Heidelberg University
Interdisciplinary Center for Scientific Computing (IWR)
Forensic Computational Geometry Laboratory (FCGL)

Bachelor-Thesis

Part-Structured Spotting of Cuneiform Characters

Maximilian Klingmann

supervised by
Dr. Hubert Mara

Matriculation Number: 2932908
Date of Submission: 07/04/2016

Declaration of Authority

I, **Maximilian Klingmann**, hereby declare that this bachelor thesis, titled **Part-Structured Spotting of Cuneiform Characters**, written at the **Forensic Computational Geometry Laboratory (FCGL)** and supervised by **Dr. Hubert Mara**, and all material presented were created by myself entirely. Furthermore I declare that all adaptions from other sources are specifically acknowledged throughout the thesis. Citation, as well as the usage of foreign sources, texts or any other aid are denoted according strict scientific rules. I understand that I must not present foreign texts or text passages as work of my own. Doing so is seen as cheating and violates basic rules of scientific work. I am aware that cheating would lead to a withdrawal of the examination among other consequences. I declare that all statements and information contained herein are true, correct and accurate to the best of my knowledge and belief.

Heidelberg, 05/04/2016 _____

Danksagung

An erster Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Bachelor-Arbeit unterstützt und motiviert haben.

Ein besonderer Dank hierbei geht vor allem an meine Eltern, die mich stets im Laufe meines Studiums und auch schon zuvor motiviert haben meine Ziele zu verfolgen und zu verwirklichen.

Weiterhin bedanke ich mich sehr bei Bartosz Bogacz und Dr. Hubert Mara, die meine Arbeit und somit auch mich durch kritisches Hinterfragen und hilfreiche, als auch wertvolle Hinweise betreut und unterstützt haben.

Ferner danke ich auch Herrn Dr. des. Kamran Vincent Zand für die grundlegende Einführung in die Keilschrift und Assyriologie und die empfohlene Hintergrundliteratur zur Vertiefung meines Wissens.

Daneben gilt mein Dank auch meinen Vorgesetzten und Kollegen bei *xmachina*, welche mir selbst bei sehr kurzfristig abgesagten Arbeitstagen die Möglichkeit gaben mehr Zeit in meine Bachelor-Arbeit zu investieren.

Nicht zuletzt danke ich Anja Pittelkow und Cathrin Augustat für die Hilfe bei Übersetzungsschwierigkeiten und für die Zeit, welche sie in Korrektur und Verbessung der vorliegenden Arbeit investiert haben.

Abstract

With an estimated amount of more than 500,000 recovered cuneiform tablets worldwide the transliteration and translation of these tablets done by assyriologists to date takes a long time, since all translations are done manually using character lists. In cuneiform script some symbols also have various meanings, which adds even another level of complexity to the translation of tablets.

Researchers in image processing have found methods to extract specific symbols from a set of symbols, such as characters from handwritten texts. In this thesis one of these methods is tested and evaluated against cuneiform tablets. This is done to find automatic solutions for the manual translation of the tablets. The method bases on the idea of supplying a cuneiform symbol to find on a cuneiform tablet also supplied both being rasterized images. Distance Transforms of the tablet are created and summed up with various offsets multiple times after a tree model is derived from the symbol's shape. The offsets of the summations follow the relative offsets of the nodes within the tree model resulting in a scalar field. The local minima of the scalar field depict congruity between the used symbol and a symbol found on the target image at the minima's location.

The method was tested using synthetic data in form of vector graphics in their original form and with the addition of synthetic noise (*Salt & Pepper*), which need to be rasterized in a pre-processing step and a tablet taken from the biggest online collection for recovered cuneiform tablets. The results show a high to medium *Precision* with increasing *Recall* and even though speed was not a focus of this work the results were received within a few seconds.

Zusammenfassung

Mit einer geschätzten Anzahl von mehr als 500.000 geborgenen Keilschrifttafeln weltweit, nimmt die Übersetzung von ebendiesen Tafeln durch Assyriologen heutzutage sehr viel Zeit in Anspruch, da die Tafeln auch heute noch von Hand, mit der Hilfe von sogenannten Zeichentafeln übersetzt werden müssen. Ein Symbol der Keilschrift kann hierbei auch mehreren Bedeutungen und Interpretationen folgen, was die Komplexität von Übersetzungen noch weiteranhebt.

Forschungen in der Bildverarbeitung haben Methoden und Algorithmen entwickelt, welche es erlauben Symbole aus Texten zu extrahieren. Ein Beispiel hierfür wäre die Extrahierung von Buchstaben aus handgeschriebenen Briefen. In dieser Bachelorarbeit wird eine dieser Methoden auf Keilschrifttafeln angewandt, getestet und anschließend evaluiert. Dies soll künftig Assyriologen Hilfestellung bei der Übersetzung der Keilschrifttafeln geben. Die Methode basiert darauf ein Symbol in Keilschrift auf einer Tafel zu suchen. Hierfür wird der Methode dieses Symbol bekanntgegeben, sowie auch die Tafel, welche durchsucht werden soll. Beides, das Symbol und die Tafel müssen hierbei in rasterisierter Form vorliegen. Für die vorliegende Tafel wird eine Distanztransformierte berechnet und mehrere Male mit verschiedenen Verschiebungen aufeinander addiert. Zuvor wird auf Basis der Form des Symbols ein Baum aufgespannt, welcher als Grundlage für die Verschiebungen der Summationen gilt. Die Verschiebungen entsprechen hierbei den Verschiebungen zwischen Eltern-Kind Beziehungen im Baum. Die lokalen Minima des resultierenden Skalarfeld zeigen eine hohe Kongruenz zwischen dem gewünschten Symbol und einem Symbol auf der Keilschrifttafel wieder.

Die Methode wurde gegen zweierlei Tafeln getestet. Zum Einen wurden Tafeln synthetischer Natur verwendet. Hierbei handelt es sich um Vektorgrafiken, welche zunächst in einem Pre-Processing Schritt rasterisiert werden

müssen. Diese Tafeln wurden einmal ohne Veränderung und einmal nach Hinzufügen von synthetischem Rauschen in Form von *Salt & Pepper*-Rauschen getestet. Weiterhin wurde eine Tafel aus der größten Online Sammlung für Keilschrifttafeln für die Tests verwendet. Die Ergebnisse zeigen ein hohes bis mittelmäßiges Maß an *Precision* bei ansteigendem *Recall* und obwohl Geschwindigkeit und Effizient des verwendeten Algorithmus nicht im Vordergrund dieser Arbeit standen wurden die Ergebnisse in wenigen Sekunden generiert.

Contents

1 Introduction

In this first chapter the bachelor thesis is motivated and previous related work is shown as well as the terminology used throughout the thesis is described.

1.1 Motivation

To preserve correct information and avoid the problems arising from oral transmission script writing systems were created about 6600 B.C. Apart from the Latin font, which is currently used in most western civilizations, there have been several other fonts through the history of mankind, which themselves have been divided in hundreds of thousands of dialects. Written texts were mostly used for economical purposes [1], such as a font called cuneiform (from the Latin word for wedge, *cuneus*).

This font was used from 3000 B.C. to 100 A.D. in the area of mesopotamia, nowadays known as the middle east. Cuneiform tablets found by archaeologists are subject to scientific research. About 500.000 tablets have been found and registered all over the world. A database of these tablets can be found at the *Cuneiform Digital Library Initiative* (CDLI), which tries to collect all cuneiform tablets found, listing them for scientific research. A cuneiform symbol is a combination of multiple wedges, which were imprinted into clay with rectangular stylus, thus cuneiform script is originally found in 3D form. The wedges got redrawn in a "Y"-shape, which in return get grouped to form various symbols.

Assyriologists distinguish two main dialects of the cuneiform font. First the *Sumerian* language used from 3300 B.C to 2000 B.C and second the *Akkadian* language used from 2000 B.C. to 100 A.D.. Although these are the main dialects there are many regional dialects or dialects changing throughout time comparable to modern languages such as German. Comparing German writings of today with letters written in *Sütterlin*, a historical form of German handwriting, one can clearly see the evolution of the style of the font. Although both are using the Latin font system, the letters differ in handwriting and calligraphy among other differences. The main difference between *Sumerian* and *Akkadian* is the meaning of the symbols used in both dialects. While *Sumerian* is ideographic, akkadian uses syllabic symbols. An ideograph/ideogram is a graphic symbol which resembles a picture that is used to illustrate a specific abstract meaning or idea, e.g. "Not-Allowed"-sign with red bars on a white background. A syllabic symbol (*Syllabogram*) resembles a syllable or a mora of a word. *Syllaborams* are used in phonetic transcription of modern languages. e.g. /fəˈnɛtɪks/ for "phonetics" The font itself does not only diverge in the meaning of a symbol but also in the visible representation of a symbol. The representation depends on individual handwriting like letters do nowadays. No letter in the Latin alphabet is written exactly the same by two different people regarding form and size. One can in fact find regional similarities of symbols, so it is possible to determine where a tablet originates from with a certain probability, but there were no unified versions of the symbols. Therefore cuneiform symbols do have different forms, appearances and meanings depending on the regional location and the time they were used in. The meaning of a symbol can vary immensely. On the one hand a symbol itself

can have a meaning by itself (*Logogramms*) comparable to cave drawings of the stone age or numeric symbols in modern languages. On the other hand a symbol can represent a syllable of a word (*Syllabogramms*) or even be part of a collection of words which can only be read together (*Determinatives*). However one must not conclude that two different symbols have similar meanings only because the appearance is similar. Once again this is comparable to modern languages. The letters $\mathcal{C}$, $\mathcal{G}$ and the number *6* are visually similar, but their meaning is significantly different.

