.3024	144.4658	19.9876	7.4278	296.7049	80.6388	107.8242	172.2162	18.0096	300001	300001	3.0049e+05
9	3.0000e+05	10.2597	113.0490	46.9321	34.9585	224.7408	66.8918	121.0551	184.5977	45.0648	2.1601e+05	2.1601e+05	3.0004e+05
10	1.0002e+05	2.0002e+05	2.0002e+05	2.0011e+05	2.0011e+05	2.0005e+05	2.0003e+05	2.0002e+05	2.0002e+05	2.0011e+05	3.4286e+03	3.4286e+03	1.0049e+05
11	1.0002e+05	2.0002e+05	2.0002e+05	2.0011e+05	2.0011e+05	2.0005e+05	2.0003e+05	2.0002e+05	2.0002e+05	2.0011e+05	3.4286e+03	3.4286e+03	1.0049e+05
12	3.0001e+05	26.4773	405.8719	35.7949	7.3703	956.1205	270.3131	278.8481	470.5195	43.6483	3.0000e+05	3.0000e+05	3.0017e+05
13	3.0001e+05	19.5473	182.3135	16.1187	6.5775	451.1840	126.8287	107.6807	178.1514	18.2752	3.0000e+05	3.0000e+05	3.0030e+05
14	1.0002e+05	2.0028e+05	2.0017e+05	2.0275e+05	2.0280e+05	2.0017e+05	2.0045e+05	2.0049e+05	2.0049e+05	2.0275e+05	133.6058	133.6058	1.0049e+05
15	1.0002e+05	2.0028e+05	2.0017e+05	2.0275e+05	2.0280e+05	2.0017e+05	2.0045e+05	2.0049e+05	2.0049e+05	2.0275e+05	133.6058	133.6058	1.0049e+05
16	3.0001e+05	20.8750	160.9055	22.0303	6.3402	318.3054	87.5472	131.4197	212.2670	20.5889	3.0000e+05	3.0000e+05	3.0030e+05
17	1.0002e+05	3.0001e+05	3.0001e+05	3.0000e+05	3.0000e+05	3.0004e+05	3.0001e+05	300001	300001	3.0000e+05	8.8115e+03	8.8115e+03	1.0049e+05
18	1.0002e+05	3.0001e+05	3.0001e+05	3.0000e+05	3.0000e+05	3.0004e+05	3.0001e+05	300001	300001	3.0000e+05	8.8115e+03	8.8115e+03	1.0049e+05
19	8.0724	3.0002e+05	3.0001e+05	3.0008e+05	3.0009e+05	3.0000e+05	3.0002e+05	3.0019e+05	3.0019e+05	3.0009e+05	1.0019e+05	1.0019e+05	2.6063

Table 3.3: Cost map for comparison between signature 1 and 2

to optimize the mappings to get the overall lowest cost when all transformations were added together. An efficient method was to use the *first estimation* approach where the lowest available transformation costs were used for mapping.

1	3	19	8,07	8	**1**	18,88		10	100 023,12		11	100 023,12		14	100 023,12		15	100 023,12
2	5	**9**	10,26		6	12,36		2	13,56		4	16,82		3	17,29		13	19,55
3	15	7	65,10	16	6	69,17	17	2	90,98	19	9	113,05	20	**3**	125,10		4	142,31
4	6	7	13,99	7	**13**	16,12		6	17,54		8	19,99		2	20,02		4	20,03
5	2	**16**	6,34		13	6,58		12	7,37		8	7,43		4	9,65		5	10,57
6	24	7	148,11	26	6	203,68	27	2	209,25	28	9	224,74	29	3	271,18	30	**8**	296,70
7	10	7	39,30	13	6	56,40	14	**2**	56,78		9	66,89		3	76,72		8	80,64
8	9	**6**	29,03		7	37,10		2	66,17		4	97,22		3	102,18		13	107,68
9	11	6	43,36	12	7	48,43	18	2	103,99	25	**4**	159,76		3	163,29		8	172,22
10	4	**7**	8,84		6	15,62		2	17,39		8	18,01		13	18,28		4	19,63
11	21	**14**	133,61		15	133,61		10	3 428,64		11	3 428,64		17	8 811,45		18	8 811,45
12	22	14	133,61	23	**15**	133,61		10	3 428,64		11	3 428,64		17	8 811,45		18	8 811,45
13	1	**19**	2,61		1	397,28		10	100 486,40		11	100 486,40		14	100 486,40		15	100 486,40

Table 3.4: First estimation of cost between signature 1 and 2

Table 3.4 illustrates how first estimation was calculated for mapping nodes from S1 into S2. Excessively high mapping costs were excluded from this table in order to save space. The brown left column represent all 13 nodes from S1. Grey columns represent the node IDs of S2, with the corresponding cost value to the right. This table is sorted so that the cheapest transformation costs for each S1 nodes from Table 3.3 are closest to the left. If we then follow the row, we see the second cheapest and so on. By taking S1 N1 (which is the first row) as an example, we see that the cheapest transformation cost was S2 N19 with a cost of 8,07. The second cheapest was S2 N1 with a cost of 18,88 and so on. We started off this first estimation calculation by selecting the lowest cost across all nodes. This was the transformation of S1 N13 into S2 N19 with a cost of 2,61. As illustrated in the table, we first marked this action as number 1 (in yellow) besides the S2 node ID and then marked that cost cell with green which means it was taken. Action number 2 occurred at S1 N5 where S2 N16 had a cost of 6,34. Third lowest cost happened at S1 N1 where S2 N19 had a cost of 8,07. The problem was that S2 N19 already had been taken at action 1. Because of that, we could not complete this mapping and marked the cell with red. These procedures were executed until all S1 nodes were mapped.

Since there were many different variants of mapping, we also needed to ensure that the calculation estimated the lowest possible cost. In order to make the script more efficient, we had to avoid checking all possible combinations. In order to do that, we used backtracking. Lowest collected cost from first estimation was always used as comparison for the backtracking algorithm in order to skip paths with higher cost. Once a higher cost was detected, the script stopped without further calculations and calculated a new possible route. Once a lower cost route was found, the backtracking algorithm used that value as benchmark. We decided to always transform the signature with highest number of nodes into the smallest one. Among other things, this had to do with the wider specter of possibilities for node transformation instead of adding *fake* nodes.

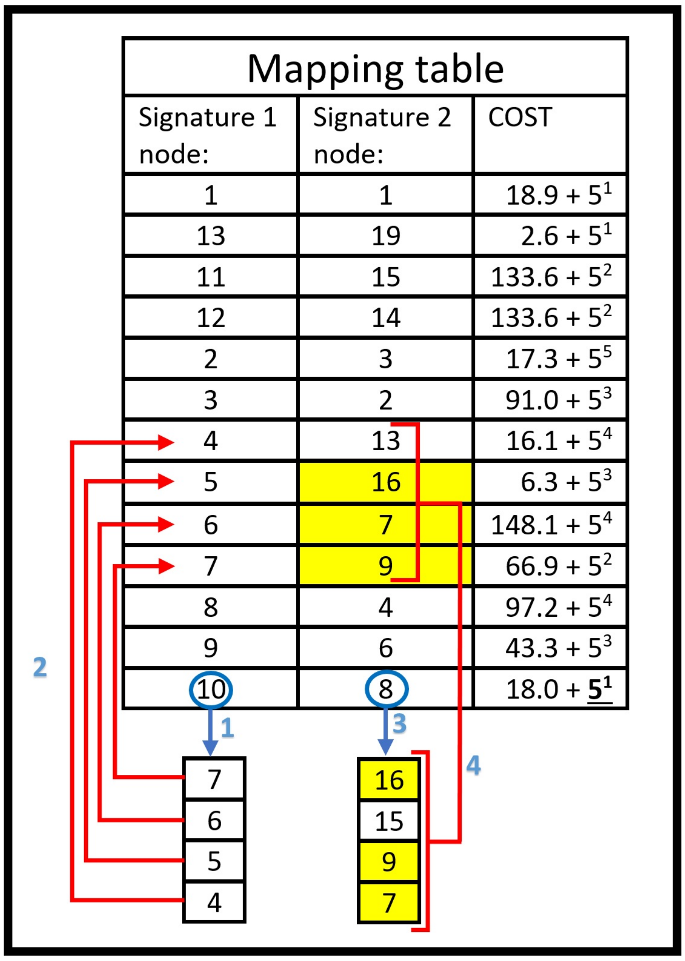
Even when we were using efficient techniques for backtracking in order to save processing time, most of the signatures contained so many nodes that one comparison could take days to calculate. For instance, a comparison between two signatures with 50 nodes each have $50! \approx 3.04 \times 10^{64}$ possible solutions. The backtracking algorithm excluded many irrelevant solutions, but it was still far from acceptable performance. To put this into a time perspective, the comparison between S1 and S2 with 13 and 19 nodes lasted for approximately 1 hour and 7 minutes. Some signatures in the dataset had over 100 nodes each after pre-processing, so we needed to find an increased calculation performance. After further analysis of the cost map (e.g table 3.3), we saw clearly that some node transaction costs were so high that it was not worth carrying in the calculations. We checked each and every reference node in order to see which probe transaction costs were below a certain threshold, which was set to 10 000.