One of the main tasks assyriologists work on is to give each and every symbol of cuneiform its meaning and interpret a symbol for its value and function within the language. However translation and transliteration take a long time and effort because of the diverse and complex nature of cuneiform style. The fact that even in *Akkadian* there are uses of *Sumerian* symbols impede that task even more, since one can not rely on the syllabic meaning of a symbol but one also has to take ideographic meanings into account. A symbol must be interpreted in the context of the surrounding symbols to get a correct translation. Assyriologists have agreed on creating character lists showing all possible forms and appearances for a symbol and assigning it with syllabic and ideographic meanings. One of the most famously used character lists was created by Borger in 1978 [2]. There are about 900 different symbols known to date. However most of the dialects use a much smaller number of approximately 100 symbols. Old-Babylonian *Akkadian* for example uses about 110 different symbols. The creation of these character lists is mainly done manually and handwritten, which again takes much time to do. [1]

— Freely translated from German

The manual translation renders it difficult to obtain real Ground Truth of any tablets needed for testing various methods to ease the translation. The research field of word-spotting tries to create automated solutions to find objects like words or letters within a given text. One major application for this word-spotting techniques are devices using *Optical Character Recognition* (OCR) [3].

Other approaches try to aim for one-shot word-spotting where only one character is given to a method which scans for the character throughout a set of characters or words. To minimize the time spent with translating cuneiform tablets manually or creating handwritten character lists these algorithms are evaluated and tested with found tablets.

1.2 Related Work

The problems stated before of manually creating word lists are the starting point for this bachelor-thesis. The aim was to connect the state-of-the-art with an algorithm for part-structured keyword spotting. The algorithm was described by Howe et al. [4] and was tested with handwritten documents by George Washington. The goal of this bachelor-thesis is to apply the algorithm to cuneiform tablets and evaluate whether future research should focus on improve on the results this algorithm creates or whether future research in general is not advisable using this algorithm for cuneiform tablets.

Furthermore this thesis is supposed to be of help for future research, which tries to generate the character list mentioned above automatically or even translates a cuneiform tablet entirely on its own. Aside Howe's approach several other paper have been released describing different approaches to automatically spot words in texts of different languages. Frinken et al. [5] described a method to spot words using neural networks. They altered an algorithm described by Graves et al. [6] called the *Connectionist Temporal Classification (CTC) Token Passing* algorithm and applied the technique to find words on handwritten transcripts . One key-feature described by Howe's algorithm was the ability to have a one-shot learning approach, meaning only one query image is to be given instead of having to create big training sets and use machine learning.

Fei-Fei, Fergus & Perona [7] described an approach of one-shot learning for different image resources, although their work was not aimed to give a solution for word spotting. The idea was used by Howe et al. in their paper. One can see clearly, that recognition of handwritten material has been subject to research for several decades now, since Revow, Williams and Hinton [8] described a method to recognize handwritten digits back in 1996.

Their generative models used what they called B-splines with Gaussian *Ink Generators*. Most of the sources mentioned above describing text recognition

methods or algorithms do so based on the English language or at least the Latin alphabet. Mara, Bogacz and Gertz [9] described a method to spot words using cuneiform tablets using graph similarity and graph isomorphisms. In previous work Mara [10] also described how to extract *Scalar Vector Graphics* (SVG) from 3D-models of cuneiform tablets. Rothacker et.al [11] published a method to retrieve cuneiform structures on scanned tablets, however this thesis is based on the method described by Howe et al..

1.3 Terminology

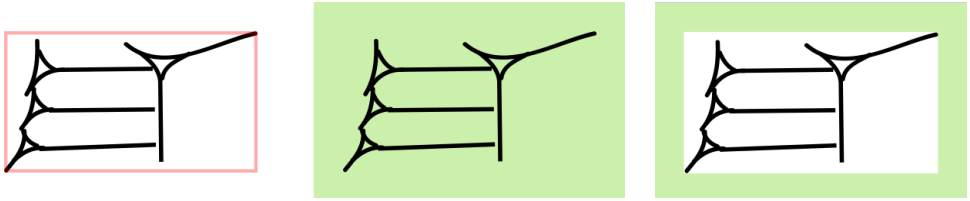
Throughout the thesis there are several terms repeatedly used which might need to be explained beforehand due to their nature of use. Most of these terms are used in computer graphics and image processing but others are defined here to illustrate what they are. A list of acronyms can be found at the very end of this thesis.

Bounding-Box A bounding box describes a rectangle parallel to the axis, which surrounds a much more complex geometrical shape. It serves as a border for the surrounding object and is often much more simple than the original object. Bounding Boxes are often used to calculate collision intersections, since the calculation of collision with simple objects is much faster and takes much less time than calculating collision with highly complex shapes. An example for a bounding box is illustrated in 1.1a.

Query Image vs. Query Symbol This is the image of the query to scan the tablet for. It consists of the background as well as the symbol itself. The query symbol however is the actual symbol the method scans the tablet for. It only consists of the lines shaping the symbol without the background or a rectangular box surrounding it.

Alpha Values In binary images it is possible to save an extra value aside the RGB (red, green, blue) values called the alpha value, where intermediate values correspond to semi-transparency. 1.1b and 1.1c show a query with active and inactive alpha values.

Heightmap & Distance Transform A heightmap is a derived representation of a map or image to store values. One prominent use is on a topographical card in an atlas, where often times green spaces show low elevation, whereas yellow, brown and white areas show high elevations, such



(a) (b) (c)

Figure 1.1: Cuneiform symbol (a) surrounded by a red bounding box surrounding, on a green background (b) with high alpha values and (c) without low alpha values.

as mountains or hills. Another popular application is using them as Distance Transforms (DTs) [12]. DTs store the distances to the nearest non-white pixel for each pixel within an image. Graph (4) of figure 1.2 shows the DT for a given cuneiform query.

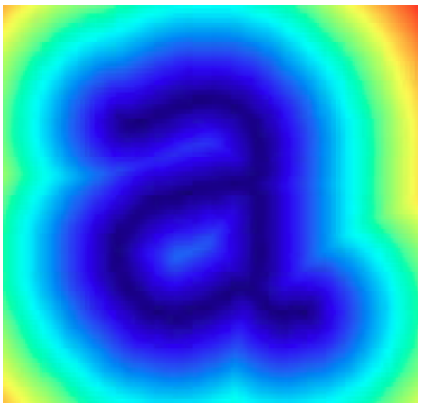
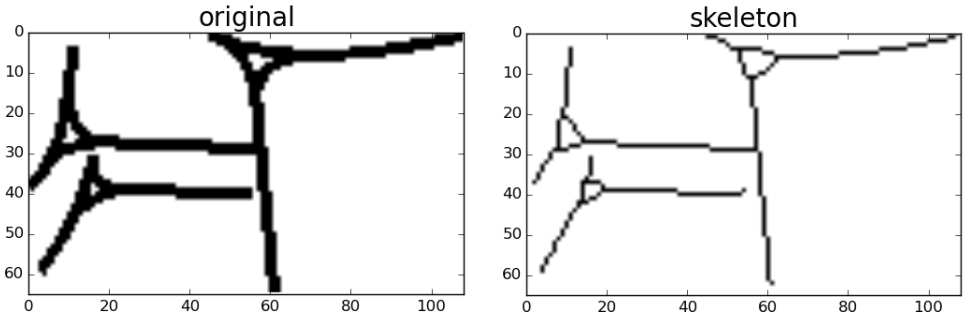


Figure 1.2: A Distance Transform created by Howe et al. [4]

Scalar Field & Energy In image processing a scalar field is a model, which assigns a scalar value or energy to each pixel of a rasterized image. A Distance Transform is an example for a scalar field, since each pixel of the original image is assigned an energy (the distance) to the nearest non-white pixel.

Medial Axis The medial axis or topological skeleton of an object is a thinned down version of the object itself. All boundaries are equidistant from the skeleton [13]. Figure 1.3b shows a skeletonized version of a cuneiform symbol in comparison to its original shape in 1.3a.



(a) (b)

Figure 1.3: Cuneiform symbol *is* (a) in a binarized version and (b) after skeletonization.

Precision & Recall Precision & Recall are used to evaluate the results of an algorithm. The precision shows the ratio between the items returned as positive matches against the items expected when considering the Ground Truth. It is defined as:

$$p = \frac{t_p}{t_p + f_p} \tag{1.1}$$

where t_p depicts the *True positives* which are the relevant results and f_p ,the *False positives* or the fraction of results that are irrelevant. The recall measures how many of the relevant items have been selected by the algorithm as positive matches:

$$r = \frac{t_p}{t_p + f_n} \tag{1.2}$$

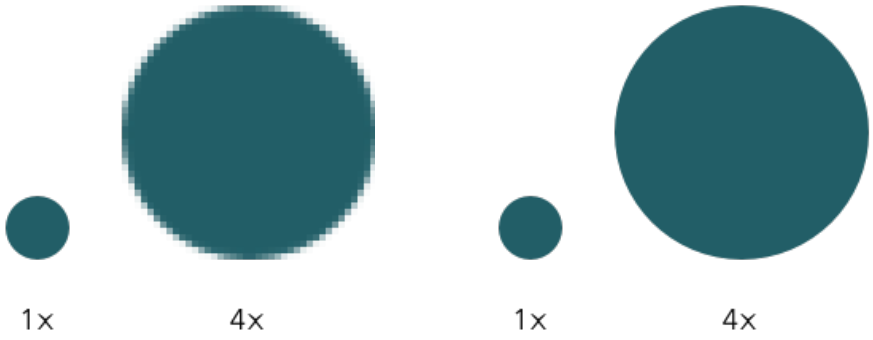
where t_p depicts the above and f_n depicts the *False negatives* or the items falsely accused to be not positive [14].

(a) (b) (c)

Figure 1.4: Lenna Test Image [15], (a) *S&P* combined, (b) Only *Pepper* added, (c) Only *Salt* added

Salt & Pepper Non synthetic images like photos shot with a digital camera can have some degree of noise. In image processing when using synthetic images, like vector paths images however, those images do not contain any noise. To add a magnitude of realism and to extent on tests for written algorithms against more realistic data this noise can be added manually. One application is the *Salt & Pepper* (S&P) approach, where white and black pixels are added randomly to the image. Various forms of this technique can be seen in figure 1.4.

Rasterization Converting vector graphics into raster image is called *Rasterization*. A raster image is an image consisting of pixels or dots in contrary to the paths that make up vector graphics. Raster graphics decrease in quality when scaled up while vector graphics can be arbitrarily scaled and do not suffer from loss of quality. Figure 1.5 illustrates the difference between the two graphic formats. One can clearly see the loss of quality arising with the scaled raster graphic.



(a) (b)

Figure 1.5: A comparison between (a) a raster graphic and (b) a vector graphic. [16]

1.4 Thesis Structure

After stating the motivation and the related work the following chapters describe a method of spotting words on a cuneiform tablet based on the work of Howe et al. [4]. First the concepts are described in chapter two and then the algorithms used are discussed in detail in chapter three. Each of these chapters are summarized in the end before proceeding. Chapter four shows test results on artificial as well as on real-world data. Finally a summary of the method and a future outlook for following research is given.

2 Part-Structured Spotting

First of all it is discussed, which inputs are needed as well as what kind of results can be expected from the method. The method is based on the idea of using a cuneiform query image and a cuneiform tablet as target computing a Distance Transform.

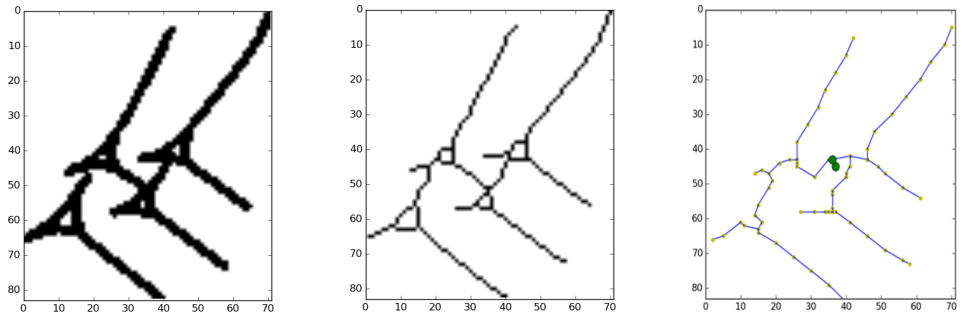
2.1 Query & Target

For the method to perform multiple inputs are required. The two most crucial inputs are the query symbol, which is represented either by vector paths or a rasterized image and the target, which can also be either a set of vector paths or a rasterized image. The query symbol represents the symbol or group of symbols to scan for on the cuneiform tablet. The target represents the cuneiform tablet which is supposed to get scanned. When using vector paths they need to be rasterized in an additional step before handing it to the method.

2.1.1 Query Symbols & Groups

The query represents the symbol or symbol group the user wants to scan for on the target tablet. The symbol representing the query inside the binarized image is thinned down using a medial axis method. The skeletonized version of the query is used to create a model in shape of the query image and can be seen in figure 2.1b in comparison to the original symbol shown in figure 2.1a.

This model is represented as a tree-structure. It consists of a set of nodes $N = \{n_i|1, ..., m\}$ where m is the number of nodes. One node $n_i \in N, 1 \leq i \leq m$ is the root node of the tree model. For convenience consider $i = 1$ without



(a) (b) (c)

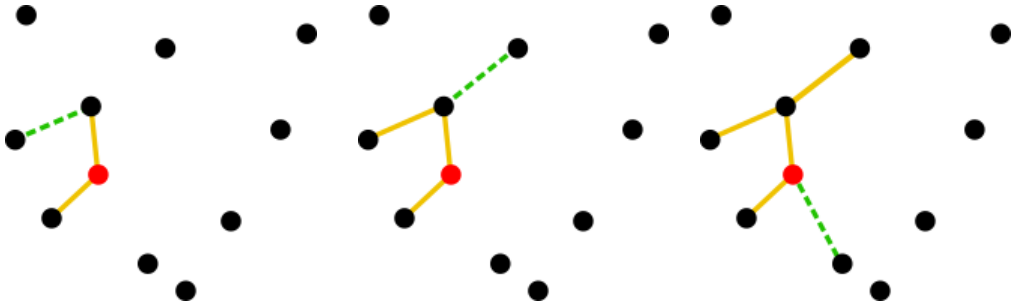
Figure 2.1: A query symbol (*še*) in its (a) original shape, (b) skeletonized and (c) as a full tree model with a minimum distance of 5 pixels. The green dots symbolize the center of mass and the closest node to it.

the loss of generality. Each node n_i has exactly one parent node denoted $p_i \in N \setminus n_i$ and a set of children denoted $c_i = \{c | c \in N \setminus (p_i \cup n_i)\}$. However the root-node does not have a parent, therefore p_1 is None. A node can only be child of one and only one other node and thus the tree has no cycles or self-references.

Before building up the tree however the nodes need to be set within the skeletonized version of the query. This is done by finding the center of mass initially and declaring the node closest to the center of mass as the root node of the tree computing the euclidean distance between the center of mass' coordinates and all nodes. The center of mass is found by calculating the mean coordinates of all nodes.

The junctions and corners of the skeleton are recognized using a method like the one described by Laganière & Elias [17]. The found locations of the junctions and corners depict the locations of the first nodes created. Starting from the junctions new nodes are added on the skeleton of the query always taking a minimum distance $d \in \mathbb{N}$ into account. The new nodes serve again as starting points for more nodes to be created. This results in various nodes which are located on the query's skeleton with a distance of at least d.

Using the existing nodes which do not have any relation yet the tree model



(a) (b) (c)

Figure 2.2: Three iterations of the greedy approach finding the closest node, which is not yet part of the tree. (a) First Iteration, (b) Second Iteration, (c) Third Iteration.

is build up using a greedy approach. Let Π be the set of nodes within the tree, Ω the set of nodes, which are not yet part of the tree and Σ the set of nodes without the root node $r \in \Pi$. For each node in Π the node having smallest euclidean distance $c \in \Sigma$ to any node in Ω is added as a new node n to the tree and thus $n \in \Pi$. It also gets removed from the set of nodes outside the tree: $n \notin \Omega$. The node in Π closest to the node, denoted as $p \in \Pi$ becomes the parent node to n. This is repeated until Ω is empty ($\Omega = \emptyset$). Three iterations of this approach can be seen in figure 2.2 and Algorithm 1. A complete result using the cuneiform symbol *še* is shown in figure 2.1c.

2.1.2 Target Image

Apart from the query the other required input for the method to work is the target image. It represents the cuneiform tablet which should be scanned for the query. The target can, like the query, be either a set of vector paths or a rasterized image. Before the method can work with the target it also needs to be rasterized if presented as vector paths. When using a rasterize image it is advisable to use a grayscale image without any *Alpha-Values* to ease the computation. Let $\omega \in \mathbb{N}_{\leq 255}$ be a threshold. This threshold determines the

Algorithm 1 Construction of the tree model used for queries

Require: Π, Ω, Σ, $r \in Pi$
Ensure: A tree model representing the form of the query

```
function GREEDY_TREE
    Ω ← Σ
    Π.append(r)
    while ω is not empty do
        c ← 0
        n ← None
        p ← None
        for t in Π do
            for o in Ω do
                d ← euclidean distance between t and o
                if d < c then
                    c ← d
                    n ← o
                    p ← t
                end if
            end for
        end for
        p.add_child(n)
        Ω.remove(n)
        Π.append(n)
    end while
end function
```

boolean outcome of each pixel for the rasterized target image as shown by Samanta et al. [18].

2.2 Distance Transform

The Distance Transform is the core feature of the algorithm. First the different measurements for DTs useful for the method are discussed. Second the application of them within the method is shown in detail.

2.2.1 Distance Measurements

As shown in 1.3 the Distance Transform is similar to a height map showing the distances from each pixel to the next obstacle in a given image. In this case the obstacles of the target image are the inked pixels representing the paths on the target image. One way to calculate the Distance Transform is thus to calculate the euclidean distance to the next inked pixel on the target image for each pixel. Since a good distinction is needed between local minima and the surrounding values the euclidean distance by itself is often not sufficient.

One alteration to the standard euclidean distance is to use the *Squared Euclidean Distance* (SDT) by squaring all euclidean distances calculated before. This allows for a greater distinction, since pixels become irrelevant more quickly but also supports greater divergences in high values. Pixels that have been considered a good match because of their low value could now not be good enough to be considered a good match. It also allows avoiding the need of rooting the DT, which saves additional computation resources. An improvement however is to use the *Generalized Distance Transform* (GDT) also used by Howe et al. [4] and described by Felszenwalb & Huttenlocher [19].