S1	1	2	3	4	5	6	7	8	9	10	11	12	13
S2	1,19	2,3,4,5, 6,7,8,9, 12,13, 16	2,3,4,5, 6,7,8,9, 12,13, 16	2,3,4,5, 6,7,8,9, 12,13, 16	2,3,4,5, 6,7,8,9, 12,13, 16	2,3,4,5, 6,7,8,9, 12,13, 16	2,3,4,5, 6,7,8,9, 12,13, 16	2,3,4,5, 6,7,8,9, 12,13, 16	2,3,4,5, 6,7,8,9, 12,13, 16	2,3,4,5, 6,7,8,9, 12,13, 16	14,15, 17,18	14,15, 17,18	1,19

Table 3.5: Defining clusters between signature 1 and 2

While using S1 and S2 as example, we see from table 3.5 that S1 nodes are numbered at the first row, while the second row contains all S2 nodes that can be transformed with a cost below 10 000. With a closer look, we could see three *clusters* of nodes from S2. One cluster contained node 1 and 19. These could only be mapped into 1 and 13 in order to get the cost below 10 000. Another cluster was node 14, 15, 17 and 18. These were preferably mapped into node 11 and 12. The last cluster in this example contained all S2-nodes that could be mapped into node 2 - 10 in S1. By doing this partitioning of nodes, we could minimize the number of backtracking possibilities. Instead of having over 13! different combinations, we were now at approx 9!. This made a large impact on time efficiency, and shrinked the completed backtracking processing time for comparing S1 and S2 to below 10 seconds.

During backtracking, we were not only considering the transformation cost between nodes, but also their corresponding neighbors. Similar node neighbors meant that the signatures had more similar characteristics. This was one of the main weaknesses in the first estimation approach, since neighbor information was not taken into account. Information about neighbor nodes were already collected in the adjacency matrix, and we used this information during the mapping with backtracking.



Mapping table

Signature 1 node:	Signature 2 node:	COST
1	1	$18.9 + 5^1$
13	19	$2.6 + 5^1$
11	15	$133.6 + 5^2$
12	14	$133.6 + 5^2$
2	3	$17.3 + 5^5$
3	2	$91.0 + 5^3$
4	13	$16.1 + 5^4$
5	16	$6.3 + 5^3$
6	7	$148.1 + 5^4$
7	9	$66.9 + 5^2$
8	4	$97.2 + 5^4$
9	6	$43.3 + 5^3$
10	8	$18.0 + \underline{\mathbf{5^1}}$

Figure 3.9: Mapping table and cost added by neighbours between S1 and S2

Figure 3.9 shows the completed mapping table for comparison between S1 and S2. In the very early stages, it was harder to determine whether or not two mapping nodes had corresponding neighbours. This information became clearer for each completed mapping. The scenario in figure 3.9 considered the thirteenth and last mapping. We divided the process into four stages, marked with numbers in the figure:

1. Checks which nodes are neighbors with signature 1 node.

2. Checks which of the signature 1 neighbours are mapped earlier in the mapping table
3. Checks which nodes are neighbors with signature 2 node.
4. Counts the number of corresponding neighbors based on the previous mappings.

The example from figure 3.9 resulted in three shared neighbours. Neighbor costs were estimated by the formula

$$COST_{neighbours} = 5^{N_{total}-N}$$

where N_{total} is the number of neighbors and N is the number of shared neighbours. This cost was added upon the previously calculated cost of node transformation. Since only one neighbor was missing in this scenario, we added a cost of 5^1.

The final step in the comparison calculation added additional cost for removing nodes. The formula used for this purpose was

$$COST_{nodeDelete} = 2^{(|N_1-N_2|)/((N_1+N_2)/2)}$$

where N_1 and N_2 is the number of nodes of each signature. This calculation took into account how many nodes there were in total, so signatures with low number of nodes got a higher cost with the same difference than with high number of nodes.

Chapter 4

Experiment

This chapter explains the experiment setup in detail.

4.1 Experiment details

The research question will be answered by experimenting with various methods of comparing signatures by edge labels.

4.1.1 Choice of reference and probe samples

In order to compare signatures, we needed to define reference and probe samples in the dataset. It was important that probe samples were the large majority, but there needed also be enough references in order to fully represent the user. In the experiment starting phase of selecting probe/reference, the first 5 samples were picked as reference, while the remaining 20 became probe.

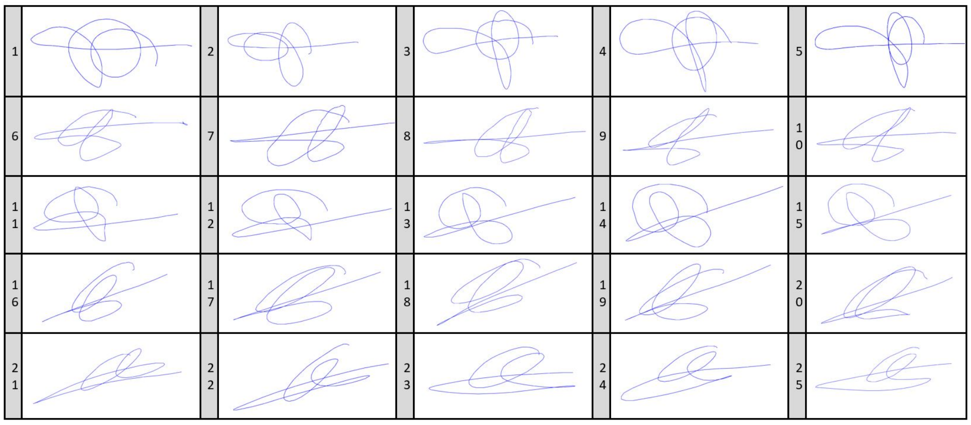


Figure 4.1: All 25 signatures of user-id 0021 ordered by naming convention in the MCYT-100 dataset

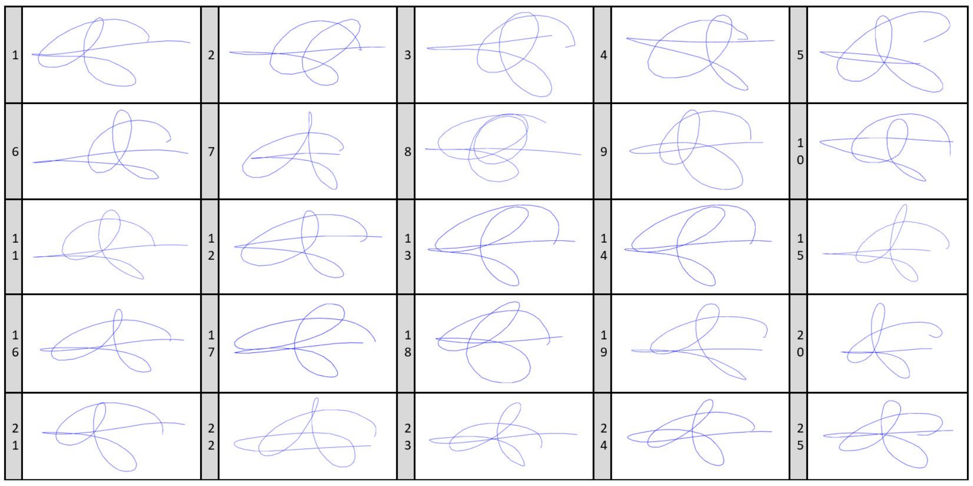


Figure 4.2: All 25 forgery signatures of user-id 0021 ordered by naming convention in the MCYT-100 dataset

As figure 4.1 shows, the person changed signature patterns slightly every 5 sample in the dataset. Signature 1-5 were fairly similar, as well as signature 6 - 11 and so on. This was because the signatures of every user in MCYT-100 were collected as sets of 5 samples [29]. We could not get a complete representation of the user by selecting a cluster of signatures close to each other as references. Therefore, every fifth sample from 1 to 25 (1,6,11,16 and 21) were defined as references for each and every individual. This method ensured that one reference signature were represented for each set of samples. Figure 4.2 shows all skilled forgery samples attached to that user. The forgery scenario was also divided into 5 sections, where 5 different users attempted to falsify the signature with 5 samples each [29].

4.1.2 Main experiment setup

The main experiment was divided into two different categories; hostile and friendly. The hostile scenario looked at the system accomplishment of separating legitimate samples from skilled forgeries. This was the most difficult task when it came to performance, but may also be the most relevant one. In the friendly scenario, we attempted to separate samples between genuine users. Since we wanted to compare three different edge labels (Pixel distance, Euclidean distance and Efficiency), the execution of each one of these scenarios were accomplished at least three times in order to complete a full experiment.

Hostile

This experiment compared the signature samples within the same user. Figure 4.3 illustrates how the calculation took place. There were defined five reference

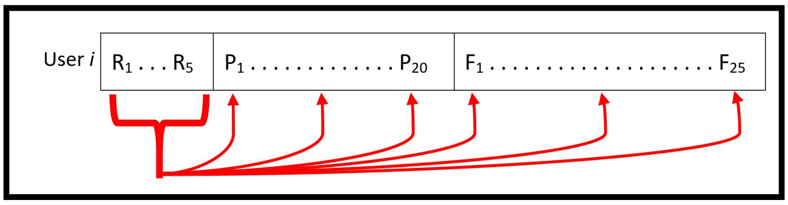

User *i*
R1 . . . R5
P1 . . . . P20
F1 . . . . F25

Figure 4.3: Illustration of probe/reference definition for forgery scenarios

signatures (R_1 - R_5). All remaining genuine samples (P_1 - P_{20}) as well as the 25 forgery samples (F_1 - F_{25}) were compared against each of the 5 references. We needed to repeat this for $i = 1..100$. A full execution of the whole dataset with 100 users gave a total number of 5 (number of reference signatures) x 45 (total number of probe samples, genuine and forgery together) x 100 (number of users) = 22 500 comparisons. This calculation was executed three times, one for each edge labeling method.

Friendly

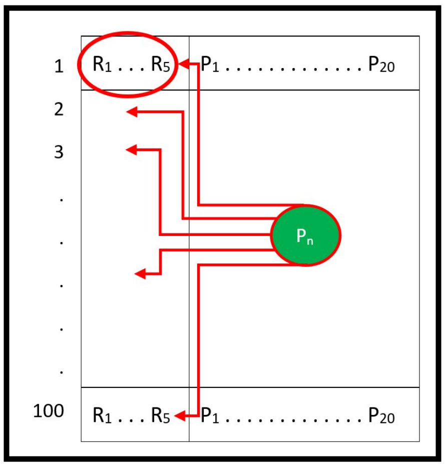

1
2
3
100
R1 . . . R5
P1 . . . . P20
Pn
R1 . . . R5
P1 . . . . P20

Figure 4.4: Illustration of probe/reference definition for friendly scenarios

The experiment of friendly scenario compared each and every genuine probe signature against all reference signatures in the dataset (figure 4.4). All forgery samples were excluded during this experiment. This gave 500 comparisons per probe signature (5x100). The number of probe signatures per user was 20, which gave a total number of probe signatures at 20 x 100 = 2 000. The total number of comparisons were 500 x 2 000 = 1 000 000 for each edge labeling method in the friendly scenario.

4.2 Experiment execution

The overall goal of the backtracking comparison script was to find the optimal mapping that gave the lowest possible cost. Unfortunately, this method proved to be inefficient with large graphs due to the high level of mapping possibilities. Even after pre-processing, many clusters were too large for using backtracking within a reasonable execution time.

Method	Edge labeling	User-ID	Probe-ID	Comparison 1	Comparison 2	Comparison 3	Comparison 4	Comparison 5
First estimation	Pixel	0021	3	3 701	208 680	145 890	508 120	408 470
Backtracking estimation	Pixel	0021	3	3 007	204 220	38 001	507 440	407 170
First estimation	Pixel	0021	26	314 390	203 310	106 440	380 470	510 590
Backtracking estimation	Pixel	0021	26	312 080	203 210	3 683	304 200	510 470

Table 4.1: Illustration of comparisons with different calculation methods

From table 4.1, we see comparisons between first estimation method and the backtracking method. Probe-ID 3 was a genuine sample, while probe-ID 26 was an imposter sample. These two probe-samples were compared against all 5 reference samples for user 21, referred to as comparison 1 - 5 in the table. By first looking at probe-ID 3, we saw relatively small differences between the methods in comparison 1, 2, 4 and 5. Some optimisations in path and/or neighbor compositions have been discovered by the backtracking algorithm in order to slightly decrease the cost values. The most significant difference between these methods appeared in comparison 3. In that comparison, the backtracking method found a node path that resulted in almost 4 times lower cost. The same situation occurred at probe-ID 26, but in that scenario the comparison 3 cost had decreased with almost 29 times with the backtracking method. These results gave valuable information for the further work in this study. In most scenarios, first estimation would give cost values slightly higher than the backtracking method. Nevertheless, there may occur cost estimations that are very different. Based on the comparisons from table 4.1, we concluded that the first estimation method would in most cases cause a slightly higher cost. Since we assumed that this tendency would be consistent, first estimation was satisfactory as the only estimation method for the largest experiments.

For the performance testing, we tested different *cost calculation* methods. Each probe sample had 5 different comparison values based on the 5 reference samples (as described in table 4.1). We tested the performance of:

- Mean cost of all 5 comparisons
- Lowest detected cost value
- Highest detected cost value
- Median cost of all 5 comparisons

We ran the entire dataset for both hostile and friendly scenarios with the use of first estimation due to its high efficiency. For backtracking, we needed to collect less complex signatures for testing because of processing time limitations. We verified the complexity both visually and by checking node size on the first sample from each user ID. The backtracking algorithm was executed on 26 collected user IDs on the hostile scenario. If the execution time of one comparison extended 15 minutes, the program returned without any calculations. This resulted in signatures with lack comparisons.

4.3 Performance values

From chapter 2.1.2 we introduced two methods of biometric performance testing; EER and TER. We are now going to explain them in more detail and describe how they were used in our results.

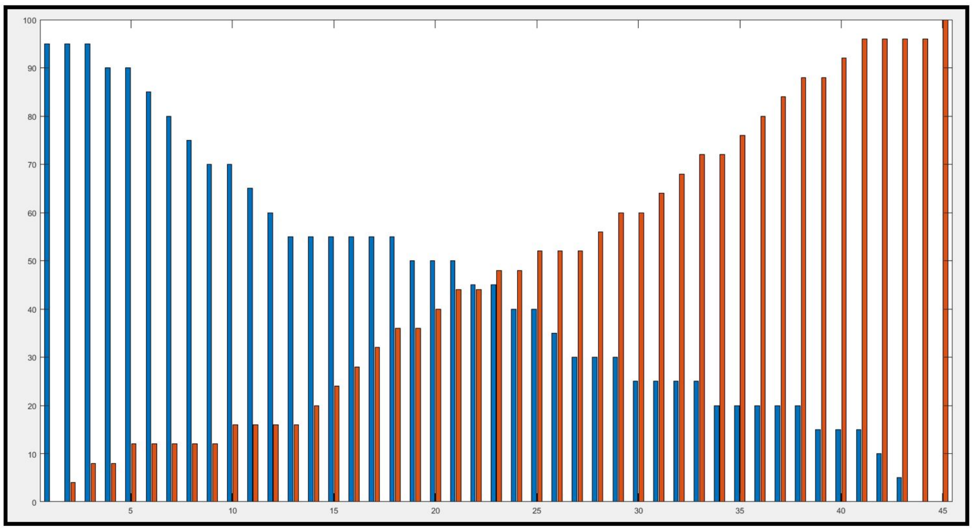


Figure 4.5: Performance illustration with user ID 0 from MCYT-100

The difference can be illustrated in figure 4.5, which is the performance of hostile scenario with the use of first estimation on user ID 0. The 45 different thresholds are selected from the lowest detected cost-values on each probe comparison with the five references. Blue bars represent FNMR, while FMR is represented by red bars. The vertical axis is the FMR/FNMR in percentage, and the horizontal axis illustrates the 45 different sample thresholds. From the definition of EER, we see that FMR and FNMR are closest at sample 22. This leaves and EER of approx 45%. It is harder to see the TER from this figure because we have to add the two bars together. The TER value can be found at sample 13 with an estimation of 80%, meaning that four out of five samples are either false match or false non-match. Because studies switches between different performance metrics, we need to find a way to easily compare TER and EER. Based on the differences, we say that $EER = TER \div 2$.

The hostile and friendly scenarios were slightly different in terms of the number of compared samples (section 4.1.2). Since there were relatively few samples for the hostile scenario (22 500), we used Total Error Rate (TER) in order to determine the best threshold. As already mentioned, friendly scenario included as much as 1 000 000 comparisons, and because of that we used EER as method for those testings.