When using the SDT, paraboloids extend from each inked pixel determining the distances of the surrounding pixels. The minimum of each pixel is then taken as the distance. Figure 2.3 shows cross-sections of a 2D image, whereas black bars determine non-inked pixels. Figure 2.3a shows the application of simple euclidean distance rooted at zero whereas figure 2.3b shows the use of the squared euclidean distances. The paraboloids are extended at each

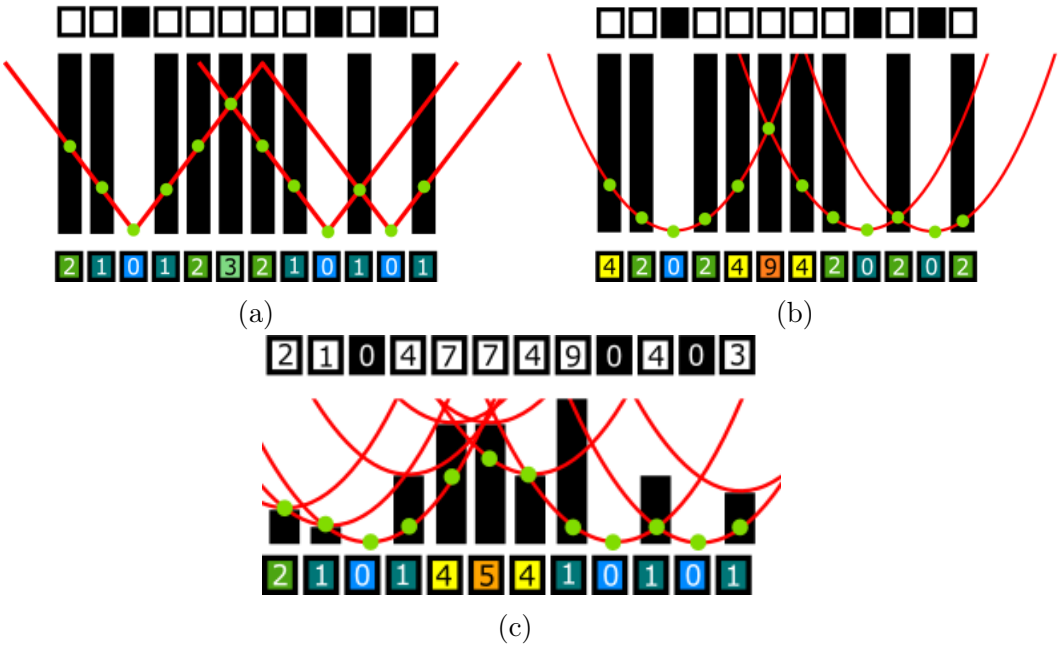
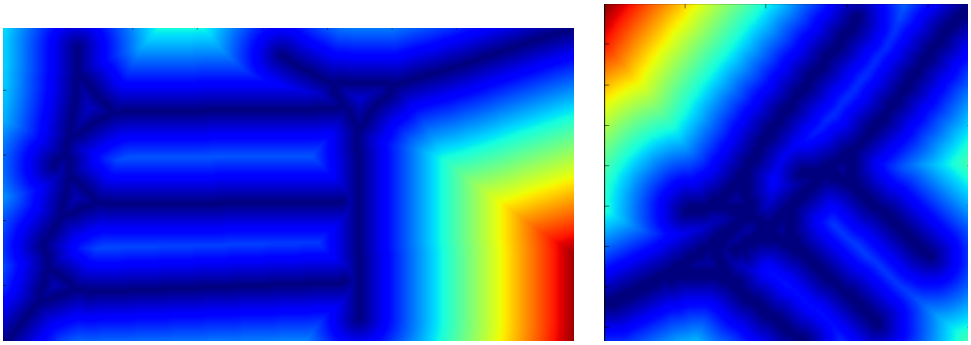


Figure 2.3: 1D-examples of different distance transforms. A distance transform created with (a) euclidean distance rooted at zero. (b) a parabola using squared euclidean distance. (c) distance transform using the GDT method rooted at the pixels value.

inked pixel rooted at zero. For each non-inked pixel the minimum of the paraboloids crossing that pixel is taken to determine the distance. When using the GDT, as seen in figure 2.3c, the paraboloids are not only extended at non-inked pixels but at every pixel of the image rooted at the pixels value and again the minimum of the paraboloids crossing a pixel determines the distance of this pixel to the nearest non-white pixel.

2.2.2 Single-Node Model

Let the tree model created with the query consist of only one node which then automatically becomes the root node. Iterating the node over each pixel of the image and storing the distance from the nodes location to the next inked pixel results in a scalar field similar to the Distance Transform of the target image. Two examples of DTs for cuneiform symbols can be seen in 2.4



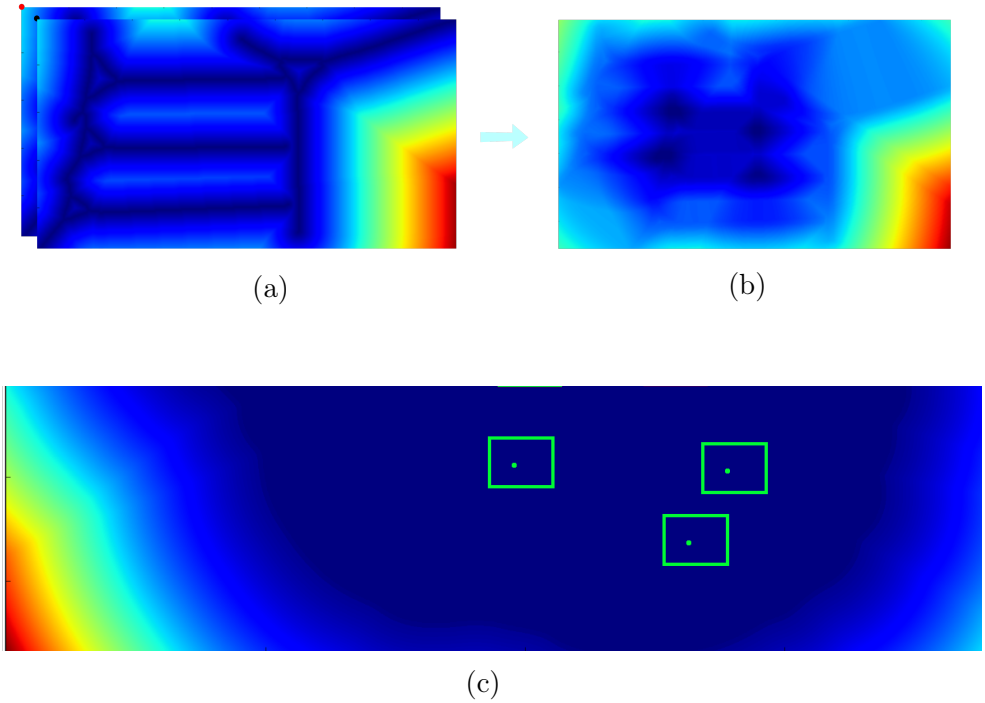
(a) (b)

Figure 2.4: Single Node DTs of the cuneiform symbols (a) *ma* and (b) *is*.

2.2.3 Dual-Node Model

Adding a node to the single node model results in a dual-node model having one root node and a child of the root node. Each node can be seen as separate single-node models and thus would create two equivalent scalar fields or Distance Transforms of the target image. However to combine them to a dual node model the two scalar fields are added together using the child's relative offset to its parent. This addition is shown in the figures 2.5a and 2.5b. This can be best imagined by placing two sheets of checkered paper on top each other with an offset similar to the parent-child offset. The sheets of paper can be seen as the scalar fields created by a Single-Node model having a scalar value imprinted in each of the squares. All overlapping squares get

added together resulting in new values. The areas to be added together are illustrated in figure 2.10.



(a) (b)

(c)

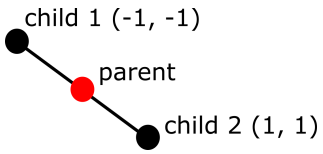
Figure 2.5: Illustration of (a) the offset addition and (b) the resulting scalar field and a DT (c) resulting from the addition of a full model with embedded minima.

2.2.4 Full Model

Howe et al. [4] suggests using a full tree model, where a node's offset is taken to calculate a dual node model based on two single node DT, which are created using the SDT. After the summation a GDT calculation is applied to the resulting offset energy and finally summed together with another single node DT. When iterating over all nodes the resulting DT depicts minima at high parities between query symbol and a found symbol on the target image. Figure 2.5c shows a resulting DT with highlighted minima.

Node	Offset
Root	(0, 0)
Child 1	(-1, -1)
Child 2	(1, 1)

(a)



(b)

Figure 2.6: An (a) example tree model with three nodes and (b) its graphical representation.

2.2.5 Merging Nodes

When summing up the overlapping areas the non-overlapping areas are not changed and therefore they are very likely to be of a smaller value than the summed up areas. This can lead to problems when finding the local minima, since these will be in the non-overlapping area. Another problem occurring is the size of the resulting scalar field. To sum up the scalar fields a new larger scalar field needs to be created, thus in the end the dimensions of the resulting scalar field do not match the target image and a local minimum at specific coordinates do not match the coordinates of the symbol within the target image. Furthermore the relative offsets need to be taken into account. In this thesis all coordinates written in $(\cdot, \cdot)$ are presented in image coordinates (pixels). Consider the model of figure 2.6.

In this case the root node has two children, one to the bottom right and one to the top left of the root node. When summing up the root node's scalar field with the scalar field of the first child the root nodes relative starting location is not at (0, 0) anymore. Imagining the analogy from before again this would result in two sheets of paper on top of each other, where the top one is further up and left than the bottom one. The sheet of paper at the bottom represents the root node's scalar field and when adding the scalar fields together the child's scalar field starts at (0, 0) whereas the root's scalar field starts at (1, 1). This problem is shown in figure 2.7.

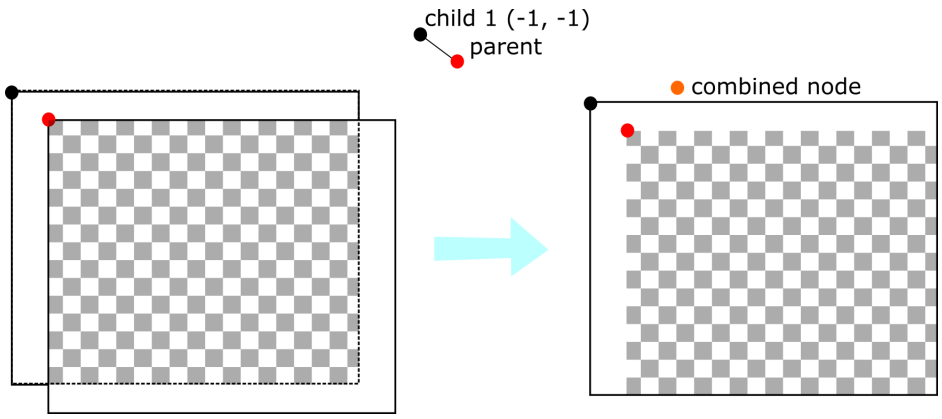
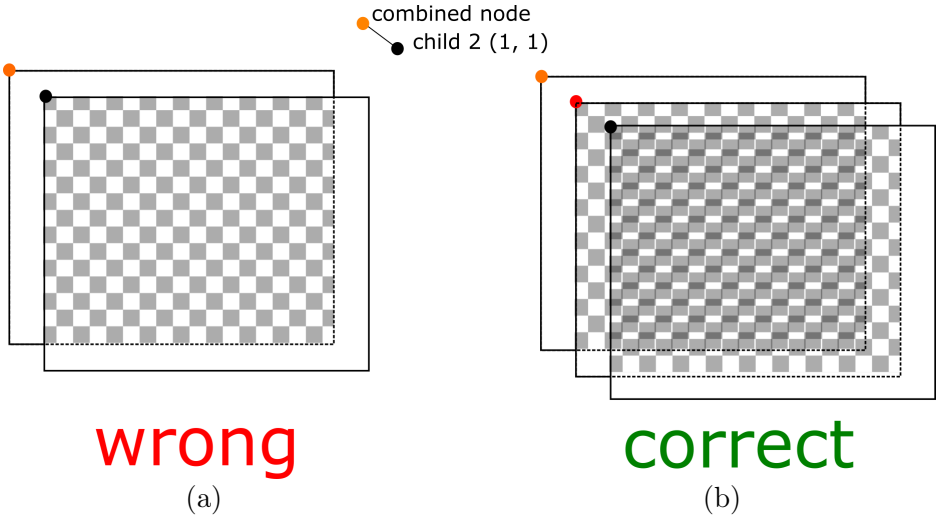


Figure 2.7: Addition of the first child to the root node resulting in a scalar field in which the root node has an absolute offset of (1, 1).