We captured Standard deviation during the performance testing. This estimation evaluates the level of spreading in a dataset, and how close to the mean. One standard deviation represents 68,2% of the dataset, two standard deviations are 95,4%, three are 99,6% etc.

Chapter 5

Results

In this chapter, we will submit results from our implemented experiments with the use of MCYT-100 dataset. We will first look at hostile scenarios and afterwards friendly scenarios.

5.1 Hostile scenario results

We are now going to present all results from the hostile scenario. Since hostile results include both first estimation and backtracking approach, we will analyse the differences closely. First, we start off by analysing the full comparison of the first estimation approach. These results will be commented separately since we don't have directly comparable data. Each result is represented with a bar chart, simulating the different TER-scores on each user ID with the average TER scores and standard deviation noted below. The average TER scores for every experiment is also displayed as a red line in their respective diagrams.

5.1.1 First estimation for all 100 users

The pre-processing and comparison calculations for first estimation approach between two signatures were finished within half a second. Due to the quick computation, we were able to collect result for all 100 users during this test. From figure 5.1, we can see that the TER performances were very similar across the different methods. This applied to both edge distance labels and cost calculations. The results for the pixel distance as edge labeling method had a spread of only 0.84% between mean, median min and max, which is nearly irrelevant for the performance result. Euclidean distance had a spread of 1.49% and the efficiency method 2.38%. By comparing cost calculations between the three different edge labeling methods, the median approach seemed to perform best in all three scenarios. For the remaining, there were no clear pattern in which performed second best, third best and worst.

For standard deviation comparisons, we observed similarities within the cost calculation techniques across the edge labeling methods. The minimum cost calcu-

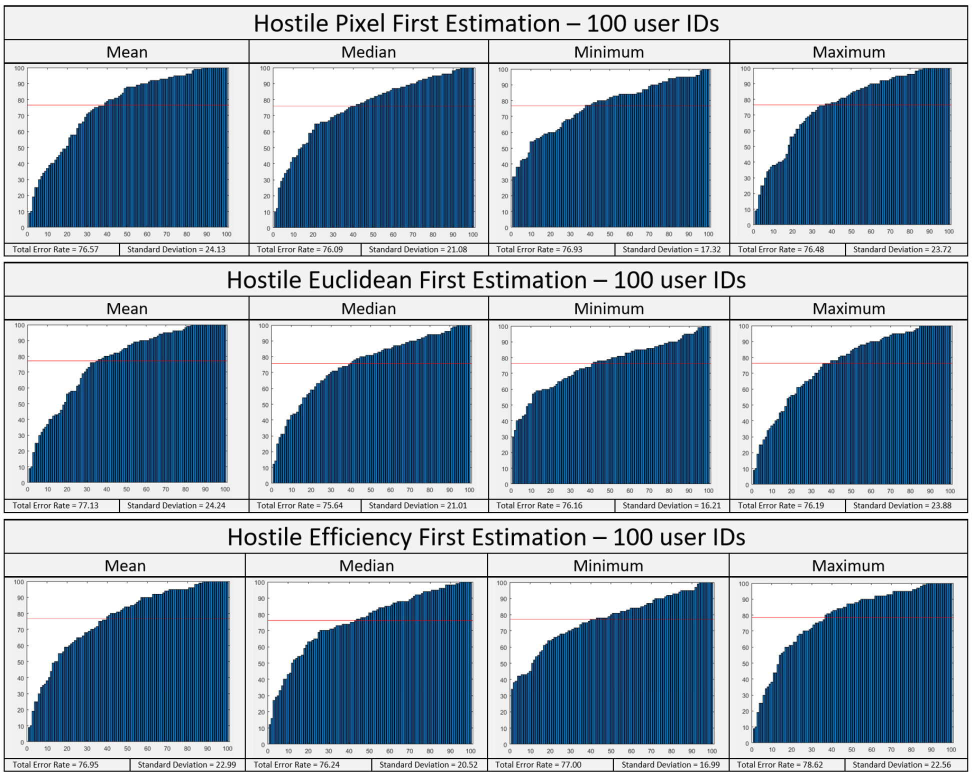


Figure 5.1: Hostile scenario for all 100 users with first estimation

lation had the most consistent TER throughout all users with a standard deviation of approx 17, while the remaining methods were slightly higher.

5.1.2 Backtracking vs First estimation

Chapter 4.2 explained why we used a subset of samples when we calculated the backtracking algorithm. We collected the following 26 user IDs from the MCYT-100 dataset:

0	3	4	7	9	14	15	16	21	22	26	28	31
36	39	47	48	57	63	68	69	73	76	82	85	88

Figure 5.2 illustrates all performance results from backtracking, which include both edge labeling and their appurtenant cost calculations. Overall, we discovered that edges labeled with efficiency performed best. This method combined with the median cost calculation technique gave the lowest captured TER-value of 72.91%. Furthermore, it was difficult to observe any clear patterns within the performances, especially not cost calculations between each labeling methods. The best performance for pixel distance was (like for efficiency) median, but that gave high TER for euclidean distance. Euclidean distance on the other hand achieved the best result with the minimum cost calculation.

The standard deviation scores were also very unpredictable in between the different approaches. The repeatedly lowest values were found in the minimum cost calculations. Signatures labeled by pixel distance seemed to fluctuate more between its user performances combined with the other two, and had the overall highest standard deviation values.

The reason for having 25 user IDs for backtracking with efficiency labeling (and not 26) was because of the few completed calculations. One signature was too complex for the backtracking algorithm, which lead to very few completed transformation cost calculations. Therefore, we had to exclude that signature.

First estimation results for similar user IDs as applied in the backtracking method are illustrated in figure 5.3. For the pixel - and euclidean distance, the TER-results were more consistent for similar cost calculations compared with backtracking. On the other hand, efficiency performed overall much worse for the first estimation. All 26 user IDs were utilized for efficiency labeling, compared to the 25 for backtracking. The median and minimum cost calculations, which gave best performance for backtracking, also showed best results for first estimation. On the other hand, the TER values were approx. 7% higher on first estimation.

For standard deviation, we observed lower values for the efficiency method. It is important to notice that standard deviation automatically will decrease when overall TER scores are becoming higher.