Adding the second child to the resulted scalar field requires a total offset of (2, 2), since the second child has a relative offset of (1, 1) to its parent node which is the root node located at location (1, 1) in the scalar field from before. A comparison between the wrong and the correct calculation can be seen in 2.8.

To tackle this challenge a first approach was to create an empty scalar field of the size of the eventually resulting scalar field and placing the single node scalar fields at the absolute locations of the tree model nodes coordinates. However this resulted in calculation errors and was eventually scrapped and replaced by a much easier solution.

The problem of the wrong calculations with relative offsets only occurs when starting the summation from the root node and then traversing to the children throughout the tree. However using a *Depth First Search* (DFS) algorithm avoids this problem, since first the children are added together and the resulting scalar fields are returned to the parents, although there are still four cases that need to be considered in the calculation. The four cases are:



(a) (b)

Figure 2.8: Results when (a) not considering the absolute and (b) correctly considering the absolute offset of the root node.

The child is to the

1. top left
2. bottom right
3. bottom left
4. top right

of the parent node. All four cases are illustrated in figure 2.9.

For each of these cases certain areas get added together. Let x, y be the offset of the child relative to its parent and the child be to the bottom right of the parent node. The overlapping part for the parent node start at x and y for width and height respectively. The areas left of x and above y are non-overlapping and therefore irrelevant. For the child however the overlapping areas start both at 0 for width and height but end at $width - x$ and $height - y$. To the right and bottom of these borders the areas of the child are non-overlapping and thus also irrelevant. Figure 2.10 shows the area

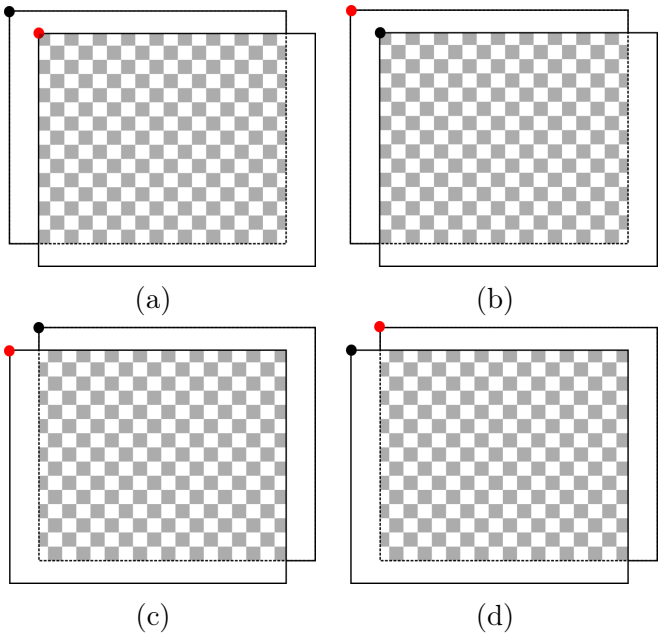


Figure 2.9: The four cases with child at (a) top left, (b) bottom right, (c) bottom left & (d) top right

of interest for two nodes in case 1. These areas differ for each case and need to be adjusted accordingly. Algorithm 2 shows the summation for a full node model. The areas of interest for parent and child are given in the squared brackets.

Algorithm 2 Pseudo-Code for summing up the Distance Transforms for all nodes

Require: The root node of the tree, height and width of the resulting DT.
Ensure: After the function is finished the summed up DT is stored inside the DT of the root node.

```
function CALCULATE_SUM(node)
    for child in node.children do
        y ← child.offset[0]
        x ← child.offset[1]
        end_y ← HEIGHT - abs(y)
        end_x ← WIDTH - abs(x)
        if y ≥ 0 and x > 0 then
            node.root_dt[abs(y) :, abs(x) :]+= CALCULATE_SUM(child)
        else if y ≤ 0 and x < 0 then
            node.root_dt[: end_y, : end_x]+= CALCULATE_SUM(child)
        else if y < 0 ≤ x then
            node.root_dt[: end_y, abs(x) :]+= CALCULATE_SUM(child)
        else if y > 0 ≥ x then
            node.root_dt[abs(y) :, : end_y]+= CALCULATE_SUM(child)
        end if
    end for
    if node.offset ≠ None then
        if y ≥ 0 and x > 0 then
            return node.root_dt[: end_y, : end_x]
        else if y ≤ 0 and x < 0 then
            return node.root_dt[abs(y) :, abs(x) :]
        else if y < 0 ≤ x then
            return node.root_dt[abs(y) :, : end_x]
        else if y > 0 ≥ x then
            return node.root_dt[: end_y, abs(x) :]
        end if
    end if
    return node.root_dt
end function
```

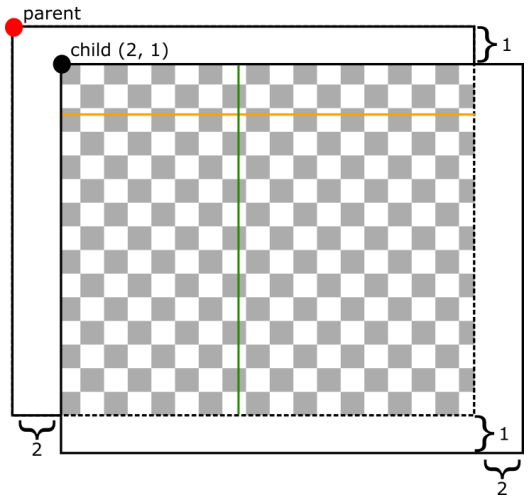


Figure 2.10: The areas that need to be summed together are overlapping. Only these parts are relevant for the summation.

2.3 Summary

In this chapter the theoretical fundamentals were discussed. It was stated that for the method to work two inputs are obligatory. The query represents the cuneiform symbol or symbol group to scan the target image for. Both, the query and the target need to be rasterized before further calculations. After the binarizazion of the images the medial axis or the skeleton of the query is calculated. This skeleton is then used to specify the locations of a set of nodes resulting which in return become part of a tree structure with an arbitrary root node in which no cycles are allowed and each node has only one parent resulting in a tree model, representing the query symbol.

A Single-Node SDT is created from the target image and for each node of the tree-model a Dual-Node DT is calculated by summing two single node SDTs, whereas one of them is offsetted by the node's relative offset to its parent node. A GDT is applied to the translated scalar field and another single node DT is added. Iterating over all nodes results in a DT, which minima depict high parity between the query and a symbol on the tablet.

3 Algorithms for Character Spotting

The following sections describe the details of the implementation for the application written for this thesis.

3.1 Parameters

Apart from the inputs required by the method described in 2.1 on page 14 the application written for this thesis allows for different other parameters:

`--scale | -s`

This argument determines the scale of the query and target image. The query needs to be equally scaled as a symbol on the target image for this approach to work. Often times however the input queries or target images are too small for the algorithm to properly execute. Using rasterized images scaling can lead to loss of quality. To avoid this issue the `--scale`-parameter was introduced. It simply scales up the query and target by multiplying the size with a number given.

`--limit | -l`

The application allows for multiple tweaks, such as limiting the number of results returned by the algorithm. `--limit` or `-l` followed by an integer number tells the application to only return the top n parities found on the target image.

Apart from the command line arguments more tweaks are available in altering constant variables:

DISTANCE This is the minimum distance between two nodes described in 2.1.1 on page 2.1.1.

THRESHOLD This is the threshold determining the Boolean outcome of the conversion from binary image to Boolean array 2.1.2 on page 16.

The application returns the height map of the target image in form of a float array. All local minima are found within this float array. How many minima are returned depends on the argument `--limit`. Furthermore it returns the symbols located at the minimas' locations. The symbols are extracted from the target image using a bounding box of the same shape as the query and placing it on top of each minima's location within the target image.

3.2 Query Symbol & Target Image

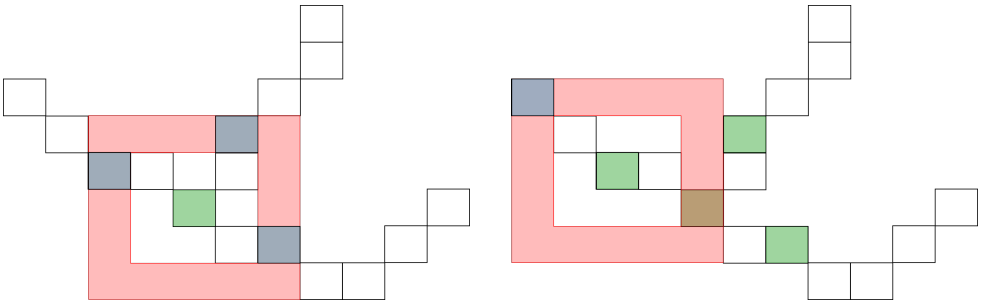
In the application written for this thesis it is either possible to read in *Scalable Vector Graphics* (SVG) files or already binarized *Portable Network Graphics* (PNG) files. Since SVG-files use paths and vectors one can define groups of paths to a symbol beforehand using third party software, which in return get read out by the application in the first step. Each one of the groups is stored inside a list and through indexing the user can choose one of the groups to be the query. These groups consist of the paths themselves and are not yet binarized. Before proceeding the vector paths need to be converted into rasterized images using Qt [20]. When using PNG images instead of SVG-files for the query or the target respectively no binary conversion is needed since PNG images themselves are already rasterized.

3.3 Placement of Nodes

Before building up the tree model it is required to create the nodes which then get added to the tree using the greedy algorithm described before. The rasterized query image is converted to a Boolean array depicting the foreground as TRUE and the background as FALSE. Furthermore the query needs to be thinned down to a maximal width of 1px using a method provided by scientific programming libraries which use the medial axis to return a *skeletonized* version of a Boolean array. The thinned down version of the query is used to determine the locations of the nodes. The only locations nodes can appear are at TRUE entries of the Boolean skeleton array, since these represent the symbol.

Therefore the first nodes are added at junctions and corners of the query symbol. Starting from each of the newly added nodes, which also get added to an *open*-list, the neighborhood of the nodes are checked for TRUE entries. The minimal distance d is defined by the DISTANCE constant variable set in the code. It represents the minimum distance between two nodes in pixels. The default is 4, resulting in a 4x4 neighborhood. Each cell lying exactly 4 entries away from the node is checked for TRUE. If a checked cell is indeed TRUE a new node with the coordinates of the cell is created and added to the *open*-list. As soon as all neighboring cells are checked the node gets added to a *closed* list. Figure 3.1 illustrates two iterations of the method for a DISTANCE of 2px.

Raising the DISTANCE lowers the calculation time but at the same time lowers the quality of the resulting tree, since less nodes are added, vice versa a small DISTANCE increases the calculation time but results in a high quality of the tree. The node closest to the center of mass of all nodes is declared the root node (n_r) of the tree. This is done until no nodes are left in the *open*-list.



(a) (b)

Figure 3.1: Two iterations of finding coordinates for new nodes using the neighborhood.

3.4 Constructing the Tree

Let the number of nodes be n. Beginning with only the root node each other node (n-1 nodes) is traversed to find the closed non-tree node to the root-node. When the closest node is found it becomes the child of the root node, thus the tree consists of two nodes, while n-2 nodes are remaining outside the tree. That step is repeated until each node is part of the tree. This means for the two existing nodes in the tree the other n-2 nodes need to be traversed to find the closest node. The total number of calculations can therefore be defined as the following sum:

$$\sum_{i=1}^{n} i(n-i) \tag{3.1}$$

Lemma 1 (Greedy Tree complexity)**.** The greedy approach to build up the tree has an algorithmic complexity of $\mathcal{O}(n^3)$

$$\sum_{i=0}^{n} i(n-i) = \frac{n(n-1)(n+1)}{6} \Rightarrow \mathcal{O}(n^3)$$

Proof:

Base-Case:

$$\sum_{i=0}^{0} i(n-i) = 0(0-0) \tag{3.2}$$

n=0:

$$\begin{aligned} \sum_{i=0}^{0} i(n-i) &= \frac{0(0-1)(0+1)}{6} \\ 0(0-0) &= \frac{0(0-1)(0+1)}{6} \ (Equation\ 3.2) \end{aligned} \tag{3.3}$$

Hypothesis:

$$\sum_{i=0}^{n} i(n-i) = \frac{n(n-1)(n+1)}{6} \text{ is valid for one } n \in \mathbb{N}_0 \tag{3.4}$$

Induction-Step:

Assuming the hypothesis in 3.3 holds for one $n \in \mathbb{N}_0$, it must now be shown that it also holds for $n+1$, that is:

$$\sum_{i=0}^{n+1} i((n+1)-i) = \frac{(n+1)((n+1)-1)((n+1)+1)}{6} \tag{3.5}$$

$$\begin{aligned} \sum_{i=0}^{n+1} i((n+1)-i) &= \sum_{i=0}^{n} i((n+1)-i) + (n+1)((n+1)-(n+1)) \\ &= \sum_{i=0}^{n} i(n-i) + i \end{aligned}$$

$$
\begin{aligned}
&= \underbrace{\sum_{i=0}^{n} i(n-i)}_{\text{Hypothesis}} + \underbrace{\sum_{i=0}^{n} i}_{\text{Gaußian Sum}} \\
&= \frac{n(n-1)(n+1)}{6} + \frac{n^2+n}{2} \\
&= \frac{n \overbrace{(n-1)(n+1)}^{\text{Third binomial formula}} + 3(n^2+n)}{6} \\
&= \frac{n(n^2-1)+3n^2+3n}{6} \\
&= \frac{n(n^2-1+3n+3)}{6} \\
&= \frac{n(n^2+3n+2)}{6} \\
&= \frac{n(n+1)(n+2)}{6} \\
&= \frac{((n+1)-1)(n+1)((n+1)+1)}{6} \\
&= \frac{(n+1)((n+1)-1)((n+1)+1)}{6} \Rightarrow \mathcal{O}(n^3)
\end{aligned}
$$

□

This proof shows that the creation of the tree has cubic complexity, where n is the number of nodes created by the neighborhood approach.

3.5 Summary

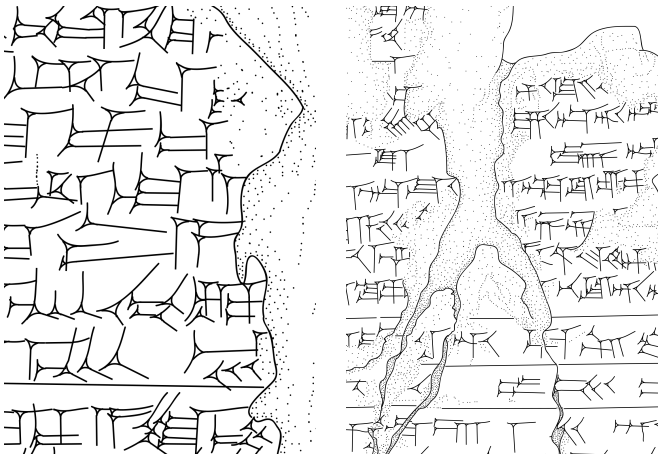
For the method to work multiple algorithms are needed. First the query needs to be thinned down to a skeleton using a medial axis method, such as the SKELETONIZE-method provided by the public Python library *SciPy*[1]. After retrieving the skeleton of the query the nodes need to be placed to represent the skeleton using the 8-neighborhood approach. The nodes, which do not have any relation are added to a tree model after choosing an arbitrary root first. The tree is built up by a greedy approach adding the node closest to the tree, which itself is not part of the tree, as a child to the node it is closest to. This approach holds a cubic complexity without taking the calculations of the GDT and SDT into account.

[1] `http://scikit-image.org/docs/dev/auto_examples/plot_skeleton.html` - Last visited: 30.03.2016

4 Results

After discussing the algorithm in great detail the following chapter presents the results created by the application. Due to the difficulties in obtaining real Ground Truth described in 1.1 on page 3 the evaluation for the method is done manually. For this multiple queries are tested against a set of results returned by the method. For each query the top 30 matches are presented and determined whether they are good (positive) or bad (negative) matches by comparing the found symbol with the query used.

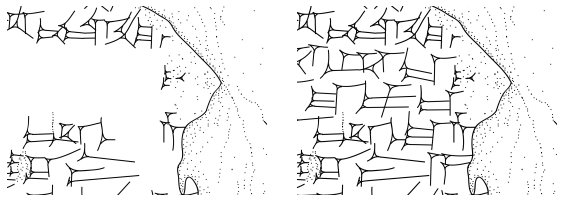
4.1 Manual Drawings



(a) (b)

Figure 4.1: Parts of the two vector based tablets (a) *VAT_10321_Vs* and (b) *VAT_09671_Rs* used for evaluation. *Vs* and *Rs* are abbreviations for the German words *Vorderseite* (Front Side) & *Rückseite* (Back Side).

For testing purposes cuneiform tablets are used. Two of the tablets are given in SVG format and can be seen in 4.1. All of these tablets are available in two versions. However one version of the tablets only consists of the paths building the cuneiform symbols on the tablet and some noise paths or borders. The other version groups paths of symbols according to their semantic meaning. The latter is used for reading in the wanted query image described in chapter 2.1.1 on page 14. Since the libraries used to load SVG files into the application written for this thesis do not support layers the latter ones can not be used as the target image because grouped paths are represented on another layer which are ignored when loading the SVG into the application. Therefore the first version of the tablet, which has no semantic grouping and thus only one layer is used for the target image to avoid losing paths or more precisely symbols on the target tablet. The difference between a tablet with grouping and one without can be seen in figure 4.2.



(a) (b)

Figure 4.2: Comparison of the two versions for *VAT_10321_Vs* (a) with an extra layer and (b) without an extra layer for grouped symbols.

The fourth tablet used is a non-artificial real world asset taken from *The British Museum* online collection [21]. It is a handwritten representation of a clay tablet found in the middle east and a part of it can be seen in figure 4.3. Multiple symbols were extracted manually to serve as queries.