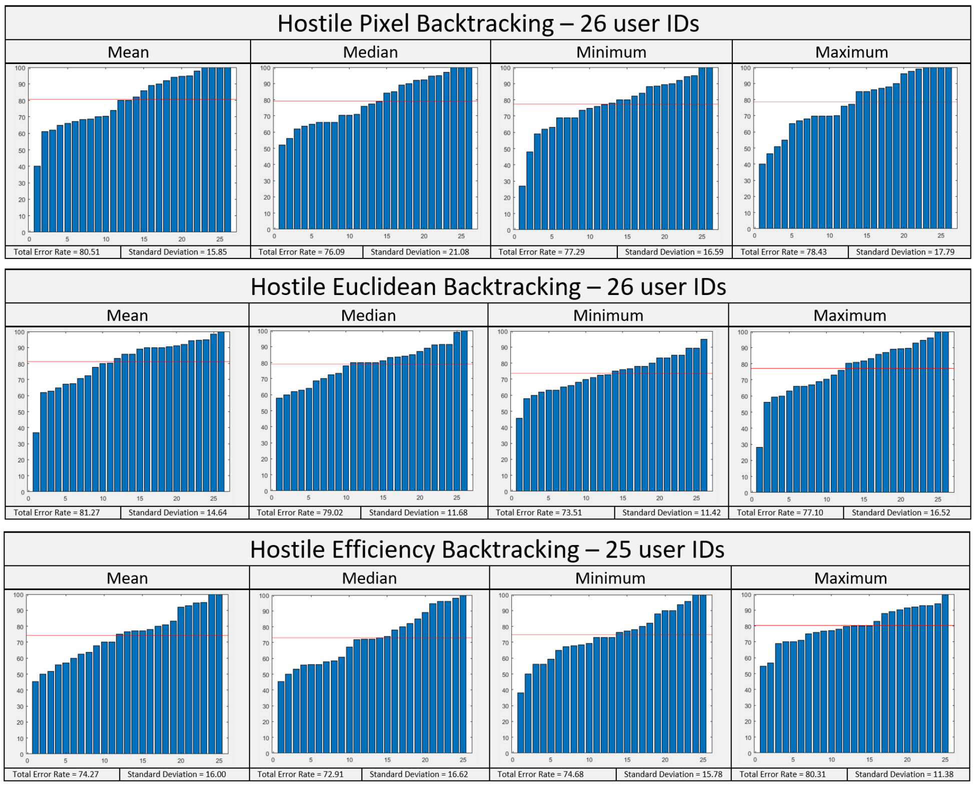


Figure 5.2: Different methods with the use of Backtracking calculation

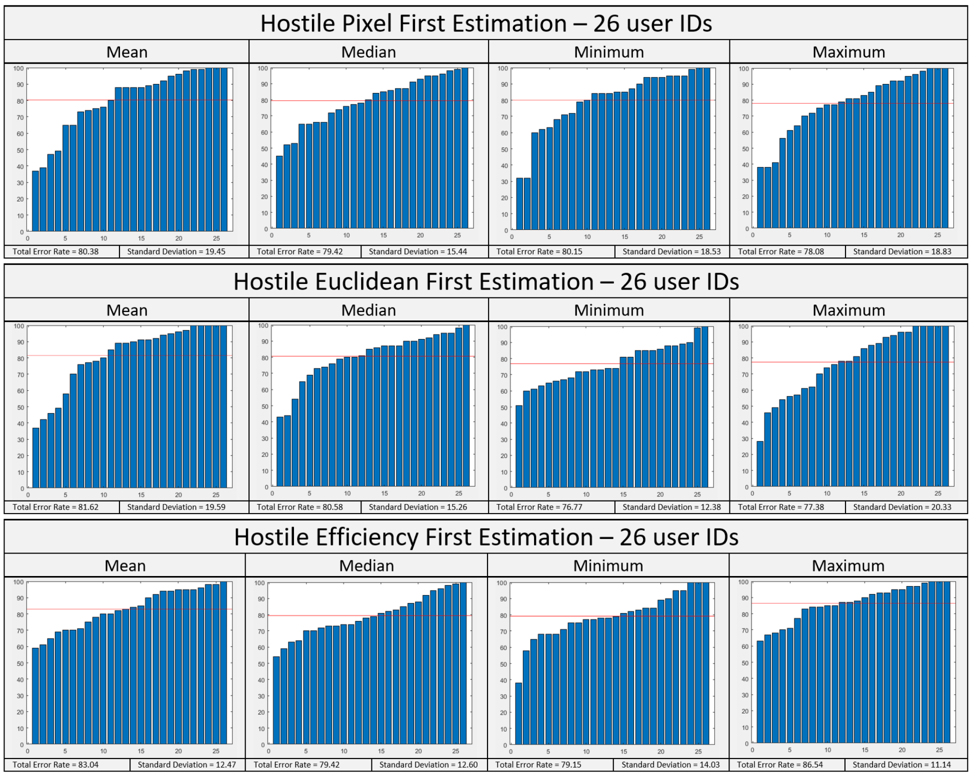
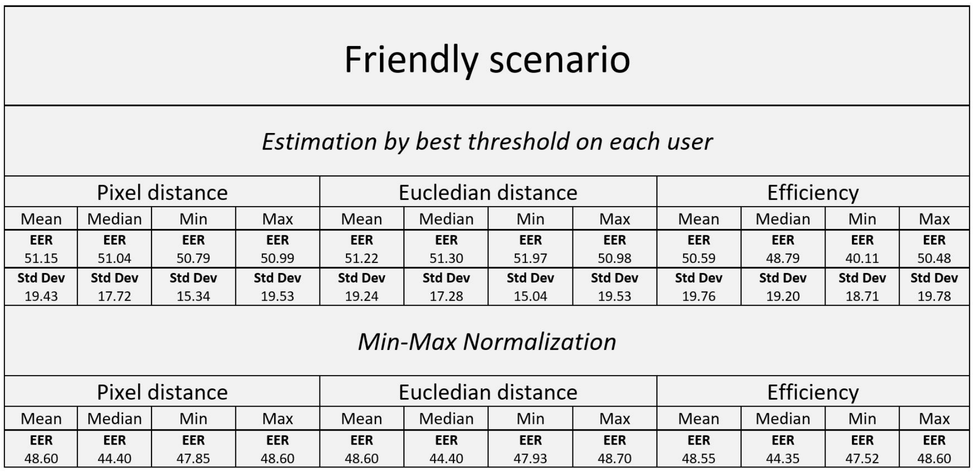


Figure 5.3: Different methods with the use of First estimation calculation

5.2 Friendly scenario results

We are now going to look closer into the performance of the friendly scenario. After the calculations of signature transformation costs, we discovered large differences between each user IDs in terms of lowest and highest cost scores. Signatures with more nodes/edges got overall higher costs because there were more transformation operations needed. Because of that, we needed to make the cost scores comparable across users. We used two techniques; estimation by best personal threshold on each user and min-max normalization. For the first approach we calculated EER performance values based on every user's best thresholds, where the final score was an average of these values. In the second approach, we compared all cost values directly between the users by using a normalization mechanism. Within the area of normalization techniques, there were two well known methods; z-normalization and min-max normalization. A major disadvantage with z-normalization in our scenario was that outlier values were smoothed out. That was not ideal because a lot of valuable information were placed there. Because of that, we chose the min-max normalization.

Table 5.1 shows an overview of the friendly scenario results. One clear observation was that the EER scores were very consistent within the similar cost calculation approaches. An example was min-max normalization between pixel distance mean, euclidean distance mean and efficiency mean scores. These edge labeling methods had almost identical values. The only outstanding exception from this pattern was the minimum cost calculation for efficiency labeling within the estimation by best threshold. Here we discovered an EER performance score of 40.11%, which was around 10% better than all comparable calculations for threshold and also the best overall score in our test. Nevertheless, we could not see that this performance resulted in any noticeable difference on standard deviation.



Friendly scenario											
Estimation by best threshold on each user											
Pixel distance				Eucledian distance				Efficiency			
Mean	Median	Min	Max	Mean	Median	Min	Max	Mean	Median	Min	Max
EER 51.15	**EER** 51.04	**EER** 50.79	**EER** 50.99	**EER** 51.22	**EER** 51.30	**EER** 51.97	**EER** 50.98	**EER** 50.59	**EER** 48.79	**EER** 40.11	**EER** 50.48
Std Dev 19.43	**Std Dev** 17.72	**Std Dev** 15.34	**Std Dev** 19.53	**Std Dev** 19.24	**Std Dev** 17.28	**Std Dev** 15.04	**Std Dev** 19.53	**Std Dev** 19.76	**Std Dev** 19.20	**Std Dev** 18.71	**Std Dev** 19.78
Min-Max Normalization											
Pixel distance				Eucledian distance				Efficiency			
Mean	Median	Min	Max	Mean	Median	Min	Max	Mean	Median	Min	Max
EER 48.60	**EER** 44.40	**EER** 47.85	**EER** 48.60	**EER** 48.60	**EER** 44.40	**EER** 47.93	**EER** 48.70	**EER** 48.55	**EER** 44.35	**EER** 47.52	**EER** 48.60

Table 5.1: Friendly results for all different methods

Chapter 6

Discussion

In this chapter we will analyse and discuss the results that were presented from chapter 5. We will both compare the different techniques used in this study and also look at results from other studies that have been using the MCYT-100 and MCYT-75 datasets. From now on, our implementation approach is referred to as *graph based distance (using backtracking/first estimation)*

6.1 Comparisons within own results

We will now analyse the different methods in order to see if there are patterns of better performances across test results. The performance values of the upcoming tables are average scores for every method. By doing this, we want to see if any of the implementations stands out.

6.1.1 Comparison method

This chapter looks at similarities and differences between the two comparison methods; backtracking and first estimation. The method comparison could only be executed for the hostile scenario. This is because friendly scenario used first estimation only, and we had no data from the backtracking method.

Comparison method	Hostile Average TER performance
First estimation for all users	76.67 %
Backtracking for selected users	77.12 %
First estimation for selected users	80.21 %

Table 6.1: Average scores for graph based distance with backtracking and first estimation

The average scores from table 6.1 did not give any clear performance difference between backtracking for selected users and first estimation for all users. As mentioned in section 4.2, signatures that were calculated by using backtracking were hand-picked in order to be smaller of size and complexity. This was to increase the execution speed and cause a feasible processing time. So even if the performance scores between backtracking for selected users and first estimation for all users gave fairly equal results, there were quite different quantum of data that was handled. For instance, backtracking did not calculate any large and complex signatures. It would have been interesting to see the difference for that scenario also.

We noticed over 3% worse performance on first estimation for selected users than the other two approaches. Since first estimation for all users performed better than the selected smaller ones, it looked like large and complex signatures were achieving slightly better performance scores for the first estimation method. This could potentially be the scenario for backtracking as well if we were able to calculate that. Maybe backtracking for all users would increase much more, or at least the 3% advantage like it had during this test. One reason for the increased performance may be that backtracking also considered node neighbors when calculating the final cost. This information may become more valuable for larger signatures with more existing nodes.

Based on scores from table 6.1 and the discussion in this section, we also assumed that the friendly scenario would perform slightly better by using the backtracking comparison method. Both performance scores were far from optimal, but we may see some potential in this method by adding additional label information about the edges as well.