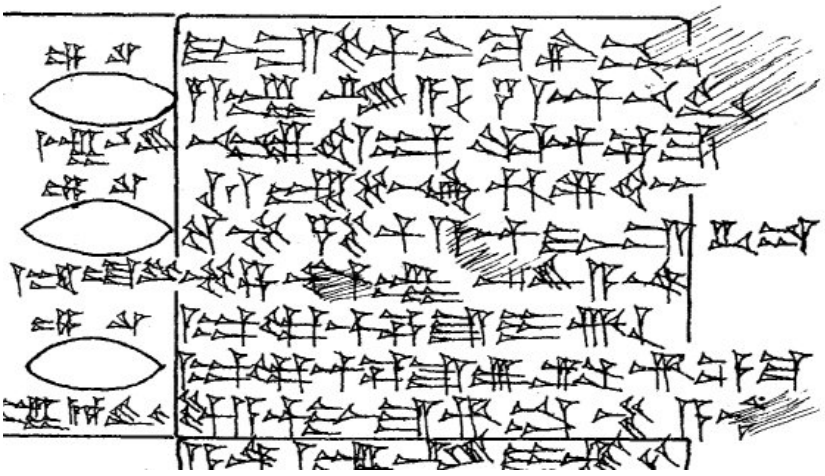


Figure 4.3: Part of *AN00850096_001_l* [21]

4.2 Tests on Synthetic Data

All results received by the application using the SVG files as inputs are seen as tests on synthetic data, since SVG files do not have any loss in quality when scaling up the image. However in real-world tests tablets are often not available in SVG format, but rather in a binarized version leading to pixelization when scaling the picture and thus reduced quality of the image. Furthermore the SVG files are often noise free and only represent the pure data, which in most cases does not apply to binary scans of tablets. For the artificial tablets multiple different queries are tested, returning the top 30 matches for each query found on the tablet. Figure 4.4 shows three of the queries with their according matches and energy levels using the *VAT_10321_Vs* tablet. Figure 4.5 shows some of the results for each query highlighted on the target tablet.

Figure 4.4: The top 30 matches for the symbols *ma*, *is* & *še*. The query symbol and the corresponding model tree can be seen at the very left. A result framed green is a positive match to the query, while red signifies a negative match to the query.

These test results are taken from the SVG without any modification to the SVGs like adding noise. However these test results as stated are synthetic, because most times tablets found in real-world are noisy or even broken, so the symbols on the tablets can not be used properly. To simulate a certain degree of realism *Salt & Pepper* was added for another test leading to different results.

(a) (b) (c)

Figure 4.5: Some of the highlighted results for the queries (a) *ma*, (b) *is* & (c) *še*.

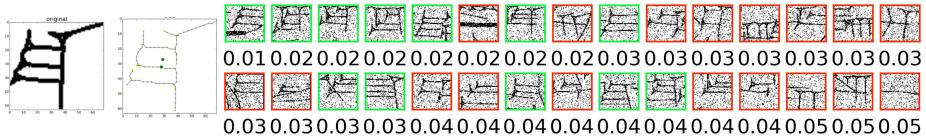


Figure 4.6: Results for query *ma* after adding *Salt & Pepper*.

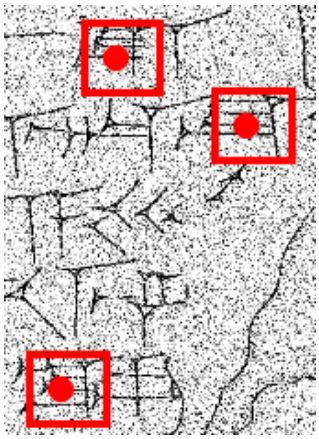
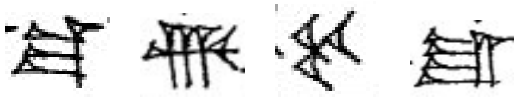


Figure 4.7: Highlighted results on the noised tablet.

The S&P test was done with the *VAT_09671_Rs* tablet with a representation of the symbol *ma*. The noise was added using a *SciPy* specific method called RANDOM_NOISE with an AMOUNT-value of 0.3. Figure 4.6 shows the according result set returned by the method, whereas figure 4.7 shows a part of the tablet with highlighted results.

4.3 Tests on Real-World Data

As previously described in 4.1 the tests on real-world data are based on a tablet taken from *The British Museum*. For the tests four symbols were extracted manually and used as queries. The queries can be seen in figures 4.8. Results for the query shown in 4.8a can be seen in figure 4.9 and a few of the results highlighted in the real-world tablet in figure 4.10.



(a) (b) (c) (d)

Figure 4.8: The four queries manually extracted from the tablet with real-world data.

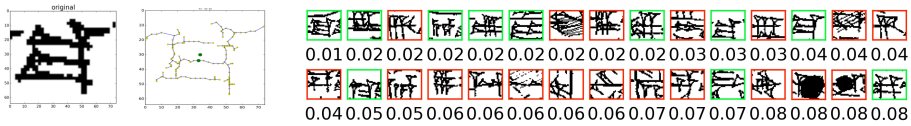


Figure 4.9: Returned results for one of the queries on the real-world tablet

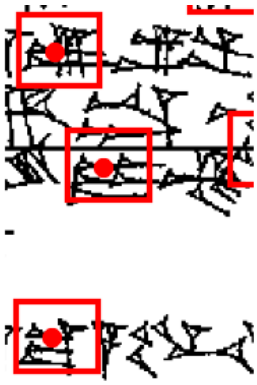


Figure 4.10: Highlighted results on the real-world tablet

4.4 Precision & Recall

Since no real *Ground Truth* is available the *Precision* and *Recall* are based on the results returned. For each query a specific number of positive results return within the limited 30 results. These are used for the precision value, whereas the energy associated with each result serves as the decision value. Using both of these values a *Precision-Recall-Graph* (PR-Graph) can be constructed. It shows the percentage of the *Precision* dependent on the increasing *Recall*. For each query a unique Graph can be constructed as can be seen in figure 4.11.

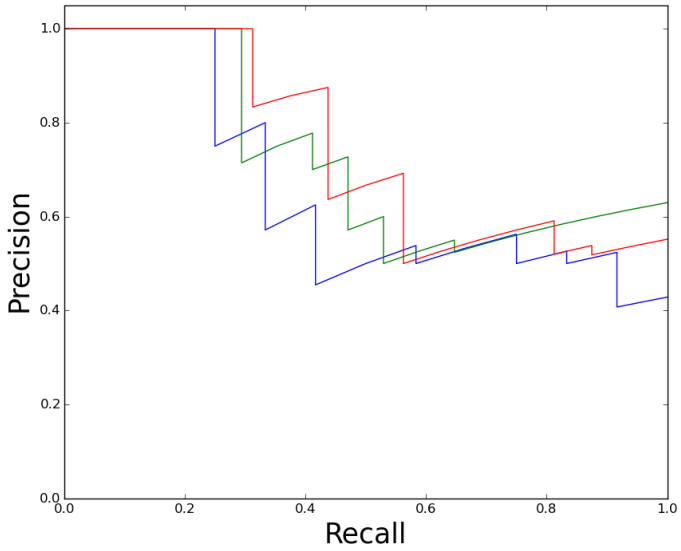


Figure 4.11: A PR-Graph showing the *Precision* & *Recall* for the three queries, *ma* (blue), *is* (green) & *še* (red)

Furthermore an all over PR-Graph for each tablet can be constructed, determining the accuracy of the method for the tablets used. Figure 4.12 shows the *Precision* & *Recall* for the two tablets iterating over all queries the tablets have available. The PR-Graph for the noised tests using *Salt &*

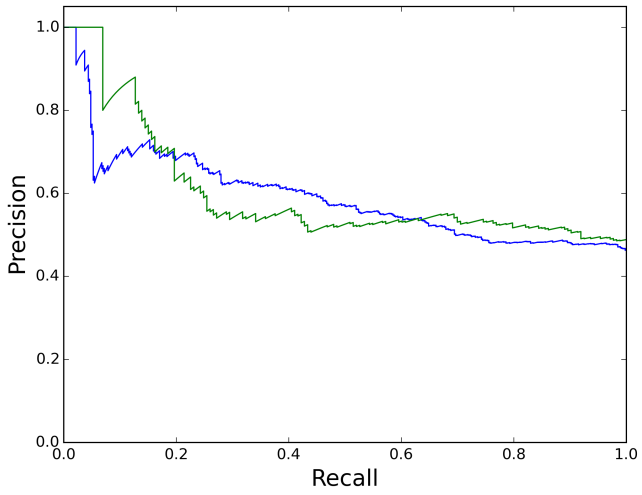


Figure 4.12: PR-Graph for the two SVG tablets, *VAT_10321_Vs* (blue) and *VAT_09671_Rs* (green) tested in their original forms.

Pepper on the *VAT_09671_Rs* tablet can be seen in 4.13.

The PR-Graph for the tests on real-world data using the handwritten tablet can be seen in figure 4.14. All of the tests, being done with synthetic data or real-world data can be combined to an overall PR-Graph to illustrate the *Precision* & *Recall* of the entire method. This PR-Graph is shown in 4.15.

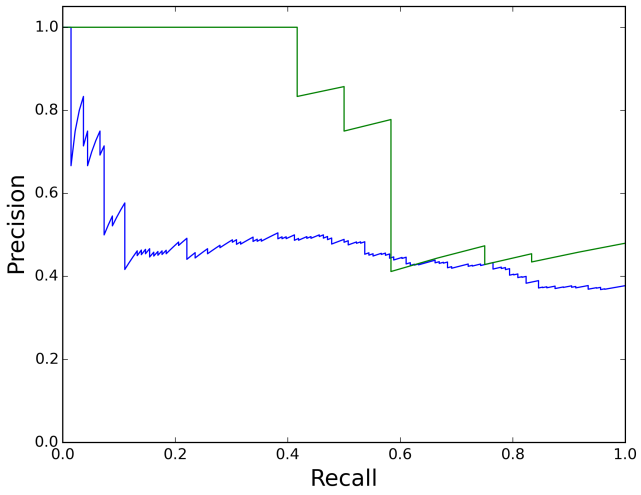


Figure 4.13: PR-Graphs for the query symbol *ma* (green) and all queries (blue) on a noised version of *VAT_09671_Rs*

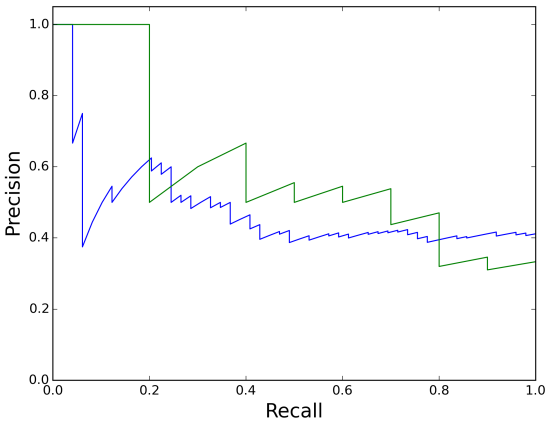


Figure 4.14: PR-Graph for the query 4.8a (green) and all four queries combined (blue) tested on the real-world data.

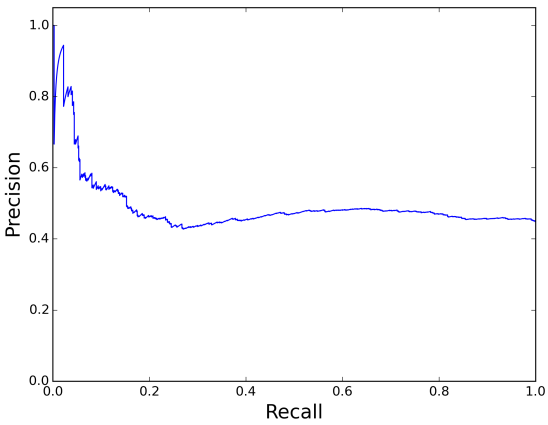


Figure 4.15: Overall PR-Graph of the method described in this thesis

5 Summary & Outlook

This thesis describes a new method to spot cuneiform symbol and groups of symbols called queries on any given cuneiform tablet based on a method first described by Howe et al. [4] tested on texts using Latin script. The query should be of same scale as a symbol appearing on the target tablet, meaning the wedges forming the query should be of approximately the same size than any wedge found on the tablet. Before finding any parities between the query and any symbol on the tablet the query needs to be pre-processed by skeletonizing it to a width of one pixel using a medial axis approach as described in section 2.1.1 on page 14. The skeleton version query serves as basis for the creation of a tree-model where the nodes of the tree can only have one parent while having multiple children. The nodes locations implicit the shape of the query.

Using multiple *Distance Transforms* such as the *Squared Distance Transform* and the *Generalized Distance Transform* on the target image one can calculate a scalar field of energies by summing up the various DTs depending on offset from each node to its parent node which was described in chapter 2.2 on page 18. The local minima of the scalar field resulting from summing up the DTs depict a high parity between a symbol located at the minima's location on the target image and the query used for the method.

As shown in chapter 4 on page 35 the method was tested against both synthetic data represented by vector images and real-world data represented by a raster image of a handwritten transcript of a clay cuneiform tablet. The results show a *Precision* which does not fall below 30% with increasing *Recall* even when adding *Salt & Pepper* to the synthetic data.

Although for this thesis the focus was not on time efficiency or low complexity of algorithms used the computation of the result set for one query

takes a few seconds. However the performance can be improved.

Parallelization of the calculation of Distance Transforms improves on the speed of the method. Parallelization is the task of distributing various calculations throughout an algorithm across a network of computers to have multiple computations done at the same time. Since most of the calculation of the method described here base on the Distance Transform being summed up multiple times and the operation of addition is associative as well as commutative it is possible to calculate these sums on different threads or using several computers at the same time.

Another improvement is calculating the Distance Transform on the graphics cards rather than the *Central Processing Unit* (CPU). This technique is often used in computer graphics, since graphics cards are often much faster then CPUs in view of computation times [22]. This could be achieved running this algorithm on *Compute Unified Device Architecture* (CUDA), which allows certain parts of an algorithm to be executed on the GPU [23].

However at the moment the biggest bottle-neck in consideration of complexity and speed is the greedy approach building up the parent-child tree with a cubic complexity. When using small distances between nodes on a high-scaled query image the cubic complexity increases the calculation drastically since the more nodes are placed by the neighborhood method described in section 2.1.1 on page 14 the more comparisons are needed to build up the tree model. There are several work tackling the so called “closest bichromatic pair challenge”.

In that challenge one is given two sets of points, one containing red points, the other one blue points. The goal is to find the closest pair of points over all red-blue pairs where one point lies in the red set and one in the blue set. This method is equivalent to the method used creating the tree-model, whereas one set consisting of nodes being part of the tree and the other set consisting of nodes not yet being part of it. A suitable method reduces the calculation time enormously like the one described by Khuller & Matias [24].

List of Acronyms

BB Bounding-Box

CDLI Cuneiform Digital Library Initiative

CPU Central Processing Unit

CTC Connectionist Temporal Classification

CUDA Compute Unified Device Architecture

DFS Depth-First Search

DT Distance Transform

FN False Negatives

FP False Positives

FCGL Forensic Computational Geometry Laboratory

GDT Generalized Distance Transform

GPU Graphical Processing Unit

GT Ground Truth

IWR Interdisciplinary Center for Scientific Computing (Institut für wissenschaftliches Rechnen)

OCR Optical Character Recognition

PNG Portable Network Graphics

PR-Graph Precision-Recall-Graph

RS "Rückseite" - (German for "Back Side")

S&P Salt und Pepper

SDT Squared Distance Transform

SVG Scalar Vector Graphics

TN True Negatives

TP True Positives

VS "Vorderseite" - (German for "Front Side")

Bibliography

[1] W. de Gruyter. *Schrift und Schriftlichkeit / Writing and Its Use*. H. Günther, O. Ludwig, Berlin, New York, 1994.

[2] R. Borger. *Assyrisch-babylonische Zeichenliste*. Neukirchener Verlag/Butzon & Bercker, Neukirchen-Vluyn, Germany, 1978.

[3] R. Mithe, S. Indalkar, and N. Divekar. Optical Character Recognition. 2:72–75, 2013.

[4] N. Howe. Part-structured Inkball Models for One-Shot Handwritten Word Spotting. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 582–586, Washington, DC, USA, 2013.

[5] V. Frinken, A. Fischer, R. Manmatha, and H. Bunke. A Novel Word Spotting Method Based on Recurrent Neural Networks. *IEEE Transactions Pattern Analysis and Machine Intelligence (TPAMI)*, 34:211–224, 2011.

[6] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber. A Novel Connectionist System for Unconstrained Handwriting Recognition. *IEEE Transactions Pattern Analysis and Machine Intelligence (TPAMI)*, 31:855–868, 2009.

[7] L. Fei-Fei, R. Fergus, and P. Perona. One-Shot Learning of Object Categories. *IEEE Transactions Pattern Analysis and Machine Intelligence (TPAMI)*, 28:594–611, 2006.

[8] M. Revow, C. Williams, and G. Hinton. Using Generative Models for Handwritten Digit Recognition. *IEEE Transactions Pattern Analysis and Machine Intelligence (TPAMI)*, 18:592–606, 1996.

[9] B. Bogacz, H. Mara, and M. Gertz. Cuneiform Character Similarity Using Graph Representations. In *Computer Vision Winter Workshop (CVWW)*, Seggau, Austria, 2015.

[10] B. Bogacz, J. Massa, and H. Mara. Homogenization of 2D & 3D Document Formats for Cuneiform Script Analysis. In *Historical Document Imaging and Processing (HIP)*, Nancy, France, 2015.

[11] L. Rothacker, D. Fisseler, G.G.W. Müller, F. Wichert, and G. Fink. Retrieving Cuneiform Structures in a Segmentation-free Word Spotting Framework. *International Workshop on Historical Document Imaging and Processing (HIP)*, (accepted for publication), 2015.

[12] H. Breu, J. Gil, D. Kirkpatrick, and M. Werman. Linear Time Euclidean Distance Transform Algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 17:529–533, 1995.

[13] Harry Blum. A Transformation for Extracting New Descriptors of Shape. In Weiant Wathen-Dunn, editor, *Models for the Perception of Speech and Visual Form*, pages 362–380. MIT Press, Cambridge, 1967.

[14] D.M.W. Powers. Evaluation: From Precision, Recall and F-Measure to ROC, Informedness, Markedness & Correlation. *Journal of Machine Learning Technologies*, 2:37–63, 2011.

[15] Add salt or pepper or random valued impulse noise to image. URL `http://www.mathworks.com/matlabcentral/fileexchange/46256-add-salt-or-pepper-or-random-valued-impulse-noise-to-image`. Last Visited: 02-04-2016.

[16] A Bit AboutSVG. URL `http://seesparkbox.com/foundry/a_bit_about_svg`. Last Visited: 02-04-2016.

[17] R. Laganière and R. Elias. The Detection of Junction Features in Images. *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 3:573–576, 2004.

[18] S. Samanta, M. Dey, P. Das, A. Suvojit, and S.S. Chaudhuri. Multilevel Threshold Based Gray Scale Image Segmentation using Cuckoo Search. *International Conference on Emerging Trends in Electrical, Communication and Information Technologies (ICECIT)*, 1:27–34, 2012.

[19] P. Felzenszwalb and D. Huttenlocher. Distance Transforms of Sampled Functions. *Theory of Computing*, 8:415–428, 2012.

[20] J. Thelin. *Foundations of Qt Development*. APress/Springer-Verlag, New York, 2007.

[21] Clay cuneiform tablet. URL `http://www.britishmuseum.org/research/collection_online/collection_object_details/collection_image_gallery.aspx?assetId=850096001&objectId=791645&partId=1`. Last Visited: 27-03-2016.

[22] S. Asano, T. Maruyama, and Y. Yamaguchi. Performance Comparison of FPGA, GPU and CPU in Image Processing. pages 126–131, Prague, Czech Republic, 2009.

[23] M. Garland. Parallel Computing with CUDA. In *IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*, page 1, Atlanta, GA, USA, 2010.

[24] S. Khuller and Y. Matias. A Simple Randomized Sieve Algorithm for the Closest-Pair Problem. *Information and Computation*, 118dd:34–37, 1995.