6.1.2 Edge labeling

For edge labeling, we were interested to see performances for both hostile and friendly scenario. These values are illustrated in table 6.2 and 6.3.

Edge labeling	Hostile First estimation (all users) Average TER performance	Hostile Backtracking (selected users) Average TER performance	Hostile First estimation (selected users) Average TER performance
Pixel distance	76.52 %	78.08 %	79.51 %
Euclidean distance	76.28 %	77.73 %	79.09 %
Efficiency	77.20 %	75.54 %	82.04 %

Table 6.2: Comparison between the different edge labeling methods for hostile scenario

For the hostile scenario (table 6.2), we registered very consistent performances between pixel distance and euclidean distance within equal circumstances. The margin was so small that we could not consider which one of them that performed best. This pattern was also transferable to the friendly scenario (table 6.3). Our conclusion was that pixel distance and euclidean distance gave approximately equal results with the use of our cost estimation formula (section 3.3.1).

Edge labeling	Friendly Best threshold on each user Average EER performance	Friendly Min-Max Normalization Average EER performance
Pixel distance	50.99 %	47.36 %
Euclidean distance	51.37 %	47.41 %
Efficiency	47.49 %	47.26 %

Table 6.3: Comparison between the different edge labeling methods for friendly scenario with fist estimation on all users

The efficiency labeling did stand out with slightly better performance with backtracking for selected users (table 6.2) and for best threshold on each user (table 6.3). On the other hand, efficiency performed worse for hostile scenario with first estimation and selected users (table 6.2). The fact that efficiency performed significantly worse (4.84%) for the selected users compared with all users for hostile scenario, may again indicate that more complex signatures favoured this edge labeling method. The friendly scenario result should in theory give most reliable results because of its 1 000 000 signature comparisons on each edge labeling performance scores. In comparison, hostile first estimation for all users had 22 500 and backtracking/first estimation for selected users had 5 850 signature comparisons for each scores. This information emphasizes the performance of the efficiency approach in table 6.3.

There could be various reasons why efficiency labeling performed slightly different from the other two. Pixel distance and euclidean distance used the exact same cost distance formula, while efficiency used a different approach. This is because edge values were in between zero and one, and not related to the distance itself. Another interesting difference between these edge labeling approaches was the signature scaling. In the pre-processing part, we did not perform any type of scaling in order to ensure that the samples were treated as equal sized signatures. There would be scenarios where users had written their signatures in different sizes, which of course affected the edge label comparisons. The efficiency method differences from pixel - and euclidean distance because of its signature size independence. We were calculating with a ratio and not the distance itself.

Because of this, we could not conclude whether it was the edge labeling method that performed better for efficiency, the mathematical formula itself that fitted better for the specific calculation or the scaling advantage that separated these edge labeling methods. This can be investigated in further work.

6.1.3 Cost calculation methods

By comparing the different cost calculation approaches in table 6.4, we saw clearly from both hostile and friendly scenarios that the median and minimum scores did stand out with better performances. Minimum was slightly better for hostile scenarios, while median was 0.32% better for the friendly.

If we look at figure 5.2 once again, we did not see any clear patterns between mean, median, minimum and maximum for backtracking. The reason for this may

Cost calculation	Hostile Average TER performance	Friendly Average EER performance
Mean	79.08 %	49.79 %
Median	77.27 %	47.38 %
Min	76.72 %	47.70 %
Max	78.79 %	49.73 %

Table 6.4: Comparison between the different cost calculations

be caused by the lack of calculation costs between the probe sample and references. As described in Section 4.2, when the backtracking calculation processing time exceeded 15 minutes, the program cancelled that calculation, leaved behind no cost and moved forward. Fewer cost estimations within one probe sample may cause more varying results since there were overall fewer comparisons. For instance, if multiple probe-samples for a user-ID only got one or two completed transformation costs with the reference-samples, the calculations of mean/median/min/max would become less valuable.

6.1.4 Threshold vs Min-Max Normalization

Cost comparison	Friendly Average EER performance
Threshold	49.95 %
Min-Max Normalization	47.34 %

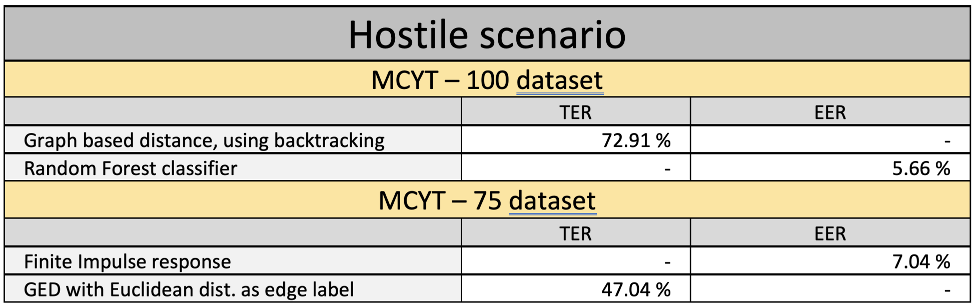
Table 6.5: Difference between the cost comparison methods

There was a noticeable gap between the average threshold approach and the Min-Max Normalization (table 6.5). One reason for this could be that we lost much information by taking the average threshold between each user ID instead of making the data comparable between all users as we did with Min-Max normalization.

6.2 Comparisons with other studies

One trustworthy way of making objective assumptions regarding system performance is to compare results with other studies that have used equal datasets. Other studies with the implementation of MCYT-100 dataset utilized all included information from the signatures (e.g. pen pressure and typing speed), while we were processing the shape only. Because of this, we had to keep in mind that not all features were used in this study. Therefore, we also compared performances with studies that used the MCYT-75 dataset. The sub-sections will be split between hostile scenario and friendly scenario, as well as comparison with the different types of datasets. We divided the TER by two while comparing the results with other studies that used EER (see section 4.3).

6.2.1 Hostile scenario



Hostile scenario		
MCYT – 100 dataset		
	TER	EER
Graph based distance, using backtracking	72.91 %	-
Random Forest classifier	-	5.66 %
MCYT – 75 dataset		
	TER	EER
Finite Impulse response	-	7.04 %
GED with Euclidean dist. as edge label	47.04 %	-

Table 6.6: Hostile scores [30][31][32]

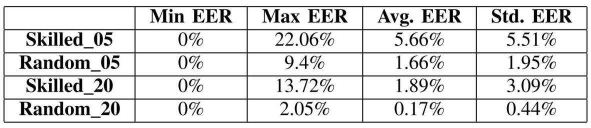
Table 6.6 summarizes performance scores for the hostile scenario. These will be discussed in detail below.

MCYT-100

The best performance of our study for hostile scenario was executed with the use of efficiency method and backtracking for median scores between probe and reference samples, illustrated in figure 5.3. This gave a TER of 72.91% (estimated EER of 36.46%). Overall, the results from backtracking became slightly better than the first estimation approach.

One study have used Random Forest classifier [30] for verification. They tested two approaches of training the algorithm with skilled forgeries (see table 6.7):

- 5 genuine samples from the specific user and one genuine sample from each of the other users. Average EER 5.66%.
- 20 genuine samples from the specific user and one genuine sample from each of the other users. Average EER 1.89%.



	Min EER	Max EER	Avg. EER	Std. EER
Skilled_05	0%	22.06%	5.66%	5.51%
Random_05	0%	9.4%	1.66%	1.95%
Skilled_20	0%	13.72%	1.89%	3.09%
Random_20	0%	2.05%	0.17%	0.44%

Table 6.7: Result with the use of Random Forest classifier [30]

[30] used all signature data information available from the MCYT-100 dataset, so the approaches may not be directly comparable. Nevertheless, the performance difference was very large. An EER (with 5 genuine samples) of 5.66% gave very accurate results and outperformed our experiment. One reason to this may be

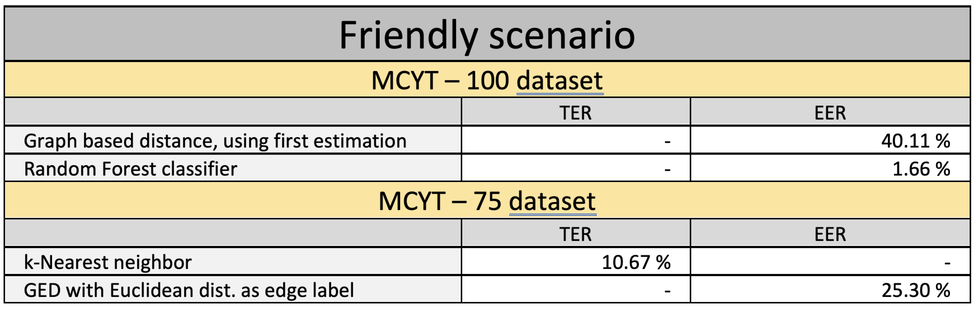
that the actual analysis of the signature shape was performing much better with a machine learning comparison approach. This claim may become more clear after our comparisons with the MCYT-75 dataset where no other factors are involved. Another theory on the very good performances from [30] is the utilization of online data. It may be very difficult for skilled forgery attacks to bypass online signature systems by knowing the signature shape only. This seems especially hard when an attacker cannot observe the typing process, which is the reality of MCYT-100 skilled forgery samples. Elements such as pen pressure and typing speed are very individual, and this information combined seems to be very powerful. If the attacker for instance attempts to imitate the typing speed, it will likely be at expense of the pattern accuracy. Nevertheless, in a majority of today's scenarios we are not in obsession of this data since people use pen and paper, and online capturing devices are not widely available. In later versions of smart phones, the touch-screens are capable of sensing features like finger pressure. This is possible to utilize for online signature capturing. On the other hand, verification on smartphones may be handled better by utilizing other already implemented biometric authenticators like fingerprint and face recognition.

MCYT-75

All system performances with the MCYT-75 dataset have a pre-processing part for skeletonization and feature extraction (section 2.2.1). This process will lead to some errors as explained and illustrated in section 3.2. Therefore, other system performances may have good signature comparison algorithms, but the pre-processing could give some errors in determining key-points of the signature. This error was irrelevant for our approach since MCYT-100 contained digital samples (section 3.2), so we have to keep in mind that pre-processing will benefit our implementation.

One study used a finite impulse response (FIR) system [31] to distinguish between genuine samples and skilled forgeries. The result gave an EER of 7.04%. Compared with our study (backtracking/median EER 36.46%), the system performance was significantly better for the FIR system. Therefore, we can conclude that graph based distance using backtracking don't give good results for the hostile scenario. From last chapter, we were not completely sure on how beneficial the online data was for the performance. Now we also know that previous studies have better performances based on the signature shape only.

Another study [32] also utilizes edge labels as signature representation. This study used only euclidean distance between nodes as distance metric. For signature comparison, [32] used the original GED implementation. The overall best performance was a TER of 47.4%, where the mean score for cost calculation were used.



Friendly scenario		
MCYT – 100 dataset		
	TER	EER
Graph based distance, using first estimation	-	40.11 %
Random Forest classifier	-	1.66 %
MCYT – 75 dataset		
	TER	EER
k-Nearest neighbor	10.67 %	-
GED with Euclidean dist. as edge label	-	25.30 %

Table 6.8: Friendly scores

6.2.2 Friendly scenario

Table 6.8 summarizes the performance scores for the friendly scenario. Now, we're going to take a look at these values.

MCYT-100

In [30], they tested the Random Forest classifier for random forgeries (table 6.7), which was comparable to our friendly scenario experiment. The average EER result was 1.66% with 5 reference samples. With the use of 20 reference samples from each user ID, the classifier could successfully separate the users with an average EER of 0.17%. One weakness with 20 reference samples for training was the few leftover probe samples. Since there were 25 legitimate samples from each user, only 5 could be used as probes. This reference/probe ratio was from our perspective skewed and gave artificial high performances. Therefore, we used the 5 reference performance score for comparison. The best performance from our friendly scenario was an EER of 40.11%. As mentioned in section 6.2.1, [30] used all online collected data. Nevertheless, it does not compensate for the big system performance difference.

MCYT-75

In [33], they tested three different machine learning algorithms for random forgery with the MCYT-75 database; Decision Tree J 48, Naive Bayes Tree and k-Nearest Neighbor. The performance values were presented slightly different from the known TER and EER performance exposition. After the algorithms were trained with the training samples, they performed "guesses" against the testing samples. Performances were stated in accuracy scores and error rates based on how many samples passes the threshold. The error rates would be equivalent to TER. By using the results from 10 users, 75% training and 25% testing, k-Nearest neighbor performed significantly best with a total error rate (TER) of 10.67%. This is much better than our best performance results with an EER of 40.11%. Nevertheless, our test applied 20% of the genuine samples as references, which gave less user information available in order to optimize the performance.

In [32], they calculated random forgeries with four different cost functions and four different cost calculations. The best performance for every cost function was by using the minimum cost calculation, where the lowest detected EER was 25.30%. As mentioned earlier, our study did not implement multiple cost functions because of long processing time for the backtracking algorithm. Our best single performance did also originate from the minimum cost calculation between reference/probe, but the average best scores were with the median estimation.

6.3 Performance evaluation and potentials

After comparing own results with other studies, it is clear that our method did not perform as good as the already existing approaches. So what could be the reason for this? One clear source of error is the edge transformation cost functions. These functions are fundamental in calculation of signature comparison costs. By giving either higher or lower costs for differences in edge values and nodes, it is highly likely that the performance will increase. It would be very interesting to try optimizing the cost functions in order to fully see the potential of this method. The disadvantage with backtracking through a cost matrix is the comprehensive processing of data. Since the calculations are very time consuming, these tests will need much computer power.

As mentioned in section 6.1.1, backtracking performs better than first estimation, and one factor is that neighbor nodes are included in the cost estimation. Since we're working on an online dataset with more available information about the signature, it would be possible to utilize the data as additional labels in our graph based distance approach. We could for instance include typing speed and pen pressure for the corresponding edges. Another interesting aspect is to gain even more information from the signature graph by also implementing nodes of degree 2, which could be change of direction.

Chapter 7

Conclusion

This thesis investigated the utilization of edge labels as characteristics for signature authentication. Signatures from the MCYT-100 dataset were transformed into graphs, and edges were defined in between nodes. Nodes were either of degree 1 (endpoints) or 4 (crossing of lines). We investigated three edge labeling methods; pixel distance, euclidean distance and efficiency. Efficiency is defined as a ratio where euclidean distance is divided by pixel distance. The graph comparison algorithm considered three main elements; length of the edges, number of nodes in the signature graph and the neighbor relationship between nodes. All edges from one signature were compared against all edges from the other one. This resulted in a cost matrix.

In order to find the lowest possible combination of edges through the matrix, we used backtracking. Since backtracking claimed very much computer processing power especially on the larger signatures, most of the results were based on first estimation calculation. First estimation found the lowest cost available through the cost matrix at first attempt, and did not consider other possible paths. This method gave generally higher costs, but the cost differences were consistently higher with a certain margin throughout the result calculations.

Overall, the efficiency edge labeling performed slightly best between the three approaches. One possible reason for this was that efficiency did not rely on signature normalization, compared with the other two edge labeling methods. Pixel distance and euclidean distance achieved almost identical results and was considered as equivalent methods in this study.

The hostile scenario experiment attempted to separate genuine samples from skilled forgeries. This is considered the most difficult task for a biometric system, but also the most relevant one. The efficiency edge labeling scored a TER of 75.54% on the backtracking approach for hostile scenario, while euclidean distance scored 77.73% and pixel distance scored 78.08%. Neither of these results were considered to be good in comparison to related studies. Since we used an online dataset with the utilization of the signature shapes only, we will now present the comparison with studies using the offline dataset MCYT-75. [31] used a finite impulse response (FIR) system, which resulted in an EER of 7.04%. In comparison, by converting TER into EER, our best performance was an EER of 37.77%.

For the friendly scenario, where genuine samples were separated from random forgeries, our best system performance gave an EER of 40.11% (with efficiency edge labeling). In comparison, [33] delivered an TER of 10.67% by using k-Nearest neighbor as machine learning technique which was a far more reliable system result.

The research question was: *Can a graph based approach with labeled edges be used for signature authentication?* Based on our implementation method, we can conclude that the performance is not satisfactorily for a biometric system. A TER of 75.54% for the hostile scenario will not justify the confidentiality of an individual's identity. On the other hand, we have only explored a few characteristics of the signature with this method. There may be other elements from the signature graph that in combination with the edge distance attribute would make signatures samples much more unique for each individual.

This lead into the further work from the study. Since our method has not been tested before, we discovered multiple areas of possible improvements. We defined nodes as either degree 1 (endpoint) or degree 4 (crossing line). One suggestion is to explore the implementation of degree 2 nodes (change of direction). By implementing more graph characteristic features, we may find even more individual uniquenesses in the shape of the signature. Since we used an online daseset (MCYT-100), there are many interesting aspects of utilizing online features in combination with the signature shape. For instance, edges could have multiple labels for even more uniqueness. Possible labels for further investigations are typing speed and pen pressure between nodes. This will on the other hand result in even more calculations for the backtracking algorithm because of larger graphs. Another aspect of implementation improvements from our study is signature normalization. The signature size is important for edge distance labeling because we are using physical distances. Variation of signature sizes will cause inaccurate results for edges labeled with pixel distance and euclidean distance.

Bibliography

[1] P. Maergner, K. Riesen, R. Ingold and A. Fischer, 'A structural approach to offline signature verification using graph edit distance,' vol. 01, pp. 1216–1222, 2017.

[2] J. Kizza, 'Authentication,' in *Guide to Computer Network Security*. Cham: Springer International Publishing, 2020, pp. 207–225, ISBN: 978-3-030-38141-7. DOI: `10.1007/978-3-030-38141-7_10`.

[3] D. Pipkin, *Information Security: Protecting the Global Enterprise*, ser. Hewlett-Packard professional books. Prentice Hall PTR, 2000, ISBN: 9780130173232.

[4] A. Jain, A. Ross and K. Nandakumar, *Introduction to Biometrics*, ser. SpringerLink : Bücher. Springer US, 2011, ISBN: 9780387773261. [Online]. Available: `https://link.springer.com/book/10.1007/978-0-387-77326-1`.

[5] C. Busch, P. Bours and G. Li, *Biometric performance*, University Lecture, 2020.

[6] H. Sharma, A. Alekseychuk, P. Leskovsky, O. Hellwich, R. Anand, N. Zerbe and P. Hufnagl, 'Determining similarity in histological images using graph-theoretic description and matching methods for content-based image retrieval in medical diagnostics,' *Diagnostic pathology*, vol. 7, p. 134, Oct. 2012. DOI: `10.1186/1746-1596-7-134`.

[7] K. Riesen and H. Bunke, 'Approximate graph edit distance computation by means of bipartite graph matching,' *Image and Vision Computing*, vol. 27, no. 7, pp. 950–959, 2009, 7th IAPR-TC15 Workshop on Graph-based Representations (GbR 2007), ISSN: 0262-8856. DOI: `https://doi.org/10.1016/j.imavis.2008.04.004`.

[8] N. Tsapanos, I. Pitas and N. Nikolaidis, 'Graph representations using adjacency matrix transforms for clustering,' in *2012 16th IEEE Mediterranean Electrotechnical Conference*, 2012, pp. 383–386. DOI: `10.1109/MELCON.2012.6196454`.

[9] *Adjacency matrix*, en, Page Version ID: 1048154108, Oct. 2021. [Online]. Available: `https://en.wikipedia.org/w/index.php?title=Adjacency_matrix&oldid=1048154108` (visited on 07/12/2021).

[10] A. Marzal and E. Vidal, 'Computation of normalized edit distance and applications,' *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 9, pp. 926–932, 1993.

[11] V. Levenshtein, 'Binary Codes Capable of Correcting Deletions, Insertions and Reversals,' *Soviet Physics Doklady*, vol. 10, p. 707, 1966.

[12] S. Schimke, C. Vielhauer and J. Dittmann, 'Using adapted levenshtein distance for on-line signature authentication,' in *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, vol. 2, 2004, 931–934 Vol.2.

[13] S. Bougleux, B. Gaüzère and L. Brun, 'Graph edit distance as a quadratic program,' pp. 1701–1706, 2016.

[14] X. Gao, B. Xiao, D. Tao, L. Dacheng and Xuelong, 'A survey of graph edit distance,' *Pattern Anal. Appl.*, vol. 13, pp. 113–129, Feb. 2010. DOI: `10.1007/s10044-008-0141-y`.

[15] M. Neuhaus and H. Bunke, 'Self-organizing maps for learning the edit costs in graph matching,' *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 35, no. 3, pp. 503–514, 2005. DOI: `10.1109/TSMCB.2005.846635`.

[16] S. Güldal, V. Baugh and S. Allehaibi, 'N-queens solving algorithm by sets and backtracking,' in *SoutheastCon 2016*, 2016, pp. 1–8. DOI: `10.1109/SECON.2016.7506688`.

[17] Anonymous, *Write a backtracking algorithm for the n-queens problem*, Jan. 2018. [Online]. Available: `https://www.transtutors.com/questions/1-write-a-backtracking-algorithm-for-the-n-queens-problem-that-uses-a-version-of-pro-2501435.htm`.

[18] H. Priestley and M. Ward, 'A multipurpose backtracking algorithm,' *Journal of Symbolic Computation*, vol. 18, pp. 1–40, Jul. 1994. DOI: `10.1006/jsco.1994.1035`.

[19] S. Ahmed, A. Ramasamy, A. Khairuddin and J. Omar, 'Automatic online signature verification: A prototype using neural networks,' in *TENCON 2009 - 2009 IEEE Region 10 Conference*, 2009, pp. 1–4.

[20] V. Bharadi and H. Kekre, 'Off-line signature recognition systems,' *International Journal of Computer Applications*, vol. 1, Feb. 2010. DOI: `10.5120/499-815`.

[21] H. Kekre and V. Bharadi, 'Specialized global features for off-line signature recognition,' Jul. 2019.

[22] D. Huang and C. Shan and M. Ardabilian and Y. Wang and L. Chen, 'Local binary patterns and its application to facial image analysis: A survey,' *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 41, no. 6, pp. 765–781, 2011.

[23] S. Armand, M. Blumenstein and V. Muthukkumarasamy, 'Off-line signature verification based on the modified direction feature,' vol. 4, Jan. 2006, pp. 509–512. DOI: `10.1109/ICPR.2006.893`.

[24] H. Baltzakis and N. Papamarkos, 'A new signature verification technique based on a two-stage neural network classifier,' *Engineering Applications of Artificial Intelligence*, vol. 14, no. 1, pp. 95–103, 2001, ISSN: 0952-1976. DOI: `https://doi.org/10.1016/S0952-1976(00)00064-6`.

[25] L. Greche, M. Jazouli, N. Es-Sbai, A. Majda and A. Zarghili, 'Comparison between euclidean and manhattan distance measure for facial expressions classification,' Apr. 2017, pp. 1–4. DOI: `10.1109/WITS.2017.7934618`.

[26] J. Ortega-Garcia, J. Fierrez-Aguilar, D. Simon, J. Gonzalez, M. Faundez-Zanuy, V. Espinosa, A. Satue, I. Hernaez, J. Igarza, C. Vivaracho, D. Escudero and Q. Moro, 'Mcyt baseline corpus: A bimodal biometric database,' *IEE Proceedings - Vision, Image and Signal Processing*, vol. 150, no. 6, pp. 395–401, 2003. DOI: `10.1049/ip-vis:20031078`.

[27] J. Ortega-Garcia, J. Fierrez, D. Simon, J. Gonzalez, M. Faundez-Zanuy, V. Espinosa, A. Satue, A. I. Hernáez, J. Igarza, C. Vivaracho-Pascual, D. Escudero and Q. Moro-Sancho, 'Mcyt baseline corpus: A bimodal biometric database. iee proc vis image signal process spec issue biom internet,' *IEE Proceedings - Vision Image and Signal Processing*, pp. 395–401, Dec. 2003. DOI: `10.1049/ip-vis:20031078`.

[28] J. Bresenham, 'Algorithm for computer control of a digital plotter,' *IBM Systems Journal*, vol. 4, no. 1, pp. 25–30, 1965. DOI: `10.1147/sj.41.0025`.

[29] J. Ortega-Garcia, J. Fierrez, J. Rello and J. Gonzalez-Rodriguez, 'Complete signal modeling and score normalization for function-based dynamic signature verification,' vol. 2688, Jun. 2003, pp. 658–667, ISBN: 978-3-540-40302-9. DOI: `10.1007/3-540-44887-X_77`.

[30] S. Dutta, R. Saini, P. Kumar and P. Roy, 'An efficient approach for recognition and verification of on-line signatures using pso,' in *2017 4th IAPR Asian Conference on Pattern Recognition (ACPR)*, 2017, pp. 882–887. DOI: `10.1109/ACPR.2017.115`.

[31] P. Thumwarin, J. Pernwong, N. Wakayaphattaramanus and T. Matsuura, 'On-line signature verification based on fir system characterizing velocity and direction change of barycenter trajectory,' in *2010 IEEE International Conference on Progress in Informatics and Computing*, vol. 1, 2010, pp. 30–34. DOI: `10.1109/PIC.2010.5687959`.

[32] D. Do, 'Offline signature verification using ged on labelled graphs,' Norwegian University of Science and Technology, Jun. 2021.

[33] A. Shah, M. Khan, F. Subhan and M. Fayaz, ‘An offline signature verification technique using pixels intensity levels,’ *International Journal of Signal Processing, Image Processing and Pattern Recognition*, vol. 9, pp. 205–222, Aug. 2016. DOI: `10.14257/ijsip.2016.9.8.18`.